Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)

Donna Dodson
Applied Cybersecurity Division
Information Technology Laboratory

Murugiah Souppaya
Computer Security Division
Information Technology Laboratory

Karen Scarfone
Scarfone Cybersecurity
Clifton, VA

June 11, 2019

National Institute of Standards and Technology
U.S. Department of Commerce

Abstract

Few software development life cycle (SDLC) models explicitly address software security in detail, so secure software development practices usually need to be added to each SDLC model to ensure the software being developed is well secured. This white paper recommends a core set of high-level secure software development practices, called a secure software development framework (SSDF), to be added to each SDLC implementation. The paper facilitates communications about secure software development practices amongst business owners, software developers, and cybersecurity professionals within an organization. Following these practices should help software producers reduce the number of vulnerabilities in released software, mitigate the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences. Software consumers can reuse and adapt the practices in their software acquisition processes.

Keywords

secure software development; secure software development framework (SSDF); secure software development practices; software acquisition; software development; software development life cycle (SDLC); software security.

Disclaimer

Additional Information

For additional information on NIST's Cybersecurity programs, projects and publications, visit the Computer Security Resource Center. Information on other efforts at NIST and in the Information Technology Laboratory (ITL) is also available.

Public Comment Period: *June 11, 2019 through August 5, 2019*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: ssdf@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

Acknowledgments

The authors wish to thank all the individuals and organizations who provided comments on the preliminary ideas and drafts, particularly BSA | The Software Alliance, the Information Security and Privacy Advisory Board (ISPAB), and the members of the Software Assurance Forum for Excellence in Code (SAFECode).

Audience

There are two primary audiences for this white paper. The first is software producers (e.g., commercial-off-the-shelf [COTS] product vendors, government-off-the-shelf [GOTS] software developers, custom software developers) regardless of size, sector, or level of maturity. The second is software consumers, both federal government agencies and other organizations. Readers of this document are not expected to be experts in secure software development in order to understand it, but such expertise is required to implement its recommended practices.

Personnel within the following Workforce Categories and Specialty Areas from the National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework [1] are most likely to find this publication of interest:

- Securely Provision (SP): Risk Management (RSK), Software Development (DEV), Systems Requirements Planning (SRP), Test and Evaluation (TST), Systems Development (SYS)
- Operate and Maintain (OM): Systems Analysis (ANA)
- Oversee and Govern (OV): Training, Education, and Awareness (TEA), Cybersecurity Management (MGT), Executive Cyber Leadership (EXL), Program/Project Management (PMA) and Acquisition
- Protect and Defend (PR): Incident Response (CIR), Vulnerability Assessment and Management (VAM)
- Analyze (AN): Threat Analysis (TWA), Exploitation Analysis (EXP)

Trademark Information

Note to Reviewers

This white paper is intended as a starting point for discussing the concept of a secure software development framework (SSDF), and it does not provide a comprehensive view of SSDFs. Future work will expand on the material in this white paper, potentially covering topics such as how an SSDF may apply to and vary for different software development methodologies, and how an organization can transition from using just their current software development practices to also incorporating the practices specified by the SSDF. It is likely that the future work will primarily take the form of use cases so the insights will be more readily applicable to certain types of development environments.

Table of Contents

1 Introduction 1
2 Secure Software Development Framework (SSDF) 3
References 17
Appendix A— Acronyms 19

1 Introduction

A *software development life cycle (SDLC)* is a formal or informal methodology for designing, creating, and maintaining software. There are many models for SDLCs, including waterfall, spiral, agile, and Development and Operations (DevOps). Few SDLC models explicitly address software security in detail, so secure software development practices usually need to be added to and integrated within each SDLC model to ensure the software being developed under that model is well secured. Regardless of which SDLC model is used to develop software, secure software development practices should be integrated throughout it for three reasons: to reduce the number of vulnerabilities in released software, to mitigate the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and to address the root causes of vulnerabilities to prevent future recurrences. Most aspects of security can be addressed at multiple places within an SDLC, but in general, the earlier in the SDLC security is addressed, the less effort is ultimately required to achieve the same level of security.

There are many existing documents on secure software development practices. This white paper does not introduce new practices or define new terminology; instead, it describes a subset of high-level practices based on established standards, guidance, and secure software development practice documents. These practices, collectively called a secure software development framework (SSDF), should be particularly helpful for the target audiences to achieve security software development objectives.

This white paper expresses secure software development practices but does not prescribe exactly how to implement them. The most important thing is implementing the practices and not the mechanisms used to do so. For example, one organization might automate a particular step, while another might use manual processes instead. Advantages of specifying the practices at a high level include the following:

- Can be used by organizations in any sector or community, regardless of size or cybersecurity sophistication
- Can be applied to software developed to support information technology (IT), industrial control systems (ICS), cyber-physical systems (CPS), or the Internet of Things (IoT)
- Can be integrated into any existing software development workflow and automated toolchain; should not negatively affect organizations that already have robust secure software development practices in place
- Makes the practices broadly applicable—not specific to particular technologies, platforms, programming languages, SDLC models, development environments, operating environments, tools, etc.
- Can help an organization document its secure software development baseline today and define its future target baseline as part of its continuous improvement process.
- Can assist an organization currently using a classic software development model in transitioning its secure software development practices for use with a modern software development model (e.g., agile, DevOps).

This white paper also provides a common language to describe fundamental secure software development practices. This is similar to the approach of the *Framework for Improving Critical Infrastructure Cybersecurity*, also known as the NIST Cybersecurity Framework [2]. Expertise in secure software development is not required to understand the practices. This helps facilitate communications about secure software practices amongst both internal and external organizational stakeholders, including:

- Business owners, software developers, and cybersecurity professionals within an organization
- Software consumers, both federal government agencies and other organizations, that want to define required or desired characteristics for software in their acquisition processes in order to have higher-quality software (particularly with fewer security vulnerabilities)
- Software producers (e.g., commercial-off-the-shelf [COTS] product vendors, government-off-the-shelf [GOTS] software developers, software developers working within or on behalf of software consumer organizations) that want to integrate secure software development practices throughout their SDLCs, express their secure software practices to their customers, or define requirements for their suppliers

This white paper's practices are not based on an assumption of all organizations having the same security objectives and priorities. Rather, the recommendations reflect that each software producer may have unique security assumptions and each software consumer may have unique security needs. While the desire is for each security producer to follow all applicable practices, the expectation is that the degree to which each practice is implemented will vary based on the producer's security assumptions. The practices provide flexibility for implementers, but they are also clear to avoid leaving too much open to interpretation.

2 Secure Software Development Framework (SSDF)

This white paper introduces a secure software development framework (SSDF) of fundamental, sound secure software development practices based on established secure software development practice documents. For the purposes of this white paper, the practices are organized into four groups:

- **Prepare the Organization (PO):** Ensure the organization’s people, processes, and technology are prepared to perform secure software development.
- **Protect the Software (PS):** Protect all components of the software from tampering and unauthorized access.
- **Produce Well-Secured Software (PW):** Produce well-secured software that has minimal security vulnerabilities in its releases.
- **Respond to Vulnerability Reports (RV):** Identify vulnerabilities in software releases and respond appropriately to address those vulnerabilities and prevent similar vulnerabilities from occurring in the future.

Each practice is defined with the following elements:

- **Practice:** A brief statement of the practice, along with a unique identifier and an explanation of what the practice is and why it is beneficial.
- **Task:** An individual action (or actions) needed to accomplish a practice.
- **Implementation Example:** An example of a type of tool, process, or other method that could be used to implement this practice; not intended to imply that any example or combination of examples is required, or that only the stated examples are feasible options.
- **Reference:** An established secure development practice document and its mappings to a particular task.

Although most practices are relevant for any software development effort, some practices are not always applicable. For example, if developing a particular piece of software does not involve using a compiler, there would be no need to follow a practice on configuring the compiler to improve executable security.

189

Practices	Tasks	Implementation Examples	References
Prepare the Organization (PO)			
Define Security Requirements for Software Development (PO.1): Ensure security requirements for software development are known at all times so they can be taken into account throughout the SDLC, and duplication of effort can be minimized because the requirements information can be collected once and shared. This includes requirements from internal sources, such as the organization's policies, business objectives, and risk management strategy, and external sources, such as applicable laws and regulations.	**PO.1.1:** Identify all applicable security requirements for the organization's general software development, and maintain the requirements over time.	• Define policies that specify the security requirements for the organization's software to meet, including secure coding practices for developers to follow. • Define policies that specify software architecture requirements, such as making code modular to facilitate code reuse and easier updates, and isolating security functionality from other functionality during code execution. • Define policies for securing the development infrastructure, such as developer workstations and code repositories. • Ensure policies cover the entire software life cycle, including notifying users of the impending end of software support and the date of software end-of-life, when the software will no longer function properly. • Use a well-known set of security requirements as a structure or lexicon for defining the organization's requirements. This set can readily be mapped to other third-party security requirements the organization is also subject to. • Review and update the requirements after each response to a vulnerability incident. • Conduct a periodic (typically at least annual) review of all security requirements. • Promptly review new external requirements and updates to existing external requirements. • Educate affected developers on the impending changes in requirements.	**BSIMM9** [3]: CP1.1, CP1.3, SR1.1 **BSA** [19]: SC.1-1, SC.2, PD.1-1, PD.1-2, PD.1-3, PD.2-2 **ISO27034** [4]: 7.3.2 **MSSDL** [5]: Practice 2 **NISTCSF** [2]: ID.GV-3 **OWASPSCP** [6]**:** Entire guide **OWASPTEST** [7]: Phase 2.1 **PCISSLRAP** [8]: 2.1 **SAMM15** [9]: PC1-A, PC1-B, PC2-A, SR1-A, SR1-B, SR2-B **SCFPSSD** [10]: Planning the Implementation and Deployment of Secure Development Practices; Establish Coding Standards and Conventions **SP80053** [11]: SA-15 **SP80064** [12]: 3.1.3.1 **SP800160** [13]: 3.1.2, 3.3.1, 3.4.2, 3.4.3 **SP800181** [1]: T0414; K0003, K0039, K0044, K0157, K0168, K0177, K0211, K0260, K0261, K0262, K0524; S0010, S0357, S0368; A0033, A0123, A0151

Practices	Tasks	Implementation Examples	References
Implement Roles and Responsibilities (PO.2): Ensure everyone inside and outside the organization involved in the SDLC is prepared to perform their SSDF-related roles and responsibilities throughout the SDLC.	**PO.2.1:** Create new roles and alter responsibilities for existing roles to encompass all parts of the SSDF. Periodically review the defined roles and responsibilities, and update them as needed.	• Define SSDF-related roles and responsibilities for all members of the software development team. • Integrate the security roles into the software development team. • Define roles and responsibilities for cybersecurity staff, security champions, senior management, software developers, product owners, and others involved in the SDLC. • Conduct an annual review of all roles and responsibilities. • Educate affected individuals on the impending changes in roles and responsibilities.	**BSA**: PD.2-1, PD.2-2 **BSIMM9**: CP3.2, SM1.1 **NISTCSF**: ID.AM-6, ID.GV-2 **PCISSLRAP**: 1.2 **SCSIC** [14]**:** Vendor Software Development Integrity Controls **SP80053**: SA-3 **SP80064**: 3.1.3.1 **SP800160**: 3.2.1, 3.2.4, 3.3.1 **SP800181**: K0233
	PO.2.2: Provide role-specific training for all personnel in roles with responsibilities that contribute to secure development. Periodically review role-specific training and update it as needed.	• Document the desired outcomes of training for each role. • Acquire or create training for each role; acquired training may need customization for the organization.	**BSA**: PD.2-2 **BSIMM9**: CP2.5, SM1.3, T1.1, T1.5, T1.6, T1.7, T2.6, T3.2, T3.4 **MSSDL**: Practice 1 **NISTCSF**: PR.AT-* **PCISSLRAP**: 1.3 **SAMM15**: EG1-A, EG2-A **SCAGILE** [15]**:** Operational Security Tasks 14, 15; Tasks Requiring the Help of Security Experts 1 **SCFPSSD**: Planning the Implementation and Deployment of Secure Development Practices **SCSIC:** Vendor Software Development Integrity Controls **SP80053**: SA-8 **SP80064**: 3.1.3.5 **SP800160**: 3.2.4 **SP800181**: OV-TEA-001, OV-TEA-002; T0030, T0073, T0320; K0204, K0208, K0220, K0226, K0243, K0245, K0252; S0100, S0101; A0004, A0057

Practices	Tasks	Implementation Examples	References
Implement a Supporting Toolchain (PO.3): Use automation to reduce the human effort needed and improve the accuracy, consistency, and comprehensiveness of security practices throughout the SDLC, as well as a way to document and demonstrate use of these practices without significant additional effort or expense.	**PO.3.1:** Specify which tools or tool types are to be included in each toolchain and which tools or tool types are mandatory, along with how the toolchain components are to be integrated with each other.	• Define categories of toolchains, and specify the mandatory tools or tool types to be used for each category. • Use automated technology for toolchain management and orchestration. • Identify security tools to integrate into the developer toolchain.	**BSA**: TC.1, TC.1-1, TC.1-2 **MSSDL**: Practice 8 **SCAGILE:** Tasks Requiring the Help of Security Experts 9 **SP80053**: SA-15 **SP800181**: K0013, K0178
	PO.3.2: Following sound security practices, deploy and configure tools, integrate them within the toolchain, and maintain the individual tools and the toolchain as a whole.	• Evaluate, select, and acquire tools. • Integrate tools with other tools and with existing software development processes and workflows. • Update, upgrade, and replace existing tools. • Monitor tool logs for potential operational and security issues.	**BSA**: TC.1-1, TC.1-6 **SCAGILE:** Tasks Requiring the Help of Security Experts 9 **SP80053**: SA-15 **SP800181**: K0013, K0178
	PO.3.3: Configure tools to collect evidence and artifacts of their support of the secure software development practices.	• Use the organization's existing workflow or bug tracking systems to create an audit trail of secure development-related actions performed. • Determine how often the collected information should be audited, and implement processes to perform the auditing.	**BSA**: PD.1.6 **MSSDL**: Practice 8 **PCISSLRAP**: 2.5 **SCAGILE:** Tasks Requiring the Help of Security Experts 9 **SP80053**: SA-15 **SP800181**: K0013
Define Criteria for Software Security Checks (PO.4): Help ensure the software resulting from the SDLC meets the organization's expectations by defining criteria for checking the software's security during development.	**PO.4.1:** Define criteria for software security checks at one or more points within the SDLC.	• Ensure the criteria adequately indicate how effectively security risk is being managed. • Define key performance indicators (KPIs) for software security. • Add software security criteria to existing checks (e.g., the Definition of Done in agile SDLC methodologies). • Review the artifacts generated as part of the software development workflow system to determine if they meet the criteria purposes. • Record security check approvals, rejections, and requests for exception as part of the workflow and tracking system.	**BSA:** TV.2-1, TV.5-1 **BSIMM9**: SM1.4, SM2.2 **ISO27034**: 7.3.5 **MSSDL**: Practice 3 **OWASPTEST**: Phase 1.3 **SAMM15**: DR3-B, IR3-B, PC3-A, ST3-B **SP80053**: SA-15 **SP800160**: 3.2.1, 3.2.5, 3.3.1 **SP800181**: K0153, K0165

Practices	Tasks	Implementation Examples	References
	PO.4.2: Implement processes, mechanisms, etc. to gather the necessary information in support of the criteria.	• Use the toolchain to automatically gather information that informs security decision making. • Deploy additional tools if needed to support generation and collection of information supporting the criteria. • Automate decision making processes utilizing the criteria.	**BSA**: PD.1-6 **BSIMM9**: SM1.4, SM2.2 **SP80053**: SA-15 **SP800160**: 3.3.7 **SP800181**: T0349; K0153
Protect Software (PS)			
Protect All Forms of Code from Unauthorized Access and Tampering (PS.1): Help prevent unauthorized changes to code, both inadvertent and intentional, which could circumvent or negate the intended security characteristics of the software. For code not intended to be publicly accessible, it helps prevent theft of the software and makes it more difficult for attackers to find vulnerabilities in the software.	**PS.1.1:** Store all forms of code, including source code and executable code, based on the principle of least privilege so that only authorized personnel have the necessary forms of access. The protection needed will vary based on the nature of the code. For example, some code may be intended for public access, in which case its integrity and availability should be protected; other code may also need its confidentiality protected.	• Store all source code in a code repository, and restrict access to it. • Use version control features of the repository to track all changes made to code with accountability to the individual developer account. • Use code signing to help protect the integrity and provenance of executables. • Use cryptographic hashes to help protect the integrity of files. • Create and maintain a software bill of materials (SBOM) for each piece of software stored in the repository.	**BSA**: IA.1, IA.2-2, SM.4-1 **IDASOAR** [16]**:** Fact Sheet 25 **NISTCSF**: PR.AC-4 **PCISSLRAP**: 6.1 **SCSIC:** Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls **SP80064**: 3.1.3.5
Provide a Mechanism for Verifying Software Release Integrity (PS.2): Help software consumers ensure the software they acquire is legitimate and has not been tampered with.	**PS.2.1:** Make verification information available to software consumers.	• Post cryptographic hashes for release files on a well-secured website. • Use an established certificate authority for code signing so consumers can confirm the validity of signatures. • Periodically review the code signing processes, including certificate renewal and protection.	**BSA**: SM.4.2, SM.4.3, SM.5.1, SM.6.1 **BSIMM9**: SE2.4 **NISTCSF**: PR.DS-6 **PCISSLRAP**: 6.2 **SAMM15**: OE3-B **SCSIC:** Vendor Software Delivery Integrity Controls **SP800181**: K0178
Archive and Protect Each Software Release (PS.3): Helps identify, analyze, and eliminate vulnerabilities discovered in the software after release.	**PS.3.1:** Securely archive a copy of each release and all of its components, such as code, package files, third-party libraries, documentation, and release integrity verification information.	• Store all release files in a repository, and restrict access to them.	**BSA**: PD.1-6 **IDASOAR:** Fact Sheet 25 **NISTCSF**: PR.IP-4 **PCISSLRAP**: 5.2, 6.2 **SCSIC:** Vendor Software Delivery Integrity Controls **SP80053**: SA-15

Practices	Tasks	Implementation Examples	References
Produce Well-Secured Software (PW)			
Take Security Requirements and Risk Information into Account During Software Design (PW.1): Determine which security requirements the software's design should meet, and determine what security risks the software is likely to face during production operation and how those risks should be mitigated by the software's design. Addressing security requirements and risks during software design instead of later helps to make software development more efficient.	**PW.1.1:** Use threat modeling, attack modeling, attack surface mapping, and/or other forms of risk modeling to help assess the security risk for the software.	• Train the development team to create threat models and attack models, and to analyze how to address the risks and implement mitigations. • Perform more rigorous assessments for high-risk areas, such as protecting sensitive data. • Review vulnerability reports and statistics for previous software.	**BSA**: SC.1-3, SC.1-4 **BSIMM9**: AM1.3, AM1.5, AM2.1, AM2.2, AM2.5, AM2.6, AM2.7 **IDASOAR**: Fact Sheet 1 **ISO27034**: 7.3.3 **MSSDL**: Practice 4 **NISTCSF**: ID.RA-* **OWASPTEST**: Phase 2.4 **PCISSLRAP**: 3.2 **SAMM15**: DR1-A, TA1-A, TA1-B, TA3-B **SCAGILE:** Tasks Requiring the Help of Security Experts 3 **SCFPSSD**: Threat Modeling **SCTTM** [17]: Entire guide **SP80053**: SA-8, SA-15, SA-17 **SP800160**: 3.3.4, 3.4.5 **SP800181**: T0038, T0062, T0236; K0005, K0009, K0038, K0039, K0070, K0080, K0119, K0147, K0149, K0151, K0152, K0160, K0161, K0162, K0165, K0297, K0310, K0344, K0362, K0487, K0624; S0006, S0009, S0022, S0078, S0171, S0229, S0248; A0092, A0093, A107
Review the Software Design to Verify Compliance with Security Requirements and Risk Information (PW.2): Help ensure the software will meet the security requirements and satisfactorily address the identified risk information.	**PW.2.1:** Have someone qualified who was not involved with the software design review it to confirm it meets all the security requirements and satisfactorily addresses the identified risk information.	• Review the software design to confirm it addresses all the security requirements. • Review the risk models created during software design to determine if they appear to adequately identify the risks. • Review the software design to confirm it satisfactorily addresses the risks identified by the risk models. • Have the software's designer correct all failures to meet the requirements.	**BSA**: TV.3, TV.3-1, TV.5 **BSIMM9**: AA1.2, AA2.1 **ISO27034**: 7.3.3 **OWASPTEST**: Phase 2.2 **SAMM15**: DR1-A, DR1-B **SP800181**: T0328; K0038, K0039, K0070, K0080, K0119, K0152, K0153, K0161, K0165, K0172, K0297; S0006, S0009, S0022, S0036, S0141, S0171

Practices	Tasks	Implementation Examples	References
Verify Third-Party Software Complies with Security Requirements (PW.3): Reduce the risk associated with using acquired software modules and services, which are potential sources of additional vulnerabilities.	**PW.3.1:** Communicate requirements to vendors, open source communities, and other third parties who may provide software modules and services to the organization for reuse by the organization's own software.	• Define a core set of security requirements, and include them in acquisition documents, software contracts, and other agreements with third parties. • Define the security-related criteria for selecting commercial and open source software. • Require the providers of commercial software modules and services to provide evidence that their software complies with the organization's security requirements.	**BSA**: SM.1, SM.2, SM.2-1, SM.2.4 **BSIMM9**: CP2.4, SR2.5, SR3.2 **IDASOAR**: Fact Sheets 19**,** 21 **MSSDL**: Practice 7 **SAMM15**: SR3-A **SCFPSSD**: Manage Security Risk Inherent in the Use of Third-Party Components **SCSIC:** Vendor Sourcing Integrity Controls **SP80053:** SA-4, SA-12 **SP800160**: 3.1.1, 3.1.2 **SP800181**: T0203, T0415; K0039; S0374; A0056, A0161
	PW.3.2: Use appropriate means to verify commercial and open source third-party software modules and services comply with the requirements.	• See if there are publicly known vulnerabilities in the software modules and services that the vendor has not yet fixed. • Ensure each software module or service is still actively maintained, especially remediating new vulnerabilities found in the software. • Determine a plan of action for each third-party software module or service no longer being maintained or available in the future. • [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] • [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]	**BSA**: SC.3-1, TV.2 **IDASOAR**: Fact Sheet 21 **MSSDL**: Practice 7 **PCISSLRAP**: 4.1 **SCAGILE:** Tasks Requiring the Help of Security Experts 8 **SCFPSSD**: Manage Security Risk Inherent in the Use of Third-Party Components **SCSIC:** Vendor Sourcing Integrity Controls **SCTPC** [18]**:** 3.2.2 **SP80053**: SA-12 **SP800160**: 3.1.2, 3.3.8 **SP800181**: SP-DEV-002; K0153, K0266 [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]

Practices	Tasks	Implementation Examples	References
Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality (PW.4): Lower the costs of software development, expedite software development, and decrease the likelihood of introducing additional security vulnerabilities into the software. These are particularly true for software that implements security functionality, such as cryptographic modules and protocols.	**PW.4.1:** Acquire well-secured software libraries, modules, middleware, frameworks, and other components from third parties for use by the organization's software.	• Review and evaluate the third-party software components in the context of their expected use. If a component is to be used in a substantially different way in the future, perform the review and evaluation again with that new context in mind. • Establish an organization-wide software repository to host sanctioned and vetted open source components. • Maintain a list of approved commercial software components and component versions. • Designate which components must be included by software to be developed.	**BSA**: SM.2, SM.2.1 **IDASOAR**: Fact Sheet 19 **MSSDL**: Practice 6 **OWASPSCP:** Communication Security, Cryptographic Practices **SAMM15**: SA1-A **SCTPC:** 3.2.1 **SP80053**: SA-12 **SP80064**: 3.1.3.5 **SP800181**: K0039
	PW.4.2: Create well-secured software components in-house following SDLC processes to meet common internal software development needs that cannot be better met by third-party software.	• Follow the organization-established security practices for secure software development. • Maintain an organization-wide software repository for these components. • Designate which components must be included by software to be developed.	**BSIMM9**: SFD1.1, SFD2.1 **IDASOAR**: Fact Sheet 19 **SP80064**: 3.1.3.5 **SP800181**: SP-DEV-001
	PW.4.3: Where appropriate, build in support for using standardized security features and services, such as integrating with log management, identity management, access control, and vulnerability management systems.	• Maintain an organization-wide software repository of modules for supporting standardized security features and services. • Designate which security features and services must be supported by software to be developed.	**BSA**: SI.2, EN.1-1, LO.1 **MSSDL**: Practice 5 **OWASPSCP:** Authentication and Password Management **SCFPSSD**: Establish Log Requirements and Audit Practices

Practices	Tasks	Implementation Examples	References
Create Source Code Adhering to Secure Coding Practices (PW.5): Decrease the number of security vulnerabilities in the software, and reduce costs by eliminating vulnerabilities during source code creation.	**PW.5.1:** Follow all secure coding practices appropriate to the development languages and environment.	• Validate all untrusted input, and validate and properly encode all output. • Avoid using unsafe functions and calls. • Handle errors gracefully. • Provide logging and tracing capabilities. • Use development environments with features that encourage or require the use of secure coding practices. • Follow procedures for manually ensuring compliance with secure coding practices.	**BSA**: SC.2, SC.4, SC.3, SC.3-2, EE.1, EE.1.2, EE.2, LO.1, **IDASOAR**: Fact Sheet 2 **ISO27034**: 7.3.5 **MSSDL**: Practice 9 **OWASPSCP:** Error Handling and Logging, General Coding Practices, Input Validation, Output Encoding **SCFPSSD**: Establish Log Requirements and Audit Practices, Handle Data Safely, Handle Errors, Use Safe Functions Only **SP800181** [1]**:** SP-DEV-001; T0013, T0077, T0176; K0009, K0016, K0039, K0070, K0140, K0624; S0019, S0060, S0149, S0172, S0266**;** A0036, A0047
	PW.5.2: Have the developer review their own human-readable code, analyze their own human-readable code, and/or test their own executable code.	• [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] • [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]	[See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]
Configure the Compilation and Build Processes to Improve Executable Security (PW.6): Decrease the number of security vulnerabilities in the software, and reduce costs by eliminating vulnerabilities before testing occurs.	**PW.6.1:** Use compiler and build tools that offer features to improve executable security.	• Consider replacing older compiler and build tools with up-to-date versions.	**BSA**: TC.1-1, TC.1-3, TC.1-4, TC.1-5 **MSSDL**: Practice 8 **SCAGILE:** Operational Security Task 3 **SCFPSSD**: Use Current Compiler and Toolchain Versions and Secure Compiler Options **SCSIC:** Vendor Software Development Integrity Controls
	PW.6.2: Determine which features should be used and how each feature should be configured, then implement the approved configuration for compilation and build tools, processes, etc.	• Enable compiler features that produce warnings for potentially poorly secured code during the compilation process. • Enable compiler features that randomize characteristics, such as memory location usage, that would otherwise be easily predictable and thus exploitable. • Conduct testing to ensure the features are working as expected and not	**BSA**: TC.1, TC.1-3, TC.1-4, TC.1-5 **SCAGILE:** Operational Security Task 8 **SCFPSSD**: Use Current Compiler and Toolchain Versions and Secure Compiler Options **SCSIC:** Vendor Software Development Integrity Controls **SP800181**: K0039, K0070

Practices	Tasks	Implementation Examples	References
		inadvertently causing any operational issues or other problems. • Verify the approved configuration is enabled for compilation and build tools, processes, etc. • Document information about the compilation and build tool configuration in a knowledge base that developers can access and search.	
Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7): Help identify vulnerabilities before software is released so they can be corrected before release, which prevents exploitation. Using automated methods lowers the effort and resources needed to detect vulnerabilities. Human-readable code is source code and any other form of code an organization deems as human readable.	**PW.7.1:** Determine whether code *review* (a person directly looks at the code to find issues) and/or code *analysis* (tools are used to find issues in code, either in a fully automated way or in conjunction with a person) should be used.	• Follow the organization's policies or guidelines for when code review should be performed and how it should be conducted. • Follow the organization's policies or guidelines for when code analysis should be performed and how it should be conducted.	**SCSIC:** Peer Reviews and Security Testing **SP80053:** SA-11 **SP800181**: SP-DEV-002; K0013, K0039, K0070, K0153, K0165; S0174
	PW.7.2: Perform the code review and/or code analysis, and document and triage all discovered issues and recommended remediations in the development team's workflow or bug-tracking system.	• Have developers review their own code. • Perform peer review of code. • Use peer reviewing tools that facilitate the peer review process and document all discussions and other feedback. • Use a static analysis tool to automatically check code for vulnerabilities and for compliance with the organization's secure coding standards, with a human reviewing issues reported by the tool and remediating them as necessary. • Use review checklists to verify the code complies with the requirements. • Use automated tools to identify and remediate documented and verified unsafe software practices on a continuous basis as human-readable code is checked into the code repository. • Identify and document the root cause of each discovered issue. • Document lessons learned from code review and analysis in a knowledge base	**BSA**: PD.1-5, TV.2, TV.3 **BSIMM9**: CR1.2, CR1.4, CR1.6, CR2.6, CR2.7 **IDASOAR**: Fact Sheets 3, 4, 5, 14, 15, 48 **ISO27034**: 7.3.6 **MSSDL**: Practices 9, 10 **OWASPTEST**: Phase 3.2, Phase 4.1 **PCISSLRAP**: 4.1 **SAMM15**: IR1-B, IR2-A, IR2-B **SCAGILE:** Operational Security Tasks 4, 7 **SCFPSSD**: Use Code Analysis Tools to Find Security Issues Early, Use Static Analysis Security Testing Tools, Perform Manual Verification of Security Features/Mitigations **SCSIC:** Peer Reviews and Security Testing **SP80053:** SA-11, SA-15 **SP80064**: 3.2.3.6 **SP800181:** SP-DEV-001, SP-DEV-002;

Practices	Tasks	Implementation Examples	References
		that developers can access and search.	T0013, T0111, T0176, T0267, T0516; K0009, K0039, K0070, K0140, K0624; S0019, S0060, S0078, S0137, S0149, S0167, S0174, S0242, S0266; A0007, A0015, A0036, A0044, A0047
Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8): Help identify vulnerabilities before software is released so they can be corrected before release, which prevents exploitation. Using automated methods lowers the effort and resources needed to detect vulnerabilities. Executable code is binaries, directly executed bytecode, directly executed source code, and any other form of code an organization deems as executable.	**PW.8.1:** Determine if executable code testing should be performed and, if so, which types should be used.	• Follow the organization's policies or guidelines for when code testing should be performed and how it should be conducted.	**BSA**: TV.3 **SCSIC:** Peer Reviews and Security Testing **SP80053:** SA-11 **SP800181:** SP-DEV-001, SP-DEV-002; T0456; K0013, K0039, K0070, K0153, K0165, K0342, K0367, K0536, K0624; S0001, S0015, S0026, S0061, S0083, S0112, S0135
	PW.8.2: Design the tests, perform the testing, and document the results.	• Perform robust functional testing of security features. • Integrate dynamic vulnerability testing into the project's automated test suite. • Incorporate tests for previously reported vulnerabilities into the project's automated test suite to ensure that errors are not reintroduced. • Use automated fuzz testing tools to find issues with input handling by native code. • Use penetration testing to simulate how an attacker might attempt to compromise the software only in high-risk scenarios if resources are available. • Use automated tools to identify and remediate documented and verified unsafe software practices on a continuous basis as executable code is checked into the code repository. • Identify and document the root cause of each discovered issue. • Document lessons learned from code testing in a knowledge base that developers can access and search.	**BSA**: PD.1-5, TV.3, TV.5, TV.5-2 **BSIMM9**: PT1.1, PT1.2, PT1.3, ST1.1, ST1.3, ST2.1, ST2.4, ST2.5, ST2.6, ST3.3, ST3.4 **IDASOAR**: Fact Sheets 7, 8, 10, 11, 38, 39, 43, 44, 48, 55, 56, 57 **ISO27034**: 7.3.6 **MSSDL**: Practice 11 **PCISSLRAP**: 4.1 **SAMM15**: ST1-B, ST2-A, ST2-B **SCAGILE:** Operational Security Tasks 10, 11; Tasks Requiring the Help of Security Experts 4, 6, 7 **SCFPSSD**: Perform Dynamic Analysis Security Testing, Fuzz Parsers, Network Vulnerability Scanning, Perform Automated Functional Testing of Security Features/Mitigations, Perform Penetration Testing **SCSIC:** Peer Reviews and Security Testing **SP80053:** SA-11, SA-15 **SP80064**: 3.2.3.6 **SP800181:** SP-DEV-001, SP-DEV-002; T0013, T0028, T0169, T0176, T0253,

Practices	Tasks	Implementation Examples	References
			T0266, T0456, T0516; K0009, K0039, K0070, K0272, K0339, K0342, K0362, K0536, K0624; S0001, S0015, S0046, S0051, S0078, S0081, S0083, S0135, S0137, S0167, S0242; A0015
Configure the Software to Have Secure Settings by Default (PW.9): Help improve the security of the software at installation time, which reduces the likelihood of the software being deployed with weak security settings that would put it at greater risk of compromise.	**PW.9.1:** Determine how to configure each setting that has an effect on security so the default settings are secure and they do not weaken the security functions provided by the platform, network infrastructure, or services.	• Conduct testing to ensure the settings are working as expected and not inadvertently causing any security weaknesses, operational issues, or other problems.	**BSA**: CF.1, TC.1 **IDASOAR:** Fact Sheet 23 **ISO27034**: 7.3.5 **OWASPSCP:** System Configuration **OWASPTEST**: Phase 4.2 **SCAGILE:** Tasks Requiring the Help of Security Experts 12 **SCSIC:** Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls **SP800181**: SP-DEV-002; K0009, K0039, K0073, K0153, K0165, K0275, K0531; S0167
	PW.9.2: Implement the default settings and document each setting for software administrators.	• Verify the approved configuration is in place for the software. • Document each setting's purpose, options, default value, security relevance, potential operational impact, and relationships with other settings. • Document how each setting can be implemented by software administrators.	**IDASOAR:** Fact Sheet 23 **OWASPSCP:** System Configuration **OWASPTEST**: Phase 4.2 **PCISSLRAP**: 8.1, 8.2 **SCAGILE:** Tasks Requiring the Help of Security Experts 12 **SCFPSSD**: Verify Secure Configurations and Use of Platform Mitigation **SCSIC:** Vendor Software Delivery Integrity Controls, Vendor Software Development Integrity Controls **SP800181**: SP-DEV-001; K0009, K0039, K0073, K0153, K0165, K0275, K0531

Practices	Tasks	Implementation Examples	References
Respond to Vulnerability Reports (RV)			
Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1): Help ensure vulnerabilities are identified more quickly so they can be remediated more quickly, reducing the window of opportunity for attackers.	**RV.1.1:** Gather information from consumers and public sources on potential vulnerabilities in the software and any third-party components the software uses, and investigate all credible reports.	• Establish a vulnerability response program, and make it easy for security researchers to learn about your program and report possible vulnerabilities. • Monitor vulnerability databases, security mailing lists, and other sources of vulnerability reports through manual or automated means.	**BSA**: VM.1-3, VM.3 **BSIMM9**: CMVM1.2, CMVM3.4 **PCISSLRAP**: 3.4, 4.1, 9.1 **SAMM15**: IM1-A **SCAGILE:** Operational Security Task 5 **SCTPC:** 3.2.4 **SP800181**: K0009, K0038, K0040, K0070, K0161, K0362; S0078
	RV.1.2: Periodically review, analyze, and/or test the software's code to identify previously undetected vulnerabilities.	• Configure the toolchain to perform automated code analysis and testing on a regular basis. • [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] • [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]	**BSA**: VM.1-2 **ISO27034**: 7.3.6 **PCISSLRAP**: 3.4, 4.1 **SP800181**: SP-DEV-002; K0009, K0039, K0153 [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]
	RV.1.3: Have an incident response capability to coordinate response to vulnerability reports.	• Have a policy that addresses vulnerability disclosure and remediation, and implement the processes needed to support that policy. • Have a security response playbook to handle a generic reported vulnerability, a report of zero-days, a vulnerability being exploited in the wild, and a major ongoing incident involving multiple parties.	**BSA**: VM.1-1, VM.2, VM.2-3 **MSSDL:** Practice 12 **SAMM15:** IM1-B, IM2-A, IM2-B **SCFPSSD:** Vulnerability Response and Disclosure **SP800160**: 3.3.8 **SP800181**: K0041, K0042, K0151, K0292, K0317; S0054; A0025

Practices	Tasks	Implementation Examples	References
Assess and Prioritize the Remediation of All Vulnerabilities (RV.2): Help ensure vulnerabilities are remediated as quickly as necessary, reducing the window of opportunity for attackers.	**RV.2.1:** Analyze each vulnerability which is not being exploited to determine how much effort would be required to remediate it, what the potential impact of vulnerability exploitation would be, what resources are required to weaponize the vulnerability (with the assumption that the vulnerability will be exploited in the near future), and how vulnerability remediation should be prioritized, along with any other relevant factors.	• Use issue tracking or bug tracking software to document each vulnerability.	**BSA**: VM.2, VM.2-1, VM.2-2 **PCISSLRAP**: 4.2 **SCAGILE:** Tasks Requiring the Help of Security Experts 10 **SP80053**: SA-10 **SP800160**: 3.3.8 **SP800181**: K0009, K0039, K0070, K0161, K0165; S0078
Analyze Vulnerabilities to Identify Their Root Causes (RV.3): Help reduce the frequency of vulnerabilities in the future.	**RV.3.1:** Analyze all identified vulnerabilities to determine the root cause of each vulnerability.	• Document the root cause of each discovered issue. • Document lessons learned from root cause analysis in a knowledge base that developers can access and search.	**BSA**: VM.2.1 **PCISSLRAP**: 4.2 **SAMM15**: IM3-A **SP800181**: T0047, K0009, K0039, K0070, K0343
	RV.3.2: Analyze the root causes over time to identify patterns, such as when a particular secure coding practice not being followed consistently.	• Document lessons learned from root cause analysis in a knowledge base that developers can access and search.	**BSA**: VM.2-1, PD.1-3 **MSSDLPG52**: Phase Two: Design **PCISSLRAP**: 4.2 **SP800160**: 3.3.8 **SP800181**: T0111, K0009, K0039, K0070, K0343
	RV.3.3: Review the software for other instances of the reported problem and fix them proactively rather than waiting for external reports.	• [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] • [See **Create Source Code Adhering to Secure Coding Practices (PW.5)**]	**BSA**: VM.2 **PCISSLRAP**: 4.2 **SP800181**: SP-DEV-001, SP-DEV-002; K0009, K0039, K0070
	RV.3.4: Review the SDLC process and update it as appropriate to prevent (or reduce the likelihood of) the root cause recurring in updates to this software or in new software that is created.	• Document lessons learned from root cause analysis in a knowledge base that developers can access and search. • Plan and implement changes to the appropriate SSDF practices.	**BSA**: PD.1-3 **BSIMM9**: CMVM3.2 **MSSDL**: Practice 2 **PCISSLRAP**: 2.6, 4.2 **SP800181**: K0009, K0039, K0070

190

191 References

[1] Newhouse W, Keith S, Scribner B, Witte G (2017) National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-181. https://doi.org/10.6028/NIST.SP.800-181

[2] National Institute of Standards and Technology (2018), Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. (National Institute of Standards and Technology, Gaithersburg, MD). https://doi.org/10.6028/NIST.CSWP.04162018

[3] McGraw G, Migues S, West J (2018) *Building Security In Maturity Model (BSIMM) Version 9*. Available at https://www.bsimm.com/download/

[4] International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), Information technology – Security techniques – Application security – Part 1: Overview and concepts, ISO/IEC 27034-1:2011, 2011. Available at https://www.iso.org/standard/44378.html

[5] Microsoft (2019) *Security Development Lifecycle*. Available at https://www.microsoft.com/en-us/sdl

[6] Open Web Application Security Project (2010) *OWASP Secure Coding Practices Quick Reference Guide*. Available at https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

[7] Open Web Application Security Project (2014) *OWASP Testing Guide 4.0*. Available at https://www.owasp.org/images/1/19/OTGv4.pdf

[8] Payment Card Industry (PCI) Security Standards Council (2019) *Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures*. Available at https://www.pcisecuritystandards.org/document_library?category=sware_sec#results

[9] Open Web Application Security Project (2017) *Software Assurance Maturity Model Version 1.5*. Available at https://www.owasp.org/index.php/OWASP_SAMM_Project

[10] Software Assurance Forum for Excellence in Code (2018) *Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program, Third Edition*. Available at https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

[11] Joint Task Force Transformation Initiative (2013) Security and Privacy Controls for Federal Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Revision 4, Includes updates as of January 22, 2015. https://doi.org/10.6028/NIST.SP.800-53r4

[12] Kissel R, Stine K, Scholl M, Rossman H, Fahlsing J, Gulick J (2008) Security Considerations in the System Development Life Cycle. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-64 Revision 2. https://doi.org/10.6028/NIST.SP.800-64r2

[13] Ross R, McEvilley M, Oren J (2016) Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-160, Volume 1, Includes updates as of March 21, 2018. https://doi.org/10.6028/NIST.SP.800-160v1

[14] Software Assurance Forum for Excellence in Code (2010) *Software Integrity Controls: An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain*. Available at http://www.safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf

[15] Software Assurance Forum for Excellence in Code (2012) *Practical Security Stories and Security Tasks for Agile Development Environments*. Available at http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf

[16] Hong Fong EK, Wheeler D, Henninger A (2016) State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016. (Institute for Defense Analyses [IDA], Alexandria, VA), IDA Paper P-8005. Available at http://www.acq.osd.mil/se/docs/P-8005-SOAR-2016.pdf

[17] Software Assurance Forum for Excellence in Code (2017) *Tactical Threat Modeling*. Available at https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf

[18] Software Assurance Forum for Excellence in Code (2017) *Managing Security Risks Inherent in the Use of Third-Party Components*. Available at https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf

[19] BSA (2019) *Framework for Secure Software.* Available at https://www.bsa.org/reports/bsa-framework-for-secure-software

193
Appendix A—Acronyms

BSIMM	Building Security In Maturity Model
COTS	Commercial-Off-the-Shelf
CPS	Cyber-Physical System
DevOps	Development and Operations
GOTS	Government-Off-the-Shelf
ICS	Industrial Control System
IDA	Institute for Defense Analyses
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISO	International Organization for Standardization
ISPAB	Information Security and Privacy Advisory Board
IT	Information Technology
ITL	Information Technology Laboratory
KPI	Key Performance Indicator
NICE	National Initiative for Cybersecurity Education
NIST	National Institute of Standards and Technology
OWASP	Open Web Application Security Project
PCI	Payment Card Industry
SAFECode	Software Assurance Forum for Excellence in Code
SAMM	Software Assurance Maturity Model
SBOM	Software Bill of Materials
SDL	[Microsoft] Security Development Lifecycle
SDLC	Software Development Life Cycle
SLC	Software Lifecycle
SOAR	State-of-the-Art Resources
SSDF	Secure Software Development Framework

194