**DevSecOps Framework Strategy**

| **Activity Plan** |  |
| --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
| | **Date** | **Activity** | | --- | --- | | | | |
| |  |  | | --- | --- | | **Mar 4, 2025** | Define scope, security objectives, and tool requirements | | | | |
| |  |  | | --- | --- | | **Mar 5, 2025** | Establish security policies, guidelines, and goals | | | | |
| |  |  | | --- | --- | | **Mar 6, 2025** | Choose and plan integration of security tools | | | | |
| |  |  | | --- | --- | | **Mar 9, 2025** | Set up GitHub Actions workflow and pipeline triggers | | | | |
| |  |  | | --- | --- | | **Mar 10-11, 2025** | Configure and integrate SonarQube for static analysis | | | | |
| |  |  | | --- | --- | | **Mar 12-13, 2025** | Set up Trivy and OWASP Dependency-Check for vulnerability scanning | | | | |
| |  |  | | --- | --- | | **Mar 16-17, 2025** | Integrate OWASP ZAP and Snyk for dynamic security testing | | | | |
| |  |  | | --- | --- | | **Mar 18-19, 2025** | Install and configure Prometheus and Grafana for monitoring | | | | |
| |  |  | | --- | --- | | **Mar 20-23, 2025** | Test the entire CI/CD pipeline and review security scans | | | | |
| |  |  | | --- | --- | | **Mar 24, 2025** | Review, refine configurations, and address any issues | | | | |
| |  |  | | --- | --- | | **Mar 25-26, 2025** | Finalize documentation and provide internal training on the tool | | | | |
| **Mar 27, 2025** Presentation | | | |
|  |  |
|  |  |

**1. Introduction to DevSecOps**

* **Definition**:
  + DevSecOps (Development, Security, and Operations) is a set of practices that integrates security into the DevOps process. It aims to build, test, and release secure applications by incorporating security at every phase of the software development lifecycle (SDLC).
* **Goal**:
  + The primary goal is to **shift security left**, meaning security is integrated early in the development process rather than being an afterthought or handled separately. It ensures vulnerabilities are caught earlier, leading to faster releases with fewer risks.

**2. DevSecOps Lifecycle Stages**

**Stage 1: Planning**

* **Objective**: Establish security goals and define standards for the entire SDLC.
* **Activities**:
  + Define **security policies** and **standards** for coding, testing, and deployment.
  + Conduct **threat modeling** to identify potential risks.
  + Plan the **integration of security tools** in the pipeline.
  + **Set security KPIs** for measuring risk and success.
* **Tools**:
  + **OWASP Threat Dragon** – For creating threat models.
  + **JIRA** – To track security-related issues and tasks.
  + **Security documentation** and internal communication tools (e.g., Slack, Confluence).

**Stage 2: Development & Code**

* **Objective**: Ensure secure coding practices are followed from the start and code is analyzed for vulnerabilities.
* **Activities**:
  + Implement **secure coding standards** (e.g., input validation, error handling).
  + Run **static analysis** tools to identify security flaws during coding.
* **Tools**:
  + Code grepper – Performs static code analysis to catch security issues early.
* **Automation**:
  + Set up automatic triggers to run code grepper on every commit to GitHub to check for vulnerabilities in real-time.

**Stage 3: Build**

* **Objective**: Secure the build process and ensure there are no vulnerabilities in dependencies or the build environment.
* **Activities**:
  + Scan dependencies for known vulnerabilities.
  + Ensure the build environment is secure.
* **Tools**:
  + **Trivy** – Container security scanning for vulnerabilities.
  + **SonarQube** – Performs static code analysis to catch security issues early.
  + **OWASP Dependency-Check** – Identifies vulnerabilities in third-party libraries and dependencies.
* **Automation**:
  + Integrate these tools into the CI/CD pipeline (e.g., GitHub Actions) to scan for vulnerabilities every time the application is built.

**Stage 4: Security Testing**

* **Objective**: Perform dynamic application security testing (DAST) to discover runtime vulnerabilities.
* **Activities**:
  + Perform penetration testing and identify security holes.
  + Automate dynamic testing as part of the CI pipeline.
* **Tools**:
  + **OWASP ZAP (Zed Attack Proxy)** – For dynamic security testing of running applications.
  + **Snyk** – Identifies security issues in dependencies and container images.
* **Automation**:
  + Automate DAST testing in the CI/CD pipeline, ensuring that every deployment undergoes dynamic testing before release.

**Stage 5: Release & Deploy**

* **Objective**: Ensure security measures are in place before releasing to production.
* **Activities**:
  + Ensure secure configurations and environment variables.
  + Validate Docker images and deployment scripts for vulnerabilities.
* **Tools**:
  + **Anchore** – Docker image scanning for vulnerabilities.
  + **Automation**:Automate the scanning of images before deployment in CI/CD pipeline. Use GitHub Actions to ensure Docker containers are secure before deployment.

**Stage 6: Monitor**

* **Objective**: Continuously monitor the application in the live environment for vulnerabilities and security threats.
* **Activities**:
  + Implement continuous monitoring to detect anomalies and breaches.
  + Set up alerting for vulnerabilities or security incidents.
* **Tools**:
  + **Prometheus** + **Grafana** – For monitoring runtime performance and security metrics.
  + **Falco** – For container security monitoring.
* **Automation**:
  + Set up automated monitoring and alerting using Prometheus, Grafana, and Falco, ensuring the system is constantly evaluated for security risks.

**3. Automation Strategy Using GitHub Actions**

1. **GitHub Workflow Configuration**:
   * Create a .github/workflows/ci-cd.yml file to automate each stage of the DevSecOps pipeline.
   * The file will define steps for triggering:
     + **Static Code Analysis** (SonarQube)
     + **Dependency Scanning** (Trivy, OWASP Dependency-Check)
     + **Dynamic Testing** (OWASP ZAP, Snyk)
     + **Automated Alerts** (Prometheus + Grafana)
2. **Secrets Management**:
   * Store sensitive tokens (SonarQube, Snyk, etc.) securely using GitHub Secrets to authenticate the tools in the pipeline.
3. **Continuous Feedback Loop**:
   * Send automated alerts to Slack, email, or JIRA if any vulnerabilities are detected at any stage, ensuring quick remediation.

**4. How to Execute DevSecOps for an App in GitHub**

1. **Codebase Setup**:
   * Ensure your code is hosted in a GitHub repository.
   * Integrate security tools (SonarQube, Trivy, ZAP, Snyk) into the GitHub Actions pipeline for automated security scanning.
2. **Triggering the Pipeline**:
   * The CI/CD pipeline will trigger upon a **push** to the main branch or any pull request to main. It will run security checks for every commit and pull request automatically.
3. **Review and Monitor**:
   * After each deployment, monitor the application for vulnerabilities using monitoring tools like Prometheus and Falco.
   * Implement feedback loops to inform the development team of any vulnerabilities found.

**5. Conclusion**

* **DevSecOps** is about shifting security left and making it part of the culture rather than just an afterthought.
* **Automation** is key in speeding up the process while ensuring that security is continuously integrated and tested throughout the development lifecycle.
* By **integrating security tools** early and automating the entire lifecycle, organizations can ensure that they produce secure, high-quality software at speed.