Week 03: DevOps 2 Part 3 Notes

Key Concepts and Principles

What is DevSecOps?

- **Definition:** DevSecOps integrates security practices into every stage of the software development lifecycle (SDLC) to create secure, efficient applications without sacrificing speed.
- **Core Goals:**

Make security a **shared responsibility**
Automate security testing
Continuously monitor for vulnerabilities

DevSecOps Principles:

1. **Everyone is responsible for security**
2. **Start security at the beginning** of the SDLC
3. **Integrate security testing into every stage**
4. **Automate security testing** to reduce human errors
5. **Measure and monitor security continuously**
6. **Deliver secure software faster while ensuring compliance**

DevSecOps Best Practices:

1. **Shift Left:** Detect vulnerabilities early in development
2. **Use Automated Security Tools:** Integrate security scanning tools into CI/CD pipelines
3. **Promote Security Awareness:** Train team members in secure coding and infrastructure practices
4. **Shift Right:** Continuously monitor deployed applications for vulnerabilities

Security Testing in CI/CD Pipelines

Common Types of Security Testing:

- **Static Application Security Testing (SAST):**

Analyzes source code for vulnerabilities before runtime
Tools: Amazon CodeGuru Security
Pipeline Stage: Post-commit or build

- **Software Composition Analysis (SCA):**

Identifies vulnerabilities in open-source software components
Tools: OWASP Dependency-Check
Pipeline Stage: Build

- **Dynamic Application Security Testing (DAST):**

Simulates real-world attacks on a running application.
Tools: OWASP ZAP
Pipeline Stage: Test

- **Interactive Application Security Testing (IAST):**

Monitors applications from within to detect vulnerabilities during runtime.
Pipeline Stage: Test

- **Secrets Detection:**

Finds hardcoded credentials, API keys, and secrets in code.
Tools: Integrated into CodeGuru Security

Security Testing Workflow in CI/CD:

1. **Commit Code:** Code is committed to the repository (e.g., AWS CodeCommit)
2. **Run SAST and SCA:** Code is scanned for vulnerabilities (e.g., using CodeGuru Security)
3. **Deploy to Test Environment:** CodeDeploy deploys the application for DAST and IAST
4. **Initiate Security Scans:** Tools like ZAP simulate attacks
5. **Review Results:** Teams review findings and remediate issues
6. **Deploy Secure Application:** Deploy to production only after all checks pass

Tools Overview

Amazon CodeGuru Security:

- **Purpose:** Performs SAST, secrets detection, and code quality analysis
- **Features:**

Detects vulnerabilities (e.g., SQL injection, resource leaks)
Provides remediation suggestions
Integrates with CI/CD pipelines

- **Example Workflow:**

Commit code to AWS CodeCommit
CodeBuild triggers CodeGuru Security for analysis
Review findings in the CodeGuru console

Amazon Inspector:

- **Purpose:** Provides continual security monitoring in production environments
- **Scans:** EC2 instances, Amazon ECR repositories, Lambda functions
- **Features:**

Automated discovery and real-time scanning
Provides risk-based remediation prioritization
Integrates with AWS Security Hub and EventBridge

- **Example Workflow:**

Activate Amazon Inspector in the AWS Console
Automatically scan resources for vulnerabilities
Review findings in Security Hub

OWASP Tools:

1. **Dependency-Check (SCA):** Scans open-source libraries for vulnerabilities
2. **ZAP (DAST):** Conducts automated and manual penetration testing

Common Software Vulnerabilities

1. **Injection Flaws:** SQL injection, cross-site scripting (XSS)
2. **Broken Access Controls:** Weak authentication or authorization mechanisms
3. **Cryptographic Failures:** Improper encryption use
4. **Insecure Designs:** Flaws in architecture or design
5. **Security Misconfigurations:** Improperly configured software or hardware
6. **Outdated Components:** Using unpatched or vulnerable libraries
7. **Identification Failures:** Weak password policies or insecure login methods
8. **Data Integrity Failures:** Tampering or malicious modifications

Mitigation Strategies:

- Employ secure coding practices (e.g., input validation).
- Use threat modeling during design
- Regularly update software libraries
- Conduct routine security testing

DevSecOps Implementation Scenarios

Scenario: Automating Vulnerability Detection

- **Challenge:** Addressing vulnerabilities detected late in development.
- **Solution:**

Adopt **shift-left testing** to detect issues early
Use **automated security tools** to integrate scans into CI/CD pipelines
Promote **security awareness** across teams
Implement **shift-right practices** to monitor applications post-deployment

Summary

- DevSecOps enhances security without compromising speed
- Automated tools like Amazon CodeGuru Security and Amazon Inspector streamline vulnerability detection and remediation
- Best practices, including shift-left and shift-right, ensure robust security throughout the SDLC
- Continuous monitoring and cross-team collaboration are critical to successful DevSecOps implementation