HUMAN INFILTRATION DETECTION SYSTEM

- ASMIT TYAGI

-ARCHIT KAUSHIK

-ARUN KUMAR

-ASHISH SIKARWAR

1 Introduction

The Human Infiltration Detection System (HIDS) is an advanced security solution utilizing AI technology to detect and prevent unauthorized human infiltrations in real-time. Designed to address the critical security challenges faced in defense and sensitive areas, HIDS employs sophisticated AI algorithms to monitor, analyze, and respond to potential threats instantly. By enhancing situational awareness, it provides proactive threat detection, reduces response times, and significantly mitigates security risks, ensuring the safety and integrity of high-security zones.

1.1 Scope

1.1.1 In Scope

- Detection and prevention of unauthorized human infiltrations in real-time.
- Integration with existing surveillance infrastructure.
- Processing of surveillance video and image data using deep learning models.

1.1.2 Out of Scope

- Detection of non-human intrusions (e.g., drones, vehicles).
- Direct hardware design or manufacturing.

1.2 Quality Objective

The primary objectives of the testing process are:

- Ensure the system meets functional and non-functional requirements.
- Validate the accuracy of intrusion detection and the efficiency of real-time processing.
- Detect and fix all critical bugs before deployment.

1.3 Roles and Responsibilities

- Test Manager – Dr. Kalpana Sagar
- Developers – Arun Kumar, Ashish Sikarwar
- DL Algorithm and Model Building – Asmit Tyagi, Archit Kaushik
- Installation Team - Asmit Tyagi, Archit Kaushik , Arun Kumar, Ashish Sikarwar

2 Test Methodology

2.1 Overview

The Waterfall methodology provides a structured and sequential approach to the testing and development process, making it particularly suitable for projects with clearly defined and stable requirements. In the context of the Human Infiltration Detection System (HIDS), this methodology ensures that each phase of the development lifecycle is rigorously validated before progressing to the next. Starting with comprehensive requirements gathering, the Waterfall model ensures that all functional, operational, and security needs are clearly documented. This is followed by meticulous design and implementation phases, where the system is developed in alignment with these predefined requirements. Testing is performed systematically, focusing on identifying and resolving any issues in functionality, performance, and reliability. By adhering to a sequential structure, the Waterfall methodology facilitates thorough validation and verification, ensuring that HIDS meets its objectives effectively. This approach is ideal for HIDS projects as it emphasizes reliability, minimizes risks, and ensures the final deployment delivers a robust and secure solution tailored to its critical application areas.

2.2 Test Levels

- **Unit Testing**
    - Validated each module, including pre-processing, CNN, and RNN models, independently.
    - Focused on the accuracy of model predictions for predefined datasets.
    - **Tool Used**: *PyTest* to automate test cases for Python code and *TensorFlow Testing Utilities* to validate model outputs.
- **Integration Testing**
    - Ensured seamless interaction between pre-processing, detection, and alert generation modules.
    - Tested API communications and data handovers between components.
- **System Testing**
    - Conducted end-to-end validation of HIDS in a simulated environment.
    - Tested its ability to detect intrusions with varying scenarios and monitored performance metrics.
- **Acceptance Testing**

- Simulated real-world intrusion scenarios to evaluate detection accuracy and response time.
- Gathered feedback from security personnel and ensured the system met operational requirements.
- **Tool Used**: Manual testing only.

2.3 Suspension Criteria and Resumption Requirements

- **Suspension Criteria**:
  - Critical bugs that block further testing.
  - Environment unavailability or incomplete setup.
- **Resumption Requirements**:
  - Resolution of blocking issues.
  - Validation of fixes in the test environment.

2.4 Test Completeness

- **Functional Coverage**
  - Tested preprocessing, intrusion detection, and alert generation.
  - Validated positive and negative test scenarios.
- **Code Coverage**
  - Used **PyTest** and **Coverage.py** for:
    - **Statement Coverage**: 95%
    - **Branch Coverage**: 90%
- **Path Coverage**
  - Verified normal and exceptional workflows from input to output.
- **Boundary Testing**
  - Tested edge cases like image size, confidence thresholds, and processing time.
- **Non-Functional Coverage**
  - Load Testing: Simulated multiple feeds.

- Response Time: Ensured alerts <1 second.

3 Test Deliverables

3.1. Test Plan Document

3.2. Test Cases and Test Execution Results

Boundary Value Analysis:

BOUNDARY VALUE ANALYSIS

ID	Test Cases	Actual Output	Expected Output	Remark
1	Frame Rate:15FPS	Video Processes without lag	Processed correctly	Pass
2	Frame Rate:60FPS	Video Processes without lag	Processed correctly	Pass
3	Frame Rate:14FPS	Error: Rate too low	Error detected	Pass
4	Frame Rate:61FPS	Error: Rate too high	Error detected	Pass
5	Confidence: 0.85	Detection successful	Detection successful	Pass
6	Confidence: 1	Detection successful	Detection successful	Pass
7	Confidence: 0.84	Error: Confidence too low	Error detected	Pass
8	Confidence: 1.1	Error: Confidence not possible	Error detected	Pass

Decision Table:

DECISION TABLE

Condition	Detection COnfidence	Environmental Conditions	Action
High Confidence, Favourable	>=0.85	Clear	Confirm Detection and alert Security
High Confidence, Unfavourable	>=0.85	Foggy/Low visibility	Confirm Detection and alert Security And verify manually
Medium Confidence, Favouable	0.60-0.84	Clear	Flag detection
Medium Confidence, Unfavouable	0.60-0.84	Foggy/Low visibility	Log event and notify for investigation
low Confidence	<0.60	Any	Ignore detection as noise

3.3. Requirement Traceability Matrix (RTM)

MODEL

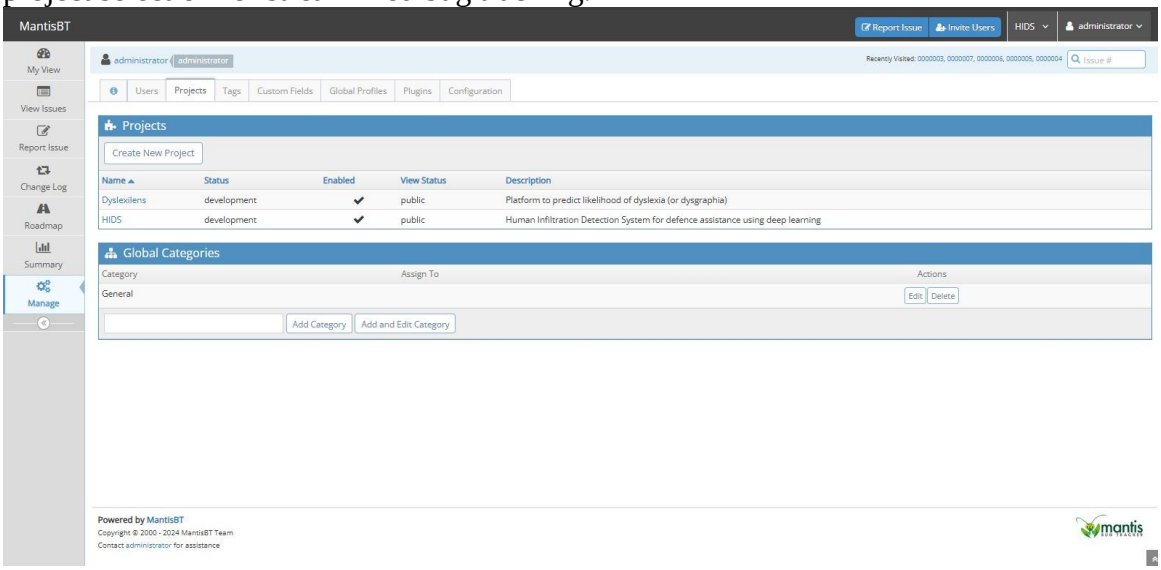
Serial Number	Test Id	Input	Module/Feature	Result
1	TC1	Live video feed with/without intrusion	Intrusion detection	covered
2	TC2	Process video stream in real-time	Data processing	covered
3	TC3	Integration with existing surveillance cameras	System integration	covered
4	TC4	Handle low light and poor visibilty	Image processing	covered
5	TC5	Provide alert immediately and report generatio	Alert Generation System	covered
6	TC6	System Scalabilty	System Architecture	covered

SIGN UP PAGE					
Serial Number	**test Id**	**Input**	**Actual Output**	**Expected Output**	**Result**
1	TC1	Signup Credentials are correct	Signup Success	Signup Success	Pass
2	TC2	Login Credentials are correct	Login Success	Login Success	Pass
3	TC3	Signup Credentials are not correct	Signup Fail	Signup Fail	Pass
4	TC4	Login Credentials are not correct	Login Fail	Login Fail	Pass
5	TC5	Signup, UserName Not Provided	Signup Fail	Signup Fail	Pass
6	TC6	Login, UserName or Password wrong	Login Fail	Login Fail	Pass

3.4. Bug Reports with Resolution Details

Steps to Report a Bug

- **Log in to MantisBT**:
  Accessed the MantisBT interface using project-specific credentials. Ensured appropriate project selection for streamlined bug tracking.



- **Navigate to Bug Reporting Section**:
  Clicked on the "Report Issue" tab to open the bug submission form.

- **Filled in Key Bug Details**:
  Provided a structured and detailed report for each bug:

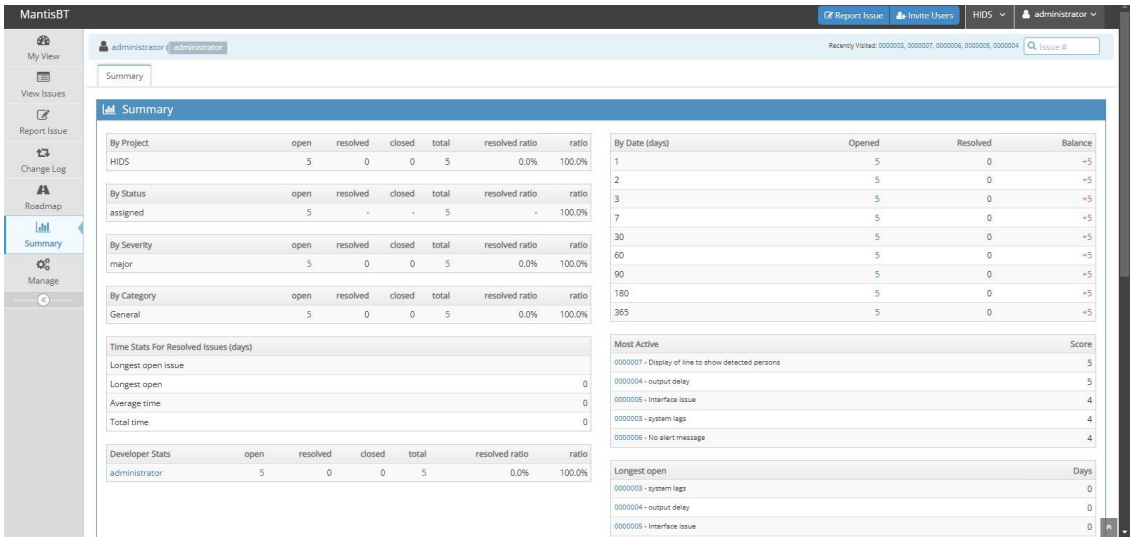
  - **Category**: Chose the relevant module or component (e.g., Preprocessing, CNN model).

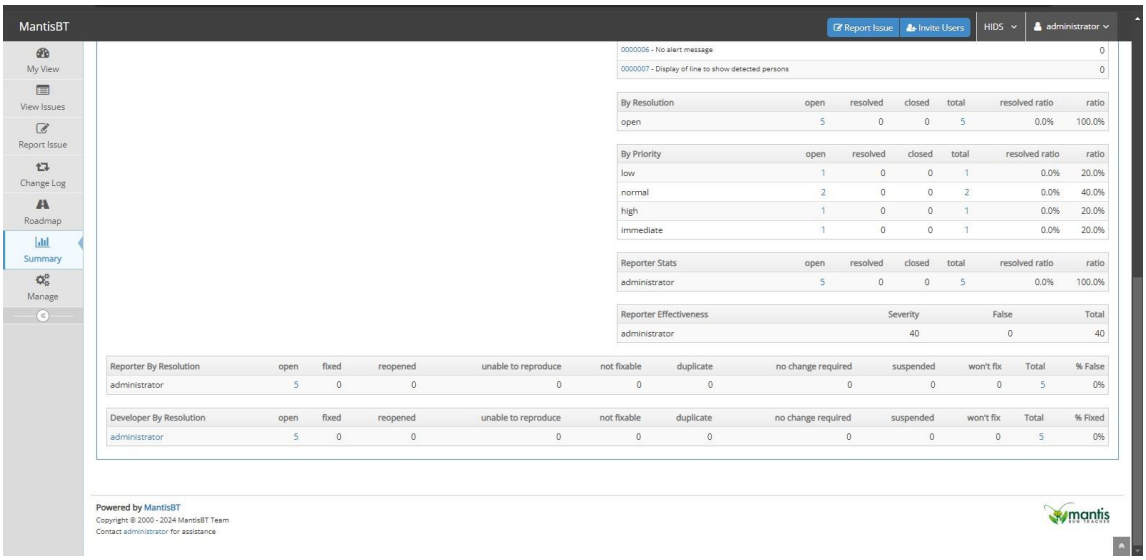
  - **Severity**: Classified the bug as *critical*, *major*, *minor*, *or trivial*. For example, a misclassification in intrusion detection was marked as critical.

  - **Priority**: Assigned urgency for fixing the bug (e.g., High for real-time data issues).

  - **Reproducibility**: Stated whether the bug was consistent or intermittent.

- **Bug Description**:
  Added a detailed explanation, including:
    - Steps to reproduce the issue.
    - Expected vs. actual behaviour.
    - Environment details (OS, tools used, dataset information).
- **Attachment of Evidence**:
  Uploaded logs, screenshots, or video clips that clearly depicted the issue. For instance, confusion matrices highlighting incorrect model predictions were attached for algorithmic bugs.
- **Assigning the Bug**:
  Assigned the bug to the relevant team member, such as developers or QA analysts, ensuring proper accountability.
- **Monitor and Update**:
  Periodically tracked the bug status in MantisBT and updated it based on resolution progress or new findings.





- **Report Generated**

Id	Project	Reporter	Assigned To	Priority	Severity	Reproducibility	Product Version	Category	Date Submitted	OS	OS Version	Platform	View Status	Updated	Summary	Status	Resolution	Fixed in Version
7	HIDS	administrator	administrator	low	feature	random		General	11/26/2024				public	11/26/2024	Display of line to show detected persons	assigned	open	
6	HIDS	administrator	administrator	immediate	major	always		General	11/26/2024				public	11/26/2024	No alert message	assigned	open	
5	HIDS	administrator	administrator	normal	minor	sometimes		General	11/26/2024				public	11/26/2024	Interface issue	assigned	open	
4	HIDS	administrator	administrator	normal	minor	sometimes		General	11/26/2024				public	11/26/2024	output delay	assigned	open	
3	HIDS	administrator	administrator	high	major	always		General	11/26/2024				public	11/26/2024	system lags	assigned	open	

4 Resource & Environment Needs

4.1 Testing Tools

- **Bug Tracking Tool**: MantisBT
- **Automation Tools**: Selenium, TensorFlow for deep learning model validation

4.2 Test Environment

- **Hardware Requirements**:
  - Minimum: 8GB RAM, i5 Processor, SSD.
  - Recommended: 16GB RAM, i7 Processor, NVIDIA GPU.
- **Software Requirements**:
  - Windows 10 and above.
  - Python 3.11, TensorFlow, OpenCV, and other libraries.

5 Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	DEFINITION
API	Application Program Interface
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
AUT	Application Under Test