

Progress Report: Web Application Development with Model Integration

Project Overview: The goal of this project is to develop a Flask-based web application that integrates a machine learning model to predict web vulnerabilities based on user inputs. Key tasks performed are

- Responsive front-end using HTML, CSS, JavaScript.
- Flask-based backend with AI model integration.
- Real-world testing and deployment.

Responsive front-end using HTML, CSS

For the front end, the AI-Powered Threat Detection system combines HTML, CSS, and JavaScript to create a clean, responsive, and appealing interface. HTML organizes content on all pages; CSS gives modern touch-and-feel and interactivity. Three different webpages are created:

- home.html
- about.html
- contact.html
- admin.html
- files include:

Home.html

Home.html : This HTML file represents the main landing page for the AI-Powered Threat Detection project. It uses Flask's `url_for` function to dynamically load static assets (CSS, and images) and provides a modern, responsive layout with the following sections:

1. Hero Section:

```
<section class="hero">
  <div class="container">
    <h2>Revolutionizing Cloud Security</h2>
    <p>AI-driven solutions to detect and mitigate threats in real-time. Stay ahead of attackers with cutting-edge technologies.</p>
    <a href="/about" class="cta-button">Learn More</a>
  </div>
```

- Highlights the project's mission with a visually striking background and a call-to-action button.

2. Features Section:

```
<section class="features">
  <div class="container">
    <h3>Why Choose AI for Cloud Security?</h3>
    <div class="feature-grid">
      <div class="feature-item">
        <h4>Real-Time Threat Detection</h4>
        
        <p>Monitor and neutralize threats instantly using AI-powered algorithms.</p>
      </div>
      <div class="feature-item">
        <h4>Advanced Data Analytics</h4>
        
        <p>Leverage big data to identify patterns and vulnerabilities proactively.</p>
      </div>
      <div class="feature-item">
        <h4>Comprehensive Coverage</h4>
        
        <p>Protect all layers of your cloud network from sophisticated attacks.</p>
      </div>
    </div>
  </div>
</section>
```

- Displays key benefits of the AI solution in a grid layout with icons and descriptions.

3. Navigation & Footer:

```
<header>
  <div class="container">
    <h1>AI-Powered Threat Detection</h1>
    <nav>
      <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/about">About</a></li>
        <li><a href="/contact">Contact</a></li>
        <li><a href="/admin">Admin</a></li>
      </ul>
    </nav>
  </div>
</header>
```

- Provides intuitive navigation links (Home, About, Contact, Admin)

```
<footer>
  <div class="container">
    <p>© 2024 AI Threat Detection Solutions | All Rights Reserved</p>
  </div>
</footer>
</body>
</html>
```

- A copyright notice in footer .

about.html

This HTML file represents the About Us page for the AI-powered threat detection system. It provides information about the mission and team members, using a clean, organized layout. It has same header and footer section as home page.

1. Mission Section:

```
<section>
  <h2>Our Mission</h2>
  <p>To provide state-of-the-art AI-driven threat detection solutions for securing cloud networks against evolving cyber threats.</p>
</section>
```

- Describes the project's goal of using AI for advanced cloud security.

2. Team Section:

```
<section>
  <h2>Our Team</h2>
  <div class="team-member">
    
    <h3>Akshitha </h3>
    <p><strong>CEO & Founder:</strong> Visionary leader with a passion for AI-driven security solutions.</p>
  </div>
  <div class="team-member">
    
    <h3>Jay</h3>
    <p><strong>CTO:</strong> Technical innovator driving advancements in AI and machine learning.</p>
  </div>
  <div class="team-member">
    
    <h3>Sonam</h3>
    <p><strong>Cybersecurity Specialist:</strong> Expert in cloud security architecture and threat mitigation.</p>
  </div>
  <div class="team-member">
    
    <h3>Harsh</h3>
    <p><strong>Cybersecurity Specialist:</strong> Expert in cloud security architecture and threat mitigation.</p>
  </div>
</section>
```

- Displays team members with images and roles, structured for easy readability.

contact.html

This HTML file creates the Contact Us page for the AI-Powered Threat Detection project. It includes a user-friendly form, essential contact details, and an embedded map for location reference.

1. Contact

Form:

```
<form action="/submit_form" method="post">
  <label for="name">Your Name:</label><br>
  <input type="text" id="name" name="name" placeholder="Enter your name" required><br><br>

  <label for="email">Your Email:</label><br>
  <input type="email" id="email" name="email" placeholder="Enter your email" required><br><br>

  <label for="message">Your Message:</label><br>
  <textarea id="message" name="message" rows="5" placeholder="Write your message here..." required></textarea><br><br>

  <button type="submit">Send Message</button>
</form>
```

- Allows users to submit inquiries with fields for name, email, and message.

2. Contact Details Section:

```
<section>
  <div class="contact-info">
    <p><i class="fas fa-phone"></i> Phone: +61 43 34 24 662</p>
    <p><i class="fas fa-envelope"></i> Email: CQU</p>
    <p><i class="fas fa-map-marker-alt"></i> Ann Street, Brisbane, Queensland</p>
  </div>
</section>
```

- Displays phone, email, and address with icons for better visual appeal.

3. Embedded Map:

```
<section class="map">
  <h3>Find Us Here</h3>
  <iframe
    src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3540.057133333333!1d153.02168267477987!3d-27.46748861661036!2m3!1f0!2f0!3m2!111024!21768!4f13.113m3!1m2!1s0x16c12181e076f7d340x917e9e"
    width="100%"
    height="400"
    style="border:0;"
    allowfullscreen=""
    loading="lazy"
  </iframe>
</section>
```

- Shows the organization's location using Google Maps for easy navigation.

admin.html

This HTML file builds the Admin Panel page, allowing users to input data points for threat detection. It is designed to support data input, processing, and results display in a structured manner.

1. Threat Detection Form:

```
<section>
  <center>
    <h2>Vulnerability Detection</h2>
    <form action="/admin" method="POST">
      <label for="message">Enter Text:</label>
      <textarea id="message" name="message" rows="4" cols="30" required></textarea>
      <br><br>
      <button type="submit">Detect Vulnerability</button>
    </form>
    {% if result %}
      <h3>{{ result }}</h3>
    {% endif %}
  </center>
</section>
```

- Users can enter textual input into the text area field of this form in order to identify a threat (for instance, logs or descriptions).
- After submission, the information is sent into the backend (/admin route) via a POST request
- Based on this input, processes and detects vulnerabilities.
- The prediction takes place dynamically below the form if available

2. Result Display:

```
</form>
<div id="result"></div>
```

- An empty div (<div id="result"></div>) is reserved for dynamically displaying the results of the threat detection.

CSS

This CSS code styles a web page with a modern, user-friendly layout. The body is set up with a gradient background and flexible column layout. The header and footer have a blue background and white text. The main content area features a gradient background, rounded corners, and shadow for depth. Styling for images, team members, contact info, forms, and buttons ensures a visually appealing and interactive design, with responsive elements and hover effects for user engagement.

Justification of methodology

HTML/CSS: HTML provides the structure of web pages, CSS styles them with responsive layouts for different devices Together, they create user-friendly, adaptable websites. CSS ensures a visually appealing design.

Flask-based backend with AI model integration.

Flask application is developed with the following features

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if request.method == 'POST':
        # Get the input from the user
        user_message = request.form['message']

        # Preprocess the input text
        clean_message = preprocess_text(user_message)

        # Vectorize the cleaned message
        vectorized_message = vectorizer.transform([clean_message]).toarray()

        # Predict the vulnerability type using the model
        prediction = model.predict(vectorized_message)[0]

        # Decode the label to get the vulnerability type
        vulnerability_type = label_encoder.inverse_transform([prediction])[0]

        # Return the result to the admin template
        return render_template('admin.html', result=f"Detected Vulnerability Type: {vulnerability_type}")

    return render_template('admin.html')
```

- Home Page describing the purpose and functionality of the system.
- About Page providing details about the project's mission to enhance web security, including information about the team members and their roles in the project.
- The Contact Page providing a simple form for users to submit queries, feedback, or support requests, including fields for name, email, and message.
- Prediction Page (Admin) allowing users to input text data (e.g., logs or descriptions of potential vulnerabilities), preprocessing the input using a predefined pipeline and using the integrated machine learning model (best_model.pkl) to predict the type of vulnerability and displaying the results dynamically.

Model Integration

The following three pre-trained components are integrated into the Flask app using joblib for ensuring accurate predictions:

```
# Load the saved model, vectorizer, and label encoder
model = load("models/best_model.pkl")
vectorizer = load("models/vectorizer.pkl")
label_encoder = load("models/label_encoder.pkl")
```

- Random Forest Model (best_model.pkl) for predicting the type of vulnerability based on the input data.
- TF-IDF Vectorizer (vectorizer.pkl) for converting the input text into numerical features consistent with the model's training data.

- Label Encoder (label_encoder.pkl) for decoding numeric predictions into human-readable vulnerability types.

Preprocessing Pipeline

```
# Preprocessing function for user input
def preprocess_text(text):
    """
    Clean and preprocess input text for prediction.
    """
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    # Lowercase the text
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'^a-z\s', '', text)

    # Remove stopwords and lemmatize words
    text = ' '.join(
        lemmatizer.lemmatize(word)
        for word in text.split()
        if word not in stop_words
    )
    return text
```

Preprocessing function is implemented to clean and prepare user inputs by converting the input text to lowercase, removing special characters and numbers, removes stopwords and lemmatizes words using NLTK libraries to ensure that the input format matches the vectorizer used during training.

Prediction Workflow

```
@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if request.method == 'POST':
        # Get the input from the user
        user_message = request.form['message']

        # Preprocess the input text
        clean_message = preprocess_text(user_message)

        # Vectorize the cleaned message
        vectorized_message = vectorizer.transform([clean_message]).toarray()

        # Predict the vulnerability type using the model
        prediction = model.predict(vectorized_message)[0]

        # Decode the Label to get the vulnerability type
        vulnerability_type = label_encoder.inverse_transform([prediction])[0]

        # Return the result to the admin template
        return render_template('admin.html', result=f"Detected Vulnerability Type: {vulnerability_type}")

    return render_template('admin.html')
```

- User input is accepted via the form (user_message).
- The input is pre-processed using preprocess_text.

- `vectorizer.transform` is used for converting the cleaned text into a numerical format
- Vectorized input is fed into the Random Forest model to predict the vulnerability type.
- Using the label encoder the numeric prediction is decoded.
- Result is displayed dynamically on the admin page.

Justification of using Flask and python

Flask is a lightweight framework ideal for building modular web applications with minimum overhead. Its built-in template engine, Jinja2, allows for generation of HTML dynamically, making it fit for building interactive user interfaces. By combining the flexibility of Flask with the vast ecosystem of Python, developers can use libraries such as NumPy, Pandas, and Scikit-learn to efficiently build enterprise-grade machine learning pipelines.

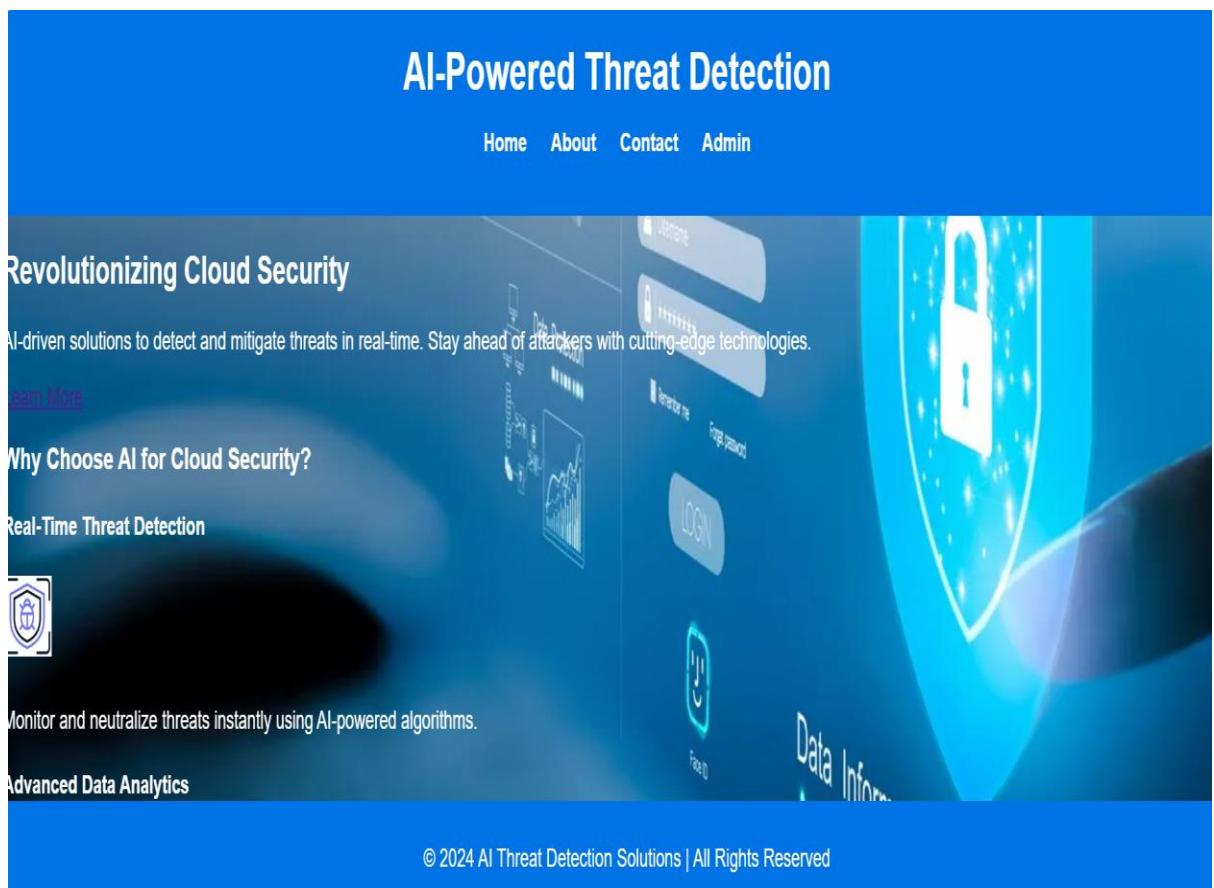
Real-World Testing and Deployment

This phase consists of real-world testing and deployment phase that indicates that the web application is functional in the actual world, processing user inputs, running predictions, and returning accurate outputs. The process of testing and deployment with expected outputs is discussed below:

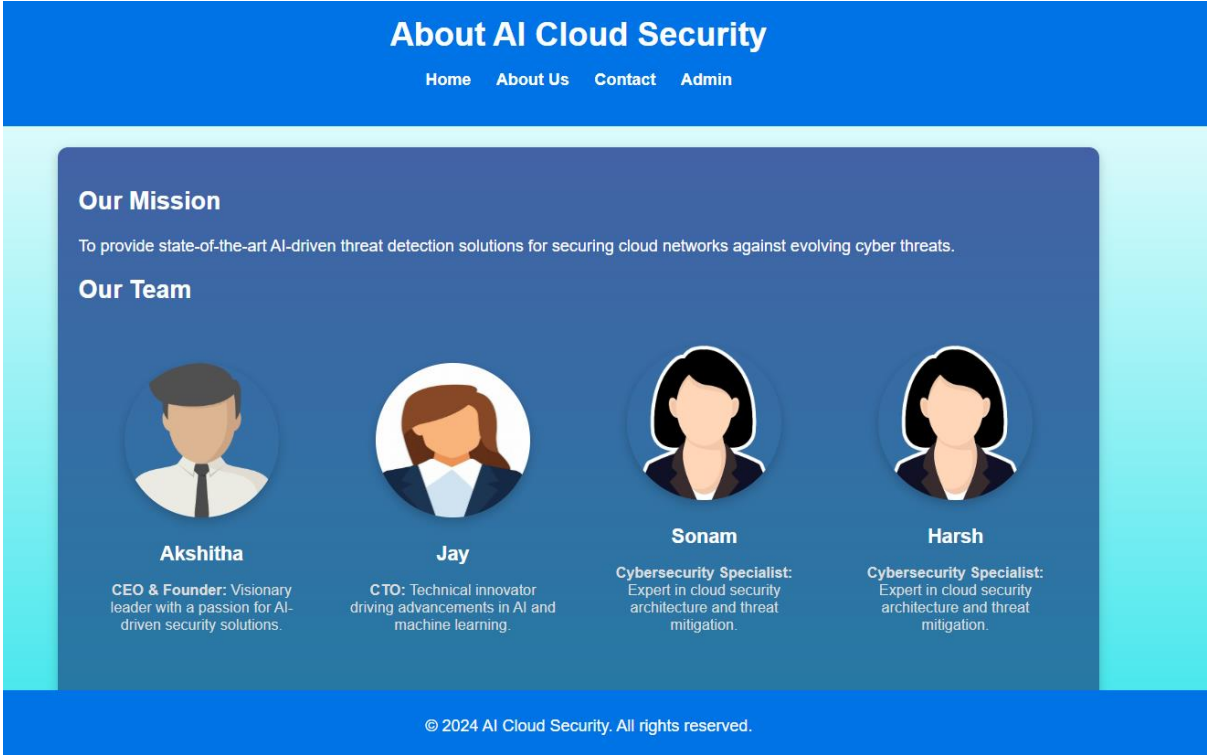
Functional Testing

Functional testing Validates that each page of the application and its functionalities are as per our requirement

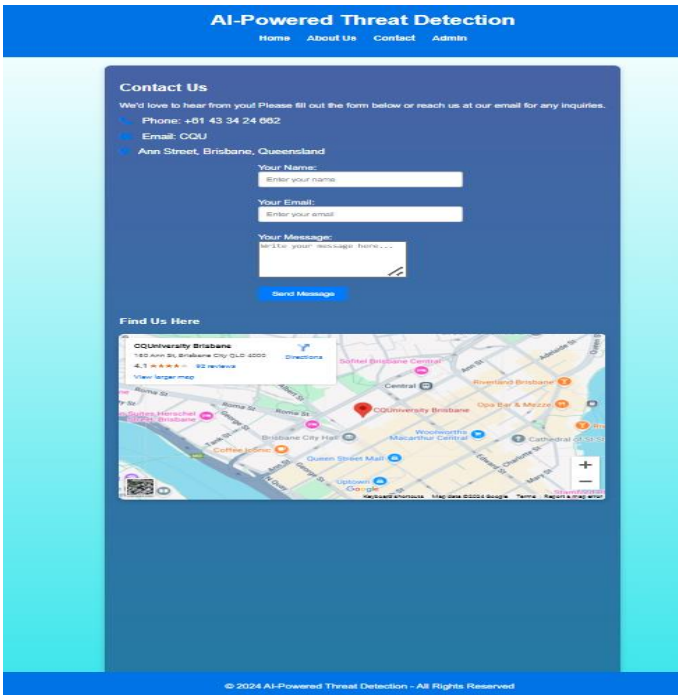
1. Home Page: Open the home page in a browser to validate that the project purpose and navigation links have been displayed.



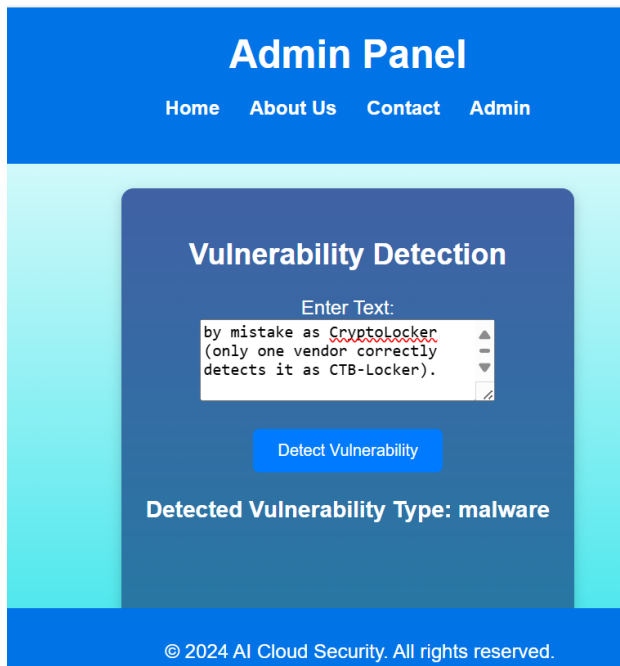
2. About page: Confirm that the mission statement and team details are appropriately displayed.



3. Contact page: The user submits some test data into the contact form, and it is verified that the input fields are processed correctly.



4. Admin Page (Prediction):

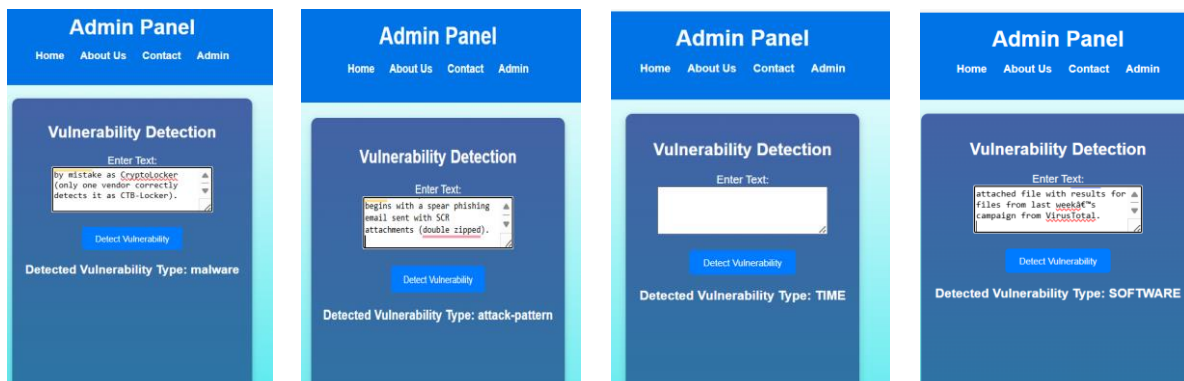


The screenshot shows the 'Admin Panel' with a navigation bar containing 'Home', 'About Us', 'Contact', and 'Admin'. The main content area is titled 'Vulnerability Detection' and features a text input field with the text: 'by mistake as CryptoLocker (only one vendor correctly detects it as CTB-Locker)'. Below the input field is a blue button labeled 'Detect Vulnerability'. The result displayed is 'Detected Vulnerability Type: malware'. At the bottom, a copyright notice reads '© 2024 AI Cloud Security. All rights reserved.'

Enter real text in the form of data related to possible vulnerabilities, e.g., logs and descriptions. Check the input and verify if the prediction result has been displayed correctly.

Backend Testing

Type any input, be it logs or possible vulnerability descriptions, in the provided text input and verify the accuracy of the prediction result. Backend testing validates smooth model integration and workflow. This includes testing the whole preprocessing pipeline to validate text cleaning and vectorization while ensuring the model predicts the correct type of vulnerability for both normal and edge-case inputs, and assessing how this system works with invalid or empty inputs to consider proper error handling.



The four screenshots show the 'Admin Panel' with the 'Vulnerability Detection' section. Each screenshot displays a different input text and its corresponding predicted vulnerability type:

- Screenshot 1:** Input: 'by mistake as CryptoLocker (only one vendor correctly detects it as CTB-Locker)'. Result: 'Detected Vulnerability Type: malware'.
- Screenshot 2:** Input: 'begins with a spear phishing email sent with SCB attachments (double zipped)'. Result: 'Detected Vulnerability Type: attack-pattern'.
- Screenshot 3:** Input: (empty). Result: 'Detected Vulnerability Type: TIME'.
- Screenshot 4:** Input: 'attached file with results for files from last week's campaign from VirusTotal'. Result: 'Detected Vulnerability Type: SOFTWARE'.

Conclusion: The web application integrates the best-trained model that is the Random Forest model successfully to provide real-time predictions of vulnerabilities based on input provided by the user in the form of text. With a robust backend and accurate predictions, the application meets the objectives of enhancing web security using machine learning.