**DevSecOps Essentials**

**1. What is DevSecOps and Why It Matters**

**Definition**

DevSecOps = **Development + Security + Operations**.  
It is the practice of integrating **security at every stage of the software development lifecycle (SDLC)**, rather than treating it as an afterthought.

**Why It Matters**

* Traditional DevOps prioritizes **speed and automation**. Without security, it may introduce vulnerabilities.
* Cyberattacks are growing in frequency and complexity (e.g., ransomware, supply chain attacks, API exploits).
* Regulatory requirements (GDPR, HIPAA, PCI-DSS) demand **security compliance**.
* DevSecOps ensures applications are **secure by design**, reducing costly fixes in production.

**Key Principle:** *“Security is everyone’s responsibility — developers, testers, and ops teams alike.”*

**2. Thinking Like a Hacker: Threat Modeling Basics**

Threat modeling helps identify **potential attack vectors** early in design and development.

**Steps in Threat Modeling**

1. **Identify Assets** → What are you protecting? (e.g., customer data, payment info).
2. **Identify Threats** → Who could attack? (e.g., hackers, insiders, competitors).
3. **Identify Vulnerabilities** → Weaknesses (e.g., weak authentication, unpatched software).
4. **Define Attack Vectors** → How could attackers exploit them? (e.g., SQL injection, DDoS).
5. **Mitigation Strategies** → Controls (e.g., input validation, rate limiting, WAFs).

**Common Threat Categories (STRIDE Model)**

* **S**poofing (fake identities)
* **T**ampering (data modification)
* **R**epudiation (denying actions taken)
* **I**nformation disclosure (data leaks)
* **D**enial of Service (system overloads)
* **E**levation of privilege (unauthorized admin access)

**Example Use Case: Online Banking App**

* Assets: User accounts, transactions.
* Threat: SQL injection attack on login page.
* Mitigation: Parameterized queries, input validation.

**3. Prevention Strategies and “Shift-Left” Security**

**Shift-Left Security**

* Traditional approach → security tested at the end (too late).
* Shift-left → move security testing **earlier** in the development lifecycle.
* Developers detect vulnerabilities during coding, not after deployment.

**Key Prevention Strategies**

1. **Secure Coding Practices**
   * Input validation, least privilege, avoid hardcoded secrets.
2. **Static Application Security Testing (SAST)**
   * Analyzes source code for vulnerabilities. (Tools: SonarQube, Checkmarx).
3. **Dynamic Application Security Testing (DAST)**
   * Tests running applications for runtime vulnerabilities. (Tools: OWASP ZAP, Burp Suite).
4. **Software Composition Analysis (SCA)**
   * Identifies vulnerabilities in open-source dependencies. (Tools: Snyk, OWASP Dependency-Check).
5. **Secrets Management**
   * Store credentials in secure vaults (e.g., HashiCorp Vault, AWS Secrets Manager).
6. **Container & Infrastructure Security**
   * Scan Docker images (Trivy, Anchore).
   * Enforce Infrastructure as Code (IaC) scanning (Terraform Checkov, KICS).

**4. Security in the Pipeline**

Integrating security into CI/CD ensures **continuous protection**.

**Typical Secure Pipeline Stages**

1. **Code Commit**
   * Run SAST & secret scanning tools.
   * Enforce secure code review policies.
2. **Build Stage**
   * Scan dependencies (SCA).
   * Run unit tests with security checks.
3. **Test Stage**
   * Run DAST tools on staging environment.
   * Penetration testing with automated scripts.
4. **Deploy Stage**
   * Verify container images are signed & scanned.
   * Apply Kubernetes Pod Security Policies.
5. **Monitor Stage**
   * Runtime monitoring with IDS/IPS (e.g., Falco, Wazuh).
   * Log analysis for suspicious activity.

**Table: Security Tools in CI/CD**

| **Stage** | **Tools/Practices** |
| --- | --- |
| Code Commit | SAST (SonarQube), Secret Scanning (GitLeaks) |
| Build | SCA (Snyk, OWASP Dependency-Check) |
| Test | DAST (OWASP ZAP, Burp Suite) |
| Deploy | Image Scanning (Trivy), IaC Scanning (Checkov) |
| Monitor | Runtime Security (Falco, Aqua, Sysdig) |

**5. Example Use Case: E-Commerce Platform**

* **Scenario:** A company is deploying a microservices-based e-commerce app.
* **DevSecOps Practices:**
  + Developers use SAST tools to catch vulnerabilities in checkout service code.
  + Jenkins pipeline integrates Snyk to scan dependencies before build.
  + Docker images are scanned with Trivy before pushing to AWS ECR.
  + DAST runs against staging environment before deployment.
  + Runtime monitoring with Falco detects suspicious container behavior.
* **Outcome:** Reduced risk of supply chain attacks, secure payments, and customer trust.

**6. Summary**

* **DevSecOps integrates security** into DevOps workflows.
* **Threat modeling** helps think like a hacker and identify vulnerabilities.
* **Shift-left security** ensures vulnerabilities are caught early.
* A secure pipeline includes **SAST, DAST, SCA, secrets management, and runtime monitoring**.
* DevSecOps fosters a culture of **shared security responsibility** across development, operations, and security teams.