Secure by Design

Version 1.1

A Guide to Assessing Software Security Practices

June 2026

CIS Center for Internet Security®
CIS Controls®
SAFECode

Acknowledgments

The Center for Internet Security® (CIS®) would like to recognize the following individuals for their support in creating this guide. Their time and expertise were a vital component of completing this important work.

Principal Authors

Curt Dukes, Center for Internet Security
Phyllis Lee, Center for Internet Security
Steve Lipner, SAFECode
Tony Rutkowski, Center for Internet Security
Tony Sager, Center for Internet Security
Sharon Shoemaker, Center for Internet Security

Community Reviewers and Contributors in their personal capacities

Martin Beauchamp, GSMA
Ray deMeo, GrammaTech
Freddy Dezeures, subject matter expert
Rick Doten, Centene
Dan Farmer, subject matter expert
Dr. Rhonda Farrell, Global Innovation Strategies
John Gilligan, Center for Internet Security
Manuel Ifland, SAFECode and Siemens Energy
Dennis Kirby, SANS
James Kyne, SANS
Rob Lee, SANS
Magdalena LoGrande, Cybersecurity Engineering Fellow SigmaDefense
Tony Rice, SAFECode and Microsoft
Ken Sutton, Australian Signals Directorate (ASD)
Bee Tapp, Australian Signals Directorate (ASD)
Paul Waller, National Cyber Security Centre (NCSC)
Rob Whelan, Australian Signals Directorate (ASD)

Contents

Executive Summary

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."[1] 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 lifecycle. 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. 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.

In 2023, the Department of Homeland Security's (DHS) Cybersecurity and Infrastructure Security Agency (CISA) sharpened national focus with its **Secure by Design (SbD)** initiative,[2] 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,"[3] Black Duck's "Building Security In Maturity Model (BSIMM),[4] the UK (National Cyber Security Centre) (NCSC) "Software Security Code of Practice,"[5] Armed Forces Communications and Electronics Association (AFCEA) "Secure by Design — Next Steps,"[6] Google's "Secure by Design at Google,"[7] and National Institute of Standards and Technology (NIST®) Secure Software Development Framework (SSDF), SP 800-218 v1.1)[8] offer recommendations, but they are not always aligned, leaving developers and users uncertain about which to follow and how to proceed.

At the time of the initial publication of these guidelines, the U.S. Government's Security by Design activity continues to evolve rapidly. As directed by Executive Order 14306,[9] NIST is updating the SSDF and has established a consortium under the auspices of the National Cybersecurity Center of Excellence (NCCoE) to develop guidance that demonstrates the implementation of best practices based on the SSDF.[10]

To help meet the SbD challenge, the **Center for Internet Security® (CIS®)** partnered with non-profit **SAFECode** and a community of experts to develop independent, practical, and actionable guidance. Our goal: strike a balance between broad principles and concrete, technology-specific practices in a way that is adaptable across lifecycle models, platforms, and development practices. After reviewing leading frameworks, we concluded that NIST's **SSDF** provides the best foundation across different development environments. This guide builds on that foundation.

The guide first 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, we accompany this guide with a detailed spreadsheet, A Measurement Aid to SbD – A Guide to Assessment Software Security Practices, that enumerates recommended

1 eWeek, "AI Might Give Security Defenders an Advantage: Researcher Explains During Black Hat 2025 Keynote."
2 CISA, "Secure-by-Design."
3 SAFECode, "Fundamental Practices for Secure Software Development."
4 BlackDuck, "Building Security In Maturity Model (BSIMM)."
5 NCSC, "Software Security Code of Practice."
6 AFCEA, "Secure by Design – Next Steps."
7 Google, "Secure by Design at Google."
8 NIST, Special Publication 800-218, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities."
9 Executive Order 14306 of June 6, 2025.
10 NIST Draft Special Publication 1800-44A, Secure Software Development, Security, and Operations (DevSecOps) Practices.

developer activities, incorporates SAFECode's **Development Groups (DGs)** model, and where possible, maps SSDF tasks to the **CIS Critical Security Controls (CIS Controls®)**.[11] For each task, we also identify responsible roles and the artifacts that demonstrate compliance.

The use of artificial intelligence (AI) is an important and rapidly evolving consideration for software development and software security. The security implications of AI are complex, multifaceted, and still emerging. This guide includes a discussion of three ways that AI can be used to improve software security and enhance SbD as well as a brief discussion of the security of AI applications. The accompanying spreadsheet includes recommended activities to exploit AI to enhance software security and to address the security issues raised by the use of AI in software development. The reader who is concerned about AI and software security is advised to understand these interactions, their evolving nature and their associated risks.

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

Major government software security initiatives under Presidential Executive Orders,[12] the EU Cyber Resilience Act (CRA),[13] among others listed in Appendix A, emphasize that software development organizations bear responsibility for delivering software that is secure by design. The authors urge software developers to adopt the practices and evidence-based approaches to risk management described in this guide, both as a matter of responsibility to customers and to enable compliance with those policies.

This publication 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.

In short, this document addresses a long-standing gap in cybersecurity: practical, evaluable, and aligned guidance for building software that is **secure by design.**

11 CIS, CIS Critical Security Controls, v8.1. Also published as ETSI, TS 103 305-1 V5.1.1 (2025-09), Cyber Security (CYBER); Critical Security Controls for Effective Cyber Defense; Part 1: The Critical Security Controls.

12 Executive Order 14028 of May 12, 2021 sustained by the Executive Order 14306 of June 6, 2025. See also, NIST, Executive Order 14028, Improving the Nation's Cybersecurity.

13 Cyber Resilience Act. See also, Cyber Risk: Cyber Resilience Act (CRA) | Updates, Compliance.

Introduction

Cybersecurity, the current term of art for electronically protecting information and operations throughout their lifetime, is primarily dependent on the secure development and continued maintenance of software among vendors and users. Moreover, there is an essential relationship between secure development practices and secure lifecycle operational management. Effective software development practices and features make operational management easier and demonstrably better.

Much of what we know as the cybersecurity industry today has occurred in reaction to the challenge of flawed systems, rather than incorporating a design with sufficiently robust defensive characteristics. Further, frequent security bulletins and advisories, disruptive patching, complex configuration management, malware analysis and detection, and vulnerability disclosure may have created more security "fatigue" than progress. Finally, well-intended public policy and economic "enforcement" machinery has resulted in dozens of security frameworks, and disconnected or competing efforts at regulation, compliance, and incentives.

In 2023, DHS CISA brought more focus to this challenge with their "Secure by Design" (SbD) initiative[14] in which they challenged technology vendors to "prioritize the security of customers as a core business requirement" and "implement Secure by Design principles to significantly decrease the number of exploitable flaws." Faced with a variety of available studies and recommendations related to cybersecurity and SbD, CISA's challenge leaves users/developers/assessors confused about which guidance to follow.

The Center for Internet Security, (CIS), partnering with non-profit SAFECode (a global industry forum promoting scalable and effective software security programs) recognized this dilemma and assembled a team of community and industry experts to produce the independent and actionable guidance within this paper — aiming for a "sweet spot" between principles and technology-specific requirements. We determined in particular that the design and implementation principles articulated in NIST's Secure Software Development Framework (NIST Pub SP 800-218 - SSDF)[15] served as a solid foundation to meet our goal of providing focused and feasible guidance across a range of development scenarios.

The next section (Background) of this paper provides a brief history of SbD. Section 3 (Achieving and Assessing Security by Design) is broken down into the principles of SbD (The Six Considerations of Secure by Design) and how to assess adherence (Assessment and Evidence) to the SbD principles by implementing practices and tasks within NIST's SSDF (SP 800-218) referred to throughout this paper as "NIST SSDF" or "SSDF") and applying effective risk management practices.

Our guide summarizes and leverages the maturity structure (referred to as Development Groups — DGs) defined in SAFECode paper "Application Software Security and the CIS Controls"[16] and where possible, we further break down the SSDF practices and tasks in terms of implementable activities defined in the Center for Internet Security (CIS) Critical Security Controls (CIS Controls)[17] and the referenced SAFECode paper, identify the role(s) responsible for SSDF tasks, and identify artifacts that show compliance. (Appendix G provides an in-depth mapping of SSDF practices and tasks to actionable activities, prioritized to allow maximum applicability across development groups and to provide a roadmap for assessment.) Finally, we also provide the reader with a glossary of terms as used in this guide, a rich set of additional appendices to amplify other information within this guide, and a checklist to help streamline any SbD assessment.

This publication 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.

In short, this document addresses a long-standing gap in cybersecurity: practical, evaluable, and aligned guidance for building software that is **secure by design.**

14 FN 2, above
15 FN 8, above
16 SAFECode "Application Software Security and the CIS Controls"
17 FN 12, above.

Background

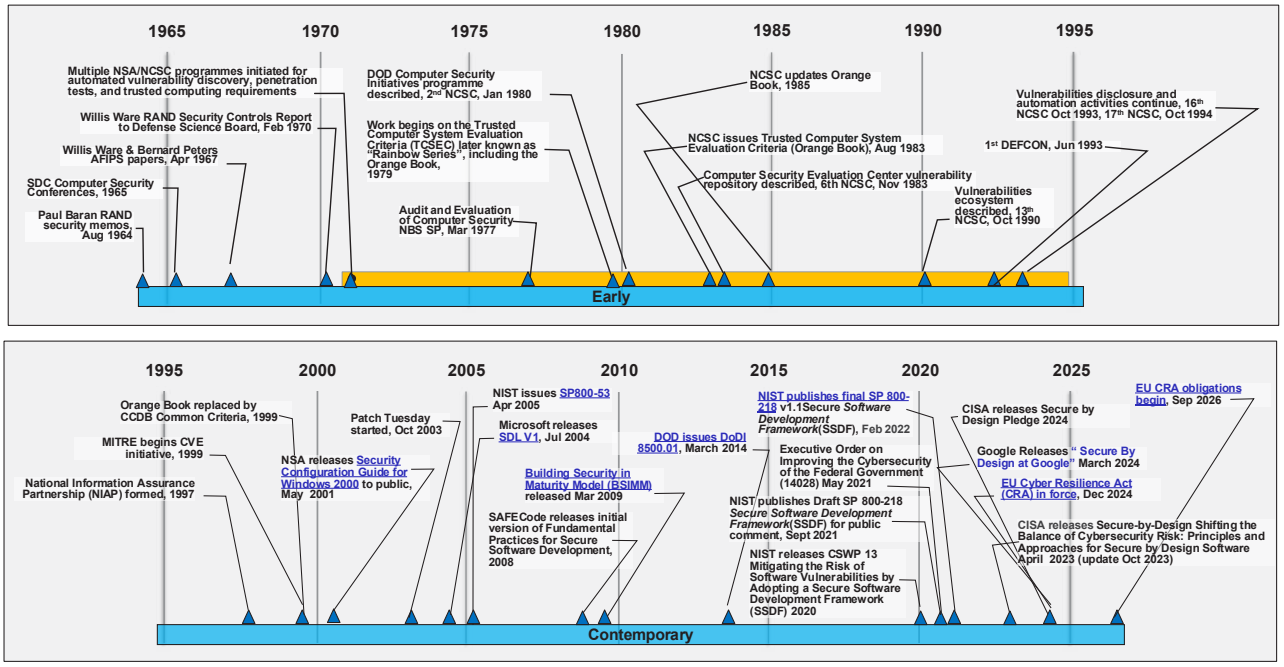
The challenges of information security emerged early. David Kahn in his reference book "The Code Breakers"[18] describes many of them over the millennia. GCHQ's (United Kingdom's Government Communications Headquarters) authorized history points to the extensive use of cryptography by spymaster Francis Walsingham during the reign of Elizabeth I.[19] 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.[20] 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 Appendix A), the following concepts emerged and remain as cybersecurity fundamentals today.

1. Security cannot be attained in the absolute sense.
2. 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.
3. Any loss which might occur must be detected.
4. There are several minimum requirements to establish an adequate security level for the software of a networked computer system.
5. Management must be aware of the need for security safeguards and be willing to support the cost of obtaining this security protection.
6. Technical expertise and practical experience must be combined to judge whether or not a desired security level has been reached.
7. No software system can attain a zero risk of loss.
8. It is necessary to reach an acceptable degree of risk.

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 systems' 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 lifecycle of systems. (A more in-depth history is available in Appendix B.)

18 Kahn, David, "The Code-Breakers The Comprehensive History of Secret Communication from Ancient Times to the Internet" December 5, 1996
19 Johnson, "The Evolution of British Sigint 1653-1939", The Stationary Office Ltd., July 1998
20 Desvignes, "A French Story", Cryptologic History Symposium, 11 May 2022, https://www.arcsi.fr/doc/CHS_2022_Presentation_ARCSI.pdf

Timeline of Secure by Design



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 lifecycle 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 CIS Control 16, "Application Software Security."[21] With that Control as one foundational catalyst, we offer these basic tenets of Secure by Design:

- 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.
- Software development organizations must 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.

21 CIS Control 16

Achieving and Assessing Security by Design

This guide and the accompanying appendices and spreadsheet are intended as resources 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."

The perspective of this guide is that there are a set of fundamental security practices that all software development organizations must 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 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.

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, security testing and other assurance activities must 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 must protect its development systems using classic IT security measures such as those defined in the CIS Controls. If it is using a cloud-based development repository such as GitHub, it must 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 must 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 must access those services using channels that are encrypted and authenticated and rely on accepted (standardized) secure protocols. Furthermore, the software development organization must have a basis for believing that the SaaS provider's software is secure — that the SaaS provider implements the fundamental security practices.

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. We use the three-part Development Group construct defined in SAFECode paper "Application Software Security and the CIS Controls"[22] to prioritize activities commensurate with the risks and business cases typical of those facing a software development organization.

An organizational culture that prioritizes security must 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. It is difficult to assess culture and commitment in isolation but easy

22 SAFECode paper defining DGs

to tell whether they are present; if a software development organization is doing an excellent job at executing the more technical practices and tasks of the SSDF, its culture "must be doing something right." At the end of the day, results are what matters.

To enable assessment of SbD, we articulate activities in terms of the practices and tasks defined in NIST SSDF. 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, we define 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. We identify roles typically associated with responsibilities for the activities as an aid to software development organizations seeking to create or assess their SbD programs.

This 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 must be 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 this 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, this 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.

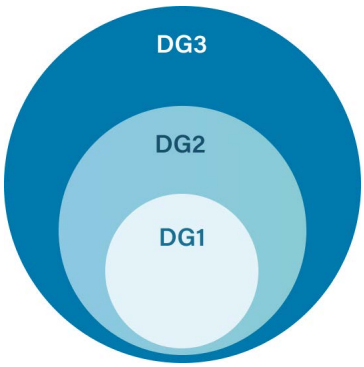
The Six Considerations of Secure by Design

Secure by Design encompasses what developers call "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. When lacking usable guidance, end users/organizations will evaluate the sufficiency (sometimes unconsciously) by using or not using a product or service, by deciding what information to expose or not expose to the system, and by complaining to suppliers or not. As one practical example of sufficiency, please see the CIS paper "A Guide to Defining Reasonable Cybersecurity,"[23] which provides specific guidance to end users/organizations seeking to develop a cybersecurity program that satisfies the general standard of reasonable cybersecurity.

As mentioned earlier, 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. Using the model defined in the SAFECode paper "Application Software Security and the CIS Controls,"[24] we address organizational maturity in terms of three tiers called "Development Groups" (DGs). These DGs are loosely tied to the complexity of a software development organization's development environments (Appendix C provides an expanded definition) and drive the prioritization of tasks and activities, thus guiding less mature software development organizations to direct limited resources toward high value activities.

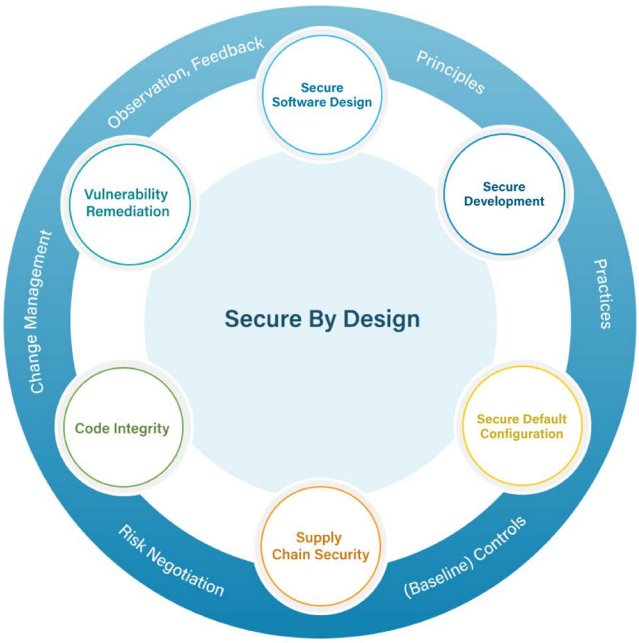
23 CIS, "Reasonable Cybersecurity Guide".
24 FN 27, above

Development Group	Definition
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.
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.
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 is necessary to unpack the considerations of "secure by design" if we're to define prescriptive requirements or criteria. Six SbD considerations are summarized in the following chart and described below in further detail. The chart also maps the considerations to SSDF Practices and Tasks that provide guidance on each. (Some of the tasks under SSDF Practice for "Prepare the Organization" (PO) are overarching rather than being mapped to individual considerations.) The reader will quickly note that each consideration is underpinned by lower-level concepts, all of which contribute to the complexity of achieving a secure design.

SbD "Considerations"	Description	SSDF Practices/Tasks
1. Secure Software Design	The functions of the system or software and the underlying structure or architecture that enables 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 don't 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 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



1. Secure Software Design

Secure software design begins with a set of security objectives that the software must satisfy 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 requirements for any secure system are usually confidentiality, integrity, and availability. A fourth common requirement is accountability.

Fundamental Requirement	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 requirements are typically achieved by the application of 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 must not only be implemented correctly but also must be 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 specifies that security features must be invoked on every access to information, must be protected from unauthorized tampering, and must be implemented correctly. A paper from the 1970s describes principles for secure design that expand on the reference monitor concept.

Secure system design teams in industry frequently rely on a process called threat modeling to review their designs. Threat modeling does not focus on threats in the sense of a hostile agent who seeks to undermine system security. Rather, in threat modeling, the designer analyzes 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 modeling process identifies specific action items for designers and developers to make software more resistant to attack. Threat modeling is a well-established and accepted approach that is supported by books, commercial and open-source tools, and training. (See Appendix D for a summary of the classes of attacks identified by a common threat modeling approach.)

In the traditional world of waterfall or spiral development lifecycles, development — coding, unit testing, system testing — followed design. With the evolution of modern development practices such as Agile and DevSecOps,[25] 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 this paper, 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 must consider threats and countermeasures and must be guided by threat modeling or a comparable process.

2. Secure Development

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.

A secure development process must include:

- A "bug bar" that identifies "must 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.

25 FN 10, above.

Although the list above is brief, (see Appendix E for a list of tool classes and links) the secure development process must be very detailed in terms of mandated or forbidden coding constructs, "must fix" errors from security tools, and configuration options that are mandated, allowed, or forbidden. As discussed below, the process must 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, the process must also include a policy that addresses updating legacy code to meet security requirements imposed after the software was released.

3. Secure Default Configuration

Software products frequently include myriad configuration settings and options that system administrators may use to enable or disable features or adjust product behavior. 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 must 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 must provide features for controlling access to elements of the attack surface, and the design and development team must 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 must be disabled by default.

While software developers must minimize the attack surface they must maximize the use of security features. Developers must 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 must 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 must be part of the secure development process, and product teams must ensure that software is locked down by default before release. Appendix E provides a list of tool classes and links.

4. Supply Chain Security

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 must 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 recommendations of this paper (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 must 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. 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.[26]

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 must 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. Code Integrity

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 must configure and operate them securely. The CIS Controls provide an effective and practical set of practices for protecting IT environments, including software development environments. Personnel vetting is outside the scope of this paper but should not be ignored by software development organizations that work on sensitive or critical software.

6. Vulnerability Remediation

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 must implement processes to accept reports of product vulnerabilities, investigate the reports, and remedy the vulnerabilities. They must encourage vulnerability researchers to report and should consider sponsoring "bug bounties" to achieve that end. They must ensure that vulnerability advisories and fixes are communicated to their users, ideally in a standardized form such as Common Security Advisory Framework (CSAF)[27] and accompanied by context on their severity[28] and exploitability.[29]

Vulnerability reports are also a critical part of secure development. All software development organizations (regardless of DG level) must be prepared to accept vulnerability reports, respond, do root cause analysis, and track their work. Software development organizations must not only fix the vulnerabilities that are reported to them

26 https://safecode.org/supply-chain/software-products-arent-cookies/
27 https://www.csaf.io/
28 https://www.first.org/cvss/
29 https://cyclonedx.org/capabilities/vex/

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 modeling 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, described here, and discussed further in Appendix G:

- 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.

Artificial Intelligence

The use of artificial intelligence (AI) has emerged as an important consideration for software development and software security. There are two separate sets of AI-related issues that software development organizations will face:

1. The organization may use AI models as part of its development and software security processes.
2. The organization may build and deliver software systems that incorporate AI models.

This section addresses each set of issues in turn.

Artificial Intelligence in Secure Software Development

The use of AI in software development is emerging as an important contributor both to software production and to software security analysis. Research has shown that simply prompting an AI model to create code that performs a function will often result in vulnerable code.[30] This phenomenon is to be expected since the AI model is trained on source code downloaded from across the Internet, and the security of that massive body of code is uncertain at best.

Prompting the model to consider security as part of its development process can help to mitigate the problem, but is not guaranteed to be effective, and can depend on the prompt providing direction about which specific security errors to avoid. The end result is likely to be code no more secure than a human programmer would write. For this reason, it is important that AI-generated software be subject to the same security assurance measures that are applied to human-written software, including use of software security tools (including AI tools — see below) and human code review.

AI has been integrated into commercial threat modeling tools.[31, 32] Those tools help program managers and architects explore their proposed system designs and enumerate potential threats (weaknesses) that the designs must mitigate. Integrating AI into threat modeling appears to enable deeper analysis and better enumeration and understanding of threats, potentially helping to identify exposed attack surface and prioritize mitigations. The enumeration of threats by AI-aided tools may not be perfect, but threat modeling has historically been an imperfect process, and the addition of AI is a worthwhile enhancement.

The use of AI models to analyze and remediate software security has emerged as an effective approach to improving software security. These models are trained by being exposed to descriptions of software vulnerabilities and successful attacks. Such models have demonstrated the capability to detect weaknesses in software and

30 Beyond Penetrate-and-Patch, Mike Hicks and Steve Lipner, submitted for publication in the Communications of the ACM, May 2026, https://mhicks.me/blog/beyond-penetrate-and-patch/

31 https://www.iriusrisk.com/

32 https://github.com/awslabs/threat-composer/blob/main/docs/AI-CLI-MCP.md

develop multi-step attacks that exploit those weaknesses.[33, 34] Although not directly within the scope of secure software development, it is important to point out that the AI models that detect vulnerabilities for developers can also be used offensively to plan or execute attacks on software.

The AI models appear to be effective at finding memory safety vulnerabilities. It seems likely that the integration of AI-based vulnerability detection systems into secure development processes will become a best practice both to prevent the introduction of vulnerabilities into new code, and to review existing code. (Even though existing code may have been reviewed previously, it may not have been reviewed by an LLM frontier model that can identify new classes of issues.) In part, those vulnerability detection systems operate by applying existing security tools such as static analysis, fuzzers and configuration testers, so they may automate parts of an organization's secure development process and do so effectively. However, the functioning of the AI-based tools is probabilistic, and the roles and performance of these systems are still evolving, so organizations should not treat their use as a replacement for other aspects of their secure development processes.

AI models have also demonstrated the ability to remediate specific vulnerabilities. As with the vulnerability-finding models, these models are trained by exposure to the "before and after" on the Internet of actual vulnerabilities and their corrections. The same cautions should apply to AI for correcting vulnerabilities as to AI for detecting them: the fixes should be subject to the organization's normal quality and secure development processes.

In summary, artificial intelligence tools can be a powerful addition to the effectiveness of a secure software development process, but their use is accompanied by significant uncertainties. Software development organizations that adopt AI tools should treat them as a supplement to other software security practices rather than a replacement for them and should conduct careful and regular analysis of their effectiveness before deciding to abandon tools and processes that have proven effective. Because the capabilities of AI models are evolving very rapidly, it is important for software security teams to monitor new developments and adapt their practices as necessary, both to take advantage of new AI security tools and to respond to emerging offensive uses of AI.

Agentic AI systems in which the AI model is used to control the operations of other software or even physical devices are of special concern. AI models are statistical rather than being composed of deterministic logic, and their outputs are probabilistic (pseudorandom). Thus, it is not possible to guarantee the security or other critical properties of the AI model. If the output of the model is used in a critical application, the best practice is to validate or constrain it with human judgment or deterministic (non-AI) software. The high-level guidance is to "proceed with caution."

33 Anthropic Frontier Red Team, "Claude Mythos Preview," April 2026, https://red.anthropic.com/2026/mythos-preview/. See also Anthropic, "Project Glasswing," https://www.anthropic.com/glasswing

34 Bobby Holley, "The zero-days are numbered," Mozilla Blog, April 2026, https://blog.mozilla.org/en/privacy-security/ai-security-zero-day-vulnerabilities/

Assessment and Evidence

The natural question now becomes: how does one know that the various principles and considerations above have been followed? Customers and regulators may ask software development organizations to demonstrate that their products are "secure by design." That does not mean that the software is vulnerability-free — there is no such thing. Rather, it means that the software development organization has addressed the six considerations listed above.

Of the six considerations, 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 the software development 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 this guide.

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.[35] 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. (Appendix F provides a list of typical roles that support the SbD lifecycle.)

Beyond the technical and product-specific activities associated with the six considerations is the matter of organizational commitment to software security. SSDF practice PO.2 speaks 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 must be considered in the context of the evidence and effectiveness of the software development organization's actions in response to the six considerations.

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.

Evidence that is created solely to satisfy an assessor or evaluator should be considered with great suspicion because it may not actually be connected to the software.

The secure by design principles articulated in U.S. government documents and in 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?

35 See NIST SP 800-216, "Recommendations for Federal Vulnerability Disclosure Guidelines, https://doi.org/10.6028/NIST.SP.800-216; CMU Software Engineering Institute "The CERT® Guide to Coordinated Vulnerability Disclosure", https://insights.sei.cmu.edu/documents/1945/2017_003_001_503340.pdf; FIRST, "Product Security Incident Response Team (PSIRT) Services Framework", https://www.first.org/standards/frameworks/psirts/psirt_services_framework_v1-1; ISO/IEC 29147, "Information technology — Security techniques — Vulnerability disclosure," (CHF 155 paywall) https://www.iso.org/standard/72311.html; ISO/IEC 30111, "Information technology — Security techniques — Vulnerability handling processes", (CHF 98 paywall) https://www.iso.org/standard/69725.html

Measures of confidence in the security attributes of a product or service are usually broken down 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?

Appendix G provides an in-depth mapping of SSDF practices and tasks to actionable activities, some of which reference the CIS Controls and others of which reference the companion SAFECode document "Application Software Security and the CIS Controls". These activities are also prioritized to allow maximum applicability across development groups and to provide a roadmap for assessment. As an example:

Practices	Tasks	Critical Security Control (CSC)/ SAFECode Companion Guide (SCG) Mapping	DG1 Activities	DG2 Activities	DG3 Activities	Artifact	Responsible Role
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.	**CSC16.1:** Establish and Maintain a Secure Application Development Process	▪ DG1 Activities	▪ DG2 Activities	**DG1/2/3** ▪ Documented organization charts and position descriptions that align with responsibilities identified in policies. ▪ Organizational communications of updated roles and responsibilities.	**DG1/2/3** ▪ Documented organization charts and position descriptions that align with responsibilities identified in policies. ▪ Organizational communications of updated roles and responsibilities.	**SDL Team:** Engineering Group Leadership
	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.	**CSC16.9:** Train Developers in Application Security Concepts and Secure Coding **SCG** ▪ Train the Developers	1 Consider providing necessary training for all developers that include details of the organization's secure development practices and associated tools the organization relies on. 2 Consider implementing CSC16.9: Ensure that all software development personnel receive training in writing secure code for their specific development environment and responsibilities. 3 Consider updating the training as needed.	▪ DG1 Activities 1 Provide just in time training associated with tools and tool outputs including error messages and how to fix reported errors. 2 Provide defensive programming training.	▪ DG2 Activities	**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.	**SDL Team**

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, one must look for:

- 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 that must be 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 SSDF 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 must execute. The secure development process tells the developers which development, analysis, and testing tools they must run, what secure coding conventions they must follow, which tool outputs represent "must fix" and "should fix" bugs, and which analysis and design steps they must follow (e.g., threat modeling). 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 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, analyzing, 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 must 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, this 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 "must 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 "must 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 must act to "fix" those bugs by updating its process and adding new secure development tools, training, or processes.

After reading this section of the guide, the reader may be left with the question "but how do I know whether the product is good enough?" The answer to that question is a matter of risk management — first by the software development organization and then by the end user/organization of the software.

The set of errors that the software development organization designates as "must 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's 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.

Conclusion

The security of software has long been recognized as one of the foundational problems threatening our economic and social lives. Rapid changes in technology and connectivity (of systems, supply chains) have made the problem even more complex and dynamic.

Most recently, the Secure by Design work by DHS/CISA and the EU's CRA have brought renewed attention to the problem and identified some key principles or expectations that must collectively be adhered to by developers of software and systems.

In this paper, CIS and SAFECode bring together a diverse community of experts to identify:

- The most important security principles to expect in the development, distribution, maintenance, and use of software;
- The specific actions expected for each principle; and
- The measures that we should assess to gain confidence in the security for each action.

All of this is done in the context of the CIS Critical Security Controls Version 8.1, and the complementary work by SAFECode (which is in turn based on existing standards and principles).

Among the guidance provided in earlier sections and the appendices, a few very important themes are worth reiterating.

- **Perfect security is impossible.** The goal is to manage risk during the development cycle and throughout operational management.
- **Assessment is key.** This paper offers concepts and qualitative and quantitative metrics to assess across SSDF.
- **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.
- **Secure design** encompasses both the overall architecture design and incremental designs of features and enhancements. Both must consider threats and countermeasures, and both must be guided by threat modeling or a comparable process.
- **The use of Artificial Intelligence (AI)** can pose challenges for the development of secure software, but AI, properly applied, can also make valuable contributions to software security.

We recognize that securing software-based systems is not a quiz — there is no final answer. But it is also a problem that has not aged well, and so we offer this work as a positive, constructive step forward and we welcome your feedback and ideas for improvement.

Glossary

The following terms are defined as follows for purposes of this paper.

Artifact	A piece of evidence that supports a response to a question. For purposes of this document, artifacts are generated as a part of the development process and not for the sole purpose of proving compliance to the process.
Attack Surface	The set of exposed interfaces and functions in a software product.
Code Integrity	Mechanisms and processes to ensure software is unchanged and functions as intended throughout its lifecycle.
Confidentiality	The system protects information from unauthorized disclosure.
Control Frameworks	A set of processes and activities that describe the security controls that are the foundation of a security program.
Default Configuration	The preset or standard configuration, setting, or behavior of a system, software, or device.
Evaluable	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.
Risk Frameworks	A set of processes to provide a consistent approach for managing and assessing risk — often in terms of business risk.
Secure by Default	Secure configurations and settings are the default for all systems and software.
Supply Chain Security	Measures that assure development teams that "third-party" components will not undermine the security of their products.
System 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.
System 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.
System Authentication	A mechanism used to identify a user. This may rely on passwords, public-key mechanism, challenge-response protocols, or other approved techniques.
System Authorization	A mechanism used to determine access levels or user/client privileges. This may rely on access control lists or classification labels that are compared with users' clearances.
System Availability	The system provides access to information and processing services when required.
System Integrity	The 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.

Appendix A

Key Sources and Other Resources

The editors, community reviewers, and contributors relied on key sources and other resources, including those listed below, to create the Secure by Design: A guide to Assessing Software Security Practices. The reader will see some of below included in the list of endnotes. This occurs when the guide contains specific references to those sources. Other sources below provided general background information and are included here for readers to investigate further should they wish.

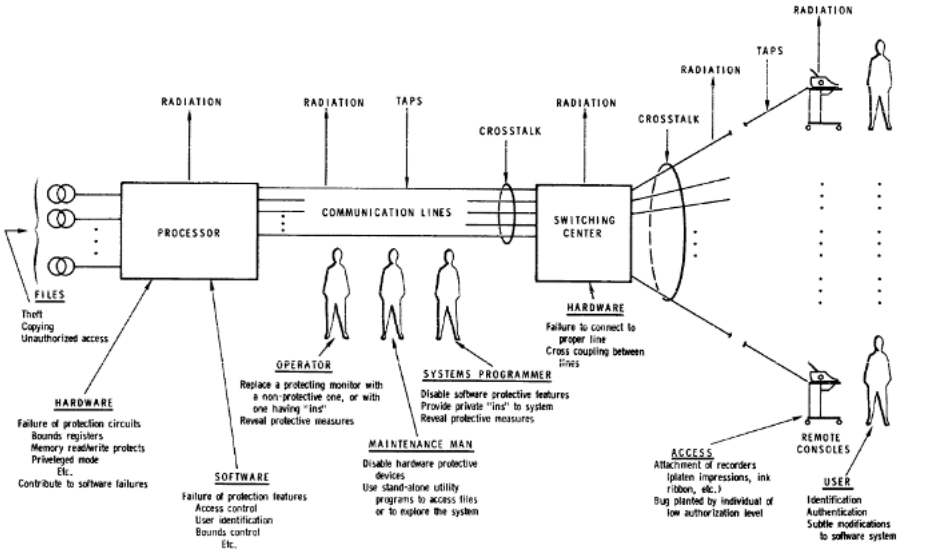
- Armed Forces Communications and Electronics Association (AFCEA) "Secure by Design — Next Steps
- Automated tools [IriusRisk, OWASP, Microsoft]
- Building Security In Maturity Model (BSIMM)
- CIS, CIS Critical Security Controls, v8.1. Also published as ETSI, TS 103 305-1 V5.1.1 (2025-09), Cyber Security (CYBER); Critical Security Controls for Effective Cyber Defense; Part 1: The Critical Security Controls.
- Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148 (NIS 2 Directive) (Text with EEA relevance)
- ENISA: Advancing Software Security in the EU, 2020.
- EU Cyber Resilience Act
- France ANSSI: Les Essentiels DEVSECOPS, Feb 2024.
- France ANSSI: Les Essentiels Sélection d'un Logiciel Libre, May 2025.
- Germany BSI: TR-03185 Secure Software Lifecycle, 2024.
- Improving the World's Cyber Resilience, at Scale. Implementing Baseline Security by Default
- Michael Howard and Steve Lipner, The Security Development Lifecycle (Microsoft Press, 2006)
- National Cyber Security Centre NCSC's Software Security Code of Practice
- 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)
- 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)
- Regulation (EU) 2022/2554 of the European Parliament and of the Council of 14 December 2022 on digital operational resilience for the financial sector and amending Regulations (EC) No 1060/2009, (EU) No 648/2012, (EU) No 600/2014, (EU) No 909/2014 and (EU) 2016/1011 (Text with EEA relevance)
- SAFECode companion paper to Control 16, Application Software Security and the CIS Controls
- Secure by Design at Google", Christoph Kern (Google Security Engineering Whitepaper) March 4, 2024 https://storage.googleapis.com/gweb-research2023-media/pubtools/7661.pdf
- Shostack, Adam: Threat Modeling: Designing for Security
- SP800-218, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities
- SP800-218A Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile
- Technical reports [SAFECode]
    - https://safecode.org/resource-centers/

    - https://safecode.org/uncategorized/fundamental-practices-secure-software-development/
    - https://safecode.org/category/focus-on-fuzzing/
    - https://safecode.org/resource-secure-development-practices/managing-security-risks-inherent-in-the-use-of-third-party-components/
    - https://safecode.org/resource-secure-development-practices/guidance-for-agile-practitioners/
    - https://safecode.org/resource-publications/threat-modeling-at-scale/
    - https://safecode.org/resource-secure-development-practices/tactical-threat-modeling/
- UK DSIT: Software Security Code of Practice, May 2025.
- US Department of Homeland Security's (DHS) Cybersecurity and Infrastructure Security Agency (CISA) Secure by Design (SbD) initiative

Appendix B

In Depth Evolution of Secure by Design

Section 2 of this paper noted 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[36] 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 eminent NSA computer scientist Bernard Peters[37] — and especially the diagram below on networked computer system vulnerabilities and associated risks — became the seminal and defining point for modern cybersecurity.



The two 1967 Atlantic City conference papers[38] 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*,[39] 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.

36 RAND Memorandum RM-3765-PR (August 1964): "On Distributed Communications - IX. Security Secrecy and Tamper-Free Considerations", Paul Baran,. https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM3767.pdf

37 Willis Ware, "Security and privacy in computer systems", AFIPS Proceedings, 1967 Spring Joint Computer Conference, Vol. 30 at p. 279, https://dl.acm.org/doi/pdf/10.1145/1465482.1465524

38 ibid

39 US DOD 1970 report, *Security Controls for Computer Systems*

Additional Chronology of Secure by Design

- **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 DoD and the National 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.

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 DoD Computer Security Center at NSA. 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 (bitsavers.computerhistory.org).
- **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 defense information from unauthorized disclosure.

TCSEC in Practice: Impacts & Limitations

- **Initial Success & Funding Incentives:** The Computer Security Center sent vendors the message that Orange Book compliance would be 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 to 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.

Stagnation and Critiques

- **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 (netlib.org+3stevelipner.org+3safecode.org+3).
- **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 (researchgate.net+2netlib.org+2safecode.org+2).

New Approaches

- **Expansion to New Evaluation Schemes:** In the 1990s and 2000s, other frameworks emerged — e.g., ITSEC (EU). 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. The Common Criteria replaced the Orange Book in the United States and has been adopted by more than 20 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."[40]

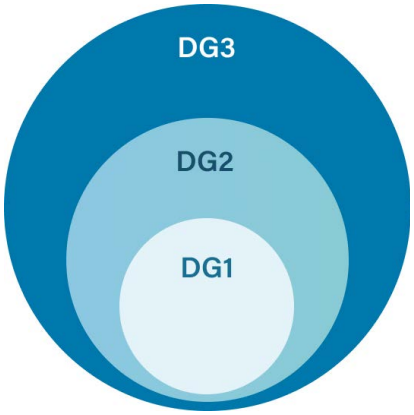
40 NSA: "The 60 Minute Network Security Guide (First Steps Towards a Secure Network Environment", 16 Oct 2001.

Appendix C

Development Groups

Development Group 1

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 CIS Controls.



Development Group 2

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

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.

Appendix D

Attack Classes

The classes of attacks are usually referred to by the mnemonic acronym STRIDE.

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

Appendix E

Tool Classes and References

Tool Classes

- Static analysis tools detect coding errors (especially memory safety errors).
- Configuration security tools 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.
- Dynamic analysis tools (e.g., fuzzing tools) test components for dangerous responses to unexpected inputs.
- Threat modeling tools are essential for identifying and mitigating design-level security risks during software development.
- Proprietary and open-source tools are tools software development organizations can use to evaluate a product's attack surface.

Links to References or Curated Lists of Tools

- Wikipedia contributors. 2025. "List of Tools for Static Code Analysis." *Wikipedia, The Free. Encyclopedia.* https://en.wikipedia.org/w/index.php?title=List_of_tools_for_static_code_analysis&oldid=1285981928
- https://www.cisecurity.org/insights/podcast/cis-podcast-episode-26-automating-the-secure-configuration-management-process
- Breeden, J. 2022. "10 Top Fuzzing Tools: Finding the Weirdest Application Errors." CSO Online. https://www.csoonline.com/article/568135/9-top-fuzzing-tools-finding-the-weirdest-application-errors.html
- https://en.wikipedia.org/wiki/Threat_model
- Source Code Security Analyzers | NIST
- GitHub — analysis-tools-dev/static-analysis: A curated list of static analysis (SAST) tools and linters for all programming languages, config files, build tools, and more. The focus is on tools which improve code quality.
- Combinatorial Methods for Trust and Assurance | CSRC
- GitHub — google/oss-fuzz: OSS-Fuzz — continuous fuzzing for open source software.

Appendix F

Roles

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 teams 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.

We also provide a mapping of just published European Union Agency for Cybersecurity (ENISA) roles[41] mapped to the SbD roles from this paper. While the ENISA roles are in line with traditional cybersecurity roles and hence broader than SbD roles, there is sufficient overlap to warrant a mapping in this paper.

ENISA Role	ENISA Main Responsibility (and references within ENISA document)	SbD Role(s)
Chief Information Security Officer (CISO)	Manages an essential or important entity's cybersecurity strategy and its implementation to align with the Directive's requirements (Articles 20 and 21).	CISO Team; Corporate Leadership
Cyber Incident Responder	Ensures an effective response to cybersecurity incidents and comprehensive reporting of these incidents (Articles 21 and 23)	Security Response Engineer
Cyber Legal, Policy, and Compliance Officer	Oversees and assures compliance with the Directive (Articles 1, 2, 3, 4, 20, 26, 27, 28, and 29)	CISO Team; SDL Team
Cyber Threat Intelligence Specialist	Collects, processes and analyses data and information to identify significant cyber threats (Articles 21, 23, 29, and 30).	SDL Team (to inform process updating)
Cybersecurity Architect	Plans and designs security-by-design solutions to align with the cybersecurity risk measures of the Directive (Article 21).	Architect
Cybersecurity Auditor	Performs cybersecurity audits to assess alignment with the Directive and uncover areas for enhancement. (Articles 20, 21, 23, and 32).	SDL Team
Cybersecurity Educator	Creates and delivers cybersecurity awareness and training programmers for members of management bodies and employees to enhance their understanding and competencies in line with the Directive's requirements (Articles 20 and 21).	SDL Team
Cybersecurity Implementer	Implements cybersecurity solutions and controls based on the cybersecurity risk-management measures of the Directive (Articles 21, 32, and 33).	CISO Team; Software Engineer; Test Engineer; Program Manager
Cybersecurity Researcher	Researches the cybersecurity domain and incorporates results in cybersecurity solutions based on the cybersecurity risk-management measures of the Directive (Article 21).	Architect; Program Manager; SDL Team

41 *Cybersecurity Roles and Skills for NIS2 Essential and Important Entities (Mapping of NIS2 obligations to ECSF)* June 2025

ENISA Role	ENISA Main Responsibility (and references within ENISA document)	SbD Role(s)
Cybersecurity Risk Manager	Manages the entity's cybersecurity-related risks aligned to its strategy and the Directive's requirements (Article 21).	CISO Team; SDL Team; Program Manager *(Note that risk management in the sense of risk acceptance is a role of Corporate Leadership and Engineering Group Leadership.)*
Digital Forensic Investigator	Investigates cybersecurity incidents and presents digital evidence to support the entity in meeting the reporting obligations of the Directive (Article 23).	Not in scope of the SbD paper although root cause analysis is the responsibility of the SDL Team, Architect, Program Manager, and Software Engineer.
Penetration Tester	Conduct penetration testing to assess the effectiveness of cybersecurity solutions against the cybersecurity risk management measures of the Directive (Articles 21, 32, and 33).	Test Engineer; SDL Team

Appendix G

A Measurement Aid to Secure by Design: A Guide to Assessing Software Security Practices

The spreadsheet 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)?" The perspective of this guide is that there are a set of fundamental security best practices that all software development organizations must adhere to. The NIST SSDF embodies those best practices in a way that is general enough to be applied widely. Additionally, we augmented the SSDF best practices with mappings to the CIS Controls and guidance from SAFECode's "Application Software Security and the CIS Controls." The use of these two additional resources resulted in specific activities software development organizations can implement, depending on maturity or Development Group (DG), artifacts to be provided to show conformance to those activities, and roles within an organization for additional guidance on who should perform the activities.

Download the spreadsheet.

Appendix H

Secure by Design Checklist

SSDF Practices	Tasks	Artifact	Completed
Define Security, and 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.	**DG1:** Documents to show implementation of CIS 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. **DG2/3:** 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.	**DG1/2/3** ▪ 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]	**DG1/2/3** ▪ Contracts, RFPs etc., with security requirements.	

SSDF Practices	Tasks	Artifact	Completed
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.	**DG1/2/3** ▪ 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.	**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.	
	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.	**DG1** ▪ Emails showing executive and/or management commitment to a secure development program. **DG2/3:** Examples of artifacts 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, and 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 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.	**DG1/2/3** ▪ List of tools and/or tool types used in secure software development. ▪ Guidance for developing prompts for artificial intelligence coding tools, if used.	
	PO.3.2 ▪ Follow recommended security practices to deploy, operate, and maintain tools and toolchains.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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	Tasks	Artifact	Completed
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.	**DG1/2/3** ▪ Configuration of bug tracking system.	
	PO.4.2 ▪ Implement processes, mechanisms, etc. to gather and safeguard the necessary information in support of the criteria.	**DG1/2/3** ▪ 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 development include development, build, test, and distribution environments.	**PO.5.1** ▪ Separate and protect each environment involved in software development.	**DG1:** Documents to show implementation of CIS 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, and Incident Response Management. ▪ Configurations to show implementation of policies/processes and safeguards. **DG2/3:** Documents to show implementation of IG2/3 safeguards to include: ▪ Policies/Process for: Network Infrastructure Management, Service Provider Management, Application Software Security, and 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) to perform development-related tasks using a risk-based approach.	**DG1/2/3** ▪ Documentation describing the organization's secure configuration process. ▪ Configurations to show implementation of the secure configuration process and safeguards in Control 4.	

SSDF Practices	Tasks	Artifact	Completed
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.	**DG1/2/3** ▪ Documentation describing access control management process. ▪ Configurations to show implementation of the access management process and safeguards in CIS Control 6.	
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.	**DG1/2/3** ▪ Signed code.	
Archive and Protect Each Software Release (PS.3) Preserve software releases in order to help identify, analyze, 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.	**DG1/2/3** ▪ 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]).	**DG1** ▪ List of approved third-party components. ▪ SBOM for each product. **DG2/3** ▪ List of SCA tools and tool outputs.	

SSDF Practices	Tasks	Artifact	Completed
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 modeling — such as threat modeling, attack modeling, or attack surface mapping — to help assess the security risk for the software.	**DG3** ▪ Documented threat model and tools used to support threat modeling. ▪ DFD describing system and potential attacks. ▪ Output of workflow bug system showing mitigations of medium and high threats.	
	PW.1.2 ▪ Track and maintain the software's security requirements, risks, and design decisions.	**DG1/2/3** ▪ 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]	**DG1/2/3** ▪ Product specification, work items in the 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.	**DG1/2/3** ▪ Design specifications or other design documents; design security review minutes; bugs for design issues in the workflow system.	

SSDF Practices	Tasks	Artifact	Completed
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.	**DG1/2/3** ▪ 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. ▪ Relevant software design and implementation documents. ▪ Prompts for artificial intelligence tools, if used. **DG 2/3** ▪ 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.	
	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 lifecycles.	**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.	

SSDF Practices	Tasks	Artifact	Completed
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. ▪ Documentation describing access control management process. ▪ Relevant software design and implementation documents. **DG 2/3** ▪ 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.	
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.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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.	

SSDF Practices	Tasks	Artifact	Completed
Review and/or Analyze 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.	**DG1/2/3** ▪ 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.	
	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.	**DG1/2/3** ▪ 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.	
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.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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	Tasks	Artifact	Completed
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.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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, analyze, and/or test the software's code to identify or confirm the presence of previously undetected vulnerabilities.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ Documented vulnerability handling policy.	
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** ▪ Analyze each vulnerability to gather sufficient information about risk to plan its remediation or other risk response.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ Entry in the workflow system and the fixed code. ▪ Entries in NVD, when appropriate.	

SSDF Practices	Tasks	Artifact	Completed
Analyze Vulnerabilities to Identify Their Root Causes (RV.3) Help reduce the frequency of vulnerabilities in the future.	**RV.3.1** ▪ Analyze identified vulnerabilities to determine their root causes.	**DG1/2/3** ▪ 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 ▪ Analyze the root causes over time to identify patterns, such as a particular secure coding practice not being followed consistently.	**DG1/2/3** ▪ 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.	**DG1/2/3** ▪ 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.	**DG3** ▪ Updates to training addressing problems found in root cause analysis to eliminate classes of vulnerabilities.	

The Center for Internet Security, Inc. (CIS®) makes the connected world a safer place for people, businesses, and governments through our core competencies of collaboration and innovation. We are a community-driven nonprofit, responsible for the CIS Critical Security Controls® and CIS Benchmarks®, globally recognized best practices for securing IT systems and data. We lead a global community of IT professionals to continuously evolve these standards and provide products and services to proactively safeguard against emerging threats. Our CIS Hardened Images® provide secure, on-demand, scalable computing environments in the cloud. CIS is home to the Multi-State Information Sharing and Analysis Center® (MS-ISAC®), the trusted resource for cyber threat prevention, protection, response, and recovery for U.S. State, Local, Tribal, and Territorial government entities, and the Elections Infrastructure Information Sharing and Analysis Center® (EI-ISAC®), which supports the rapidly changing cybersecurity needs of U.S. election offices. To learn more, visit http://cisecurity.org or follow us on X: @CISecurity.

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services. For more information, please visit www.safecode.org. Follow SAFECode on LinkedIn and X.

cisecurity.org
email@cisecurity.org
518-266-3460
Center for Internet Security
@CISecurity
CenterforIntSec
cisecurity
TheCISecurity

CIS Center for Internet Security®

CIS Controls®

01010 0100000 010001 01000 0100001 SAFECode