DevPipeline Security Scanning Guide

Document ID: PRD-DP-015 **Last Updated:** 2024-02-10 **Owner:** DevPipeline Security Team **Classification:** Public

---

Overview

DevPipeline includes integrated security scanning to help you identify vulnerabilities early in your development process. This guide covers available scanning types, configuration, and best practices.

Scanning Types

Static Application Security Testing (SAST)

Analyzes source code for security vulnerabilities.

Supported Languages:

Language	Scanner	Coverage
JavaScript/TypeScript	Semgrep, ESLint Security	High
Python	Bandit, Semgrep	High
Java	SpotBugs, Semgrep	High
Go	gosec, Semgrep	High
Ruby	Brakeman	High
C/C++	Flawfinder	Medium
PHP	PHPCS Security	Medium
C#	Security Code Scan	Medium

What it Detects: - SQL injection - Cross-site scripting (XSS) - Command injection - Path traversal - Hardcoded secrets - Insecure cryptography

Software Composition Analysis (SCA)

Scans dependencies for known vulnerabilities.

Package Managers: - npm / yarn / pnpm - pip / poetry / pipenv - Maven / Gradle - Go modules - RubyGems - NuGet - Cargo

Vulnerability Databases: - National Vulnerability Database (NVD) - GitHub Advisory Database - OSV (Open Source Vulnerabilities) - Snyk Vulnerability DB

Container Scanning

Analyzes container images for vulnerabilities.

Capabilities: - OS package vulnerabilities - Application dependencies in image - Misconfigurations - Malware detection - Secret detection in layers

Supported Registries: - Docker Hub - Amazon ECR - Google Container Registry - Azure Container Registry - Private registries

Secret Detection

Identifies hardcoded secrets in code and configuration.

Detects: - API keys - Access tokens - Private keys - Passwords - Connection strings - AWS credentials - Database credentials

---

Configuration

Enable Security Scanning

```yaml
# .devpipeline.yml
security:
  enabled: true

  sast:
    enabled: true
    languages:
      - javascript
      - python

  sca:
    enabled: true
    fail_on: critical  # critical, high, medium, low

  container:
    enabled: true
    images:
      - $IMAGE_NAME:$IMAGE_TAG

  secrets:
    enabled: true
    block_push: true
```

Scan on Every Commit

```yaml
jobs:
  security-scan:
    stage: test
    script:
      - devpipeline scan --all
    artifacts:
      reports:
        sast: sast-report.json
        dependency_scanning: dependency-report.json
        container_scanning: container-report.json
        secret_detection: secrets-report.json
```

Configure Severity Thresholds

```yaml
security:
  thresholds:
    sast:
      critical: 0      # Fail if any critical
      high: 5          # Fail if more than 5 high
      medium: 20       # Fail if more than 20 medium
    sca:
      critical: 0
      high: 10
    container:
      critical: 0
      high: 5
```

Scan Results

Viewing Results

In Pipeline: - Results appear in pipeline UI - Summary in job logs - Detailed report in artifacts

In Dashboard: - Security tab shows all findings - Filter by severity, type, project - Track findings over time

Result Format

```json
{
  "vulnerabilities": [
    {
      "id": "CVE-2024-1234",
      "severity": "high",
      "title": "SQL Injection in query builder",
      "description": "...",
      "location": {
        "file": "src/db/query.js",
        "line": 42
      },
      "remediation": "Update to version 2.1.0 or later",
      "identifiers": [
        {"type": "cve", "value": "CVE-2024-1234"}
      ]
    }
  ],
  "scan_info": {
    "scanner": "semgrep",
    "version": "1.50.0",
    "duration": "45s"
  }
}
```

Severity Levels

Level	Description	Action
Critical	Actively exploited, easy to exploit	Fix immediately
High	Significant risk, likely exploitable	Fix within 24 hours
Medium	Moderate risk, harder to exploit	Fix within 1 week
Low	Minor risk, limited impact	Fix within 1 month
Info	Best practice recommendations	Consider addressing

Managing Findings

Triaging Vulnerabilities

```
# .devpipeline-security.yml
ignore:
  # Ignore specific CVE (with justification)
  - id: CVE-2023-5678
    reason: "False positive - we don't use affected function"
```

```yaml
    expires: 2024-06-01

  # Ignore path
  - path: "tests/**"
    reason: "Test files only"

  # Ignore by rule
  - rule: generic-api-key
    path: "docs/examples/**"
    reason: "Example API keys in documentation"
```

False Positive Management

1. **Review finding** - Is it actually a vulnerability?
2. **Document decision** - Add to ignore list with reason
3. **Set expiration** - Re-evaluate periodically
4. **Track metrics** - Monitor false positive rate

Auto-Fix Suggestions

For some vulnerabilities, DevPipeline suggests fixes:

```bash
# View suggested fixes
devpipeline scan --suggest-fixes

# Apply automatic fixes (where safe)
devpipeline scan --auto-fix

# Create PR with fixes
devpipeline scan --create-fix-pr
```

Integration

Pull Request Comments

Security findings appear as PR comments: - Summary of new vulnerabilities - Inline comments on affected lines - Suggested fixes where available

IDE Integration

VS Code extension shows security issues: - Real-time scanning as you type - Inline vulnerability warnings - Quick-fix suggestions

Notifications

```yaml
security:
  notifications:
    slack:
      channel: "#security-alerts"
      on: [critical, high]

    email:
      recipients: ["security@company.com"]
      on: [critical]

    webhook:
      url: "https://..."
      on: [critical, high, medium]
```

Compliance

Generating Reports

```bash
# Generate compliance report
devpipeline security report \
  --format pdf \
  --standard soc2 \
  --output security-report.pdf
```

Supported Standards

Standard	Report Type
SOC 2	Security controls evidence
PCI DSS	Vulnerability scan report
HIPAA	Security assessment
ISO 27001	Risk assessment

Audit Trail

All security scans are logged: - Scan timestamp - Scanner versions - Findings at time of scan - Triage decisions - Who approved/ignored findings

---

Best Practices

Shift Left

1. **Pre-commit hooks** - Catch issues before commit
2. **PR scanning** - Block merges with critical issues
3. **Scheduled scans** - Catch new CVEs in existing code

Prioritize Effectively

1. **Critical/High first** - Focus on real risks
2. **Reachable code** - Prioritize code paths in use
3. **Public-facing** - APIs and user inputs first

Reduce Noise

1. **Tune rules** - Disable noisy rules
2. **Baseline** - Ignore existing issues temporarily
3. **Context** - Use path-specific ignores

Stay Current

1. **Update scanners** - New rules catch new issues
2. **Monitor CVEs** - New vulnerabilities daily
3. **Review ignores** - Expired ignores may be exploitable

Troubleshooting

Scan Takes Too Long

```yaml
security:
  sast:
    timeout: 30m
    exclude:
      - "vendor/**"
      - "node_modules/**"
      - "**/*.min.js"
```

Too Many False Positives

1. Update scanner to latest version
2. Review and tune rules
3. Add appropriate ignores with justification

Scan Fails

```
# Debug mode
devpipeline scan --debug

# Check scanner health
devpipeline scan --health-check
```

Resources

- Security Rule Reference
- CVE Database
- Best Practices
- Support

Related Documents: CI/CD Best Practices (PRD-DP-005), Pipeline YAML Reference (PRD-DP-010), Compliance Guide (PRD-DP-025)