ETSI TS 104 219 V1.1.1 (2026-03)

TECHNICAL SPECIFICATION

Cyber Security (CYBER); Software Security Development and Implementation Framework

Reference
DTS/CYBER-00190

Keywords
cybersecurity

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document marries the secure software development process of the widely supported NIST Secure Software Development Framework (SSDF) tasks (based in part on the SAFECode Fundamental Practices) to the Critical Security Controls pioneered by the Center for Internet Security (CIS). This marriage creates 6 Essential Practices that group 42 SSDIF specific actions in three different Development Groups (depending on their resources) or DGs to implement those tasks, articulates the resulting artifacts for the three DGs, Responsible Roles by providers and provides examples originally contained in the SSDF. The SSDIF provides implementable and proportionate improved software security for producer-user communities. Numerous annexes are included with informative mappings to dozens of software development frameworks including the EU Cyber Resilience Act (CRA) [i.12] and the UK NCSC Cyber Resilience Testing (CRT) Assurance Principles and Claims (APT) framework [i.11] .

Introduction

The challenge of protecting information is not new. From the earliest days, people sought to protect the integrity and visibility of data at rest and in transit. Experts quickly realized two hard truths: perfect security is impossible, and no software system is risk-free. To quote a respected security researcher, "if it's smart it's vulnerable" [i.34]. This guide builds on that reality—software systems are inherently vulnerable and their protection requires care and effort.

Today, "cybersecurity" means protecting information and business operations across their entire life cycle. At its core, this depends on creating and maintaining secure software. Strong and effective development practices directly improve the security and manageability of software in operation across its lifecycle [i.37]. Unfortunately, much of the cybersecurity industry has grown in reaction to insecure systems, rather than designing them to be resilient from the start. The result has been a cycle of patches, complex configuration management, malware response, and regulatory frameworks—often producing more fatigue than progress.

After many decades of Secure by Design initiatives primarily within government national security communities, NIST undertook the creation of the Secure Software Development Framework (SSDF) in 2020 and publishing a definitive document in 2022, SP 800-218 v 1.1 [i.1]. In 2023, the Cybersecurity and Infrastructure Security Agency (CISA) in cooperation with multiple national security agencies worldwide began a Secure by Design (SbD) initiative [i.5], urging technology vendors to make customer security a business priority and to reduce exploitable flaws at the source. The challenge is clear, but the available guidance is fragmented. Multiple groups—SAFECode "Fundamental Practices for Secure Software Development" [i.2], Black Duck's "Building Security In Maturity Model (BSIMM) [i.63], the UK (National Cyber Security Centre) (NCSC) "Software Security Code of Practice" [i.6], a new NIST NCCoE Consortium [i.17], and Google's "Secure by Design at Google" [i.64] offer recommendations, but they are not always aligned, leaving developers and users uncertain about which to follow and how to proceed. Furthermore, the secure software materials omit critically important user configuration and implementation found in the Critical Security Controls that is essential in achieving actual cybersecurity and infrastructure resilience.

The present document provides independent, practical, and specific actions for software security including assessment artifacts. The goal is to strike a balance between broad principles and concrete, technology-specific practices in a way that is adaptable across lifecycle models, platforms, and development practices [i.37]. It builds on the NIST SSDF [i.1] which provides the best foundation across different development environments.

The present document also provides a short history of SbD, then explains how to apply SbD principles through the SSDF and how to verify that those principles have been adhered to. To make the guidance usable across software development organizations of different maturity levels, a Development Group (DG) model was used as well as mapping to the Critical Security Controls [1] originally developed by CIS [i.4] but not necessary for SSDIF implementation. For each practice, responsible roles are identified as are the artifacts that demonstrate compliance.

The use of artificial intelligence and machine learning (AI/ML) is an important consideration for software development and software security. The security implications of AI/ML are complex, multifaceted, and still an emerging area. This guide includes a brief discussion of four ways that AI/ML and software security interact and refers to some useful resources. Those concerned about AI/ML and software security should understand these interactions, their evolving nature and their associated risks.

Because there are no accepted quantitative measures of software security, both developers and end users rely on qualitative, risk-based judgment to determine sufficiency of any secure development effort. To support this judgment, the present document provides straightforward risk management approaches.

Emerging government policy initiatives such as the UK/Canadian Software Security Principles [i.6] and its implementation as the Cyber Resilience Testing (CRT) Assurance Principles and Claims (APC) [i.11], the U.S. Executive Order 14306 [i.13] and its predecessors, the European Cyber Resilience Act (CRA) [i.12] among others [i.7], [i.8], [i.9], [i.10], [i.14], [i.15], [i.19] emphasize that software development organizations bear responsibility for delivering software that is secure by design. Software developers should adopt the practices and evidence-based approaches to risk management described in the present document both as a matter of responsibility to customers and to enable compliance with those policies.

A mapping from the CRA [i.12] Annex requirements to the SSDIF DG Specific Actions is provided in Annex A. A mapping from the CRT APC [i.11] to SSDIF DG Specific Actions is provided in clause B.1. Implementing SSDIF actions helps the organization meet CRA requirements. The mappings are informative and do not imply authoritative compliance with CRA or CRT APC provisions.

The present document is designed to serve three audiences:

- **End users/customers** - what to ask for and what steps to take.
- **Software development organizations** - what to do in practice.
- **Government and industry bodies** - what specifics to look for to verify Secure by Design adoption.

A long-standing gap in cybersecurity is addressed - practical, evaluable, and aligned guidance for building software that is **Secure by Design.**

1 Scope

The present document provides specific actions differentiated by available resources for three parties in the software ecosystem toward implementing evaluable, effective software security. For end user customers, it specifies what to ask. For software development organizations, it specifies what to do. For governmental or industry bodies, it defines the specifics to demonstrate that secure by design practices have been followed.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found in the ETSI docbox.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 103 305-1: "Cyber Security (CYBER); Critical Security Controls for Effective Cyber Defence; Part 1: The Critical Security Controls".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

[i.1] NIST: "SP 800-218, "Secure Software Development Framework (SSDF) Version 1.1", February 2022.

[i.2] SAFECode: "Fundamental Practices for Secure Software Development", Third Edition.

[i.3] SAFECode: "Application Software Security and the CIS Controls".

[i.4] CIS: "CIS Critical Security Controls®".

[i.5] CISA et al: "Secure by Design", October 2023.

[i.6] NCSC, DSIT, CCCS: "Software Security Code of Practice", May 2025.

[i.7] Bundesamt für Sicherheit in der Informationstechnik (BSI): "BSI TR-03185, Secure Software Lifecycle".

[i.8] ANSSI: "Les Essentiels DEVSECOPS, February 2024".

[i.9] ANSSI: "Les Essentiels Sélection d'un Logiciel Libre, May 2025".

[i.10] ENISA: "Advancing Software Security in the EU, 2020".

[i.11] NCSC: "Product Cyber Resilience Testing (CRT) Assurance Principles and Claims" Version 1.0, April 2025.

[i.12] 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) (Text with EEA relevance).

[i.13] U.S. President: "EO 14028 of May 12, 2021, Improving the Nation's Cybersecurity" amended by "EO 14036 of June 6, 2025, Sustaining Select Efforts To Strengthen the Nation's Cybersecurity and Amending Executive Order 13694 and Executive Order 14144".

[i.14] Singapore CSA: "Protecting Your Software from Malicious Third-party Dependencies", January 2023.

[i.15] Japan METI: "Collection of Use Case Examples Regarding Management Methods for Utilizing Open Source Software and Ensuring Its Security", May 2022.

[i.16] ACM: "Pearce et al., Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions", January 2025.

[i.17] NIST: "NIST Consortium and Draft Guidelines Aim to Improve Security in Software Development", July 2025.

[i.18] ARCSI: "Desvignes, A French Story, Cryptologic History Symposium", 11 May 2022.

[i.19] CERT-EU: "Dezeure et al., Improving the world's cyber resilience, at scale: Implementing baseline security by default", February 2024.

[i.20] CIS: "A Guide to Defining Reasonable Cybersecurity", Version 1.1, October 2024.

[i.21] CISCO: "25th Annual Report: The State of AI Security".

[i.22] CMU-SEI: "The CERT® Guide to Coordinated Vulnerability Disclosure".

[i.23] CSET: "Rolf, AI and the Software Vulnerability Lifecycle", August 2025.

[i.24] ETSI TR 104 003: "Cyber Security (CYBER); The vulnerability disclosure ecosystem".

[i.25] ETSI TR 104 034: "Cyber Security (CYBER); Software Bill of Materials (SBOM) Compendium".

[i.26] ETSI TR 104 092: "Reasonable Cybersecurity", October 2025.

[i.27] ETSI TS 104 874: "Cyber Security (CYBER); Global Vulnerability Reporting Framework".

[i.28] FIRST: "Common Vulnerability Scoring System".

[i.29] FIRST: "Product Security Incident Response Team (PSIRT) Services Framework".

[i.30] IriusRisk: "Build-Safer-Faster: The AI Threat Modeling Tool".

[i.31] ISO 29147: "Information technology — Security techniques — Vulnerability disclosure".

[i.32] ISO 30111: "Information technology — Security techniques — Vulnerability handling processes".

[i.33] Johnson: "The Evolution of British Sigint 1653-1939", The Stationary Office Ltd., July 1998.

[i.34] eWeek: "AI Might Give Security Defenders an Advantage: Researcher Explains During Black Hat 2025 Keynote", August 2025.

[i.35] Kahn, David: "The Code-breakers Breakers The Comprehensive History of Secret Communication from Ancient Times to the Internet", December 5, 1996.

[i.36] Microsoft Ignite: "Microsoft Threat Modeling Tool".

[i.37] Microsoft: "Michael Howard & Steve Lipner, The Security Development Lifecyle, 2006".

[i.38] National Academies: "Defense Software for a Contested Future: Agility, Assurance, and Incentives (2025)".

[i.39] NIST: "Anderson, Computer Security Planning Study, October 1972".

[i.40] NIST SP 800-216: "Recommendations for Federal Vulnerability Disclosure Guidelines".

[i.41] NIST SP 800-218A: "Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile", April 2024.

[i.42] NIST NCCOE: "Secure Software Development, Security, and Operations (DevSecOps) Practices".

[i.43] OASIS: "Common Security Advisory Framework (CSAF)".

[i.44] OWASP: "OWASP Threat Dragon".

[i.45] OWASP: "Vulnerability Exploitability eXchange (VEX)".

[i.46] SAFECode: "Focus on Fuzzing".

[i.47] SAFECode: "Security Capabilities to Support Code Integrity".

[i.48] SAFECode: "Guidance for Agile Practitioners".

[i.49] SAFECode: "Managing Security Risks Inherent in the Use of Third-party Components, 2017".

[i.50] SAFECode: "Resource Centers".

[i.51] SAFECode: "Tactical Threat Modeling", May 2019.

[i.52] SAFECode: "Threat Modeling at Scale", June 2023.

[i.53] SAFECode: "Software products aren't cookies".

[i.54] Wiley: "Shostack, Adam: Threat Modeling: Designing for Security".

[i.55] Wikipedia: "List of Tools for Static Code Analysis".

[i.56] CIS: "Episode 26: Automating the Secure Configuration Management Process".

[i.57] CSO: "10 Top Fuzzing Tools: Finding the Weirdest Application Errors".

[i.58] Wikipedia: "Threat Model".

[i.59] NIST: "Source Code Security Analyzers".

[i.60] GitHub: "analysis-tools-dev/static-analysis".

[i.61] NIST: "Combinatorial Methods for Trust and Assurance".

[i.62] GitHub: "google/oss-fuzz".

[i.63] BLACKDUCK: "Building Security in Maturity Model (BSIMM)".

[i.64] Google: "Secure by Design at Google", March 2024.

[i.65] RAND: Memorandum RM-3765-PR (August 1964): "On Distributed Communications - IX. Security Secrecy and Tamper-Free Considerations", Paul Baran.

[i.66] Willis Ware: "Security and privacy in computer systems", AFIPS Proceedings, 1967 Spring Joint Computer Conference, Vol. 30 at p. 279.

[i.67] RAND: "Security Controls for Computer Systems", originally February 1970.

[i.68] Univ. of Virginia: "Saltzer and Schroeder, The Protection of Information in Computer Systems", 1975.

[i.69] NSA: "The 60 Minute Network Security Guide (First Steps Towards a Secure Network Environment", 16 October 2001.

[i.70] NSA/NCSC: "Rainbow Series".

[i.71] IEEE™: "Steven Lipner, The Birth and Death of the Orange Book", IEEE Annals of the History of Computing, 2015.

[i.72] Lipner: "Cybersecurity historical materials".

[i.73] Wikipedia: "ITSEC".

NOTE: The original document can be found at WaybackMachine: "Information Technology Security Evaluation Criteria (ITSEC)".

[i.74] CCDB: "The Common Criteria".

[i.75] Bernard Peters: "Security considerations in a multi-programmed computer system", AFIPS Proceedings, 1967 Spring Joint Computer Conference, Vol. 30 at p. 283.

[i.76] BSA: "The BSA Framework for Secure Software", Version 1.1, 2020.

[i.77] BSIMM Foundations: "BSIMM 13 Foundations Report 2022".

[i.78] CNCF: "Software Supply Chain Best Practices V2".

[i.79] IDA: "State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016".

[i.80] IEC 62443-4-1: 2018: Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements".

[i.81] NISTIR 8397: "Guidelines on Minimum Standards for Developer Verification of Software".

[i.82] ISO/IEC 27034-1:2011: "Information technology – Security techniques – Application security – Part 1: Overview and concepts".

[i.83] Microsoft: "Microsoft Security Development Lifecycle".

[i.84] NIST: "Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1".

[i.85] NIST: "Recommended Criteria for Cybersecurity Labeling of Consumer Software, February 2022".

[i.86] NTIA: "The Minimum Elements For a Software Bill of Materials (SBOM)", July 2021.

[i.87] OWASP: "OWASP Application Security Verification Standard".

[i.88] OWASP: "OWASP Mobile Application Security Verification Standard".

[i.89] OWASP: "Software Assurance Maturity Model Version".

[i.90] OWASP: "OWASP Software Component Verification Standard".

[i.91] PCI: "Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures".

[i.92] SAFECode: "An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain".

[i.93] NIST SP 80053r5: "Security and Privacy Controls for Information Systems and Organizations".

[i.94] NIST SP 800-160v1r1: "Engineering Trustworthy Secure Systems".

[i.95] NIST SP 800-161r1-upd1: "Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations".

[i.96] NIST SP 800-181r1: "Workforce Framework for Cybersecurity Education (NICE Framework)".

[i.97] SAFECode: "How to Build an Effective Security Champions Program".

[i.98] MITRE: "CVE™ Program Mission".

[i.99] GCVE.EU: "Global CVE Allocation System".

[i.100] NIST SP 800-128: "SP 800-128, Guide for Security-Focused Configuration Management of Information Systems".

[i.101] Microsoft Research Blog: "Equation of a Fuzzing Curve — Part 1/2".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

agile: software lifecycle model that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans

artifact: digital evidence generated as a part of the development process and not for the sole purpose of proving compliance to the process

attack surface: set of exposed interfaces and functions in a software product

bug bar: set of criteria that identifies "shall fix" coding errors whose presence would block software from being released

code integrity: echanisms and process to ensure software is unchanged and functions as intended throughout its lifecycle

confidentiality: system protects information from unauthorized disclosure

control frameworks: set of processes and activities that describe the security controls that are the foundation of a security program

default configuration: preset or standard configuration, setting or behaviour of a system, software or device

defense-in-depth: information security strategy integrating people, technology, and operations capabilities to establish variable barriers across multiple layers and missions of the organization

DevSecOps: software development methodology that integrates security into every phase of the software development lifecycle (SDLC) through collaboration, automation, and continuous monitoring

evaluable: subject to effective evaluation of the *secure by design* processes of assessing, measuring, and quantifying the effectiveness of a software development organization's cybersecurity products and posture using various metrics, tools, and methodologies to gain insights into a software development organization's strengths, weaknesses, and potential vulnerabilities

framework: guidance in the form of controls or risk management to organizations for implementing security by design

organisation: organized body of people with a cybersecurity or information infrastructure resilience purpose, including a business, society, association or government agency

risk frameworks: set of processes to provide a consistent approach for managing and accessing risk - often in terms of business risk

SCRUM team: framework for developing and sustaining complex products and consists of accountabilities, events, artifacts, and the rules that bind the team together

secure by default: secure configurations and settings are the default for all systems and software

shall fix: indication of a required action or obligation in software development to address and resolve issues, errors, or bugs within the codebase before release

should fix: indication of a recommended action in software development to address and resolve issues, errors, or bugs within the codebase, that can be subsequently fixed but would not prevent release

spiral development lifecycle: software development lifecycle that combines the iterative and incremental nature of modern methods with the controlled and systematic aspects of the waterfall model, emphasizing risk analysis and consists of four main phases that are repeated in each spiral, moving the project outwards with increasing completeness: Planning, Risk Analysis, Engineering, and Evaluation

supply chain security: measures that assure development teams that "third-party" components will not undermine the security of their products

system authentication: mechanism used to identify a user

EXAMPLE: Passwords, public-key based challenge-response protocols, or other approved techniques.

system authorization: mechanism used to determine access levels or user, client, or process privileges

NOTE: The authorization mechanism may rely on access control lists or classification labels that are compared with user clearances.

system availability: system provides access to information and processing services when required

system integrity: system protects information from unauthorized modification or destruction

vulnerability remediation: activities to correct vulnerabilities within specific software and corresponding activities that serve as feedback to improve the products and processes

waterfall development lifecycle: linear, sequential process where each phase is completed and signed off before the next begins which typically includes phases such as requirements gathering, design, implementation, verification (testing), deployment, and maintenance

NOTE: A key characteristic is its rigidity; changes are difficult and costly to implement once a phase is complete, making it unsuitable for projects where requirements are expected to change.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACM	Association for Computing Machinery
AFCEA	Armed Forces Communications and Electronics Association
AFIPS	American Federation of Information Processing Societies
AI	Artificial Intelligence
ANSSI	Agence nationale de la sécurité des systèmes d'information
API	Application Programming Interface
APC	Assurance Principles and Claims
ARCSI	Association des Réservistes du Chiffre et de la Sécurité de l'Information
BSA	Business Software Alliance
BSAFSS	BSA Framework for Secure Software
BSI	Bundesamt für Sicherheit in der Informationstechnik
BSIMM	Building Security In Maturity Model
CCCS	Canadian Centre for Cyber Security
CIA	Confidentiality, Integrity, Availability
CISA	Cybersecurity and Infrastructure Security Agency
CIS	Center for Internet Security
CISO	Chief Information Security Officer
CRA	Cyber Resilience Act
CRT	Cyber Resilience Testing
CSC	ETSI Critical Security Control

CMU-SEI	Carnegie Mellon University - Software Engineering Institute
CNCF	Cloud Native Computing Foundation
CSAF	Common Security Advisory Framework
CSET	Center for Security and Emerging Technology
CSRC	Combination Methods for Trust and Assurance
CVE	Common Vulnerabilities and Exposures
DG	Development Group
DevSecOps	Development-Security-Operations
DFD	Data Flow Diagram
DHS	Department of Homeland Security
DoD	Department of Defence
DSIT	Department for Science, Innovation and Technology
EEA	European Economic Area
ENISA	European Union Agency for Cybersecurity
EU	European Union
EUVD	European Union Vulnerability Database
GCVE	Global Common Vulnerabilities and Exposures
IAD	Information Assurance Directorate
IDA	Institute for Defense Analysis
IT	Information Technology
MFA	Multi-Factor Authentication
ML	Machine Learning
NCCOE	National Cybersecurity Center of Excellence
NCSC	National Cyber Security Centre (UK)
NCSC	National Computer Security Center (US)
N/A	Not Applicable
NIS	Network and Information Systems
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NVD	National Vulnerability Database
OSS	Open-Source Software
OWASP	Open Worldwide Application Security Project
PCI	Payment Card Industry
PSIRT	Product Security Incident Response Team
SaaS	Security as a Service
SAFECode	Software Assurance Forum for Excellence in Code
SAST	Static Application Security Testing
SNAC	Systems and Network Attack Center
SCA	Software Composition Analysis
SbD	Secure by Design
SBOM	Software Bill of Materials
SC	Security Champions
SDL	Secure Development Lifecycle
SDLC	Software Development Life Cycle
SRE	Site Reliability Engineering
SSCoP	Software Security Code of Practice
SSDF	Secure Software Development Framework
SSDIF	Secure Software Development and Implementation Framework
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
TCSEC	Trusted Computer System Evaluation Criteria
TLS	Transport Layer Security
TPC	Third Party Code

4 Evolution of the Security by Design Ecosystem

4.1 Introduction

The challenges of information security emerged early. David Kahn in his reference book "The Code Breakers" [i.35] describes many of them over the millennia. GCHQ's (United Kingdom's Government Communications Headquarters) authorized history points to the extensive use of cryptography by spymaster Francis Walsingham during the reign of Elizabeth I. [i.33]. The French information security community at the 2022 NSA History Symposium described how cryptographers in the early 19th century protected the messages for Claude Chappe's extensive visual telegraph networks throughout France [i.18]. The guides and manuals of their times effectively became the earliest security by design publications. The inflection point for modern communication was 1964 when Paul Baran and Sharla Boehm at the RAND Corporation and Donald Davies at the UK National Physical Laboratory articulated the use of digital computers to effect large, distributed networks. From a multitude of further studies and research (see Annex D), the following concepts emerged and remain as cybersecurity fundamentals today.

"Security cannot be attained in the absolute sense

For each activity, which exposes private, valuable, or classified information to possible loss, it is necessary that reasonable steps be taken to reduce the probability of loss

Any loss which might occur must be detected

There are several minimum requirements to establish an adequate security level for the software of a networked computer system

Management must be aware of the need for security safeguards and be willing to support the cost of obtaining this security protection

Technical expertise and practical experience must be combined to judge whether or not a desired security level has been reached

No software system can approach a zero risk of loss

It is necessary to reach an acceptable degree of risk" [i.75].

From that time forward, rapid market-driven changes in technology, use of shared resources (e.g. cloud computing), universal connectivity, and complex and dynamic software supply chains that are intended to improve system functionality have also contributed to a pervasive and sprawling problem: development processes and market complexities have outpaced security processes. There have also been numerous attempts to improve the quality and security of software-based systems, and to bring more rigor and focus "leftward" in the life cycle of systems. (A more in-depth history is available in Annex D).

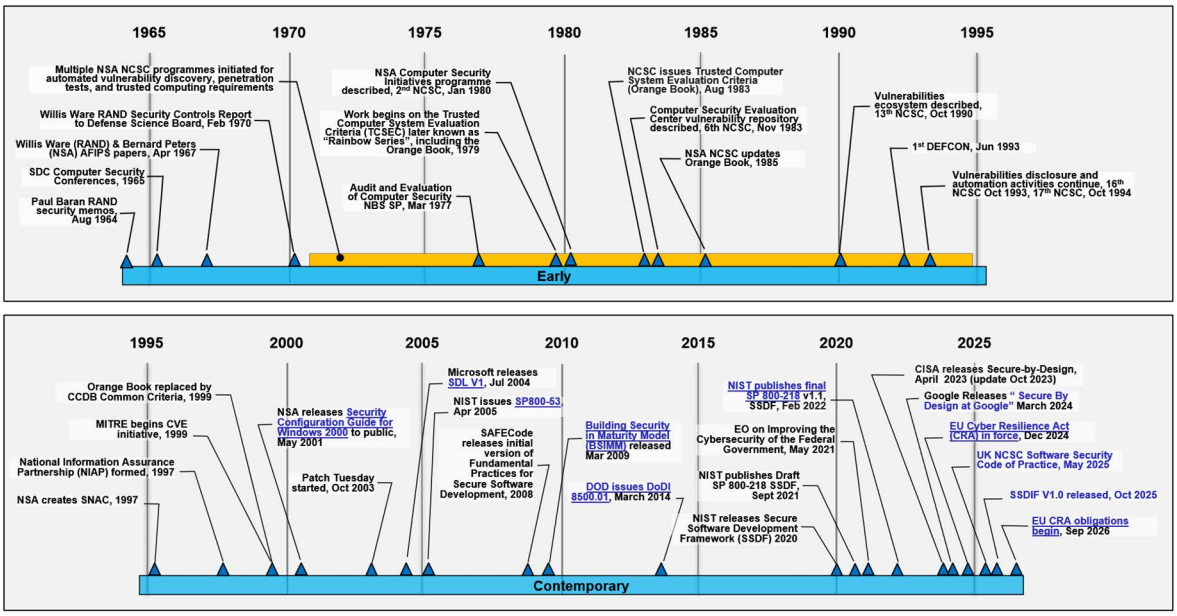

Figure 4.1-1: Software Security Design Development Timeline

An array of developments over recent decades leaves secure software development significantly more challenging including the continuing discovery of new classes of software vulnerabilities, the growing complexity of software, and the move to much more rapid deployment of new software versions.

The importance of managing the security life cycle of in-house developed, hosted, or acquired software to prevent, detect, and remediate security weaknesses before they can impact the enterprise cannot be overstated. In fact, this is exactly Critical security Control 16, "Application Software Security" [1]. With that Control as one foundational catalyst, the basic tenets of Secure by Design (SbD)emerge:

- Perfect security is impossible. The goal is to manage risk.
- Not all security practices are of equal value or cost, so prioritization is essential.
- Prescriptive actions are needed, and assessment of both design effectiveness and operational management of end-systems is key.
- Raising the bar for legacy code is essential.
- "Secure" is not a finished state, but a continuum of components, composition, and operational practice - creating an information feedback loop of improvement.
- It is essential that software development organizations produce artifacts and evidence that attest to SbD best practices as part of their development process. The assessment of SbD should not require the creation of documentation that is inherently removed from the artifacts and evidence created by the development process.

4.2 Achieving and Assessing Security by Design

The present document is intended as a resource for software development organizations, end users, and compliance authorities who seek to answer the question "is this software (or the software delivered by this vendor or service provider) Secure by Design (SbD)?" This has long been a difficult question, and it is more complex today as users move from on-premises self-managed Information Technology (IT) systems to complex environments involving Software as a Service (SaaS) and applications that may be distributed among local clients and servers and an array of vendors and service providers "in the cloud."

There are a set of fundamental security practices that all software development organizations should adhere to. Those practices involve creating code that is resistant to attack, configuring it securely, protecting it from unauthorized change, and continuous improvement of the code in response to the discovery of software vulnerabilities. The NIST SSDF [i.1] elaborates those practices in a way that is general enough to be applied widely but specific enough to serve as a basis for assessing a development organization's practices and products [i.38].

The specific implementation of the practices in the SSDF will vary widely.

- If a software development organization is building software that will be released as a classic "boxed product", it may be appropriate to conduct some security testing on versions or major subsystems that are completed on a cadence of days or weeks. If a software development organization is building software for an online service that is updated many times a day, it is essential that security testing and other assurance activities be integrated into a "Continuous Integration/Continuous Deployment" (CI/CD) infrastructure that supports rapid release. (Some security assurance activities will in any case best be implemented by individual software developers as the code is being written.)

- If a software development organization is doing development using its own systems and facilities, it should protect its development systems using classic IT security measures. If it is using a cloud-based development repository such as GitHub, it should configure its repositories to protect them from unauthorized modification and disclosure and take steps to ensure that its developers access them securely.

- If a software development organization is building a monolithic software application that will run on a single server or server farm, it should configure that server to ensure that users are authenticated and user operations are authorized and audited.

- If a software development organization is building a distributed application that relies on services in the cloud, it is essential to access those services using channels that are encrypted and authenticated and rely on accepted (standardized) secure protocols. Furthermore, it is essential that the software development organization have a basis for believing that the SaaS provider's software is secure - that the SaaS provider implements the fundamental security practices described in the present document.

The definitions of practices and tasks in the SSDF are sufficiently general to accommodate all of the scenarios sketched above.

Not all software development organizations or contexts are the same in terms of security threats or the complexity or criticality of the software being developed. A three-part Development Group construct is used to prioritize activities commensurate with the risks and business cases typical of those facing a software development organization.

It is essential that an organizational culture that prioritizes security underlie implementation of engineering and operational practices for SbD. The NIST SSDF includes practices and tasks that speak to organizational commitment, organizational roles, and personnel training but factors that lead to a culture committed to SbD vary widely: for one organization, the priority may be customer feedback, for another revenue and market share, and for a third the professional pride of the development staff.

To enable assessment of SbD, the present document articulates activities in terms of the practices and tasks defined in NIST SSDF supported by national security authorities worldwide. Some activities include a well-defined metric, such as "provide an initial response to the finders of certain classes of vulnerabilities within one business day". A qualitative activity, such as "provide a means for sensitive vulnerability information to be communicated confidentially" is equally important. For each activity, the present document defines the artifacts or evidence that indicate that the corresponding task has been implemented effectively. It is important to note that even artifacts that cannot be tested automatically (e.g. the definition of processes) can be traced to the requirements of specific activities. The present document identifies roles typically associated with responsibilities for the activities as an aid to software development organizations seeking to create or assess their SbD programs.

The present document does not tell end users/organizations how to decide whether software is "secure enough" for their purposes. That decision is based on risk -the threats to the end users/organizations using the software and the consequences of successful attacks considered in deciding what level of residual vulnerability is acceptable. The consequences of different classes of residual vulnerabilities will differ. However, understanding the level of residual vulnerability implied by the artifacts defined in the present document and comparing those residual vulnerabilities with the causes of historical incidents and attacks should enable end users/organizations to better evaluate and manage their risk.

As a whole, the present SSDIF document describes critical elements to achieve and assess conformance to the SSDF, thus providing confidence that a system is designed and built to be secure.

A mapping from the CRA [i.12] Annex requirements to the SSDIF DG Specific Actions is provided in Annex A. A mapping from the CRT APC [i.11] to SSDIF DG Specific Actions is provided in clause B.1.

5 Software Security Development and Implementation Framework (SSDIF)

5.0 Structure of the SSDIF

5.0.1 Objectives and methods

Secure by Design encompasses what developers regard as "design" as well as security of implementation and security of default configuration. The objective of Secure by Design is that systems and particularly software be "sufficiently secure" to meet the needs of the end users/organizations that rely on them and that those end users/organizations have a sound way to evaluate that sufficiency. The goal is reasonable cybersecurity [i.20], [i.26].

The basic objective of the SSDIF is to motivate cyber "secure by design" as described in clause 4 by articulating a set of actionable Essentials that encompass specific Tasks for software producers and users in a manner that is sufficiently specific to be implementable as well as reasonable and proportionate. The structure is depicted in Figure 5.0-1, below.

The SSDIF Task Definitions and Implementation includes examples identical to those found in the NIST SSDF Special Publication 800-218 [i.1]. For purposes of consistency, where the SSDF uses the word "must", it is identified with quotes. All normative requirements identified by "shall" in the present document are those believed essential to effectively implement SSDIF tasks. The SSDIF implementation actions are described in detail with normative verbs in bold and given an identifier consisting of the NIST SSDF identifier prefaced by SSDIF. Thus, the action required to implement SSDF Task PO.1.2 is SSDIF PO.1.2. The identified practices of the Software Assurance Forum for Excellence in Code (SAFECode) as well as the Critical Security Controls were developed and continuously evolved over many years to improve software assurance programs and encourage the industry-wide adoption of fundamental secure development practices that are proven to be both effective and implementable [i.2], [1], [i.4].

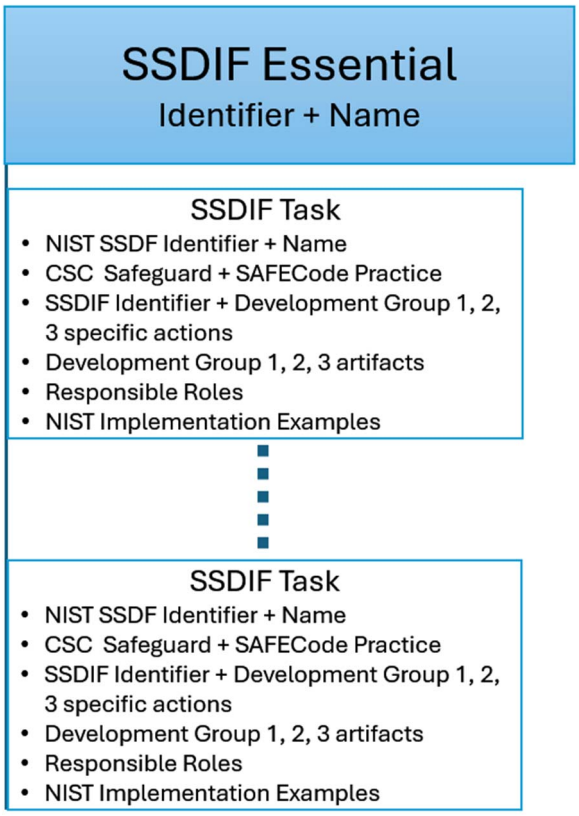

Figure 5.0-1: SSDIF Framework Structure

The Six SSDIF Essentials are summarized in Table 5.0-1, below, and specified in detail in clauses 5.1 to 5.6. The NIST SSDF Special Publication provides a foundational listing of software development tasks. The SSDF organisation into practice groups is, however, less useful for engineer-oriented and action-oriented purposes for the community of development organizations. Table 5.0-1 below shows how the SSDIF in the present document has restructured the SSDF practices into six SSDIF Essentials.

Table 5.0-1: List of the 6 SSDIF Essentials

	SSDIF Essentials	Summary Description	SSDF Tasks
1	**Secure Software Design**	The functions of the system or software and the underlying structure or architecture that enable an organization to create it.	PO.1.2, PW.1.1, PW.1.2, PW.1.3, PW.2.1
2	**Secure Development**	The process of creating the components that make up the system and assuring that those components do not include weaknesses that can undermine the security of the system.	PO.2.1, PO.2.2, PO.2.3, PO.3.1, PO.3.3, PO.4.1, PO.4.2, PW.4.2, PW.5.1, PW.6.1, PW.6.2, PW.7.1, PW.7.2, PW.8.1, PW.8.2
3	**Secure Default Configuration**	The process of creating secure configurations and settings that are the default for all systems and software.	PW.9.1, PW.9.2
4	**Supply Chain Security**	The process of assuring that "third party" components will not undermine the security of product or service.	PO.1.3, PW.4.1, PW.4.4
5	**Code Integrity**	The process to protect against malicious actions during development and during delivery of completed software.	PO.1.1, PO.3.2, PO.5.1, PO.5.2, PS.1.1, PS.2.1, PS.3.1, PS.3.2
6	**Vulnerability Disclosure and Remediation**	A program that supports the reporting and timely remediation of vulnerabilities and ensures that reported vulnerabilities are used as feedback to improve products and processes.	RV.1.1, RV.1.2, RV.1.3, RV.2.1, RV.2.2, RV.3.1, RV.3.2, RV.3.3, RV.3.4

5.0.2 SSDIF Development Groups

Not all software development organizations possess the same degree of technical expertise or complexity. As such, processes will differ across software development organizations in concert with their characteristics. This need for differentiation by available resources, including SMEs, and organizational maturity is reflected in the present document as three tiers called "Development Groups" (DGs). These DGs are loosely tied to the complexity of a software development organization development environments and drive the prioritization of tasks and activities, thus guiding less mature software development organizations to direct limited resources towards high value activities. The three Development Groups are described in detail in Table 5.0-2.

Table 5.0-2: List of SSDIF Development Groups

Development Group	Description
Development Group 1 (DG1)	The organization largely relies on Off-the-Shelf or Open Source (OSS) software and packages with only the occasional addition of small applications or website coding. The organization is capable of applying basic operational and procedural best practices and of managing the security of its vendor-supplied software by following the guidance of the Critical Security Controls [1].
Development Group 2 (DG2)	The organization relies on some custom (in-house or contractor-developed) web and/or native code applications integrated with third-party components and running on premises or in the cloud. The organization has a development staff that applies software development best practices. The organization is attentive to the quality and maintenance of third-party open source or commercial code on which it depends.
Development Group 3 (DG3)	The organization makes a major investment in custom software that it requires to run its business and serve its customers. It may host software on its own systems, in the cloud, or both and may integrate a large range of third-party open source and commercial software components. Software vendors and organizations that deliver software as a service should consider Development Group 3 as a minimum set of requirements but may well have to exceed those requirements in some areas.

5.0.3 SSDIF Roles

The SSDIF identifies specific roles that support the Secure by Design lifecycle and are responsible for the identified practices and actions. These roles are listed in Table 5.0-3, below.

Table 5.0-3: List of SSDIF Roles

Role Type	Role Description
CISO Team	Responsible for policies, standards, configuration, and operation of organization's system and network security.
SDL Team	Responsible for secure development policies, standards, and operational criteria as well as tool selection and configuration standards, and verifying that product teams have correctly implemented the organization's secure development processes.
Product Management	Responsible for identifying customer requirements and for communicating product features and benefits to the organization's customers.
Procurement	Responsible for selection of third-party suppliers to the organization, for establishing contractual requirements for suppliers and for verifying that suppliers meet those requirements on an ongoing basis.
Engineering Group Leadership	Responsible for overall leadership of the development team(s) that produce and sustain a product or online service, a group of products or online services, or all products and services developed by the organization. Responsibilities include engineering strategy and culture.
Corporate Leadership	Organizational top management responsible for the organization's strategy and culture.
Engineering Group Build Team	Responsible for operating and managing an engineering team's build systems that maintain source code repositories and workflow systems and create test and final versions of online services from developer-created source code and third-party components.
Release Engineer or SRE	Responsible for deploying software to an online service, sustaining the configuration of the software and online service, and responding to outages or failures.
Program Manager	Responsible for translating customer requirements (as assembled by product managers) into technical requirements for a product or service, as well as for managing the workflow of the development process including bug and change processing and tracking, scheduling, and release.
Architect	A very senior software engineer with broad technical guidance responsibilities for a product or online service. Responsible for overall technical structure of a product or service including modular structure, interfaces and protocols, and approach to integration of security features into the product or service.
Software Engineer	Responsible for developing software modules or components consistent with the product or service architecture (defined by the architect), product technical requirements (defined by the program manager), and secure development process (defined by the SDL team). The software engineer is responsible for applying the SDL process and tools to ensure the elimination of software vulnerabilities.
Test Engineer	Responsible for testing software components and the finished product or service to detect functional and security errors and flag them to the program manager and software engineer or architect for correction.
Writer	Responsible for producing user documentation for the product or service.
Security Response Engineer	Responsible for managing and executing the organization's security response process including coordination with vulnerability reporters, initial triage of reports, and engaging with the development team (software engineer, program manager, tester, architect) to reproduce and evaluate the reported vulnerability. The security response engineer also manages the process of releasing vulnerability remediations to customers and the public.

5.0.4 Assessment and Evidence

For evaluation and legal purposes, software development organizations need to demonstrate that their products are "Secure by Design (SbD)." This does not mean that the software is vulnerability-free - there is no such thing. For purposes of the present document, secure-by-design means that the software development organization has sufficiently addressed the six SSDIF Essentials.

Of the six Essential Elements, the first three - design, secure development, and default configuration - are the primary focus of the software development organization. Supply chain security implies reliance on the software development organization's upstream suppliers, and thus the organization takes the role of a customer that seeks evidence and assurance and/or performs acceptance testing to ensure that its suppliers' products are "secure by design". In practical terms, the software development organization demands that its supplier meet the requirements of the present document.

Code integrity is in part the realm of the software development organization and in part the realm of operational IT security measures. This applies to both the software development environment and the targeted run-time environments.

Vulnerability remediation is a component of Coordinated Vulnerability Disclosure guided by applicable legal obligations and normative specifications [i.22], [i.24], [i.29], [i.31], [i.32], [i.40], [i.98], [i.99]. Vulnerability remediation is also a critical input to secure design, implementation, and default configuration because root cause analysis of discovered vulnerabilities is the key source of updates to secure development processes, developer training, and the tools that support secure development. Table 5.0-3, above, provides a list of typical roles that support the SbD lifecycle.

Beyond the technical and product-specific activities associated with the six Essentials is the matter of organizational commitment to software security. The three SSDF PO.2 practices [i.1] speak to this matter and is reflected in organization charts, training curricula, and management communications. The evidence of organizational commitment will vary widely from one software development organization to the next, depending on culture, personalities, and history. Evidence of organizational commitment should be considered in the context of the evidence and effectiveness of the software development organization actions in response to the six Essentials.

The end product of secure software design is source code and binaries. However, it is not practical to assure or assess the security of source code and binaries in isolation, so development teams apply processes and tools and create supporting artifacts such as threat models, test results, and the outputs of scanning tools that facilitate both secure development and security assessment [i.44]. Evidence that is created solely to satisfy an assessor or evaluator should be avoided.

The Secure by Design principles articulated in government materials including the SSDF describe product attributes and development processes or techniques that enable systems to protect their sensitive information from unauthorized access or disruption. A key question for software development organizations, for end users/organizations, and for government and industry bodies is how to gain confidence that a particular product or service possesses the required attributes and was developed using the required techniques.

Measures of confidence in the security attributes of a product or service are usually broken into two broad categories:

- Does the product or service meet requirements for security functions or features that enable it to protect information?
- Has the product or service been developed so that it is assured to be free from exploitable vulnerabilities - from errors that would allow an adversary to defeat its security functions?

Clauses 5.1 to 5.6 provides an in-depth mapping of SSDF practices and tasks to actionable activities, some of which reference the Critical Security Controls [1] and others of which reference the companion SAFECode document "Application Software Security and the Critical Security Controls" [i.3], [i.2]. These activities are also mapped or allocated to development groups to provide a roadmap for both implementation and assessment.

Confidence in the presence and correct implementation of some security features can be gained by testing the features to ensure that they meet their specifications or, in some cases, by mathematically verifying their correctness. Testing is insufficient to confirm that a system is free from exploitable errors - any finite amount of testing can confirm the presence of errors, but not their absence.

The process of gaining confidence that a system is free from exploitable errors is referred to as assurance. Assurance addresses both the security of the implementation of the system and the ability of the system's security features and their configuration to resist attack.

In most cases, it is not feasible to demonstrate with absolute confidence that software is secure: real-world systems are imperfect, new classes of vulnerabilities are discovered frequently, and tools and analysis techniques make errors. Pragmatically, real-world assurance relies on a combination of system attributes and processes to demonstrate that a system is "good enough" to meet anticipated threats.

A key artifact of secure development is the report of product vulnerabilities. The root cause analysis and process updates in the bug tracking system reflect the seriousness and quality of the development team's work to create and sustain a secure development process. As part of measuring conformance with SbD, the following factors should be considered:

- The source code includes security features as well as evidence that necessary checks have been performed. The bug tracking system reflects the outputs of security tools as work items that are marked "fixed" when security errors have been corrected.
- When threat models identify attack surfaces and potential weaknesses or requirements for addition of security features, those reports and requirements flow into the bug tracking system and are "fixed" by developers who make code changes. The threat models themselves are retained in the repository with plans and specifications.
- Vulnerability reports lead to bugs are marked "fixed" when the developers remediate the vulnerability and also lead to work items for the developers to review code for related vulnerabilities and to the security team to update tools and process as needed to prevent recurrence.

The SSDIF provides a basis to assess "good enough" for products or services. It specifies requirements that apply to the software development organization, the design, development, and support processes, and the software that makes up the product or service. As an example, software development teams track their work. They keep source code and build files in controlled repositories. They document their work plans and the plans' completion in bug tracking systems (to record the reasons for changes or additions to the software and when those changes or additions have been completed and released to end users/organizations). In addition, development teams maintain configuration-controlled repositories of plans and specifications when they are necessary parts of software development.

For the software security team, the "product" is the secure development process that the development teams execute. The secure development process tells the developers which development, analysis, and testing tools they run, what secure coding conventions they follow, which tool outputs represent "shall fix" and "should fix" bugs, and which analysis and design steps they follow (e.g. threat modelling). Just as the developers maintain their code in controlled repositories and track their work items in bug tracking systems, the security team tracks the evolving process in a repository and uses a bug tracking system to control changes.

The NIST SSDF divides the work of building secure software into four practice groups:

- Prepare the Organization (PO) identifies the practices and tasks associated with defining the secure development process and enabling the development organization to execute the process. These tasks and practices are implemented by the organization's software security team.
- Protect the Software (PS) identifies practices and tasks associated with software integrity. These tasks and practices are implemented by the organization's Information Technology and Security teams as well as the development team's release managers.
- Produce Well-Secured Software (PW) identifies practices and tasks associated with designing, analysing, and creating secure software (including supply chain security for third-party components). These practices and tasks are executed primarily by the development teams (and by the development and procurement teams for third-party components).
- Respond to Vulnerabilities (RV) identifies practices and tasks for remediating reported vulnerabilities and reflecting lessons learned from those vulnerabilities in the secure development process. These practices and tasks are performed by the development team and the software security team.

Supporting evidence should be very specific to a product or service, to the software development organization, and to the development process and tools that the software development organization uses. As an example, the artifacts for demonstrating compliance at DG 3 are source code, tool outputs, threat models, design documents, and items in the workflow systems. (As mentioned above, not only are product errors or vulnerabilities items in the workflow system, but so are the outputs of the root cause analysis that may trigger work items to update the secure development process). A third-party assessment would rely on those artifacts, and especially the workflow items, to demonstrate compliance.

Because these elements vary widely from software development organization to software development organization, a document such as this one cannot provide the level of detail required to tell any specific software development organization exactly what it should do. For that reason, the present document describes the evidence in terms that are intended to be sufficiently specific to guide the software development organization but sufficiently general to apply across a range of software development organizations.

As an example, evidence for the use of a static analysis tool consists of specific artifacts associated with running the tool. These artifacts include the tool itself (of the specific version required in the secure development process), the source code for the program, and the output of the tool as reflected in an output file or in a workflow database where the tool reports "shall fix" and "should fix" errors that it detects. As the process evolves in response to changing threats and security techniques, new kinds of artifacts are produced. An evaluator always has the ability to not only read the current source code and look at the most recent file that contains the tool output but also run the current version of the tool on the source code to reproduce the artifacts and gain assurance that the tool run was clean (e.g. without "shall fix" or "should fix" errors). Artifacts have an overarching virtue - as a byproduct of the development process, they are locked or attached to a specific version of code under development.

Artifacts for the definition of processes cannot be tested automatically, but they still provide traceability from the SSDF requirements to what the software development organization does. For example, responding to a reported vulnerability not only leads to a bug in the product workflow system, but also to a root cause analysis. If multiple bugs implicate the same area of the secure development process, then the software development organization acts to "fix" those bugs by updating its process and adding new secure development tools, training, or processes.

The question of knowing whether a product is good enough remains. The answer is a matter of risk management - first by the software development organization and then by the end user/organization of the software. The risk management responsibility is both context dependent and shared.

The set of errors that the software development organization designates as "shall fix" will be based on the software development organization's and the software industry's experience with classes of proven or potential vulnerabilities and their consequences. The software development organization's security team will learn that a particular class of error can enable the creation of pervasive and damaging malware or attacks and determine that the risk from that class of error is unacceptable - even if some errors of that class turn out to be hard to exploit. The collection of those learnings - by the software development organization or others in the industry - will underpin the software development organization software security policy. The evidence that the software development organization is applying risk management to the development and sustainment of its security policy is collected in the workflow database for maintenance of the security policy. The evidence that the software development organization is adhering to its security policy is collected in the workflow database for the software, the source code, the tool outputs, and the design documents and threat models.

In most cases, the end user/organization of the software will not have access to the full set of evidence associated with a particular software product or service. But every software development organization should have a way to report product vulnerabilities and a policy for responding to them - and a website that makes it clear that the software development organization is actually implementing that policy. As to the core secure development practices, it is reasonable for a software development organization to be expected to publish its secure development process, or at least to make it available for review under nondisclosure agreement. A comparison of the requirements in the software development organization's secure development process with the history of discovered vulnerabilities in the software development organization's software can be very indicative - about the quality of the software development organization's secure development process and about how well the software development organization is executing its process. That comparison can be valuable in informing the end user/organization's risk management decisions about software products and services.

5.0.5 Use of Artificial Intelligence and Machine Learning

The use of Artificial Intelligence and Machine Learning (AI/ML) is an important consideration for software development and software security. There are four domains for application of AI/ML in secure software development:

1) The components under development may incorporate AI/ML components whose security is assured.

Assuring the security of AI/ML components is an evolving area. Current guidance emphasizes the use of and some extensions to best practices such as those required by the present SSDIF document - threat modelling, risk management, application of common security best practices, and penetration testing [i.21]. NIST has released a "profile" that extends the SSDF to address the secure development considerations raised by the inclusion of AI/ML components in software [i.41]. The high-level guidance is to "proceed with caution."

2) The software development organization may use AI/ML to assist in the creation of software.

Research results have demonstrated that there is significant risk associated with the use of AI/ML to generate software. A recent paper in the **Communications of the Association for Computing Machinery** emphasizes this point [i.16]. The present document suggests that software development organizations apply the practices defined in the SSDIF to AI/ML-generated software just as they would to software written by human developers.

3) The software development organization may use AI/ML tools to help assure the security of the software.

The use of AI/ML to aid in assuring the security of software is an emerging but promising field. See for example the development by IriiusRisk of an AI-based tool to aid in threat modelling [i.30] and the work at Georgetown University on vulnerability discovery and remediation [i.23]. Such tools appear to be useful additions to the software development organization's execution of the activities specified in the SSDIF.

4) Adversaries may use AI/ML tools to attack the software.

Finally, the use of AI/ML offensively to enable the detection and exploitation of vulnerabilities is a likely outgrowth of work to enable detection of vulnerabilities for defensive purposes. Capabilities are likely to evolve with, or ahead of, defensive capabilities. Again, implementation of the practices specified in the SSDIF is the best countermeasure.

5.1 Essential 1: Secure Software Design

5.1.0 Overview

Secure software design begins with a set of security objectives that the software satisfies and a set of security threats that the software should resist. Although specific systems - especially systems targeted at specific applications or customers - may have very specific objectives, most commercial products and online services are designed to meet more general classes of objectives.

The fundamental properties for any secure system are usually (CIA) confidentiality, integrity and availability. A fourth common property is accountability as described in Table 5.1-1.

Table 5.1-1: List of SSDIF Fundamental System Properties

Fundamental Property	Definition
Confidentiality	The system protects information from unauthorized disclosure.
Integrity	The system protects information from unauthorized modification or destruction.
Availability	The system provides access to information and processing services when required.
Accountability	The system supports non-repudiation by creating permanent records that can be used to hold individuals accountable for their actions on information and the system.

These properties are typically achieved by the application of security features as described in Table 5.1-2.

Table 5.1-2: List of SSDIF System Security Features

Feature	Intent
Authentication	Specific users or remote systems are reliably identified - may rely on passwords, public-key mechanisms, or challenge-response protocols.
Authorization	Only specific users or remote systems are granted access to information, and that access reflects their rights in the context of the system - may rely on access control lists that list the rights of individual users or groups, or on sensitivity labels that are compared with users' privileges or entitlements.
Auditing	The system creates reliable records of users and their actions - relies on the creation of (preferably immutable and unforgeable) records of users and their actions.

The security features listed above should not only be implemented correctly but also supported by underlying mechanisms that ensure that they cannot be subverted or bypassed. Designing a secure system is an art, but resources are available that can aid the process. The reference monitor concept [i.39] specifies that security features be invoked on every access to information, be protected from unauthorized tampering, and be implemented correctly. A paper from the 1970s describes principles for secure design that expand on the reference monitor concept [i.68].

Secure system design teams in industry frequently rely on a process called threat modelling to review their designs. Threat modelling does not focus on threats in the sense of a hostile agent who seeks to undermine system security. Rather, in threat modelling, the designer analyses a data flow diagram of a system to identify points where an attacker might undermine the system's security and approaches to countering those attacks. The threat modelling process identifies specific action items for designers and developers to make software more resistant to attack. Threat modelling is a well-established and accepted approach that is supported by books, commercial and open-source tools, and training. A summary of the classes of attacks identified by a common threat modelling approach is provided in Table 5.1-3. The classes of attacks are usually referred to by the mnemonic acronym STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) [i.36].

Table 5.1-3: List of Attack Classes

Attack Class	Attack Intent
Spoofing	Misrepresent an attacker's identity (and authorization)
Tampering	Modify or destroy information without authorization
Repudiation	Bypass auditing
Information disclosure	Observe information without authorization
Denial of service	Disrupt the system's operation
Elevation of privilege	Gain control of the system or application without authorization

In the traditional world of waterfall or spiral development lifecycles, development - coding, unit testing, system testing followed design [i.37]. With the evolution of modern development practices such as agile and DevSecOps [i.42], the "design" phase produces an overall system structure or architecture and then detailed feature design, coding, testing, and deployment to customers proceed in an overlapping fashion. For the purposes of the present document and of security by design, the overall architecture design and incremental designs of features and enhancements as well as choice of default configuration settings are design - all should consider threats and countermeasures and be guided by threat modelling or a comparable process.

5.1.1 SSDIF Essential 1 Tasks - Secure Software Design

Task ID PO.1.2	**Identify and document all security requirements for organization-developed software to meet and maintain the requirements over time.**
CSC Safeguard; SAFECode practices	16.1, 16.2; N/A
SSDIF PO.1.2 DG Specific Actions	DG 1/2/3: The organisation 1. **shall** establish a documented secure development process (in hard copy, on an internal website, or in another organization-selected form) that addresses secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. 2. **shall** review and update the documented process annually, or when significant enterprise changes occur.
DG Artifacts	DG 1/2/3: • Documented a secure development process that is updated yearly or when significant changes occur. • Work item tracking system showing the organization is following the documented process and updates it at least annually.
Responsible Roles	SDL Team; Product Management
Implementation Examples	1. Define policies that specify risk-based software architecture and design requirements, such as making code modular to facilitate code reuse and updates; isolating security components from other components during execution; avoiding undocumented commands and settings; and providing features that will aid software acquirers with the secure deployment, operation, and maintenance of the software. 2. Define policies that specify the security requirements for the organization's software and verify compliance at key points in the SDLC (e.g. classes of software flaws verified by gates, responses to vulnerabilities discovered in released software). 3. Analyse the risk of applicable technology stacks (e.g. languages, environments, deployment models) and recommend or require the use of stacks that will reduce risk compared to others. 4. Define policies that specify what needs to be archived for each software release (e.g. code, package files, third-party libraries, documentation, data inventory) and how long it needs to be retained based on the SDLC model, software end-of-life, and other factors. 5. Ensure that policies cover the entire software life cycle, including notifying users of the impending end of software support and the date of software end-of-life. 6. Review all security requirements at least annually, or sooner if there are new requirements from internal or external sources, a major vulnerability is discovered in released software, or a major security incident targeting organization-developed software has occurred. 7. Establish and follow processes for handling requirement exception requests, including periodic reviews of all approved exceptions.

Task ID PW.1.1	**Use forms of risk modeling - such as threat modeling, attack modeling, or attack surface mapping - to help assess the security risk for the software.**
CSC Safeguard; SAFECode practices	16.14; Threat Model
SSDIF PW.1.1 DG Specific Actions	DG 1/2: The organisation: None DG 3: The organisation: 1. **should** consider an appropriate threat modelling methodology such as STRIDE, CIA, or similar mechanism. 2. **should** track and document threat models and use automated threat modelling tools where the term "threat" in the threat modelling process refers to vulnerabilities and weaknesses in design rather than threat actors. 3. **shall** identify a set of possible threats that are relevant to the system being analysed, how they present themselves in various possible scenarios and what can be done to mitigate them. 4. **shall** rate threat severity and mitigate high and medium threats. 5. **shall** create bugs in a workflow system that address the high severity threats and fix them. 6. **should** create bugs that address medium threats and fix them. 7. **should** create an initial description of the structure, use cases, misuse and abuse cases, and resources the system is subjected to or constrained by. It does not need to be a complete description. This is often represented as a diagram (e.g. a data-flow diagram, DFD [i.54]) that describes the system and maps (some of) the potential attack points from outside the system. It is supported by annotations about the internals of the system, data transformations and storage, and particulars such as deployment modes or asset descriptions. This may be done at varying levels of formality, from specification documents to drawings on the back of an envelope. 8. **shall,** if the organisation creates a description, accurately depict the system being modelled.
Artifacts	DG 1/2: • N/A DG 3: • Documented threat model and tools used to support threat modelling. • DFD describing system and potential attacks. • Content of workflow system showing mitigations of medium and high threats.
Responsible Roles	Architect; Program Manager; Software Engineer
Implementation Examples	1. Train the development team (security champions, in particular) or collaborate with a risk modelling expert to create models and analyse how to use a risk-based approach to communicate the risks and determine how to address them, including implementing mitigations. 2. Perform more rigorous assessments for high-risk areas, such as protecting sensitive data and safeguarding identification, authentication, and access control, including credential management. 3. Review vulnerability reports and statistics for previous software to inform the security risk assessment. 4. Use data classification methods to identify and characterize each type of data that the software will interact with.

Task ID PW.1.2	Track and maintain the software's security requirements, risks, and design decisions.
CSC Safeguard; SAFECode practices	16.2; Track Your Security Work
SSDIF PW.1.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** create and maintain the product specification. 2. **shall** document changes to the specification in the product workflow system until product is finalized.
DG Artifacts	DG 1/2/3: • The organization provides the product specification that is maintained in a repository. • The organization tracks changes to the specification in their workflow system.
Responsible Roles	Program Manager
Implementation Examples	1. Record the response to each risk, including how mitigations are to be achieved and what the rationales are for any approved exceptions to the security requirements. Add any mitigations to the software's security requirements. 2. Maintain records of design decisions, risk responses, and approved exceptions that can be used for auditing and maintenance purposes throughout the rest of the software life cycle. 3. Periodically re-evaluate all approved exceptions to the security requirements, and implement changes as needed.

Task ID PW.1.3	Where appropriate, build in support for using standardized security features and services (e.g. enabling software to integrate with existing log management, identity management, access control, and vulnerability management systems) instead of creating proprietary implementations of security features and services.
CSC Safeguard - SAFECode practices	16.11; Leverage Vetted Modules or Services for Application Security Components [i.49]
SSDIF PW.1.3 DG Specific Actions	DG 1/2/3: The organisation: 1. **should** use peer review to determine if the underlying platform, service environment or accepted Third-Party Component (TPC) provides a function that could be relied upon as an alternative to creating its own.
DG Artifacts	DG 1/2/3: • Product specification, work items in the in the product workflow system.
Responsible Roles	Architect; Program Manager; Software Engineer
Implementation Examples	1. Maintain one or more software repositories of modules for supporting standardized security features and services. 2. Determine secure configurations for modules for supporting standardized security features and services, and make these configurations available (e.g. as configuration-as-code) so developers can readily use them. 3. Define criteria for which security features and services "must" [i.1] be" supported by software to be developed.

Task ID PW.2.1	**Have 1) a qualified person (or people) who were not involved with the design and/or 2) automated processes instantiated in the toolchain review the software design to confirm and enforce that it meets all of the security requirements and satisfactorily addresses the identified risk information.**
CSC Safeguard - SAFECode practices;	16.10; Apply secure design principles in application architectures [i.3].
SSDIF PW.2.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** include tests in its test suite that check for regressions of previously fixed vulnerabilities. DG 2/3: The organisation: 1. **shall** apply secure design principles in application architectures which include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trust user input". 2. **shall** ensure that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. 3. **shall** minimize the application infrastructure attack surface, including turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.
DG Artifacts	DG 1/2/3: • Design specifications or other design documents; design security review minutes; bugs for design issues in the workflow system.
Responsible Roles	Program Manager
Implementation Examples	1. Review the software design to confirm that it addresses applicable security requirements. 2. Review the risk models created during software design to determine if they appear to adequately identify the risks. 3. Review the software design to confirm that it satisfactorily addresses the risks identified by the risk models. 4. Have the software designer correct failures to meet the requirements. 5. Change the design and/or the risk response strategy if the security requirements cannot be met. 6. Record the findings of design reviews to serve as artifacts (e.g. in the software specification, in the issue tracking system, in the threat model).

5.2 Essential 2: Secure Development

5.2.0 Overview

Software development, encompassing coding, testing, rollout to customers, and maintenance throughout the lifecycle, has historically been the source of most product vulnerabilities. At a high level, all secure development processes are similar and encompass the practices enumerated in the SSDF. However, the details of each software development organization's practices are likely to differ. Different software development organizations use different programming languages, development tools, and security tools, and target different platforms - from desktop operating systems to mobile devices to embedded controllers to virtual machines or containers hosted on servers or in the cloud. The combinations lead to myriad secure development processes that are similar at a high level but differ in detail.

It is essential for a secure development process to include:

- A "bug bar" that identifies "shall fix" coding errors whose presence would block software from being released.
- Coding standards and requirements such as using only "safe" runtime libraries, memory-safe languages, and approved encryption software.
- Enabling security or mitigation features that make it difficult or impossible to exploit vulnerabilities that may have gone undetected by the secure development process and remain in software.
- Requirements to use organization-validated and actively supported third-party and open-source components (see the discussion of supply chain security below).
- Requirements to use specific configuration, development, and security tools including:
  - Compilers.
  - Static analysis tools that detect coding errors (especially memory safety errors).

- Configuration security tools that ensure that installation of the software does not undermine the security by default of the software itself or the underlying platform.
- Dynamic analysis tools that test components for dangerous responses to unexpected inputs.

- Requirements for security-related testing beyond dynamic analysis - for example, regression testing to ensure that previously eliminated vulnerabilities have not been reintroduced or adversarial penetration testing.
- Training of development staff to follow the practices and meet the requirements of the process.

Although the list above is brief, it is essential that the secure development process be very detailed in terms of mandated or forbidden coding constructs, "shall fix" errors from security tools, and configuration options that are mandated, allowed, or forbidden. As discussed below, the process should also be updated regularly in response to improvements in security tools and improved understanding of vulnerabilities that may occur. Because vulnerabilities in existing "legacy" code can be frequent and damaging, it is essential that the process also include a policy that addresses updating legacy code to meet security requirements imposed after the software was released. References to tool classes with informative references curated lists of tools are provided in Table 5.2-1, below. A valuable set of guidance materials is also available [i.46], [i.2], [i.48], [i.49], [i.50], [i.51], [i.52], [i.53].

Table 5.2-1: List of Tool Classes and References

Tool Class	Class Description
Static Analysis	Detect coding errors (especially memory safety errors) [i.55].
Configuration Security	Help software development organizations manage and secure their IT environments by ensuring systems and applications are configured according to established security policies and best practices [i.56], [i.100].
Dynamic Analysis	Test components for dangerous responses to unexpected inputs, for example, fuzzing tools [i.57], [i.45].
Threat Modeling	Essential for identifying and mitigating design-level security risks during software development [i.54], [i.58].
References	Tools that software development organizations can use to evaluate a product's attack surface and implementation. Useful references include: Source Code Security Analysers [i.59]. Static analysis (SAST) tools and linters for all programming languages, config files, build tools, and more with a focus is on tools which improve code quality [i.60]. Combinational methods for trust and assurance [i.61]. Continuous fuzzing for open-source software [i.62].

5.2.1 SSDIF Essential 2 Tasks - Secure Development

Task ID PO.2.1	**Create new roles and alter responsibilities for existing roles as needed to encompass all parts of the SDLC. Periodically review and maintain the defined roles and responsibilities, updating them as needed.**
CSC Safeguard; SAFECode practices	16.1; N/A
SSDIF PO.2.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** establish and maintain organizational roles and responsibilities appropriate to the organization and its policies and culture.
DG Artifacts	DG1/2/3: • Documented organization charts and position descriptions that align with responsibilities identified in policies. • Organizational communications of updated roles and responsibilities.
Responsible Roles	SDL Team; Engineering Group Leadership
Implementation Examples	1. Define SDLC-related roles and responsibilities for all members of the software development team. 2. Integrate the security roles into the software development team. 3. Define roles and responsibilities for cybersecurity staff, security champions, project managers and leads, senior management, software developers, software testers, software assurance leads and staff, product owners, operations and platform engineers, and others involved in the SDLC. 4. Conduct an annual review of all roles and responsibilities. 5. Educate affected individuals on impending changes to roles and responsibilities, and confirm that the individuals understand the changes and agree to follow them. 6. Implement and use tools and processes to promote communication and engagement among individuals with SDLC-related roles and responsibilities, such as creating messaging channels for team discussions. 7. Designate a group of individuals or a team as the code owner for each project.

Task ID PO.2.2	**Provide role-based training for all personnel with responsibilities that contribute to secure development. Periodically review personnel proficiency and role-based training, and update the training as needed.**
CSC Safeguard; SAFECode practices	16.9; N/A
SSDIF PO.2.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **should** provide necessary training for all developers that includes details of the organization's secure development practices and associated tools the organization relies on. 2. **should** implement CSC Safeguard 16.9: Ensure that all software development personnel receive training in writing secure code for their specific development environment and responsibilities. 3. **should** update the training as needed. DG 2/3: The organisation: 1. **shall** provide just in time training associated with tools and tool outputs including error messages and how to fix reported errors. 2. **shall** provide defensive programming training.
DG Artifacts	DG1: • Documented list of mandatory online developer training. DG2/3: • Examples of tools and tool outputs that provide just in time training. • Documented list of defensive programming training.
Responsible Roles	SDL Team
Implementation Examples	1. Document the desired outcomes of training for each role. 2. Define the type of training or curriculum required to achieve the desired outcome for each role. 3. Create a training plan for each role. 4. Acquire or create training for each role; acquired training may need to be customized for the organization. 5. Measure outcome performance to identify areas where changes to training may be beneficial.

Task ID PO.2.3	**Obtain upper management or authorizing official commitment to secure development, and convey that commitment to all with development-related roles and responsibilities.**
CSC Safeguard; SAFECode practices	16.1; Motivate the Organization: [i.97]
SSDIF PO.2.3 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** obtain management/executive support for secure development programme. DG 2/3: The organisation: 1. **should** obtain management/executive support for a security champion programme. This can be achieved by presenting factual data on how that program can improve the organisation, create value, or minimize risk such as through better resiliency or compliance. 2. **should** obtain engineering/SCRUM team support for a SC program. This can be achieved by explaining that an SC program is a way to resolve upcoming problems early on in a practical and risk-oriented way.
DG Artifacts	DG1: • Examples of Emails or other communications showing executive and/or management commitment to a secure development program. DG 2/3: • Examples of artifacts an organization can provide include: copies of organization-wide emails, videos of executive talks at virtual or live meetings, motivational videos or internal event recordings, release checklist endorsed by upper management. • Names of security champions and notes/bugs from team meetings.
Responsible Roles	SDL Team; Engineering Group Leadership; Corporate Leadership
Implementation Examples	• EXAMPLE 1: Appoint a single leader or leadership team to be responsible for the entire secure software development process, including being accountable for releasing software to production and delegating responsibilities as appropriate. • EXAMPLE 2: Increase authorizing officials' awareness of the risks of developing software without integrating security throughout the development life cycle and the risk mitigation provided by secure development practices. • EXAMPLE 3: Assist upper management in incorporating secure development support into their communications with personnel with development-related roles and responsibilities. • EXAMPLE 4: Educate all personnel with development-related roles and responsibilities on upper management's commitment to secure development and the importance of secure development to the organization.

Task ID PO.3.1	**Specify which tools or tool types "must" [i.1] or should be included in each toolchain to mitigate identified risks, as well as how the toolchain components are to be integrated with each other.**
CSC Safeguard; SAFECode practices	16.1; N/A
SSDIF PO.3.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** select tools and enable tests appropriate to detect or mitigate vulnerabilities associated with the organisation's programming languages and platforms.
DG Artifacts	DG 1/2/3: • List of tools and/or tool types used in secure software development.
Responsible Roles	SDL Team
Implementation Examples	1. Define categories of toolchains and specify the mandatory tools or tool types to be used for each category. 2. Identify security tools to integrate into the developer toolchain. 3. Define what information is to be passed between tools and what data formats are to be used. 4. Evaluate tools' signing capabilities to create immutable records/logs for auditability within the toolchain. 5. Use automated technology for toolchain management and orchestration.

Task ID PO.3.3	Configure tools to generate artifacts of their support of secure software development practices as defined by the organization.
CSC Safeguard; SAFECode practices	N/A; Track Your Security Work
SSDIF PO.3.3 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** ensure that the tool configuration standards are documented in the workflow system that is used to manage the process. 2. **shall** ensure that the tools are configured as documented and necessary practices are implemented.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	SDL Team; Engineering Group Build teams.
Implementation Examples	1. Use existing tooling (e.g. workflow tracking, issue tracking, value stream mapping) to create an audit trail of the secure development-related actions that are performed for continuous improvement purposes. 2. Determine how often the collected information should be audited and implement the necessary processes. 3. Establish and enforce security and retention policies for artifact data. 4. Assign responsibility for creating any needed artifacts that tools cannot generate.

Task ID PO.4.1	Define criteria for software security checks and track throughout the SDLC.
CSC Safeguard; SAFECode practices	16.1, 16.6; Track Your Security Work; Have a Rating System and a Bug Bar
SSDIF PO.4.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** consider security problems to be software bugs and track them using the normal software bug tracking system. 2. **shall** use an appropriate threat modelling methodology such as STRIDE, CIA or similar mechanism to categorize security bugs. See Table 5.1-3. 3. **shall** use the bug tracking system to record and track mitigations and proactive security measures that are part of the secure development process 4. **shall** create a bug bar that establishes severity levels for internally discovered errors that may result in vulnerabilities as well as externally reported vulnerabilities, including ease of discovery and exploitation of the vulnerability, impact of the vulnerability on either the function of the product or the security of the data it contains. 5. **shall** use the bug bar to determine whether the potential impact and exploitability are high enough to delay releasing a product until a fix is made. 6. **shall** set a minimum level of security acceptability (based on the bug bar) for shipping.
DG Artifacts	DG 1/2/3: • Configuration of bug tracking system
Responsible Roles	SDL Team
Implementation Examples	1. Ensure that the criteria adequately indicate how effectively security risk is being managed. 2. Define key performance indicators (KPIs), key risk indicators (KRIs), vulnerability severity scores, and other measures for software security. 3. Add software security criteria to existing checks (e.g. the Definition of Done in agile SDLC methodologies). 4. Review the artifacts generated as part of the software development workflow system to determine if they meet the criteria. 5. Record security check approvals, rejections, and exception requests as part of the workflow and tracking system. 6. Analyse collected data in the context of the security successes and failures of each development project and use the results to improve the SDLC.

Task ID PO.4.2	**Implement processes, mechanisms, etc. to gather and safeguard the necessary information in support of the criteria.**
CSC Safeguard; SAFECode practices	N/A; Track Your Security Work
SSDIF PO.4.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** use a workflow system configured to record relevant information about tools used to verify necessary secure configurations including bug histories.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	SDL Team; Engineering Group Build team.
Implementation Examples	1. Use the toolchain to automatically gather information that informs security decision-making. 2. Deploy additional tools if needed to support the generation and collection of information supporting the criteria. 3. Automate decision-making processes utilizing the criteria and periodically review these processes. 4. Only allow authorized personnel to access the gathered information and prevent any alteration or deletion of the information.

Task ID PW.4.2	**Create and maintain well-secured software components in-house following SDLC processes to meet common internal software development needs that cannot be better met by third-party software components.**
CSC Safeguard; SAFECode practices	6.1 thru 6.6, 6.8, 16.10, 16.12; Avoid code vulnerabilities
SSDIF PW.4.2 DG Specific Actions	NOTE: PW.8.2 actions further the implementation of the actions below. DG 1/2/3: The organisation**:** 1. **shall** include in its test-suite tests that check for regressions of previously fixed vulnerabilities. 2. **should** ensure that all the operations and all the data access is validated for security permissions. 3. **shall** ensure complete mediation, requiring that every operation on information by a user or process be authorized. 4. **shall** ensure that information supplied to a program - whether from a human user, another program, or a network interface - be validated so that it is only requesting access to which it is entitled. 5. **shall** include in each program tests to ensure that information supplied to a program is properly structured so that it cannot fool or evade security controls. 6. **shall** plan for the safe and reliable installation of security updates. No system is likely to remain free from security vulnerabilities. 7. **shall** use least-privilege access practices, to ensure that each program requires granting to users or processes only the rights required to do their job, using CSC Safeguards 6.1 (access granting) and 6.2 (access revoking). 8. **should** should support the implementation of MFA for externally-exposed applications" (See CSC Safeguard 6.3). 9. **should** ensure that software supports the implementation of CSC Safeguard 6.4: Implement MFA for remote network access. 10. **should** ensure that software supports the implementation of CSC Safeguard 6.5: Implement MFA for all administrative access. 11. **should** implement CSC Safeguard 6.7: Centralize access control. DG 2/3: The organisation**:** 1. **shall** minimize the set of unprotected ports, services, and files and the set of default-enabled options and features that comprise a product's attack surface. 2. **should** write new code in memory safe languages and if a code change has a significant impact on a component, redesign, restructure, then rewrite in memory safe language instead. 3. **should** write code that supports the creation and maintenance of secure audit logs. Applications that deal with sensitive information may be required to keep track of "who did what" in the context of the application and the information it manages. 4. **should** leverage operating system mechanisms as much as possible. An application that is using platform identification and authentication services will have a simpler task of recording the "who" in the audit log it maintains. 5. **should** implement CSC Safeguard 6.8: Define and maintain role-based access control. DG 3: The organisation**:**

Task ID PW.4.2	**Create and maintain well-secured software components in-house following SDLC processes to meet common internal software development needs that cannot be better met by third-party software components.**
	1. **shall** during software design keep the design of the system as simple and as small as possible. 2. **shall** design the human interface for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. 3. **should** design the system so that it can resist attack even if a single security vulnerability is discovered or a single security feature is bypassed. 4. **should** during software design incorporate defence-in-depth including multiple levels of security mechanisms or design the system so that it crashes rather than allowing an attacker to gain complete control. 5. **should** design the system to remain secure even if it encounters an error or crashes. Failing securely is essential for defence-in-depth. 6. **shall** train developers to avoid dangerous coding constructs. 7. **shall** create, maintain and communicate relevant coding standards and conventions that support developers in their efforts to write secure code and use built-in platform security features and capabilities. 8. **shall** run code analysis tools that automate the process of checking for code-level vulnerabilities and verify that developers are following secure coding standards and conventions and avoiding some classes of secure coding errors. 9. **should** use the code analysis tools at the developer desktop during the normal build cycle so that developers receive timely feedback on potential security problems. 10. **shall** run dynamic testing tools that integrate dynamic testing into the organisation development process in the same way as other product-level testing. 11. **shall** evaluate alternative tools tailored for the organization's technology. 12. **should** use code-level penetration testing for the most critical or security-sensitive parts of the software. Penetration testing requires specialized skills and can be time-consuming and expensive, so it may not be possible to use it more widely. 13. **should** maintain a bug bounty program that is a complement to a secure development process. 14. **should** avoid paying bounty testers for repeatedly for finding similar bugs. Alternative actions include: making software available to a select set of testers, and a bug bounty process that identifies classes of vulnerabilities and then updates secure development processes to eliminate them *en masse*.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. • Documentation of implementation of secure development practices through developer guidance and provided training. • Relevant software design and implementation documents. DG2/3 • Results of attack surface testing for the component. DG3: • Documentation of training on coding standards. • Documentation of bug bounty program. • Code-level penetration test reports and remediation plans.
Responsible Roles	Architect; Program Manager; Software Engineer
Implementation Examples	1. Follow organization-established security practices for secure software development when creating and maintaining the components. 2. Determine secure configurations for software components, and make these available (e.g. as configuration-as-code) so developers can readily use the configurations. 3. Maintain one or more software repositories for these components. 4. Designate which components "must" [i.1] be included in software to be developed. 5. Implement processes to update deployed software components to newer versions, and maintain older versions of software components until all transitions from those versions have been completed successfully.

Task ID PW.5.1	**Follow all secure coding practices that are appropriate to the development languages and environment to meet the organization's requirements.**
CSC Safeguard; SAFECode practices	16.10, 16.12; Avoid Code Vulnerabilities
SSDIF PW.5.1 DG Specific Actions	NOTE: The actions for PW.5.1 are the same as those for PW.4.2 because both tasks apply to organization-developed software
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. • Documentation of implementation of secure development practices through developer guidance and provided training. • Relevant software design and implementation documents. DG2/3 • Results of attack surface testing. DG3: • Documentation of training on coding standards. • Documentation of bug bounty program. • Code-level penetration test reports and remediation plans.
Responsible Roles	Architect; Software Engineer
Implementation Examples	1: Validate all inputs and validate and properly encode all outputs. 2: Avoid using unsafe functions and calls. 3: Detect errors and handle them gracefully. 4: Provide logging and tracing capabilities. 5: Use development environments with automated features that encourage or require the use of secure coding practices with just-in-time training-in-place. 6: Follow procedures for manually ensuring compliance with secure coding practices when automated methods are insufficient or unavailable. 7: Use tools (e.g. linters, formatters) to standardize the style and formatting of the source code. 8: Check for other vulnerabilities that are common to the development languages and environment. 9: Have the developer review their own human-readable code to complement (not replace) code review performed by other people or tools. See PW.7.

Task ID PW.6.1	**Use compiler, interpreter, and build tools that offer features to improve executable security.**
CSC Safeguard; SAFECode practices	N/A; Integrate security into development, Use safe libraries.
SSDIF PW.6.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** ensure that decisions about tools and tool options are documented in the process workflow system. 2. **shall** use options that can be verified indirectly either by: looking at the tool configurations in the build scripts (or CI/CD pipeline) or a scanning tool that looks at the binaries (e.g. BinScope or BinSkim).
DG Artifacts	DG 1/2/3: • Configuration of compiler, interpreter and build tools, work items in the secure development process workflow system, and bug history of security bugs in the product workflow system.
Responsible Roles	Architect; Software Engineer; SDL Team.
Implementation Examples	1. Use up-to-date versions of compiler, interpreter, and build tools. 2. Follow change management processes when deploying or updating compiler, interpreter, and build tools, and audit all unexpected changes to tools. 3. Regularly validate the authenticity and integrity of compiler, interpreter, and build tools. See PO.3.

Task ID PW.6.2	**Determine which compiler, interpreter, and build tool features should be used and how each should be configured, then implement and use the approved configurations.**
CSC Safeguard; SAFECode practices	N/A; Integrate security into development, Use safe libraries.
SSDIF PW.6.2 DG Specific Actions	Same as SSDIF PW.6.1
DG Artifacts	DG 1/2/3: • Configuration of compiler, interpreter and build tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	SDL Team; Engineering Group Build Team
Implementation Examples	1. Enable compiler features that produce warnings for poorly secured code during the compilation process. 2. Implement the "clean build" concept, where all compiler warnings are treated as errors and eliminated except those determined to be false positives or irrelevant. 3. Perform all builds in a dedicated, highly controlled build environment. 4. Enable compiler features that randomize or obfuscate execution characteristics, such as memory location usage, which would otherwise be predictable and thus potentially exploitable. 5. Test to ensure that the features are working as expected and are not inadvertently causing any operational issues or other problems. 6. Continuously verify that the approved configurations are being used. 7. Make the approved tool configurations available as configuration-as-code so developers can readily use them.

Task ID PW.7.1	**Determine whether code review (a person looks directly at the code to find issues) and/or code analysis (tools are used to find issues in code, either in a fully automated way or in conjunction with a person) should be used, as defined by the organization.**
CSC Safeguard; SAFECode practices	N/A; Select tools and enable tests cautiously
SSDIF PW.7.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **should** document the rules for determining whether code review or code analysis is to be used, record decisions in the process workflow system, and track the results of the reviews in the product workflow system. 2. **should** document results of the reviews in a product workflow system in the form of bugs filed manually or created by the tools.
DG Artifacts	DG 1/2/3: • The documented rules. • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Program Manager; Architect; SDL Team
Implementation Examples	1. Follow the organization's policies or guidelines for when code review should be performed and how it should be conducted. This may include third-party code and reusable code modules written in-house. 2. Follow the organization's policies or guidelines for when code analysis should be performed and how it should be conducted. 3. Choose code review and/or analysis methods based on the stage of the software.

Task ID PW.7.2	**Perform the code review and/or code analysis based on the organization's secure coding standards, and record and triage all discovered issues and recommended remediations in the development team's workflow or issue tracking system.**
CSC Safeguard; SAFECode practices	16.12; Run code analysis tools.
SSDIF PW.7.2 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** track the results of the review and/or code analysis and triage all discovered issues and recommendations. 2. **shall** include in its test-suites tests that check for regressions of previously fixed vulnerabilities.
DG Artifacts	DG 1/2/3: • The resulting tracking materials. • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Software Engineer
Implementation Examples	1. Perform peer review of code, and review any existing code review, analysis, or testing results as part of the peer review. 2. Use expert reviewers to check code for backdoors and other malicious content. 3. Use peer reviewing tools that facilitate the peer review process and document all discussions and other feedback. 4. Use a static analysis tool to automatically check code for vulnerabilities and compliance with the organization's secure coding standards with a human reviewing the issues reported by the tool and remediating them as necessary. 5. Use review checklists to verify that the code complies with the requirements. 6. Use automated tools to identify and remediate documented and verified unsafe software practices on a continuous basis as human-readable code is checked into the code repository. 7. Identify and document the root causes of discovered issues. 8. Document lessons learned from code review and analysis in a wiki that developers can access and search.

Task ID PW.8.1	**Determine whether executable code testing should be performed to find vulnerabilities not identified by previous reviews, analysis, or testing and, if so, which types of testing should be used.**
CSC Safeguard; SAFECode practices	16.12; Select tools and enable tests cautiously, Run dynamic testing tools.
SSDIF PW.8.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** perform test-suite tests that check for regressions of previously fixed vulnerabilities. 2. **shall** analyse and document the results of the tests. DG 2/3: The organisation**:** 1. **shall** perform unit tests and/or system tests that leverage security features.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Program Manager; Test Engineer
Implementation Examples	1. Follow the organization's policies or guidelines for when code testing should be performed and how it should be conducted (e.g. within a sandboxed environment). This may include third-party executable code and reusable executable code modules written in-house. 2. Choose testing methods based on the stage of the software.

Task ID PW.8.2	**Scope the testing, design the tests, perform the testing, and document the results, including recording and triaging all discovered issues and recommended remediations in the development team's workflow or issue tracking system.**
CSC Safeguard; SAFECode practices	16.12, 16.13; Select tools and enable tests cautiously, Run dynamic testing tools.
SSDIF PW.8.2 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** include in its test-suite tests that consist of regressions of previously fixed vulnerabilities. DG 2/3: The organisation**:** 1. **shall** perform unit tests and/or system tests that leverage security features. DG 3: The organisation: 1. **should** run dynamic testing tools. 2. **should** run fuzz testing tools with a large number of iterations against network interfaces, parsers, and APIs and fix any exploitable errors. Empirical analysis finds that 500 000 are sufficient [i.101]. 3. **should** use code-level penetration testing. 4. **shall** create and maintain a list of "shall fix" static analysis errors and run the static analysis tool before code release to ensure that those errors are fixed. 5. **should,** when a vulnerability is detected and static analysis tools that include a fixed set of built-in tests are used, run the tool to determine whether the vulnerable code triggers one or more static analysis errors. 6. **should** review a sample of the tool's reports of the error across a significant amount of code to ensure that a sufficient percentage (such as 80 %) are valid rather than false positives. 7. **should** consider a static analysis error as a strong candidate for "shall fix" for all developers for all code in the future, if there have been multiple instances (typically three or more) of externally reported vulnerabilities (e.g. CVE/GCVEs) that trigger an error. 8. **should** for static analysis tools that support customer-developed error signatures develop a new signature for any error that is associated with multiple instances (typically three or more) of externally reported vulnerabilities (e.g. CVE/GCVEs). 9. **shall** consider adding any newly developed error signature with an acceptably low false positive rate to the "shall fix" list.
DG Artifacts	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	SDL Team, Test Engineer
Implementation Examples	1. Perform robust functional testing of security features. 2. Integrate dynamic vulnerability testing into the project's automated test suite. 3. Incorporate tests for previously reported vulnerabilities into the project's test suite to ensure that errors are not reintroduced. 4. Take into consideration the infrastructures and technology stacks that the software will be used with in production when developing test plans. 5. Use fuzz testing tools to find issues with input handling. 6. If resources are available, use penetration testing to simulate how an attacker might attempt to compromise the software in high-risk scenarios. 7. Identify and record the root causes of discovered issues. 8. Document lessons learned from code testing in a wiki that developers can access and search. 9. Use source code, design records, and other resources when developing test plans.

5.3 Essential 3: Secure Default Configuration

5.3.0 Overview

Software products frequently include myriad configuration settings and options that system administrators may use to enable or disable features or adjust product behaviour. Although users *can* adjust those settings, many will fail to do so. Experience has shown that many users and administrators accept default settings. To the extent that the default settings leave more software features and interfaces exposed they increase the risk of misconfiguration and a successful attack on software security. To the extent that the default settings make permissive choices for security settings such as password length, file access permissions, or use of encryption, they make an attacker's task easier. For this reason, software development organizations should seek to make their products "secure by default."

A simple definition of "secure by default" is that the software blocks access to any function or interface that is potentially dangerous or not required by almost all users of the product and configures all other options by default in a secure manner (opt-out of secure configuration rather than opt-in) and defaults to the most secure acceptable configuration of features and mechanisms.

Engineers use the term "attack surface" to refer to the set of exposed interfaces and functions in a software product. Attack surface encompasses enabled network ports and protocols, usable software features, privileges, options, and accessible files. The software product, or the underlying operating system platforms should provide features for controlling access to elements of the attack surface, and the design and development team should evaluate each element of the attack surface to determine whether it is widely needed by users or whether it should be disabled by default. A good rule of thumb is that if a feature, privilege, or interface is not required by 80 % of users, it should be disabled by default.

While software developers should minimize the attack surface they should maximize the use of security features. Developers should enable security features by default and choose more secure options for features subject to the caveat that the software remains usable by its intended customer base. Engineers focused on security should work with usability engineers to devise options that make more options and configurations acceptable to the majority of the product's end users/organizations.

There are many proprietary and open-source tools that software development organizations can use to evaluate a product's attack surface and secure configuration. Application of such tools before product release should be part of the secure development process, and product teams should ensure that software is locked down by default before release. Table 5.2-1 provides a List of Tool Classes and References.

5.3.1 SSDIF Essential 3 Tasks - Secure Default Configuration

Task ID PW.9.1	**Define a secure baseline by determining how to configure each setting that has an effect on security or a security-related setting so that the default settings are secure and do not weaken the security functions provided by the platform, network infrastructure, or services.**
CSC Safeguard; SAFECode practices	16.7, 16.10; Use a Secure Design, Minimize Attack Surface
SSDIF PW.9.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **should** select and implement secure configuration hardening measures for application infrastructure based on industry best practices.
DG Artifacts	DG 1/2/3: • Documented secure configuration for software. • Output of configuration scanning tools and records in workflow system.
Responsible Roles	Architect; Program Manager
Implementation Examples	1. Conduct testing to ensure that the settings, including the default settings, are working as expected and are not inadvertently causing any security weaknesses, operational issues, or other problems.

Task ID PW.9.2	**Implement the default settings (or groups of default settings, if applicable), and document each setting for software administrators.**
CSC Safeguard; SAFECode practices	16.7; N/A
SSDIF PW.9.2 DG Specific Actions	Same as SSDIF PW.9.1
DG Artifacts	DG 1/2/3: • Documentation for administrators of secure default configuration.
Responsible Roles	Software Engineer; Writer
Implementation Examples	1. Verify that the approved configuration is in place for the software. 2. Document each setting's purpose, options, default value, security relevance, potential operational impact, and relationships with other settings. 3. Use authoritative programmatic technical mechanisms to record how each setting can be implemented and assessed by software administrators. 4. Store the default configuration in a usable format and follow change control practices for modifying it (e.g. configuration-as-code).

5.4 Essential 4: Supply Chain Security

5.4.0 Overview

The term "supply chain security" refers to the challenge to Secure by Design that is posed by software development organizations' use of software that they did not themselves create (often referred to as "third party code") Such software should have been developed with the full panoply of SbD requirements for the system at hand, but it may not have been. Developers may have ignored the need for security features, written code without attention to the introduction of potential security vulnerabilities or left the attack surface "wide open." They may have stored their source code in an unprotected repository or released it without a digital signature that would protect it from malicious modification.

Software development organizations that integrate Third Party Code (TPC) assume responsibility for ensuring that software does not undermine the security of their integrated product. Ideally, they should be assured that the third-party developer adheres to the requirements of the SSDF, for example by having a contract with the third-party developer or by auditing the third-party developer in accordance with the requirements recommendations of the present document (or both). Many software development organizations enforce policies that only approved TPC can be included and maintain repositories of vetted TPC for use in products or services. Where the third-party falls short, the relying software development organization should take steps to fill the gap, for example assessing the risk that the TPC might pose and performing internal reviews, code scans, or tests.

In recent years, software development organizations have adopted the concept of a Software Bill of Material (SBOM) to track their dependence on TPC [i.25]. An SBOM specifies the specific version of a software component and its origin. End users/organizations sometimes request access to products' SBOMs to gain confidence in the software they are acquiring, but the onus of understanding and addressing third-party risk falls on the software development organization. The SBOM can be an important aid to knowing when a reported vulnerability affects a third-party component, but only the developer can fully understand and remediate the risk inherited from vulnerable TPC [i.49].

The concept of supply chain security applies not only to software components that are integrated into a monolithic product or online service, but also to components that are invoked remotely in a distributed or cloud environment. The software development organization building a product or service should be assured of the security of the components or services that it depends on, including the security of the APIs and protocols used to invoke the component or service, the security of the design and implementation of the component or service, and the operational security of the platform and infrastructure where the component or service is hosted.

5.4.1 SSDIF Essential 4 Tasks - Supply Chain Security

Task ID PO.1.3	**Communicate requirements to all third parties who will provide commercial software components to the organization for reuse by the organization's own software. [Formerly PW.3.1]**
CSC Safeguard; SAFECode practices	16.4; Manage security risk inherent in use of third party components
SSDIF PO.1.3 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **should** include security requirements in contracts and Requests For Proposals.
DG Artifacts	DG 1/2/3: • Contracts, RFPs etc. with security requirements.
Responsible Roles	SDL Team; Procurement
Implementation Examples	1. Define a core set of security requirements for software components, and include it in acquisition documents, software contracts, and other agreements with third parties. 2. Define security-related criteria for selecting software; the criteria can include the third party's vulnerability disclosure program and product security incident response capabilities or the third party's adherence to organization-defined practices. 3. Require third parties to attest that their software complies with the organization's security requirements. 4. Require third parties to provide provenance data and integrity verification mechanisms for all components of their software. 5. Establish and follow processes to address risk when there are security requirements that third-party software components to be acquired do not meet; this should include periodic reviews of all approved exceptions to requirements.

Task ID PW.4.1	**Acquire and maintain well-secured software components (e.g. software libraries, modules, middleware, frameworks) from commercial, open-source, and other third-party developers for use by the organization's software.**
CSC Safeguard; SAFECode practices	16.5; Manage security risk inherent in use of third party components
SSDIF PW.4.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** assess the risk that each TPC poses to the software. 2. **shall** determine known vulnerabilities and their impact. 3. **shall** review the maturity of TPC's provider by noting the maintenance cadence, stability of the TPC over time, development practices employed by the TPC provider, and whether the TPC will reach end of life within the expected lifetime of a product. 4. **shall** based on the above factors, create a risk assessment score for a TPC of interest. This may be binary such as acceptable/unacceptable or high/medium/low. 5. **shall** look for vulnerability reports for the TPC in the appropriate vulnerability databases and other vulnerability reporting channels. 6. **shall** if applicable vulnerability reports exist, compare them with the TPC security update website. 7. **shall** use the TPC only if the reported vulnerabilities have been mitigated by TPC security updates. 8. **shall** mitigate or accept risks from vulnerable TPCs based on the risk assessment. Mitigations include strict input validation and output sanitization to "wrap" the TPC, reducing the privileges or access of code using the TPC, or changing the configuration of the TPC.
DG Artifacts	DG 1/2/3: • Documented criteria for accepting third party components; notes, checklists, or minutes of reviews for component acceptability.
Responsible Roles	Program Manager; Procurement
Implementation Examples	1. Review and evaluate third-party software components in the context of their expected use. If a component is to be used in a substantially different way in the future, perform the review and evaluation again with that new context in mind. 2. Determine secure configurations for software components, and make these available (e.g. as configuration-as-code) so developers can readily use the configurations. 3. Obtain provenance information (e.g. SBOM, source composition analysis, binary software composition analysis) for each software component and analyse that information to better assess the risk that the component may introduce. 4. Establish one or more software repositories to host sanctioned and vetted open-source components. 5. Maintain a list of organization-approved commercial software components and component versions along with their provenance data. 6. Designate which components "must" [i.1] be included in software to be developed. 7. Implement processes to update deployed software components to newer versions and retain older versions of software components until all transitions from those versions have been completed successfully. 8. If the integrity or provenance of acquired binaries cannot be confirmed, build binaries from source code after verifying the source code's integrity and provenance.

Task ID PW.4.4	Verify that acquired commercial, open-source, and all other third-party software components comply with the requirements, as defined by the organization, throughout their life cycles.
CSC Safeguard; SAFECode practices	16.5; Manage security risk inherent in use of third party components
SSDIF PW.4.4 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** continuously monitor the TPC to ensure that its risk profile remains acceptable over time. 2. **shall** at a minimum, revisit the TPC's risk and maintenance status when making updates to the software that consumes the TPC. DG 2/3: The organisation: 1. **should** run tools to verify attack surface and freedom from known vulnerabilities for third party components.
DG Artifacts	DG1: • Documented process for TPC and associated risks. • Tool outputs to support TPC risk and maintenance status. DG2/3: • Tool outputs to validate TPC security.
Responsible Roles	Program Manager; Procurement
Implementation Examples	1. Regularly check whether there are publicly known vulnerabilities in the software modules and services that vendors have not yet fixed. 2. Build into the toolchain automatic detection of known vulnerabilities in software components. 3. Use existing results from commercial services for vetting the software modules and services. 4. Ensure that each software component is still actively maintained and has not reached end of life; this should include new vulnerabilities found in the software being remediated. 5. Determine a plan of action for each software component that is no longer being maintained or will not be available in the near future. 6. Confirm the integrity of software components through digital signatures or other mechanisms. 7. Review, analyse, and/or test code. See PW.7 and PW.8.

5.5 Essential 5: Code Integrity

5.5.0 Overview

Secure design and development practices enable a software development organization to produce software that addresses potential threats once it is in use, but the software development organization also plays a role in ensuring that the software that end users/organizations rely on is the software that they intended to deliver. For example, a malicious party might introduce a design or coding error into source code or binaries or manipulate the final product before delivery to its end users/organizations.

Preserving software integrity involves maintaining control over the design and code to make sure that only authorized and intended changes are made and that those changes are tracked and traceable to individual developers. That control is in part the domain of common software development practices (change control, version control, configuration management, code signing) and in part the domain of operational security (configuration of development systems to enforce least privilege and zero trust, and ensure proper authentication, authorization, and auditing of developers' actions) of the development organization's IT environment. Personnel selection and vetting may also be important in some cases.

Common software development tools and repositories can provide effective controls to protect designs and code, but the software development organization should configure and operate them securely. The Critical Security Controls [1] provide an effective and practical set of practices for protecting IT environments, including software development environments. Personnel vetting is outside the scope of the present publication but should not be ignored by software development organizations that work on sensitive or critical software.

For the implementation of this essential, an IG2 organisation employs individuals responsible for managing and protecting IT infrastructure. These organisations support multiple departments and people with differing risk profiles based on job function and mission. An IG3 enterprise employs security experts that specialize in the different facets of cybersecurity. IG3 assets and data contain sensitive information or functions that are subject to regulatory and compliance oversight.

5.5.1 SSDIF Essential 5 Tasks - Code Integrity

Task ID PO.1.1	**Identify and document all security requirements for the organization's software development infrastructures and processes, and maintain the requirements over time.**
CSC Safeguard; SAFECode practices	1.1, 2.1, 3.1, 4.1, 5.1, 6.1, 7.1, 8.1, 11.1, 12.3, 15.2, 16.1, 17.4, 18.1: N/A
SSDIF PO.1.1 DG Specific Actions	DG 1/2/3: The organisation: **1. should** implement CSC Safeguard 1.1: Establish and maintain detailed enterprise asset inventory. **2. should** implement CSC Safeguard 2.1: Establish and maintain a software inventory. **3. shall** implement CSC Safeguard 3.1: to Establish and maintain a data management process. **4. shall** implement CSC Safeguard 4.1: Establish and maintain a secure configuration process. **5. shall** implement CSC Safeguard 5.1: Establish and maintain an inventory of accounts. **6. should** implement CSC Safeguard 6.1: Establish an access granting process. **7. should** implement CSC Safeguard 7.1: Establish and maintain a vulnerability management process. **8. should** implement CSC Safeguard 8.1: Establish and maintain an audit log management process. **9. should** implement CSC Safeguard 11.1: Establish and maintain a data recovery process. DG 2/3: The organisation: **1. should** implement CSC Safeguard 12.3: Securely manage network infrastructure. **2. should** implement CSC Safeguard 15.2: Establish and maintain a service provider management policy. **3. shall** implement CSC Safeguard 16.1: Establish and maintain a secure application development process. 4. **shall** implement CSC Safeguard 17.4: Establish and maintain an incident response process. 5. **should** implement CSC Safeguard 18.1: Establish and maintain a penetration testing program.
DG Artifacts	DG 1: • Documents to show implementation of Critical Security Controls to include policies/processes for: enterprise asset inventory, software asset inventory, data management, secure configuration of enterprise assets and software, account management, access control management, vulnerability management, audit log management, data recovery, and incident response management. • Configurations to show implementation of policies/processes and safeguards. DG 2/3: • Documents to show implementation of IG2/3 safeguards to include: • Policies Process for: Network Infrastructure Management, Service Provider Management, Application Software Security, Penetration Testing. • Configurations to show implementation of policies/processes and safeguards.
Responsible Roles	CISO Team
Implementation Examples	1. Define policies for securing software development infrastructures and their components, including development endpoints, throughout the SDLC and maintaining that security. 2. Define policies for securing software development processes throughout the SDLC and maintaining that security, including for open-source and other third-party software components utilized by software being developed. 3. Review and update security requirements at least annually, or sooner if there are new requirements from internal or external sources, or a major security incident targeting software development infrastructure has occurred. 4. Educate affected individuals on impending changes to requirements.

Task ID PO.3.2	Follow recommended security practices to deploy, operate, and maintain tools and toolchains.
CSC Safeguard; SAFECode practices	4.1 thru 4.12, 7.1 thru 7.7; N/A
SSDIF PO.3.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** implement CSC Safeguard 4.1: Establish and maintain a secure configuration process. 2. **should** implement CSC Safeguard 4.6: Securely manage enterprise assets and software. 3. **should** implement CSC Safeguard 4.7: Manage default accounts on enterprise assets and software. 4. **should** implement CSC Safeguard 7.1: Establish and maintain a vulnerability management process. 5. **should** implement CSC Safeguard 7.2: Establish and maintain a remediation process. 6. **should** implement CSC Safeguard 7.3: Perform automated operating system patch management. 7. **should** implement CSC Safeguard 7.4: Perform automated application patch management. DG 2/3 1. **should** implement CSC Safeguard 4.8: Uninstall or disable unnecessary services on enterprise assets and software. 2. **should** implement CSC Safeguard 7.5: Perform automated vulnerability scans of internal enterprise assets. 3. **should** implement CSC Safeguard 7.6: Perform automated vulnerability scans of externally-exposed enterprise assets.
DG Artifacts	DG 1/2/3: Configurations of tools, work items in the secure development process workflow systems
Responsible Roles	CISO team; Engineering Group Build team; SDL team
Implementation Examples	1. Evaluate, select, and acquire tools, and assess the security of each tool. 2. Integrate tools with other tools and existing software development processes and workflows. 3. Use code-based configuration for toolchains (e.g. pipelines-as-code, toolchains-as-code). 4. Implement the technologies and processes needed for reproducible builds. 5. Update, upgrade, or replace tools as needed to address tool vulnerabilities or add new tool capabilities. 6. Continuously monitor tools and tool logs for potential operational and security issues, including policy violations and anomalous behavior. 7. Regularly verify the integrity and check the provenance of each tool to identify potential problems. 8. See PW.6 regarding compiler, interpreter, and build tools. 9. See PO.5 regarding implementing and maintaining secure environments.

Task ID PO.5.1	Separate and protect each environment involved in software development.
CSC Safeguard; SAFECode practices	6.3, 6.4, 6.5, 12.2, 16.8; N/A
SSDIF PO.5.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **should** implement CSC Safeguard 6.3: Require all externally-exposed enterprise or third-party applications to enforce Multi-Factor Authentication (MFA), where supported. Enforcing MFA through a directory service or SSO provider is a satisfactory implementation of this Safeguard for CSC IG2 2. **should** implement CSC Safeguard 6.4: Require MFA for remote network access for IG2. 3. **should** implement CSC Safeguard 6.5: Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a service provided for CSC IG2. 4. **shall** implement CSC Safeguard 12.2: Design and maintain a secure network architecture that addresses segmentation, least privilege, and availability, at a minimum. Example implementations may include documentation, policy, and design components for CSC IG2. 5. **should** implement CSC Safeguard 16.8: Maintain separate environments for production and non-production systems for CSC IG2. DG 2/3: The organisation**:** 1. **should** implement CSC Safeguard 6.3: Require all externally-exposed enterprise or third-party applications to enforce MFA, where supported. Enforcing MFA through a directory service or SSO provider is a satisfactory implementation of this Safeguard for CSC IG3. 2. **should** implement CSC Safeguard 6.4: Require MFA for remote network access for CSC IG3. 3. **should** implement CSC Safeguard 6.5: Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a service provider for CSC IG3. 4. **shall** implement CSC Safeguard 12.2: Design and maintain a secure network architecture which addresses segmentation, least privilege, and availability, at a minimum. Example implementations may include documentation, policy, and design components for CSC IG3. 5. **should** implement CSC Safeguard 16.8: Maintain separate environments for production and non-production systems for CSC IG3.
DG Artifacts	DG 1: Documents to show implementation of Critical Security Controls to include: • Policies/Process for: enterprise asset inventory, software asset inventory, data management, secure configuration of enterprise assets and software, account management, access control management, vulnerability management, audit log management, data recovery, security awareness and skills training, incident response management. • Configurations to show implementation of policies/processes and safeguards. DG 2/3: • Documents to show implementation of IG2/3 safeguards to include: • Policies Process for: Network Infrastructure Management, Service Provider Management, Application Software Security, Penetration Testing. • Configurations to show implementation of policies/processes and Safeguards.
Responsible Roles	CISO Team

Implementation Examples	1. Use multi-factor, risk-based authentication and conditional access for each environment. 2. Use network segmentation and access controls to separate the environments from each other and from production environments, and to separate components from each other within each non-production environment, in order to reduce attack surfaces and attackers' lateral movement and privilege/access escalation. 3. Enforce authentication and tightly restrict connections entering and exiting each software development environment, including minimizing access to the internet to only what is necessary. 4. Minimize direct human access to toolchain systems, such as build services. Continuously monitor and audit all access attempts and all use of privileged access. 5. Minimize the use of production-environment software and services from non-production environments. 6. Regularly log, monitor, and audit trust relationships for authorization and access between the environments and between the components within each environment. 7. Continuously log and monitor operations and alerts across all components of the development environment to detect, respond, and recover from attempted and actual cyber incidents. 8. Configure security controls and other tools involved in separating and protecting the environments to generate artifacts for their activities. 9. Continuously monitor all software deployed in each environment for new vulnerabilities and respond to vulnerabilities appropriately following a risk-based approach. 10. Configure and implement measures to secure the environments hosting infrastructures following a zero trust architecture.
Task ID PO.5.2	**Secure and harden development endpoints (i.e. endpoints for software designers, developers, testers, builders, etc.) to perform development-related tasks using a risk-based approach.**
CSC Safeguard; SAFECode practices	4.1 thru 4.12; N/A

SSDIF PO.5.2 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** implement CSC Safeguard 4.1: Establish and maintain a documented secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard. 2. **shall** implement CSC Safeguard 4.2: Establish and maintain a documented secure configuration process for network devices. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard. 3. **shall** implement CSC Safeguard 4.3: Configure automatic session locking on enterprise assets after a defined period of inactivity. For general purpose operating systems, the period customary maximum is 15 minutes. For mobile end-user devices, the period is a maximum of 2 minutes. 4. **shall** implement CSC Safeguard 4.4: Implement and manage a firewall on servers, where supported. Example implementations include a virtual firewall, operating system firewall, or a third-party firewall agent. 5. **shall** implement CSC Safeguard 4.5: Implement and manage a host-based firewall or port-filtering tool on end-user devices, with a default-deny rule that drops all traffic except those services and ports that are explicitly allowed. 6. **shall** implement CSC Safeguard 4.6 Securely manage enterprise assets and software. Example implementations include managing configuration through version-controlled Infrastructure-as-Code (IaC) and accessing administrative interfaces over secure network protocols, such as Secure Shell (SSH) and Hypertext Transfer Protocol Secure (HTTPS). Do not use insecure management protocols, such as Telnet (Teletype Network) and HTTP, unless operationally essential. 7. **shall** implement CSC Safeguard 4.7: Manage default accounts on enterprise assets and software, such as root, administrator, and other pre-configured vendor accounts. Example implementations can include: disabling default accounts or making them unusable. 8. **shall** implement CSC Safeguard 4.8: Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function. 9. **shall** implement CSC Safeguard: 4.9: Configure trusted DNS servers on network infrastructure. Example implementations include configuring network devices to use enterprise-controlled DNS servers and/or reputable externally accessible DNS servers. 10. **shall** implement CSC Safeguard 4.10: Enforce automatic device lockout following a predetermined threshold of local failed authentication attempts on portable end-user devices, where supported. For laptops, do not allow more than 20 failed authentication attempts; for tablets and smartphones, no more than 10 failed authentication attempts.. 11. **shall** implement CSC Safeguard 4.11: Remotely wipe enterprise data from enterprise-owned portable end-user devices when deemed appropriate such as lost or stolen devices, or when an individual no longer supports the enterprise. DG 2/3: The organisation**:** 1. **should** implement CSC Safeguard 4.12: Ensure separate enterprise workspaces are used on mobile end-user devices, where supported. Work Profile to separate enterprise applications.
DG Artifacts	DG 1/2/3: • Documentation describing the organization's secure configuration process. • Configurations to show implementation of the secure configuration process and Safeguards 4.1 thru 4.12.
Responsible Roles	CISO Team
Implementation Examples	1. Configure each development endpoint based on approved hardening guides, checklists, etc.; for example, enable FIPS-compliant encryption of all sensitive data at rest and in transit. 2. Configure each development endpoint and the development resources to provide the least functionality needed by users and services and to enforce the principle of least privilege. 3. Continuously monitor the security posture of all development endpoints, including monitoring and auditing all use of privileged access. 4. Configure security controls and other tools involved in securing and hardening development endpoints to generate artifacts for their activities. 5. Require multi-factor authentication for all access to development endpoints and development resources. 6. Provide dedicated development endpoints on non-production networks for performing all development-related tasks. Provide separate endpoints on production networks for all other tasks. 7. Configure each development endpoint following a zero trust architecture.

Task ID PS.1.1	**Store all forms of code - including source code, executable code, and configuration-as-code - based on the principle of least privilege so that only authorized personnel, tools, services, etc. have access.**
CSC Safeguard; SAFECode practices	6.1 thru 6.8; N/A
SSDIF PS.1.1 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** implement CSC Safeguard 6.1: Establish and follow a documented process, preferably automated, for granting access to enterprise assets upon new hire or role change of a user. 2. **shall** implement CSC Safeguard 6.2: Establish and follow a process, preferably automated, for revoking access to enterprise assets, through disabling accounts immediately upon termination, rights revocation, or role change of a user. Disabling accounts, instead of deleting accounts, may be necessary to preserve audit trails. 3. **shall** implement CSC Safeguard 6.3: Require MFA for Externally-Exposed Applications 4. **shall** implement CSC Safeguard 6.4: Require MFA for Remote Network Access 5. **shall** implement CSC Safeguard 6.5: Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a service provider. 6. **shall** implement CSC Safeguard 6.6: Establish and maintain an inventory of the enterprise's authentication and authorization systems, including those hosted on-site or at a remote service provider. Review and update the inventory, at a minimum, annually, or more frequently. 7. **shall** implement CSC Safeguard 6.7: Centralize access control for all enterprise assets through a directory service or SSO provider, where supported. DG 2/3: The organisation**:** 1. **shall** implement CSC Safeguard 6.8: Define and maintain role-based access control, through determining and documenting the access rights necessary for each role within the enterprise to successfully carry out its assigned duties. Perform access control reviews of enterprise assets to validate that all privileges are authorized, on a recurring schedule at a minimum annually, or more frequently.
DG Artifacts	DG 1/2/3: • Documentation describing access control management process. • Configurations to show implementation of the access management process and Safeguards 6.1 thru 6.8.
Responsible Roles	CISO Team; Engineering Group Build team
Implementation Examples	1. Store all source code and configuration-as-code in a code repository and restrict access to it based on the nature of the code. For example, open-source code intended for public access may need its integrity and availability protected; other code may also need its confidentiality protected. 2. Use version control features of the repository to track all changes made to the code with accountability to the individual account. 3. Use commit signing for code repositories. 4. Have the code owner review and approve all changes made to the code by others. 5. Use code signing to help protect the integrity of executables. 6. Use cryptography (e.g. cryptographic hashes) to help protect file integrity.

Task ID PS.2.1	**Make software integrity verification information available to software acquirers.**
CSC Safeguard; SAFECode practices	N/A; Organization's software release policy mandates that code be signed before release to customers [i.47]
SSDIF PS.2.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** sign all code delivered to customers.
DG Artifacts	DG 1/2/3: • Signed code.
Responsible Roles	Release Engineer or SRE
Implementation Examples	1. Post cryptographic hashes for release files on a well-secured website. 2. Use an established certificate authority for code signing so that consumers' operating systems or other tools and services can confirm the validity of signatures before use. 3. Periodically review the code signing processes, including certificate renewal, rotation, revocation, and protection.

Task ID PS.3.1	Securely archive the necessary files and supporting data (e.g. integrity verification information, provenance data) to be retained for each software release.
CSC Safeguard; SAFECode practices	11.3; N/A
SSDIF PS.3.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** document a process to securely archive necessary files and supporting data. 2. **shall** implement its process to securely archive necessary files and supporting data.
DG Artifacts	DG 1/2/3: • Documented process. • Workflow records created as part of the secure development process. For example, the configuration management repository contains source code and documentation and the workflow system records review and approved changes to source code. • Files and data present in archive repository.
Responsible Roles	Release Engineer or SRE
Implementation Examples	1. Store the release files, associated images, etc. in repositories following the organization's established policy. Allow read-only access to them by necessary personnel and no access by anyone else. 2. Store and protect release integrity verification information and provenance data, such as by keeping it in a separate location from the release files or by signing the data.

Task ID PS.3.2	Collect, safeguard, maintain, and share provenance data for all components of each software release (e.g. in a software bill of materials [SBOM]).
CSC Safeguard; SAFECode practices	16.4; Manage security risk inherent in use of third party components
SSDIF PS.3.2 DG Specific Actions	DG 1/2/3: The organisation**:** 1. **shall** maintain a list of approved third-party components. 2. **shall** maintain a list of third-party components that are used in each product or online service and where they're used. (SBOM plus internal where used). DG 2/3: The organisation**:** 1. **shall** use software composition analysis (SCA) tools to identify embedded third-party components and verify that only approved components are used.
DG Artifacts	DG 1: • List of approved third-party components. • SBOM for each product DG 2/3: • List of SCA tools and tool outputs.
Responsible Roles	Program Manager; Release Engineer
Implementation Examples	1. Make the provenance data available to software acquirers in accordance with the organization's policies, preferably using standards-based formats. 2. Make the provenance data available to the organization's operations and response teams to aid them in mitigating software vulnerabilities. 3. Protect the integrity of provenance data and provide a way for recipients to verify provenance data integrity. 4. Update the provenance data every time any of the software's components are updated.

5.6 Essential 6: Vulnerability Disclosure and Remediation

5.6.0 Overview

Vulnerability response and remediation are part of any secure development process. Software that is perfect and vulnerability-free is infeasible except in very limited circumstances. Development teams should implement processes to accept reports of product vulnerabilities, investigate the reports, and remedy the vulnerabilities. They should encourage vulnerability researchers to report and should consider sponsoring "bug bounties" to achieve that end. They should ensure that vulnerability advisories and fixes are communicated to their users, ideally in a standardized form such as Common Security Advisory Framework (CSAF) [i.43] using CVE/NVD [i.100] or GCVE/EUVD [i.101] and accompanied by context on their severity [i.28] and exploitability [i.45].

Vulnerability reports are also a critical part of secure development. All software development organizations (regardless of DG level) should be prepared to accept vulnerability reports, respond, do root cause analysis, and track their work. Software development organizations should not only fix the vulnerabilities that are reported to them but also search for other similar vulnerabilities that have not yet been discovered and use the results of root cause analysis as input to process improvement - by updating tools or building new ones, by updating training, and by revising threat models and updating default configurations. Where a software development organization relies on tools developed by other software development organizations - for example, a commercial or open-source threat modelling or static analysis tool - it should provide feedback to the tool vendor if improvements would make the tool better able to detect a new class of vulnerability.

There are three phases of vulnerability response:

- Immediate remediation of reported vulnerabilities and release of software updates.
- Remediation of vulnerabilities similar to the one reported and release of software updates or planned service releases.
- Updating of secure development practices on a regularly scheduled basis to keep newly discovered classes of vulnerabilities from occurring in the future.

5.6.1 SSDIF Essential 6 Tasks - Vulnerability Disclosure and Remediation

Task ID RV.1.1	**Gather information from software acquirers, users, and public sources on potential vulnerabilities in the software and third-party components that the software uses and investigate all credible reports.**
CSC Safeguard; SAFECode practices	7.2, 7.2, 7.3, 16.2; Create and Manage a Vulnerability Response Process
SSDIF RV.1.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** create an externally-facing policy for outside stakeholders on how they report a potential vulnerability and what response they can expect when a potential vulnerability is found. 2. **shall** comply with policy requirements for applicable vulnerability reporting specifications. 3. **shall** provide a visible location for vulnerability reporters, security researchers, customers or other stakeholders to report vulnerabilities. 4. **shall** ensure that knowledge of where, how and to whom report product vulnerabilities is easily discoverable. Discovery can occur via the organisation website along with a dedicated email address. 5. **shall** provide a way for sensitive vulnerability information to be communicated confidentially, if possible, such as upload to an TLS-protected website, encrypted email, or encrypted messaging app. 6. **shall** provide information concerning where to report, how, and to whom.
DG Artifacts	DG 1/2/3: • Policy for vulnerability response. • Available and visible location for external reporting of vulnerabilities. • Available service to securely accept sensitive vulnerability information.
Responsible Roles	Program Manager; Procurement; Security Response Engineer
Implementation Examples	1. Monitor vulnerability databases, security mailing lists, and other sources of vulnerability reports through manual or automated means. 2. Use threat intelligence sources to better understand how vulnerabilities in general are being exploited. 3. Automatically review provenance and software composition data for all software components to identify any new vulnerabilities they have.

Task ID RV.1.2	Review, analyse, and/or test the software's code to identify or confirm the presence of previously undetected vulnerabilities.
CSC Safeguard; SAFECode practices	16.2; Create and Manage a Vulnerability Response Process
SSDIF RV.1.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** ensure the team that owns the vulnerable code initially triage the reported vulnerability to validate it, prioritize it by severity and, finally, remediate it. 2. **shall** ensure timely completion of the initial triage - commonly accomplished in one week.
DG Artifacts	DG 1/2/3: • Configuration of tools, vulnerability and remediation work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Test Engineer; Software Engineer
Implementation Examples	1. Configure the toolchain to perform automated code analysis and testing on a regular or continuous basis for all supported releases. 2. See Task IDs PW.7 and PW.8.

Task ID RV.1.3	Have a policy that addresses vulnerability disclosure and remediation, and implement the roles, responsibilities, and processes needed to support that policy.
CSC Safeguard; SAFECode practices	16.2; Create and Manage a Vulnerability Response Process
SSDIF RV.1.3 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** maintain a vulnerability handling policy. The policy may exist in hard copy, on an internal website, or in another organization-selected form that defines who is responsible in each stage of the vulnerability handling process and how they handle information on potential and confirmed vulnerabilities. 2. **shall** ensure the policy complies with the requirements of relevant vulnerability reporting norms.
DG Artifacts	DG 1/2/3: • Documented vulnerability handling policy.
Responsible Roles	Security Response Engineer
Implementation Examples	1. Establish a vulnerability disclosure program and make it easy for security researchers to learn about your program and report possible vulnerabilities. 2. Have a Product Security Incident Response Team (PSIRT) and processes in place to handle the responses to vulnerability reports and incidents, including communications plans for all stakeholders. 3. Have a security response playbook to handle a generic reported vulnerability, a report of zero-days, a vulnerability being exploited in the wild, and a major ongoing incident involving multiple parties and open-source software components. 4. Periodically conduct exercises of the product security incident response processes.

Task ID RV.2.1	Analyse each vulnerability to gather sufficient information about risk to plan its remediation or other risk response.
CSC Safeguard; SAFECode practices	16.2; Create and Manage a Vulnerability Response Process
SSDIF RV.2.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** review the source code to understand the bug and plan an update or mitigation.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. • Records of bug analysis in the product workflow system.
Responsible Roles	Security Response Engineer; Software Engineer; Program Manager
Implementation Examples	1. Use existing issue tracking software to record each vulnerability. 2. Perform risk calculations for each vulnerability based on estimates of its exploitability, the potential impact if exploited, and any other relevant characteristics.

Task ID RV.2.2	Plan and implement risk responses for vulnerabilities.
CSC Safeguard; SAFECode practices	16.2; Create and Manage a Vulnerability Response Process
SSDIF RV.2.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** implement and test the vulnerability fix. 2. **shall,** where possible, include a common identifier for each reported vulnerability, such as an entry in the required Vulnerability Database and Common Vulnerabilities and Exposures specifications. 3. **shall** assign a criticality to help users decide how quickly to apply the patch.
DG Artifacts	DG 1/2/3: • Entry in the workflow system and the fixed code. • Entries in the required Vulnerability Database, when appropriate.
Responsible Roles	Security Response Engineer; Software Engineer; Program Manager
Implementation Examples	1. Make a risk-based decision as to whether each vulnerability will be remediated or if the risk will be addressed through other means (e.g. risk acceptance, risk transference) and prioritize any actions to be taken. 2. If a permanent mitigation for a vulnerability is not yet available, determine how the vulnerability can be temporarily mitigated until the permanent solution is available, and add that temporary remediation to the plan. 3. Develop and release security advisories that provide the necessary information to software acquirers, including descriptions of what the vulnerabilities are, how to find instances of the vulnerable software, and how to address them (e.g. where to get patches and what the patches change in the software; what configuration settings may need to be changed; how temporary workarounds could be implemented). 4. Deliver remediations to acquirers via an automated and trusted delivery mechanism. A single remediation could address multiple vulnerabilities. 5. Update records of design decisions, risk responses, and approved exceptions as needed. See PW.1.2.

Task ID RV.3.1	Analyse identified vulnerabilities to determine their root causes.
CSC Safeguard; SAFECode practices	16.3; Perform Root Cause Analysis
SSDIF RV.3.1 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** ensure the software development team consider the nature of the defect and determine if it is a recurring issue, and if so, prioritize the aspect of the software security program that specifically targets that flaw. 2. **should** add tools or update developer training accordingly. 3. **should** document trends over time.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Architect; Software Engineer; SDL Team
Implementation Examples	1. Record the root cause of discovered issues. 2. Record lessons learned through root cause analysis in a wiki that developers can access and search.

Task ID RV.3.2	**Analyse the root causes over time to identify patterns, such as a particular secure coding practice not being followed consistently.**
CSC Safeguard; SAFECode practices	16.3; Perform Root Cause Analysis
SSDIF RV.3.2 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** ensure the software development team consider the nature of the defect and determine if it is a recurring issue, and if so, prioritize the aspect of the software security program that specifically targets that flaw. 2. **should** add tools or update developer training accordingly. 3. **should** document trends over time.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	SDL Team
Implementation Examples	1. Record lessons learned through root cause analysis in a wiki that developers can access and search. 2. Add mechanisms to the toolchain to automatically detect future instances of the root cause. 3. Update manual processes to detect future instances of the root cause.

Task ID RV.3.3	**Review the software for similar vulnerabilities to eradicate a class of vulnerabilities and proactively fix them rather than waiting for external reports.**
CSC Safeguard; SAFECode practices	16.3, 16.9; Perform Root Cause Analysis
SSDIF RV.3.3 DG Specific Actions	DG 1/2/3: The organisation: 1. **shall** ensure the software development team review relevant components for vulnerabilities similar to the one reported, plan a strategy to fix them, and fix them in an update or service release. DG 3: 1. **should** use root cause analysis results to select tools and/or methods that address the types of vulnerabilities most often found.
DG Artifacts	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Responsible Roles	Architect; Program Manager; Software Engineer
Implementation Examples	1. See PW.7 and PW.8.

Task ID RV.3.4	**Review the SDLC process and update it if appropriate to prevent (or reduce the likelihood of) the root cause recurring in updates to the software or in new software that is created.**
CSC Safeguard; SAFECode practices	16.3, 16.9; Perform Root Cause Analysis
SSDIF RV.3.4 DG Specific Actions	DG 1/2: The organisation: None DG 3: The organisation: 1. **should** focus training initiatives on methods that will help solve the problems seen from root cause analysis or problems similar organizations are seeing to eliminate classes of vulnerabilities. 2. **should** use root cause analysis results to select tools and/or methods that address the types of vulnerabilities most often found
DG Artifacts	DG 3: • Updates to training addressing problems found in root cause analysis to eliminate classes of vulnerabilities.
Responsible Roles	SDL Team
Implementation Examples	1. Record lessons learned through root cause analysis in a wiki that developers can access and search. 2. Plan and implement changes to the appropriate SDLC practices.

Annex A (informative): EU Cyber Resilience Act (CRA) annex requirements to SSDIF Actions

CRA Annex	CRA Provisions	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
I.I.	**Cybersecurity requirements relating to the properties of products with digital elements** [i.12]		
I.I(1)	designing, developing and producing products with digital elements in such a way that they ensure an appropriate level of cybersecurity based on the risks	All	
I.I(2)(a)	making products with digital elements available on the market without known exploitable vulnerabilities	All	
I.I(2)(b)	making products with digital elements available on the market with a secure by default configuration	Secure Default Configuration	*PW.9.1, *PW.9.2
I.I(2)(c)	ensuring that vulnerabilities in products with digital elements can be addressed through security updates	Vulnerability Disclosure and Remediation	*RV.2.1, *RV.2.2
I.I(2)(d)	ensuring protection of products with digital elements from unauthorised access and reporting on possible unauthorised access	Secure Software Design	*PO.1.2, *PW.1.1, *PW.1.2, *PW.1.3, *PW.2.1
I.I(2)(e)	protecting the confidentiality of data stored, transmitted or otherwise processed by a product with digital elements	Secure Software Design	*PO.1.2, *PW.1.1, *PW.1.2, *PW.1.3, *PW.2.1
I.I(2)(f)	protecting the integrity of data, commands, programs by a product with digital elements, and its configuration against any manipulation or modification not authorised by the user, as well as reporting on corruptions	Secure Software Design	*PO.1.2, *PW.1.1, *PW.1.2, *PW.1.3, *PW.2.1
I.I(2)(g)	processing only personal or other data that are adequate, relevant and limited to what is necessary in relation to the intended purpose of the product with digital elements ('minimisation of data')	Secure Software Design	*PO.1.2, *PW.1.1
I.I(2)(h)	protecting the availability of essential and basic functions of the product with digital elements	Secure Software Design	*PO.1.2, *PW.1.1, *PW.1.2, *PW.1.3, *PW.2.1
I.I(2)(i)	minimising the negative impact of a product with digital elements or its connected devices on the availability of services provided by other devices or networks	Secure Software Design	*PO.1.2, *PW.1.1, *PW.1.2, *PW.1.3, *PW.2.1
I.I(2)(j)	designing, developing and producing products with digital elements with limited attack surfaces	Secure Default Configuration	*PW.9.1, *PW.9.2
I.I(2)(k)	designing, developing and producing products with digital elements that reduce the impact of an incident using appropriate exploitation mitigation mechanisms and techniques	Secure Software Design; Secure Development	*PW.1.1, *PW.6.2
I.I(2)(l)	providing security related information by recording and/or monitoring relevant internal activity of products with digital elements with an opt-out mechanism for the user	Secure Software Design	*PW.1.1
I.I(2)(m)	securely and easily removing or transferring all data and settings of a product with digital elements.	Not covered in SSDIF	
I.II.	**Vulnerability handling requirements**		
I.II(1)	identify and document vulnerabilities and components contained in the product, 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 product;	Supply Chain Security	*PW.4.1, *PW.4.4,
I.II(2)	in relation to the risks posed to the products with digital elements, address and remediate vulnerabilities without delay, including by providing security updates; where technically feasible, new security updates "shall" [i.12] be provided separately from functionality updates;	Vulnerability Disclosure and Remediation	*RV.1 (all tasks), *RV.2 (all tasks)

CRA Annex	CRA Provisions	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
I.II(3)	apply effective and regular tests and reviews of the security of the product with digital elements;	Secure Development	*PW.8.1, * PW.8.2
I.II(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;	Vulnerability Disclosure and Remediation	*RV.2.2
I.II(5)	put in place and enforce a policy on coordinated vulnerability disclosure;	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3
I.II(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;	Vulnerability Disclosure and Remediation	*RV.1.1
I.II(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;	Vulnerability Disclosure and Remediation	*RV.2.2
I.II(8)	ensure that, where security updates are available to address identified security issues, they are disseminated without delay and, unless otherwise agreed between manufacturer and 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	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3, *RV.2.1, *RV.2.2
III.	**Important products with digital elements**		
III.I.1	identity management systems and privileged access management software and hardware, including authentication and access control readers, including biometric readers	All	
III.I.2	standalone and embedded browsers	All	
III.I.3	password managers	All	
III.I.4	software that searches for, removes, or quarantines malicious software	All	
III.I.5	products with digital elements with the function of virtual private network (VPN)	All	
III.I.6	network management systems	All	
III.I.7	Security information and event management (SIEM) systems	All	
III.I.8	boot managers	All	
III.I.9	public key infrastructure and digital certificate issuance software	All	
III.I.10	physical and virtual network interfaces	All	
III.I.11	operating systems	All	
III.I.12	routers, modems intended for the connection to the internet, and switches	All	
III.I.13	microprocessors with security-related functionalities	All	
III.I.14	microcontrollers with security-related functionalities	All	
III.I.15	application specific integrated circuits (ASIC) and field-programmable gate arrays (FPGA) with security-related functionalities	All	
III.I.16	smart home general purpose virtual assistants	All	

CRA Annex	CRA Provisions	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
III.I.17	smart home products with security functionalities, including smart door locks, security cameras, baby monitoring systems and alarm systems	All	
III.I.18	Internet connected toys covered by Directive 2009/48/EC that have social interactive features (e.g. speaking or filming) or that have location tracking features	All	
III.I.19	personal wearable products to be worn or placed on a human body that have a health monitoring (such as tracking) purpose and to which Regulation (EU) 2017/745 or Regulation (EU) 2017/746 do not apply or personal wearable products that are intended for the use by and for children	All	
III.II.1	hypervisors and container runtime systems that support virtualised execution of operating systems and similar environments	All	
III.II.2	firewalls, intrusion detection and/or prevention systems, including specifically those intended for industrial use	All	
III.II.3	tamper-resistant microprocessors	All	
III.II.4	tamper-resistant microcontrollers	All	
IV.	**Critical products with digital elements**		
IV.1	hardware devices with security boxes	All	
IV.2	smart meter gateways within smart metering systems as defined in Article 2 (23) of Directive (EU) 2019/944 and other devices for advanced security purposes, including for secure cryptoprocessing	All	
IV.3	smartcards or similar devices, including secure elements	All	
V.	**EU declaration of conformity**	All	
VI.	**Simplified EU declaration of conformity**	All	
VII	**Contents of technical documentation**	All	
VIII	**Conformity assessment procedures**	All	

Annex B (informative): SSDIF Additional Informative Mappings

B.1 UK NCSC Cyber Resilience Testing (CRT) Assurance Principles and Claims (APC) to SSDIF Actions

	CRT APC Principle		CRT APC Claim	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
1	Secure design and development				
1.1	Follow an established secure development framework.	1.1.1	The development framework used is documented.	Secure Development	*PO.1.2
		1.1.2	Developers are trained in the use of the framework and tools.	Secure Development	*PO.2.2
		1.1.3	Tools are maintained and updated.	Secure Development	*PO.3.1, *PO.3.2
		1.1.4	Items requiring configuration control are identified and version control is used.	Secure Development	*PO.3.2, *PO.3.3, *PW.1.2 (many tasks are documented by artifacts in the configuration management system)
		1.1.5	Requirements are captured and recorded.	Secure Software Design; Secure Development	*PO.4.1, *PO.4.2, *PW.1.2
		1.1.6	Software is designed for user need.	Secure Software Design	*PW.1.1, *PW.1.2, *PW.1.3
1.2	Understand the composition of the software and assess risks linked to the ingestion and maintenance of third-party components throughout the development lifecycle.	1.2.1	All third-party components are identified and documented.	Supply Chain Security	*PW.4.1
		1.2.2	Integrity of third-party components and updates is verified.	Supply Chain Security	*PW.4.1
		1.2.3	Each third-party component is tested before being first deployed.	Supply Chain Security	*PO.1.3, *PW.4.1
		1.2.4	Third-party component updates are tested.	Supply Chain Security	*PO.1.3, *PW.4.1
		1.2.5	Processes are in place to manage and deploy updates to third-party components.	Supply Chain Security, Vulnerability Disclosure and Remediation	*PO.1.3, *PW.4.1, *RV [all tasks]
1.3	Have a clear process for testing software and software updates before distribution.	1.3.1	A test plan exists that covers all requirements and third-party components.	Secure Development	*PW.8.1, *PW.8.2
		1.3.2	Execution of the test plan is automated and repeatable wherever possible.	Secure Development	*PW.8.2
		1.3.3	Defects identified during testing are addressed.	Secure Development	*PW.8.2

	CRT APC Principle		CRT APC Claim	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
1.4	Follow secure by design and secure by default principles throughout the development lifecycle of the software.	1.4.1	Techniques to understand how the software might be exploited (threat modelling) have been used in the design of the software.	Secure Software Design	*PW.1.1
		1.4.2	Multi-factor authentication for privileged users of the software is enforced.	Secure Software Design; Secure Default Configuration	*PW.1.1, *PW.9.1, *PW.9.2
		1.4.3	Default (and persistent) passwords are not used.	Secure Default Configuration	*PW.9.1, *PW.9.2
		1.4.4	Data input to the software is validated.	Secure Development	*PW.5.1
		1.4.5	Credentials and sensitive data are securely stored.	Secure Software Design; Secure Development, Secure Default Configuration	*PW.1.1, *PW.2.1, *PW.9.1, *PW.9.2
2	Build environment security				
2.1	Protect the build environment against unauthorised access.	2.1.1	Roles are defined that specify the data and functionality that each role is allowed to access.	Code Integrity	*PO.5.1, *PS.1.1
		2.1.2	Users of the build environment are required to authenticate on a regular basis.	Code Integrity	*PO.5.1, *PO.5.2
		2.1.3	Users of the build environment are issued with credentials bound to their role.	Code Integrity	*PO.5.1, *PO.5.2, *PS.1.1
		2.1.4	Credentials are securely managed and stored.	Code Integrity	*PO.5.1
		2.1.5	Credentials are multi factor.	Code Integrity	*PO.5.1, *PO.5.2
		2.1.6	Users with access to the build environment are regularly reviewed to ensure they still have a legitimate need.	Code Integrity	*PO.5.1, *PS.1.1
2.2	Control and log changes to the build environment.	2.2.1	Access and changes to the build environment are logged.	Code Integrity	*PO.5.1, *PS.3.2
		2.2.2	Only authorised personnel can make changes to the build environment.	Code Integrity	*PO.5.1, *PO.5.2, *PS.1.1
		2.2.3	Logs are auditable and retained for an agreed period.	Code Integrity	*PS.3.1
		2.2.4	The confidentiality and integrity of logs is protected.	Code Integrity	*PS.3.1
3	Secure deployment and maintenance				
3.1	Distribute software securely to customers.	3.1.1	The integrity of software (including updates) can be verified in the customer environment.	Code Integrity	*PS.2.1
		3.1.2	Software (including updates) is distributed over trusted channels.	Code Integrity	*PS.2.1, *PS.3.2

	CRT APC Principle		CRT APC Claim	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
3.2	Implement and publish an effective vulnerability disclosure process.	3.2.1	A vulnerability disclosure policy and process are published.	Vulnerability Disclosure and Remediation	*RV.1.1
		3.2.2	The vulnerability disclosure process describes how to confidentially report vulnerabilities.	Vulnerability Disclosure and Remediation	*RV.1.1
3.3	Have processes and documentation in place for proactively detecting, prioritising and managing vulnerabilities in software components.	3.3.1	Knowledge of public vulnerabilities is kept up to date.	Vulnerability Disclosure and Remediation	*RV.1.1
		3.3.2	A vulnerability management plan exists that assesses and prioritises responses to vulnerabilities.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3
3.4	Report vulnerabilities to relevant parties where appropriate.	3.4.1	Internal security teams are informed.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3, *RV.2.1
		3.4.2	Affected customers are informed.	Vulnerability Disclosure and Remediation	*RV.2.2
3.5	Provide timely security updates, patches and notifications to customers.	3.5.1	Security updates are distributed as soon as is practicable.	Vulnerability Disclosure and Remediation	*RV.2.2
		3.5.2	Security updates are tested and secure by default.	Vulnerability Disclosure and Remediation	*PW.9.1, *RV.2.2
4	Communication with customers				
4.1	Provide information to the customer specifying the level of support and maintenance provided for the software being sold.	4.1.1	'End of support' dates are published for all software components.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3
		4.1.2	A policy on frequency of updates and the process for applying them is published.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3
		4.1.3	User documentation describes how to correctly and securely apply updates and use software.	Vulnerability Disclosure and Remediation	*PW.9.1, *RV.2.2
4.2	Provide information to the customer specifying the level of support and maintenance provided for the software being sold.	4.2.1	Customers are given at least 1 year's notice of when software will no longer be supported.	Vulnerability Disclosure and Remediation	*RV.1.1
4.3	Make information available to customers about notable incidents that may cause significant impact to customer organisations.	4.3.1	An incident support plan is published.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3
		4.3.2	Customers are informed of relevant incidents in a timely manner.	Vulnerability Disclosure and Remediation	*RV.1.1, *RV.1.3, *RV.2.2

	CRT APC Principle		CRT APC Claim	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
5	Produce specific usage, design and operation				
5.1	Design the product to be usable	5.1.1	The product's usability has been demonstrated to support people in its secure installation, use and maintenance	Secure Development	PW.8.2
		5.1.2	Guidance and support on product configuration is available to those installing and using the product	Secure Default Configuration	PW.9.2
		5.1.3	People have a mechanism to report difficulties in working with the product to the developers	Secure Default Configuration	PW.9.1
		5.1.4	If the product is reset it defaults to a known safe state	Secure Default Configuration	PW.9.1, PW.9.2
		5.1.5	Accessibility testing demonstrates the design supports disabled people in securely installing, using and maintaining the product		
		5.1.6	People are notified if the product is insecurely configured	Secure Default Configuration	PW.9.2
		5.1.7	The product is designed to allow people to easily recover from errors	Secure Development	PW.5.1
5.2	Ensure only authorised users have access to product data and functionality	5.2.1	Users are only allowed access to the data and functionality necessary to perform their role	Secure Software Design	PW.2.1
		5.2.2	Logging and auditing of access to the product is in place	Secure Software Design	PW.1.3, PW.5.1
		5.2.3	Users are required to enter their credentials before accessing any data or functionality	Secure Software Design	PW.1.3
		5.2.4	Credentials are managed securely	Secure Software Design	PW.1.3
		5.2.5	Privileged roles are defined and credentials for such roles are multi-factor	Secure Software Design	PW.1.1
		5.2.6	Credentials are unique per device or defined by the user when first used	Secure Default Configuration	PW.9.1, PW.9.2
		5.2.7	There is a way for users to recover from loss of credential	Secure Software Design	PW.1.3

	CRT APC Principle		CRT APC Claim	SSDIF Essentials	SSDIF DG Specific Action ID SSDIF-[*]
5.3	Protect sensitive data and the integrity of the product	5.3.1	All types of sensitive data are identified and documented	Secure Software Design	PW.1.1, PW.1.3
		5.3.2	Sensitive data is only transmitted to/from trusted connections	Secure Software Design	PW.1.1, PW.1.3
		5.3.3	The confidentiality and integrity of sensitive data is protected during transmission	Secure Software Design	PW.1.1, PW.1.3
		5.3.4	Where the product uses encryption a recognised cryptographic standard is used	Secure Software Design	PW.1.1, PW.1.3
		5.3.5	The integrity of software or firmware is verified when loaded	Code Integrity	PS.2.1
		5.3.6	Remote management commands that affect product operation are verified before being acted upon	Secure Software Design	PW.1.1, PW.1.3
5.4	Enable the logging and monitoring of security events	5.4.1	All security events are defined	Secure Software Design	PW.1.1, PW.1.3
		5.4.2	When a security event occurs it is logged	Secure Software Design	PW.1.1, PW.1.3
		5.4.3	The format of logging data is defined	Secure Software Design	PW.1.1, PW.1.3
		5.4.4	The integrity of logging data is protected	Secure Software Design	PW.1.1, PW.1.3
		5.4.5	Logs can only be accessed by authorised users	Secure Software Design	PW.1.1, PW.1.3

B.2 Common Frameworks and Specifications

Framework/Spec	SSDF Task [task id:framework-spec provision, paragraph]
Business Software Alliance Framework for Secure Software (BSAFSS) [i.76]	PO.1.1:SM.3, DE.1, IA.1, IA.2 ; PO.1.2:SC.1-1, SC.2, PD.1-1, PD.1-2, PD.1-3, PD.2-2, SI, PA, CS, AA, LO, EE; PO.1.3:SM.1, SM.2, SM.2-1, SM.2-4; PO.2.1:PD.2-1, PD.2-2; PO.2.2:PD.2-2; PO.3.2:DE.2; PO.3.3:PD.1-5; PO.4.1:TV.2-1, TV.5-1; PO.4.2:PD.1-4, PD.1-5; PO.5.1:DE.1, IA.1, IA.2; PO.5.2:DE.1-1, IA.1, IA.2; PS.1.1:IA.1, IA.2, SM.4-1, DE.1-2; PS.2.1:SM.4, SM.5, SM.6; PS.3.1:PD.1-5, DE.1-2, IA.2; PW.1.1:SM.2; PW.1.1:SC.1; PW.1.2:SC.1-1, PD.1-1; PW.1.3:SI.2-1, SI.2-2, LO.1; PW.2.1:TV.3; PW.4.1:SM.2; PW.4.4:SC.3-1, SM.2-1, SM.2-2, SM.2-3, TV.2, TV.3; PW.5.1:SC.2, SC.3, LO.1, EE.1; PW.6.1:DE.2-1; PW.6.2:DE.2-3, DE.2-4, DE.2-5; PW.7.2:TV.2, PD.1-4; PW.8.1:TV.3; PW.8.2:TV.3, TV.5, PD.1-4; PW.9.1:CF.1; PW.9.2:CF.1; RV.1.1:VM.1-3, VM.3; RV.1.2:VM.1-2, VM.2-1; RV.1.3:VM.1-1, VM.2; RV.2.1:VM.2; RV.2.2:VM.1-1, VM-2; RV.3.1:VM.2-1; RV.3.2:VM.2-1, PD.1-3; RV.3.3:VM.2; RV.3.4:PD.1-3
Building Security In Maturity Model (BSIMM) [i.77]	PO.2.1:SM1.1, SM2.3, SM2.7, CR1.7; PO.2.2:T1.1, T1.7, T1.8, T2.5, T2.8, T2.9, T3.1, T3.2, T3.4; PO.2.3:SM1.3, SM2.7, CP2.5; PO.3.1:CR1.4, ST1.4, ST2.5, SE2.7; PO.3.2:SR1.1, SR1.3, SR3.4; PO.3.3:SM1.4, SM3.4, SR1.3; PO.4.1:SM1.4, SM2.1, SM2.2, SM2.6, SM3.3, CP2.2; PO.4.2:SM1.4, SM2.1, SM2.2, SM3.4; PS.1.1:SE2.4; PS.2.1:SE2.4; PW.1.1:SE3.6; PW.1.1:AM1.2, AM1.3, AM1.5, AM2.1, AM2.2, AM2.5, AM2.6, AM2.7, SFD2.2, AA1.1, AA1.2, AA1.3, AA2.1; PW.1.2:SFD3.1, SFD3.3, AA2.2, AA3.2; PW.1.3:SFD1.1, SFD2.1, SFD3.2, SR1.1, SR3.4; PW.2.1:AA1.1, AA1.2, AA1.3, AA2.1, AA3.1; PW.4.1:SFD2.1, SFD3.2, SR2.4, SR3.1, SE3.6; PW.4.2:SFD1.1, SFD2.1, SFD3.2, SR1.1; PW.4.4:CP3.2, SR2.4, SR3.1, SR3.2, SE2.4, SE3.6; PW.5.1:SR3.3, CR1.4, CR3.5; PW.6.1:SE2.4; PW.6.2:SE2.4, SE3.2; PW.7.1:CR1.5; PW.7.2:CR1.2, CR1.4, CR1.6, CR2.6, CR2.7, CR3.4, CR3.5; PW.8.1:PT2.3; PW.8.2:ST1.1, ST1.3, ST1.4, ST2.4, ST2.5, ST2.6, ST3.3, ST3.4, ST3.5, ST3.6, PT1.1, PT1.2, PT1.3, PT3.1; PW.9.1:SE2.2; PW.9.2:SE2.2; RV.1.1:AM1.5, CMVM1.2, CMVM2.1, CMVM3.4, CMVM3.7; RV.1.2:CMVM3.1; RV.1.3:CMVM1.1, CMVM2.1, CMVM3.3, CMVM3.7; RV.2.1:CMVM1.2, CMVM2.2; RV.2.2:CMVM2.1; RV.3.1:CMVM3.1, CMVM3.2; RV.3.2:CP3.3, CMVM3.2; RV.3.3:CR3.3, CMVM3.1; RV.3.4:CP3.3, CMVM3.2
Cloud Native Computing Foundation Software Supply Chain Practices (CNCFSSP) [i.78]	PO.3.1:Securing Materials—Verification; Securing Build Pipelines—Verification, Automation, Secure Authentication/Access; Securing Artefacts—Verification; Securing Deployments—Verification ; PO.3.2:Securing Build Pipelines—Verification, Automation, Controlled Environments, Secure Authentication/Access; Securing Artefacts—Verification, Automation, Controlled Environments, Encryption; Securing Deployments—Verification, Automation; PO.3.3:Securing Build Pipelines—Verification, Automation, Controlled Environments; Securing Artefacts—Verification; PO.5.1:Securing Build Pipelines—Controlled Environments; PS.1.1:Securing the Source Code—Verification, Automation, Controlled Environments, Secure Authentication; Securing Materials—Automation; PS.2.1:Securing Deployments—Verification; PS.3.1:Securing Artefacts—Automation, Controlled Environments, Encryption; Securing Deployments—Verification ; PW.1.1:Securing Materials—Verification, Automation; PW.4.1:Securing Materials—Verification; PW.4.4:Securing Materials—Verification, Automation; PW.6.1:Securing Build Pipelines—Verification, Automation; PW.6.2:Securing Build Pipelines—Verification, Automation; RV.1.1:Securing Materials—Verification
USA Presidential Executive Order EO14028 [i.13]	PO.1.1:4e(ix); PO.1.2:4e(ix); PO.1.3:4e(vi), 4e(ix); PO.2.1:4e(ix); PO.2.2:4e(ix); PO.2.3:4e(ix); PO.3.1:4e(iii), 4e(ix); PO.3.2:4e(i)(F), 4e(ii), 4e(iii), 4e(v), 4e(vi), 4e(ix); PO.3.3:4e(i)(F), 4e(ii), 4e(v), 4e(ix); PO.4.1:4e(iv), 4e(v), 4e(ix); PO.4.2:4e(iv), 4e(v), 4e(ix); PO.5.1:4e(i)(A), 4e(i)(B), 4e(i)(C), 4e(i)(D), 4e(i)(F), 4e(ii), 4e(iii), 4e(v), 4e(vi), 4e(ix); PO.5.2:4e(i)(C), 4e(i)(E), 4e(i)(F), 4e(ii), 4e(iii), 4e(v), 4e(vi), 4e(ix); PS.1.1:4e(iii), 4e(iv), 4e(ix); PS.2.1:4e(iii), 4e(ix), 4e(x); PS.3.1:4e(iii), 4e(vi), 4e(ix), 4e(x); PW.1.1:4e(vi), 4e(vii), 4e(ix), 4e(x); PW.1.1:4e(ix); PW.1.2:4e(v), 4e(ix); PW.1.3:4e(ix); PW.2.1:4e(iv), 4e(v), 4e(ix); PW.4.1:4e(iii), 4e(vi), 4e(ix), 4e(x); PW.4.2:4e(ix); PW.4.4:4e(iii), 4e(iv), 4e(vi), 4e(ix), 4e(x); PW.5.1:4e(iv), 4e(ix); PW.6.1:4e(iv), 4e(ix); PW.6.2:4e(iv), 4e(ix); PW.7.1:4e(iv), 4e(ix); PW.7.2:4e(iv), 4e(v), 4e(ix); PW.8.1:4e(ix); PW.8.2:4e(iv), 4e(v), 4e(ix); PW.9.1:4e(iv), 4e(ix); PW.9.2:4e(iv), 4e(ix); RV.1.1:4e(iv), 4e(vi), 4e(viii), 4e(ix); RV.1.2:4e(iv), 4e(vi), 4e(viii), 4e(ix); RV.1.3:4e(viii), 4e(ix); RV.2.1:4e(iv), 4e(viii), 4e(ix); RV.2.2:4e(iv), 4e(vi), 4e(viii), 4e(ix); RV.3.1:4e(ix); RV.3.2:4e(ix); RV.3.3:4e(iv), 4e(viii), 4e(ix); RV.3.4:4e(ix)
Institute for Defence Analyses State-of-the Art (IDASOAR) [i.79]	PO.1.3:19,21; PS.1.1:FactSheet25; PS.3.1:25; PW.1.1:1; PW.4.1:19; PW.4.2:19; PW.4.4:21; PW.5.1:2; PW.7.2:3,4,5,14,15,48; PW.8.2:7,8,10,11,38,39,43,44,48,55,56,57; PW.9.1:23; PW.9.2:23

Framework/Spec	SSDF Task [task id:framework-spec provision, paragraph]
International Electrotechnical Commission IEC 62443-4-1 [i.80]	PO.1.1:SM-7, SM-9; PO.1.2:SR-3, SR-4, SR-5, SD-4; PO.1.3:SM-9, SM-10; PO.2.1:SM-2, SM-13; PO.2.2:SM-4; PO.3.2:SM-7; PO.3.3:SM-12, SI-2; PO.4.1:SI-1, SI-2, SVV-3; PO.4.2:SI-1, SVV-1, SVV-2, SVV-3, SVV-4; PO.5.1:SM-7; PO.5.2:SM-7; PS.1.1:SM-6, SM-7, SM-8; PS.2.1:SM-6, SM-8, SUM-4; PS.3.1:SM-6, SM-7; PW.1.1:SM-4, SR-1, SR-2, SD-1; PW.1.2:SD-1; PW.1.3:SD-1, SD-4; PW.2.1:SM-2, SR-2, SR-5, SD-3, SD-4, SI-2; PW.4.1:SM-9, SM-10; PW.4.4:SI-1, SM-9, SM-10, DM-1; PW.5.1:SI-1, SI-2; PW.6.1:SI-2; PW.6.2:SI-2; PW.7.1:SM-5, SI-1, SVV-1; PW.7.2:SI-1, SVV-1, SVV-2; PW.8.1:SVV-1, SVV-2, SVV-3, SVV-4, SVV-5; PW.8.2:SM-5, SM-13, SI-1, SVV-1, SVV-2, SVV-3, SVV-4, SVV-5; PW.9.1:SD-4, SVV-1, SG-1; PW.9.2:SG-3; RV.1.1:DM-1, DM-2, DM-3; RV.1.2:SI-1, SVV-2, SVV-3, SVV-4, DM-1, DM-2; RV.1.3:DM-1, DM-2, DM-3, DM-4, DM-5; RV.2.1:DM-2, DM-3; RV.2.2:DM-4; RV.3.1:DM-3; RV.3.2:DM-4; RV.3.3:SI-1, DM-3, DM-4; RV.3.4:DM-6
National Institute of Standards and Technology Internal Report IR8397 [i.81]	PO.3.2:2.2; PW.1.1:2.1; PW.4.4:2.11; PW.6.2:2.5; PW.7.2:2.3, 2.4; PW.8.2:2.6, 2.7, 2.8, 2.9, 2.10, 2.11
International Organization for Standardization ISO27034-1 [i.82]	PO.1.2:7.3.2; PO.4.1:7.3.5; PW.1.1:7.3.3; PW.1.2:7.3.3; PW.2.1:7.3.3; PW.5.1:7.3.5; PW.7.2:7.3.6; PW.8.2:7.3.6; PW.9.1:7.3.5; RV.1.2:7.3.6
ISO29147 [i.31]	RV.1.1:6.2.1, 6.2.2, 6.2.4, 6.3, 6.5; RV.1.2:6.4; RV.1.3:All
ISO30111 [i.32]	RV.1.1:7.1.3; RV.1.2:7.1.4; RV.1.3:All; RV.2.1:7.1.4; RV.2.2:7.1.4, 7.1.5; RV.3.1:7.1.4; RV.3.2:7.1.7; RV.3.3:7.1.4; RV.3.4:7.1.7
Microsoft Security Development Lifecycle MSSDL [i.83]	PO.1.2:2, 5; PO.1.3:7; PO.2.2:1; PO.3.1:8; PO.3.3:8; PO.4.1:3; PW.1.1:4; PW.1.2:4; PW.1.3:5; PW.4.1:6; PW.4.4:7; PW.5.1:9; PW.6.1:8; PW.6.2:8; PW.7.2:9, 10; PW.8.2:10, 11; RV.1.3:12; RV.3.4:2
National Institute of Standards and Technology Cybersecurity Framework (NISTCSF) [i.84]	PO.1.1:ID.GV-3; PO.1.2:ID.GV-3; PO.1.3:ID.SC-3; PO.2.1:ID.AM-6, ID.GV-2; PO.2.2:PR.AT; PO.2.3:ID.RM-1, ID.SC-1; PO.5.1:PR.AC-5, PR.DS-7; PO.5.2:PR.AC-4, PR.AC-7, PR.IP-1, PR.IP-3, PR.IP-12, PR.PT-1, PR.PT-3, DE.CM; PS.1.1:PR.AC-4, PR.DS-6, PR.IP-3; PS.2.1:PR.DS-6; PS.3.1:PR.IP-4; PW.1.1:ID.RA; PW.4.1:ID.SC-2; PW.4.4:ID.SC-4, PR.DS-6
Technology Cybersecurity Framework NISTLABEL [i.85]	PS.2.1:2.2.2.4; PW.1.2:2.2.2.2; PW.4.4:2.2.2.2; PW.7.1:2.2.2.2; PW.7.2:2.2.2.2; PW.8.1:2.2.2.2; PW.8.2:2.2.2.2; RV.1.3:2.2.2.3; RV.2.1:2.2.2.2; RV.2.2:2.2.2.2
National Telecommunications and Information Administration NTIA SBOM [i.86]	PW.1.1:All
Open Worldwide Application Security Project Application Security Verification Standard (OWASPASVS) [i.87]	PO.1.1:1.1.1; PO.3.2:1.14.3, 1.14.4, 14.1, 14.2; PS.1.1:1.10, 10.3.2; PW.1.1:1.1.2, 1.2, 1.4, 1.6, 1.8, 1.9, 1.11, 2, 3, 4, 6, 8, 9, 11, 12, 13; PW.1.2:1.1.3, 1.1.4; PW.1.3:1.1.6; PW.2.1:1.1.5; PW.4.1:1.1.6; PW.4.2:1.1.6; PW.4.4:10, 14.2; PW.5.1:1.1.7, 1.5, 1.7, 5, 7; PW.6.2:14.1, 14.2.1; PW.7.2:1.1.7, 10
OWASP Mobile Application Security Verification Standard (OWASPMASVS) [i.88]	PO.1.1:1.1; PO.1.2:1.12; PO.3.2:7.9; PS.1.1:7.1; PW.1.1:1.6, 1.8, 2, 3, 4, 5, 6; PW.1.2:1.3, 1.6; PW.4.4:7.5; PW.5.1:7.6; PW.6.2:7.2; PW.7.2:7.5; PW.8.2:7.5; RV.1.3:1.11
OWASP Software Assurance Maturity Model) OWASPSAMM [i.89]	PO.1.1:PC1-A, PC1-B, PC2-A; PO.1.2:PC1-A, PC1-B, PC2-A, PC3-A, SR1-A, SR1-B, SR2-B, SA1-B, IR1-A; PO.1.3:SR3-A; PO.2.2:EG1-A, EG2-A; PO.2.3:SM1.A; PO.3.1:IR2-B, ST2-B; PO.3.3:PC3-B; PO.4.1:PC3-A, DR3-B, IR3-B, ST3-B; PO.4.2:PC3-B; PS.1.1:OE3-B; PS.2.1:OE3-B; PW.1.1:TA1-A, TA1-B, TA3-B, DR1-A; PW.1.2:DR1-B; PW.1.3:SA2-A; PW.2.1:DR1-A, DR1-B; PW.4.1:SA1-A; PW.4.4:TA3-A, SR3-B; PW.7.2:IR1-B, IR2-A, IR2-B, IR3-A; PW.8.2:ST1-A, ST1-B, ST2-A, ST2-B, ST3-A; PW.9.2:OE1-A; RV.1.1:IM1-A, IM2-B, EH1-B; RV.1.3:IM1-A, IM1-B, IM2-A, IM2-B; RV.3.1:IM3-A; RV.3.2:IM3-B
OWASP Software Component Verification Standard (OWASPSCVS) [i.90]	PO.3.2:3, 5; PO.3.3:3.13, 3.14; PS.2.1:4; PS.3.1:1, 3.18, 3.19, 6.3; PW.1.1:1.4, 2; PW.4.1:4; PW.4.4:4, 5, 6; RV.1.1:4
Payment Card Industry Secure Software Lifecycle (PCISSLC) [i.91]	PO.1.1:2.1, 2.2; PO.1.2:2.1, 2.2, 2.3, 3.3; PO.2.1:1.2; PO.2.2:1.3; PO.2.3:1.1; PO.3.3:2.5; PO.4.1:3.3; PO.4.2:2.5; PS.1.1:5.1, 6.1; PS.2.1:6.1, 6.2; PS.3.1:5.2, 6.1, 6.2; PW.1.1:3.2, 3.3; PW.1.2:3.2, 3.3; PW.2.1:3.2; PW.4.4:3.2, 3.4, 4.1; PW.6.2:3.2; PW.7.2:3.2, 4.1; PW.8.2:4.1; PW.9.2:8.1, 8.2; RV.1.1:3.4, 4.1, 9.1; RV.1.2:3.4, 4.1; RV.1.3:9.2, 9.3; RV.2.1:3.4, 4.2; RV.2.2:4.1, 4.2, 10.1; RV.3.1:4.2; RV.3.2:2.6, 4.2; RV.3.3:4.2; RV.3.4:2.6, 4.2

Framework/Spec	SSDF Task [task id:framework-spec provision, paragraph]
SAFECode SCAGILE [i.48]	PO.1.3:Tasks Requiring the Help of Security Experts 8; PO.2.2:Operational Security Tasks 14, 15; Tasks Requiring the Help of Security Experts 1; PO.3.1:Tasks Requiring the Help of Security Experts 9; PO.3.2:Tasks Requiring the Help of Security Experts 9; PO.3.3:Tasks Requiring the Help of Security Experts 9; PO.5.1:Tasks Requiring the Help of Security Experts 11; PO.5.2:Tasks Requiring the Help of Security Experts 11; PW.1.1:Tasks Requiring the Help of Security Experts 3; PW.4.4:Tasks Requiring the Help of Security Experts 8; PW.6.1:Operational Security Task 3; PW.6.2:Operational Security Task 8; PW.7.2:Operational Security Tasks 4, 7; Tasks Requiring the Help of Security Experts 10; PW.8.2:Operational Security Tasks 10, 11; Tasks Requiring the Help of Security Experts 4, 5, 6, 7; PW.9.1:Tasks Requiring the Help of Security Experts 12; PW.9.2:Tasks Requiring the Help of Security Experts 12; RV.1.1:Operational Security Task 5; RV.1.2:Operational Security Tasks 10, 11; RV.2.1:Operational Security Task 1, Tasks Requiring the Help of Security Experts 10; RV.2.2:Operational Security Task 2
SAFECode SCFPSSD [i.2]	PO.1.1:Planning the Implementation and Deployment of Secure Development Practices; PO.1.2:Establish Coding Standards and Conventions; PO.1.3:Manage Security Risk Inherent in the Use of Third-Party Components; PO.2.2:Planning the Implementation and Deployment of Secure Development Practices; PO.3.2:Use Current Compiler and Toolchain Versions and Secure Compiler Options; PW.1.1:Threat Modeling; PW.1.3:Standardize Identity and Access Management; Establish Log Requirements and Audit Practices; PW.4.4:Manage Security Risk Inherent in the Use of Third-Party Components; PW.5.1:Establish Log Requirements and Audit Practices, Use Code Analysis Tools to Find Security Issues Early, Handle Data Safely, Handle Errors, Use Safe Functions Only; PW.6.1:Use Current Compiler and Toolchain Versions and Secure Compiler Options; PW.6.2:Use Current Compiler and Toolchain Versions and Secure Compiler Options; PW.7.2:Use Code Analysis Tools to Find Security Issues Early, Use Static Analysis Security Testing Tools, Perform Manual Verification of Security Features/Mitigations; PW.8.2:Perform Dynamic Analysis Security Testing, Fuzz Parsers, Network Vulnerability Scanning, Perform Automated Functional Testing of Security Features/Mitigations, Perform Penetration Testing; PW.9.2:Verify Secure Configurations and Use of Platform Mitigation; RV.1.1:Vulnerability Response and Disclosure; RV.1.3:Vulnerability Response and Disclosure; RV.2.2:Fix the Vulnerability, Identify Mitigating Factors or Workarounds; RV.3.1:Secure Development Lifecycle Feedback; RV.3.2:Secure Development Lifecycle Feedback; RV.3.4:Secure Development Lifecycle Feedback
SAFECode SCSIC [i.92]	PO.1.3:Vendor Sourcing Integrity Controls; PO.2.1:Vendor Software Development Integrity Controls; PO.2.2:Vendor Software Development Integrity Controls; PO.3.1:Vendor Software Delivery Integrity Controls; PO.3.2:Vendor Software Delivery Integrity Controls; PO.3.3:Vendor Software Delivery Integrity Controls; PO.4.2:Vendor Software Delivery Integrity Controls; PO.5.1:Vendor Software Delivery Integrity Controls; PO.5.2:Vendor Software Delivery Integrity Controls; PS.1.1:Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls; PS.2.1:Vendor Software Delivery Integrity Controls; PS.3.1:Vendor Software Delivery Integrity Controls; PW.1.1:Vendor Software Delivery Integrity Controls; PW.4.1:Vendor Sourcing Integrity Controls; PW.4.4:Vendor Sourcing Integrity Controls, Peer Reviews and Security Testing; PW.6.1:Vendor Software Development Integrity Controls; PW.6.2:Vendor Software Development Integrity Controls; PW.7.1:Peer Reviews and Security Testing; PW.7.2:Peer Reviews and Security Testing; PW.8.1:Peer Reviews and Security Testing; PW.8.2:Peer Reviews and Security Testing; PW.9.1:Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls; PW.9.2:Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls
SAFECode SCTPC [i.49]	PW.1.1:MAINTAIN3; PW.4.1:MAINTAIN; PW.4.2:MAINTAIN; PW.4.4:MAINTAIN, ASSESS; RV.1.1:MONITOR1; RV.2.2:MITIGATE
SAFECode SCTTM [i.51]	PW.1.1:Entire guide
NIST Special Publication SP 800-53 [i.93]	PO.1.1:SA-1, SA-8, SA-15, SR-3; PO.1.2:SA-8, SA-8(3), SA-15, SR-3; PO.1.3:SA-4, SA-9, SA-10, SA-10(1), SA-15, SR-3, SR-4, SR-5; PO.2.1:SA-3; PO.2.2:SA-8; PO.3.1:SA-15; PO.3.2:SA-15; PO.3.3:SA-15; PO.4.1:SA-15, SA-15(1); PO.4.2:SA-15, SA-15(1), SA-15(11); PO.5.1:SA-3(1), SA-8, SA-15; PO.5.2:SA-15; PS.1.1:SA-10; PS.2.1:SA-8; PS.3.1:SA-10, SA-15, SA-15(11), SR-4; PW.1.1:SA-8, SR-3, SR-4; PW.1.1:SA-8, SA-11(2), SA-11(6), SA-15(5); PW.1.2:SA-8, SA-10, SA-17; PW.4.1:SA-4, SA-5, SA-8(3), SA-10(6), SR-3, SR-4; PW.4.2:SA-8(3); PW.4.4:SA-9, SR-3, SR-4, SR-4(3), SR-4(4); PW.6.1:SA-15; PW.6.2:SA-15, SR-9; PW.7.1:SA-11; PW.7.2:SA-11, SA-11(1), SA-11(4), SA-15(7); PW.8.1:SA-11; PW.8.2:SA-11, SA-11(5), SA-11(8), SA-15(7); PW.9.2:SA-5, SA-8(23); RV.1.1:SA-10, SR-3, SR-4; RV.1.2:SA-11; RV.1.3:SA-15(10); RV.2.1:SA-10, SA-15(7); RV.2.2:SA-5, SA-10, SA-11, SA-15(7); RV.3.3:SA-11; RV.3.4:SA-15
NIST Special Publication SP 800-160 [i.94]	PO.1.1:3.1.2, 3.2.1, 3.2.2, 3.3.1, 3.4.2, 3.4.3; PO.1.2:3.1.2, 3.2.1, 3.3.1; PO.1.3:3.1.1, 3.1.2; PO.2.1:3.2.1, 3.2.4, 3.3.1; PO.2.2:3.2.4, 3.2.6; PO.4.1:3.2.1, 3.2.5, 3.3.1; PO.4.2:3.2.5, 3.3.7; PW.1.1:3.3.4, 3.4.5; PW.4.4:3.1.2, 3.3.8; RV.1.3:3.3.8; RV.2.1:3.3.8; RV.2.2:3.3.8; RV.3.2:3.3.8

Framework/Spec	SSDF Task [task id:framework-spec provision, paragraph]
NIST Special Publication SP 800-161 [i.95]	PO.1.1:SA-1, SA-8, SA-15, SR-3; PO.1.2:SA-8, SA-15, SR-3; PO.1.3:SA-4, SA-9, SA-9(1), SA-9(3), SA-10, SA-10(1), SA-15, SR-3, SR-4, SR-5; PO.2.1:SA-3; PO.2.2:SA-8; PO.3.1:SA-15; PO.3.2:SA-15; PO.3.3:SA-15; PO.4.1:SA-15, SA-15(1); PO.4.2:SA-15, SA-15(1), SA-15(11); PO.5.1:SA-3, SA-8, SA-15; PO.5.2:SA-15; PS.1.1:SA-8, SA-10; PS.2.1:SA-8; PS.3.1:SA-8, SA-10, SA-15(11), SR-4; PW.1.1:SA-8, SR-3, SR-4; PW.1.1:SA-8, SA-11(2), SA-11(6), SA-15(5); PW.1.2:SA-8, SA-17; PW.4.1:SA-4, SA-5, SA-8(3), SA-10(6), SR-3, SR-4; PW.4.2:SA-8(3); PW.4.4:SA-4, SA-8, SA-9, SA-9(3), SR-3, SR-4, SR-4(3), SR-4(4); PW.6.1:SA-15; PW.6.2:SA-15, SR-9; PW.7.1:SA-11; PW.7.2:SA-11, SA-11(1), SA-11(4), SA-15(7); PW.8.1:SA-11; PW.8.2:SA-11, SA-11(5), SA-11(8), SA-15(7); PW.9.2:SA-5, SA-8(23); RV.1.1:SA-10, SR-3, SR-4; RV.1.2:SA-11; RV.1.3:SA-15(10); RV.2.1:SA-15(7); RV.2.2:SA-5, SA-8, SA-10, SA-11, SA-15(7); RV.3.3:SA-11; RV.3.4:SA-15
NIST Special Publication SP 800-181 [i.96]	PO.3.1:K0013, K0178; PO.3.2:K0013, K0178; PO.3.3:K0013; T0024; PO.4.1:K0153, K0165; PO.4.2:T0349; K0153; PO.5.1:OM-NET-001, SP-SYS-001; T0019, T0023, T0144, T0160, T0262, T0438, T0484, T0485, T0553; K0001, K0005, K0007, K0033, K0049, K0056, K0061, K0071, K0104, K0112, K0179, K0326, K0487; S0007, S0084, S0121; A0048; PO.5.2:OM-ADM-001, SP-SYS-001; T0484, T0485, T0489, T0553; K0005, K0007, K0077, K0088, K0130, K0167, K0205, K0275; S0076, S0097, S0121, S0158; A0155; PS.2.1:K0178; PW.1.1:T0038, T0062; K0005, K0009, K0038, K0039, K0070, K0080, K0119, K0147, K0149, K0151, K0152, K0160, K0161, K0162, K0165, K0297, K0310, K0344, K0362, K0487, K0624; S0006, S0009, S0022, S0078, S0171, S0229, S0248; A0092, A0093, A0107; PW.1.2:T0256; K0005, K0038, K0039, K0147, K0149, K0160, K0161, K0162, K0165, K0344, K0362, K0487; S0006, S0009, S0078, S0171, S0229, S0248; A0092, A0107; PW.2.1:T0328; K0038, K0039, K0070, K0080, K0119, K0152, K0153, K0161, K0165, K0172, K0297; S0006, S0009, S0022, S0036, S0141, S0171; PW.4.1:K0039; PW.4.2:SP-DEV-001; PW.4.4:SP-DEV-002; K0153, K0266; S0298; PW.5.1:SP-DEV-001; T0013, T0077, T0176; K0009, K0016, K0039, K0070, K0140, K0624; S0019, S0060, S0149, S0172, S0266; A0036, A0047; PW.6.2:K0039, K0070; PW.7.1:SP-DEV-002; K0013, K0039, K0070, K0153, K0165; S0174; PW.7.2:SP-DEV-001, SP-DEV-002; T0013, T0111, T0176, T0267, T0516; K0009, K0039, K0070, K0140, K0624; S0019, S0060, S0078, S0137, S0149, S0167, S0174, S0242, S0266; A0007, A0015, A0036, A0044, A0047; PW.8.1:SP-DEV-001, SP-DEV-002; T0456; K0013, K0039, K0070, K0153, K0165, K0342, K0367, K0536, K0624; S0001, S0015, S0026, S0061, S0083, S0112, S0135; PW.8.2:SP-DEV-001, SP-DEV-002; T0013, T0028, T0169, T0176, T0253, T0266, T0456, T0516; K0009, K0039, K0070, K0272, K0339, K0342, K0362, K0536, K0624; S0001, S0015, S0046, S0051, S0078, S0081, S0083, S0135, S0137, S0167, S0242; A0015; PW.9.1:SP-DEV-002; K0009, K0039, K0073, K0153, K0165, K0275, K0531; S0167; PW.9.2:SP-DEV-001; K0009, K0039, K0073, K0153, K0165, K0275, K0531; RV.1.1:K0009, K0038, K0040, K0070, K0161, K0362; S0078; RV.1.2:SP-DEV-002; K0009, K0039, K0153; RV.1.3:K0041, K0042, K0151, K0292, K0317; S0054; A0025; RV.2.1:K0009, K0039, K0070, K0161, K0165; S0078; RV.2.2:T0163, T0229, T0264; K0009, K0070; RV.3.1:T0047, K0009, K0039, K0070, K0343; RV.3.2:T0111, K0009, K0039, K0070, K0343; RV.3.3:SP-DEV-001, SP-DEV-002; K0009, K0039, K0070; RV.3.4:K0009, K0039, K0070
NIST Special Publication SP 800-216 [i.40]	RV.1.3:All

Annex C (informative): Secure by Design Checklist

This checklist is provides a list of SSDIF artifacts for the SSDF Tasks sorted by SSDF Practices.

SSDF Practices	SSDF Tasks	SSDIF Artifact
Define Security Requirements for Software Development (PO.1): Ensure that security requirements for software development are known at all times so that they can be taken into account throughout the SDLC and duplication of effort can be minimized because the requirements information can be collected once and shared. This includes requirements from internal sources (e.g. the organization's policies, business objectives, and risk management strategy) and external sources (e.g. applicable laws and regulations).	**PO.1.1**: Identify and document all security requirements for the organization's software development infrastructures and processes and maintain the requirements over time.	DG 1: Documents to show implementation of Critical Security Controls to include: • Policies/Process for: enterprise asset inventory, software asset inventory, data management, secure configuration of enterprise assets and software, account management, access control management, vulnerability management, audit log management, data recovery, security awareness and skills training, incident response management. • Configurations to show implementation of policies/processes and safeguards. DG 2/3: Documents to show implementation of IG2/3 safeguards to include: • Policies Process for: Network Infrastructure Management, Service Provider Management, Application Software Security, Penetration Testing. • Configurations to show implementation of policies/processes and safeguards.
	PO.1.2: Identify and document all security requirements for organization-developed software to meet and maintain the requirements over time.	DG 1/2/3: • Documented secure development process that is updated yearly or when significant changes occur. • Work item tracking system showing the organization is following the documented process and updates it at least annually.
	PO.1.3: Communicate requirements to all third parties who will provide commercial software components to the organization for reuse by the organization's own software. [Formerly PW.3.1]	DG 1/2/3: • Contracts, RFPs etc. with security requirements.
Implement Roles and Responsibilities (PO.2): Ensure that everyone inside and outside of the organization involved in the SDLC is prepared to perform their SDLC-related roles and responsibilities throughout the SDLC.	**PO.2.1**: Create new roles and alter responsibilities for existing roles as needed to encompass all parts of the SDLC. Periodically review and maintain the defined roles and responsibilities, updating them as needed.	DG 1/2/3: • Documented organization charts and position descriptions that align with responsibilities identified in policies. • Organizational communications of updated roles and responsibilities.
	PO.2.2: Provide role-based training for all personnel with responsibilities that contribute to secure development. Periodically review personnel proficiency and role-based training, and update the training as needed.	DG 1: • Documented list of mandatory online developer training. DG 2/3: • Examples of tools and tool outputs that provide just in time training.

SSDF Practices	SSDF Tasks	SSDIF Artifact
		• Documented list of defensive programming training.
	PO.2.3: Obtain upper management or authorizing official commitment to secure development and convey that commitment to all with development-related roles and responsibilities.	DG 1: • Emails showing executive and/or management commitment to a secure development program. DG 2/3: • Examples of artefacts an organization can provide include: copies of all hands emails, videos of executive talks at virtual or live meetings, motivational videos or internal event recordings, release checklist endorsed by upper management. • Names of security champions and notes/bugs from team meetings.
Implement Supporting Toolchains (PO.3): Use automation to reduce human effort and improve the accuracy, reproducibility, usability, and comprehensiveness of security practices throughout the SDLC, as well as provide a way to document and demonstrate the use of these practices. Toolchains and tools may be used at different levels of the organization, such as organization-wide or project-specific, and may address a particular part of the SDLC, like a build pipeline.	**PO.3.1**: Specify which tools or tool types "must" [i.1] or should be included in each toolchain to mitigate identified risks, as well as how the toolchain components are to be integrated with each other.	DG 1/2/3: • List of tools and/or tool types used in secure software development.
	PO.3.2: Follow recommended security practices to deploy, operate, and maintain tools and toolchains.	DG 1/2/3: • Configurations of tools, work items in the secure development process workflow systems
	PO.3.3: Configure tools to generate artifacts of their support of secure software development practices as defined by the organization.	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Define and Use Criteria for Software Security Checks (PO.4): Help ensure that the software resulting from the SDLC meets the organization's expectations by defining and using criteria for checking the software's security during development.	**PO.4.1**: Define criteria for software security checks and track throughout the SDLC.	DG 1/2/3: • Configuration of bug tracking system
	PO.4.2: Implement processes, mechanisms, etc. to gather and safeguard the necessary information in support of the criteria.	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Implement and Maintain Secure Environments for Software Development (PO.5): Ensure that all components of the environments for software development are strongly protected from internal and external threats to prevent compromises of the environments or the software being developed or maintained within them. Examples of environments for software	**PO.5.1**: Separate and protect each environment involved in software development.	DG 1: Documents to show implementation of Critical Security Controls to include: • Policies/Process for: enterprise asset inventory, software asset inventory, data management, secure configuration of enterprise assets and software, account management, access control management, vulnerability management, audit log management, data recovery, security awareness and skills training, incident response management. • Configurations to show implementation of policies/processes and safeguards.

SSDF Practices	SSDF Tasks	SSDIF Artifact
development include development, build, test, and distribution environments.		DG 2/3: Documents to show implementation of IG2/3 safeguards to include: • Policies Process for: Network Infrastructure Management, Service Provider Management, Application Software Security, Penetration Testing. • Configurations to show implementation of policies/processes and safeguards.
	PO.5.2: Secure and harden development endpoints (i.e. endpoints for software designers, developers, testers, builders, etc.) to perform development-related tasks using a risk-based approach.	DG 1/2/3: • Documentation describing the organization's secure configuration process. • Configurations to show implementation of the secure configuration process and Safeguards 4.1 thru 4.12.
Protect All Forms of Code from Unauthorized Access and Tampering (PS.1): Help prevent unauthorized changes to code, both inadvertent and intentional, which could circumvent or negate the intended security characteristics of the software. For code that is not intended to be publicly accessible, this helps prevent theft of the software and may make it more difficult or time-consuming for attackers to find vulnerabilities in the software.	**PS.1.1**: Store all forms of code - including source code, executable code, and configuration-as-code - based on the principle of least privilege so that only authorized personnel, tools, services, etc. have access.	DG 1/2/3: • Documentation describing access control management process. • Configurations to show implementation of the access management process and Safeguards 6.1 thru 6.8.
Provide a Mechanism for Verifying Software Release Integrity (PS.2): Help software acquirers ensure that the software they acquire is legitimate and has not been tampered with.	**PS.2.1**: Make software integrity verification information available to software acquirers.	DG 1/2/3: Signed code.
Archive and Protect Each Software Release (PS.3): Preserve software releases in order to help identify, analyse, and eliminate vulnerabilities discovered in the software after release.	**PS.3.1**: Securely archive the necessary files and supporting data (e.g. integrity verification information, provenance data) to be retained for each software release.	DG 1/2/3: • Documented process. • Workflow records created as part of the secure development process. For example, the configuration management repository contains source code and documentation and the workflow/bugtraq system records review and approved changes to source code. • Files and data present in archive repository
	PS.3.2: Collect, safeguard, maintain, and share provenance data for all components of each software release (e.g. in a software bill of materials [SBOM]).	DG 1: • List of approved third-party components. • SBOM for each product DG2/3: • List of SCA tools and tool outputs.

SSDF Practices	SSDF Tasks	SSDIF Artifact
Design Software to Meet Security Requirements and Mitigate Security Risks (PW.1): Identify and evaluate the security requirements for the software; determine what security risks the software is likely to face during operation and how the software's design and architecture should mitigate those risks; and justify any cases where risk-based analysis indicates that security requirements should be relaxed or waived. Addressing security requirements and risks during software design (secure by design) is key for improving software security and also helps improve development efficiency.	**PW.1.1**: Use forms of risk modelling - such as threat modelling, attack modelling, or attack surface mapping - to help assess the security risk for the software.	DG 3: • Documented threat model and tools used to support threat modelling. • DFD describing system and potential attacks. • Content of workflow system showing mitigations of medium and high threats.
	PW.1.2: Track and maintain the software's security requirements, risks, and design decisions.	DG 1/2/3: • The organization provides the product specification that is maintained in a repository. • The organization tracks changes to the specification in their workflow system
	PW.1.3: Where appropriate, build in support for using standardized security features and services (e.g. enabling software to integrate with existing log management, identity management, access control, and vulnerability management systems) instead of creating proprietary implementations of security features and services. [Formerly PW.4.3]	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Review the Software Design to Verify Compliance with Security Requirements and Risk Information (PW.2): Help ensure that the software will meet the security requirements and satisfactorily address the identified risk information.	**PW.2.1**: Have 1) a qualified person (or people) who were not involved with the design and/or 2) automated processes instantiated in the toolchain review the software design to confirm and enforce that it meets all of the security requirements and satisfactorily addresses the identified risk information.	DG 1/2/3: Design specifications or other design documents; design security review minutes; bugs for design issues in the workflow system
Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality (PW.4): Lower the costs of software development, expedite software development, and decrease the likelihood of introducing additional security vulnerabilities into the software by reusing software modules and services that have already had their security posture checked. This is particularly important for software that implements security functionality, such as cryptographic modules and protocols.	**PW.4.1**: Acquire and maintain well-secured software components (e.g. software libraries, modules, middleware, frameworks) from commercial, open-source, and other third-party developers for use by the organization's software.	DG 1/2/3: • Documented criteria for accepting third party components; notes, checklists, or minutes of reviews for component acceptability.
	PW.4.2: Create and maintain well-secured software components in-house following SDLC processes to meet common internal software development needs that cannot be better met by third-party software components.	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. • Documentation describing access control management process. • Configurations to show implementation of the access management process and Safeguards 6.1 thru 6.8. • Documentation of implementation of secure development practices through guidance documentation and provided training.

SSDF Practices	SSDF Tasks	SSDIF Artifact
		DG 2/3: • Penetration test results and results of attack surface testing focused on the component.
	PW.4.4: Verify that acquired commercial, open-source, and all other third-party software components comply with the requirements, as defined by the organization, throughout their life cycles.	DG 1: • Documented process for TPC and associated risks. • Tool outputs to support TPC risk and maintenance status. DG 2/3: • Tool outputs to validate TPC security.
Create Source Code by Adhering to Secure Coding Practices (PW.5): Decrease the number of security vulnerabilities in the software, and reduce costs by minimizing vulnerabilities introduced during source code creation that meet or exceed organization-defined vulnerability severity criteria.	**PW.5.1**: Follow all secure coding practices that are appropriate to the development languages and environment to meet the organization's requirements.	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. DG 3: • Documentation of training on coding standards. • Code-level penetration reports and remediation plans.
Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security (PW.6): Decrease the number of security vulnerabilities in the software and reduce costs by eliminating vulnerabilities before testing occurs.	**PW.6.1**: Use compiler, interpreter, and build tools that offer features to improve executable security.	DG 1/2/3: Configuration of compiler, interpreter and build tools, work items in the secure development process workflow system, and bug history of security bugs in the product workflow system.
	PW.6.2: Determine which compiler, interpreter, and build tool features should be used and how each should be configured, then implement and use the approved configurations.	DG 1/2/3: Configuration of compiler, interpreter and build tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Review and/or Analyse Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7): Help identify vulnerabilities so that they can be corrected before the software is released to prevent exploitation. Using automated methods lowers the effort and resources needed to detect vulnerabilities. Human-readable code includes source code, scripts, and any other form of code that an organization deems human-readable.	**PW.7.1**: Determine whether code review (a person looks directly at the code to find issues) and/or code analysis (tools are used to find issues in code, either in a fully automated way or in conjunction with a person) should be used, as defined by the organization.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	PW.7.2: Perform the code review and/or code analysis based on the organization's secure coding standards, and record and triage all discovered issues and recommended remediations in the development team's workflow or issue tracking system.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.

SSDF Practices	SSDF Tasks	SSDIF Artifact
Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8): Help identify vulnerabilities so that they can be corrected before the software is released in order to prevent exploitation. Using automated methods lowers the effort and resources needed to detect vulnerabilities and improves traceability and repeatability. Executable code includes binaries, directly executed bytecode and source code, and any other form of code that an organization deems executable.	**PW.8.1**: Determine whether executable code testing should be performed to find vulnerabilities not identified by previous reviews, analysis, or testing and, if so, which types of testing should be used.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	PW.8.2: Scope the testing, design the tests, perform the testing, and document the results, including recording and triaging all discovered issues and recommended remediations in the development team's workflow or issue tracking system.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
Configure Software to Have Secure Settings by Default (PW.9): Help improve the security of the software at the time of installation to reduce the likelihood of the software being deployed with weak security settings, putting it at greater risk of compromise.	**PW.9.1**: Define a secure baseline by determining how to configure each setting that has an effect on security or a security-related setting so that the default settings are secure and do not weaken the security functions provided by the platform, network infrastructure, or services.	DG 1/2/3: • Documented secure configuration for software. • Output of configuration scanning tools and records in workflow system
	PW.9.2: Implement the default settings (or groups of default settings, if applicable), and document each setting for software administrators.	DG 1/2/3: Documentation for administrators of secure default configuration.
Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1): Help ensure that vulnerabilities are identified more quickly so that they can be remediated more quickly in accordance with risk, reducing the window of opportunity for attackers.	**RV.1.1**: Gather information from software acquirers, users, and public sources on potential vulnerabilities in the software and third-party components that the software uses and investigate all credible reports.	DG 1/2/3: • Policy for vulnerability response. • Available and visible location for external reporting of vulnerabilities. • Available service to securely accept sensitive vulnerability information.
	RV.1.2: Review, analyse, and/or test the software's code to identify or confirm the presence of previously undetected vulnerabilities.	DG 1/2/3: Configuration of tools, vulnerability and remediation work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	RV.1.3: Have a policy that addresses vulnerability disclosure and remediation, and implement the roles, responsibilities, and processes needed to support that policy.	DG 1/2/3: Documented vulnerability handling policy.

SSDF Practices	SSDF Tasks	SSDIF Artifact
Assess, Prioritize, and Remediate Vulnerabilities (RV.2): Help ensure that vulnerabilities are remediated in accordance with risk to reduce the window of opportunity for attackers.	**RV.2.1**: Analyse each vulnerability to gather sufficient information about risk to plan its remediation or other risk response.	DG 1/2/3: • Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system. • Records of bug analysis in the product workflow system.
	RV.2.2: Plan and implement risk responses for vulnerabilities.	DG 1/2/3: • Entry in the workflow system and the fixed. code. • Entries in NVD, when appropriate.
Analyse Vulnerabilities to Identify Their Root Causes (RV.3): Help reduce the frequency of vulnerabilities in the future.	**RV.3.1**: Analyse identified vulnerabilities to determine their root causes.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	RV.3.2: Analyse the root causes over time to identify patterns, such as a particular secure coding practice not being followed consistently.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	RV.3.3: Review the software for similar vulnerabilities to eradicate a class of vulnerabilities and proactively fix them rather than waiting for external reports.	DG 1/2/3: Configuration of tools, work items in the secure development process workflow systems, and bug history of security bugs in the product workflow system.
	RV.3.4: Review the SDLC process and update it if appropriate to prevent (or reduce the likelihood of) the root cause recurring in updates to the software or in new software that is created.	DG 3: Updates to training addressing problems found in root cause analysis to eliminate classes of vulnerabilities.

Annex D (informative): Evolution of Secure by Design

Referencing Figure 4.1-1, Software Security Design Development Timeline, clause 4 notes that the inflection point for modern communication was 1964 when Paul Baran and Sharla Boehm at the RAND Corporation and Donald Davies at the UK National Physical Laboratory articulated the use of digital computers to effect large, distributed networks. Baran in his papers [i.65] realized at the outset in 1964 that fundamental vulnerabilities existed in the use of the technology. This reality led to two-years of NSA and RAND studies, and security scientists Bernard Peters and Willis Ware to assemble colleagues from their organizations at the 1967 AFIPS Spring Joint Computer Conference. The event followed two earlier conferences in 1965 hosted by the RAND spin-off, System Development Corporation (SDC), on safeguarding computer systems "operated in a time-shared mode." Ware was the founding president of American Federation of Information Processing Societies (AFIPS). His presentation at the conference together with NSA computer scientist Bernard Peters - and especially the diagram below on networked computer system vulnerabilities and associated risks became the seminal and defining point for modern cybersecurity [i.66], [i.75].

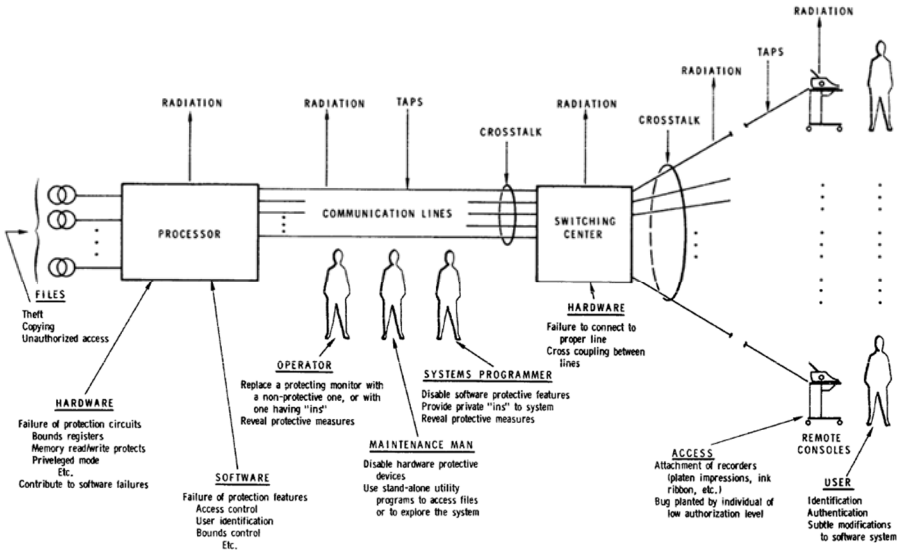

Figure Annex D-1: Networked Computer System Vulnerabilities [i.66]

The two 1967 Atlantic City conference papers [i.66] initiated a major responsive ARPA task force. Its report was assembled under the Defense Science Board and published as a 78-page US DOD 1970 report, Security Controls for Computer Systems, [i.67] informally known as the "Ware Report." The Report identifies and extensively treats cybersecurity vulnerabilities, establishes an ontology, the strategic implications, and related processes - establishing the essential elements for security by design.

The years that immediately followed these seminal developments were marked by numerous initiatives.

- **Post-Cold War Launch (1970s-1980s):**
  Motivated by rising security concerns - and by the perspective on computer security put forth in the Defense Science Board's Ware Report - the U.S. Department of Defense (DoD) funded early research into secure computing systems. Notable early efforts included the *Multics* project, which pioneered the use of kernels, rings, and access control models to enforce protection

- **Government & Academic Collaboration:**
  These foundational efforts were often partnerships among government agencies, university research teams, and vendors. The goal was to build systems with rigorous, demonstrable controls to manage sensitive or classified military data.

- **Focusing on Product Evaluation**
  A series of workshops sponsored by U.S. DoD and the National Bureau of Standards (later the Institute of Standards and Technology) determined that, rather than funding and sponsoring the creation of secure computer systems, DoD should publish criteria for secure systems and evaluation commercial products against those criteria. The assumption was that this approach would lead to systems that were secure and suitable for wide use while reducing the cost of such systems to DoD.

The initiative of the 1970s led to the creation of the Trusted Computer System Evaluation Criteria (TCSEC).

- **The Orange Book Emerges (Early 1980s):**
  In response to the findings of the DoD/NIST workshops and to an increasing need for secure systems, DoD established the NSA Computer Security Center. The center published the Trusted Computer System Evaluation Criteria in 1983—nicknamed the "Orange Book." It introduced a ranking scheme (D to A) based on progressively more stringent security requirements: mandatory access control, object reuse control, and formal verification. [i.70], [i.72].

- **Adoption by Vendors & Agencies:**
  Major government and commercial vendors began designing or certifying systems to meet Orange Book standards (e.g. for handling classified data). The ranking hierarchy aimed to assure buyers that products met defined security thresholds. The higher-ranking levels of the Orange Book emphasized features that enabled systems to protect classified defence information from unauthorized disclosure.

The experiences of TCSEC in Practice led to an appreciation of the adverse impacts and limitations of TSEC approaches to cybersecurity and SbD.

- **Initial Success & Funding Incentives:**
  The NSA Computer Security Center required vendors to comply with the Orange Book compliance as a requirement for selling secure systems to the U.S. government, leading to substantial investment by vendors developing hardened products. Some government agencies embraced these systems, but vendors found little market for the higher-ranked systems that were tailored to protect classified information. Most evaluated systems were lower ranked and included only features that appealed to commercial customers.

- **Technical and Practical Challenges:**
  Over time, many Orange Book requirements, such as formal validation and elimination of covert channels that could compromise classified information, proved overly stringent or cost prohibitive. The graded D-A evaluation became cumbersome. Only a handful of systems ever achieved high B- or A-level evaluation, and even those provided limited practical usability and found very limited markets.

The depreciating value of TSEC led to stagnation and critiques in an evolving and expanding public marketplace.

- **Mismatch with System Complexity:**
  As computing environments grew in scale and diversity (e.g. networked systems, distributed apps), Orange Book rules - designed originally for monolithic mainframes - became increasingly difficult to apply. [i.72].

- **Evolving Threat Landscape:**
  With greater focus on network security, secure software engineering, and emerging attack vectors (malware, web vulnerabilities), Orange Book approaches were seen as increasingly narrow and outdated . [i.71]

The period beginning in the 1990s was marked by significantly different new ICT technologies, products, networks in a rapidly expanding global marketplace. New organizations and approaches occurred.

- **Expansion to New Evaluation Schemes:**
  In the 1990s and 2000s, other frameworks emerged - e.g. ITSEC (EU). [i.73] These systems offered more modular, flexible approaches, better suited to new computer system functions and models and to multi-vendor, cross-platform environments.

- **Fade from Prominence:**
  By the late 1990s, the Orange Book was effectively eclipsed. It was officially supplanted by the international Common Criteria, established in 1999. [i.74] The Common Criteria replaced the Orange Book in the United States and has been adopted by more than twenty countries.

- **Network Security and other Configuration Guides**
  The period from 1995 onwards is primarily marked by extensive introduction of open, autonomous, complex computational systems and networks coupled with an increase of threat surfaces from all manner of vulnerabilities and actors that dramatically increased the challenges of risk management. Perhaps the most seminal event marking this paradigm change was the creation by NSA of the Systems and Network Attack Center (SNAC) in August 1995 known organizationally as C4. SNAC set in motion a considerable array of initiatives and standards that were subsequently explained in 2001 via "The 60 Minute Network Security Guide (First Steps Towards a Secure Network Environment)" to "better understand how to reduce and manage network security risk" [i.69].

Annex E (informative): Bibliography

- AFCEA: "From the Desk of the Cyber Committee", August 2025.
- BLACKDUCK®: "White Papers Library".
- NASA: "Secure Software Development Self-Attestation Resources and Knowledge".
- National Academies of Sciences, Engineering, and Medicine: "Defense Software for a Contested Future: Agility, Assurance, and Incentives", 2025.es Press.

History

Version	Date	Status
V1.1.1	March 2026	Publication