Understand Security Considerations for Application Lifecycle Management Solutions

100 XP

8 minutes

The **Microsoft Security Development Lifecycle (SDL)** introduces security and privacy considerations throughout all phases of the development process. It helps developers build highly secure software, address security compliance requirements, and reduce development costs. The guidance, best practices, tools, and processes in the SDL are practices used internally at Microsoft to build more secure products and services.

Since first sharing the SDL in 2008, the practices have been continuously updated to cover new scenarios such as cloud services, IoT, and AI.

Provide training

Security is everyone's job. Developers, service engineers, and program and project managers must understand security basics. They all must know how to build security into software and services to make products more secure, while still addressing business needs and delivering user value. Effective training will complement and reinforce security policies, SDL practices, standards, and requirements of software security, and be guided by insights derived through data or newly available technical capabilities.

Although security is everyone's job, it's important to remember that not everyone needs to be a security expert nor strive to become a proficient penetration tester. However, ensuring everyone understands the attacker's perspective, their goals, and the art of the possible will help capture the attention of everyone and raise the collective knowledge bar.

Define security requirements

Security and privacy is a fundamental aspect of developing highly secure applications and systems. Regardless of development methodology in use, security requirements must be updated continuously in order to address changes in required functionality and changes to the threat landscape. The optimal time to define the security requirements is during the initial design and planning stages. Early planning allows development teams to integrate security in ways that minimize disruption.

Factors that influence security requirements include, but are not limited to:

- Legal and industry requirements
- Internal standards and coding practices
- Review of previous incidents
- Known threats

These requirements should be tracked through a work-tracking system, or through telemetry that is derived from the engineering pipeline.

Define metrics and compliance reporting

It's essential for an organization to define the minimum acceptable levels of security quality, and to hold engineering teams accountable to meeting that criteria. Defining these expectations early helps a team understand the risks that are associated with security issues, identify and fix security defects during development, and apply the standards throughout the entire project. Setting a meaningful security bar involves clearly defining the severity thresholds of security vulnerabilities, and helps to establish a plan of action when vulnerabilities are encountered. For example, all known vulnerabilities discovered with a "critical" or "important" severity rating must be fixed with a specified time frame.

To track key performance indicators (KPIs) and ensure security tasks are completed, bug tracking and/or work tracking mechanisms used by an organization (such as Azure DevOps) should allow for security defects and security work items to be clearly labeled as security, and marked with their appropriate security severity. This tracking allows for accurate tracking and reporting of security work.

You can read more about defining metrics and compliance reporting at:

- SDL Privacy Bug Bar Sample
- Add or modify an Azure DevOps field to track work
- SDL Security Bug Bar Sample

Perform threat modeling

Threat modeling should be used in environments where there is a meaningful security risk. As a practice, it allows development teams to consider, document, and discuss the security implications of designs in the context of their planned operational environment, and in a structured fashion. Applying a structured approach to threat scenarios helps a team more effectively and less expensively identify security vulnerabilities, determine risks from those threats, and then make security feature selections and establish appropriate mitigations. You can apply threat modeling at the component, application, or system level.

More information is available at Threat Modeling.

Establish design requirements

The SDL is typically thought of as assurance activities that help engineers implement more secure features, meaning the features are well engineered for security. To achieve this assurance, engineers typically rely on security features such as cryptography, authentication, and logging. In many cases, selecting or implementing security features has proven to be so complicated that design or implementation choices are likely to result in vulnerabilities. Therefore, it's crucial that they are applied consistently and with a consistent understanding of the protection they provide.

Define and use cryptography standards

With the rise of mobile and cloud computing, it's important to ensure all data - including security-sensitive information and management and control data - are protected from unintended disclosure or alteration when it's being transmitted or stored. Encryption is typically used to achieve this protection. However, making an incorrect choice when using any aspect of cryptography can be catastrophic. Therefore, it's best to develop clear encryption standards that provide specifics on every element of the encryption implementation.

Encryption should be left to experts. A good general rule is to only use industry-vetted encryption libraries and ensure they're implemented in a way that allows them to be easily replaced if needed.

For more information on encryption, see the Microsoft SDL Cryptographic Recommendations whitepaper.

Manage security risks from using third-party components

The vast majority of software projects today are built using third-party components (both commercial and open source). When selecting which third-party components to use, it's important to understand the impact that a security vulnerability in them could have to the security of the larger system into which they are integrated. Having an accurate inventory of these components, and a plan to respond when new vulnerabilities are discovered, will go a long way toward mitigating risks. However, you should also consider additional validation, depending on your organization's risk tolerance, the type of component being used, and potential impact of a security vulnerability.

Learn more about managing the security risks of using third-party components at:

- Managing Security Risks Inherent in the Use of Third-Party Components
- Managing Security Risks Inherent in the Use of Open-Source Software

Use approved tools

Define and publish a list of approved tools and their associated security checks, such as compiler/linker options and warnings. Engineers should strive to use the latest version of approved tools (such as compiler versions), and to utilize new security analysis functionality and protections.

For more information, see:

- Recommended Tools, Compilers and Options for x86, x64, and ARM processors (whitepaper)
- SDL Resources

Perform Static Analysis Security Testing

Analyzing source code prior to compilation provides a highly scalable method of security code review, and helps ensure that secure coding policies are being followed. Static Analysis Security Testing (SAST) is typically integrated into the commit pipeline to identify vulnerabilities each time the software is built or packaged. However, some offerings integrate into the developer environment to spot certain flaws such as the existence of unsafe or other banned functions, and then replace those functions with safer alternatives while the developer is actively coding. There is no one-size-fits-all solution; development teams should decide the optimal frequency for performing SAST, and consider deploying multiple tactics to balance productivity with adequate security coverage.

More information is available at:

- Microsoft DevSkim on GitHub
- Roslyn Security Guard Rules
- Visual Studio Marketplace
- Analyzing C/C++ Code Quality by Using Code Analysis
- Microsoft BinSkim on GitHub

Perform Dynamic Analysis Security Testing

Performing run-time verification of your fully compiled or packaged software checks functionality that is only apparent when all components are integrated and running. This verification is typically achieved using a tool, a suite of pre-built attacks, or tools that specifically monitor application behavior for memory corruption, user privilege issues, and other critical security problems. Similar to SAST, there is no one-size-fits-all solution and while some tools (such as web app scanning tools) can be more readily integrated into the CI/CD pipeline, other Dynamic Application Security Testing (DAST) such as fuzzing requires a different approach.

More information is available at:

- Visual Studio Marketplace
- Automated Penetration Testing with White-Box Fuzzing

Perform penetration testing

Penetration testing is a security analysis of a software system that is performed by skilled security professionals who simulate the actions of a hacker. The objective of a penetration test is to uncover potential vulnerabilities resulting from coding errors, system configuration faults, or other operational deployment weaknesses. Penetration tests typically find the broadest variety of vulnerabilities, and are often performed in conjunction with automated and manual code reviews to provide a greater level of analysis than would ordinarily be possible.

More information is available at:

- Attack Surface Analyzer
- SDL Security Bug Bar Sample

Establish a standard incident response process

Preparing an incident response plan is crucial for addressing new threats that can emerge over time, and your plan should be created in coordination with your organization's dedicated Product Security Incident Response Team (PSIRT). Your incident response plan should:

- Include who to contact if a security emergency occurs
- Establish the protocol for security servicing (including plans for code inherited from other groups within the organization and for third-party code)
- Be tested before it is needed

For more information about incident responses, see:

- Using Azure Security Center for an incident response
- Microsoft Incident Response and shared responsibility for cloud computing
- Microsoft Security Response Center

By introducing standardized security and compliance considerations throughout all phases of the development process, developers can reduce the likelihood of vulnerabilities in products and services, and avoid repeating the same security mistakes. Similarly, security integration throughout the operations lifecycle will assist in maintaining the integrity of those products and services. Operational Security Assurance practices should align with your development processes; this arrangement will result in less time and cost spent on triage and response after the fact, and provide your customers with assurance that your products are highly secure.

Next unit: Summary

Continue