The Security Development LifeCycle

Article01/16/2024

Introduction

The Security Development Lifecycle (or SDL) is a process that Microsoft has adopted for the development of software that needs to withstand malicious attack. The process encompasses the **addition of a series of security-focused activities and deliverables to each of the phases** of Microsoft's software development process. These activities and deliverables include the development of threat models during software design, the use of static analysis code-scanning tools during implementation and the conduct of code reviews and security testing during a focused "security push."
The SDL involves modifying a software development organization's processes by integrating measures that lead to improved software security: the intention of these modifications is not to totally overhaul the process, but rather to add well-defined security checkpoints and security deliverables.
Figure 1 depicts the seven phases that define the SDL process.

Expand table

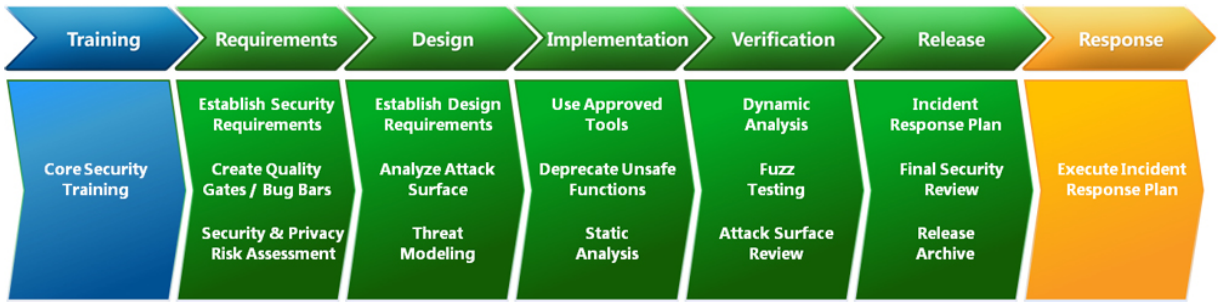

Figure 1: the seven phases of the Security Development Lifecycle Process.

↑ Return to top

Phase 1: Core Security Training

This step is a **prerequisite** for implementaing the SDL: individuals in technical roles (developers, testers, and program managers) who are directly involved with the development of software programs must attend at least one unique security training class each year.
By allowing individuals involved with the development of software programs to stay informed about security basics and latest trends in security and privacy, their commitment to writing more secure software will be increased.
Basic core security training should cover foundational concepts such as:

- **Secure Design**, including:
  - attack surface reduction;
  - defense in depth;
  - principle of least privilege;
  - secure defaults.
- **Threat Modeling**, including:
  - overview of threat modeling;
  - design implications of a threat model;
  - coding constraints based on a threat model.
- **Secure Coding**, including:
  - buffer overruns (for applications using C and C++);
  - integer arithmetic errors (for applications using C and C++);
  - cross-site scripting (for managed code and web applications);
  - SQL injection (for managed code and web applications);
  - weak cryptography.
- **Security Testing**, including:
  - differences between security testing and functional testing;
  - risk assessment;
  - security testing methods.
- **Privacy**, including:
  - types of privacy-sensitive data;
  - privacy design best practices;
  - risk assessment;
  - privacy development best practices;
  - privacy testing best practices.

↑ Return to top

Phase 2: Requirements

The Requirements phase of the SDL includes the **project inception** (when you consider security and privacy at a foundational level) and a **cost analysis** (when you determine if development and support costs for improving security and privacy are consistent with business needs).
This phase includes the following practices

- **Establish Security Requirements**: assigning security experts, defining minimum security and privacy criteria for the application, deploying a security vulnerability/ work item tracking system allowing for creation, triage, assignment, tracking, remediation and reporting of software vulnerabilities.
- **Create Quality Gates/bug Bars**: a bug bar is a **quality gate** that applies to the entire software development project and **defines the severity thresholds of security vulnerabilities** (for example, no known vulnerabilities in the application with a “critical” or “important” rating at time of release). The bug bar, once set, should never be relaxed.
- **Security and Privacy Assessment**: Security risk assessments (SRAs) and privacy risk assessments (PRAs) identify functional aspects of the software that require closer review.

↑ Return to top

Phase 3: Design

The Design phase is when you build the plan for how you will take your project through the rest of the SDL process: from implementation, to verification, to release. During the Design phase you **establish best practices** to follow for this phase by way of functional and design specifications and you **perform risk analysis** to identify threats and vulnerabilities in your software.
This phase includes the following practices

- **Establish Design Requirements**: creation of security and privacy design specifications, specification review and specification of minimal cryptographic design requirements.
- **Attack Surface Analysis/Reduction**: a thorough analysis will provide better awareness of overall product attack surface: with this information, design considerations should be put in place to reduce attack surface, including but not

limited to disabling or restricting access to system services, applying the principle of least privilege and employing layered defenses where possible.
- **Threat Modeling**: this practice is a process by which you can understand security threats to a system, determine risks from those threats and establish appropriate mitigations.

↑ Return to top

Phase 4: Implementation

The Implementation phase is when the end user of your software is foremost in your mind.
During this phase you **create the documentation and tools** the customer uses to make informed decisions about how to deploy your software securely. To this end, the Implementation phase is when you **establish development best practices to detect and remove** security and privacy issues early in the development cycle.
This phase includes the following practices

- **Use Approved Tools**: define and publish a list of approved tools and associated security checks, such as compiler/linker options and warnings. Regularly update the list with the latest versions of the tools.
- **Deprecate Unsafe Functions**: determine the list of banned functions, use header files, newer compliers, or code scanning tools to check code for the existence of banned functions and replace those banned functions with safer alternatives.
- **Perform Static Analysis**: analyze the source code prior to compile.

↑ Return to top

Phase 5: Verification

During the Verification phase, you **ensure that your code meets the security and privacy tenets** you established in the previous phases. This is done through security and privacy testing, and a security push, which is a team-wide focus on threat model updates, code review, testing, and thorough documentation review and edit. A **public release privacy review** is also completed during the Verification phase.
This phase includes the following practices

- **Dynamic Analysis**: run-time verification of your software, leveraging tools which

monitor application behavior for memory corruption, user privilege issues and other critical security problems.
- **Fuzz Testing**: a specialized form of dynamic analysis that induces program failure by deliberately introducing malformed or random data to an application.
- **Attack Surface Review**: ensures any design or implementation changes to the system have been taken into account and that any new attack vectors created as a result of the changes have been reviewed and mitigated including threat models.

↑ Return to top

Phase 6: Release

The Release phase is when you **ready your software for public consumption** and, perhaps more importantly, you ready yourself and your team for what happens once your software is in the hands of the user. One of the core concepts in the Release phase is **planning** (mapping out a plan of action, should any security or privacy vulnerabilities be discovered in your release) and this carries over to post-release, as well, in terms of response execution. To this end, a **Final Security Review and privacy review** is required prior to release.
This phase includes the following practices

- **Incident Response Plan**: the IRP identifies the appropriate points of contact in case of a security emergency and includes security servicing plans for code inherited from other groups within the organization and for licensed third-party code.
- **Final Security Review**: the FSR is a deliberate examination of all security activities performed on software prior to release: it includes an examination of threat models, tools outputs and performance against the quality gates and bug bars defined during the Requirements Phase. The FSR results in one of three different outcomes: Passed FSR, Passed FSR with exceptions, FSR with escalation.
- **Release/Archive**: certify that the project team has satisfied the security and privacy requirements prior to software release and archive all pertinent information and data, including specifications, source code, binaries, private symbols, threat models, documentation, emergency response plans and license and servicing terms for any third-party software.

↑ Return to top

Phase 7: Response

After a software program is released, the product development team must be available to respond to any possible security vulnerabilities or privacy issues that warrant a response. In addition, **develop a response plan** that includes preparations for potential post-release issues.
This phase includes the following practice

- **Execute Incident Response Plan**: programs with no known vulnerabilities at the time of release can be subject to new threats that emerge over time. The Microsoft Security Response Center (MSRC) identifies, monitors, resolves and responds to security incidents and Microsoft software security vulnerabilities. Their goal is to protect customers by delivering security updates and authoritative security guidance.

↑ Return to top

Community Resources

MSDN Pages

- Security Development Lifecycle (SDL) Banned Function Calls

Blogs

- The Security Development Lifecycle Blog

Forums

- Microsoft Security Development Lifecycle (SDL) - A forum for discussing the Microsoft SDL Process and guidance.

Papers

- Essential Software Security Training for the Microsoft SDL - This paper discusses why a commitment to software security training is a key tenet of the Microsoft Security Development Lifecycle (SDL) and vital to ensuring that secure software can

take its place as a top priority along with software features and delivery timelines.
- Simplified Implementation of the Microsoft SDL - This document illustrates the core concepts of the Microsoft Security Development Lifecycle (SDL) and discusses the individual security activities that should be performed in order to follow the SDL process.

Technical Articles

- A Look Inside the Security Development Lifecycle at Microsoft by Michael Howard (MSDN Magazine, November 2005)

See Also

NOTE: these links are external to TechNet Wiki.

Books

- The Security Development Lifecycle by Michael Howard, Steve Lipner (Microsoft Press, May 2006)
- Writing Secure Code, 2nd Edition by Michael Howard, David LeBlanc (Microsoft Press, December 2004)

Social Media

- Microsoft SDL on Twitter