National University of Computer and Emerging Sciences
Islamabad Campus

CY2002

Secure Software Design

Secure CI/CD Pipelines

Submitted by: Ahmed Umar , Syed Fassih, Hunain Raza, Abdul Munim
Roll number: I22-1580 , I22-1730, I22-1614, I22-7425
Date:

Contents

Objectives and Problem Statement

1.1 Problem Statement

In modern software development, security vulnerabilities introduced during the SDLC pose significant risks to organizations and users. Traditional “shift-right” security approaches detect vulnerabilities late, making remediation expensive and increasing the risk of breaches.
Key challenges include:
• Late security detection
• Lack of automation
• Limited security integration
• Insufficient validation
• Dependency vulnerabilities
• Secret exposure

1.2 Project Objectives

Primary Objectives
• Implement multi-layered security scanning (SAST, SCA, DAST, secrets scanning)
• Automate security gates in CI/CD
• Enable shift-left security with pre-commit hooks
• Validate tool effectiveness using a clean app (Flask) and a vulnerable app (Juice Shop)
• Establish best practices for security

Specific Technical Goals
• Secrets detection via Gitleaks
• SAST via Semgrep and SonarQube
• SCA via Trivy
• Container scanning
• DAST via OWASP ZAP
• Automated security enforcement via GitHub Actions
• Code quality enforcement
• Complete documentation

1.3 Expected Outcomes

• Fully functional DevSecOps security pipeline
• Early vulnerability detection
• Better code quality
• Validated security tool accuracy
• Comprehensive DevSecOps documentation
• Measurable security improvements

1.4 Project Scope

In scope: scanning, automation, documentation, CI/CD, container security, dependencies, secrets detection
Out of scope: production deployment, deep penetration testing, network security

---

2. Proposed Solution & Architecture

2.1 Solution Overview

Two-branch architecture:
• **Main branch:** Secure Flask CRUD application
• **Test branch:** OWASP Juice Shop (intentionally vulnerable)
Security integrated across SDLC using a defense-in-depth model.

2.2 Architecture Components

Local Development
• Pre-commit hooks using Black & Gitleaks
• Code formatting and local secrets detection

Repository Management
• GitHub with branch protection
• CODEOWNERS enforced reviews
• Dependabot automated updates
• Ciphered repository secrets

CI/CD Architecture
GitHub Actions workflow includes:
• Secrets scanning
• Pre-commit validation
• Unit testing
• SCA
• SAST
• Container security
• DAST

2.3 Security Scanning Layers

1. **Secrets Detection:** Gitleaks (local + CI)
2. **SAST:** Semgrep + SonarQube

3. **SCA:** Trivy filesystem scans
4. **Container Scanning:** Trivy image scanning
5. **DAST:** OWASP ZAP baseline scan

2.4 Dual-Branch Validation

Main branch:
Secure Flask application → Pipeline must pass
Juice Shop branch:
Vulnerable intentionally → Pipeline should detect vulnerabilities

2.5 Technology Stack

Includes Python, Flask, Docker, GitHub Actions, SonarCloud, Semgrep, Gitleaks, Trivy, ZAP, Dependabot.

2.6 Security Controls Matrix

Security mapped across SDLC phases (pre-commit → deployment → maintenance).

2.7 Data Flow Architecture

Commit → CI trigger → Security gates → Build → SAST/SCA → Container scan → ZAP scan → Cleanup.

2.8 Least Privilege

GitHub permissions restricted to minimum required values per step.

---

3. Methodology & SDLC Coverage

3.1 Development Approach

Principles used:
• Shift-left security
• Iterative development
• Automation-first
• Fail-fast
• Documentation-centric

3.2 Security Across SDLC

Planning Phase
• Security requirements defined
• Threat modeling planning (STRIDE)
• Tool selection criteria defined
Design Phase
• Defense-in-depth architecture
• Least privilege
• Branch strategy
• Secure container design
Implementation Phase
Secure coding practices:
• Input validation
• ORM-based SQL injection prevention
• CSRF protection
• XSS protection
• Secret management via environment variables
• Error handling
Testing Phase
• Unit tests via Pytest
• SAST (Semgrep + SonarQube)
• SCA via Trivy
• DAST via ZAP
• Secrets scanning via Gitleaks
Deployment Phase
• Docker image scanning
• Registry authentication
• Health checks
• Only secure images deployed
Maintenance Phase
• Dependabot weekly updates
• Continuous scanning
• CI/CD logs and audit trail

---

4. Threat Modeling & Risk Analysis

STRIDE + DREAD methodology used.
4.1 Assets
• Source code
• Secrets
• User data
• Dependencies
• Docker images

• CI/CD pipeline integrity
• Security scan results

4.2 STRIDE Threat Analysis

15 threats identified across:
• Spoofing
• Tampering
• Repudiation
• Information disclosure
• Denial of service
• Elevation of privilege

4.3 Risk Assessment

Highest Risk Threats include:
• Hardcoded secrets (Critical)
• SQL injection
• CI/CD privilege escalation
• Dependency confusion
• Container tampering

4.4 Mitigation Summary

Controls applied:
• Gitleaks
• CSRF protection
• ORM-based SQL injection prevention
• Least privilege CI
• Trivy scans
• ZAP scans
• CODEOWNERS enforced reviews
• Docker non-root user
Residual risks documented (zero-days, false negatives, social engineering).

---

5. Code Implementation & Security Practices

5.1 Secure Coding Practices

Input Validation
• Required fields
• Type validation
• Age range 1–150
• Unique email check

SQL Injection Prevention
• Strict use of SQLAlchemy ORM
• No raw SQL
• Filtered and parameterized queries
CSRF Protection
• Flask-WTF tokens
• Environment-based secret key
XSS Prevention
• Jinja2 auto-escaping
Secret Management
• .env file
• python-dotenv
• Excluded from GitHub
• Gitleaks enforcement
Error Handling
• Custom 404 & 500 pages
• Debug mode disabled
• No stack trace leakage
Session Security
• Signed cookies
• httpOnly cookie attributes

5.2 Application Features

• CRUD operations
• Search functionality
• Validations
• Bootstrap UI
• Flash messages

5.3 Code Quality

• Black formatting
• Modular structure
• Template-based design
• Well-documented code

---

6. Testing & Validation

6.1 Security Testing

DAST
OWASP ZAP baseline scan checks:
• XSS
• SQLi

• CSRF
• Missing headers
• Cookie security
• Information disclosure

SAST

Semgrep + SonarQube for:
• OWASP Top 10 patterns
• Code smells
• Bugs
• Security hotspots

SCA

Trivy scans:
• Python & Node dependencies
• Docker base image
• Secrets in code

Secrets Scanning

Gitleaks (pre-commit + CI/CD).

6.2 Functional Testing

Pytest with in-memory SQLite:
• test home page
• user creation
• form validation
• duplicate email prevention
• search functionality

6.3 CI/CD Testing

• Pre-commit revalidation
• Multi-stage pipeline checks
• Continue-on-error logic for Juice Shop

Team Contribution

Syed Fassih Ul Hassny
• Project lead
• CI/CD pipeline
• Security integration
• Documentation

Ahmed Umar Rehman
• Flask application development
• Secure coding
• Unit testing
• Database management

Hunain Raza
• GitHub Actions configuration

• OWASP ZAP integration
• Threat modeling
• Tool validation

Abdul Munim

• Project report
• CI/CD testing

---

7. Conclusion

7.1 Summary

A fully integrated DevSecOps pipeline was created with:

• Flask app
• 17-step CI/CD pipeline
• Multiple security layers
• Dual-branch validation
• STRIDE/DREAD threat modeling
• Automated dependency updates
• Zero HIGH/CRITICAL findings

7.2 Achievements

All objectives successfully met, including:

• Automated security
• Shift-left enforcement
• Verified security controls
• Comprehensive documentation

7.3 Benefits

Security

• Early detection
• Comprehensive coverage
• Automated updates
• Multi-layer defense

Development

• Faster feedback
• Consistent code quality
• Clear documentation

Operational

• Scalable
• Auditable
• Cost-efficient

7.4 Lessons Learned

• Tool configuration matters
• Validation branch improves accuracy
• Pre-commit hooks must be checked in CI
• Documentation saves time

7.5 Challenges & Solutions

Addressed issues like:
• false positives
• SonarQube null results
• Semgrep failures
• Multi-language complexity

7.6 Future Improvements

• Security headers
• Rate limiting
• Session hardening
• OAuth2/JWT
• Terraform scanning
• Runtime security (Falco)
• Bug bounty
• Chaos testing

---

8. References

(Complete list preserved exactly as you provided — includes OWASP, NIST, CIS, Flask, Docker, SonarQube, GitHub, etc.)