

Introduction to Secure Software Development

Introduction to Secure Software Development
Key concepts, processes, and best practices for building resilient applications.

1. Why Security Matters in Software Development

Reason	Impact
Data breaches	Financial loss, legal penalties, brand damage
Regulatory compliance	GDPR, HIPAA, PCI-DSS require protection of personal data
Threat landscape	Malware, ransomware, supply-chain attacks are increasing
Customer trust	Secure products retain users and attract new business

2. Core Principles of Secure Development (The “Secure Development Lifecycle” – SDL)

Principle	Description	Typical Activities
Shift-Left Security	Integrate security early, not as an after-thought	Threat modeling, security requirements, code reviews
Least Privilege	Give components only the permissions they need	Role-based access control, sandboxing
Defense in Depth	Multiple layers of protection	Input validation, encryption, network segmentation
Fail Securely	System should remain safe when errors occur	Graceful degradation, safe defaults
Secure by Design	Architecture built with security constraints	Secure design patterns, threat modeling
Continuous Monitoring	Ongoing detection of anomalies	Logging, intrusion detection, runtime scanning

3. Secure Development Lifecycle (SDL) Stages

Introduction to Secure Software

1. Planning & Requirements

Development

- Identify security & compliance requirements.
- Define security objectives (confidentiality, integrity, availability).
- Create a **risk register**.

2. Threat Modeling

- Use STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) or PASTA.
- Produce data flow diagrams (DFDs) and identify attack surfaces.

3. Secure Architecture & Design

- Apply security patterns (e.g., input validation gateway, reference monitor).
- Document security controls and trust boundaries.

4. Secure Coding

- Follow language-specific secure coding standards (e.g., OWASP Secure Coding Practices, CERT).
- Use static analysis tools (SAST) and linters.
- Enforce code reviews with a security checklist.

5. Testing & Verification

- **Static testing:** SAST, linting, dependency checking (SBOM, SCA).
- **Dynamic testing:** DAST, interactive application security testing (IAST), fuzzing.
- **Penetration testing** and **red team exercises** for high-risk components.
- Verify cryptographic implementations (e.g., correct key management).

6. Deployment & Release

- Harden environments (OS, containers, CI/CD pipelines).
- Use infrastructure-as-code with security policies (e.g., Terraform Sentinel, OPA).
- Sign artifacts, enable secure supply chain verification (SBOM, provenance).

7. Operations & Maintenance

- Continuous monitoring (SIEM, EDR).

Introduction to Secure Software

- Patch management & vulnerability remediation.
- Incident response plan & post-mortem analysis

Development

8. Retirement

- Secure data deletion, key revocation, and decommissioning procedures.

4. Common Vulnerabilities (OWASP Top10 – 2023 Snapshot)

Category	Example	Mitigation
A01: Broken Access Control	Unauthorized API calls	Enforce server-side authorization, use RBAC
A02: Cryptographic Failures	Weak TLS, hard-coded keys	Use modern algorithms, rotate keys, store secrets in vaults

**A03