NIST Special Publication 800-218

Secure Software Development Framework (SSDF) Version 1.1:

Recommendations for Mitigating the Risk of Software Vulnerabilities

Murugiah Souppaya
Computer Security Division
Information Technology Laboratory

Karen Scarfone
Scarfone Cybersecurity
Clifton, VA

Donna Dodson*
* *Former NIST employee; all work for this publication was done while at NIST.*

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-218

February 2022

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce for Standards and Technology & Director, National Institute of Standards and Technology

Executive Summary

This document describes a set of fundamental, sound practices for secure software development called the Secure Software Development Framework (SSDF). Organizations should integrate the SSDF throughout their existing software development practices, express their secure software development requirements to third-party suppliers using SSDF conventions, and acquire software that meets the practices described in the SSDF. Using the SSDF helps organizations to meet the following secure software development recommendations:

- Organizations should ensure that their people, processes, and technology are prepared to perform secure software development.
- Organizations should protect all components of their software from tampering and unauthorized access.
- Organizations should produce well-secured software with minimal security vulnerabilities in its releases.
- Organizations should identify residual vulnerabilities in their software releases and respond appropriately to address those vulnerabilities and prevent similar ones from occurring in the future.

The SSDF does not prescribe how to implement each practice. The focus is on the outcomes of the practices rather than on the tools, techniques, and mechanisms to do so. This means that the SSDF can be used by organizations in any sector or community, regardless of size or cybersecurity sophistication. It can also be used for any type of software development, regardless of technology, platform, programming language, or operating environment.

The SSDF defines only a high-level subset of what organizations may need to do, so organizations should consult the references and other resources for additional information on implementing the practices. Not all practices are applicable to all use cases; organizations should adopt a risk-based approach to determine what practices are relevant, appropriate, and effective to mitigate the threats to their software development practices.

Organizations can communicate how they are addressing the clauses from Section 4 of the President's Executive Order (EO) on "Improving the Nation's Cybersecurity (14028)" by referencing the SSDF practices and tasks described in Appendix A.

Table of Contents

List of Tables

1 Introduction

A *software development life cycle (SDLC)*[1] is a formal or informal methodology for designing, creating, and maintaining software (including code built into hardware). There are many models for SDLCs, including waterfall, spiral, agile, and – in particular – agile combined with software development and IT operations (DevOps) practices. Few SDLC models explicitly address software security in detail, so secure software development practices usually need to be added to and integrated into each SDLC model. Regardless of which SDLC model is used, secure software development practices should be integrated throughout it for three reasons: to reduce the number of vulnerabilities in released software, to reduce the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and to address the root causes of vulnerabilities to prevent recurrences. Vulnerabilities include not just bugs caused by coding flaws, but also weaknesses caused by security configuration settings, incorrect trust assumptions, and outdated risk analysis. [IR7864]

Most aspects of security can be addressed multiple times within an SDLC, but in general, the earlier in the SDLC that security is addressed, the less effort and cost is ultimately required to achieve the same level of security. This principle, known as *shifting left*, is critically important regardless of the SDLC model. Shifting left minimizes any technical debt that would require remediating early security flaws late in development or after the software is in production. Shifting left can also result in software with stronger security and resiliency.

There are many existing documents on secure software development practices, including those listed in the References section. This document does not introduce new practices or define new terminology. Instead, it describes a set of high-level practices based on established standards, guidance, and secure software development practice documents. These practices, collectively called the Secure Software Development Framework (SSDF), are intended to help the target audiences achieve secure software development objectives. Many of the practices directly involve the software itself, while others indirectly involve it (e.g., securing the development environment).

Future work may expand on this publication and potentially cover topics such as how the SSDF may apply to and vary for particular software development methodologies and associated practices like DevOps, how an organization can transition from their current software development practices to also incorporating the SSDF practices, and how the SSDF could be applied in the context of open-source software. Future work will likely take the form of use cases so that the insights will be more readily applicable to specific types of development environments, and it will likely include collaboration with the open-source community and other groups and organizations.

This document identifies secure software development practices but does not prescribe how to implement them. The focus is on the outcomes of the practices to be implemented rather than on

[1] Note that SDLC is also widely used for "system development life cycle." All usage of "SDLC" in this document is referencing software, not systems.

the tools, techniques, and mechanisms used to do so. 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 practices today and define its future target practices 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)
- Can assist organizations that are procuring and using software to understand secure software development practices employed by their suppliers

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

- Business owners, software developers, project managers and leads, cybersecurity professionals, and operations and platform engineers within an organization who need to clearly communicate with each other about secure software development
- Software acquirers, including federal 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 significant security vulnerabilities)[3]

[2] The SSDF practices may help support the NIST Cybersecurity Framework Functions, Categories, and Subcategories, but the SSDF practices do not map to them and are typically the responsibility of different parties. Developers can adopt SSDF practices, and the outcomes of their work could help organizations with their operational security in support of the Cybersecurity Framework.

[3] Future work may provide more practical guidance for software acquirers on how they can leverage the SSDF in specific use cases.

- 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 acquirer 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 document's practices are not based on the assumption that all organizations have the same security objectives and priorities. Rather, the recommendations reflect that each software producer may have unique security assumptions, and each software acquirer may have unique security needs and requirements. While the aim is for each software producer to follow all applicable practices, the expectation is that the degree to which each practice is implemented and the formality of the implementation 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.

Although most of these practices are relevant to any software development effort, some are not. 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. Some practices are foundational, while others are more advanced and depend on certain foundational practices already being in place. Also, practices are not all equally important for all cases.

Factors such as risk, cost, feasibility, and applicability should be considered when deciding which practices to use and how much time and resources to devote to each practice.[4] Automatability is also an important factor to consider, especially for implementing practices at scale. The practices, tasks, and implementation examples represent a starting point to consider; they are meant to be changed and customized, and they are not prioritized. Any stated frequency for performing practices is notional. The intention of the SSDF is not to create a checklist to follow, but to provide a basis for planning and implementing a risk-based approach to adopting secure software development practices.

The responsibility for implementing the practices may be distributed among different organizations based on the delivery of the software and services (e.g., infrastructure as a service, software as a service, platform as a service, container as a service, serverless). In these situations, it likely follows a shared responsibility model involving the platform/service providers and the tenant organization that is consuming those platforms/services. The tenant organization should establish an agreement with the providers that specifies which party is responsible for each practice and task and how each provider will attest to their conformance with the agreement.

[4] Organizations seeking guidance on how to get started with secure software development can consult many publicly available references, such as "SDL That Won't Break the Bank" by Steve Lipner from SAFECode (https://i.blackhat.com/us-18/Thu-August-9/us-18-Lipner-SDL-For-The-Rest-Of-Us.pdf), "Application Software Security and the CIS Controls: A Reference Paper" by Steve Lipner and Stacy Simpson from SAFECode (https://safecode.org/resource-publications/cis-controls/), and "Simplified Implementation of the Microsoft SDL" by Microsoft (https://www.microsoft.com/en-us/download/details.aspx?id=12379).

2 The Secure Software Development Framework

This document defines version 1.1 of the Secure Software Development Framework (SSDF) with fundamental, sound, and secure recommended practices based on established secure software development practice documents. The practices are organized into four groups:

1. **Prepare the Organization (PO):** Organizations should ensure that their people, processes, and technology are prepared to perform secure software development at the organization level. Many organizations will find some PO practices to also be applicable to subsets of their software development, like individual development groups or projects.
2. **Protect the Software (PS):** Organizations should protect all components of their software from tampering and unauthorized access.
3. **Produce Well-Secured Software (PW):** Organizations should produce well-secured software with minimal security vulnerabilities in its releases.
4. **Respond to Vulnerabilities (RV):** Organizations should identify residual vulnerabilities in their software releases and respond appropriately to address those vulnerabilities and prevent similar ones from occurring in the future.

Each practice definition includes the following elements:

- **Practice:** The name of the practice and a unique identifier, followed by a brief explanation of what the practice is and why it is beneficial
- **Tasks:** One or more actions that may be needed to perform a practice
- **Notional Implementation Examples:** One or more notional examples of types of tools, processes, or other methods that could be used to help implement a task. No examples or combination of examples are required, and the stated examples are not the only feasible options. Some examples may not be applicable to certain organizations and situations.
- **References:** Pointers to one or more established secure development practice documents and their mappings to a particular task. Not all references will apply to all instances of software development.

Table 1 defines the practices. They are only a **subset** of what an organization may need to do. The information in the table is space constrained; much more information on each practice can be found in the references. Note that the order of the practices, tasks, and notional implementation examples in the table is not intended to imply the sequence of implementation or the relative importance of any practice, task, or example.

The table uses terms like “sensitive data,” “qualified person,” and “well-secured,” which are not defined in this publication. Organizations adopting the SSDF should define these terms in the context of their own environments and use cases. The same is true for the names of environments, like “development,” “build,” “staging,” “integration,” “test,” “production,” and “distribution,” which vary widely among organizations and projects. Enumerating your environments is necessary in order to secure them properly, and especially to prevent lateral movement of attackers from environment to environment.