FACULDADE ENGENHARIA

Departamento de Informática

Qualidade de Software

(14450)

DevSecOps

Today's Goals

- Cover the basics of DevSecOps
- Introduce the concept of software component analysis
- Introduce the idea of static and dynamic application security testing
- Compliance as code
- Discover DevSecOps tools
- Hands-on activity

What is DevSecOps?

- DevSecOps integrates security practices into DevOps.
- Shifts security 'left' in the software development lifecycle.
- Ensures security is a shared responsibility across teams.
- Automates security checks to keep pace with rapid development.

Why DevSecOps Matters

- Reduces vulnerabilities by addressing security issues early.
- Increases collaboration between development, operations, and security teams.
- Ensures continuous compliance with security standards.
- Supports faster and more secure software delivery.

Key Principles of DevSecOps

- **Shift Left**: Integrate security early in the development process.
- **Automation**: Automate security testing within CI/CD pipelines.
- **Continuous Monitoring**: Monitor for threats and vulnerabilities.
- **Culture of Shared Responsibility**: Encourage collaboration and security awareness.

What is Software Component Analysis (SCA)?

- ✧ SCA identifies known vulnerabilities in third-party and open-source components.
- ✧ Scans software dependencies for outdated or insecure versions.
- ✧ Uses vulnerability databases (e.g. the National Vulnerability Database, USA).
- ✧ Helps organizations manage risk in their software supply chain.

Why SCA is Important

- Modern software often relies on open-source components.
- Vulnerabilities in dependencies can expose the entire application.
- Reduces the risk of software supply chain attacks (e.g., Equifax breach).
- Ensures compliance with security standards and regulations.

How SCA Works

- **Step 1**: Identify all third-party components used in the application.
- **Step 2**: Match components against known vulnerability databases (e.g., NVD).
- **Step 3**: Provide alerts and remediation suggestions for vulnerable components.
- **Step 4**: Monitor for new vulnerabilities and update components as needed.

Real-World Examples of SCA Tools

- ✧ **Snyk**: Scans for vulnerabilities in open-source libraries and suggests fixes.
- ✧ **WhiteSource**: Monitors software components for license compliance and vulnerabilities.
- ✧ **Black Duck**: Provides detailed reports on open-source component risks.
- ✧ **Nexus Lifecycle**: Automates security checks for software dependencies.

What is Static Application Security Testing (SAST)?

- SAST analyzes source code or binaries for vulnerabilities without executing the code.
- Helps identify coding flaws, such as SQL injection or buffer overflows.
- Performed early in the software development lifecycle (SDLC).
- Provides developers with specific code locations for issues detected.

Why SAST is Important

- Detects vulnerabilities early in the development process, reducing remediation costs.
- Helps ensure secure coding practices are followed from the start.
- Reduces the likelihood of vulnerabilities being introduced into production.
- Enhances developer awareness of secure coding principles.

How SAST Works

- **Step 1**: Scans the source code, bytecode, or binaries for known patterns of vulnerabilities.
- **Step 2**: Identifies specific lines of code where vulnerabilities may exist.
- **Step 3**: Generates a report with details on the detected issues and recommendations for fixing them.
- **Step 4**: Integrates into CI/CD pipelines for continuous security analysis.

Real-World Examples of SAST Tools

- **SonarQube**: Identifies security vulnerabilities in various programming languages.
- **Checkmarx**: Offers detailed code analysis for a wide range of languages.
- **Fortify**: Provides static analysis for enterprise-level applications.
- **Veracode**: Integrates SAST with cloud-based scanning for continuous security monitoring.

What is Dynamic Application Security Testing (DAST)?

- ✧ DAST involves testing a running application to find vulnerabilities in real-time.
- ✧ It is a black-box testing method that simulates attacks to detect security weaknesses.
- ✧ Focuses on vulnerabilities such as SQL injection, cross-site scripting (XSS), and authentication flaws.
- ✧ Provides insight into the application's behavior and security from an attacker’s perspective.

Why DAST is Important

- Identifies security issues that occur during runtime, which static testing might miss.
- Helps detect configuration and deployment-related vulnerabilities.
- Provides a realistic assessment of the application's security posture.
- Suitable for testing web applications, APIs, and services in their deployed state.

How DAST Works

- **Step 1**: The application is deployed in a test or production environment.
- **Step 2**: The DAST tool performs automated scans to simulate attacks and detect vulnerabilities.
- **Step 3**: Reports provide information on detected vulnerabilities and potential impact.
- **Step 4**: Developers or security teams use the findings to patch or fix the issues.

Real-World Examples of DAST Tools

- **OWASP ZAP**: An open-source DAST tool that performs automated security scans for web applications.
- **Burp Suite**: Widely used for security testing of web applications, offering both manual and automated tools.
- **Acunetix**: Automated tool that scans web applications for various security vulnerabilities.
- **Netsparker**: Uses a unique scanning algorithm to accurately detect vulnerabilities.

What is Compliance as Code?

- Compliance as Code automates the enforcement of compliance policies.
- Uses code to define, manage, and apply compliance rules.
- Integrates compliance checks within the software development lifecycle.
- Enables continuous compliance monitoring and auditing.

Why Compliance as Code Matters

- Ensures adherence to regulatory requirements (e.g., GDPR, HIPAA, PCI DSS).
- Reduces manual compliance checks, saving time and resources.
- Automates auditing and reporting for compliance standards.
- Helps detect non-compliance issues early in the development process.

Key Principles of Compliance as Code

- ✧ **Define Compliance Rules as Code**: Policies are codified and version-controlled.
- ✧ **Automate Compliance Checks**: Integrate into CI/CD pipelines.
- ✧ **Continuous Monitoring**: Automatically detect non-compliance issues.
- ✧ **Remediation Automation**: Automatically fix compliance issues when possible.

Compliance as Code Workflow

- ✧ **Step 1**: Define compliance requirements as code.
- ✧ **Step 2**: Integrate compliance checks into CI/CD pipelines.
- ✧ **Step 3**: Continuously monitor for compliance issues in deployed environments.
- ✧ **Step 4**: Generate reports and take remediation actions as needed.

Tools for Implementing Compliance as Code

- **Open Policy Agent**: Policy engine for enforcing rules.
- **InSpec**: Framework for automated testing and auditing of compliance.
- **Chef Compliance**: Ensures that systems comply with desired state policies.
- **HashiCorp Sentinel**: Policy as code framework for Terraform and other tools.

Benefits of Compliance as Code

- Automates repetitive compliance tasks.
- Provides a consistent approach to managing compliance.
- Enables continuous compliance across all environments.
- Reduces risk by catching issues early in the development process.

Challenges in Adopting Compliance as Code

- Requires a shift in mindset towards treating compliance as part of the development process.
- May need integration with legacy systems and existing compliance frameworks.
- Policy definitions must be kept up to date with changing regulations.
- Complexity in automating certain compliance requirements.

Best Practices for Compliance as Code

- Start by codifying high-priority compliance requirements.
- Integrate compliance checks into every stage of the CI/CD pipeline.
- Regularly update policy definitions to reflect regulatory changes.
- Train teams on compliance requirements and the tools used.

Hands-on activity

- ✧ Warming up!

What are your favorites science fiction novels or films?

Analysis of a Science Fiction Film

- **Plot Summary**: Briefly describe the story, setting, and main characters.
- **Themes and Messages**: Identify core themes like technology, AI, dystopia, or human nature and the film’s commentary on these.
- **Technical Aspects**: Examine special effects, cinematography, and sound, highlighting how they support the sci-fi atmosphere.

Analysis of a Science Fiction Film

- **Characters and Development**: Analyze character roles, motivations, and their interactions with technology or society.
- **Real-World Relevance**: Discuss parallels to real-world issues (e.g., AI, cybersecurity, ethics) and how the film speculates on future impacts.
- **Impact and Legacy:** Reflect on the film's influence, cultural significance, and contributions to science fiction.

Analysis of a Science Fiction Film

Group Discussion

What went wrong in the scenario?

What security measures were missing?

How could DevSecOps practices have prevented this situation?

Designing a DevSecOps Solution

Identify a real-world system similar to the one presented in the film (e.g., military systems, IoT devices, AI-powered platforms).

Outline a basic CI/CD pipeline with security integrations to prevent the security flaw identified in the film.

Have them specify which DevSecOps tools they would use (e.g., Snyk, OWASP ZAP, Aqua Security).

Design a logical and/or physical architecture of the proposed solution.

Presentation Time!

3-minutes pitch

Think critically! (which tools and practices might work best in different contexts)?

Last Challenge :)

How security in fictional settings can parallel real-world challenges?

How DevSecOps may to improve cybersecurity in futuristic scenarios?

HOW
WHY
HOW
WHEN
WHERE
WHO
WHERE
WHERE
WHY
WHO
WHO
WHEN
WHAT
WHAT
WHY
WHAT
WHEN
HOW
WHAT
WHAT
WHEN
WHO
WHERE
WHY
HOW