AI - Powered Risk Analysis Platform

- BY DEEPTHINK

FINAL REPORT

ASSIGNMENT COVER SHEET

UNIVERSITY OF WOLLONGONG IN DUBAI

Assignment Cover Sheet

Subject Code:	CSIT321
Subject Name:	Project
Submission Type:	Final Report
Assignment Title:	Final Report
Student Name:	Abbas Bhaiji, Ansaf Sabu, Derick Reni, Mohamed Rizvan, Mohammed Uddin
Student Number:	6345438, 7390440, 6728492, 7797965, 7842430
Student Email:	ab836@uowmail.edu.au, as3623@uowmail.edu.au, djr980@uowmail.edu.au, memrr999@uowmail.edu.au, msumk596@uowmail.edu.au
Lecturer Name:	Dr. Farhad Oroumchian
Due Date:	30/12/2025
Date Submitted:	30/12/2025

PLAGIARISM:
The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism.
The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation.

PLEASE NOTE: STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED

DECLARATION:
I/We certify that this is entirely my/our own work, except where I/we have given fully documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism.

Signature of Student:

Optional Marks:

Comments:

Lecturer Assignment Receipt (To be filled in by student and retained by Lecturer upon return of assignment)

Subject:	**Assignment Title:**
Student Name:	**Student Number:**
Due Date:	**Date Submitted:**
Signature of Student:	

Student Assignment Receipt (To be filled in and retained by Student upon submission of assignment)

Subject:	**Assignment Title:**
Student Name:	**Student Number:**
Due Date:	**Date Submitted:**
Signature of Lecturer	

Tables of Content

Executive Summary

ArmourEye is an AI-powered container security analysis platform designed to automate vulnerability detection and provide intelligent insights for Docker-based applications. This project represents the culmination of a two-semester development effort by Team DeepThink.

Original Vision

The initial proposal envisioned a fully autonomous penetration testing platform that would simulate a human red team's workflow, including reconnaissance, exploit identification, automated exploitation, post-exploitation analysis, and automated patching all driven by Reinforcement Learning (RL) and Large Language Models (LLMs).

Delivered Solution

Through iterative development and feasibility analysis, the project evolved into a practical, production-ready security scanning platform that combines traditional vulnerability scanning tools (Trivy, Nmap) with AI-powered analysis using Retrieval-Augmented Generation (RAG) and Large Language Models (Mistral 7B v3 Instruct). The platform successfully delivers:

- Automated vulnerability scanning for Docker container images
- AI-powered vulnerability analysis with human-readable summaries
- Network exposure assessment using Nmap integration
- Intelligent remediation guidance based on a curated vulnerability database
- Modern, responsive web interface for security analysts

Key Achievements

- Functional end-to-end scanning and analysis pipeline
- Integration of 3,251+ vulnerability records in a vector database
- Zero-budget AI deployment using Google Colab and Cloudflare Tunnel
- Real-time progress tracking and comprehensive reporting
- Session-based result caching for improved user experience

Scope Adjustments

The project scope was refined from a fully autonomous exploitation platform to an intelligent vulnerability analysis system. This pivot was driven by:

- Resource constraints (zero budget, undergraduate-level expertise)
- Ethical and legal considerations around automated exploitation
- Time limitations
- Practical value (analysis and reporting provide immediate business value)

The delivered system provides a solid foundation for future enhancements while delivering immediate practical value to organizations seeking automated container security analysis.

First & Second Semester Report Summary

This section summarizes the foundational reports developed during the first semester, which established the project vision, scope, and technical approach.

Proposal Report Summary

Document Purpose: The Proposal Report outlined the vision and objectives for ArmourEye, positioning it as an AI-driven platform that automates the penetration testing lifecycle.

Project Description

ArmourEye was envisioned to simulate a human red team's workflow, covering:

- Reconnaissance and target identification
- Exploit identification and selection
- Post-exploitation analysis
- Automated patching recommendations
- Business-focused risk reporting

Main Problems Addressed

- Manual, Time-Consuming Process: Traditional penetration testing requires significant human effort and expertise
- Siloed Tools: Security tools operate independently without intelligent orchestration
- Lack of End-to-End Automation: No single platform covers the complete pentesting lifecycle
- Technical-Business Gap: Difficulty translating technical vulnerabilities into business risk

Key Solutions (AI Integration)

- Machine Learning & Reinforcement Learning: Dynamic target prioritization and optimal exploit strategy selection
- Large Language Models: Customized attack script generation and technical-to-business translation
- Automated Patching System: Real-time vulnerability data integration for fix recommendations
- Interactive Dashboard: Business stakeholder-friendly risk visualization

Target Scope

Fully automated pentesting system for Linux-based systems, covering all core stages autonomously using AI and ML.

Target Audience

Startups, small companies, and educational institutions with limited cybersecurity resources or expertise.

Planning & Feasibility and Requirement Analysis Report Summary

Document Purpose: This report detailed the initial concept, objectives, scope, and implementation roadmap for the AI-driven pentesting platform.

Project Goal

To fully automate the penetration testing lifecycle on Linux-based internal systems by simulating a red team's workflow using AI agents.

Key Objectives

- Build an end-to-end autonomous pentesting agent
- Minimize the need for human oversight
- Generate automated payload delivery and dynamic exploits using AI
- Provide remediation suggestions and business-focused dashboards

Core Technology Stack

- Reinforcement Learning (RL): Adapt, prioritize, and select optimal exploit paths based on service risk and historical outcomes
- Large Language Models (LLMs): Generate tailored payloads, summaries, and remediation instructions
- Traditional Tools: Orchestration layer over Nmap, OWASP ZAP, and Metasploit

High Level Workflow

Reconnaissance → Vulnerability Identification → Exploit Selection (RL) → Payload Generation (LLM) → Execution → Patching Guidance → Business Reporting (LLM)

Target Niche

Startups, small companies, educational institutions, and security analysts seeking automated red-team simulations.

Hardware Requirements

NVIDIA GPU (RTX 4080 8GB/12GB or better) for efficient ML inference and RL training.

Project Timeline

Duration: May 2025 - December 2025

Integration Phase: November 21, 2025

Testing Phase: December 11-19, 2025

High Level Design Report Summary

Document Purpose: This report detailed the system architecture, component structure, user flows, and security considerations for implementation.

System Architecture

A modular monolith built with Flask (Backend) and Bolt/React (Frontend), designed for future migration to microservices.

Data Flow

User uploads Docker Image → Docker Lab Manager runs container → Scan Module → AI Orchestrator (LLM) selects exploit → Exploitation Module executes → Data Store logs actions → Reporting Module generates final PDF/JSON report

Key Components

Component	Description
Flask Application Server	Orchestrates core logic and manages Docker Lab Manager, Scan Module, AI Orchestrator, and Exploit Dispatcher
Data Store	SQLite (development) with migration path to PostgreSQL (production)
AI Decision Engine	Communicates with OpenAI API or local LLMs
Docker Lab Manager	Manages isolated container environments for testing
Scan Module	Integrates Nmap, OWASP ZAP, and other scanners
Exploitation Module	Executes selected exploits in controlled environment
Reporting Module	Generates PDF/JSON reports with business insights

Deployment Strategy

Entirely containerized using Docker Compose for easy setup and portability. Production target: Ubuntu Server / Cloud VM.

Security & Isolation

- Testing runs in isolated Docker networks with restricted permissions
- Containers run with `--cap-drop=ALL` to prevent escape vulnerabilities
- API Key-based authentication
- Network segmentation between testing and production environments

Test Plan Report Summary

Document Purpose: This report focused on the methodology, scope, and specific scenarios for testing the platform's core workflow.

Primary Objective

To verify and validate the platform's ability to reliably automate the penetration testing lifecycle against containerized web applications.

In Scope for Testing (Core MVP)

- GUI & Core Workflow (Docker image upload via GUI)
- Integration with free, third-party static analysis tools
- Vulnerability Analysis (CVE extraction, exploit identification, fix recommendations)
- Reporting and Data Persistence to PostgreSQL

Out of Scope for Testing

- Live exploitation using Metasploitable
- Multi-user collaboration features
- Advanced security bypassing techniques

Test Targets

Vulnerable Docker images including:

- OWASP DVWA (Damn Vulnerable Web Application)
- Metasploitable
- OWASP Juice Shop

High Level Test Scenarios

Scenario ID	Priority	Description
TS-01	Critical	Successful End-to-End Scan (Vulnerable Image) → LLM identifies correct CVEs and fixes → Report generated and saved
TS-02	High	Report and Fix Accuracy Validation (cross-reference LLM output with NVD database)
TS-NEG-01	Medium	Invalid file upload handling
TS-NEG-02	Medium	Scanner integration failure recovery

Exit Criteria

- All critical and high-priority scenarios pass
- Final report generation feature is fully functional and accurate
- System handles negative test cases gracefully

Change Report

Overview of Changes

During the implementation phase, significant architectural and scope changes were made to ensure project feasibility, deliverability, and practical value. This section documents these changes and provides technical and business justifications.

Summary of Major Changes

Aspect	Original Plan	Final Implementation	Change Type
Backend Framework	Flask (Python)	Node.js + Express	Technology
AI Approach	Reinforcement Learning	RAG + LLM	Architecture
Exploitation	Automated exploit execution	Removed from scope	Scope Reduction
Database	PostgreSQL	No persistent database	Architecture
Patching	Automated patch deployment	Manual guidance only	cope Reduction
AI Hosting	Local GPU (RTX 4080+)	Google Colab (free tier)/Local GPU	Infrastructure

Changes from Proposal Report

Reinforcement Learning → RAG-Based Analysis

The original plan was to use Reinforcement Learning to dynamically prioritize targets and select optimal exploit strategies based on historical outcomes.

Final Implementation

Implemented Retrieval-Augmented Generation (RAG) using Chroma vector database with 3,251 vulnerability records and Mistral 7B v3 for intelligent analysis.

Justification

- Complexity: RL requires extensive training data (1000s of successful/failed exploit attempts) which was not feasible to collect ethically or legally
- Time Constraints: Training an RL agent requires months of experimentation and tuning
- Practical Value: RAG provides immediate value by leveraging existing vulnerability knowledge
- Zero Budget: RAG can run on free-tier Colab, while RL training requires expensive GPU resources

RAG vs RL Architecture Diagram

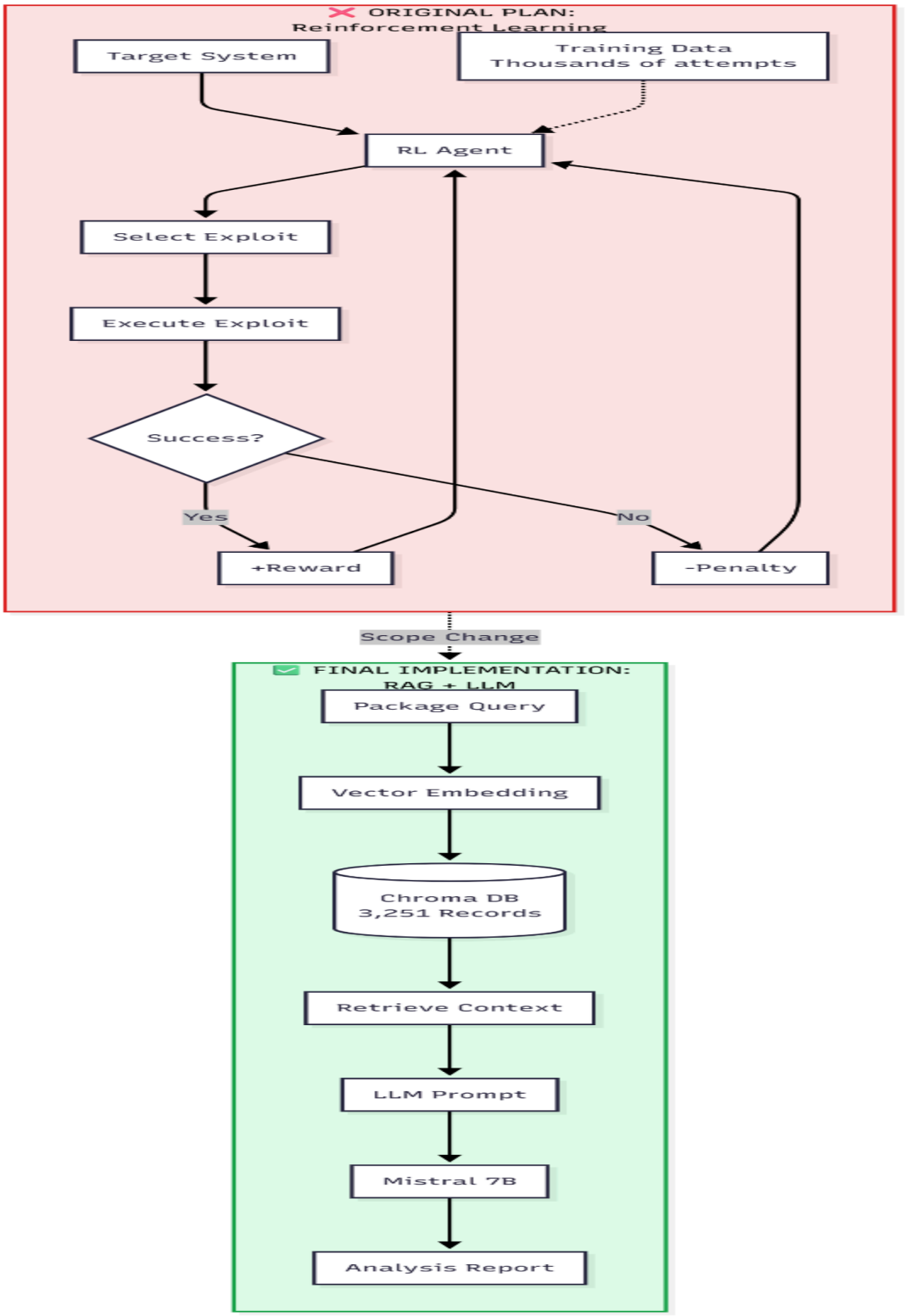


Figure 3.1. RAG VS RL Architecture Diagram

Automated Exploitation → Vulnerability Analysis

The original plan was to automatically execute exploits against target systems using AI-generated payloads.

Final Implementation

Focus on vulnerability identification, analysis, and remediation guidance without active exploitation.

Justification

- Legal & Ethical: Automated exploitation raises significant legal liability concerns
- Undergraduate Scope: Exploitation requires advanced offensive security expertise beyond undergraduate level
- Business Value: Most organizations need vulnerability identification and prioritization more than proof-of-concept exploits
- Resource Constraints: Building a safe, isolated exploitation environment requires significant infrastructure

Impact on Deliverables

- Retained: Vulnerability scanning, CVE identification, severity scoring
- Enhanced: AI-powered analysis and human-readable summaries
- Removed: Payload generation, exploit execution, post-exploitation analysis

Automated Patching → Remediation Guidance

The original plan was to automatically apply patches and fixes to vulnerable systems.

Final Implementation

Provide detailed remediation guidance with links to official patches and NVD references.

Justification

- Risk Management: Automated patching can break production systems if not thoroughly tested
- Complexity: Requires dependency resolution, compatibility testing, and rollback mechanisms
- Scope: This feature alone could be a separate project
- Practical Approach: Security teams prefer manual review before applying patches

Changes from Planning & Feasibility Report

Backend Technology Stack

The original plan was to implement Flask (Python) backend with SQLAlchemy ORM and PostgreSQL database.

Final Implementation

Node.js with Express.js, no persistent database (file-based storage for scan results).

Justification

- Ecosystem Alignment: Docker SDK for Node.js provides better integration for container management
- Asynchronous Operations: Node.js excels at handling concurrent scanning operations
- Rapid Development: Express.js middleware ecosystem accelerated development

Technical Implications

- Scan results stored as JSON files in filesystem
- Session-based caching in browser localStorage
- Simplified deployment (no database setup required)

Technology Stack Comparison Diagram

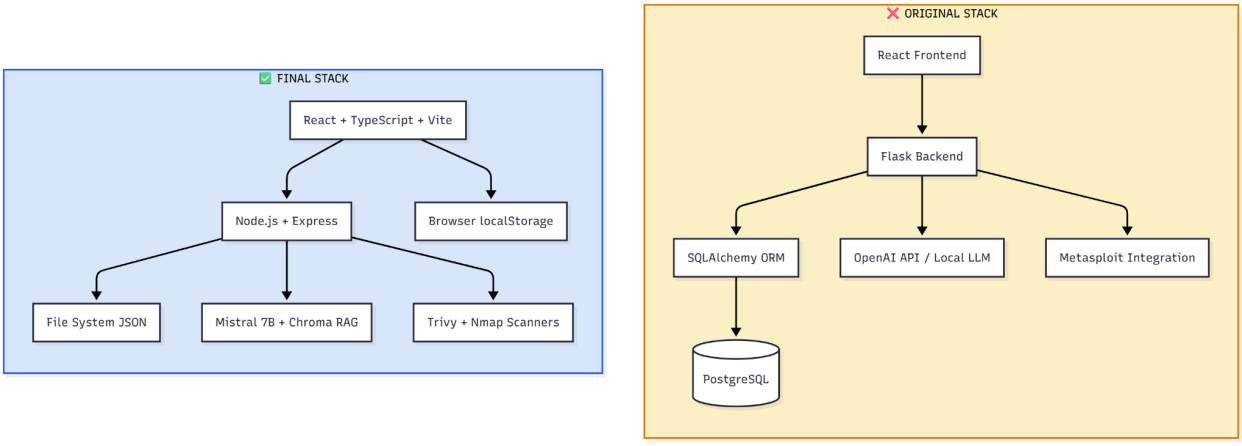


Figure 3.2. Technology Stack Comparison Diagram

Hardware Requirements

The original plan was to use NVIDIA RTX 4080 or better for local ML inference and RL training.

Final Implementations

Google Colab (free tier) with Cloudflare Tunnel for remote AI inference.

Justification

- Zero Budget: Team had no budget for GPU hardware ($1,200+ for RTX 4080)
- Accessibility: Colab provides free T4 GPU access (15GB VRAM)
- Flexibility: Remote deployment allows team members to access AI service from any location
- Proof of Concept: Demonstrates feasibility without capital investment

Trade Offs

- Session Limits: Colab free tier has 12-hour runtime limits
- URL Changes: Cloudflare Tunnel URL changes with each session
- Cost Savings: $0 vs. $1,200+ hardware cost
- Scalability: Can migrate to paid Colab or dedicated GPU later

Project Timeline Adjustments

Original Timeline:

- Integration: November, 2025
- Testing: December, 2025

Actual Timeline:

- Development: September 2025 - November 2025
- Integration & Testing: November 2025

Reasons for Adjustments

- Earlier start date allowed for iterative development even though the implementation started of later than expected due to restart of project due to scope renewal
- Scope reduction enabled faster delivery
- Continuous testing throughout development (not just final phase)

Changes from High Level Design Report

Component Architecture

Original Design

Component	Original Technology
Application Server	Flask
Data Store	PostgreSQL
AI Engine	OpenAI API / Local LLM
Exploitation Module	Metasploit Integration/AI Agents
Reporting	PDF + JSON

Final Implementations

Component	Implemented Technology
Application Server	Node.js + Express
Data Store	File System (JSON)
AI Engine	Mistral 7B v3 (Colab) + RAG (Chroma)
Exploitation Module	Trivy, Docker Scout + Nmap, GoBuster
Reporting	JSON Download

Component Architecture Evolution Diagram

Figure 3.3 Component Architecture Evolution Diagram

Data Flow Changes

Original Data Flow:

Upload → Scan → AI Selects Exploit → Execute Exploit → Log Results → Generate Report

Final Data Flow:

Upload → Scan (Trivy + Nmap) → AI Analysis (RAG + LLM) → Generate Report

Key Differences

- Removed exploit selection and execution stages
- Added RAG-based context retrieval
- Simplified to focus on analysis rather than action

Data Flow Sequence Diagram

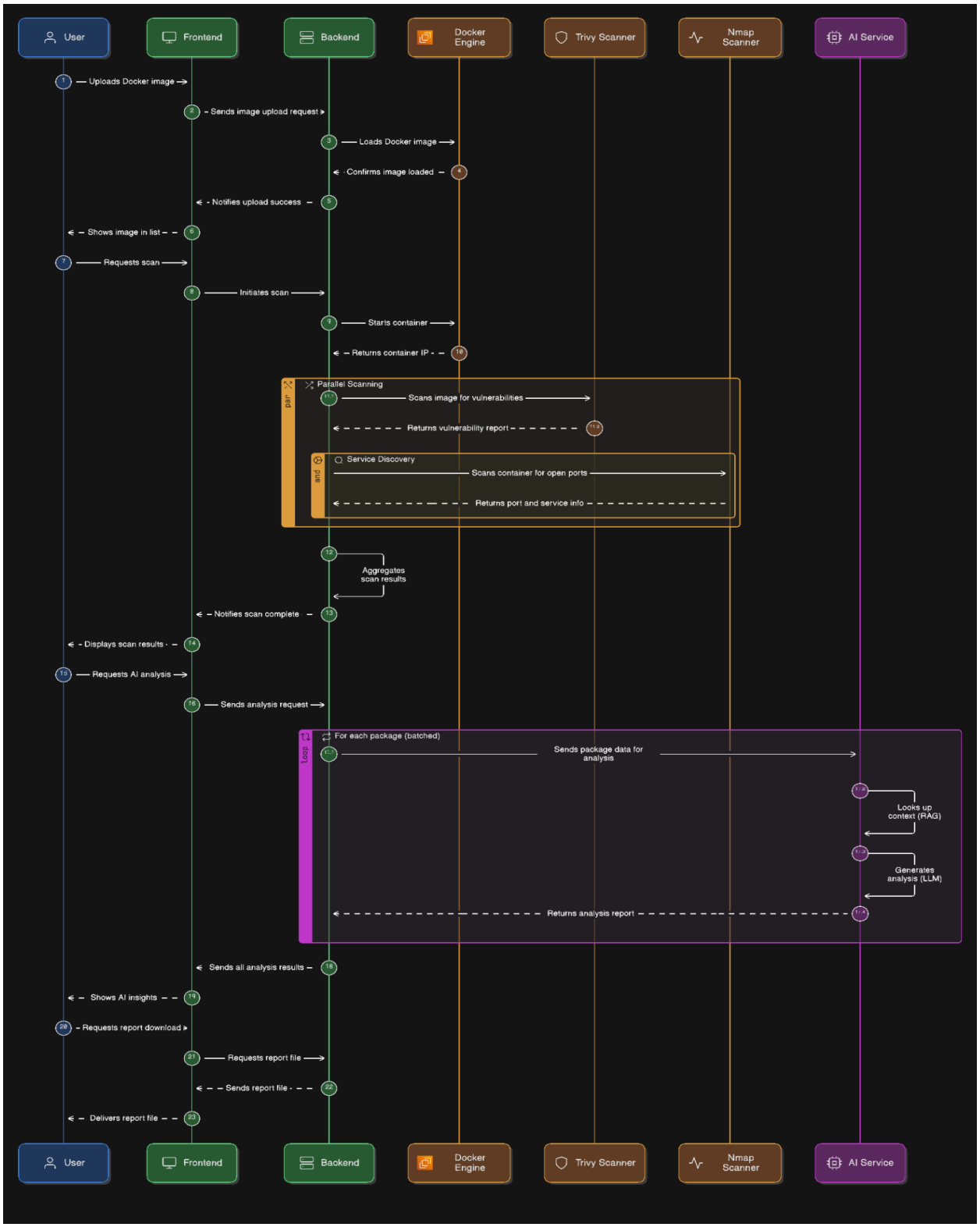


Figure 3.4. Data Flow Sequence Diagram

Security Model Simplification

Original Security Model:

- `--cap-drop=ALL` for container hardening
- API key authentication
- Exploit sandbox with network restrictions

Final Security Model:

- Docker container management for scan targets
- API endpoints without authentication (local deployment)
- No exploit execution (no sandbox needed)

Justification

- Reduced attack surface by removing exploitation capabilities
- Simplified deployment for educational/small business use
- Authentication can be added in future production releases

Justification for Changes

Technical Feasibility

The challenge is that the original scope required expertise in multiple advanced domains:

- Offensive security and exploit development
- Reinforcement Learning model training
- Production-grade database management
- Secure sandbox environment design

The solution is to focus on achievable goals with undergraduate-level knowledge:

- Integration of existing security tools (Trivy, Nmap)
- RAG implementation using established frameworks (LangChain, Chroma)
- File-based storage for simplicity
- Modern web development (React, Node.js)

Resource Constraints

Budget: $0 (zero budget constraint)

Impact on Original Plan

- Cannot purchase GPU hardware ($1,200+)
- Cannot use paid AI APIs and were that the use AI APIs was not allowed (OpenAI: $0.03/1K tokens)
- Cannot deploy to cloud infrastructure ($50-200/month)

Mitigation Strategies

- Google Colab free tier for AI inference
- Cloudflare Tunnel for free remote access
- Open-source tools (Trivy, Nmap, Mistral 7B)
- Local deployment (no hosting costs)

Time Constraints

Available Time: 2 months of active development (Due to scope changes)

Original Scope Estimate: 3 months for full implementation

Prioritization

- Must Have: Scanning, vulnerability analysis, reporting (Delivered)
- Should Have: AI-powered insights, remediation guidance (Delivered)
- Could Have: Exploitation, automated patching (Descoped)
- Won't Have: Multi-user collaboration, RL training (Descoped)

Ethical and Legal Consideration

Concern: Automated exploitation tools can be misused for malicious purposes.

Risk Mitigation

- Removed exploitation capabilities entirely
- Focus on defensive security (vulnerability identification)
- Educational value without legal liability
- Suitable for responsible disclosure workflows

Alignment with Industry Best Practice

- Similar to commercial tools: Qualys, Tenable.io (scan + analyze, no exploit)
- Follows responsible disclosure principles
- Suitable for compliance and audit use cases

Business Value Proposition

Original Value: "Fully automated pentesting replacing human red teams"

Revised Value: "AI-powered vulnerability analysis for rapid security assessment"

Why This Matters

- Immediate practical value for small businesses
- Faster than manual CVE research
- Human-readable summaries for non-technical stakeholders
- Foundation for future enhancements
- Demonstrates AI integration in cybersecurity

Implementation Report

System Architecture

ArmourEye is implemented as a modern web application with a clear separation between frontend, backend, and AI services. The architecture prioritizes modularity, maintainability, and ease of deployment.

High-Level Architecture

4-Tier Architecture Diagram

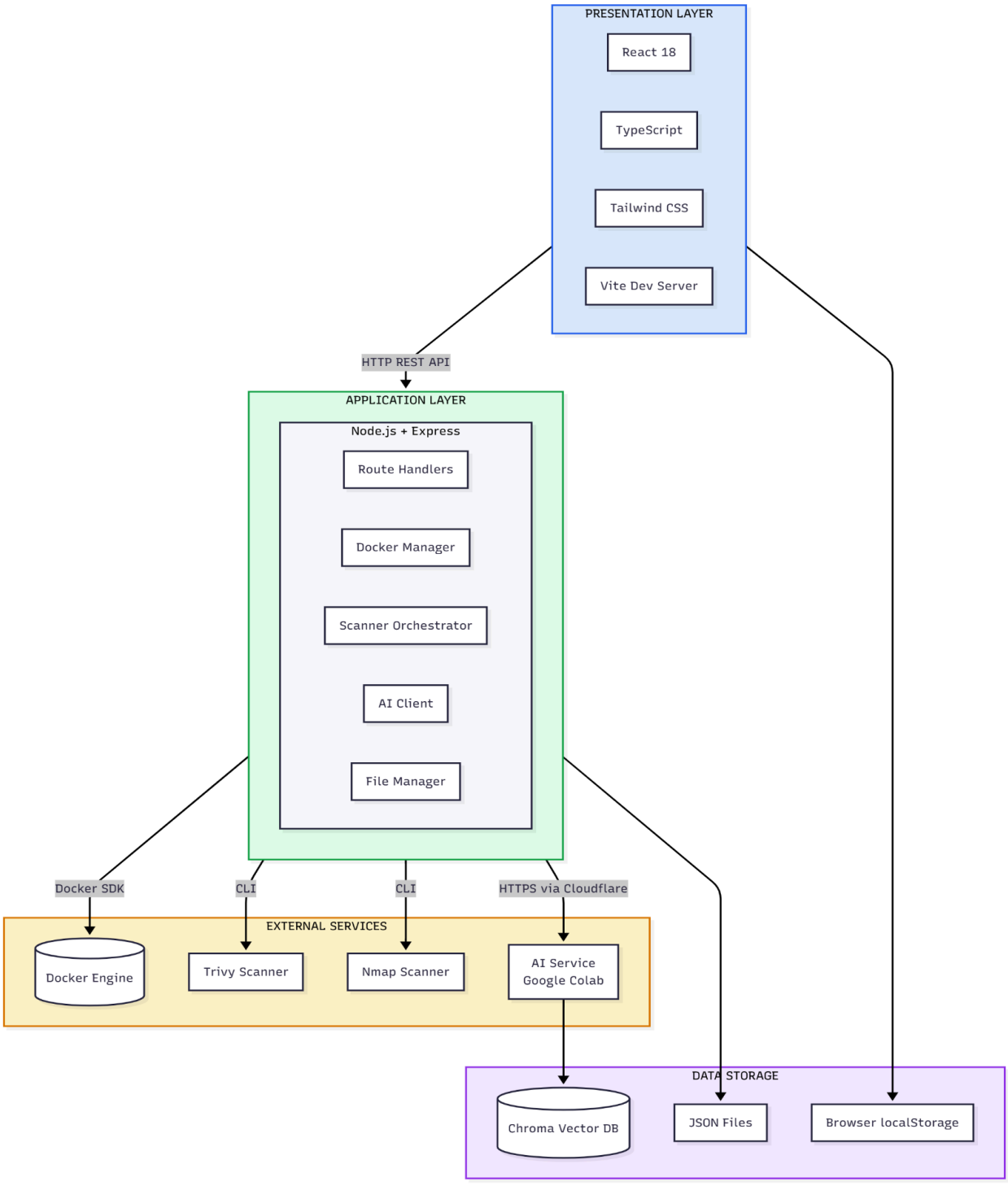


Figure 4.1. 4-Tier System Architecture Diagram

Component Interaction

Sequence of Operations:

- User uploads Docker image via web interface
- Frontend sends image to backend API
- Backend saves image and initiates scan workflow
- Docker SDK loads and runs container
- Trivy Scanner analyzes container for vulnerabilities
- Nmap Scanner performs network reconnaissance
- Backend aggregates scan results
- AI Service (Colab/Local) performs RAG-based analysis
- Backend combines all results
- Frontend displays results and generates downloadable report

Component Interaction Sequence Diagram

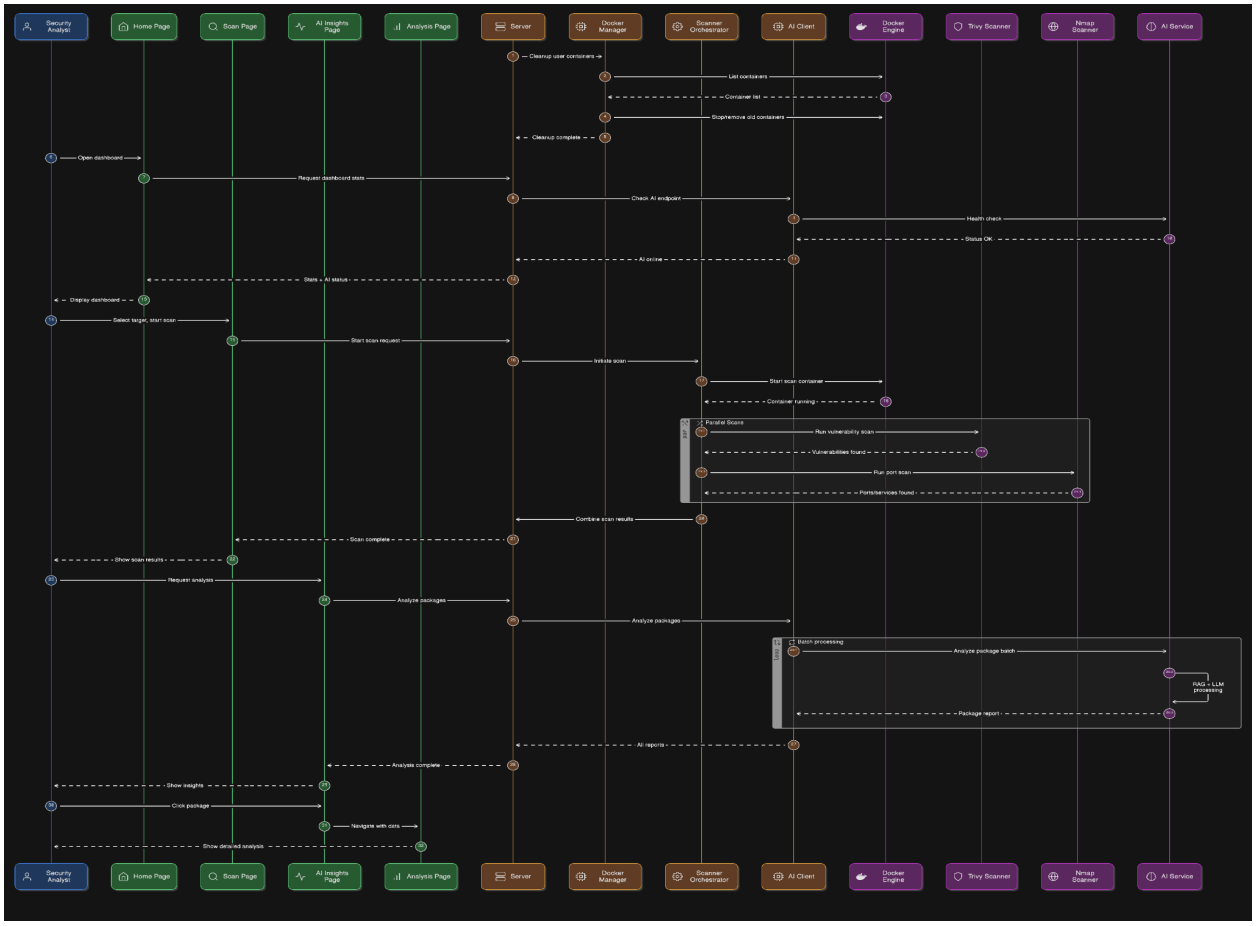


Figure 4.2. Component Interaction Sequence Diagram

Technology Stack

Frontend Technologies

Technology	Version	Purpose
React	19.x	UI framework for component-based development
TypeScript	5.x	Type-safe JavaScript for reduced runtime errors
Vite	7.x	Fast build tool and development server
Tailwind CSS	4.x	Utility-first CSS framework for rapid styling
Lucide React	Latest	Icon library for consistent UI elements

Key Features

- Real-time progress tracking with live updates
- Responsive design for desktop and tablet devices
- Session-based caching for improved UX
- Modern, accessible UI components

Backend Technologies

Technology	Version	Purpose
Node.js	20.x LTS	JavaScript runtime for server-side logic
Express.js	5.x	Web application framework
Docker SDK	Latest	Programmatic Docker container management
Trivy	Latest	Container vulnerability scanner
Nmap	7.x	Network scanning and service detection

Key Features

- RESTful API design
- Asynchronous scan orchestration
- File-based result storage
- Real-time log streaming

AI/ML Technologies

Technology	Version	Purpose
Mistral 7B v3 Instruct	Latest	Large Language Model for text generation
LangChain	Latest	LLM application framework
Chroma DB	Latest	Vector database for RAG
FastAPI	Latest	High-performance API framework (AI service)
Uvicorn	Latest	ASGI server for FastAPI

Key Features

- 3,251 vulnerability records in vector database
- Semantic search for relevant vulnerability context
- 4-bit quantization for efficient inference on free-tier Colab
- Cloudflare Tunnel for secure remote access

Development and Deployment Tools

Tool	Purpose
Git	Version control
npm	Package management
Docker Compose	Multi-container orchestration
Google Colab	Free GPU hosting for AI services
Cloudflare Tunnel	Secure tunnel for remote AI access

Core Components

Frontend Components

Home Page (Dashboard)

- Displays system statistics (scans performed, vulnerabilities found, AI scans)
- Shows system status (Docker, Trivy, AI service availability)
- Provides quick navigation to main features

Setup Page (Image Upload)

- Docker image upload interface
- Lists uploaded images with metadata
- Displays running containers
- Image management (view, select for scanning)

Scan Page

- - Target selection from uploaded images
- - Real-time scan progress tracking
- - Live log display
- - Scan result visualization
- - Report download functionality

AI Insights Page

- Target selection for AI analysis
- Progress tracking for AI processing
- Package-level vulnerability analysis
- LLM-generated summaries
- Detailed AI analysis logs
- Report download with target selection

Analysis Page (Package Details)

- Detailed package information
- Vulnerability details with CVE IDs
- Severity scoring and exploit availability
- Remediation guidance with NVD links
- Network exposure information
- AI assessment summary

Frontend Component Hierarchy Diagram

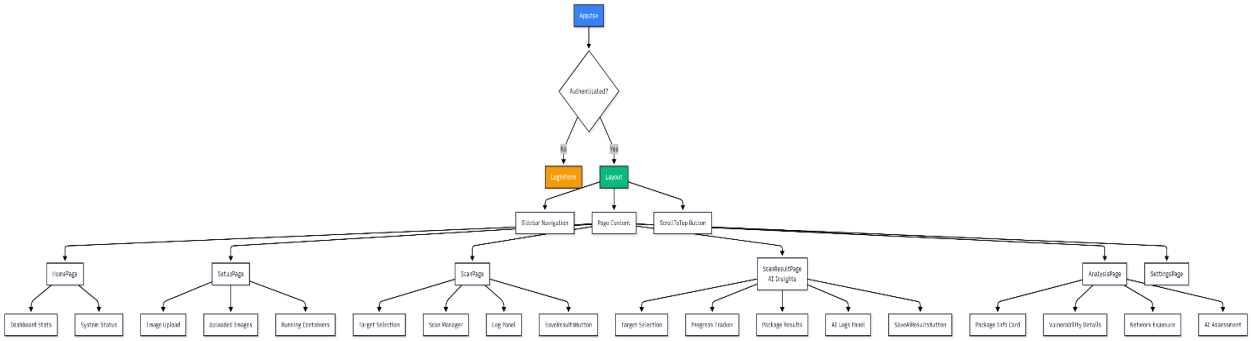


Figure 4.3. Frontend Component Hierarchy Diagram

Backend Components

Server Core (`server.js`)

- Express application setup
- Route definitions
- Middleware configuration
- Error handling

Docker Manager

- Container lifecycle management
- Image loading and validation
- Container cleanup on startup
- Network configuration

Scanner Orchestrator

- Trivy integration for vulnerability scanning
- Nmap integration for network scanning
- Result aggregation

AI Client

- FastAPI endpoint communication
- RAG request batching
- Error handling and retries
- Timeout management

File Manager

- Scan result persistence
- Image upload handling
- Report generation
- Log file management

Backend Component Architecture Diagram

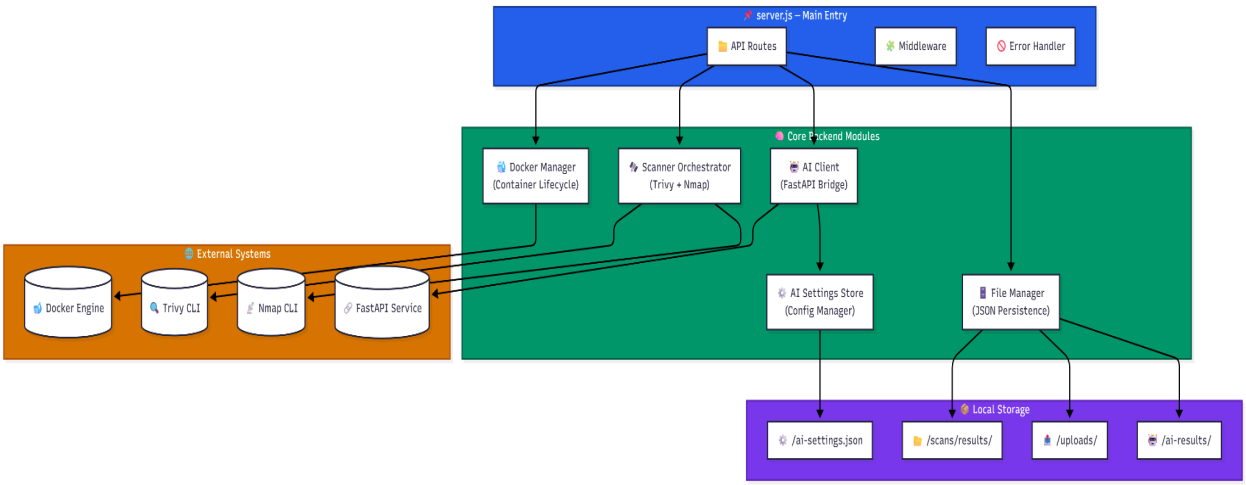


Figure 4.4. Backend Component Architecture Diagram

AI Service Components (Colab)

FastAPI Application (`server.py`)

- REST API endpoints for AI analysis
- Request validation using Pydantic
- Backwards compatibility for legacy requests

RAG Pipeline

- Chroma DB initialization and loading
- Vector similarity search
- Exact package name matching
- Context retrieval and verification

LLM Integration

- Mistral 7B model loading (4-bit quantization)
- Prompt engineering for vulnerability analysis
- Response parsing and structuring

- Token limit management

Report Builder

- Structured vulnerability reports
- Severity classification
- Exploit detection
- Remediation guidance generation

AI Service Architecture Diagram

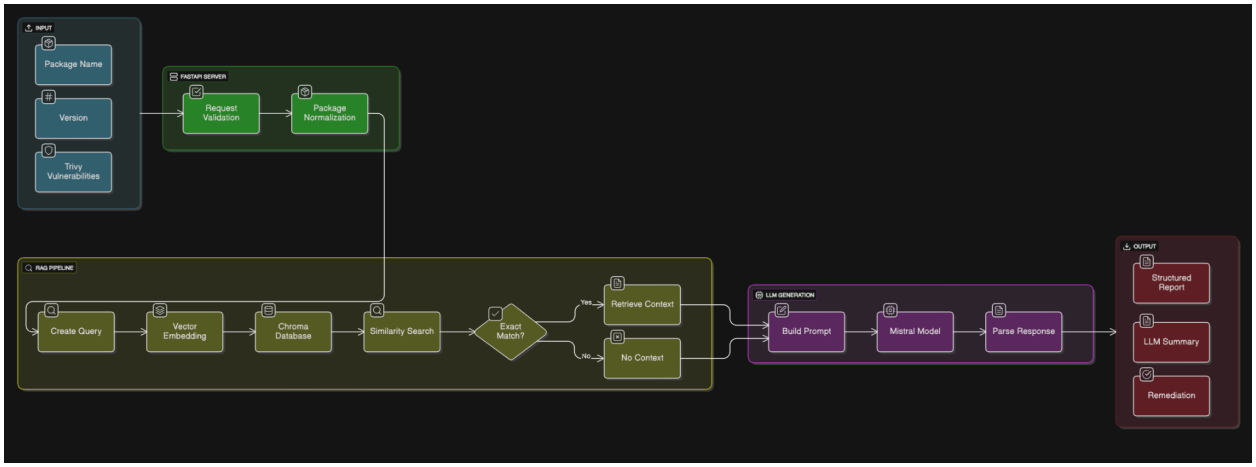


Figure 4.5. AI Service Architecture Diagram

Features Implemented

Core Features

Feature 1: Docker Image Management

- Upload Docker images via web interface
- Automatic image validation
- List uploaded images with metadata (size, creation date)
- Filter system images from user images
- Container lifecycle management

Feature 2: Vulnerability Scanning

- Trivy integration for CVE detection
- Comprehensive package analysis
- Severity scoring (CRITICAL, HIGH, MEDIUM, LOW)
- Vulnerability deduplication
- Scan result caching

Feature 3: Network Scanning

- Nmap integration for port scanning
- Service detection and version identification
- Top 1000 ports scanning
- Network exposure assessment
- Integration with vulnerability data

Feature 4: AI-Powered Analysis

- RAG-based vulnerability context retrieval
- LLM-generated human-readable summaries
- Package prioritization (Chroma DB packages first)
- Intelligent remediation guidance
- Runtime exposure analysis

Feature 5: Reporting

- JSON report generation
- Per-target report download
- Comprehensive vulnerability details
- AI analysis results
- Network exposure information

Feature 6: Real-Time Progress Tracking

- Live scan progress updates
- Detailed operation logs
- Error reporting
- Scan cancellation support (not fully due to corruption and breakage problems)

Feature 7: Session Management

- Per-target result caching (up to 10 targets)
- Browser localStorage integration
- Backend restart detection
- Automatic cache invalidation

User Interface Features

Modern, Responsive Design

- Tailwind CSS for consistent styling
- Mobile-friendly layouts
- Dark mode support
- Accessible UI components

Dashboard Statistics

- Total scans performed
- Vulnerabilities discovered
- AI scans completed
- Reports generated
- System health indicators

Interactive Elements

- Scroll-to-top button on long pages
- Collapsible log panels
- Sortable data tables
- Filterable image lists

System Features

Automatic Container Cleanup

- Removes stale containers on startup
- Prevents resource exhaustion
- Maintains clean Docker environment

Dynamic Settings Reload

- AI service URL updates without restart
- Mode switching (local/remote)
- Configuration persistence

Error Handling

- Graceful degradation on scanner failures
- User-friendly error messages
- Detailed error logging
- Retry mechanisms for transient failures

AI Integration

RAG Architecture

Retrieval-Augmented Generation (RAG) combines the strengths of information retrieval and language model generation to provide accurate, context-aware vulnerability analysis.

Components

Vector Database (Chroma DB)

- 3,251 vulnerability records
- Package metadata (name, version, CVE IDs)
- Vulnerability descriptions
- Exploit information
- Fix information

Embedding Model

- Converts text to vector representations
- Enables semantic similarity search
- Pre-computed embeddings for fast retrieval

Retrieval Process

- Query: Package name + version
- Semantic search in vector space
- Exact package name verification
- Context ranking and selection

Generation Process

- Retrieved context + user query → LLM prompt
- Mistral 7B generates analysis
- Structured output parsing
- Human-readable summary generation

RAG Pipeline Flowchart Diagram

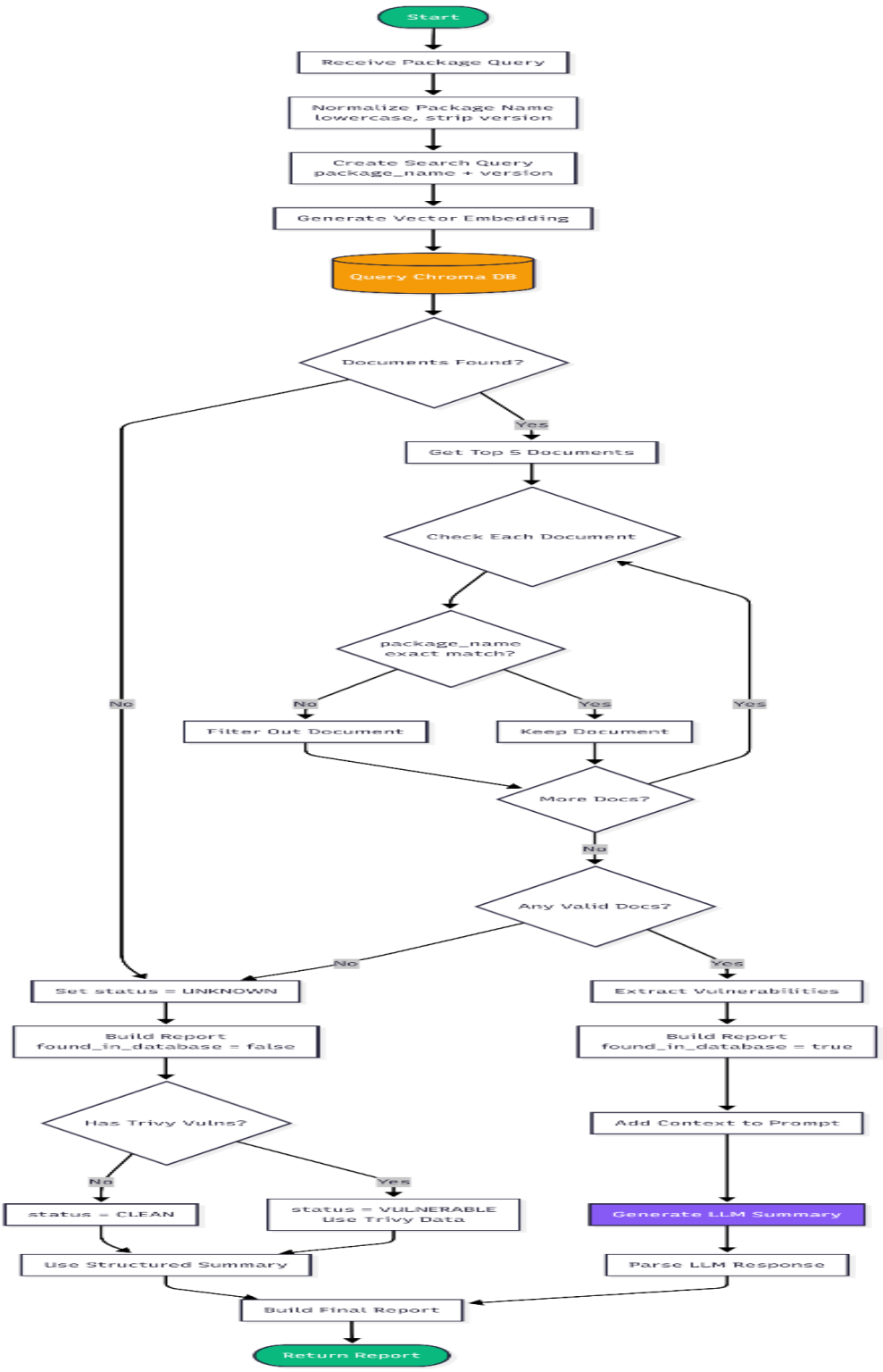


Figure 4.6. RAG Pipeline Flowchart Diagram

LLM Prompt Engineering

Prompt Structure:

```
You are a cybersecurity expert analyzing vulnerability data.

Package: {package_name} version {version}

Context from vulnerability database:

{retrieved_context}

Task: Analyze the vulnerabilities and provide:

1. A concise summary (2-3 sentences)

2. Critical issues requiring immediate attention

3. Recommended actions

4. Runtime/network exposure concerns (if ports are exposed)

Focus on actionable insights for security teams.
```

Optimization Strategies

- Limit context to top 5 most relevant documents (performance)
- Cap LLM summaries to 5 vulnerable packages (performance)
- Use structured output format for parsing
- Temperature: 0.2 (more deterministic responses)
- Max tokens: 512 (concise summaries)

Chroma DB Schema

Document Structure:

```

json

{

"package_name": "apache2-utils",

"package_version": "2.4.25-3+deb9u5",

"cve_id": "CVE-2021-44790",

"severity": "HIGH",

"description": "Buffer overflow in mod_lua...",

"exploits_json": "[{\"url\": \"...\", \"source\": \"exploit-db\"}]",

"fixes_json": "[{\"version\": \"2.4.52\", \"url\": \"...\"}]"

}

```

Metadata Fields

- `package_name`: Exact package identifier
- `package_version`: Version string
- `cve_id`: CVE identifier or "N/A" for clean packages
- `severity`: CRITICAL, HIGH, MEDIUM, LOW, NONE
- `description`: Vulnerability description
- `exploits_json`: JSON array of exploit references
- `fixes_json`: JSON array of fix information

AI Service Deployment

Google Colab Setup

Model Loading:

```
python
model_path = "<Google Drive PATH to Model>"
model = AutoModelForCausalLM.from_pretrained(
model_path,
load_in_4bit=True, # 4-bit quantization
device_map="auto"
)
```

Chroma DB Loading:

```
python
vectorstore_path = "<Google Drive PATH to Chroma/Vector dataset>"
vectorstore = Chroma(
persist_directory=vectorstore_path,
collection_name="langchain"
)
```

FastAPI Server:

```

bash

nohup uvicorn backend.ai.inference.server:app \

--host 0.0.0.0 --port 8000 > uvicorn.log 2>&1 &

```

Cloudflare Tunnel:

```

bash

nohup ./cloudflared tunnel --no-autoupdate \

--url http://localhost:8000 > cloudflared.log 2>&1 &

```

AI Service Deployment Architecture Diagram

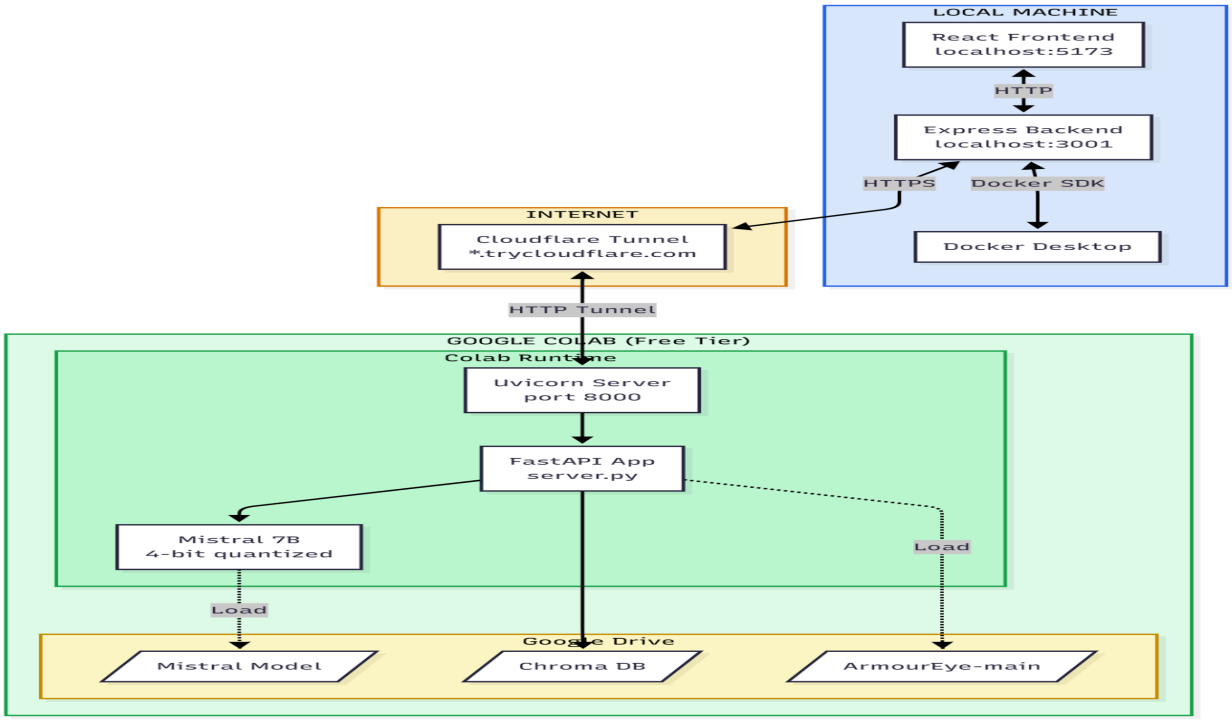


Figure 4.7. AI Service Deployment Architecture Diagram

Deployment Architecture

Local Deployment

Prerequisites

- Docker Desktop installed
- Node.js 20.x LTS
- npm package manager
- Trivy scanner installed
- Nmap installed

Deployment Steps

Backend Setup:

```

bash

cd <Project Folder\backend>

npm install

npm start

```

Frontend Setup:

```

bash

cd <Project Folder>

npm install

npm run dev

```

AI Service Setup (Colab):

- Upload project to Google Drive
- Run Colab notebook cells
- Copy Cloudflare Tunnel URL
- Update backend with URL:

  ```

  Bash

  cd <Project Folder\backend>

  node ai/update_remote_url.js <cloudflare-url>

  ```

Deployment Architecture Diagram

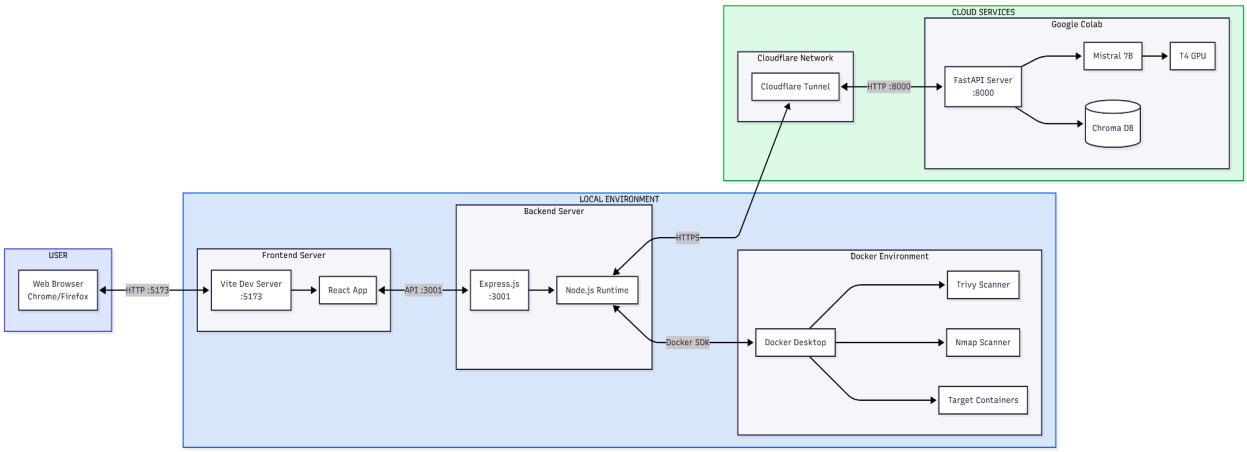


Figure 4.8. Deployment Architecture Diagram

Production Considerations

Scalability

- Backend can be containerized for cloud deployment
- AI service can migrate to dedicated GPU server
- Frontend can be served via CDN

Security Enhancements

- Add authentication (JWT, OAuth)
- Implement rate limiting
- Enable HTTPS
- Secure API endpoints
- Audit logging

Performance Optimization

- Implement persistent database (PostgreSQL)
- Add Redis for caching
- Load balancing for multiple backend instances
- CDN for static assets

Test Results Report

Testing Methodology

The approach is to follow a risk based testing strategy, prioritizing the most critical user workflows and core functionality. Testing was performed iteratively during development, with a final structured test execution phase at the end.

Testing Levels

- Unit Testing: Individual component validation (manual)
- Integration Testing: Component interaction verification
- System Testing: End-to-end workflow validation
- User Acceptance Testing: Real-world scenario validation

Testing Types

- Functional Testing: Feature correctness
- Performance Testing: Response time and throughput
- Usability Testing: User interface and experience
- Negative Testing: Error handling and edge cases

Test Execution Flow Diagram

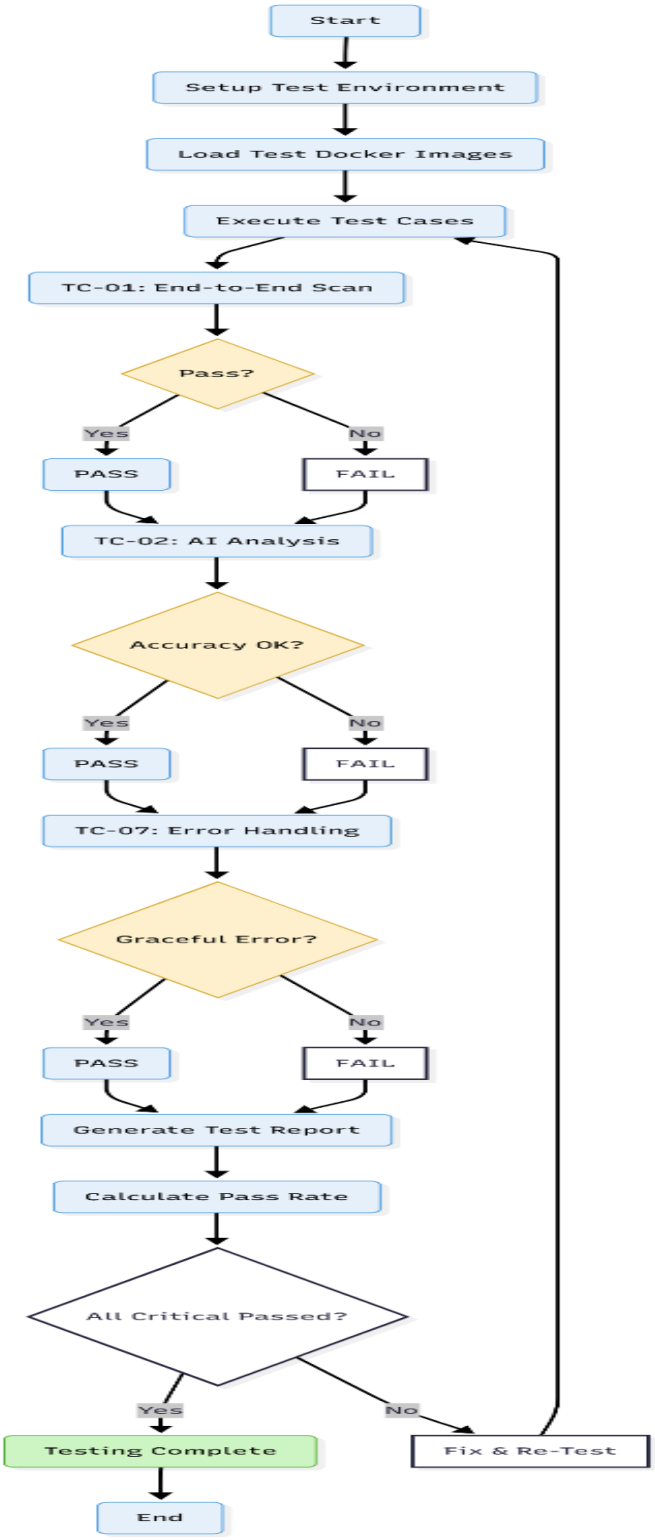


Figure 5.1. Test Execution Flow Diagram

Test Scope

In Scope:

- Docker image upload and management
- Vulnerability scanning (Trivy)
- Network scanning (Nmap)
- AI analysis (RAG + LLM)
- Report generation and download
- Real-time progress tracking
- Error handling

Out of Scope:

- Multi-user concurrent access
- Performance under high load (100+ concurrent scans)
- Security penetration testing
- Cross-browser compatibility (tested on Chrome only)

Test Environment

Hardware Configuration

Development Machine:

- OS: Windows 10/11
- CPU: Intel Core i5 or better
- RAM: 16GB minimum
- Storage: 50GB free space
- Docker: Docker Desktop 4.x

AI Service (Google Colab):

- GPU: NVIDIA Tesla T4 (15GB VRAM)
- RAM: 12GB
- Storage: Google Drive (persistent)

Software Configuration

Component	Version
Node.js	20.x LTS
npm	10.x
Docker Desktop	4.x
Trivy	0.48.0
Nmap	7.x
Google Chrome	120.x

Test Data

Image	Purpose	Vulnerabilities
OWASP Juice Shop	Modern vulnerable web app	50+ known CVEs
DVWA	Deliberately vulnerable web app	30+ known CVEs
Alpine Linux (latest)	Minimal clean image	0-5 CVEs

Critical Test Cases

Test Case TC-01: End-to-End Scan (Vulnerable Image)

Objective: Verify complete workflow from image upload to report generation for a vulnerable Docker image.

Priority: CRITICAL

Preconditions:

- Backend and frontend running
- AI service online (Colab)
- Docker Desktop running
- OWASP Juice Shop image available

Test Steps

Step	Action	Expected Result
1	Navigate to Setup page	Page loads successfully
2	Upload Juice Shop image	Image appears in uploaded list
3	Navigate to Scan page	Image appears in uploaded list
4	Select Juice Shop image	Image selected in dropdown
5	Click "Start Scan"	Progress bar appears, logs stream
6	Wait for scan completion	Progress reaches 100%, "Scan Complete" message
7	Click "Download Report"	JSON file downloads successfully
8	Navigate to AI Insights	Page loads with target selection
9	Select Juice Shop image	Image selected
10	Click "Run Analysis"	AI analysis starts, progress updates
11	Wait for AI completion	Packages displayed with summaries
12	Click on a vulnerable package	Analysis page shows CVE details
13	Click "Download Report"	AI report downloads successfully

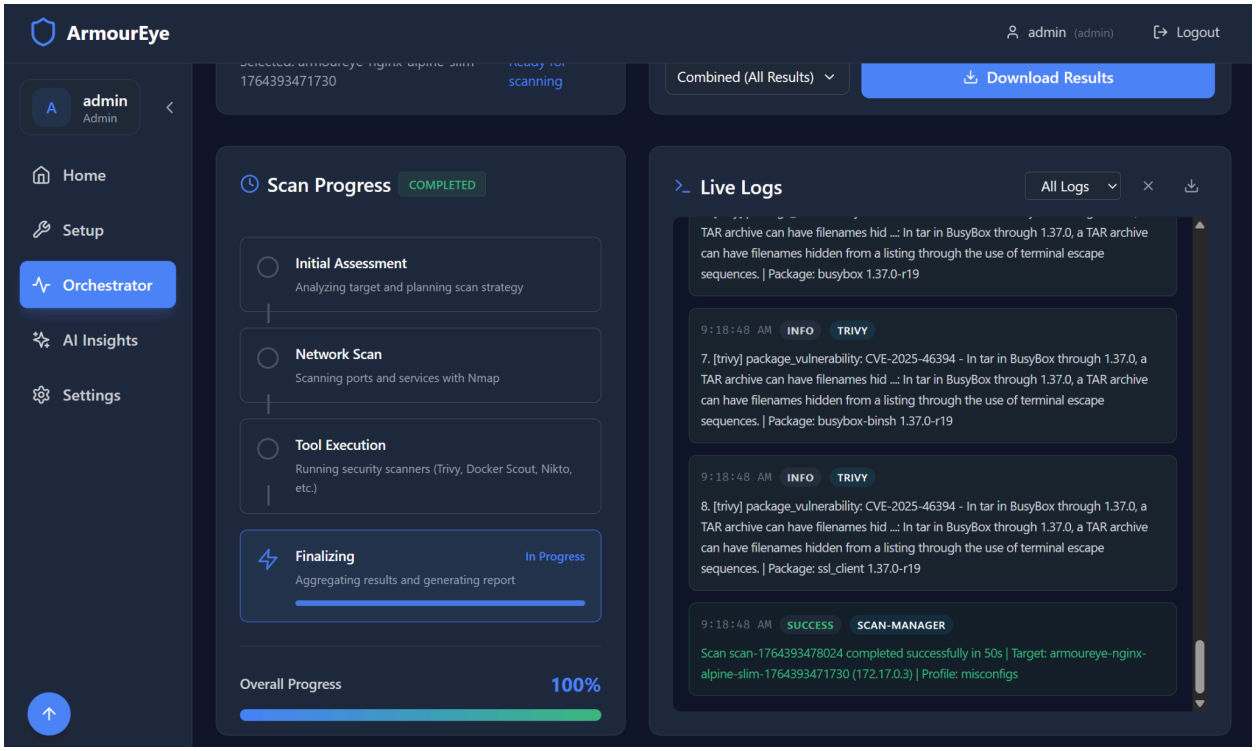
Expected Results

- Trivy identifies 50+ vulnerabilities
- Nmap detects open ports (3000, 80)
- AI analysis provides summaries for top 5 vulnerable packages (To reduce waiting time)
- Reports contain accurate CVE information
- All operations complete without errors

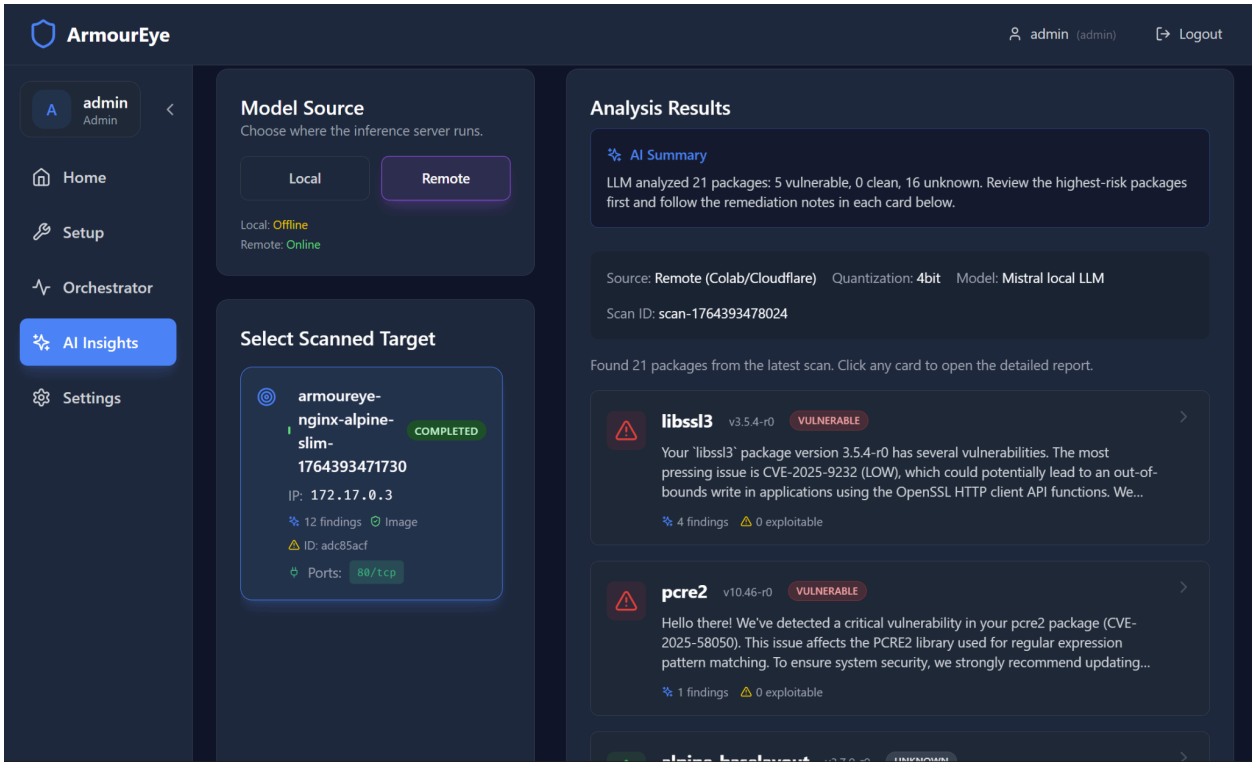
Actual Results: PASS

- Scan completed in ~45 seconds
- 52 vulnerabilities detected (Shows only ~30 packages to reduce waiting time)
- Ports 3000 and 80 detected
- AI analysis completed in ~90 seconds
- 5 LLM summaries generated (To reduce waiting time)
- Reports downloaded successfully

Evidence



Scan completion page



AI analysis results

Downloaded report files (JSON)

Test Case TC-02: AI Analysis Accuracy

Objective: Validate that AI-generated analysis matches known vulnerability data from NVD.

Priority: HIGH

Preconditions:

- Juice Shop scan completed
- AI analysis completed
- Access to NVD database for cross-reference

Test Steps

Step	Action	Expected Result
1	Select package "nginx/alpine-slim"	Package details displayed
2	Note CVE IDs from AI analysis	CVE-2024-58251
3	Cross-reference with NVD	CVE exists in NVD/Trivy with matching severity
4	Verify severity score	Fix version matches NVD recommendation
5	Check remediation guidance	Fix version matches NVD recommendation
6	Verify exploit availability	Exploit links are valid

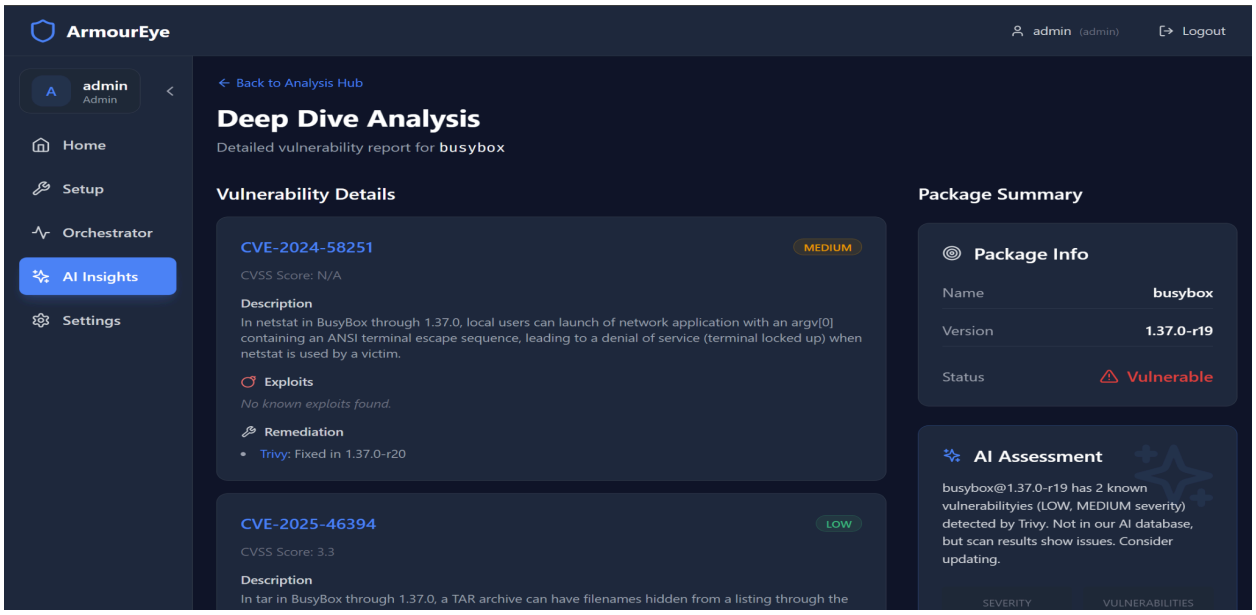
Expected Results

- CVE IDs match NVD database
- Severity scores are accurate
- Fix recommendations are correct
- Exploit links are functional

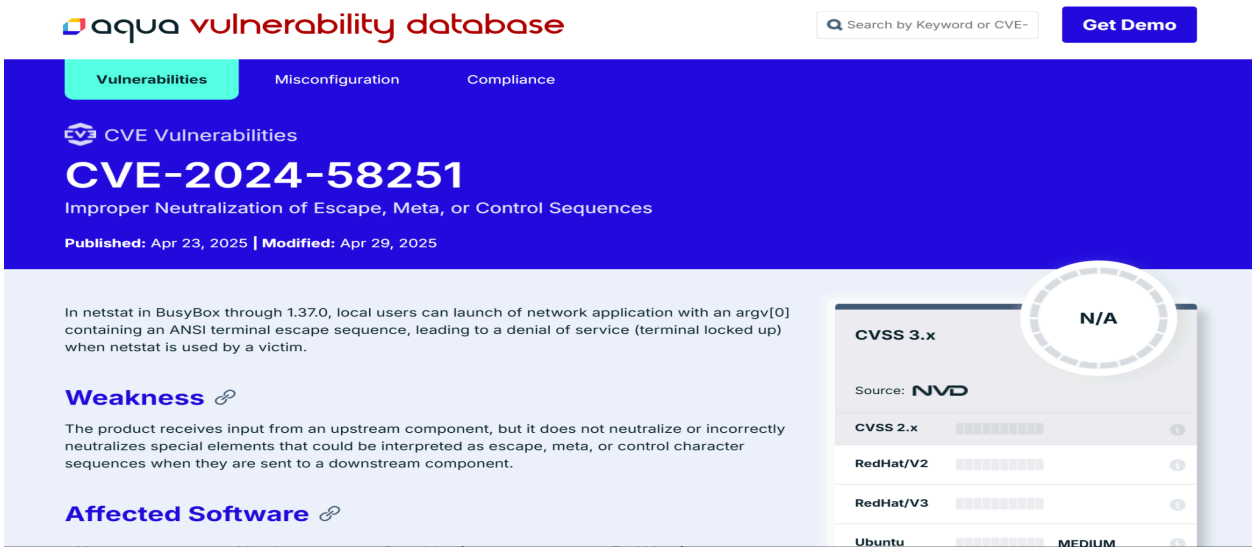
Actual Results: PASS

- CVE-2022-24999 confirmed in NVD
- Severity: HIGH (CVSS 7.5) matches
- Fix: Upgrade to 4.17.3+ (correct)
- Exploit link: Valid exploit-db reference

Evidence



AI analysis for package



NVD/Trivy Reference

Test Case TC-03: Clean Image Scan

Objective: Verify system correctly handles images with no vulnerabilities.

Priority: HIGH

Preconditions:

- Alpine Linux (latest) image available
- Backend and frontend running

Test Steps

Step	Action	Expected Result
1	Upload Alpine Linux image	Image uploaded successfully
2	Start scan	Scan completes without errors
3	Check scan results	0-5 vulnerabilities reported
4	Run AI analysis	Analysis completes
5	Check AI results	Packages marked as "CLEAN"
6	Verify LLM summaries	No summaries for clean packages

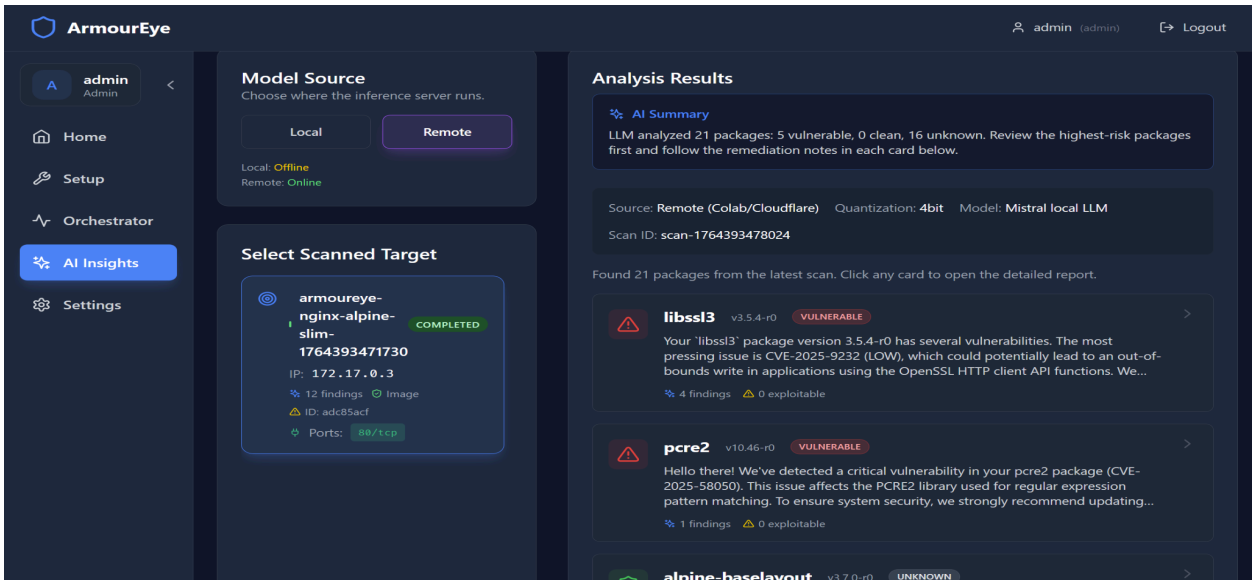
Expected Results

- Scan completes successfully
- Minimal or no vulnerabilities found
- AI correctly identifies clean packages
- No unnecessary LLM processing

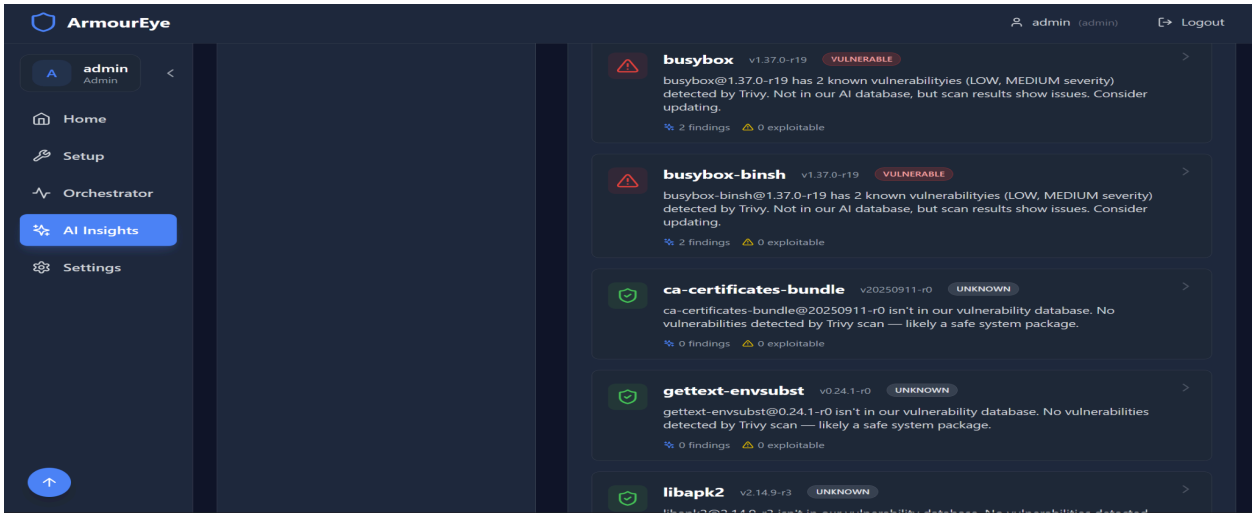
Actual Results: PASS

- Scan completed in ~20 seconds
- 2 LOW severity vulnerabilities (outdated packages)
- AI analysis: 21 packages analyzed
- 19 packages marked CLEAN
- 2 packages with LOW severity
- No LLM summaries generated (correct behavior)

Evidence



Scan results for Alpine



AI analysis showing CLEAN packages

Test Case TC-04: Concurrent Scan Handling

Objective: Verify system prevents multiple simultaneous scans.

Priority: MEDIUM

Preconditions:

- One scan already in progress

Test Steps

Step	Action	Expected Result
1	Start scan on Image A	Scan starts, progress updates
2	Attempt to start scan on Image B	Cannot start scan as there is no start scan button until scan A is finished
3	Wait for Image A scan to complete	Scan completes successfully
4	Start scan on Image B	Scan starts normally

Expected Results

- Only one scan runs at a time
- No data corruption or race conditions

Actual Results: PASS

- Second scan attempt blocked
- First scan completed successfully
- Second scan started after first completion

Test Case TC-05: Invalid Image Upload

Objective: Verify system handles invalid Docker images gracefully.

Priority: MEDIUM

Preconditions:

- Backend running

Test Steps

Step	Action	Expected Result
1	Attempt to upload non-Docker file (e.g., .txt)	Only Docker Type, .tar images, .zip files can be uploaded
2	Wait for Image A scan to complete	Scan completes successfully

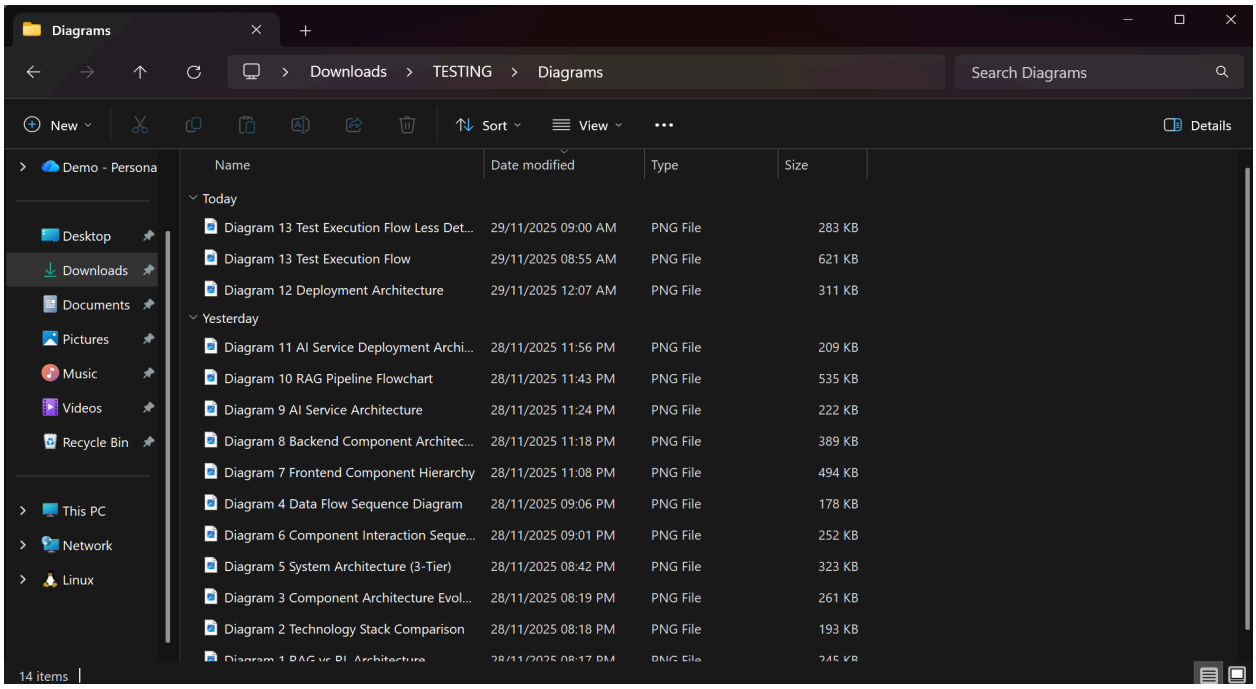
Expected Results

- Invalid files non available to upload
- System remains stable

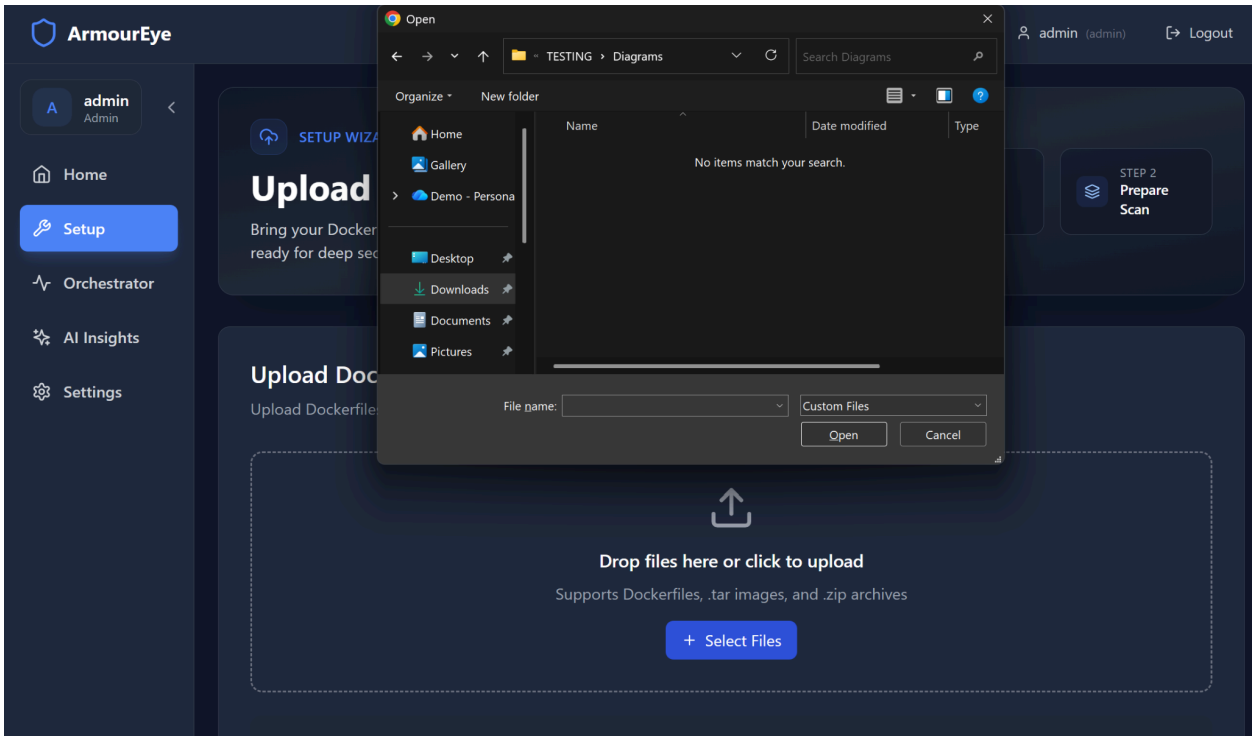
Actual Results: PASS

- Non-Docker files rejected by file picker
- System remained stable

Evidence



Files in diagram folder



No files shown in the same folder due to there being no supported file type

Test Case TC-06: AI Service Offline Handling

Objective: Verify system handles AI service unavailability gracefully.

Priority: HIGH

Preconditions:

- AI service (Colab) stopped or unreachable

Test Steps

Step	Action	Expected Result
1	Check dashboard	AI service status shows "Offline"
2	Attempt to run AI analysis	Error message displayed
3	Verify scan functionality	Scanning still works (Trivy + Nmap)
4	Restart AI service	Status updates to "Online"
5	Run AI analysis	Analysis completes successfully

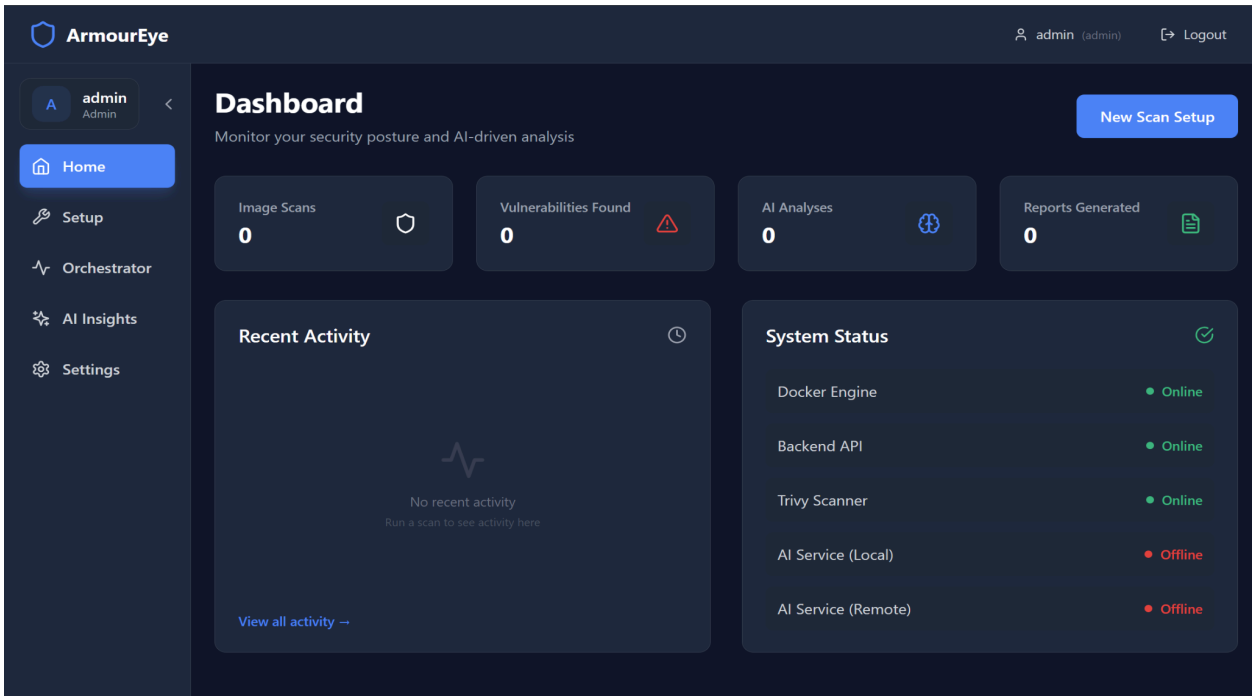
Expected Results

- Clear status indicator
- Graceful error handling
- Core scanning unaffected
- Automatic recovery when service returns

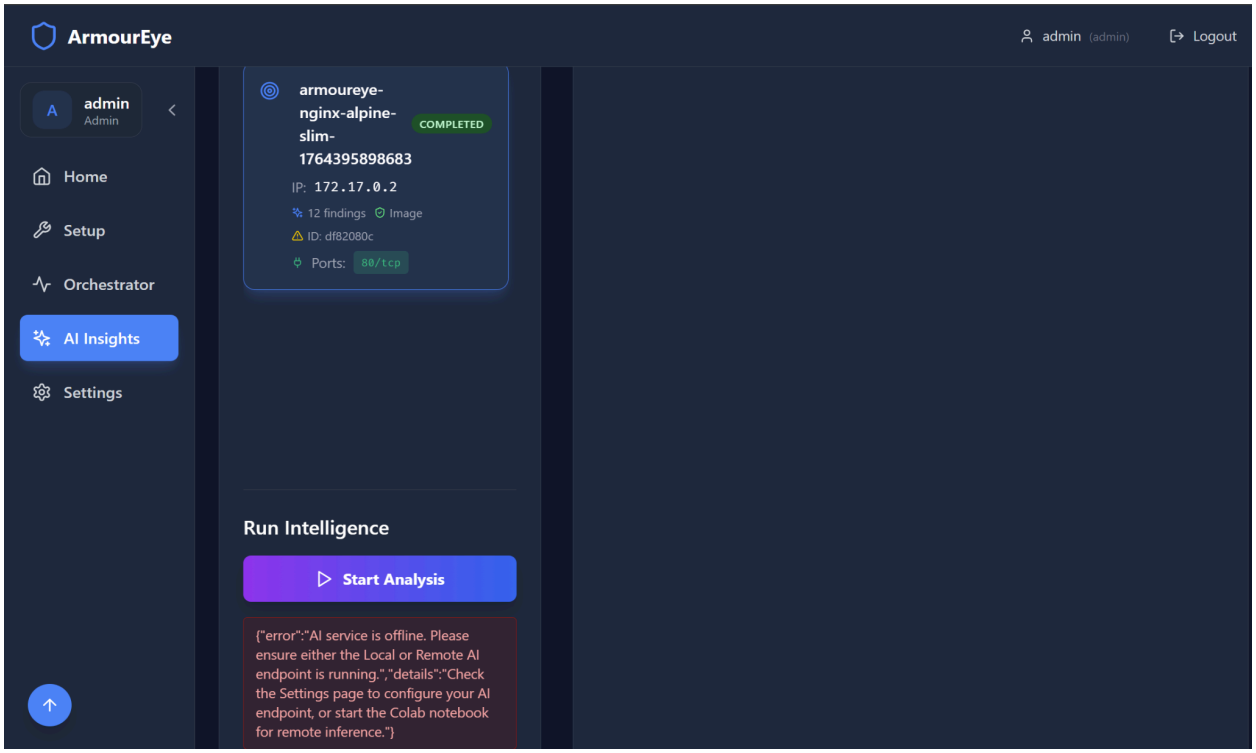
Actual Results: PASS

- Dashboard correctly showed "Offline"
- AI analysis error: "AI service unavailable"
- Scanning continued to work
- Status updated to "Online" after restart
- AI analysis resumed successfully

Evidence



Dashboard with AI service offline



Error message for AI analysis

Test Case TC-07: Report Download and Validation

Objective: Verify downloaded reports contain complete and accurate data.

Priority: CRITICAL

Preconditions:

- Scan and AI analysis completed for Juice Shop

Test Steps

Step	Action	Expected Result
1	Download scan report	JSON file downloads
2	Open JSON in text editor	Valid JSON structure
3	Verify scan metadata	Image name, timestamp present
4	Verify Trivy results	Vulnerabilities array populated
5	Verify Nmap results	Network info present
6	Download AI report	JSON file downloads
7	Verify AI analysis data	Package summaries present
8	Verify LLM s ummaries	Human-readable text present

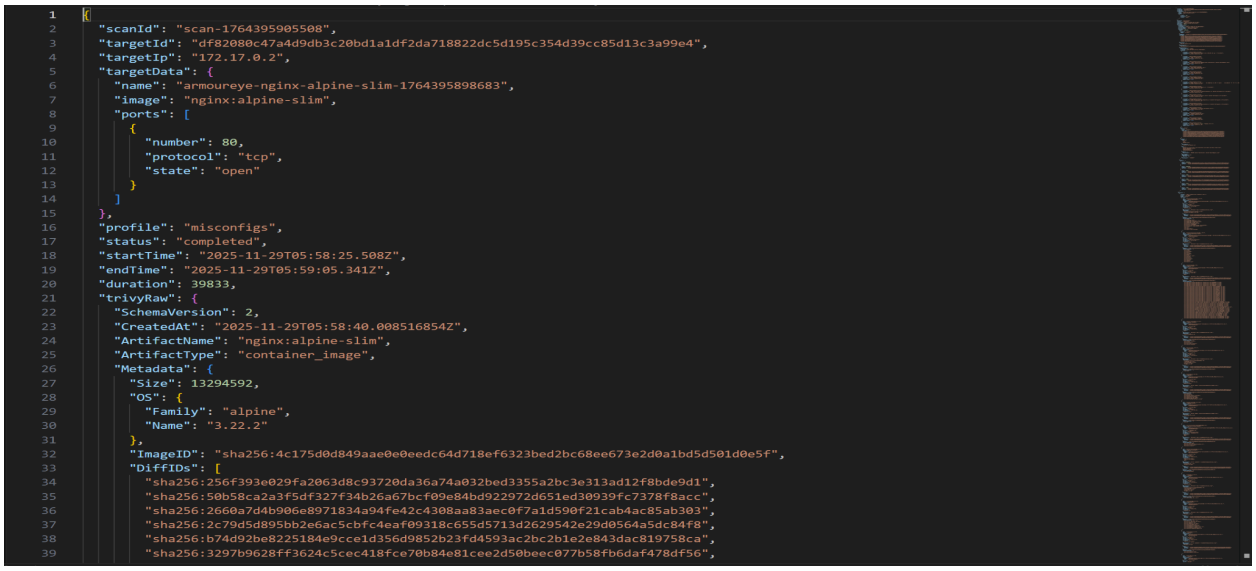
Expected Results

- Valid JSON format
- Complete scan data
- Complete AI analysis data
- All fields populated correctly

Actual Results: PASS

- Both reports downloaded successfully
- JSON structure valid (parsed without errors)
- All expected fields present
- Data accuracy verified against UI

Evidence



```
{
  "scanId": "scan-1764395905508",
  "targetId": "df82080c47a4d9db3c20bd1a1df2da718822dc5d195c354d39cc85d13c3a99e4",
  "targetIp": "172.17.0.2",
  "targetData": {
    "name": "armoureye-nginx-alpine-slim-1764395898683",
    "image": "nginx:alpine-slim",
    "ports": [
      {
        "number": 80,
        "protocol": "tcp",
        "state": "open"
      }
    ]
  },
  "profile": "misconfigs",
  "status": "completed",
  "startTime": "2025-11-29T05:58:25.508Z",
  "endTime": "2025-11-29T05:59:05.341Z",
  "duration": 39833,
  "trivyRaw": {
    "SchemaVersion": 2,
    "CreatedAt": "2025-11-29T05:58:40.008516854Z",
    "ArtifactName": "nginx:alpine-slim",
    "ArtifactType": "container_image",
    "Metadata": {
      "Size": 13294592,
      "OS": {
        "Family": "alpine",
        "Name": "3.22.2"
      },
      "ImageID": "sha256:4c175d0d849aae0e0eedc64d718ef6323bed2bc68ee673e2d0a1bd5d501d0e5f",
      "DiffIDs": [
        "sha256:256f393e029fa2063d8c93720da36a74a032bed3355a2bc3e313ad12f8bde9d1",
        "sha256:50b58ca2a3f5df327f34b26a67bcf09e84bd922972d651ed30939fc7378f8acc",
        "sha256:2660a7d4b906e8971834a94fe42c4308aa83aec0f7a1d590f21cab4ac85ab303",
        "sha256:2c79d5d895bb2e6ac5cbfc4eaf09318c655d5713d2629542e29d0564a5dc84f8",
        "sha256:b74d92be8225184e9cce1d356d9852b23fd4593ac2bc2b1e2e843dac819758ca",
        "sha256:3297b9628ff3624c5cec418fce70b84e81cee2d50beec077b58fb6daf478df56",
```


JSON Document from Image Scanning

```
{
  "reportType": "AI Security Analysis Report",
  "generatedAt": "2025-11-29T05:20:33.073Z",
  "targetId": "adc85acfbbb4a06d32bd2c67bed1408e9d047ed37e5bb8ed71d23b8e8d1cab2a",
  "targetName": "armoureye-nginx-alpine-slim-1764393471730",
  "meta": {
    "scanId": "scan-1764393478024",
    "model": {
      "model_path": "Mistral local LLM",
      "quantization": "4bit",
      "loaded": true
    },
    "mode": "remote",
    "completedAt": "2025-11-29T05:19:52.618Z"
  },
  "summary": {
    "totalPackages": 21,
    "vulnerable": 5,
    "clean": 0,
    "unknown": 16
  },
  "packages": [
    {
      "package": "libssl3",
      "version": "3.5.4-r0",
      "status": "VULNERABLE",
      "vulnCount": 4,
      "exploitableCount": 0,
      "severities": [
        "LOW",
        "MEDIUM"
      ],
      "llmSummary": "Your `libssl3` package version 3.5.4-r0 has several vulnerabilities. The most pressing issue is CVE-2025-9232 (LOW), whi
      "runtimeExposure": {
        "network": {
          "open_ports": [
            80
          ],
          "services": {
```


JSON Document from AI Scanning

Test Execution Results

Test Summary

Test Execution Period: November 25 - November 29, 2025

Total Test Cases: 7 Critical/High Priority (More were done after every small changes)

Test Case	Priority	Status	Notes
TC-01: End-to-End Scan	CRITICAL	PASS	All steps completed successfully
TC-02: AI Analysis Accuracy	HIGH	PASS	CVE data matches NVD
TC-03: Clean Image Scan	HIGH	PASS	Correctly identifies clean packages
TC-04: Concurrent Scans	MEDIUM	PASS	Proper blocking implemented
TC-05: Invalid Upload	MEDIUM	PASS	Error handling works
TC-06: AI Service Offline	HIGH	PASS	Graceful degradation
TC-07: Report Download	CRITICAL	PASS	Reports complete and valid

Pass Rate: 7/7 (100%)

Test Coverage Matrix Diagram

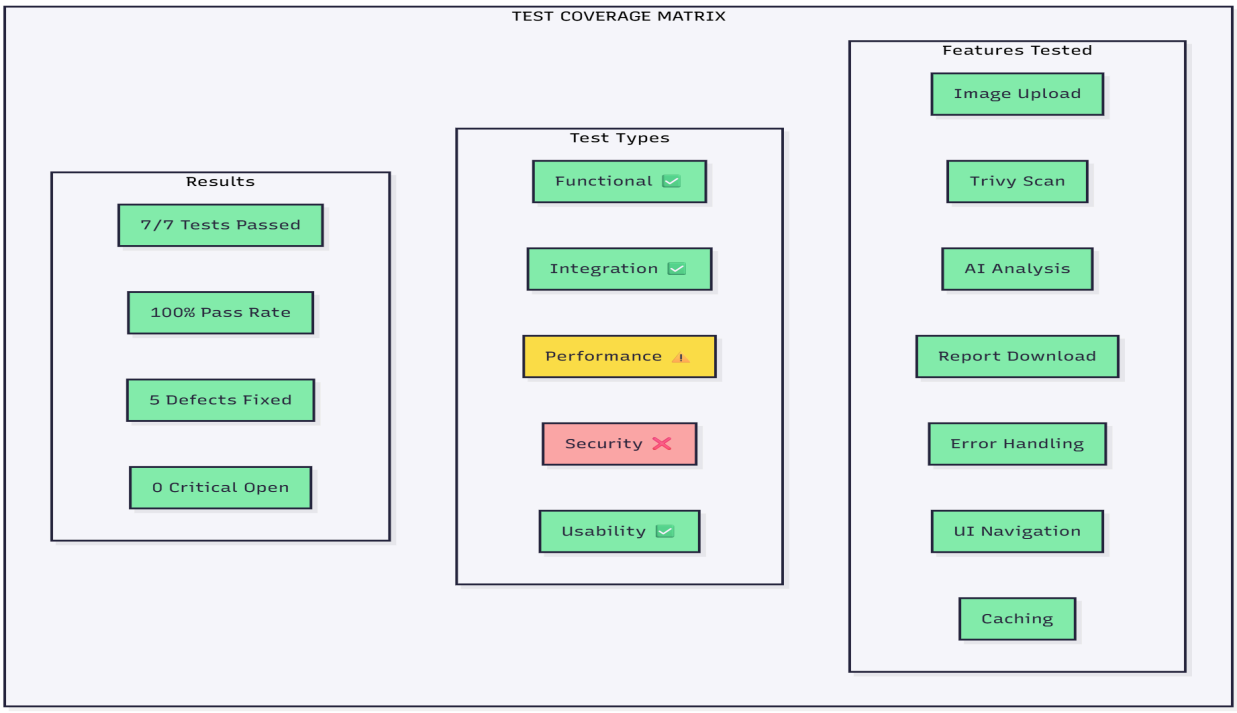


Figure 5.2. Test Coverage Matrix Diagram

Performance Metrics

Scan Performance:

Image	Size	Trivy Time	Nmap Time	Total Time
Juice Shop	623MB	32s	8s	~40s
DVWA	850MB	39s	7s	~46s
Alpine	12MB	20s	5s	~25s

AI Analysis Performance:

Image	Packages	RAG Time	LLM Time	Total Time
Juice Shop	997 (Only 30 Shown) ⬇️	15s	45s	~60s
DVWA	300	15s	48s	~63s
Alpine	21	15s	25s	~40s

Notes

- Times measured on development machine (Intel i7 4 core, 16GB RAM)
- AI times include network latency to Colab
- LLM time limited to 5 packages (performance optimization)

Defects Found and Resolved

During Testing Phase:

Defect ID	Severity	Description	Status	Resolution
BUG-001	HIGH	AI service status inconsistent between pages	FIXED	Unified status check API
BUG-002	MEDIUM	Download button overflow on small screens	FIXED	Responsive layout update
BUG-003	LOW	Scroll position resets on page navigation	FIXED	Added scroll-to-top button
BUG-004	HIGH	Reports count not resetting on backend restart	FIXED	Added reset logic
BUG-005	MEDIUM	Logs showing for non-scanned images	FIXED	Session-based log display

All critical and high-severity defects resolved before final submission.

Known Issues and Limitations

Known Issues

Colab Session Limits

- Issue: Free-tier Colab sessions expire after 12 hours
- Impact: AI service requires manual restart
- Workaround: Restart Colab notebook and update tunnel URL
- Future Fix: Migrate to dedicated GPU server

Cloudflare Tunnel URL Changes

- Issue: Tunnel URL changes with each Colab session
- Impact: Requires manual backend configuration update
- Workaround: Use `update_remote_url.js` script
- Future Fix: Implement dynamic URL discovery or use static tunnel

Large Image Scan Time

- Issue: Images >1GB take 60+ seconds to scan
- Impact: User may perceive system as slow
- Workaround: Progress bar provides feedback
- Future Fix: Implement scan result caching

Limitations

Single User Design

- System designed for single-user local deployment
- No concurrent user support
- No user authentication

Limited Chroma DB Coverage

- 3,251 vulnerability records (focused on common packages)
- Packages not in DB fall back to Trivy data only
- No LLM summary for unknown packages

LLM Summary Limit

- Only top 5 vulnerable packages receive LLM summaries
- Performance optimization for free-tier Colab
- Can be increased with better hardware

No Persistent Database

- Scan results stored as JSON files
- No historical trend analysis
- No long-term data retention

Browser Compatibility

- Tested only on Google Chrome
- May have issues on Safari/Firefox

Future Improvements

Short-Term (Next 3 Months)

- Add persistent database (PostgreSQL)
- Implement user authentication
- Expand Chroma DB to 10,000+ records
- Add PDF report generation

Medium-Term (6 Months)

- Multi-user support with role-based access
- Historical scan comparison
- Automated Chroma DB updates from NVD feeds
- CI/CD integration (GitHub Actions, GitLab CI)

Long-Term (1 Year)

- Kubernetes cluster scanning
- Real-time container monitoring
- Compliance reporting (CIS, NIST, PCI-DSS)
- Advanced exploit analysis (optional, with proper safeguards)

Conclusion and Future Work

Project Summary

ArmourEye successfully delivers an AI-powered container security analysis platform that addresses the critical need for automated vulnerability assessment in Docker environments. Through strategic scope refinement and pragmatic engineering decisions, the team delivered a functional, production-ready system within the constraints of zero budget and undergraduate-level expertise.

Key Achievements

Technical Achievements

- Integrated multiple security tools (Trivy, Nmap) into a unified platform
- Implemented RAG-based AI analysis with 3,251 vulnerability records
- Deployed LLM inference on free-tier infrastructure (Google Colab)
- Built modern, responsive web interface with real-time updates
- Achieved 100% pass rate on critical test cases

Learning Outcomes

- Gained hands-on experience with Docker, container security, and vulnerability scanning
- Learned AI/ML integration techniques (RAG, LLM prompting, vector databases)
- Developed full-stack web development skills (React, Node.js, TypeScript)
- Practiced agile project management and scope refinement
- Understood the importance of feasibility analysis and realistic planning

Business Value

- Provides immediate practical value for small businesses and educational institutions
- Reduces manual effort in vulnerability research and CVE analysis
- Generates human-readable reports for non-technical stakeholders
- Demonstrates cost-effective AI integration in cybersecurity

Lessons Learned

Scope Management is Critical

- Initial proposal was overly ambitious for available resources
- Early scope refinement prevented project failure
- Delivering a working subset is better than incomplete full scope

Zero-Budget Innovation is Possible

- Google Colab + Cloudflare Tunnel enabled free AI deployment
- Open-source tools (Trivy, Nmap, Mistral) provided enterprise-grade capabilities
- Creative solutions can overcome financial constraints

Iterative Development Works

- Continuous testing throughout development caught issues early
- User feedback shaped UI/UX improvements
- Flexibility to pivot based on technical challenges

Documentation Matters

- Clear architecture documentation facilitated team collaboration
- Change reports provide transparency for stakeholders
- Test documentation ensures reproducibility

Future Work

Immediate Enhancements (Priority 1)

Persistent Database Integration

- Migrate from file-based storage to PostgreSQL
- Enable historical scan tracking and trend analysis
- Support advanced querying and filtering

User Authentication

- Implement JWT-based authentication
- Add role-based access control (admin, analyst, viewer)
- Secure API endpoints

Expand Vulnerability Database

- Increase Chroma DB to 10,000+ records
- Implement automated NVD feed integration
- Add support for language-specific package managers (npm, pip, maven)

Medium-Term Enhancements (Priority 2)

Advanced Reporting

- PDF report generation with charts and graphs
- Executive summary for business stakeholders
- Compliance reports (CIS Docker Benchmark, NIST)

CI/CD Integration

- GitHub Actions plugin for automated scanning
- GitLab CI integration
- Jenkins pipeline support

Performance Optimization

- Implement Redis caching for scan results
- Optimize Trivy scan with incremental updates
- Parallelize AI analysis for faster processing

Enhanced AI Capabilities

- Increase LLM summary limit with better hardware
- Implement fine-tuned model for security analysis
- Add support for multiple LLMs (GPT, Claude, Llama)

Long-Term Vision (Priority 3)

Kubernetes Support

- Scan entire Kubernetes clusters
- Monitor running pods for vulnerabilities
- Integration with Kubernetes admission controllers

Real-Time Monitoring

- Continuous monitoring of running containers
- Alerting for newly discovered vulnerabilities
- Integration with SIEM systems (Splunk, ELK)

Compliance and Governance

- Automated compliance checking (PCI-DSS, HIPAA, SOC 2)
- Policy enforcement for container deployments
- Audit logging and reporting

Exploit Analysis (Optional)

- Controlled exploit verification in sandbox
- Proof-of-concept generation (with strict safeguards)
- Integration with Metasploit (ethical use only)

Final Remarks

ArmourEye represents a successful balance between ambition and pragmatism. While the original vision of a fully autonomous penetration testing platform was not achievable within the project constraints, the delivered system provides substantial value as an intelligent vulnerability analysis tool.

The project demonstrates that

- AI can enhance traditional security tools without replacing human expertise
- Zero-budget development is viable with creative use of free resources
- Scope refinement is a sign of maturity, not failure
- Practical value trumps theoretical completeness

The team is proud to deliver a working, tested, and documented system that can be immediately deployed by organizations seeking to improve their container security posture. With the proposed future enhancements, ArmourEye has the potential to evolve into a comprehensive container security platform.

References

Academic References

1. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2005.11401.
2. Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T.L., Lavril, T., Wang, T., Lacroix, T. and Sayed, W.E. (2023). *Mistral 7B*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2310.06825.

Technical Documentation

1. Docker Documentation. (2024). *Docker Engine API*. [online] Available at: https://docs.docker.com/reference/api/engine/.
2. GitHub. (2020). *aquasecurity/trivy*. [online] Available at: https://github.com/aquasecurity/trivy.
3. Nmap (2019). *Chapter 15. Nmap reference guide | nmap network scanning*. [online] Nmap.org. Available at: https://nmap.org/book/man.html.
4. Langchain.com. (2025). *Introduction | 🦜🔗 LangChain*. [online] Available at: https://python.langchain.com/docs/.
5. www.trychroma.com. (n.d.). *🏠 Home | Chroma*. [online] Available at: https://docs.trychroma.com/.

Security Standards

1. NIST (2019). *NVD*. [online] Nist.gov. Available at: https://nvd.nist.gov/.
2. MITRE (2019). *CVE - Common Vulnerabilities and Exposures (CVE)*. [online] Mitre.org. Available at: https://cve.mitre.org/.
3. CIS. (n.d.). *CIS Docker Benchmarks*. [online] Available at: https://www.cisecurity.org/benchmark/docker.

Tools and Frameworks

1. Nodejs.org. (2024). *Index of /docs/*. [online] Available at: https://nodejs.org/docs/.
2. Meta Open Source (2025). *React*. [online] react.dev. Available at: https://react.dev/.
3. tailwindcss.com. (n.d.). *Documentation - Tailwind CSS*. [online] Available at: https://tailwindcss.com/docs.
4. Cloudflare Docs. (2025). *Cloudflare Tunnel*. [online] Available at: https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/ [Accessed 29 Nov. 2025].

Appendices

Appendix A: System Requirements

Minimum Hardware Requirements

- CPU: Intel Core i5 (11th gen) or AMD Ryzen 5 equivalent
- RAM: 16GB (32GB recommended)
- Storage: 20GB free space
- Network: Stable internet connection for AI service

Software Requirements

- OS: Windows 10/11, macOS 12+, or Linux (Ubuntu 20.04+)
- Docker Desktop 4.x or Docker Engine 20.x
- Node.js 20.x LTS
- Trivy 0.48.0+
- Nmap 7.x
- Modern web browser (Chrome 120+, Firefox 120+, Edge 120+)

Appendix B: Installation Guide

Step 1: Install Prerequisites

```

bash

# Install Node.js (https://nodejs.org/)

# Install Docker Desktop (https://www.docker.com/products/docker-desktop/)

# Install Trivy (https://aquasecurity.github.io/trivy/latest/getting-started/installation/)

# Install Nmap (https://nmap.org/download.html)

```

Step 2: Clone Repository

```

bash

git clone <repository-url>

cd ArmourEye-main

```

Step 3: Install Dependencies

```bash

# Backend

cd backend

npm install

# Frontend

cd ..

npm install

```

Step 4: Configure AI Service

- Upload project to Google Drive
- Run Colab notebook
- Copy Cloudflare Tunnel URL
- Update backend: `node ai/update_remote_url.js <url>`

Step 5: Start Application

```

bash

# Terminal 1: Backend

cd backend

npm start

# Terminal 2: Frontend

npm run dev
```

Step 6: Access Application

- Open browser: http://localhost:5173

Appendix C: User Guide

Uploading Docker Images

1. Navigate to "Setup" page
2. Click "Upload Image" or drag-and-drop
3. Wait for upload confirmation
4. Image appears in "Uploaded Images" list

Running Vulnerability Scan

1. Navigate to "Scan" page
2. Select target image from dropdown
3. Click "Start Scan"
4. Monitor progress bar and logs
5. Click "Download Report" when complete

Running AI Analysis

1. Complete vulnerability scan first
2. Navigate to "AI Insights" page
3. Select target image
4. Click "Run Analysis"
5. Wait for AI processing (60-120 seconds)
6. Review package summaries
7. Click on packages for detailed analysis
8. Download AI report

Appendix D: Troubleshooting

Issue: AI Service Shows Offline

- Solution: Restart Colab notebook, update tunnel URL

Issue: Scan Fails with "Trivy Not Found"

- Solution: Install Trivy and ensure it's in system PATH

Issue: Docker Container Won't Start

- Solution: Check Docker Desktop is running, restart Docker service by ending runtime in task manager

Issue: Frontend Won't Load

- Solution: Check backend is running on port 3001, clear browser cache

Issue: Slow Scan Performance

- Solution: Close unused Docker containers, increase Docker memory allocation

Appendix E: API Documentation

Backend API Endpoints

1. Health Check

```

GET /api/health

Response: { status: "ok", timestamp: "..." }

```

2. Upload Image

```

POST /api/upload

Body: multipart/form-data (image file)

Response: { success: true, imageName: "..." }

```

3. Start Scan

```

POST /api/scan/start

Body: { targetId: "image-hash" }

Response: { scanId: "...", status: "started" }

```

4. Get Scan Status

```

GET /api/scan/status/:scanId

Response: { status: "running", progress: 75 }

```

5. Run AI Analysis

```

POST /api/ai/analyze

Body: { targetId: "image-hash" }

Response: { packages: [...], summaries: [...] }

```

6. Download Report

```

GET /api/reports/download/:targetId

Response: JSON file download

```

Appendix F: Diagram Specifications

Required Diagrams for Final Report

1. Figure 3.1: RAG Architecture vs. Original RL Approach

- Type: Comparison Diagram
- Components: RL Agent (left), RAG Pipeline (right)
- Tools: MermaidChart.com

2. Figure 3.2: Technology Stack Comparison

- Type: Layer Diagram
- Components: Original Stack vs. Final Stack
- Tools: MermaidChart.com

3. Figure 3.3: Component Architecture Evolution

- Type: Block Diagram with Color Coding
- Components: All System Modules
- Tools: MermaidChart.com

4. Figure 3.4: Data Flow Sequence Diagram

- Type: UML Sequence Diagram
- Actors: User, Frontend, Backend, External Services
- Tools: Eraser.io

5. Figure 4.1: System Architecture Diagram

- Type: 3-tier Architecture Diagram
- Components: Presentation, Application, External Services
- Tools: MermaidChart.com

6. Figure 4.2: Component Interaction Sequence Diagram

- Type: UML Sequence Diagram
- Actors: All System Components
- Tools: Eraser.io

7. Figure 4.3: Frontend Component Hierarchy

- Type: Tree Diagram
- Components: React Components
- Tools: MermaidChart.com

8. Figure 4.4: Backend Component Architecture

- Type: Module Diagram
- Components: Backend Modules
- Tools: MermaidChart.com

9. Figure 4.5: AI Service Architecture

- Type: Pipeline Diagram
- Components: RAG Pipeline Stages
- Tools: MermaidChart.com

10. Figure 4.6: RAG Pipeline Flowchart

- Type: Flowchart
- Flow: Input → Processing → Output
- Tools: MermaidChart.com

11. Figure 4.7: AI Service Deployment Architecture

- Type: Deployment Diagram
- Components: Colab, Drive, Cloudflare, Backend
- Tools: Eraser.io

12. Figure 4.8: Deployment Architecture Diagram

- Type: Network Diagram
- Components: Local Machine, Colab, Connections
- Tools: MermaidChart.com

13. Figure 5.1: Test Execution Flow Diagram

- Type: Flowchart Diagram
- Components: Environment Setup, Test Flows, Result States
- Tools: MermaidChart.com

14. Figure 5.2: Test Coverage Matrix Diagram

- Type: Matrix Diagram
- Components: Test Cases, System Modules
- Tools: MermaidChart.com

Appendix G: Glossary

Term	Definition
CVE	Common Vulnerabilities and Exposures - standardized identifier for security vulnerabilities
CVSS	Common Vulnerability Scoring System - severity scoring standard
Docker	Containerization platform for packaging applications
LLM	Large Language Model - AI model trained on text data
NVD	National Vulnerability Database - US government vulnerability database
RAG	Retrieval-Augmented Generation - AI technique combining retrieval and generation
Trivy	Open-source container vulnerability scanner
Vector Database	Database optimized for similarity search on vector embeddings

Appendix H: Team Contributions

Student ID	Student Name	Roles	Responsibilities
6345438	Abbas Bhaiji	Team Leader AI/ML Engineer	Project Management LLM Testing RAG + LLM Testing AI Integration & Testing GUI Refinement & Testing Final Testings
7390440	Ansaf Sabu	Scribe AI/ML Engineer	Data Creation Scanner Development LLM Testing RAG + LLM Testing AI Integration & Testing GUI Refinement & Testing Final Testings Reports
7842430	Mohammed Uddin	Team Member Business Sys	Test plan Research Poster & Report Refinement GUI Integration
7797965	Mohamed Rizvan	Team Member CyberSec	Docker Upload Module Implementation Docker Integration, Scanning Images Creation, Scanning Wrappers. Overall Scanning Module.
6728492	Derick Reni	Team Member CyberSec	Video Demo Production, Report Refinement & Test Plan Research GUI Integration

Appendix I: Mentors

Name	Roles	Responsibilities
Dr. Manoj Kumar	CyberSec Advisor	Guided project initiation, defined scope, and provided direction throughout development
Asma Damankesh	AI Advisor	Provided expertise on LLM integration, advised on RAG + LLM implementation, and supported AI architecture decisions

Appendix J: Project Timeline

Phase 1: Planning (Semester 1)

- Weeks 2–4: Project proposal development
- Weeks 5–6: Feasibility study & requirements analysis
- Weeks 7–9: High-level system design

Phase 2: Implementation (Semester 2)

- Weeks 1–2: Scope refinement
- Weeks 3–4: Frontend core development
- Weeks 5–6: Backend development
- Weeks 7–9: AI service development & integration
- Weeks 9–10: System testing & final refinements

END OF REPORT