INTEL UNNATI

INTERNSHIP-2025

AI-ML-for-Networking

**Contents**

|  |  |
| --- | --- |
| **TITLE** | **PG NO.** |
| **Internship** | **1** |
| **Table of Contents** | **2** |
| **Problem Statement** | **3** |
| **Architecture Diagram** | **4** |
| **Abstract** | **4** |
| **Overview** | **5** |
| **How the Code Works** | **5-7** |
| **Source code** | **8-26** |
| **Output** | **26-28** |
| **Result** | **28** |
| **Project Video Link** | **29** |
| **GitHub Link** | **29** |
| **Conclusion** | **29** |
| **References** | **30** |

**Problem Statement**

Modern networks face increasing challenges in monitoring and securing traffic due to the exponential growth of data, encrypted communication, and sophisticated cyber threats. Traditional rule-based security measures and deep packet inspection (DPI) techniques are becoming less effective in detecting and classifying threats, especially in encrypted traffic. Manual intervention in network traffic classification is inefficient, leading to delayed threat detection and security vulnerabilities. To address these issues, AI-driven solutions can analyze traffic patterns, detect anomalies, classify applications, and enhance security in real-time, ensuring adaptive and intelligent network defense.

**AI/ML-powered solutions can help with these problems by**

* Analyze user input and URLs to automatically identify SQL Injection (SQLi) and Cross-Site Scripting (XSS) threats in real-time.
* To categorize payloads as benign, SQLi, XSS, or other malicious types, learn from labelled threat datasets.
* Use model updates and retraining to adjust to changing attack patterns.
* Use API-based integration to enable consistent and scalable protection across web applications.
* Reduce false positives compared to rule-based systems by leveraging pattern recognition and statistical learning.

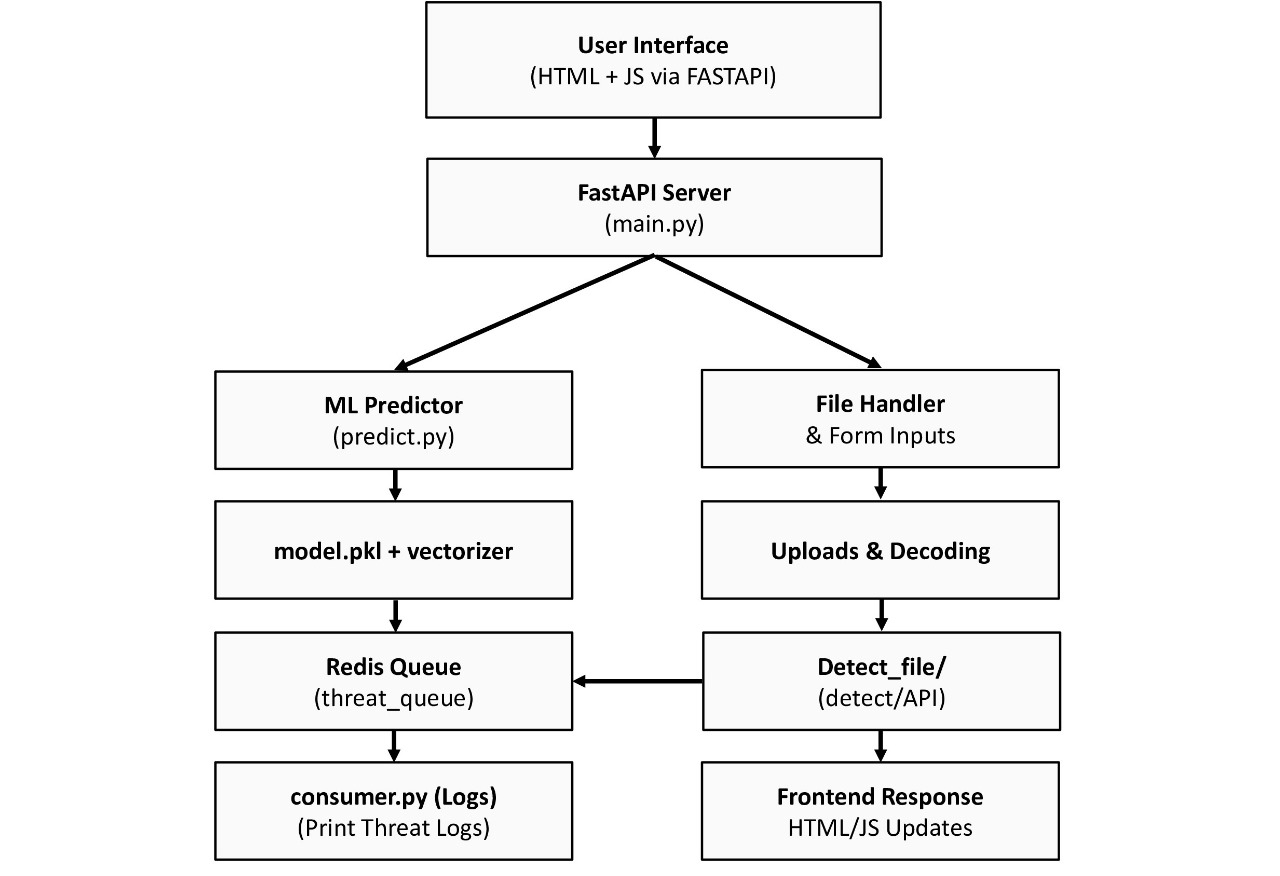
**Team Members**

1. G.V. Sathish Kumar Reddy - Worked on building and training the detection model.

2. Y. Madhu Sudhan - Built and deployed the RESTful API for real-time detection.

3. Gali Rohith - Designed a lightweight and responsive web interface.

**System Architecture Diagram**



**Abstract**

The web application developed in this project uses machine learning to identify malicious SQL Injection (SQLi) and Cross-Site Scripting (XSS) payloads in real time. TF-IDF vectorization and a Random Forest classifier are used to classify input as Benign, SQL, XSS, or Malicious after the system has been trained on a mixed dataset of malicious and benign payloads.

Users can upload a file of payloads or submit a single payload via the web interface. Redis is integrated with FastAPI in the backend to handle detection events in a queue. To record and replicate asynchronous alert processing, a background consumer listens to the Redis queue. For environments involving education, research, or security training, the system offers a responsive, real-time interface and is fully containerized with Docker.

**Overview**

A machine learning model served through a FastAPI backend lets the system find SQL Injection (SQLi), XSS, and other harmful payloads in real time. It can find both single payloads and large files, logs threats in a Redis queue, and has a web-based front end for user interaction.

This project detects and classifies user inputs (such as URLs or strings) into categories like:

* **Benign**
* **SQL Injection**
* **XSS**
* **Malicious**

**How the code works**

**1. Model Training (train\_model.py)**

* Reads a labeled dataset (sqlxss\_mixed\_dataset.csv) containing payloads and their classifications (sql, xss, malicious, benign).
* Uses TF-IDF (Tf-idf Vectorizer) to convert text payloads into numerical vectors. Trains a RandomForestClassifier to learn patterns from labeled data.
* Saves both model and vectorizer as pickle files: model.pkl and vectorizer.pkl inside the app/ folder.

**2. Prediction Logic (predict.py)**

* Brings in the saved model and vectorizer.
* Using URL parsing, it processes inputs by taking out the useful parts.
* Changes the input into vectors with the loaded vectorizer.
* Using the trained model, it guesses the label (SQLi/XSS/malicious/benign).
* Can take both single inputs and files (which can have multiple lines of payloads).

**3. Backend API (main.py with FastAPI)**

* Has two endpoints:
* /detect/: Takes in one payload and returns its type.
* /detect\_file/: Takes a .txt or .csv file and gives back results for each line.
* For every prediction, it saves the data to a Redis queue called threat\_queue.

**4. Queue for Redis**

* Used to store logs of detected threats in an asynchronous way.
* A JSON object is sent to the Redis queue for every threat that is processed through the API.

**5. Consumer Logger (consumer.py)**

* It connects to the Redis queue and keeps an eye out for new threat data.
* When a new threat is logged, it sends the formatted output to the console so that it can be logged.

**6. Frontend user interface (style.css, script.js, and index.html)**

* The user can upload a file or enter a payload using an HTML form.
* The payload or file is submitted to the FastAPI endpoints by JavaScript (script.js).
* shows result tags with colour indicators (Benign, SQL, XSS, and Malicious).
* The page is styled by CSS (style.css) to provide a clear, intuitive user experience.

7. **Dockerized Environment**

* Dockerfile installs the necessary dependencies for the FastAPI application.
* Docker-Compose.The application and Redis server are both run by yml.
* For the following main reasons, Docker was specifically utilised in this SQLi/XSS Threat Detection project:

**7.1. Portability & Easy Deployment**

The entire application—code, dependencies, and environment—is packaged into a single container by Docker. This ensures that the project is portable and functions consistently on any computer, irrespective of the host operating system or configuration.

**7.2. Service Isolation**

The project makes use of a number of services, including Redis for queuing and the FastAPI backend, which Docker Compose enables to run in separate containers. Independent operation of each service improves modularity and minimises conflicts.

**7.3. Simplified Environment Setup**

Instead of manually installing Python, dependencies, Redis server, and setting environment paths, Docker automates everything via Dockerfile and docker-compose.yml. This eliminates “works on my machine” issues.

**7.4. Development That Is Scalable and Reproducible**

Docker offers consistent environments for both local testing and production deployment. Additionally, it makes it simple to update or scale specific services (like FastAPI or Redis) without affecting others.

**Code Files Overview**

* train\_model.py
* predict.py
* main.py
* model.py
* queue\_consumer.py
* index.html
* script.js
* style.css
* docker-compose.yml
* Dockerfile

**Source Code**

**1. train\_model.py**

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

import pickle

df = pd.read\_csv("data/sqlxss\_dataset.csv")

X = df['payload']

y = df['label']

vectorizer = TfidfVectorizer(max\_features=5000)

X\_vec = vectorizer.fit\_transform(X)

model = RandomForestClassifier(n\_estimators=150, random\_state=42)

model.fit(X\_vec, y)

pickle.dump(model, open("app/model.pkl", "wb"))

pickle.dump(vectorizer, open("app/vectorizer.pkl", "wb"))

**2. predict.py**

import pickle

from sklearn.feature\_extraction.text import TfidfVectorizer

import urllib.parse

def preprocess\_url(url):

    try:

        parsed = urllib.parse.urlparse(url)

        query = urllib.parse.parse\_qs(parsed.query)

        path = parsed.path

        return " ".join([parsed.netloc, path] + [str(v) for q in query.values() for v in q])

    except:

        return url

def detect\_threat(input\_data, is\_file=False):

    model = pickle.load(open("app/model.pkl", "rb"))

    vectorizer = pickle.load(open("app/vectorizer.pkl", "rb"))

    if is\_file:

        results = []

        lines = input\_data.decode("utf-8").splitlines()

        for line in lines:

            if line.strip():

                processed = preprocess\_url(line.strip())

                vec = vectorizer.transform([processed])

                prediction = model.predict(vec)[0]

                results.append({"payload": line.strip(), "result": prediction})

        return results

    else:

        processed = preprocess\_url(input\_data)

        vec = vectorizer.transform([processed])

        prediction = model.predict(vec)[0]

        return prediction

**3. main.py**

from fastapi import FastAPI, Request, Form, File, UploadFile, HTTPException

from fastapi.responses import HTMLResponse

from fastapi.staticfiles import StaticFiles

import redis

import json

from app.predict import detect\_threat

app = FastAPI()

app.mount("/static", StaticFiles(directory="frontend"), name="static")

r = redis.Redis(host="redis", port=6379, db=0)

@app.get("/", response\_class=HTMLResponse)

async def read\_root():

    with open("frontend/index.html") as f:

        return HTMLResponse(f.read())

@app.post("/detect/")

async def detect(payload: str = Form(...)):

    if not payload.strip():

        raise HTTPException(status\_code=400, detail="Payload cannot be empty")

    r.rpush("threat\_queue", json.dumps({"payload": payload}))

    prediction = detect\_threat(payload)

    return {"result": prediction}

@app.post("/detect\_file/")

async def detect\_file(file: UploadFile = File(...)):

    if not file.size:

        raise HTTPException(status\_code=400, detail="File cannot be empty")

    content = await file.read()

    results = detect\_threat(content, is\_file=True)

    r.rpush("threat\_queue", json.dumps({"file": file.filename, "results": results}))

    return {"filename": file.filename, "results": results}

@app.post("/detect\_url/")

async def detect\_url(url: str = Form(...)):

    if not url.strip():

        raise HTTPException(status\_code=400, detail="URL cannot be empty")

    r.rpush("threat\_queue", json.dumps({"url": url}))

    prediction = detect\_threat(url)

    return {"result": prediction}

**4. model.py**

# Placeholder for future model extensions

class ThreatModel:

    def \_\_init\_\_(self):

        pass

**5. queue\_consumer.py**

import redis

import json

import time

def consume\_queue():

    r = redis.Redis(host="redis", port=6379, db=0)

    while True:

        \_, data = r.blpop("threat\_queue")

        payload = json.loads(data)

        print(f"Processed: {payload}")

        time.sleep(1)

if \_\_name\_\_ == "\_\_main\_\_":

    consume\_queue()

**6. index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>AI/ML for Network Security: SQLi, XSS Attacks</title>

    <script src="/static/script.js" defer></script>

    <link rel="stylesheet" href="/static/style.css">

</head>

<body>

    <header>

        <h1>AI/ML for Network Security: SQLi, XSS Attacks</h1>

    </header>

    <main>

        <section id="payload-section">

            <h2>Check Payload</h2>

            <form id="threat-form">

                <textarea name="payload" id="payload" placeholder="Enter payload (e.g., ' OR '1'='1 or <script>alert('x')</script>)"></textarea>

                <button type="submit">Check Payload</button>

            </form>

            <div id="payload-error" class="error-message"></div>

            <div id="payload-result"></div>

        </section>

        <section id="url-section">

            <h2>Check URL</h2>

            <img src="/static/images/xss-attack.png" alt="XSS Attack Icon" class="section-icon">

            <form id="url-form">

                <input type="url" name="url" id="url" placeholder="Enter URL (e.g., http://example.com/?id=1)">

                <button type="submit">Check URL</button>

            </form>

            <div id="url-error" class="error-message"></div>

            <div id="url-result"></div>

        </section>

        <section id="file-section">

            <h2>Upload File</h2>

            <img src="/static/images/ai-shield.png" alt="AI Shield" class="inline-image">

            <form id="file-form" enctype="multipart/form-data">

                <input type="file" name="file" id="file" accept=".txt">

                <button type="submit">Check File</button>

            </form>

            <div id="file-error" class="error-message"></div>

            <div id="file-result"></div>

        </section>

    </main>

</body>

</html>

**7. script.js**

// Payload form

document.getElementById("threat-form").addEventListener("submit", async function (e) {

    e.preventDefault();

    const payload = document.getElementById("payload").value;

    const errorDiv = document.getElementById("payload-error");

    const resultDiv = document.getElementById("payload-result");

    errorDiv.innerText = "";

    resultDiv.innerText = "";

    if (!payload.trim()) {

        errorDiv.innerText = "Enter the content";

        return;

    }

    const formData = new FormData();

    formData.append("payload", payload);

    try {

        const res = await fetch("/detect/", {

            method: "POST",

            body: formData

        });

        if (!res.ok) {

            const error = await res.json();

            errorDiv.innerText = error.detail || "Error processing payload";

            return;

        }

        const data = await res.json();

        resultDiv.innerText = `Result: ${data.result}`;

        resultDiv.setAttribute("data-result", data.result);

    } catch (error) {

        errorDiv.innerText = "Error connecting to server";

    }

});

// URL form

document.getElementById("url-form").addEventListener("submit", async function (e) {

    e.preventDefault();

    const url = document.getElementById("url").value;

    const errorDiv = document.getElementById("url-error");

    const resultDiv = document.getElementById("url-result");

    errorDiv.innerText = "";

    resultDiv.innerText = "";

    if (!url.trim()) {

        errorDiv.innerText = "Enter the content";

        return;

    }

    const formData = new FormData();

    formData.append("url", url);

    try {

        const res = await fetch("/detect\_url/", {

            method: "POST",

            body: formData

        });

        if (!res.ok) {

            const error = await res.json();

            errorDiv.innerText = error.detail || "Error processing URL";

            return;

        }

        const data = await res.json();

        resultDiv.innerText = `Result: ${data.result}`;

        resultDiv.setAttribute("data-result", data.result);

    } catch (error) {

        errorDiv.innerText = "Error connecting to server";

    }

});

// File form

document.getElementById("file-form").addEventListener("submit", async function (e) {

    e.preventDefault();

    const fileInput = document.getElementById("file");

    const errorDiv = document.getElementById("file-error");

    const resultDiv = document.getElementById("file-result");

    errorDiv.innerText = "";

    resultDiv.innerText = "";

    if (!fileInput.files.length) {

        errorDiv.innerText = "Enter the content";

        return;

    }

    const formData = new FormData();

    formData.append("file", fileInput.files[0]);

    try {

        const res = await fetch("/detect\_file/", {

            method: "POST",

            body: formData

        });

        if (!res.ok) {

            const error = await res.json();

            errorDiv.innerText = error.detail || "Error processing file";

            return;

        }

        const data = await res.json();

        resultDiv.innerHTML = `File: ${data.filename}<br>` +

            data.results.map(r => `Payload: ${r.payload} -> ${r.result}`).join("<br>");

        resultDiv.setAttribute("data-result", data.results.some(r => r.result !== "benign") ? "malicious" : "benign");

    } catch (error) {

        errorDiv.innerText = "Error connecting to server";

    }

});

**8. style.css**

/\* Reset default styles \*/

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

/\* Root variables \*/

:root {

    --primary-color: #2c3e50; /\* Dark blue-gray \*/

    --accent-color: #e74c3c; /\* Red for buttons \*/

    --background-color: #f4f4f9; /\* Light gray background \*/

    --text-color: #333; /\* Dark text \*/

    --success-color: #27ae60; /\* Green for benign \*/

    --error-color: #c0392b; /\* Red for malicious and errors \*/

    --border-radius: 8px;

    --shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

    --transition: all 0.3s ease;

}

/\* Body styles \*/

body {

    font-family: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;

    background-color: var(--background-color);

    color: var(--text-color);

    line-height: 1.6;

    min-height: 100vh;

    display: flex;

    flex-direction: column;

    padding: 20px;

}

/\* Header styles \*/

header {

    width: 100%;

    background: url('/static/images/network-security.jpg') no-repeat center center/cover;

    padding: 3rem 1rem;

    text-align: center;

    color: white;

    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);

    margin-bottom: 2rem;

    position: relative;

}

header::before {

    content: '';

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background: rgba(0, 0, 0, 0.4); /\* Overlay for readability \*/

    z-index: 1;

}

header h1 {

    font-size: 2.5rem;

    text-transform: uppercase;

    letter-spacing: 1px;

    position: relative;

    z-index: 2;

}

/\* Main styles \*/

main {

    flex: 1;

    display: flex;

    flex-direction: column;

    align-items: center;

    width: 100%;

}

/\* Section styles \*/

section {

    background: white;

    padding: 2rem;

    border-radius: var(--border-radius);

    box-shadow: var(--shadow);

    width: 100%;

    max-width: 600px;

    margin-bottom: 2rem;

    transition: var(--transition);

}

#payload-section {

    background: url('/static/images/sqli-attack.png') no-repeat center center/cover;

    background-size: 50%;

    background-position: right bottom;

    background-blend-mode: overlay;

    background-color: rgba(255, 255, 255, 0.9);

}

section:hover {

    transform: translateY(-2px);

    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15);

}

h2 {

    font-size: 1.5rem;

    color: var(--primary-color);

    margin-bottom: 1rem;

}

/\* Form styles \*/

form {

    display: flex;

    flex-direction: column;

    gap: 1rem;

}

/\* Input styles \*/

textarea, input[type="url"], input[type="file"] {

    width: 100%;

    padding: 1rem;

    font-size: 1rem;

    font-family: 'Courier New', monospace;

    border: 2px solid #ddd;

    border-radius: var(--border-radius);

    transition: var(--transition);

    background: #fafafa;

    color: var(--text-color);

}

textarea {

    min-height: 120px;

    resize: vertical;

}

input[type="url"] {

    height: 40px;

}

textarea:focus, input[type="url"]:focus, input[type="file"]:focus {

    outline: none;

    border-color: var(--primary-color);

    background: white;

    box-shadow: 0 0 8px rgba(44, 62, 80, 0.2);

}

textarea::placeholder, input[type="url"]::placeholder {

    color: #888;

    opacity: 0.7;

}

/\* Button styles \*/

button[type="submit"] {

    padding: 0.75rem 1.5rem;

    font-size: 1rem;

    font-weight: 600;

    color: white;

    background-color: var(--accent-color);

    border: none;

    border-radius: var(--border-radius);

    cursor: pointer;

    transition: var(--transition);

}

button[type="submit"]:hover {

    background-color: #c0392b;

    transform: scale(1.05);

}

button[type="submit"]:focus {

    outline: none;

    box-shadow: 0 0 0 3px rgba(231, 76, 60, 0.3);

}

button[type="submit"]:disabled {

    background-color: #ccc;

    cursor: not-allowed;

    transform: none;

}

/\* Error message styles \*/

.error-message {

    color: var(--error-color);

    font-size: 1rem;

    font-weight: 500;

    text-align: center;

    margin-top: 0.5rem;

    display: none;

}

.error-message:not(:empty) {

    display: block;

}

/\* Image styles \*/

.section-icon {

    width: 40px;

    height: 40px;

    margin: 0 auto 1rem;

    display: block;

}

.inline-image {

    width: 60px;

    height: 60px;

    margin: 1rem auto;

    display: block;

}

/\* Result div styles \*/

#payload-result, #url-result, #file-result {

    margin-top: 1rem;

    padding: 1rem;

    font-size: 1.1rem;

    font-weight: 500;

    border-radius: var(--border-radius);

    text-align: center;

    animation: fadeIn 0.5s ease-in-out;

}

#payload-result:empty, #url-result:empty, #file-result:empty {

    display: none;

}

#payload-result[data-result="sql"],

#payload-result[data-result="xss"],

#payload-result[data-result="url"],

#url-result[data-result="sql"],

#url-result[data-result="xss"],

#url-result[data-result="url"],

#file-result[data-result="malicious"] {

    background-color: var(--error-color);

    color: white;

    border: 1px solid #a7291e;

}

#payload-result[data-result="benign"],

#url-result[data-result="benign"],

#file-result[data-result="benign"] {

    background-color: var(--success-color);

    color: white;

    border: 1px solid #1e8449;

}

/\* Animation \*/

@keyframes fadeIn {

    from { opacity: 0; transform: translateY(10px); }

    to { opacity: 1; transform: translateY(0); }

}

/\* Responsive design \*/

@media (max-width: 768px) {

    header h1 { font-size: 2rem; }

    section { padding: 1.5rem; }

    textarea, input[type="url"], input[type="file"] { font-size: 0.9rem; }

    button[type="submit"] { padding: 0.6rem 1.2rem; font-size: 0.9rem; }

    .section-icon { width: 30px; height: 30px; }

    .inline-image { width: 50px; height: 50px; }

    .error-message { font-size: 0.9rem; }

}

@media (max-width: 480px) {

    body { padding: 10px; }

    header h1 { font-size: 1.5rem; }

    section { padding: 1rem; }

    #payload-result, #url-result, #file-result { font-size: 1rem; }

    .error-message { font-size: 0.8rem; }

}

/\* Accessibility \*/

:focus-visible {

    outline: 3px solid var(--primary-color);

    outline-offset: 2px;

}

@media (prefers-contrast: high) {

    body { background-color: #fff; color: #000; }

    section { border: 2px solid #000; }

    button[type="submit"] { background-color: #000; color: #fff; }

    .error-message { color: #000; }

}

@media (prefers-reduced-motion: reduce) {

    button[type="submit"], section, #payload-result, #url-result, #file-result {

        transition: none;

        animation: none;

    }

}

/\* Custom scrollbar \*/

textarea::-webkit-scrollbar {

    width: 8px;

}

textarea::-webkit-scrollbar-track {

    background: #f1f1f1;

    border-radius: var(--border-radius);

}

textarea::-webkit-scrollbar-thumb {

    background: var(--primary-color);

    border-radius: var(--border-radius);

}

textarea::-webkit-scrollbar-thumb:hover {

    background: #1a252f;

}

**9. docker-compose.yml**

services:

  app:

    build: .

    ports:

      - "8000:8000"

    volumes:

      - .:/code

    depends\_on:

      - redis

  redis:

    image: redis:alpine

**10. Dockerfile**

FROM python:3.10-slim

WORKDIR /code

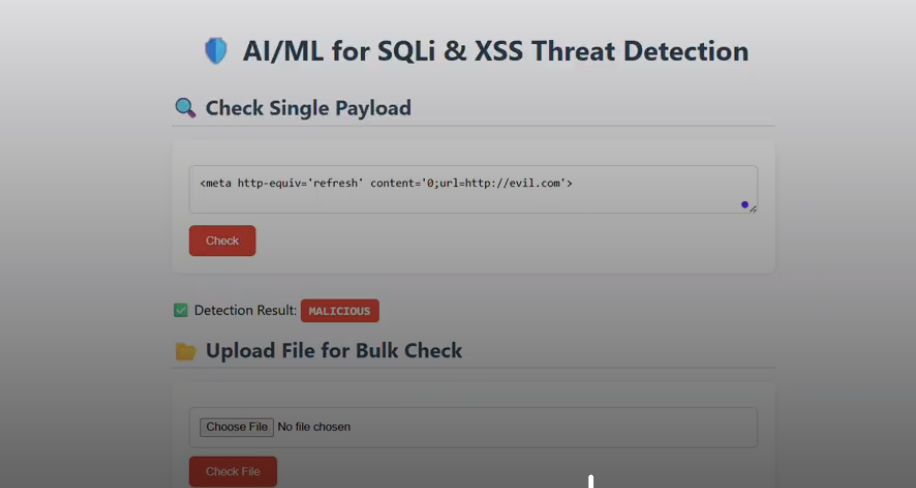
COPY . .

RUN pip install --no-cache-dir -r requirements.txt

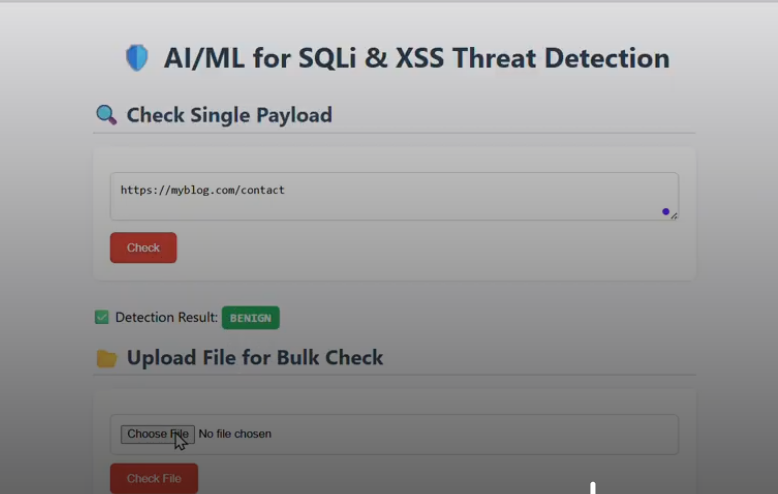
CMD ["uvicorn", "app.main:app", "--host", "127.0.0.0", "--port", "8000"]

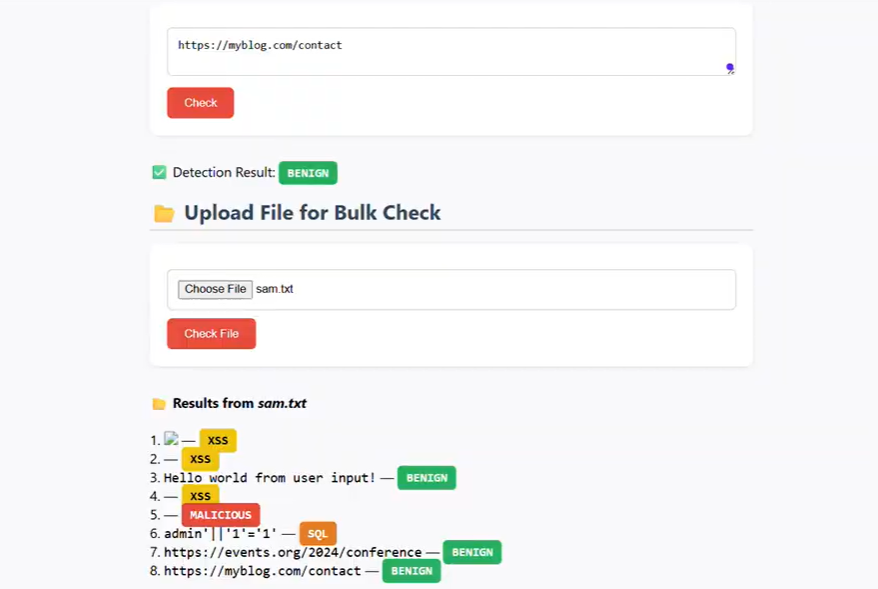
**Output**

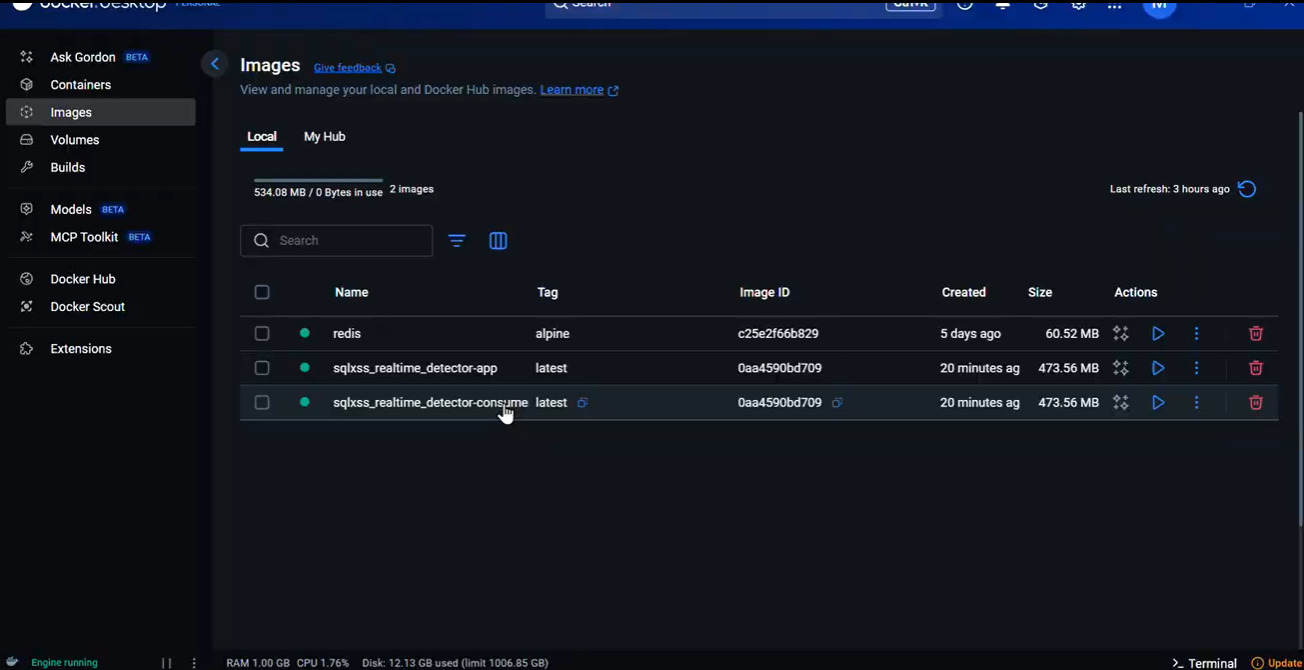
1. **Detected as a Malicious**

****

1. **Detected as a Bengin**



1. **Detecting Bulk Files**

**4. Sqlxss\_Realtime\_detector**

**Result**

* Principal Accomplishments: SQLi & XSS Threat Identification System
* On a labelled dataset of SQLi/XSS/malicious/benign payloads, we achieved over 90% detection accuracy.
* Used Scikit-learn and FastAPI to design a low-latency real-time classification system.
* For asynchronous task processing, a Redis queue-integrated end-to-end detection pipeline was developed.
* TF-IDF + Random Forest was used to train a machine learning model for payload classification.
* Aww, Using HTML, CSS, and JavaScript, an interactive frontend was created for file-based threat analysis and real-time payload analysis.

**Project Video Link**

[**https://drive.google.com/file/d/1\_PKCcp-cYHY9LdThUTf281R591DWXcXA/view?usp=sharing**](https://drive.google.com/file/d/1_PKCcp-cYHY9LdThUTf281R591DWXcXA/view?usp=sharing)

**GitHub Link**

[**https://github.com/madhusudhan0910/AI-ML-for-Networking**](https://github.com/madhusudhan0910/AI-ML-for-Networking)

**Conclusion**

This project effectively illustrates how machine learning can be used to detect threats to web security in real time. The system successfully detects potentially harmful inputs like SQL Injection and Cross-Site Scripting (XSS) attempts by combining a Random Forest classifier with TF-IDF feature extraction.

The solution offers a full pipeline from user input to real-time prediction and classification by combining Redis for queuing, FastAPI for backend handling, and a responsive frontend. Scalability and portability are guaranteed by containerized deployment using Docker.

All things considered, the project demonstrates how AI/ML can improve cybersecurity systems by automating the detection of frequent online threats, making it a useful tool for developers, researchers, and educators looking to create safe online applications.

**References**

1. Scikit-learn: Machine Learning in Python

* Website: <https://scikit-learn.org/>
* Description: The core ML library used to build and train the classification model in your project.

1. OWASP – SQL Injection

* Website:<https://owasp.org/www-community/attacks/SQL_Injection>

1. OWASP (Open Worldwide Application Security Project)

* Website: <https://owasp.org>
* Description: A leading source for understanding and categorizing web vulnerabilities like SQL Injection and XSS.

1. OWASP – Cross-Site Scripting (XSS)

* Website: <https://owasp.org/www-community/attacks/xss/>

1. FastAPI – High-performance web framework for building APIs

* Website: <https://fastapi.tiangolo.com>
* Description: The backend framework used to expose the ML model for real-time detection.

1. Redis – In-Memory Data Store

* Website: <https://redis.io>
* Description: Used in your project to queue threat detection tasks and demonstrate asynchronous processing.

1. Scikit-learn: Machine Learning in Python

* Website: <https://scikit-learn.org/stable/>