Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core components and implementation details for **DevSecOpsLab - Secure CI/CD Pipeline Simulator**. The simulator provides a hands-on environment for secure DevOps practices, integrating static code analysis, secrets management, container security, and CI/CD security best practices using Jenkins, Docker, and GitHub Actions.

---

1. System Components

Component	Description	Key Responsibilities
Web UI	User interface for simulation setup and results	User input, display results, trigger pipeline
Pipeline Orchestrator	Manages CI/CD pipeline execution	Trigger stages, manage flow
Static Code Analyzer	Scans code for vulnerabilities	Run SAST tools, report findings
Secrets Manager	Detects and manages secrets in code	Scan for secrets, mask/remove, alert
Container Security Module	Scans Docker images for vulnerabilities	Run image scans, report issues
Security Test Runner	Executes automated security tests	Run DAST/SAST, collect results
Integration Layer	Interfaces with Jenkins, Docker, GitHub Actions	API calls, job management
Results Aggregator	Collects and summarizes security findings	Aggregate, format, and store results

---

2. Class/Interface Overview

Class/Interface	Description	Key Methods/Attributes
`PipelineManager`	Orchestrates pipeline stages	`run_pipeline()`, `add_stage()`
`StaticCodeAnalyzer`	Handles static code analysis	`analyze_code(repo_url)`
`SecretsScanner`	Scans for secrets in code	`scan(repo_path)`, `mask_secrets()`
`ContainerScanner`	Scans Docker images	`scan_image(image_id)`
`SecurityTestRunner`	Runs security test suites	`run_tests(target_url)`
`IntegrationAdapter`	Abstracts CI/CD tool integration	`trigger_job(tool, config)`
`ResultsAggregator`	Aggregates and formats results	`collect(results)`, `summarize()`

Relationships:

- `PipelineManager` composes all other modules.

- `IntegrationAdapter` interfaces with Jenkins, Docker, GitHub Actions.

---

3. Data Structure Overview

Data Model	Fields / Schema Example	Purpose
`PipelineConfig`	`id`, `stages`, `repo_url`, `image_id`, `test_targets`	Defines pipeline parameters
`ScanResult`	`type`, `issues[]`, `severity`, `timestamp`	Stores scan/test findings
`SecretFinding`	`file`, `line`, `secret_type`, `masked_value`	Details of secrets found
`AggregatedReport`	`pipeline_id`, `summary`, `detailed_results[]`	Final user-facing report

---

4. Algorithms / Logic

Pipeline Execution Flow (Pseudocode):

```
def run_pipeline(config):
    for stage in config.stages:
        if stage == "static_analysis":
            results = StaticCodeAnalyzer.analyze_code(config.repo_url)
        elif stage == "secrets_scan":
            results = SecretsScanner.scan(config.repo_url)
        elif stage == "container_scan":
            results = ContainerScanner.scan_image(config.image_id)
        elif stage == "security_tests":
            results = SecurityTestRunner.run_tests(config.test_targets)
        ResultsAggregator.collect(results)
    return ResultsAggregator.summarize()
```

---

5. Error Handling

Scenario	Handling Approach
Tool/API Unavailable	Retry, log error, notify user
Invalid Pipeline Configuration	Validate input, return error to UI
Scan/Test Failure	Log details, continue pipeline, mark as failed
Secrets Exposure Detected	Mask secret, alert user, halt if critical
Integration Failure (Jenkins/etc)	Fallback to alternate tool, log, notify user

---

End of Document