Intel Unnati Industrial Training

Project Report - AI/ML for Networking

Team - ByteSentinel

NetGuardian: ML based Web Attack Detection

Trainees:

Aditya Kumar

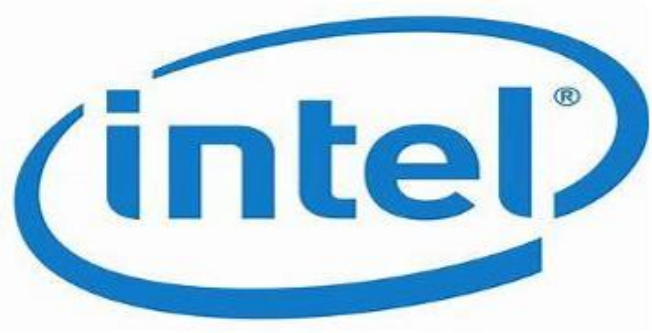
Putta Nithish Reddy

College Mentor:

Dr. V. Sivakumar

Associate Professor

Manipal Institute of Technology, Manipal



Index

Project Live Link - https://netguardian.streamlit.app/

Video Demonstration Link - https://youtu.be/PllRsvbmCo8

Github Link - https://github.com/adityak814/NetGuardian

Objective:

The primary objective of this project is to detect and classify web-based cyber attacks, particularly SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks, using machine learning techniques. The system aims to:

- Build a robust, data-driven model to distinguish between normal and malicious HTTP requests.
- Develop a binary classifier to detect whether a request is benign or malicious.
- Implement an anomaly detection model using One-Class SVM to flag suspicious traffic patterns.
- Design a multi-class classifier to identify the specific type of attack: SQLi, XSS, or other unknown forms.
- Provide a user-friendly dashboard interface for real-time testing and visualization of detection results.
- Enable early and automated threat detection to strengthen web application security.

This project bridges machine learning with web security by combining natural language processing (NLP), vectorization, and classification algorithms to safeguard web servers from common yet dangerous vulnerabilities.

Abstract:

This project aims to detect web-based cyber threats such as SQL Injection (SQLi) and Cross-Site Scripting (XSS) using machine learning. Models are trained on the CSIC 2010 HTTP dataset to classify HTTP requests as benign, SQLi, XSS, or other anomalies. Techniques used include binary classification with SVM, anomaly detection with One-Class SVM, and multi-class classification with LightGBM. Feature extraction combines TF-IDF, count vectorization, and statistical metrics. The system is deployed with a Flask API and an interactive Streamlit dashboard. Results demonstrate high accuracy and real-time detection capabilities for securing web applications.

Flow of the Project:

Fig 1: project flow

Detailed Explanation:

1. Dataset Description - csic_database.csv

The csic_database.csv file is derived from the CSIC 2010 HTTP dataset, which contains simulated HTTP traffic representing both benign and malicious web requests. It is widely used for research in web application security, particularly for detecting SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks.

The dataset includes key fields such as Method, URL, content, User-Agent, and a classification label, where 0 denotes normal traffic and 1 indicates an attack. These features are combined into a single text string for preprocessing.

The data is used to train three types of models:

- A binary classifier (SVM) to detect attack vs normal requests.
- An anomaly detector (One-Class SVM) to identify deviations from normal patterns.
- A multi-class classifier (LightGBM) to categorize attacks into SQLi, XSS, or Others.

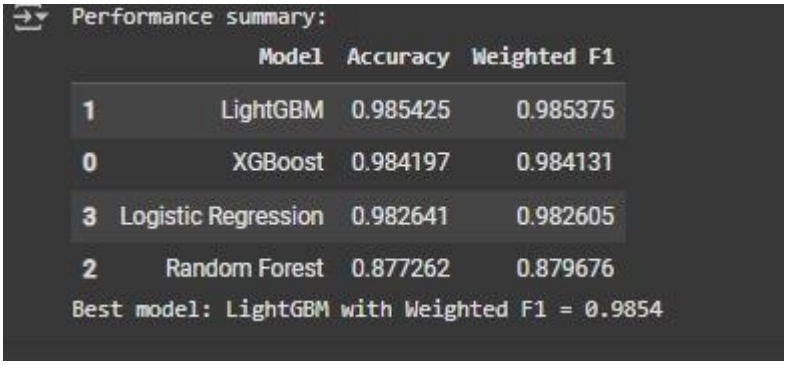
Additional statistical features such as request length, special character ratio, and entropy are also extracted. The dataset forms the backbone of the entire system, supporting training, evaluation, and real-time prediction of web threats.

2. Model Evaluation and Selection

During the model development phase, several machine learning algorithms were explored and evaluated, including Support Vector Machine (SVM), Logistic Regression, Random Forest, XGBoost, and LightGBM. The performance of each model was assessed based on Accuracy and Weighted F1-score.

As shown in the figures below, LightGBM, SVM, and Logistic Regression consistently achieved the highest evaluation metrics. In the **multi-class classification** task, **LightGBM** achieved an **accuracy of 98.54%** with a weighted **F1-score of 0.9854**, outperforming XGBoost and Random Forest. Similarly, for **binary classification**, **SVM** yielded the best results with an **accuracy of 98.32%** and an **F1-score** of **0.9833**, making it the most suitable model for detecting normal versus attack traffic. We evaluated two **unsupervised anomaly detectors**, **One-Class SVM** and LOF (with PCA for dimensionality reduction) on normal traffic. Both performed strongly, with **One-Class SVM** achieving **91.35% accuracy** compared to LOF's 90.47%, giving OCSVM only a slight edge.

Based on these results, only LightGBM, SVM, and One-Class SVM were selected and deployed in the final system, ensuring optimal detection accuracy and generalization.

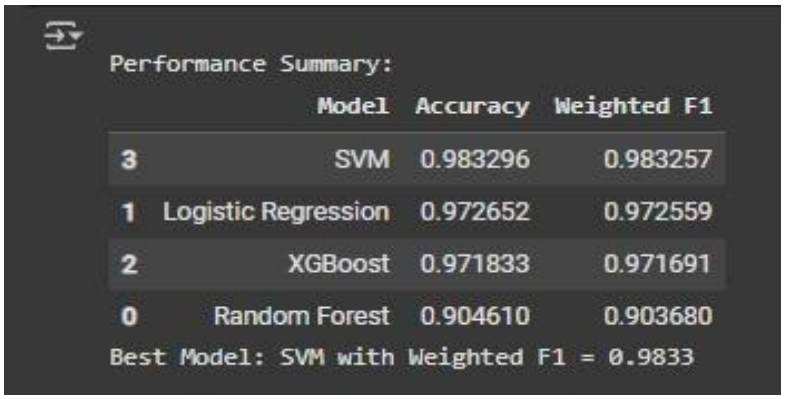


Performance summary:

	Model	Accuracy	Weighted F1
1	LightGBM	0.985425	0.985375
0	XGBoost	0.984197	0.984131
3	Logistic Regression	0.982641	0.982605
2	Random Forest	0.877262	0.879676

Best model: LightGBM with Weighted F1 = 0.9854

Fig 2: Multi-class classification Performance Summary

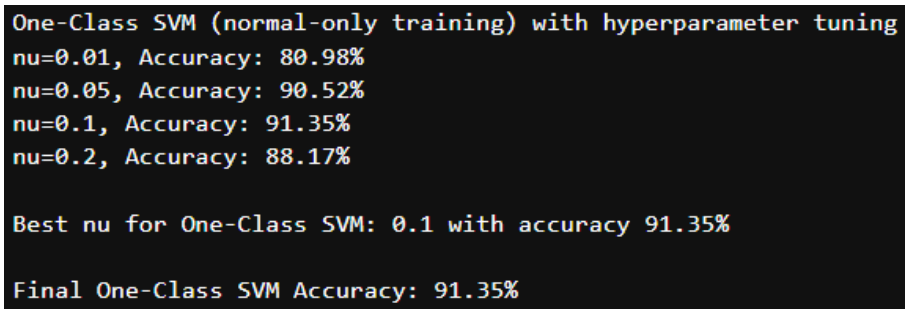


Performance Summary:

	Model	Accuracy	Weighted F1
3	SVM	0.983296	0.983257
1	Logistic Regression	0.972652	0.972559
2	XGBoost	0.971833	0.971691
0	Random Forest	0.904610	0.903680

Best Model: SVM with Weighted F1 = 0.9833

Fig 3: Binary classification Performance Summary

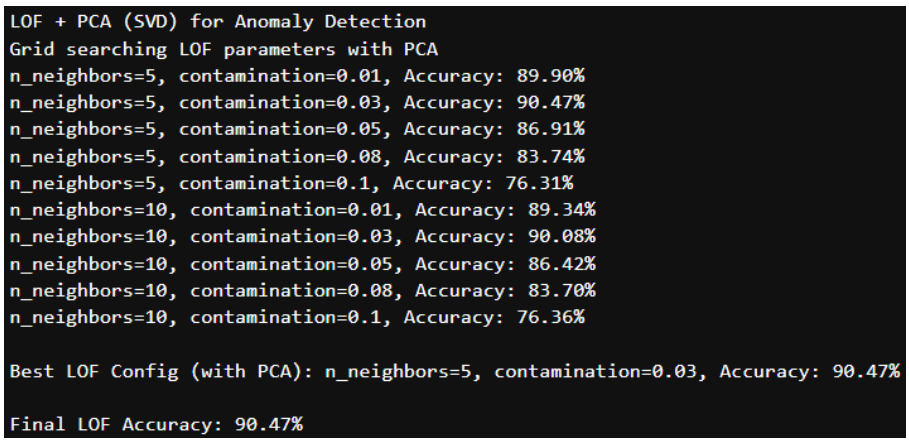


```
One-Class SVM (normal-only training) with hyperparameter tuning
nu=0.01, Accuracy: 80.98%
nu=0.05, Accuracy: 90.52%
nu=0.1, Accuracy: 91.35%
nu=0.2, Accuracy: 88.17%

Best nu for One-Class SVM: 0.1 with accuracy 91.35%

Final One-Class SVM Accuracy: 91.35%
```

Fig 4: Anomaly Detection (One-Class SVM)



```
LOF + PCA (SVD) for Anomaly Detection
Grid searching LOF parameters with PCA
n_neighbors=5, contamination=0.01, Accuracy: 89.90%
n_neighbors=5, contamination=0.03, Accuracy: 90.47%
n_neighbors=5, contamination=0.05, Accuracy: 86.91%
n_neighbors=5, contamination=0.08, Accuracy: 83.74%
n_neighbors=5, contamination=0.1, Accuracy: 76.31%
n_neighbors=10, contamination=0.01, Accuracy: 89.34%
n_neighbors=10, contamination=0.03, Accuracy: 90.08%
n_neighbors=10, contamination=0.05, Accuracy: 86.42%
n_neighbors=10, contamination=0.08, Accuracy: 83.70%
n_neighbors=10, contamination=0.1, Accuracy: 76.36%

Best LOF Config (with PCA): n_neighbors=5, contamination=0.03, Accuracy: 90.47%

Final LOF Accuracy: 90.47%
```

Fig 5: Anomaly Detection (LocalOutlierFactor with PCA)

3. Explanation - AnomalyDetection_BinaryClassifier.ipynb

This Jupyter Notebook implements two key models to detect web-based attacks from HTTP request data:

Objective

To develop:

- A **Binary Classifier** to distinguish between normal and attack traffic.
- An **Anomaly Detection model** to flag deviations from normal behavior.

Dataset Used

The notebook uses csic_database.csv, which contains labeled HTTP requests with fields like Method, URL, content, User-Agent, and a classification label (0 = benign, 1 = attack).

Data Preprocessing

- Missing values in the four textual fields are filled with empty strings.
- All four fields are combined into a single text column.
- A **TF-IDF vectorizer** is fitted on this combined text to extract relevant features from HTTP requests.

Binary Classification using SVM

- The dataset is split into training and testing sets (80–20).
- A **Support Vector Machine (SVM)** with a linear kernel is trained to classify inputs as attack (1) or normal (0).
- Evaluation is done using **accuracy score** and **classification report** (precision, recall, F1-score).
- The trained model is saved as classifier.pkl.

Anomaly Detection using One-Class SVM

- Only the **normal traffic samples (label 0)** from the training set are used to train a **One-Class SVM**.
- This model learns the pattern of benign requests and flags outliers (potential attacks).
- The anomaly score is derived using the decision function, and results are converted into binary labels (normal vs anomaly).
- This model is saved as anomaly_detector.pkl.

4. Explanation - MultiClass_Classification_Model.ipynb

This notebook focuses on building a **multi-class classification model** to identify the specific type of web attack in an HTTP request. Unlike binary classification, which simply detects whether a request is benign or malicious, this model distinguishes between different types of attacks.

Objective

To classify HTTP requests into:

- 0: Normal
- 1: SQL Injection (SQLi)
- 2: Cross-Site Scripting (XSS)
- 3: Other attacks

Dataset Preparation

- The notebook uses the same csic_database.csv file.
- Fields (Method, URL, content, User-Agent) are cleaned and combined into a unified text feature.
- Labels are assigned based on regex pattern matching and keyword scoring to identify SQLi and XSS content.

Feature Engineering

Two types of features are extracted:

1. **Textual Features**:
    - TF-IDF (word-based)
    - TF-IDF (character n-grams)
    - CountVectorizer (character-based)
2. **Statistical Features**:
    - Request length, URL depth, entropy
    - Ratios of digits, special characters, spaces, quotes
    - Frequency of SQLi/XSS-related keywords and symbols

All features are combined using scipy.hstack() into a single feature matrix.

Model Training

- A **LightGBM Classifier** is used due to its speed and high performance on large feature sets.

- Data is split into 80% training and 20% testing using stratified sampling.
- The model is evaluated using **accuracy** and **weighted F1-score**.

5. Explanation – flask_app.py

The flask_app.py script serves as the **backend API** of the web attack detection system. It provides **three REST endpoints** that allow real-time HTTP request analysis using the machine learning models trained earlier.

Objective

To expose ML models for:

- Binary classification
- Anomaly detection
- Multi-class attack classification through a web-accessible **Flask API**.

Model Loading

Upon startup, the Flask app loads all the required serialized components using pickle, including:

- **Binary classifier model** (classifier.pkl)
- **Anomaly detector** (anomaly_detector.pkl)
- **Multi-class LightGBM model** (lightgbm_model.pkl)
- Corresponding **TF-IDF and Count vectorizers** and **scaler**

Preprocessing Functions

Functions are defined to:

- **Preprocess input text**: decode, normalize, and sanitize user input
- **Extract numerical/statistical features** such as length, special character ratio, entropy, SQLi/XSS pattern scores
- **Build combined text fields** for vectorization

API Endpoints

The Flask app provides three endpoints:

- POST /predict_binary: returns 0 (benign) or 1 (attack)
- POST /predict_anomaly: returns an anomaly score; values > 0 are considered anomalous
- POST /predict_multiclass: returns 0, 1, 2, or 3 based on attack type (Normal, SQLi, XSS, Other)

Each endpoint:

- Accepts a JSON payload with fields: method, url, content, user_agent
- Transforms the input into feature vectors
- Runs inference using the corresponding model
- Returns the prediction as a JSON response

Integration Role

This API is consumed by the **Streamlit dashboard** to provide real-time attack predictions and display the results to the user interactively.

Deployment

The flask_app.py backend is deployed on the Railway Cloud Platform (https://railway.com/) to handle requests from the dashboard.py Streamlit frontend.

6. Explanation - dashboard.py

The dashboard.py script implements an **interactive user interface** using **Streamlit**, allowing users to test and visualize the detection of web-based attacks such as **SQL Injection (SQLi)** and **Cross-Site Scripting (XSS)** in real-time.

Objective

To provide a **web dashboard** that:

- Displays random HTTP request samples
- Sends them to the backend Flask API
- Shows predictions from the models with user-friendly labels

Dataset Loading

The csic_database.csv file is loaded, and a combined text field is created from the HTTP fields: Method, URL, content, and User-Agent.

Attack Type Labeling

Requests labeled as attacks are further classified into:

- **SQLi (1)**
- **XSS (2)**
- **Other attacks (3)** based on regular expressions and pattern matching.

Sample Generation

For demonstration, the dashboard randomly selects up to 5 samples for each category:

- Benign
- SQLi
- XSS
- Other

Users can click buttons in the sidebar to view a **random request** from each category.

API Integration

When a sample is selected:

- A request payload is built and sent to the **Flask API** endpoints:
  - predict_anomaly
  - predict_binary
  - predict_multiclass
- Each model's response is displayed with icons, color coding, and explanations

User Interface

- Built using Streamlit with clear section titles and markdown formatting
- The sidebar contains sample selection buttons
- The main page shows the request, prediction results, and diagnostic messages (e.g., "Anomaly Detected")

Real-Time Visualization

This dashboard provides a **visual and interactive experience** to understand how different types of attacks are detected, making the ML system interpretable and usable.

Glimpse of the Project:

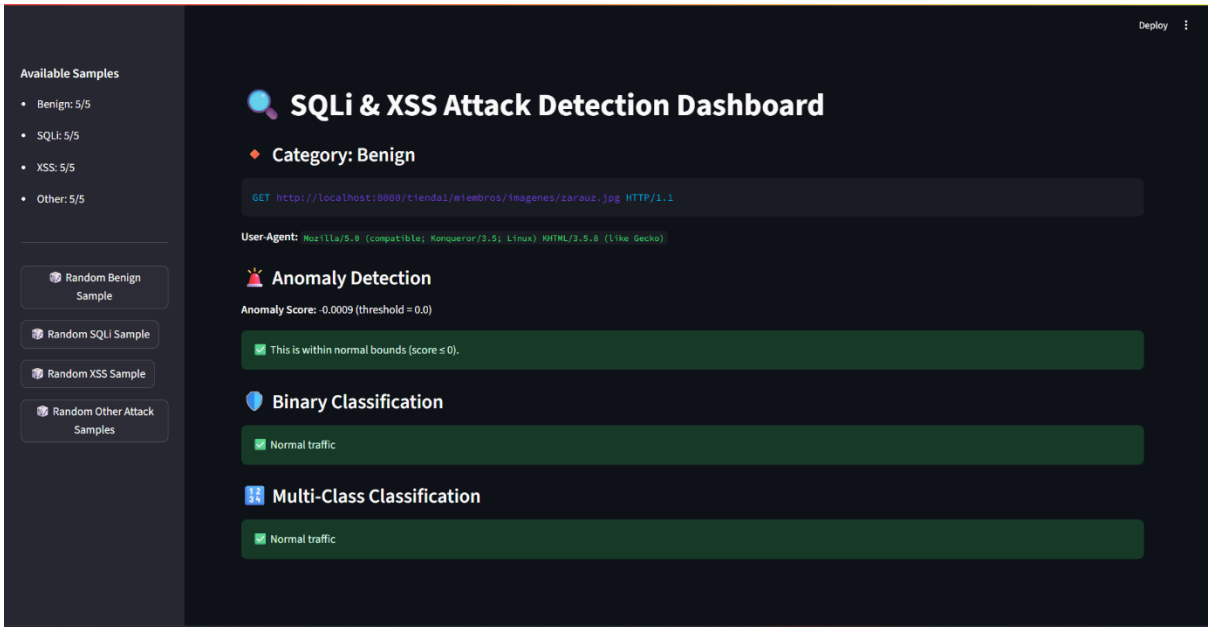


Fig 6: Random Benign Sample

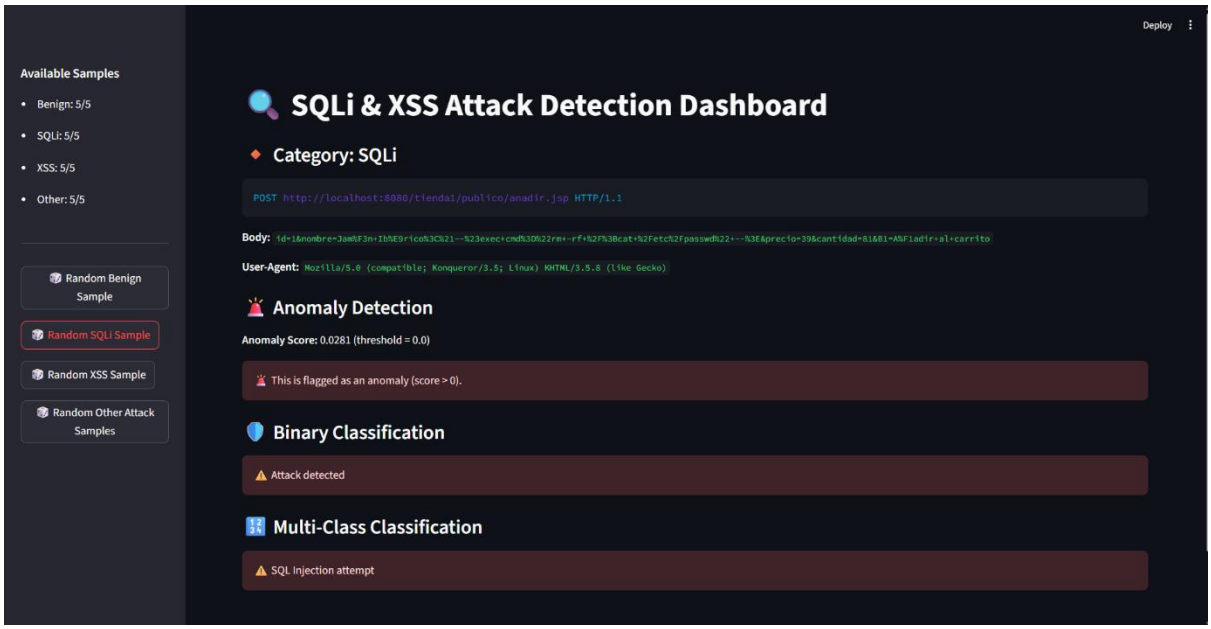


Fig 7: Random SQLi Sample

Fig 8: Random XSS Sample

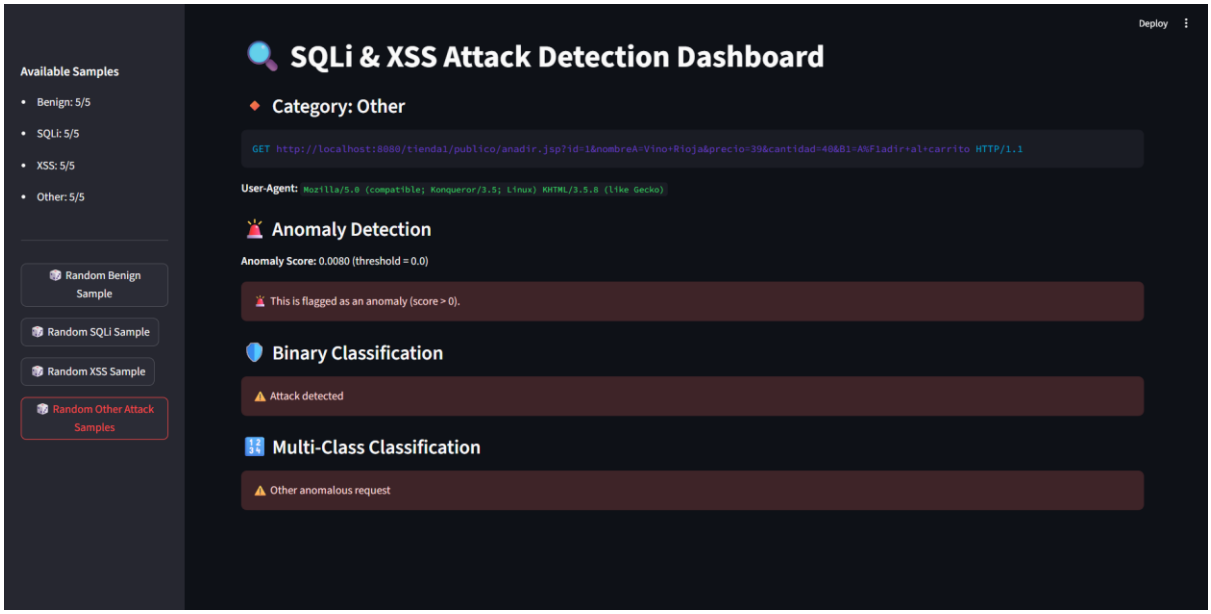


Fig 9: Random Other Attack Samples

Individual Contributions

Aditya Kumar: Multi-class Classification model using LightGBM, handling advanced feature engineering and model optimization, Flask backend cloud deployment, Streamlit dashboard.

Nithish Reddy: Binary classification and Anomaly Detection models, including preprocessing, model training, and API integration, Flask backend.

Conclusion

This project successfully demonstrates the use of machine learning techniques to detect and classify web-based attacks, specifically SQL Injection (SQLi), Cross-Site Scripting (XSS), and other anomalies. By combining binary classification, anomaly detection, and multi-class classification models, the system ensures both accuracy and adaptability in identifying malicious HTTP requests. The integration of a Flask backend and Streamlit dashboard allows for real-time predictions and a user-friendly interface. Experimental results show high detection accuracy and generalizability across various attack types. Overall, this solution offers a scalable and intelligent approach to enhancing web application security.