# Full Documentation of the Adminis Backend

This document provides comprehensive documentation for the Adminis Backend, a web application built using the Flask framework in Python. This application aims to detect and analyze network attacks using the UNSW-NB15 dataset.

## Table of Contents

Introduction

Setup and Configuration

Core Components

API Endpoints

Authentication and User Management

Health and Metrics

Security Incidents and Analysis

Alert System

React App Service

How Autonomous Detection Works

Security

Error Handling

## 1. Introduction

The Adminis Backend is a core part of a security platform designed to monitor and analyze network traffic to detect attacks. It uses a machine learning model (Random Forest Classifier) pre-trained on known attack data. This server provides a set of Application Programming Interfaces (APIs) that allow for seamless integration with a frontend, offering a comprehensive view of security threats.

## 2. Setup and Configuration

The application requires some environment variables for setup.

Environment Variables:

ALLOWED\_ORIGINS: A list of origins allowed to access the API (e.g., http://localhost:3000,http://127.0.0.1:3000).

JWT\_SECRET\_KEY: A secret key for signing JWT tokens (automatically generated if not provided).

FLASK\_ENV: The operating environment (e.g., development or production).

PORT: The port on which the server will run (default: 8080).

UNSW\_DATA\_PATH: The path to the CSV file for training data (default: ./data/UNSW\_NB15\_training-set.csv).

Required Files:

UNSW\_NB15\_training-set.csv: CSV file for model training.

random\_forest\_model\_multiclass.pkl: The trained Random Forest model (if it exists, it will be loaded instead of training).

attack\_label\_encoder.pkl: Label encoder for attack categories.

cat\_encoders.pkl: Label encoders for categorical features.

scaler.pkl: Scaler for numerical features.

Application Startup:

The application is initialized and started automatically when mohamed.py is executed. It performs the following steps:

Loads the dataset (or creates a mock dataset if unavailable).

Loads or trains the machine learning model.

Initializes default alert rules.

Starts the autonomous detection thread.

## 3. Core Components

Flask and Flask-CORS: The web framework for the API and CORS support for cross-origin access.

Pandas and Numpy: For data manipulation and analysis.

Scikit-learn: For implementing machine learning models (Random Forest Classifier, LabelEncoder, StandardScaler).

Joblib: For saving and loading machine learning models.

PyJWT: For creating and verifying JSON Web Tokens (JWT) for authentication.

threading and queue: For managing background tasks and streams (e.g., autonomous detection and incident streaming).

Alert System: For creating and managing alerts based on defined rules.

User Management: A simple user system (admin, analyst, viewer) with roles and permissions.

## 4. API Endpoints

All API endpoints return JSON responses with the following structure:

{

"status": "success" | "error",

"data": { ... },

"message": "Descriptive message",

"error\_code": "Error Code" | null

}

### a. Authentication and User Management

POST /api/auth/login:

Logs in a user and obtains a JWT token.

Input: username, password.

Output: token, user information (id, username, email, role, permissions).

POST /api/auth/logout (Token Required):

Logs out the user and invalidates the token.

GET /api/auth/me (Token Required):

Retrieves information about the current user.

GET /api/users (Token Required, Admin Privileges):

Retrieves a list of all users.

POST /api/users (Token Required, Admin Privileges):

Creates a new user.

Input: username, password, email, role (default: 'viewer').

PUT /api/users/<int:user\_id> (Token Required, Admin Privileges):

Updates user information by ID.

Input: email, role, active, password (optional).

DELETE /api/users/<int:user\_id> (Token Required, Admin Privileges):

Deletes a user by ID.

### b. Health and Metrics

GET /api/health:

Checks the service health, including dataset availability and model loading status.

GET /api/metrics (Token Required, Read Permissions):

Retrieves threat metrics (number of malicious URLs, IoT attacks, ransomware incidents, etc.).

GET /api/system-health (Token Required, Read Permissions):

Retrieves detailed system health information (CPU usage, memory, disk, network, model status).

### c. Security Incidents and Analysis

POST /api/predict (Token Required, Read Permissions):

Predicts the attack type based on network features.

Input: features (dictionary containing network features).

Output: attack\_category, prediction, confidence, probabilities, timestamp.

GET /api/security-events-timeline (Token Required, Read Permissions):

Retrieves security events timeline data categorized by attack types.

GET /api/recent-incidents (Token Required, Read Permissions):

Retrieves recent incidents with support for filtering and pagination.

Query Parameters: scenario, src\_ip, mac\_address, src\_country, src\_city, page, per\_page.

GET /api/incidents (Token Required, Read Permissions):

Retrieves recent security incidents.

Query Parameters: limit, scenario.

GET /api/incidents/stream (Token Required, Read Permissions):

Streams real-time incidents using Server-Sent Events (SSE).

GET /api/locations (Token Required, Read Permissions):

Retrieves a list of unique countries and cities from recent incidents. Useful for creating filters in the frontend.

GET /api/models (Token Required, Read Permissions):

Lists available models (currently: "Random Forest").

GET /api/performance (Token Required, Read Permissions):

Retrieves model performance metrics.

GET /api/statistics (Token Required, Read Permissions):

Retrieves general security statistics (total incidents, attack categories, scenarios, recent activity).

GET /api/reports/generate (Token Required, Read Permissions):

Generates a report in LaTeX format (or PDF if pdflatex is installed) for a specific scenario.

Query Parameters: scenario, type (summary, detailed, statistics), format (latex, pdf).

### d. Alert System

GET /api/alerts (Token Required, Read Permissions):

Retrieves alerts, with support for filtering by status, severity, and result limit.

POST /api/alerts/<int:alert\_id>/acknowledge (Token Required, Write Permissions):

Acknowledges a specific alert.

POST /api/alerts/<int:alert\_id>/resolve (Token Required, Write Permissions):

Resolves a specific alert.

GET /api/alert-rules (Token Required, Read Permissions):

Retrieves configured alert rules.

POST /api/alert-rules (Token Required, Write Permissions):

Creates a new alert rule.

### e. React App Service

/ and /<path:path>:

Serves the React Frontend application from the build folder (static folder). It defaults to serving index.html for any non-existent path.

## 5. How Autonomous Detection Works

The autonomous\_detection function runs in a separate daemon thread. It simulates real-time attack detection by:

Randomly sampling rows from the loaded (or mock) dataset.

Preprocessing features using encoders and scalers.

Using the Random Forest model to predict the attack category and confidence probability.

Creating a detailed incident record that includes the attack category, date, threat level, confidence, source IP address, MAC address, and now, country and city information.

Adding the incident to the recent\_incidents list and the incident\_queue (for live streaming to the frontend).

Checking if the incident triggers any defined alert rules and generating necessary alerts.

Repeating the process every 5 seconds.

## 6. Security

The application uses a JWT-based authentication system with roles and permissions.

@token\_required: This decorator ensures that the request contains a valid JWT token in the Authorization header (in the format Bearer <token>). It verifies the token's validity and presence in active sessions.

@permission\_required(permission): Ensures that the current user has the specified permission (e.g., read, write, admin) to access the API endpoint.

@admin\_required: Ensures that the current user has the admin role to access the API endpoint.

Default User Roles:

admin: admin123 (read, write, admin)

analyst: analyst123 (read, write)

viewer: viewer123 (read)

## 7. Error Handling

The application includes error handlers for:

404 Not Found: When attempting to access a non-existent endpoint.

500 Internal Server Error: For unexpected internal server errors.

Standardized JSON responses are returned for these errors with status: "error" and specific error codes.