The Software Alliance

BSA

The BSA Framework for Secure Software

A NEW APPROACH TO SECURING THE SOFTWARE LIFECYCLE

www.bsa.org

CONTENTS

I. Executive Summary 1

II. Introduction 2

Defining "Software Security" 4

Framework Basics 5

Framework Purpose 7

Guiding Principles 7

Implementing the Framework for Secure Software 10

III. BSA Framework for Secure Software 12

IV. References 31

Definitions 31

Acronyms 32

Sources 33

I. Executive Summary

Developments over the last several years have resulted in the dramatic expansion of software-powered capabilities from traditional computers and industrial control systems into diverse personal devices, widely deployed sensors, smart appliances, connected vehicles, robotic systems, and beyond. These innovations are driving the creation of a new, connected digital economy and can yield tremendous economic and social benefits. Yet, because these technologies also have the potential to create economic, legal, and even physical risk, software developers must have the joint goals of building software securely and ensuring that it can be securely maintained throughout its lifecycle.

Software development organizations, their customers, and policymakers are increasingly seeking ways of assessing and encouraging security across the software lifecycle. While standards and guidelines exist to aid and inform developers in achieving these goals, there is no consolidated framework that brings together best practices in a manner that can be effectively measured, regardless of the development environment or the purpose of the software. BSA | The Software Alliance has developed The BSA Framework for Secure Software (the "Framework") to fill that gap.

Specifically, the Framework is intended to be used to help software development organizations:

(1) describe the current state of software security in individual software products;

(2) describe the target state of software security in individual software products;

(3) identify and prioritize opportunities for improvement in development and lifecycle management processes;

(4) assess progress toward the target state; and

(5) communicate among internal and external stakeholders about software security and security risks.

The Framework is intended to focus on software products (including Software-as-a-Service) by considering both the process by which a software development organization develops and manages software products and the security capabilities of those products. It is intended to complement, rather than replace, guidance for organizational risk management processes. To the greatest extent possible, it seeks alignment with recognized international standards and to remain flexible, adaptable, outcome-focused, and risk-based.

The Framework is intended to become a living document, to be updated and improved based on ongoing feedback from BSA's members and other relevant stakeholders.

II. Introduction

Modern society is built on software. Software powers personal technologies, critical infrastructure, scientific research, and industries across every sector. It drives emerging innovations such as the Internet of Things (IoT), blockchain, and artificial intelligence (AI). As software becomes increasingly central to our lives, making it secure and reliable becomes ever more critical in the face of an evolving and expansive cybersecurity threat landscape.

From within the software community, best practices are emerging that help software developers address important aspects of software security, including security-by-design principles, secure development lifecycle processes, and internationally recognized standards for key security elements such as identity management, encryption, and secure coding. Although attention to each specific security consideration can achieve marginal security gains, effective security requires a comprehensive and risk-informed approach that combines individual considerations into a holistic, lifecycle-long framework. And a comprehensive approach must be tailored to address the nuanced, diverse, and evolving challenges associated with different types of software and connected devices, from the "bare metal" to the most advanced.

Building on best practices pioneered by many of its members, BSA | The Software Alliance has developed a software security framework to bring consistency to these complex challenges. The BSA Framework for Secure Software is intended to establish an approach to software security that is flexible, adaptable, outcome-focused, risk-based, cost-effective, and repeatable. Eschewing a one-size-fits-all solution, this voluntary framework will provide a common organization and structure to capture multiple approaches to software security by identifying standards, guidelines, and practices that can help software development organizations achieve desired security outcomes while accounting for the wide spectrum of intended uses, risk profiles, and technological solutions among software products.

Recent technological developments illustrate the increasing ubiquity of software and the need for a flexible, comprehensive software security framework. Software-powered capabilities are rapidly expanding from desktop computers and industrial systems into nearly every corner of personal lives and business activities, including diverse personal devices, widespread sensors, smart appliances, diverse business applications, connected vehicles, and robots. As these capabilities evolve, software development is growing increasingly diverse and complex.

> ***The BSA Framework for Secure Software is intended to establish an approach to software security that is flexible, adaptable, outcome-focused, risk-based, cost-effective, and repeatable.***

Consider the different ways software is used in several emerging technologies:

Internet of Things

Software is at the core of the IoT, and secure software must be at the core of IoT security. IoT devices, like other computing devices, have many different forms, functions, and levels of complexity. At the low end, some "bare metal" sensors lack even a basic operating system and contain only software code sufficient to perform one or two simple functions. More complex devices may include operating systems, AI algorithms, or the hundreds of millions of lines of code needed to operate many of today's connected vehicles. How can we achieve confidence in the security of software products across this spectrum?

Software-as-a-Service (SaaS)

Many software applications are now being operated as services from a cloud-based architecture in which code is segmented across multiple container environments, updated constantly and in real-time, and accessed via Internet connections rather than installed locally. Some SaaS applications are updated dozens or even hundreds of times each day, with little or no disruption to the user experience. How can we craft a software security framework that accounts for the new technical approaches to software security that SaaS development may demand, while at the same time driving secure outcomes in traditional software development?

Artificial Intelligence

AI also brings new considerations to software development, including new security challenges. AI software often integrates multiple software components, frameworks, and platforms, potentially introducing new risk with each additional element. Moreover, AI generally must ingest and process enormous data sets, introducing risk through the exposure of the data itself. Combined, these risks demonstrate the importance of software security for AI products. Yet, at the same time, AI products are creating promising new approaches to integrating security into software development. How can we address the risks — and harness the benefits — for security in AI software?

These diverse and constantly evolving software development techniques and products demonstrate the need for an outcome-focused approach that can consistently ensure security across a broad array of technical considerations. Additionally, static, inflexible approaches will either disrupt innovation or fail to keep pace with evolving threats because software is constantly changing.

The intent of the Framework is to provide the entire software industry with a comprehensive, adaptable, and relevant framework for software security. By adopting a flexible, outcome-focused approach rooted in industry best practices and international standards, the Framework is structured to be applicable to the entire spectrum of (1) software development organizations and vendors, from the individual entrepreneur to large-scale, multi-national businesses; (2) software development methods, from traditional to DevOps; and (3) software products, from simple IoT sensors to complex AI algorithms.

> ***Software security encompasses what a software development organization does to protect a software product and the associated critical data from vulnerabilities, internal and external threats, critical errors, or misconfigurations that can affect performance or expose data.***

Defining "Software Security"

Software security encompasses what a software development organization does to protect a software product and the associated critical data from vulnerabilities, internal and external threats, critical errors, or misconfigurations that can affect performance or expose data. It comprises both organizational processes and product capabilities.

- **Organizational processes** include governance structures, strategies, guidance, and clearly defined procedures that guide the development of software in a manner that identifies and incorporates security objectives throughout a product's lifecycle, protects the integrity of the development environment, applies resources to incident and vulnerability management, and manages the supply chain that supports the software development project.

- **Product security capabilities** are technical aspects of specific software products that are useful in enabling the products to address common security challenges, such as protecting data, preventing unauthorized access or use, tracking incidents and vulnerabilities, and managing unforeseen events.

Both organizational processes and product security capabilities are vital elements of software security.

Software security is often discussed in relation to *software assurance*. Software assurance has been defined[1] as the "level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner." It has also been defined[2] as "the development and implementation of methods and processes for ensuring that software functions as intended and is free of design defects and implementation flaws." While such definitions may suggest that the level of security associated with a given software product could be ascertained simply by measuring the presence and extent of defects or vulnerabilities in its code base, software security is rarely that straightforward.

One challenge is that — at least currently — it is impractical to expect complex software code to be entirely free of vulnerabilities. Indeed, according to some estimates, software products currently average roughly 1–5 defects per 1,000 lines of code, with many complex software products incorporating tens or hundreds of millions of lines of code in total.[3] While defect-free code should always be a developer's goal, it is not a realistic industry standard. Instead, the goal should be the widespread adoption of practices and processes that minimize code defects, and particularly known software vulnerabilities, and to maintain a proactive security posture oriented to identifying and addressing problems before they can be exploited. In fact, researchers have documented substantial improvements in average software defect density among leading software developers through the implementation of secure development lifecycle approaches and other software security best practices.

A second challenge is that any approach to software security that is distilled into a test or series of tests at a single point in time is inherently flawed. As developers increasingly adopt iterative approaches to development, incorporate third-party components, and face evolving security threats, a software product may change continually and substantially over its lifecycle. Testing methodologies undergo evolution as well; for example, the set of known software vulnerabilities assessed by certain testing methodologies may be frequently updated to include newly discovered flaws. Security is a persistent requirement; while software testing is a critical element of secure development, it is not a stand-

[1] https://www.hsdl.org/?view&did=7447

[2] https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

[3] https://resources.sei.cmu.edu/asset_files/Webinar/2014_018_100_295971.pdf

in for a sustained, security-focused approach to lifecycle management.

Other models exist for informing or assessing software security. Some of these models, including SAFECode's *Fundamental Practices for Secure Software Development*, the Software Assurance Maturity Model, and various secure software development lifecycle methodologies, serve as important starting points for the Framework described in this document. They provide detailed guidance, informed by broad industry best practices, on a wide range of considerations organizations should address to maximize their ability to produce secure software in a verifiable, repeatable, transparent manner. However, in many cases, these guidance documents lack specificity and are primarily targeted toward organizations, focusing almost exclusively on organizational approaches, processes, and methodologies that collectively constitute the *input* of software development. They offer limited guidance on security considerations in relation to the *output* of software development; that is, the software product.

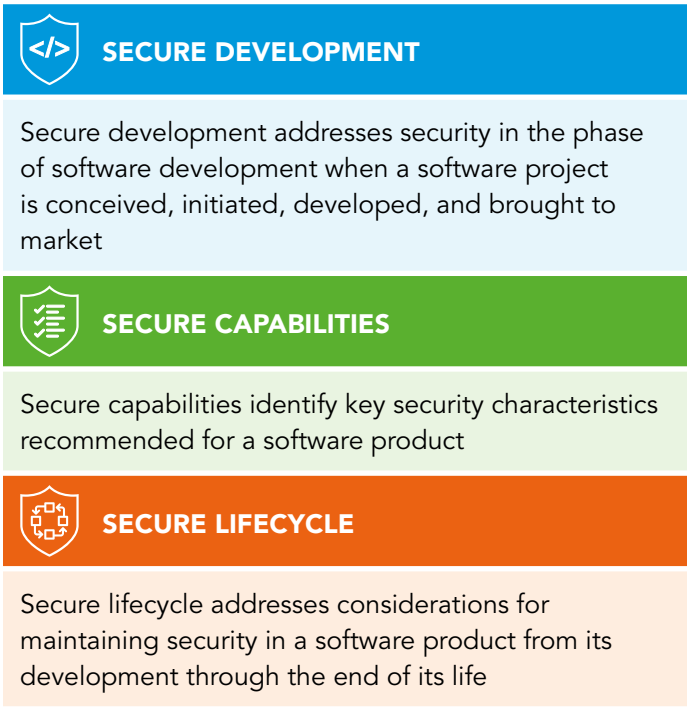
The Framework takes the approach of defining software security by considering both input and output; that is, it includes considerations of organizational processes that guide how vendors approach the development and maintenance of a software product as well as security capabilities and considerations relevant to the product itself. Moreover, it provides this guidance at a level of detail that is specific enough to be measurable, without compromising the flexibility necessary to ensure that all organizations can tailor the guidance according to the type, use, and associated risk of a software product.

The Framework is intended to apply to all types of software. Yet, because of the tremendous diversity in types of software, software development processes, and risks, some security considerations will be more relevant to certain types of software than others. Moreover, organizations will vary in how they customize approaches to achieving the outcomes described in the Framework. The Framework is intended as a tool to create a common language for discussions about how software approaches security, enabling stakeholders to hone in on the security outcomes most relevant to the circumstances. Rather than serving as a box-checking exercise, such a common language enables organizations to describe how they approach a specific security outcome or why that outcome may not be applicable to their product.

Framework Basics

The Framework identifies best practices relating to both organizational processes and product capabilities across the entire software lifecycle. It is organized into six columns: Functions, Categories, Subcategories, Diagnostic Statements, Implementation Notes, and Informative References.

Functions organize fundamental software security activities at their highest level, consistent with the software lifecycle. The Functions are:



Categories divide a Function into distinct considerations and disciplines relevant to the Function. Many Categories are fundamentally interwoven with other Categories; for example, the "Vulnerability Management" and "Vulnerability Notification and Patching" Categories are conceptually closely related, as successful vulnerability management necessarily involves vulnerability notification and patching. However, the Categories seek to distill best practices into distinct subjects or disciplines; in this example, "Vulnerability Management" provides guidance for organizational processes to identify, prioritize, and mitigate vulnerabilities, whereas "Vulnerability Notification and Patching" identifies best practices for developing and issuing patches, mitigations, and notifications to customers. Categories within the same Function may involve different communities of

> ***By "software development organizations," the Framework intends to address all parts of an organization involved in the design, development, deployment, and maintenance of software, recognizing that each organization must determine how it can assign roles and responsibilities to most effectively achieve desired security outcomes.***

practices within the software development organization; for example, "Secure Coding" practices will may be most relevant to a different part of a software development team than those members responsible for "Supply Chain Risk Management" practices.

Subcategories further divide a Category into distinct, unitary concepts that express identified software security best practices.

Diagnostic Statements identify specific, verifiable outcomes. They provide a set of results that help support achievement of the outcomes in each Category. Diagnostic Statements are not intended as an exhaustive list of best practices, but as a set of desired outcomes that are universally relevant, to the maximum extent possible, to enhancing security across all classes and types of software. The Framework does not intend that every Diagnostic Statement will apply to every development environment or software product. Instead, through an examination of risk, software development organizations will apply the Diagnostic Statements appropriate for their environment and product, and identify cases in which Diagnostic Statements are inapplicable or irrelevant. This approach is consistent with other risk-based frameworks that seek to encourage and guide secure activities while avoiding becoming simple checklists.

Implementation Notes provide additional information, where necessary, such as examples of how organizations may achieve security outcomes described in the Diagnostic Statements, interpretations of how Diagnostic Statements may apply in different development environments, and guidance on aligning implementation with risk.

Informative References are additional resources that identify and describe best practices, guidelines, or further information for the implementation of an associated Diagnostic Statement. They may describe methods for achieving the described outcome, provide technical specifications or related best practices, and offer further clarity and specificity on the security benefits of the described outcome. Informative References include internationally recognized technical standards, best practice manuals and guidelines, and references to Common Weakness Enumerators (CWEs). A current list of CWEs is maintained at https://cwe.mitre.org/. In some cases, multiple standards may offer alternative approaches to achieve similar outcomes. Similarly, CWE references are drawn from a community-developed taxonomy of software weaknesses that serves as a common language for describing weaknesses and provides a baseline for identification, mitigation, and prevention of such weaknesses. Numerous CWE references may be related in some form to a specific Diagnostic Statement; the Framework attempts to identify the most relevant weaknesses resulting when the Diagnostic Statement is incompletely or improperly addressed. In all cases, Informative References are illustrative and are not intended to be either exhaustive or prescriptive.

The Framework's Subcategories and Diagnostic Statements are often focused on the individuals and team that actually develop software. In practice, entities developing software are complex organizations that often include separate software development teams that interact with security teams, corporate governance structures, and external requirements, each of which play key roles in driving the security outcomes the Framework describes. By "software development organizations," the Framework intends to address all parts of an organization involved in the design, development, deployment, and maintenance of software, recognizing that each organization must determine how it can assign roles and responsibilities to most effectively achieve desired security outcomes.

Framework Purpose

The Framework is intended to be used to help software development organizations:

1. Describe the current state of software security in individual software products.
2. Describe the target state of software security in individual software products.
3. Identify and prioritize opportunities for improvement in development and lifecycle management processes.
4. Assess progress toward the target state.
5. Communicate among internal and external stakeholders about software security and security risks.

The Framework is intended to focus on software products (including Software-as-a-Service), by considering both the process by which a software development organization develops and manages software products and the security capabilities of products. It is intended to complement, rather than replace, guidance for organizational risk management processes. To the greatest extent possible, it seeks alignment with recognized international standards.

The Framework is intended to become a living document, to be updated and improved based on ongoing feedback from BSA's members and other relevant stakeholders.

Guiding Principles

The Framework is based on five key principles:

- Risk-based
- Outcome-focused
- Flexible
- Adaptable
- Aligned with Internationally Recognized Standards

Risk-Based.

Software is enormously diverse, ranging from applications that perform only a few basic functions to highly sophisticated AI programs, and it is used in an enormously diverse array of contexts, from home computing networks to the very backbone of the Internet. The different types and uses of software carry different risks; for example, the software behind a mobile phone game may pose far less threat to cyber or physical security than the software operating an electricity grid's control system.

To manage the risks associated with software, organizations should build software development processes around careful analysis of the risks associated with their products, the potential resulting impacts, and their organization's risk tolerance. With an understanding of risk tolerance, organizations can prioritize security activities in their software development and lifecycle management processes, enabling informed decisions about where to prioritize improvements and how to align financial and human resources.

> ***Many elements of the Framework are intentionally structured to provide software development organizations with the flexibility to tailor their approaches based on the risk profile of the product.***

Risk informs the Framework throughout its three functions and is intended to guide software development organizations and vendors to address security considerations in operational processes and product security capabilities according to the level of risk associated with the product.

For example, consider the first Subcategory articulated in the Framework which reads: "Threat modeling and risk analysis are employed during software design to identify threats and potential mitigations." This risk analysis is designed to guide software development organizations toward adopting the security controls most appropriate to the type and uses of their products. Understanding of the risk subsequently informs the development of a plan to address security considerations in the software's development and deployment.

Outcome-Focused.

The Framework communicates best practices in their most detailed form through Diagnostic Statements that identify specific, measurable outcomes. These statements are intended to be neutral with respect to coding language, development process, and technical approach. Rather than dictating specific security techniques, the Framework focuses on the outcomes software development organizations and vendors ideally should achieve to enhance the security profile of the software.

Flexible.

Software development as a discipline is constantly evolving based on innovations in efficiency and management, emerging customer demands, new approaches to coding languages or software development tools, and technical breakthroughs. Moreover, cybersecurity requires constant innovation to keep pace with changing threats. Any approach to software security must be flexible enough to enable software developers to develop new approaches to new challenges, and to deliver innovative products to the customers who depend on them.

The Framework approaches this vital principle by ensuring that it specifies outcomes that are neutral with regard to coding language, development process, and technical approach. Similarly, the Framework recognizes that some Diagnostic Statements may be more important to some organizations than others. For example, companies securing SaaS products will find statements relating to securing containers, such as TC.1-6, more applicable to their software development environment than businesses providing mostly out-of-the-box software. Likewise, organizations developing out-of-the-box software may find Diagnostic Statements relating to anti-tamper techniques, like SM.4-1, more useful. The Framework is structured in a way such that each Diagnostic Statement is intended to maintain flexibility while remaining applicable to software of all types, languages, and development processes.

Many elements of the Framework are intentionally structured to provide software development organizations with the flexibility to tailor their approaches based on the risk profile of the product. For example, the "Support for Identity Management and Authentication (SI)" category recognizes that not all software products will require an identity management and authentication mechanism but includes clear guidelines for those that do. It directs that software "avoids hard-coded passwords" and "avoids authentication mechanisms that allow insufficiently complex passwords, insufficient password aging management, unlimited log-on attempts, commonly used password topologies, or unverified password changes." For some software products, these guidelines will mean adopting strong identity management and authentication mechanisms, such as multi-factor authentication, single sign-on technologies, and log-on limits. For others, they will mean ensuring that third-party identity management and authentication tools meet those guidelines before they are incorporated. For still others, they will mean

EXAMPLE

Preventing SQL Injection Attacks.

Hackers may use SQL injection — a code injection technique in which malicious SQL statements are inserted into an entry field for execution — to compromise the confidentiality, integrity, and/or availability of data used in a software program. SQL injection attacks are particularly common in database-driven applications and are among the common types of malicious cyber activity.

Concatenation of untrusted data with string constants (string concatenation, or the combining of multiple strings of untrusted data into a single string) is a common and dangerous weakness that SQL injection attacks can take advantage of. To mitigate the risk of SQL injection attacks, the Framework includes the following diagnostic statements in the **Secure Coding** category of the **Secure Development** function:

> **SC.3-1. Software avoids, or includes documented mitigations for, known security vulnerabilities in included functions and libraries.**
>
> **SC.3-2. Software development organizations validate input and output to mitigate common vulnerabilities in software.**

By focusing on secure outcomes, the Framework avoids mandating specific technical approaches to structuring SQL statements, such as prescribing certain stored procedures or whitelisting techniques. SQL statements can be created and parameterized using many different programming languages, libraries, and frameworks; the Framework establishes clear security outcomes that are targeted and meaningful but retains the flexibility to enable its achievement through each of these differing languages, libraries, and frameworks. In each case, the outcome specified in the diagnostic statement is linked to references to informative material that provides further detail on achieving the outcome, including references specifying techniques to prevent SQL injection attacks.

Not all software products are at risk of SQL injection attacks, and not all software products utilize dynamic SQL statements. The security outcomes specified by the Framework are met equally by the software product that develops properly parameterized SQL statements as by the software product that excludes dynamic SQL statements altogether. The appropriate approach to meeting the specified security outcome will be based on a risk-informed software design and security architecture.

validating that such measures are not needed based on the product's risk and architecture.

Adaptable.

In today's development context, software is constantly changing. Many products are continually updated with new features and additional security measures long after their original market deployment. For that reason, software security must be conceptualized in a way that is adaptable to this lifecycle, as well as to the constant innovation of new technologies, processes, and standards in the software industry. For that reason, approaches to software security that mandate specific technical measures or that endeavor to subject software products to batteries of tests that assess security at a single point in time will fail to keep pace with the constant evolution of software. Instead, this Framework provides a tool to assess the characteristics of software security throughout a software product's lifecycle, using outcome-focused diagnostic statements that are adaptable to diverse and evolving technical approaches.

EXAMPLE

Vulnerability Advisories to SaaS Customers.

To ensure that users are properly informed of relevant security information associated with software updates, the **Vulnerability Notification and Patching** category of the **Secure Lifecyle** function includes the following diagnostic statement:

> **VN.3-1. Users are notified of a significant security issue when a remediation is in place for each supported version of the affected product.**

As important as such notifications can be when users are asked to install updates that could potentially have broader impacts to their own devices or systems, it may not be feasible for notifications to accompany every software update in some contexts. For example, many SaaS vendors operate in a continuous delivery environment, meaning software is produced in short cycles of testing, staging, pre-production, and production. Because SaaS is a web-based model in which software is maintained on remote servers rather than installed on user devices, SaaS software updates are also generally not installed on user devices. Continuous integration and continuous delivery methodologies make it possible to quickly deploy new versions of, or security updates to, a SaaS application without customer disruptions or losses of service. Sophisticated SaaS vendors may deploy dozens, or even hundreds, of software updates to an application each day.

By focusing on information relevant to *significant* security issues, the Framework avoids onerous notification requirements, which may be impossible to meet in a SaaS environment, while ensuring customers are well-informed regarding the security of their products and services.

Aligned with Internationally Recognized Standards.

Internationally recognized technical standards provide widely vetted, consensus-based information and guidance for defining and implementing effective approaches to cybersecurity and facilitate common approaches to common challenges, thus enabling collaboration and interoperability. Industry leaders have developed a range of international standards and best practices for secure-by-design software development. To ensure international interoperability and express consensus best practices, the Framework seeks to align, to the greatest extent possible, with internationally recognized technical standards wherever they exist. Currently, the most notable example relevant to secure software development is the ISO/IEC 27034 series of standards, which sets out guidance on "integrating security seamlessly throughout the lifecycle" of software applications.

Implementing the Framework for Secure Software

The Framework is designed to support the systematic processes used by software development organizations to identify, assess, and minimize cybersecurity risk throughout the lifecycle of software products. Using the Framework as a cybersecurity risk management tool, an organization can establish a holistic secure development lifecycle that identifies likely risks, enables conscientious decisions about risk mitigation and risk tolerance, improves software quality, and prepares the

> ***Using the Framework as a cybersecurity risk management tool, an organization can establish a holistic secure development lifecycle that identifies likely risks, enables conscientious decisions about risk mitigation and risk tolerance, improves software quality, and prepares the organization to address emerging security considerations throughout the software's lifecycle.***

organization to address emerging security considerations throughout the software's lifecycle. Specifically, software development organizations may find the Framework to be a useful tool for the following purposes, among others:

- **Development process guidance.** A software development organization should publish definitive direction on the policies and processes that development of a new software product is expected to follow in order to ensure that all involved stakeholders understand roles, responsibilities, and expectations. Organizations may choose to amend software development processes and process guidance to ensure the elements of the Framework are accounted for throughout the product development lifecycle.
- **Training and awareness.** A software development organization may consider developing internal training and education programs to build a culture of security and to ensure that stakeholders are trained in responsibilities and methodologies appropriate to their roles in the software development lifecycle. Organizations may choose to incorporate elements of the Framework into internal training and awareness modules. In addition, the Framework may provide a useful tool for educating executives about how security is addressed in the development process, how resources are aligned to security considerations, and how individual products incorporate cybersecurity.
- **Tracking and assessment.** Software development organizations may wish to use the Framework as a tool to track a product as it is developed or to assess its security profile according to concrete metrics. For example, software development lifecycles often establish release gates that require a project to meet an established measure or obtain a waiver before advancing; elements of the Framework may be incorporated into release gate criteria. Additionally, the Framework may help an organization identify metrics that define and measure software security for its products.
- **Vendor relations.** A software development organization should implement measures to ensure the integrity of its supply chain. Organizations may choose to use the Framework to guide purchasing decisions and/or the development of vendor contracts that ensure third-party software components will not jeopardize the organization's security objectives and compliance requirements.
- **Public security narrative.** Software development organizations may wish to communicate information about a product's security features and its approach to mitigating cybersecurity risk to a public audience. The Framework may be useful in enabling organizations to build a narrative about their secure development lifecycle and product security.

III. BSA Framework for Secure Software

The Framework does not intend that every Diagnostic Statement will apply to every development environment or software product. Software development organizations will identify and apply the Diagnostic Statements appropriate for their environment and product based on analysis of risk.

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Secure Coding (SC)	**SC.1.** Threat modeling and risk analysis are employed during software design to identify threats and potential mitigations.	**SC.1-1.** Software development organizations document likely threats.	Threat modeling attempts to identify and prioritize the potential threats against a software product or component in order to guide software development decisions that defend against identified threats. Some software developers work in accordance with "zero trust" principles, which assume a pervasively hostile environment. Yet, even with zero trust approaches, threat modeling is important for identifying sensitive data and prioritizing threats for mitigation. Developers should consider the risk profile of the product when determining the level of detail to provide in such documentation.	ISO/IEC 27034; OWASP Application Security Verification Standard; SAFECode "Fundamental Practices"; SAFECode "Tactical Threat Modeling"; SAMM; BSIMM; CWSS; CAPEC; OWASP Threat Modeling Cheat Sheet
		SC.1-2. Threats are rated and prioritized according to risk.		ISO/IEC 27034; SAFECode "Fundamental Practices"; SAMM; CWSS; CAPEC; OWASP Threat Modeling Cheat Sheet
		SC.1-3. Software development organizations apply common threat modeling methodologies.		ISO/IEC 27034; SAFECode "Fundamental Practices"; SAMM; CWSS; CAPEC; OWASP Threat Modeling Cheat Sheet; SAFECode "Tactical Threat Modeling"
		SC.1-4. Compensating controls are identified and mapped to threats.		ISO/IEC 27034; SAFECode "Fundamental Practices"; SAMM; CWSS; CAPEC; OWASP Threat Modeling Cheat Sheet

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Secure Coding (SC) *(continued)*	**SC.2.** Software is developed according to recognized, enforceable coding standards.	**SC.2-1.** Standards are formally identified and documented.		ISO/IEC TS 17961; SEI CERT C Coding Standard; SEI CERT C++ Coding Standard; SEI CERT Java Coding Standard; NCSC
		SC.2-2. Software uses canonical data formats.		SAFECode "Fundamental Practices"; CWE-21; CWE-22; CWE-35; CWE-36; CWE-37; CWE-38; CWE-39; CWE-40
	SC.3. The software is secure against known vulnerabilities, unsafe functions, and unsafe libraries.	**SC.3-1.** Software avoids, or includes documented mitigations for, known security vulnerabilities in included functions and libraries.	Software should avoid known vulnerabilities to the greatest extent possible. In some instances, there may be reasons for software to incorporate functions or libraries known to include vulnerabilities; such functions or libraries should only be incorporated when developers include documented mitigations that ensure the vulnerabilities are not exploitable.	NIST NVD; CWE/SANS Top 25 Most Dangerous Software Errors; OWASP Top 10; CWE-1006; CWE-242
		SC.3-2. Software validates input and output to mitigate common vulnerabilities in software.		SAFECode "Fundamental Practices"; OWASP Input Validation Cheat Sheet; CWE-20; CWE-89; CWE-119; CWE-120; CWE-183; CWE-184; CWE-242; CWE-625; CWE-675; CWE-805
		SC.3-3. Software encodes data and/or uses anti-cross site scripting (XSS) libraries.		SAFECode "Fundamental Practices"; CWE-79
	SC.4. Standard software assurance measures are employed in the software architecture and design.	**SC.4-1.** The software employs segmentation through sandboxing, containerization, or similar methodologies.		SAFECode "Fundamental Practices"; CWE-265
		SC.4-2. The software employs fault isolation mechanisms.		DoD-PPP

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Secure Coding (SC) *(continued)*	**SC.4.** Standard software assurance measures are employed in the software architecture and design.	**SC.4-3.** The software employs system element isolation mechanisms.		DoD-PPP; OWASP Application Security Verification Standard
		SC.4-4. Software uses robust integer operations for dynamic memory allocations and array offsets.	Where errors in integer computation cannot result in security-relevant errors, use of robust integer operations may not be necessary.	SAFECode "Fundamental Practices"; CWE-129; CWE-131; CWE-190; CWE-680; CWE-805
Testing and Verification (TV)	**TV.1.** Analysis and validation of the software attack surface is conducted.	**TV.1-1.** Attack surface is identified and mapped.		OWASP Attack Surface Analysis Cheat Sheet, SAMM
		TV.1-2. Analysis is informed by threat model(s) and risk analysis.		SAFECode "Fundamental Practices"; OWASP Attack Surface Analysis Cheat Sheet
	TV.2. Code review using manual and/or automated tools is conducted.	**TV.2-1.** Code review release gates are established to guide software development.	To the extent possible, automated tools should be implemented and integrated with the software development process to ensure rigor and consistency. Manual tools can be substituted in cases where automation isn't feasible.	SAFECode "Fundamental Practices"; BSIMM; SAMM; OWASP Testing Guide; OWASP Code Review Guide
	TV.3. A comprehensive test plan for testing the functionality and security of software is established.	**TV.3-1.** Test plan is based on threat model(s) and risk analysis.		SAFECode "Fundamental Practices"; OWASP Testing Guide
		TV.3-2. The software is tested in a least privilege environment.		SAFECode "Fundamental Practices"
	TV.4. Software security controls are properly tested with appropriate techniques.			ISO/IEC 27034; SAFECode "Fundamental Practices"; SAMM; BSIMM; OWASP Testing Guide
	TV.5. Software is subjected to adversarial security testing techniques.	**TV.5-1.** Software development organizations establish security testing release gates.		SAFECode "Fundamental Practices"; SAMM
		TV.5-2. Software is subjected to penetration testing.		ISO/IEC 27034; SAFECode "Fundamental Practices"; SAMM; BSIMM; OWASP Testing Guide

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Process and Documentation (PD)	**PD.1.** Secure development processes are documented throughout software development.	**PD.1-1.** Security requirements for the software are gathered from stakeholders and documented.	Developers should consider the risk profile of the product when determining the level of detail to provide in such documentation.	SAMM; Microsoft SDL
		PD.1-2. Security guidance for the development of the software is documented.		SAMM; Microsoft SDL
		PD.1-3. Security guidance for the development of software is updated to reflect the results of root cause analyses of new vulnerabilities.		SAFECode "Fundamental Practices"; BSIMM
		PD.1-4. Security documentation outlining best practices for software use by end-users and developers is made available electronically.		Microsoft SDL
		PD.1-5. Testing and validation activities, including results, are documented.		SAFECode "Fundamental Practices"; NIST IR 7622
		PD.1-6. Software development organizations maintain an up-to-date product history that documents changes to elements and configurations.	Depending on the development process, software developers may opt to maintain changelogs or change histories manually, or use automated tools such as project management software, source code management tools, and configuration management tools. It is increasingly recognized as a best practice for software developers to use automated tools that are capable of tracking the origin of code (date, time, rationale, responsible individual) on a line-by-line basis. Developers should consider the risk profile of the product when determining the level of detail to provide in such documentation.	

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Process and Documentation (PD)	**PD.2.** Software development personnel are accountable for software security.	**PD.2-1.** A security advisor is assigned to the software development team.		Microsoft SDL
		PD.2-2. Software development personnel are trained on identified coding standards and role-specific best practices.		BSIMM; SAMM
Supply Chain (SM)	**SM.1.** Software development is informed by supply chain risk management.	**SM.1-1.** An organizational supply chain management plan and processes for identification and reporting of supply chain incidents are established.		NIST IR 7622; NIST SP 800-53
	SM.2. Approved acquisition measures are in place to ensure the visibility, traceability, and security of third-party components.	**SM.2-1.** Information about providers of third-party components is identified and collected.	Relevant information may include the provider's processes for controlling access to software components, product development and testing standards, supply chain risk management practices, development environment, and vulnerability management processes.	SAFECode "Software Supply Chain Integrity Framework"; BSIMM; NIST Interagency Report 7622; NIST SP 800-53; CWE-505; CWE-506; CWE-507; CWE-510; CWE-511
		SM.2-2. Software development organization employs measures to document and, to the extent feasible, trace to their original source all third-party components directly acquired and incorporated into the software by the developer.		SAFECode "Software Supply Chain Integrity Framework"; NIST IR 7622; NIST SP 800-53; CWE-505; CWE-506; CWE-507; CWE-510; CWE-511
		SM.2-3. To the maximum feasible through the use of manual and automated technologies, subcomponents integrated in third-party components are documented, and their lineage and dependencies traced.		SAFECode "Software Supply Chain Integrity Framework"; NIST IR 7622; NIST SP 800-53; CWE-505; CWE-506; CWE-507; CWE-510; CWE-511

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Supply Chain (SM) *(continued)*	**SM.2.** Approved acquisition measures are in place to ensure the visibility, traceability, and security of third-party components.	**SM.2-4.** Security requirements are incorporated into contracts, policies, and standards for vendors supplying software components.		SAMM; BSIMM; NIST IR 7622; NIST SP 800-53
	SM.3. Supply chain data — including information about software elements, design, testing, evaluation, threat assessments, delivery processes, and agreements language — is protected against unauthorized disclosure, access, modification, dissemination, destruction, and use.	**SM.3-1.** Supply chain data is protected at rest.		NIST IR 7622
		SM.3-2. Supply chain data is protected in transit against unauthorized access.		NIST IR 7622
	SM.4. Software incorporates measures to prevent counterfeiting and tampering.	**SM.4-1.** Software includes mechanisms to ensure the integrity of the software, such as code-signing, anti-reverse engineering, or anti-tamper mechanisms.		SAMM; BSIMM; NIST IR 7622; NIST SP 800-53
		SM.4-2. Software includes supplier source certification or authentication indicators and protects those indicators against tampering and counterfeiting.		BSIMM; NIST IR 7622
		SM.4-3. Identification markers unique to the software's specific version are applied to each delivered product.		NIST IR 7622; BSIMM; NIST SP 800-53

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Supply Chain (SM) *(continued)*	**SM.5.** The software is identifiable through clear, discoverable information communicated in a standardized format.	**SM.5-1.** The software includes descriptive information about the software's identity.	Descriptive information should generally include the software's name, creator, version, licensing details and, where possible, information about the software's dependencies.	ISO/IEC 19770-2; SPDX Version 2.1; NIST IR 8060
	SM.6. Deployment procedures ensure that the proper usages of software are established.	**SM.6-1.** The software includes mechanisms to reduce the likelihood that it is installed on unauthorized hardware or by unauthorized users, such as validating code-signing, authentication, or credentialing.		NIST IR 7622
Tool Chain (TC)	**TC.1.** Software is developed using tools configured for security.	**TC.1-1.** Software is developed using up-to-date versions of all tools and platform elements within the development environment.		SAFECode "Fundamental Practices"; Microsoft SDL; OWASP C-Based Tool Chain Hardening Cheat Sheet; CWE-691; CWE-908
		TC.1-2. Development frameworks used in developing software use secure configurations.		NCSC
		TC.1-3. Compilers are configured to prevent common vulnerabilities and weaknesses.		Microsoft SDL; OWASP Development Guide; CWE-1038
		TC.1-4. Compilers are configured to avoid unintentional removal or modification of security-critical code.		Microsoft SDL; OWASP Development Guide; CWE-733; CWE-1038
		TC.1-5. Compilers are configured to automatically add defense code.		Microsoft SDL; OWASP Development Guide; CWE-1038
		TC.1-6. Containers and other virtualization technologies used in deploying the software use secure configurations.		BSIMM

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE DEVELOPMENT				
Identity and Access Management (IA)	**IA.1.** Throughout the supply chain and product lifecycle, the software development environment uniquely identifies and authenticates users and operators.	**IA.1-1.** Strong authentication methods are required for access to the development environment.	Strong authentication is generally understood to describe mechanisms that require authentication factors from at least two of three categories (knowledge, or something a user knows; ownership, or something a user has; and inherence, or something a user is), but may also utilize contextual information (e.g., geolocation or device information) and other factors to confirm a user's identity. Diagnostic Statements in the IA Category address identity and access management in the development environment. See the SI and AA Categories for information regarding security capabilities in software products themselves.	NCSC: NIST SP 800-53; NIST IR 7622
		IA.1-2. User and operator credentials are stored securely and revoked or disabled when no longer needed.		NCSC
	IA.2. Policies to control access to data and processes for all users and operators are developed, documented, and applied throughout the development environment.	**IA.2-1.** Specific access controls for creation, read access, update, deletion, and execution are applied based on clearly identified and approved user and operator roles.		SAMM; DHS/DACS
		IA.2-2. Access controls are set for individual users and operators that provide only the necessary privileges required to perform an assigned task and only for the necessary time required to perform it.		SAMM; DHS/DACS; DoD-PPP
		IA.2-3. Unauthorized changes or deletions to code, development artifacts, and tools are prevented and logged.		OWASP Logging Cheat Sheet; DHS/DACS; NIST IR 7622; CWE-778

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Support for Identity Management and Authentication (SI)	**SI.1.** The software avoids architectural weaknesses that create risk of authentication failure.	**SI.1-1.** The software avoids hard-coded passwords.		ISO/IEC 9798; OWASP Authentication Cheat Sheet; CWE-259; CWE-798
		SI.1-2. Software source code does not contain secrets.	Secrets may include credentials or keys.	
		SI.1-3. Authentication mechanisms used by the software employ typical security techniques and avoid common security weaknesses.	Typical techniques and common weaknesses are rapidly evolving; software development organizations should stay abreast of current best practices. Current common security weaknesses include allowing insufficiently complex passwords, insufficient password aging management, unlimited log-on attempts, commonly used password topologies, and unverified password changes.	ISO/IEC 9798; OWASP Authentication Cheat Sheet; NIST SP 800-63; CWE-521; CWE-262; CWE-263; CWE-620; CWE-308
		SI.1-4. The software does not store sensitive authentication information, which may include passwords or keys, in source code or publicly accessible infrastructure.		NCSC
		SI.1-5. Any passwords or sensitive authentication information stored by the software is stored in accordance with current best practices.	Best practices for password storage are rapidly evolving; software development organizations should stay abreast of current best practices.	OWASP Password Storage Cheat Sheet
	SI.2. The software supports strong identity management and authentication.	**SI.2-1.** The software implements features, configurations, and protocols that establish or support standard, tested authentication services.		ISO/IEC 9798; SAFECode "Fundamental Practices"
		SI.2-2. The software is interoperable with applicable common industry standards for identity management and authentication.		OAuth 2.0; OIDC; SAML 2.0; WS-FED; UAF; U2F; SAFECode "Fundamental Practices"

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Support for Identity Management and Authentication (SI) *(continued)*	**SI.2.** The software supports strong identity management and authentication.	**SI.2-3.** Authentication controls fail securely.	When authentication controls fail securely, they prevent access by unauthenticated users even after encountering an error.	OWASP Secure Coding Practices
Patchability (PA)	**PA.1.** Software is capable of receiving secure updates and security patches.	**PA.1-1.** Software is capable of validating the integrity of a transmitted patch or update.	The Patchability category refers to technical aspects relating to the ability of the software to receive secure updates and patches. Activities of software developers relating to the development and dissemination of updates and patches are discussed in the Secure Lifecycle function.	NTIA "Voluntary Framework for Enhancing Update Process Security"; NIST SP 800-147; CWE-924
		PA.1-2. Software includes a mechanism to notify end users of patch or update installation.		NTIA "Voluntary Framework for Enhancing Update Process Security"
		PA.1-3. Software reverts to a known-good state upon failed installation of updates or security patches.		NTIA "Voluntary Framework for Enhancing Update Process Security"
Encryption (EN)	**EN.1.** Software is developed in accordance with an encryption strategy that defines what data should be encrypted and which encryption mechanisms should be used.	**EN.1-1.** Software enables the use of encryption to protect sensitive data from unauthorized disclosure.		SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-311
		EN.1-2. Software enables the use of encryption to protect the software itself from tampering.		
		EN.1-3. Software does not expose sensitive data upon failure of encryption mechanisms.		OWASP Secure Coding Practices; CWE-636; FIPS 140-2

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Encryption (EN) *(continued)*	**EN.2.** Software avoids weak encryption.	**EN.2-1.** Software avoids custom encryption algorithms and implementations.	In unique circumstances when a developer identifies a need to use a custom algorithm or implementation, the developer should establish and document a robust procedure to validate the security of the custom algorithm or implementation prior to deployment.	ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-325; CWE-326; CWE-327
		EN.2-2. Software enables the use of authenticated encryption.		ISO/IEC 19772; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-326; CWE-327
		EN.2-3. Encryption employed by the software enables strong algorithms.	Standards for strong algorithms change over time; in general, strong algorithms will have no structural weaknesses, will maintain key sizes of sufficient length to defeat brute force attacks, and will have been standardized and deployed across a reasonably sized user base.	ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-326; CWE-327; CWE-330; CWE-331; CWE-338
		EN.2-4. Encryption employed by the software enables strong key lengths.	Standards for strong key lengths will change over time based on advancements in computing power and factoring techniques; in general, strong key lengths are of sufficient length to ensure brute force attacks are infeasible.	ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-326; CWE-327; CWE-330; CWE-331; CWE-338
		EN.2-5. Encryption capabilities employed by the software are configured to select strong cipher modes and exclude weak ciphers by default.		ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-326; CWE-327; CWE-330; CWE-331; CWE-338

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Encryption (EN) *(continued)*	**EN.2.** Software avoids weak encryption.	**EN.2-6.** Software is configured to disable or prevent the use of weak encryption algorithms and key lengths.	It may be necessary for software to support weak encryption algorithms and key lengths for reasons of backward compatibility. Where such support is required, the implementation should be carefully engineered and thoroughly reviewed to ensure that it does not allow an attacker to bypass the default or user selection of strong encryption.	CWE-326; CWE-327; CWE-330; CWE-331; CWE-338
	EN.3. Software protects and validates encryption keys.	**EN.3-1.** Software ensures that cryptographic keys can be securely stored and managed, separate from encrypted data.		ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57
		EN.3-2. Software includes a mechanism to manage key and certificate lifecycles.	Mechanisms for managing key and certificate lifecycles may include use of third-party key management systems.	ISO/IEC 18033-1; ISO/IEC 19790; FIPS 140-2; FIPS 186-4; FIPS 197; FIPS 202; SAFECode "Fundamental Practices"; OWASP Cryptographic Storage Cheat Sheet; NIST SP 800-57; CWE-324
		EN.3-3. Software includes a mechanism to validate certificates.	Not all software uses certificates; however, it is imperative that software that does use certificates is able to validate the authenticity of those certificates. This diagnostic statement should be applied consistent with the encryption strategy described in EN.1.	OWASP Cryptographic Storage Cheat Sheet; CWE-347
Authorization and Access Controls (AA)	**AA.1.** Software design reflects the principle of least privilege.	**AA.1-1.** The software operates using only those privileges or permissions necessary for software to run correctly.		SAFECode "Fundamental Practices"; DoD-PPD; CWE-250; CWE-271; CWE-272; CWE-274
		AA.1-2. Privileges are set in a configuration that is resistant to unauthorized changes.		SAFECode "Fundamental Practices"; DoD-PPD; CWE-250

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Authorization and Access Controls (AA) *(continued)*	**AA.1.** Software design reflects the principle of least privilege.	**AA.1-3.** An authorization strategy that applies authorization policies, access controls, and design principles to classes of data is implemented in the software.		SAFECode "Fundamental Practices"; CWE-285; CWE-862; CWE-863
	AA.2. The software's design supports authorization and access controls.	**AA.2-1.** The software avoids functions that enable unauthorized privilege escalations.		DHS/DACS
		AA.2-2. In the case of failure, the software does not grant access to unauthorized or unauthenticated users.		OWASP Secure Coding Practices
Logging (LO)	**LO.1.** Software implements logging of all critical security incident and event information.	**LO.1-1.** Software differentiates between monitoring logs and auditing logs.	Monitoring logs record data relevant to analyzing usage and performance, troubleshooting, and informing ongoing software development. Auditing logs support analysis of and response to security events.	SAFECode "Fundamental Practices"; CWE-779
		LO.1-2. Software is capable of logging all security-relevant failures, errors, and exceptions.	Software development organizations should determine what information is security-relevant as part of threat-modeling (see SC.1) and risk assessment.	OWASP Secure Coding Practices; OWASP Logging Cheat Sheet; CWE-778; CWE-223
		LO.1-3. Software is capable of logging timestamp and identifying information associated with security incidents and events.		SAFECode "Fundamental Practices"; OWASP Logging Cheat Sheet; CWE-778
	LO.2. Software security incident and event information logging mechanisms are implemented securely.	**LO.2-1.** Access to logs is restricted to authorized individuals.		OWASP Secure Coding Practices; OWASP Logging Cheat Sheet
		LO.2-2. Logging mechanisms include anti-tamper protections.		SAFECode "Fundamental Practices"; OWASP Logging Cheat Sheet

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE CAPABILITIES				
Logging (LO) *(continued)*	**LO.2.** Software security incident and event information logging mechanisms are implemented securely.	**LO.2-3.** Logs do not store sensitive information, such as unnecessary user information, system details, session identifiers, or passwords.		OWASP Secure Coding Practices; OWASP Logging Cheat Sheet; CWE-532
		LO.2-4. Software logging mechanisms employ input validation and output encoding.		OWASP Secure Coding Practices; OWASP Logging Cheat Sheet; CWE-117
Error and Exception Handling (EE)	**EE.1.** Software integrates error and exception handling capabilities.	**EE.1-1.** Software identifies predictable exceptions and errors that could occur during software execution and defines how the software will handle each instance.		DHS/DACS; OWASP Code Review Guide: Error Handling; SAFECode "Fundamental Practices"; CWE-388; CWE-390; CWE-391; CWE-396; CWE-397; CWE-544
		EE.1-2. Software defines how it will handle unpredicted exceptions and errors and safeguards against continued execution in an insecure state.		DHS/DACS; OWASP Code Review Guide: Error Handling; SAFECode "Fundamental Practices"; CWE-388; CWE-390; CWE-391; CWE-396; CWE-397; CWE-544
		EE.1-3. Notifications of errors and exceptions do not disclose sensitive technical or human information.		DHS/DACS; OWASP Code Review Guide: Error Handling; OWASP Secure Coding Practices; SAFECode "Fundamental Practices"; CWE-209
	EE.2. Software fails securely; if a program is forced to terminate unexpectedly, it shuts down in a safe and responsible manner.	**EE.2-1.** Software is designed to continue operating in a degraded manner until a threshold is reached that triggers orderly, secure termination.		DHS/DACS; CWE-636
		EE.2-2. In the case of failure, software reverts to secure default states that preserve confidentiality and integrity.		CWE-636

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE LIFECYCLE				
Vulnerability Management (VM)	**VM.1.** The vendor maintains an up-to-date vulnerability management plan.	**VM.1-1.** The vulnerability management plan outlines policies, responsibilities, and expectations for both internal and external stakeholders throughout the following phases of vulnerability management: (1) the vendor's identification or receipt of a vulnerability, (2) verification of the vulnerability, (3) remediation or mitigation of the vulnerability, (4) release of a solution, and (5) post-release.		ISO/IEC 29147; ISO/IEC 30111; SAFECode "Fundamental Practices"; SAMM
		VM.1-2. The vulnerability management plan addresses security testing and vulnerability identification methodologies to be applied throughout a product's lifecycle.		
		VM.1-3. The vulnerability management plan includes a process for gaining timely awareness of and managing vulnerabilities that are discovered in third-party components of the software.		SAFECode "Fundamental Practices"; SAMM
	VM.2. Vulnerabilities are identified and resolved rapidly and comprehensively, according to risk-based prioritization.	**VM.2-1.** Upon identification, vulnerabilities are verified and subjected to root cause and risk analysis.		ISO/IEC 30111; SAFECode "Fundamental Practices"; SAMM
		VM.2-2. Vulnerabilities are assigned a unique identification number.		ISO/IEC 30111; SAFECode "Fundamental Practices"

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE LIFECYCLE				
Vulnerability Management (VM) *(continued)*	**VM.2.** Vulnerabilities are identified and resolved rapidly and comprehensively, according to risk-based prioritization.	**VM.2-3.** Vulnerabilities are assigned a severity value based on risk, using a standardized scoring methodology.		CVSS
		VM.2-4. Remediation and mitigation activities are informed by the severity of the vulnerability.		ISO/IEC 30111; SAFECode "Fundamental Practices"; SAMM
	VM.3. The vendor maintains a coordinated vulnerability disclosure program.	**VM.3-1.** The vendor establishes a clearly defined and easily accessible intake mechanism to accept vulnerability information (email, portal, etc.).		ISO 29147; SAFECode "Fundamental Practices"; SAMM; ENISA Good Practice Guide on Vulnerability Disclosure; IoT Security Foundation Vulnerability Disclosure Best Practice Guidelines
		VM.3-2. A vendor's intake mechanism provides for secure and confidential communication of sensitive vulnerability information.		ISO 29147; SAFECode "Fundamental Practices"; IoT Security Foundation Vulnerability Disclosure Best Practice Guidelines
		VM.3-3. The vendor publishes, in simple and clear language, its policies for interacting with vulnerability reporters, addressing, at minimum: (1) how the vendor would like to be contacted, (2) options for secure communication, (3) expectations for communication from the vendor regarding the status of a reported vulnerability, (4) desired information regarding a potential vulnerability, (5) issues that are out of scope of the vulnerability disclosure program, (6) how submitted vulnerability reports are tracked, and (7) expectations for whether and how a reporter will be credited.		ISO 29147; ENISA Good Practice Guide on Vulnerability Disclosure; IoT Security Foundation Vulnerability Disclosure Best Practice Guidelines

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE LIFECYCLE				
Vulnerability Management (VM) *(continued)*	**VM.3.** The vendor maintains a coordinated vulnerability disclosure program.	**VM.3-4.** The vendor maintains a system to record and track all reports of potential vulnerabilities.		ISO 29147
		VM.3-5. The vendor notifies vulnerability reporters of when reported vulnerabilities are remediated or mitigated.		ISO 29147
Configuration (CF)	**CF.1.** The software is deployed with configurations and configuration guidance that facilitate secure installation and operation.	**CF.1-1.** The software documentation specifies configuration parameters that are as restrictive as feasible, to make sure the software is as resistant as possible to anticipated attacks and exploits.		DHS/DACS
		CF.1-2. The software documentation describes secure installation procedures for initial installation and installation for additional components, updates, and patches.		BSIMM; DHS/DACS
		CF.1-3. The software documentation describes configurations and procedures for secure configuration under normal operation.		
		CF.1-4. The software prompts users to change any default passwords before the software becomes operational.		DHS/DACS
		CF.1-5. Configuration guidance statements and configuration controls are clearly communicated and automated wherever possible.		NIST Special Publication 800-126

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE LIFECYCLE				
Configuration (CF) *(continued)*	**CF.1.** The software is deployed with configurations and configuration guidance that facilitate secure installation and operation.	**CF.1-6.** Software configuration settings can be altered to tailor security settings to the operating environment.	User configuration may not always be possible or necessary. However, where viable, the software should be delivered in a configuration that is as secure as possible based on its anticipated usage, and should support the ability of users to modify security settings to accommodate changing environments or requirements.	
Vulnerability Notification and Patching (VN)	**VN.1.** Vendors disseminate timely patches or updates to address identified security issues.	**VN.1-1.** Patches or updates are developed and disseminated based on risk-informed prioritization, in accordance with the vendor's vulnerability management program.		ISO/IEC 30111; SAFECode "Fundamental Practices"; DHS/DACS; Microsoft SDL; SAMM
		VN.1-2. Patches or updates are subjected to testing for functionality and security prior to release.		DHS/DACS; Microsoft SDL
		VN.1-3. All patches and updates are documented.		DHS/DACS
		VN.1-4. Development and dissemination of patches or updates are coordinated with other vendors where appropriate to address multi-vendor security issues or supply chain security issues.		ISO/IEC 30111; FIRST "Guidelines and Practices for Multi-Party Vulnerability Coordination and Disclosure"
	VN.2. Patches or updates are disseminated securely.	**VN.2-1.** Patches or updates are transmitted in a manner that prevents exposure of the software image.		NTIA "Voluntary Framework for Enhancing Update Process Security"
		VN.2-2. The patch or update deliverable is cryptographically signed to ensure its integrity and authenticity.		ISO/IEC 29147; NTIA "Voluntary Framework for Enhancing Update Process Security"

Category	Subcategory	Diagnostic Statement	Comments on Implementation	Relevant Standards and Informative Resources
SECURE LIFECYCLE				
Vulnerability Notification and Patching (VN) *(continued)*	**VN.3.** Patches or updates for security issues are accompanied by advisory messages informing users of relevant information.	**VN.3-1.** Users are notified of a significant security issue when a remediation is in place for each supported version of the affected product.		SAFECode "Fundamental Practices"
		VN.3-2. Advisory messages notifying users of security issues include information on affected products, applicable versions, and platforms; a unique identification number; and a brief description of the vulnerability and its potential impact.		ISO/IEC 29147; SAFECode "Fundamental Practices"
End-of-Life (EL)	**EL.1.** Vendor maintain consistent lifecycle guidance.	**EL.1-1.** Vendor communicates realistic assumptions and expectations regarding the nature and lifespan of product support in tandem with initial software delivery.		
		EL.1-2. Vendor clearly communicates decisions to terminate support for a software product to customers and users, identifying the expected support termination date; the anticipated risk of continued product use beyond the termination of support; possible mitigation actions; and options for technical migration to replacement products.		
		EL.1-3. Software is continually monitored to ensure that third-party components have not reached end-of-life milestones or are removed or otherwise remediated.		

IV. References

Definitions

Access Control. Means to ensure that access to assets is authorized and restricted based on business and security requirements. *(Source: ISO/IEC 27000: 2018)*

Algorithm. A finite set of well-defined rules for the solution of a problem in a finite number of steps, sequence of operations for performing a specific task, or finite ordered set of well-defined rules for the solution of a problem. *(Source: ISO/IEC/IEEE 24765: 2017)*

Authentication. Provision of assurance that a claimed characteristic of an entity is correct. *(Source: ISO/IEC 27000: 2018)*

Control. A measure that is modifying risk. Controls include any process, policy, device, practice, or other actions that modify risk. *(Source: ISO/IEC 27000: 2018)*

Error. Discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. *(Source: ISO/IEC 15026-1: 2019)*

Exception. An event that causes suspension of normal program execution, or an indication that an operation request was not performed successfully. *(Source: ISO/IEC/IEEE 24765: 2017)*

Fault isolation. The ability of a subsystem to prevent a fault within the subsystem from causing consequential faults in other subsystems. *(Source: ISO/IEC/IEEE 24765: 2017)*

Fuzzing. A means of testing that causes a software program to consume deliberately malformed data to see how the program reacts. *(Source: Microsoft Security Development Lifecycle Process Guidance Version 5.2)*

Lifecycle. States involved in the management of an asset; evolution of a system, product, service, project, or other human-made entity from conception through retirement. *(Sources: ISO/IEC 12207: 2017; ISO/IEC 27034: 2011)*

Mitigation. The process of remediating a weakness, leaving the software in a more secure state. *(Source: Common Weakness Enumeration/MITRE)*

Patch. A modification made directly to an object program without reassembling or recompiling from the source program, or a software component that, when installed, directly modifies files or device settings related to a different software component without changing the version number or release details for the related software component. *(Source: ISO/IEC 19770-2: 2015)*

Penetration testing. A test method in which the security of a computer program or network is subjected to deliberate simulated attack. *(Source: Microsoft Security Development Lifecycle Process Guidance Version 5.2)*

Release gate. A specific point established in the software development lifecycle where a project may not move forward until it meets certain security conditions established by an organization at the project's inception. *(Adapted from Software Assurance Maturity Model, Version 1.0)*

Risk. An expression of the effect of uncertainty on cybersecurity objectives, as understood through the analysis of identified threats to a product or system, the known vulnerabilities of that product or system, and the potential consequences of the compromise of the product or system. *(Source: BSA International Cybersecurity Policy Framework)*

Sandboxing. A restricted, controlled execution environment that prevents potentially malicious software, such as mobile code, from accessing any system resources except those for which the software is authorized. *(Source: Committee on National Security Systems No. 4009)*

Software. All or part of the programs that process or support the processing of digital information. *(Source: ISO/IEC 12207: 2017)*

Third-party components. Components of a software project of external origin, including open-source components, purchased commercial off-the-shelf software, and online services used by the software project. *(Adapted from Software Assurance Maturity Model, Version 1.5)*

Threat modeling. A systematic exploration technique to expose any circumstance or event having the potential to cause harm to a system in the form of destruction, disclosure, modification of data, or denial of service. *(Source: ISO/IEC/IEEE 24765: 2017)*

Vulnerability. Weakness of software, hardware, or online service that can be exploited. *(Source: ISO/IEC 30111: 2013)*

Weakness. A type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. *(Source: Common Weakness Enumeration/MITRE)*

Acronyms

BSIMM	Building Security in Maturity Model, Version 9
CAPEC	Common Attack Pattern Enumeration and Classification
CVSS	Common Vulnerability Scoring System
CWSS	Common Weakness Scoring System
DHS/DACS	Department of Homeland Security/ Data & Analysis Center for Software, *Enhancing the Development Life Cycle to Produce Secure Software*, Version. 2.0.
DoD-PPP	Department of Defense, "Software Assurance Countermeasures in Program Protection Planning"
FIPS	Federal Information Processing Standards
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
Microsoft SDL	Microsoft's Security Development Lifecycle Process Guidance, Version 5.2
NCSC	United Kingdom National Cyber Security Centre Secure Development and Deployment Guidance
NIST	National Institute for Standards and Technology
NIST IR	NIST Interagency Report
NIST SP	NIST Special Publication
NTIA	National Telecommunications and Information Administration
NVD	National Vulnerability Database
OAuth	Initiative for Open Authentication
OIDC	OpenID Connect
OWASP	Open Web Application Security Project
SAFECode "Fundamental Practices"	SAFECode *Fundamental Practices for Secure Software Development*, Version 3.0
SAML	Security Assertion Markup Language
SAMM	Software Assurance Maturity Model, Version 1.5
SEI	Carnegie Mellon University's Software Engineering Institute
SPDX	Software Package Data Exchange, Version 2.1
U2F	Universal Second Factor
UAF	Universal Authentication Framework
WS-FED	Web Services Federation Language, Version 1.2

Sources

Adobe, Adobe Secure Engineering Overview, March 2018. https://www.adobe.com/content/dam/acom/en/security/pdfs/adobe-secure-engineering-wp.pdf.

Apple, *Secure Coding Guide*. https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html.

Box, *Box Platform Guidelines and Security*. https://developer.box.com/docs/security-guidelines.

BSA | The Software Alliance, *BSA International Cybersecurity Policy Framework*. https://bsacybersecurity.bsa.org/wp-content/uploads/2018/04/BSA_cybersecurity-policy.pdf.

Carnegie Mellon University Software Engineering Institute, *SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems*, 2016 Edition, June 2016. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=454220.

Carnegie Mellon University Software Engineering Institute, *SEI CERT C++ Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems*, 2016 Edition, March 2017. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=494932.

Carnegie Mellon University Software Engineering Institute, *SEI CERT Oracle Coding Standard for Java*, October 11, 2016. https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java.

Committee on National Security Systems (CNSS), *Committee on National Security Systems Glossary*, CNSS Instruction No. 4009, April 6, 2015. https://www.cnss.gov/CNSS/issuances/Instructions.cfm.

European Union Agency for Network and Information Security, *Good Practice Guide on Vulnerability Disclosure*, January 18, 2016. https://www.enisa.europa.eu/publications/vulnerability-disclosure.

FIDO Alliance, *Universal 2nd Factor Overview*, April 11, 2017. https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.pdf.

FIDO Alliance, *Universal Authentication Framework Architectural Overview*, Version 1.1, February 2, 2017. https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html.

Forum for Incident Response and Security Teams, *Common Vulnerability Scoring System: Specification Document*, Version 3.0. https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf.

Forum for Incident Response and Security Teams, *Guidelines and Practices for Multi-Party Vulnerability Coordination and Disclosure*, Version 1.0, Summer 2017. https://www.first.org/global/sigs/vulnerability-coordination/multiparty/FIRST-Multiparty-Vulnerability-Coordination-latest.pdf?20180320.

Howard, Michael and Steve Lipner, *The Security Development Lifecycle: A Process for Developing Demonstrably More Secure Software*, 2006, Redmond, WA: Microsoft Press.

IBM, *Security in Development: The IBM Secure Engineering Framework*, 2010. https://www.redbooks.ibm.com/redpapers/pdfs/redp4641.pdf.

Initiative for Open Authentication, *OAuth 2.0*, October 2012. https://oauth.net/2/.

International Organization of Standardization, *Information Technology—IT Asset Management—Parts 1–2*, ISO/IEC 19770 (1: 2017–2: 2015).

International Organization of Standardization, *Information Technology—Security Techniques—Information Security Management Systems—Overview and Vocabulary*, ISO/IEC 27000: 2018.

International Organization of Standardization, *Information Technology—Security Techniques—Entity Authentication—Parts 1–3*, ISO/IEC 9798- (1: 2010–3: 2019).

International Organization of Standardization, *Information Technology—Programming Languages, Their Environments and System Software Interfaces—C Secure Coding Rules*, ISO/IEC TS 17961: 2013.

International Organization of Standardization, *Information Technology—Security Techniques—Encryption Algorithms—Parts 1–5*, ISO/IEC 18033 (1: 2015–5: 2015).

International Organization of Standardization, *Information Technology—Security Techniques—Authenticated Encryption*, ISO/IEC 19772: 2009.

International Organization of Standardization, *Information Technology—Security Techniques—Security Requirements for Cryptographic Modules*, ISO/IEC 19790: 2012.

International Organization of Standardization, *Information Technology—Security Techniques—Application Security; Parts 1–7*, ISO/IEC 27034 (1:2011–7:2018).

International Organization of Standardization, *Information Technology—Security Techniques—Vulnerability Disclosure*, ISO/IEC 29147: 2018, October 23, 2018.

International Organization of Standardization, *Information Technology—Security Techniques—Vulnerability Handling Processes*, ISO/IEC 30111: 2013(E), November 1, 2013.

International Organization of Standardization, *Systems and Software Engineering—Software Lifecycle Processes*, ISO/IEC/IEEE 12207: 2017.

International Organization of Standardization, *Systems and Software Engineering—Systems and Software Assurance—Part 1: Concepts and Vocabulary*, ISO/IEC/IEEE 15026 (1: 2019).

International Organization of Standardization, *Systems and Software Engineering—Vocabulary*, ISO/IEC/IEEE 24765: 2017.

IoT Security Foundation, *Vulnerability Disclosure: Best Practice Guidelines, Release 1.1*, December 2017. https://iotsecurityfoundation.org/wp-content/uploads/2017/01/Vulnerability-Disclosure.pdf.

The Linux Foundation, Software Package Data Exchange, Specification Version 2.1, 2016. https://spdx.org/sites/cpstandard/files/pages/files/spdxversion2.1.pdf.

McGraw, Gary, Sammy Migues, and Jacob West, *Building Security in Maturity Model (BSIMM)*, Version 9, 2018. https://www.bsimm.com.

Microsoft, *Security Development Lifecycle: SDL Process Guidance*, Version 5.2, May 23, 2012. https://www.microsoft.com/en-us/download/details.aspx?id=29884.

MITRE Corporation, *Common Attack Pattern Enumeration and Classification*, Version 3.0. https://capec.mitre.org/data/index.html.

MITRE Corporation, Common Weakness Enumeration, Version 3.2. https://cwe.mitre.org/data/index.html.

MITRE Corporation, *Common Weakness Scoring System*, Version 1.0.1, September 5, 2014. https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

MITRE Corporation and the SANS Institute, *CWE/SANS Top 25 Most Dangerous Software Errors*, Version 1.0.3, September 13, 2011. https://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf.

OASIS, *Security Assertion Markup Language*, Version 2.0, March 25, 2008. http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf.

OASIS, *Web Services Federation Language*, Version 1.2, May 22, 2009. http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html.

Okta, *Okta Security Technical White Paper*. https://www.okta.com/sites/default/files/Okta%20Technical%20Security%20Whitepaper.pdf.

Open ID Foundation, *Open ID Connect*, Version 1.0, November 8, 2014. https://openid.net/connect/.

Open Web Application Security Project (OWASP), *Application Security Verification Standard*, Version 3.0, October 2015. https://www.owasp.org/images/6/67/OWASPApplicationSecurityVerificationStandard3.0.pdf.

Oracle, *Security Practices: Oracle Software Security Assurance*. https://www.oracle.com/corporate/security-practices/assurance/.

OWASP, *Attack Surface Analysis Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.md.

OWASP, *Authentication Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Authentication_Cheat_Sheet.md.

OWASP, *C-Based Toolchain Hardening Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/C-Based_Toolchain_Hardening_Cheat_Sheet.md.

OWASP, *Code Review Guide*, Version 2.0, July 2017. https://www.owasp.org/images/5/53/OWASP_Code_Review_Guide_v2.pdf.

OWASP, *Cryptographic Storage Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cryptographic_Storage_Cheat_Sheet.md.

OWASP, *Development Guide*, Version 2.0.1, June 2014. https://github.com/OWASP/DevGuide/tree/dc5a2977a4797d9b98486417a5527b9f15d8a251/DevGuide2.0.1.

OWASP, *Input Validation Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md.

OWASP, *Logging Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Logging_Cheat_Sheet.md.

OWASP, *OWASP Top 10 — 2017: The Ten Most Critical Web Application Security Risks*, 2017. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

OWASP, *Password Storage Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md.

OWASP, *Secure Coding Practices Quick Reference Guide*, Version 2.0, November 2010. https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf.

OWASP, *Software Assurance Maturity Model*, Version 1.5, April 2017. https://owaspsamm.org/v1-5/downloads/.

OWASP, *Testing Guide*, Version 4.0, September 2014. https://www.owasp.org/images/1/19/OTGv4.pdf.

OWASP, *Threat Modeling Cheat Sheet*. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Threat_Modeling_Cheat_Sheet.md.

SAFECode, *Fundamental Practices for Secure Software Development*, Third Edition, March 2018. https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf.

SAFECode, *Fundamental Practices for Secure Software Development*, Second Edition, February 2011. https://safecode.org/publication/SAFECode_Dev_Practices0211.pdf.

SAFECode, *Managing Security Risks Inherent in the Use of Third-Party Components*, 2017. https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf.

SAFECode, *The Software Supply Chain Integrity Framework: Defining Risks and Responsibilities for Securing Software in the Global Supply Chain*, July 21, 2009. http://safecode.org/publication/SAFECode_Supply_Chain0709.pdf.

SAFECode, *Tactical Threat Modeling*, May 2017. https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf.

Salesforce, *Secure Coding Guide*, Version 45.0, January 30, 2019. https://resources.docs.salesforce.com/218/latest/en-us/sfdc/pdf/secure_coding.pdf.

Symantec, "Executive Summary: Symantec Software Security Process," 2019. https://www.symantec.com/content/dam/symantec/docs/other-resources/symantec_software_security_process.pdf.

United Kingdom National Cyber Security Centre Secure, *Guidance for Secure Development and Deployment*, December 11, 2017. https://www.ncsc.gov.uk/guidance/secure-development-and-deployment.

United States Department of Defense, "Software Assurance Countermeasures in Program Protection Planning," March 2014. https://www.acq.osd.mil/se/docs/swa-cm-in-ppp.pdf.

United States Department of Homeland Security/ Data & Analysis Center for Software, *Enhancing the Development Life Cycle to Produce Secure Software*, Version. 2.0, October 2008. http://www.seas.upenn.edu/~lee/09cis480/papers/DACS-358844.pdf.

United States National Institute for Standards and Technology, *BIOS Protection Guidelines: Recommendations of the National Institute of Standards and Technology*, Special Publication 800-147, April 2011. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-147.pdf.

United States National Institute for Standards and Technology, *Digital Identity Guidelines, Special Publication 800-63-3*, June 2017. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf.

United States National Institute for Standards and Technology, *Federal Information Processing Standards*. https://www.nist.gov/standardsgov/compliance-faqs-federal-information-processing-standards-fips.

United States National Institute for Standards and Technology, *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags, Interagency Report 8060*, April 2016. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf.

United States National Institute for Standards and Technology, *National Vulnerability Database*. https://nvd.nist.gov/.

United States National Institute for Standards and Technology, *Notional Supply Chain Risk Management Practices for Federal Information Systems, Interagency Report 7622*, October 2012. https://csrc.nist.gov/publications/detail/nistir/7622/final.

United States National Institute for Standards and Technology, *Recommendation for Key Management: Part I: General, Special Publication 800-57*, Revision 4, January 2016. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

United States National Institute for Standards and Technology, *Security and Privacy Controls for Federal Information Systems and Organizations, Special Publication 800-53*, Revision 4, April 2013. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-53r4.pdf.

United States National Institute for Standards and Technology, *The Technical Specification for the Security Content Automation Protocol, Special Publication 800-126*, Revision 3, February 2018. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-126r3.pdf.

United States National Telecommunications and Information Administration, *Voluntary Framework for Enhancing Update Process Security*, October 31, 2017. https://www.ntia.doc.gov/files/ntia/publications/ntia_iot_capabilities_oct31.pdf.

www.bsa.org

BSA Worldwide Headquarters

20 F Street, NW
Suite 800
Washington, DC 20001

+1.202.872.5500
@BSAnews
@BSATheSoftwareAlliance

BSA Asia-Pacific

300 Beach Road
#25-08 The Concourse
Singapore 199555

+65.6292.2072
@BSAnewsAPAC

BSA Europe, Middle East & Africa

65 Petty France
Ground Floor
London, SW1H 9EU
United Kingdom

+44.207.340.6080
@BSAnewsEU