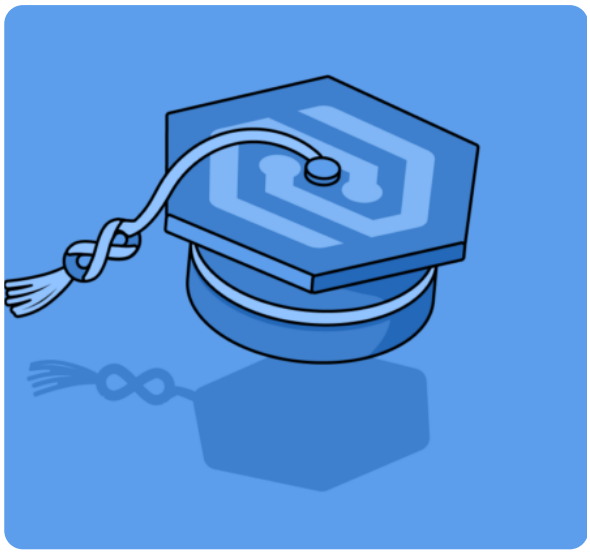
Mastering Software Development Lifecycle Security: Best Practices



LAST UPDATED: SEPTEMBER 1, 2024

10 MIN

Julie Peterson
Sr. Product Marketing Manager

In the ever-evolving landscape of software development, it's become absolutely *paramount* to ensure robust security measures throughout the Software Development Lifecycle (SDLC).

Need proof? **In the last three years alone, we've witnessed a surge of high-profile supply chain attacks including the SolarWinds breach, the Codecov compromise, and the breach of Nissan's Global Network.** Each of these have illuminated different (but equally important) vulnerabilities that can be exploited within the software supply chain, and have created urgency amongst security teams *and* developers to reevaluate and enhance their SDLC security practices.

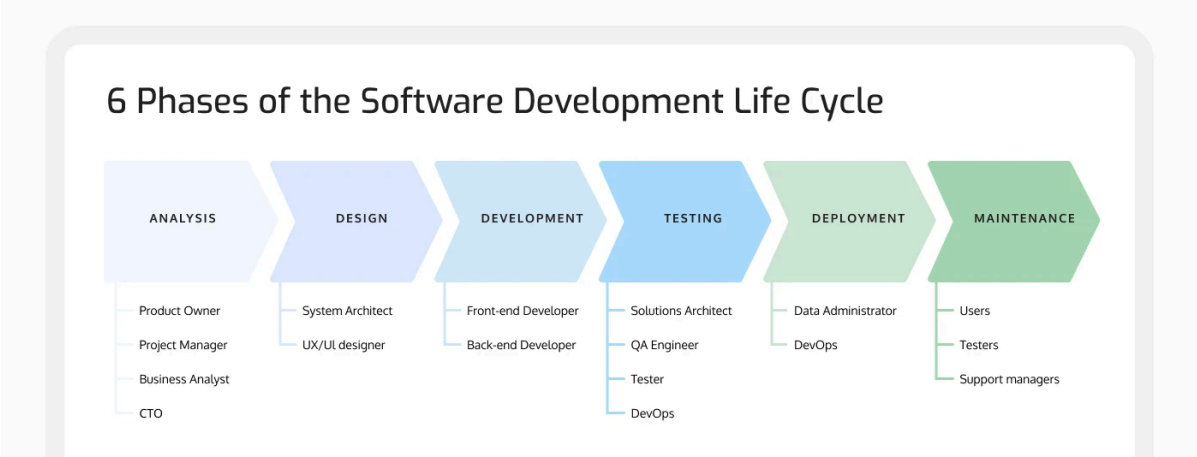
In this article, we explore the importance of SDLC security, highlight common vulnerabilities, and share strategies, best practices, and tools to help organizations

like yours ensure security at every stage of the development process.

But first, let's define what the software development lifecycle is...

What Is The Software Development Lifecycle ?

The Software Development Lifecycle is a systematic process or set of phases used in the development of software applications. It provides a structured approach to planning, creating, testing, deploying, and maintaining software.



The primary goal? **To produce high-quality software that meets or exceeds customer expectations, is delivered on time and within budget, and is easily maintainable.**

Security phase examples include:

1. Initiation and planning
2. Requirements analysis
3. Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance and continuous improvement

Software Supply Chain Vs. SDLC

Because we talk about software supply chain attacks in the context of SDLC security, it's essential to clarify the difference between the two.

In short, the software supply chain comprises **everything and everyone** that touches an organization's application in the SDLC, including people, processes, dependencies, and tools. It's a broader view of the entire ecosystem that covers software creation, distribution, and maintenance. Meanwhile, the SDLC is a *specific framework* for guiding the development and delivery of a particular software product.

Why Is SDLC Security Important?

Secure software development lifecycle processes arecrucial to protect against cyber threats and attacks, minimize the risk of data breaches, ensure compliance, and maintain customer trust.

But, given the number of attack vectors within and throughout the SDLC, most organizations have a significant gap in their software supply chain coverage. Consider how many developer accounts, repositories, or misconfigured tools organizations have that could be compromised or leaked. Historically, teams have relied on AppSec tools like Static Application Security Test (SAST) and Software Composition Analysis (SCA), **but even these leave modern organizations vulnerable**.

It's no wonder the majority of AppSec teams (71%) say today's attack surface is unmanageable.

That's precisely why Application Security Posture Management (ASPM) was introduced.

ASPM platforms like Cycode gives security and devs teams complete visibility and control of their risk posture throughout the software development lifecycle, across on-prem and cloud-based environments.

Common Security Vulnerabilities In The Software Development Lifecycle

Attack vectors in the SDLC include DevOps tools and infrastructure, developer accounts, insecure coding practices, code leaks, and more.

Vulnerability	Description	Examples	Potential Impact
DevOps Tools and Infrastructure	Tools like SCMs, build systems, and package repositories are often configured for efficiency rather than security.	– Default settings on CI/CD tools – Unmanaged dependencies	– Tool compromise leading to unauthorized access – Exposure of sensitive data
Code Tampering	Alteration or injection of malicious code at any stage in the CI/CD pipeline.	– Injecting malicious code during a build – Modifying security policies stored as code	– Introduction of backdoors – Compromised software integrity
Insecure Coding Practices	Failure to follow secure coding standards, exposing systems to risks.	– Hardcoded secrets – Misconfigured IaC templates	– Unauthorized access – Replication of insecure configurations across environments
Code Leaks	Unintended exposure of proprietary source code to	– Developer accidentally commits code to a public repo	– Exposure of sensitive information

	unauthorized parties.	– Malicious actor leaks code	– Increased attack surface for exploiting vulnerabilities

DevOps Tools and Infrastructure

Out of the box, components like SCMs, build systems, and package repositories are configured for release efficiency and velocity, **not security**.

More often than not, these tools are implemented outside the purview of security teams. Because of this, DevOps tooling could be compromised. For example, organizations with tools running on their default settings instead of being managed by consistent and rigorous security policies are vulnerable to attack.

Code Tampering

Code tampering occurs when source code is altered or malicious code is injected. It can happen at any point in the CI/CD pipeline, and preventing it requires organizations to monitor **all stages of the SDLC**. Moreover, trends like everything as code and GitOps now store other things as code that can be tampered, including security policies, infrastructure templates, build rules, and more. This expands the potential attack surface for code tampering beyond feature code and enables attackers to compromise more of the SDLC.

Insecure Coding Practices

Beyond software vulnerabilities, coding can introduce a number of risks.

Not following secure coding best practices exposes valuable resources and enables attackers to gain access to your systems. This includes such things as using hardcoded secrets, having infrastructure as code (IaC) misconfigurations, not standardizing on naming conventions, or storing sensitive information beyond secrets in code comments.

Hardcoded secrets become dangerous when API keys, encryption keys, tokens, or passwords are exposed, and misconfigurations in IaC templates risk being replicated across many cloud environments, effectively exposing application components or data to the public.

Code Leaks

Code leaks – proprietary source code being exposed publicly either accidentally by a developer copying code to the wrong repo or intentionally by a malicious actor – present a serious risk to any organization. Not only is code an incredibly valuable asset in itself, but a code leak could expose secrets and often widens the attack surface so that unauthorized users are able to gain access to internal systems or data.

Discover the top source code leaks from 2020-2024.

How To Integrate Security In The SDLC

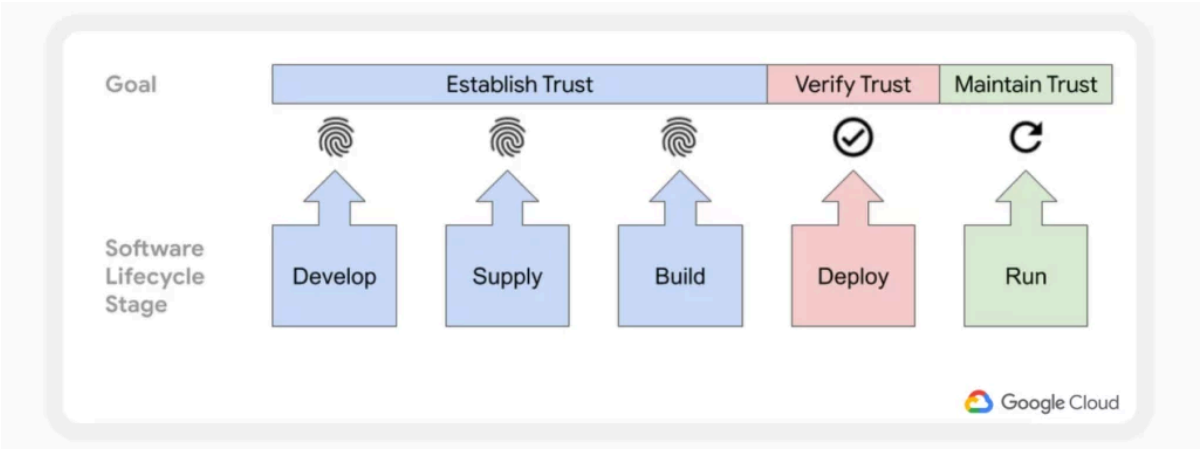
Generally speaking, the core components of a secure SDLC include:

1. 1. **Security planning**: Identify and document security requirements alongside functional requirements, with consideration for security features, access controls, and data protection requirements.
   2. **Security testing**: This includes static and dynamic analysis, penetration testing, and security scanning for vulnerabilities and misconfigurations
   3. **Secure deployment:** Ensure secure configuration of servers and environments, implement access controls, and and use secure deployment to minimize potential vulnerabilities
   4. **Maintenance and monitoring:** Regularly update and patch software to address known vulnerabilities, monitor and log security-related events, and implement incident response procedures
   5. **Training and awareness:** Train developers, testers, and other team members in secure coding practices, and raise awareness about security best practices company-wide. *Remember: security is a team sport!*

Frameworks like NIST's Secure Software Development Framework (SSDF) and Google's Supply Chain Levels for Software Artifacts (SLSA) have been developed to help organizations integrate security into the SDLC.

NIST's SSDF outlines four areas to guide secure software development: Preparing the organization, protecting the software, producing secure software, and responding to vulnerabilities.

Google's SLSA, created in collaboration with the OpenSSF, emphasizes the integrity of software artifacts throughout the supply chain and maps three goals to five software lifecycle stages.



The Role Of DevOps

DevOps plays a critical role in enhancing security throughout the SDLC. The integration of security practices into DevOps (often referred to as DevSecOps) is a holistic approach that emphasizes collaboration and communication between development, operations, and security teams. In particular, DevOps contributes to SDLC security by:

- Identifying and addressing security vulnerabilities at the earliest stages of development
- Promoting CI/CD practices, automating the build, testing, and deployment processes
- Defining and deploying infrastructure in a consistent and repeatable way
- Emphasizing and enabling cross-functional collaboration and communication
- Embracing continuous monitoring practices to detect and response to security incidents

6 SDLC Best Practices

To help ensure a secure SDLC and reduce the risk of security breaches, we recommend the following software development life cycle best practices:

1. Involve security experts from the beginning of the process

Incorporating security experts from the earliest stages of the SDLC ensures that security considerations are embedded into the design and architecture of the software. This proactive approach allows potential vulnerabilities to be identified and mitigated before they become entrenched in the codebase, reducing the risk of costly and time-consuming fixes later in the development process.

This also helps improve the relationship between security and development teams, something that 90% of AppSec teams say is needed.

2. Train developers on secure coding best practices

Educating developers on secure coding practices is essential for building software that is resilient against common vulnerabilities such as SQL injection, cross-site scripting (XSS), and buffer overflows. Regular training sessions, combined with up-to-date resources on the latest threats, empower developers to write code that adheres to security standards and reduces the attack surface of the application.

3.Adopt a strong SDLC governance program

A robust SDLC governance program establishes clear policies, procedures, and guidelines that ensure security is consistently prioritized across all phases of software development. This includes enforcing code reviews, security checkpoints, and adherence to compliance standards, which collectively contribute to maintaining the integrity and security of the software throughout its lifecycle.

4. Monitor the software for security vulnerabilities after it has been deployed to maintain good pipeline hygiene

Security doesn't end with deployment; continuous monitoring is crucial to detect and respond to vulnerabilities that may emerge in the production environment. Implementing real-time security monitoring and logging allows teams to quickly identify anomalies, potential breaches, or newly discovered vulnerabilities,

enabling rapid remediation to minimize damage and to maintain good pipeline hygiene.

5. Update the SDLC as new security threats emerge

The threat landscape is constantly evolving, making it essential to regularly update the SDLC to incorporate new security practices, tools, and threat intelligence. This includes revising security requirements, integrating new testing methodologies, and adapting to emerging threats, ensuring that the SDLC remains effective in mitigating risks.

6. Use security tools and technologies to help identify and mitigate risks

Leveraging automated security tools such as SAST and SCA allows organizations to identify vulnerabilities throughout the SDLC efficiently. These tools provide comprehensive coverage by scanning for known issues in code, configuration files, and third-party components, enabling teams to address risks early and reduce the likelihood of security breaches.

Looking for more secure software development best practices? Subscribe to our newsletter to get tips from experts every month.

How ASPM Helps With SDLC Security

While many tools can enhance security throughout the SDLC, only one offers complete protection and peace of mind.

ASPM is the only tool that offers continuous security *in and of* the pipeline. ASPM platforms automatically ingest data from multiple sources throughout the software lifecycle, giving security teams an ongoing, real-time view of their risk. This makes it faster and easier to detect, respond to, and remediate issues.

A complete ASPM solution is one that can provide you with a suite of application security testing tools like SCA and SAST, can deliver CI/CD security, and also ingest data from other third-party scanners.

How Cycode Helps With SDLC Security

Cycode's security-first, developer-friendly AppSec platform provides visibility, prioritization, and remediation for security, engineering, and DevOps teams throughout the software development lifecycle.

By offering a single, unified security platform that consolidates SAST, SCA, IaC scanning, pipeline security, secrets scanning, and code leak detection, Cycode gives security teams and developers peace of mind. In addition to our own suite of scanning tools, we can ingest data from third-party scanners to give you a full view of your application risk.

Learn more about hardcoded secrets detection, CI/CD security, and source code leakage detection now, or book a demo to see the platform in action.

Originally published: December 12, 2023