**WOPA: The Intelligent Chat Safeguarder**

**CS 588 Cybersecurity Capstone**

**Test Plan**

Date: 12 December 2024

**Prepared by**: Shucheng Fang, Yongcheng Liu

**Table of Contents**

Contents

[**1. Introduction 3**](#_heading=h.30j0zll)

[**1.1. Objectives 3**](#_heading=)

[**1.2. Team Members 3**](#_heading=)

[**2. Scope 3**](#_heading=h.yszzxlt51mc4)

[**3. Assumptions / Risks 4**](#_heading=h.hwxz6waqv365)

[**3.1. Assumptions 4**](#_heading=h.3dy6vkm)

[**3.2. Risks 5**](#_heading=h.iqpnevmi4psl)

[**4. Requirement Verification Matrix (RVM) 6**](#_heading=h.5gd34wdufk9n)

[**5. Test Cases 7**](#_heading=h.o5qy73qkbgm8)

[**5.1. Test Environment 7**](#_heading=)

[**5.2. Test Approach 8**](#_heading=)

[**6. Test Procedures 12**](#_heading=h.xthlve3jfvog)

# Introduction

The Test Plan has been created to communicate the test approach to team members and stakeholders. It includes the objectives, scope, schedule, risks, and approach. This document will identify the test deliverables and what is deemed in and out of scope. The goal is to ensure that **WOPA** is adequately validated for its core functionalities, specifically in **real-time threat detection, static content analysis**, and **user-behavior-simulation-based dynamic analysis** on mobile platforms.

## Objectives

The main objective of the test plan is to validate that **WOPA** meets its functional and non-functional requirements, ensuring accurate, real-time threat detection in mobile communication platforms. Specifically, the testing will aim to:

* Verify the usability and isolation capabilities of the secure sandbox environment, ensuring it can safely open suspicious links and files while isolating potential threats from the user's device.
* Test the accuracy and performance of the text-based analysis module, which uses Large Language Models (LLMs) for analyzing logs and assessing the legitimacy of links/files.
* Evaluate the visual-based behavior simulation module, ensuring it can autonomously simulate user interactions with mobile apps to detect hidden threats or privacy violations.
* Ensure the system can generate a detailed report based on static and dynamic analysis for threat detection.

## Team Members

| **Name** | **Role** |
| --- | --- |
| Shucheng Fang | Developer / Documentation / Tester |
| Yongcheng Liu | DevOps / Project Manager / Documentation / Tester |

# Scope

This document identifies the testing approach for verifying all functional and non-functional requirements outlined in the **Requirements Analysis Document (RAD)** originally delivered on **09 September 2024**. Testing will support all use cases defined within the RAD, and evidence of test execution will be provided for each requirement and use case.

Specifically, the testing will focus on these scenarios:

* 1. **Act as a user with no security expertise**: Test that the user can use an **intuitive interface** to analyze suspicious message content, such as links or files, and receive a detailed report on the potential threats or risks involved.
  2. **Act as developers utilizing the AI-powered text-based analysis modules. Test the system to** detect threats in the provided content (e.g., links or files) and conduct text-based security assessments based on the system activities or network traffic analysis.
  3. **Act as developers using the visual-based behavior simulation module**: Test that the system can simulate user interactions within mobile apps to detect hidden threats, such as privacy violations or unauthorized access to sensitive resources.

Section 4 below identifies the verification methodology (test, demonstration, analysis or inspection) which will be used for each functional and nonfunctional requirement.

Section 5 below describes the testing plan which will be used to verify operation of each Use Case.

# Assumptions / Risks

## **Assumptions**

This section lists assumptions that are made specific to testing performed for this project.

**Server Availability**: The provided server is available and fully operational throughout the testing period.

**Know Threats Database**: The server has a database of known phishing and benign URLs for testing purposes.

**Internet Access**: The app has reliable internet access to fetch real-time data for live URL analysis.

**URL Format**: The URLs provided for analysis are properly formatted and adhere to standard URL syntax (e.g., http:// or https://).

**Phishing Characteristics**: The URL features for phishing detection (e.g., domain age, SSL certificate, suspicious patterns) are accessible and can be analyzed by the app.

**Security Policies**: The app is allowed to access external APIs for domain information and reputation checks during the testing phase.

**User Inputs**: Users provide URLs that are less than 2083 characters in length (the maximum length for most browsers).

**Time Constraints**: The average response time for URL analysis, including network latency and API calls, will not exceed X seconds.

**Browser Support**: The app will be tested only on modern, up-to-date browsers (e.g., Chrome, Firefox, Safari).

**User Volume**: The test assumes that a limited number of users (e.g., 5) will use the app simultaneously during peak load scenarios.

**Output Format**: The analysis results will be provided in a simple format (e.g., 'Phishing,' 'Benign', or 'Suspicious') with optional user recommendations for now, which will be improved in later iterations.

**Text-Based Threat Characteristics**: The suspicious activities can be captured through analyzing the basic execution logs, system calls, API interactions, or network traffic generated during testing.

**Visual-Based Behavior Simulation**: The tested apps are easy for the agent to use to simulate user interactions (e.g., clicking links, navigating interfaces), and the behaviors during these simulations represent real-world usage scenarios.

## 

## **Risks**

The following risks have been identified and the appropriate action identified to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is what milestone or event would cause the risk to become an issue to be dealt with.

| **#** | **Risk** | **Impact** | **Trigger** | **Mitigation Plan** |
| --- | --- | --- | --- | --- |
| 1 | Inaccurate Phishing Detection | High | False positives or false negatives during testing, leading to incorrect URL classification. | Implement comprehensive testing with diverse datasets, continuously update the phishing database, and adjust detection algorithms based on feedback and edge cases. |
| 2 | Third-Party API Downtime | High | Failure of external APIs (e.g., for domain lookup or reputation services) during URL analysis. | Include fallback mechanisms, cache previous results, and display a temporary error message to users when APIs are unavailable. |
| 3 | Latency in Real-Time URL Analysis | Moderate | Slow response time from external services or network latency, causing delays in URL analysis. | Optimize network calls, reduce dependency on external services, and provide asynchronous processing with user notification. |
| 4 | Insufficient Test Data for Analysis | Low | Lack of sufficient phishing and benign URL data for accurate test results. | Expand datasets by sourcing URLs from multiple reliable sources, including public phishing databases and historical data. |
| 5 | User Input Errors | Low | Users submitting improperly formatted URLs or irrelevant data, leading to errors or inaccurate analysis. | Implement input validation, provide clear instructions for users, and return user-friendly error messages. |
| 6 | Security Vulnerabilities | High | Discovery of vulnerabilities in the app that could lead to exploitation or unauthorized access during testing. | Conduct regular security audits, implement secure coding practices, and test for vulnerabilities like SQL injection and XSS. |
| 7 | Inconsistent Test Environment | Moderate | Variations in network conditions or server performance, leading to inconsistent testing results. | Standardize the testing environment, ensure consistent network conditions, and use controlled test setups. |
| 8 | Legal and Privacy Compliance Issues | Moderate | Collection or analysis of user data without following relevant data protection regulations (e.g., GDPR). | Ensure compliance with local and international privacy laws, anonymize user data during testing, and implement clear data handling policies. |
| 9 | Integration Issues with Other Systems | Moderate | Failures during the integration of the phishing link analyzer with other internal systems or platforms. | Perform incremental testing of integrations, involve stakeholders early in the integration process, and provide contingency plans for integration failures. |
| 10 | Unclear Results Interpretation by Users | Low | Users not understanding the analysis results, leading to confusion or incorrect decision-making. | Provide detailed but user-friendly explanations of the results and recommendations, with visual aids (e.g., color-coded risk levels). |

# Requirement Verification Matrix (RVM)

(T) Test where the system is stimulated with defined input and output is captured, measured and/or evaluated by some instrumentation and the results are then compared to the expected results.

(D) Demonstration where the system is stimulated with defined input and output is observed using one or more of the senses without the need of instrumentation.

(A) Analysis is where a mathematical model of the system (or simulation) is used to evaluate how the system will perform.

(I) Inspection is where visual examination of some part of the system is used to verify compliance with requirement.

| **Requirement** | **Verification** | | | | **Phase** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **D** | **A** | **I** | **U** | **I** | **S** |
| The system must correctly classify URLs as phishing, benign, or suspicious based on predefined criteria. | X |  |  |  | X | X | X |
| The system must provide feedback to users with a clear risk level (e.g., phishing, benign) and an explanation of the result. |  | X |  | X |  | X | X |
| The system must perform URL analysis within an average response time of 60 seconds. |  |  | X |  | X | X | X |
| The system must use valid external APIs to conduct domain information and URL reputation checks. | X | X |  |  | X | X | X |
| The system must have a fallback mechanism for when third-party APIs are unavailable. | X | X |  |  | X | X | X |
| The system must validate user inputs to ensure proper formatting and prevent injection attacks. | X |  |  | X | X | X | X |
| The system must log and report any analysis failures or unexpected behavior for debugging purposes. | X |  |  | X | X | X | X |
| The system must provide error messages for invalid content format or unresponsive APIs. | X | X |  |  | X | X | X |
| The system must comply with GDPR and data privacy regulations regarding user data. |  |  |  | X | X | X | X |
| The system must scale to handle simultaneous requests from up to 5 users. |  |  | X |  | X | X | X |
| The system must analyze text-based logs (e.g., system calls, API interactions, network traffic) to detect threats. | X |  |  | X |  | X | X |
| The system must simulate user interactions in apps for visual-based threat detection (e.g., privacy violations). | X |  |  | X | X | X | X |

# Test Cases

This section provides information about each test that will be done to verify that the software fulfills stakeholder needs.

## Test Environment

The following elements are required to execute testing for the WOPA system, ensuring that the environments, tools, and systems provide appropriate conditions for testing the application across different platforms and scenarios.

**Android Environments**

* Android on Raspberry Pi: To simulate an ARM-based architecture, providing a real-world testing environment for mobile security on ARM devices.
* BLISS OS in VirtualBox: To simulate Android x86 architecture, providing a virtual environment for running the WOPA system on PC hardware.
* Docker-Android: Docker containerized Android environment to simulate scalable, cloud-based Android testing. This will be used for automated testing of multiple instances of the WOPA system.

**Networking**

* Internal Network Setup: A stable and secure internal network will be required for testing network traffic generated by the WOPA system during analysis.
* External Provider Connectors:
* Connections to third-party APIs for URL reputation checks, domain information, and other external data providers.
* Secure communication channels for retrieving and sending data to these external services.

**Databases**

* Phishing URL Database: A regularly updated database of phishing URLs for testing the detection and classification capabilities of the WOPA system.
* Benign URL Dataset: A curated dataset of non-malicious URLs to validate the system’s ability to classify benign traffic accurately.

**Tools**

* Wireshark: A good tool for capturing and analyzing network traffic generated during testing, especially for monitoring communication between the WOPA system and external providers.
* ELK Stack (Elasticsearch, Logstash, Kibana): A powerful tool for collecting, analyzing, and visualizing logs generated by the WOPA system during testing.
* Strace: A diagnostic tool to monitor and trace application system calls. This tool is essential for tracking suspicious activity during text-based analysis.
* MobSF (Mobile Security Framework): A comprehensive static analysis tool for analyzing potentially harmful APK files. It helps detect vulnerabilities, misconfigurations, and malicious code in Android applications.
* AppAgent for Visual Simulation: A testing tool that uses LLM-V to simulate user interactions with mobile apps, helping to test the WOPA system's visual-based behavior simulation.
* Locust: A load testing tool to simulate multiple users interacting with the WOPA system simultaneously, ensuring scalability and stability under high traffic conditions.

## Test Approach

**Test1: URL Classification Accuracy Test**

* Verification Methodology: Test (T)
* Testing Type: Equivalence Testing
* Description:
  + Setup: Prepare a dataset with URLs that are classified as phishing, benign, or suspicious based on predefined criteria. Ensure that the dataset includes diverse and edge cases for accurate testing.
  + Execution: Feed the URLs into the system, stimulating it with both phishing and benign URLs. Capture the system’s classifications and compare them to the expected output.
  + Observation: Evaluate the accuracy by measuring the number of correctly classified URLs and identify any false positives or negatives.

**Test2: User Feedback Clarity Test**

* Verification Methodology: Demonstration (D)
* Testing Type: Path Testing
* Description:
  + Setup: Set up the system with various URLs that will trigger phishing, benign, and suspicious classifications.
  + Execution: Observe the user interface to ensure that the feedback displays the risk level (e.g., phishing, benign) and provides a clear explanation of the results.
  + Observation: Ensure that all possible outcomes display correctly and can be easily understood by users without requiring technical expertise.

**Test3: URL Analysis Response Time Test**

* Verification Methodology: Analysis (A)
* Testing Type: Boundary Testing
* Description:
  + Setup: Measure the system’s response time under various conditions (e.g., different URL types, load conditions) and establish acceptable boundary limits for performance.
  + Execution: Test the system’s response time for URL analysis across a range of URLs and usage conditions, simulating scenarios like high API load.
  + Observation: Capture the response times and ensure that they fall within the predefined acceptable limits (60 seconds). Identify and log any instances where response times exceed the limit.

**Test4: External API Integration Test**

* Verification Methodology: Test (T), Demonstration (D)
* Testing Type: Integration Testing
* Description:
  + Setup: Ensure the system has access to the necessary third-party APIs for domain information and reputation checks.
  + Execution: Run tests to validate that the system correctly sends requests to the APIs, receives responses, and handles any delays or failures in communication.
  + Observation: Verify that the APIs are correctly integrated and provide the expected information. Check for graceful degradation if APIs are unavailable, including the system’s ability to fall back on cached data or display error messages.

**Test5: Fallback Mechanism Test for Unavailable APIs**

* Verification Methodology: Test (T), Demonstration (D)
* Testing Type: Path Testing
* Description:
  + Setup: Simulate a scenario where third-party APIs are unavailable (e.g., by disabling network access).
  + Execution: Run URL analysis under the conditions of unavailable APIs, ensuring that the fallback mechanism is triggered.
  + Observation: Verify that the system logs the API failure, provides fallback results (e.g., cached data), and presents a clear message to the user about the degraded functionality.

**Test6: User Input Validation Test**

* Verification Methodology: Test (T), Inspection (I)
* Testing Type: Boundary Testing
* Description:
  + Setup: Prepare a series of possible user inputs, including properly formatted URLs or files and edge cases (e.g., malformed URLs, potentially harmful inputs).
  + Execution: Submit various inputs and validate that the system properly formats and sanitizes each input to prevent injection attacks.
  + Observation: Capture and review the system’s behavior, ensuring that only valid user inputs are processed and improper inputs are flagged with appropriate error messages.

**Test7: Logging and Reporting Test**

* Verification Methodology: Test (T), Inspection (I)
* Testing Type: Path Testing
* Description:
  + Setup: Enable system logging and submit several invalid content inputs for analysis, some of which are expected to cause failures or errors.
  + Execution: Run the test cases and induce errors to ensure that failures are captured and logged.
  + Observation: Inspect the logs to verify that any failures or unexpected behavior are recorded accurately, and that logs provide useful debugging information.

**Test8: Error Messaging for Invalid User Input and API Failures**

* Verification Methodology: Test (T), Demonstration (D)
* Testing Type: Equivalence Testing
* Description:
  + Setup: Prepare a set of invalid URLs (e.g., incorrectly formatted or unsupported protocols) and simulate API failures (e.g., disable access to third-party services).
  + Execution: Submit the invalid URLs and trigger API failures, capturing the system’s responses.
  + Observation: Ensure that appropriate error messages are displayed clearly to users, indicating whether the issue was related to user input or API unavailability. Validate that these messages are actionable and easy to understand for non-technical users.

**Test9: GDPR Compliance Test**

* Verification Methodology: Inspection (I)
* Testing Type: Peer Inspection
* Description:
  + Setup: Review the system’s data handling and storage processes, focusing on the use of personal data and privacy policies.
  + Execution: Manually inspect the system to ensure that all data collection, storage, and usage comply with GDPR and other relevant data privacy regulations.
  + Observation: Verify that user data is anonymized or deleted as per GDPR requirements and that users are informed about data usage.

**Test10: Scalability Test (Simultaneous Users)**

* Verification Methodology: Analysis (A)
* Testing Type: Load Testing
* Description:
  + Setup: Simulate concurrent users submitting URLs for analysis.
  + Execution: Measure the system’s ability to handle the concurrent requests while maintaining acceptable response times and system stability.
  + Observation: Analyze the system's performance under the load and verify it scales appropriately without significant degradation.

**Test11: Text-Based Threat Detection Test**

* Verification Methodology: Test (T), Analysis (A), Inspection (I)
* Testing Type: Integration Testing
* Description:
  + Setup: Prepare system logs, API interactions, and network traffic representing suspicious behaviors, such as malicious system calls or unauthorized API access.
  + Execution: Feed these logs and traffic data into the system for analysis.
  + Observation: Verify that the text-based analysis module correctly detects and flags threats based on the logs. Ensure that the system's detection is accurate and efficient, with minimal false positives.

**Test12: Visual-Based Threat Detection Test**

* Verification Methodology: Test (T), Demonstration (D), Inspection (I)
* Testing Type: Path Testing
* Description:
  + Setup: Use mobile apps that simulate user interactions, including clicking links or accessing sensitive data (e.g., contacts, camera).
  + Execution: Run visual-based simulations that imitate real user behaviors, allowing the system to monitor and detect privacy violations or unauthorized app behaviors.
  + Observation: Ensure that the system successfully detects privacy violations or dynamic threats during simulated user interactions. Verify that the results are accurately presented in the analysis.

# Test Procedures

This section provides detailed step-by-step instructions for how each test case will be conducted, including preconditions, dependencies, expected outcomes, and any post-test actions. Each test procedure is designed to ensure that the **WOPA** system is thoroughly validated for its functionality, performance, and compliance with the defined requirements.

Test procedures include both functional and non-functional testing, covering areas such as **error handling**, **threat detection**, **system scalability**, and **compliance** with regulations like **GDPR**. The procedures also ensure that edge cases, such as system failures and invalid user input, are tested and that the system behaves appropriately in all conditions.

The following test cases have been outlined in detail:

| Test Case: URL Classification Accuracy Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Verifies that the system correctly classifies URLs as phishing, benign, or suspicious. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The system is operational with a trained URL classification model. | | | | | | | |
| Dependencies: Dataset of phishing, benign, and suspicious URLs is available. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit phishing URLs | phishing URLs | URL is classified as phishing | | IsPhishing == true | pass | A detailed threat report is generated with classification and risk level. |
| 2 | Submit normal URLs | normal URLs | URL is classified as not phishing | | IsPhishing == false | pass | Should report as high confidence safe link. |
|  | | | | | | | |
| Postconditions: The system correctly classifies all URLs as phishing, benign, or suspicious, and logs the results. | | | | | | | |

| Test Case: User Feedback Clarity Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Confirms that feedback provided to users on URL risk levels is clear and easily understood. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The user interface is fully developed and accessible. | | | | | | | |
| Dependencies: URL classification logic is functional. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit a phishing URL | phishing URL | A clear warning message indicating the URL is dangerous | | IsPhishing == true | pass | Ensure detection matches the intended behavior |
|  | | | | | | | |
| Postconditions: Clear and understandable feedback is displayed for each URL classification. | | | | | | | |

| Test Case: URL Analysis Response Time Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Measures the system’s response time to ensure it meets acceptable performance limits. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The system is operational with a functioning URL analysis engine. | | | | | | | |
| Dependencies: No other system performance issues affecting response time. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit multiple URLs (phishing, non-phishing) | Varying URLs | Response time should not exceed the acceptable limit (e.g., 2 seconds) | | IsPhishing == true | pass | Validate the system against edge cases, such as URLs with ambiguous patterns. |
|  | | | | | | | |
| Postconditions: The system processes URLs within the acceptable response time, with performance issues logged. | | | | | | | |

| Test Case: External API Integration Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Validates that third-party APIs are correctly integrated and handle communication issues effectively. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: Third-party APIs are accessible, and the system is configured to use them. | | | | | | | |
| Dependencies: API keys and permissions are set up. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit a URL requiring external API lookup | URL needing domain info from API | System fetches domain information and correctly classifies the URL | | Classify correctly. | pass | Test network reliability and ensure the system handles API timeouts gracefully. |
|  | | | | | | | |
| Postconditions: The system successfully retrieves and uses data from external APIs for URL classification. | | | | | | | |

| Test Case: Fallback Mechanism Test for Unavailable APIs | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Ensures that fallback mechanisms function properly when external APIs are unavailable. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The system is integrated with external APIs, and fallback mechanisms are in place. | | | | | | | |
| Dependencies: Simulated API failure. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Disable external API access | API failure scenario | The system uses fallback logic and displays a message about reduced functionality | | URLs requiring API lookup fail gracefully with appropriate error messages. | pass | Test fallback mechanisms for when API access is unavailable. |
|  | | | | | | | |
| Postconditions: The system employs fallback mechanisms and notifies users when external APIs are unavailable. | | | | | | | |

| Test Case: User Input Validation Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Checks that URL inputs are validated and sanitized to prevent injection attacks. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The input validation module is functional. | | | | | | | |
| Dependencies: Input sanitization and validation rules are implemented. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Input invalid/malformed URL | Malformed URL | System rejects the URL and displays a validation error message | | The system rejects the input with a clear error message. | pass | Validate error handling for various malformed URL formats. |
|  | | | | | | | |
| Postconditions: The system rejects invalid URLs and provides appropriate validation error messages. | | | | | | | |

| Test Case: Logging and Reporting Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Verifying that the system correctly logs errors and failures for debugging purposes. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: a) Logging is enabled on the system. b) The system is ready to analyze submitted content. | | | | | | | |
| Dependencies: a) The logging system is configured and operational. b) One minimal analyzer is available (phishing). | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit invalid inputs for analysis. | Invalid URLs and data inputs. | Errors are logged accurately. | | Invalid inputs are rejected. | pass | Test with edge cases (e.g., null, empty strings, or binary data). |
| 2 | Induce system errors or failures. | Faulty input that causes errors. | All failures are captured in the logs. | | The system logs the error details and continues operating for unaffected features. | pass | Verify that the system isolates failures without cascading effects. |
| 3 | Inspect logs for failures. | System logs. | Logs contain sufficient information for debugging. | | Logs capture detailed information about errors, including timestamps, causes, and stack traces. | pass | Ensure logs are stored securely and do not expose sensitive user information. |
|  | | | | | | | |
| Postconditions: Logs should be examined for accuracy and completeness. | | | | | | | |

| Test Case: Error Messaging for Invalid User Input and API Failures | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Verifying that the system displays appropriate error messages for invalid inputs and API failures. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: a) The system is connected to third-party APIs. b) The system is able to process URL inputs. | | | | | | | |
| Dependencies: a) External APIs are online or simulating failures. | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Submit invalid URLs. | Malformed or incorrect URLs. | Error messages clearly explain invalid input. | | Invalid URLs are rejected with appropriate error messages. | pass | Confirm that no invalid URLs bypass validation. |
| 2 | Disable API access to simulate failure. | API Path | Error messages explain API failure. | | The system operates with limited functionality and displays fallback behavior. | pass | Validate the robustness of fallback processes and user notifications. |
| 3 | Observe error messages. | Error output. | Get error messages at the front end. | | All error messages are user-friendly and provide actionable feedback. | pass | Ensure error messages avoid exposing internal implementation details. |
|  | | | | | | | |
| Postconditions: The user is notified of input issues or API problems in a user-friendly manner. | | | | | | | |

| Test Case: GDPR Compliance Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Verifying that the system complies with GDPR regulations regarding data collection, storage, and deletion. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: N/A | | | | | | | |
| Dependencies: N/A | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Review data collection & storage processes | user input | Data collection and storage are minimal and lawful | | Data is stored securely and complies with privacy policies. | pass | Cross-check with GDPR or relevant data protection standards. |
|  | | | | | | | |
| Postconditions: All data processing and storage operations comply with GDPR requirements. | | | | | | | |

| Test Case: Scalability Test (Simultaneous Users) | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Testing the system's ability to handle multiple simultaneous user requests. | | | | Executed By: | | | |
| SW Baseline: 0.0.1 | | | |
| Preconditions: The system is operational and accessible.. | | | | | | | |
| Dependencies: Load Testing Tool | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Simulate multiple user requests. | Concurrent URL submissions. | The system handles concurrent requests smoothly. | | The system handles concurrent requests effectively without degradation in response time. | pass | Monitor for bottlenecks under heavy loads and optimize accordingly. |
| 2 | Measure response times under load. | Load testing metrics. | Response times are within acceptable limits. | | Response times remain within acceptable limits under varying loads. | pass | Test peak loads and ensure scalability. |
|  | | | | | | | |
| Postconditions: System performance under load is within the expected range. | | | | | | | |

| Test Case: Text-Based Threat Detection Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Verifying the text-based threat detection functionality. | | | | Executed By: | | | |
| SW Baseline: 0.0.2 | | | |
| Preconditions: Text-based analysis modules are available | | | | | | | |
| Dependencies: N/A | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Upload suspicious content | Suspicious links + Files. | The system gets the suspicious contents and conducts analysis. | | Content is analyzed, and any threats are flagged accurately. | pass | Ensure false positives and negatives are within acceptable thresholds. |
| 2 | Analysis | Threat data vs. system output. | The system conducts text-based analysis towards selected metrics (syscall/network). | | Threats are identified, categorized, and explained in the report. | pass | Check for performance on complex text patterns or novel threats. |
| 3 | Reporting | Report | The system generates a report with minimal false positives and negatives | | A detailed, structured report is generated, including threat categories and suggested actions. | pass | Verify report clarity and completeness. |
|  | | | | | | | |
| Postconditions: All identified threats based on text-based patterns are logged and categorized correctly. | | | | | | | |

| Test Case: Visual-Based Threat Detection Test | | | | Test Organization: WOPA Test Team | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Version: v0 | | | | Execution Date: | | | |
| Description: Testing the visual-based threat detection functionality. | | | | Executed By: | | | |
| SW Baseline: 0.0.3 | | | |
| Preconditions: Visual-based analysis modules are available | | | | | | | |
| Dependencies:N/A | | | | | | | |
|  | | | | | | | |
| Step | Actions | Data | Expected Result | | Actual Results | Pass/Fail | Notes |
| 1 | Upload the app via frontend | App apk | The system receives the App and runs it in the sandbox | | The app is uploaded and processed without errors. | pass | Test with apps of varying sizes and file types. |
| 2 | The system simulates user behaviors based on instruction and tries to find threats. | App interaction | The system successfully interacts with the App and can locate issues | | The simulation accurately mimics user interactions and identifies potential threats. | pass | Confirm that behavior simulations cover edge cases. |
| 3 | Reporting | Report | The system generates a report with minimal false positives and negatives | | A comprehensive report of detected threats, user behaviors, and system performance is generated. | pass | Ensure the report format is consistent and actionable. |
|  | | | | | | | |
| Postconditions: All threats identified through visual patterns (e.g., images, screenshots) are accurately logged. | | | | | | | |