AI - DRIVEN PENTESTER

- BY DEEPTHINK

TEST REPORT

ASSIGNMENT COVER SHEET

UNIVERSITY OF WOLLONGONG IN DUBAI

Assignment Cover Sheet

Subject Code:	CSIT321
Subject Name:	Project
Submission Type:	Test Plan Report
Assignment Title:	Capstone Project
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:	12/05/2025
Date Submitted:	12/05/2025

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

Objective

The primary objective of this test plan is to verify and validate the functionality of the ArmourEye platform. The goal is to ensure the system can reliably automate the penetration testing lifecycle against containerized web applications.

Testing will focus on:

- Verifying that all core features, from Docker image upload to final report generation, function as designed.
- Ensuring the AI Orchestrator makes logical decisions for web-specific vulnerabilities.
- Verifying that user data privacy is maintained by ensuring sensitive information within uploaded images is not exposed or mishandled during the scanning process.
- Identifying and documenting as many defects as possible to deliver a high-quality, stable application for the final project submission.

Scope

In Scope:

The following modules and features will be tested rigorously:

- **GUI & Core Workflow:** The user's ability to upload a Docker image via the GUI, which is connected to Docker Desktop.
- **Image Scanning:** The integration with a free, third-party tool to perform static analysis on the uploaded Docker image.
- **Vulnerability Analysis:** The system's ability to process the scanner's output (image fingerprint/details) to retrieve information on associated CVEs, exploit details, and potential fixes.
- **Reporting:** The generation of an accurate, comprehensive report detailing all findings.
- **Data Persistence:** Successful storage of all session data and results in a PostgreSQL database.

Out Of Scope:

The following items will not be tested in this phase:

- Live Exploitation: The planned module to connect with Metasploitable for live exploitation is currently out of scope. Its feasibility will be evaluated after the core image-scanning workflow is complete.
- Advanced techniques for bypassing security controls (e.g., WAFs).
- Multi-user collaboration or role-based access control (RBAC).

Testing Methodology

A multi-layered testing methodology will be adopted:

- **Functional Testing:** To ensure the software works as it should.
- **Unit Testing:** Developers will create tests for individual backend functions to verify they work in isolation.
- **Integration Testing:** This will focus on ensuring the different components communicate correctly, especially the GUI ↔ Flask API, the API ↔ Docker Desktop, and the image scanner ↔ AI/LLM processor.
- **System Testing:** End-to-end tests will be performed to simulate the complete user journey.
- **Usability Testing:** Manual testing will be conducted to assess the user-friendliness and overall experience of the platform.
- **Compatibility Testing:** To ensure the web based GUI functions correctly on major browsers
- **Report Validation:** Testing will explicitly verify that the fixes for identified CVEs are correctly retrieved and displayed in the final report.

Approach

Our approach will be scenario-driven, focusing on validating the complete image analysis workflow.

- **High-Level Scenarios:** We will execute tests based on realistic user stories. A primary scenario is: "As a user, I upload a vulnerable Docker image (e.g., an old version of a web server), and expect ArmourEye to scan it and report the known CVEs and their corresponding patches."

- **Application Flow:** The testing will follow the system's logical flow:

    - The user uploads a target application's Docker image.
    - The system runs the image through an integrated scanning tool.
    - The scanner extracts image details (fingerprint).
    - The LLM processes this information to find CVEs, exploit details, and fixes.
    - The Reporting Module compiles and displays all findings, including the fixes.

Test Case Design

This section outlines the high-level categories of test cases that will be designed to validate the ArmourEye platform.

The test cases will be grouped into the following categories:

- GUI & Usability Test Cases:
    - Verify that all buttons, links, and navigation elements in the GUI (Dashboard, Setup, Reports) function correctly.
    - Ensure the Docker image upload provides clear user feedback (e.g., a progress bar and a "success" message).
    - Validate that the final vulnerability report is displayed in a clear, readable, and well-organized format.
- Functional (End-to-End) Test Cases:
    - Focus on the "happy path" or the main user workflow.
    - Example: A test case for successfully uploading the owasp/dvwa image, running a scan, and verifying that the final report correctly lists known CVEs and their associated fixes.
- Error Handling & Negative Test Cases:
    - Focus on the "sad path" to ensure the system is robust. Example:
    - What happens if a user tries to upload an invalid file (e.g., a .txt file instead of a Docker image)?
    - What happens if the third-party scanner fails or returns an error?
    - What happens if the LLM cannot find a fix for a specific CVE?
- Report & Data Validation Test Cases:
    - These test cases will focus on the accuracy of the results.
    - Verify that the CVEs listed in the report are correct for the target Docker image.
    - Verify that the "Fix" information provided by the LLM is accurate and relevant to the identified CVE.
    - Verify that all report data is successfully and correctly saved to the PostgreSQL database.

High-Level Test Scenarios

This section outlines the primary scenarios for end-to-end system testing. These scenarios will be the basis for developing detailed, step-by-step test cases.

Scenario ID	Scenario Title	Description	Priority
TS-01	**Successful End-to-End Scan (Vulnerable Image)**	A user uploads a known-vulnerable Docker image (e.g., owasp/dvwa). The system successfully scans it, the LLM identifies the correct CVEs and fixes, and the report is accurately generated and saved to PostgreSQL.	**Critical**
TS-02	**Successful Scan (Clean Image)**	A user uploads a clean, up-to-date Docker image (e.g., the latest official nginx image). The system scans it and generates a "clean" report, correctly stating that no known CVEs were found.	**High**
TS-03	**Invalid File Upload (Error Handling)**	A user attempts to upload a file that is not a Docker image (e.g., a .txt or .jpg file). The system must reject the file gracefully and show a clear error message to the user.	**Medium**
TS-04	**Report and Fix Accuracy Validation**	A tester runs a scan on a known-vulnerable image and manually cross-references the CVEs and fixes in the report with a public database (like NVD) to ensure the LLM's output is accurate and relevant.	**High**

TS-05	**Data Persistence Verification**	After a scan is complete, the tester queries the PostgreSQL database directly to verify that the session data (e.g., image name, timestamp) and all report findings have been correctly saved.	**High**
TS-06	**Scanner Integration Failure (Error Handling)**	Simulate a scenario where the integrated image scanner fails (e.g., it times out or returns an error code). The system must not crash and should display a relevant error message to the user.	**Medium**

Assumptions

This test plan is based on the following assumptions:

- The team has access to a stable test environment with Docker Desktop and all required software dependencies installed.
- A suitable and reliable free tool for Docker image scanning can be successfully integrated into the platform.
- All team members are available and will collaborate effectively to execute tests and resolve defects.

Risks

The following potential project risks have been identified:

- **Integration Risk:** The various modules (GUI, API, Docker Desktop, third-party scanner) may not integrate smoothly, leading to complex, hard-to-diagnose bugs.
- **Dependency Risk:** The project relies on a free, third-party image scanner. If this tool has limitations, produces inconsistent output, or changes its API, it could impact the project's functionality and timeline.
- **Data Processing Risk:** The LLM may have difficulty accurately parsing the scanner's output to extract CVEs and map them to the correct fixes, requiring significant prompt engineering.
- **Data Privacy Risk:** User-uploaded Docker images could contain sensitive data (e.g., API keys, source code, credentials). The generated vulnerability reports are also highly confidential, and there is a risk of this information being mishandled or exposed.

- **Accuracy Risk:** The possibility that the integrated scanner or the LLM provides incorrect results, such as false positives (flagging non-existent vulnerabilities) or false negatives (missing actual CVEs).
- **Scope Creep Risk:** The temptation to add unplanned features, like the live exploitation module, before the core functionality is stable, which could jeopardize the delivery of the main project goals.
- **Environment Consistency Risk:** The chance that the application behaves differently on various team members' machines due to subtle differences in Docker Desktop, OS, or network configurations, making bugs hard to reproduce.
- **Performance Risk:** The risk that the image scanning and LLM analysis process is too slow, leading to a poor user experience and making the tool impractical for larger images.

Mitigation Plan

To address the identified risks, the following strategies will be employed:

- **Prioritization (MVP Focus):** We will focus on ensuring one complete end-to-end image scan scenario works perfectly before adding more complex features or targets.
- **Early Integration:** Integration of the third-party scanner will be prioritized to identify and resolve potential issues early in the development cycle.
- **Modular Design:** The LLM processing logic will be kept in a separate module, allowing it to be refined or even replaced without affecting the entire application.

Roles and Responsibilities

Role	Name(s)	Responsibilities
Team Leader **AI/LLM Engineer**	Abbas Bhaiji	Overall test management, schedule tracking, and final approval of the Test Summary Report.

AI/LLM Engineer Scribe	Ansaf Sabu	Testing the LLM's ability to process scanner data and the integration with other modules.
UI / CyberSec	Mohammed Uddin & Derick Reni	Jointly responsible for all GUI and Usability testing. Will also lead the execution of end-to-end System Tests.
UI / CyberSec	Mohamed Rizvan	Preparing the test environment, selecting target Docker images, and validating the accuracy of the final vulnerability reports.

Success Metrics

The success of the testing phase will be evaluated based on the following key metrics:

- Test Case Pass Rate: The percentage of executed test cases that have passed. Our goal is to achieve a 100% pass rate for all critical-path scenarios.
- Defect Severity Distribution: Tracking the number of critical, high, medium, and low severity defects. A successful test phase will see all critical and high severity defects resolved before the exit criteria are met.
- Defect Resolution Rate: The percentage of reported defects that have been fixed and verified.

Test Environment

- **Hardware:** A Windows system capable of running Docker Desktop smoothly.
- **Software:** Docker Desktop, Python 3.10+, PostgreSQL, and a modern web browser.
- **Test Targets** (Docker Images to be scanned):
    - OWASP DVWA
    - Metasploitable
    - Optional: OWASP Juice Shop/Other WebApps

Entry and Exit Criteria

Entry Criteria (To begin System Testing):

- The backend implementation for the core MVP workflow is complete and integrated.
- A stable build is deployed via Docker Compose.
- The test environment and target Docker images are ready.

Suspension Criteria:

- Testing will be suspended if a critical showstopper bug blocks the primary workflow(e.g. Docker images upload fails 100% of the time)
- Test environment is unstable or unusable for more than 4 hours

Exit Criteria (To conclude testing):

- All critical and high-priority test scenarios pass.
- There are no known "showstopper" bugs.
- The final report generation feature is fully functional and accurate.
- The team agrees the project is ready for final submission.

Test Automation

- **Manual Testing:** The primary method will be manual, including all GUI/Usability and end-to-end system tests.
- **Automated Testing:** Developers will implement automated unit tests for backend Python modules.