AI-Powered Phishing URL Detection Web Application

Table of Contents

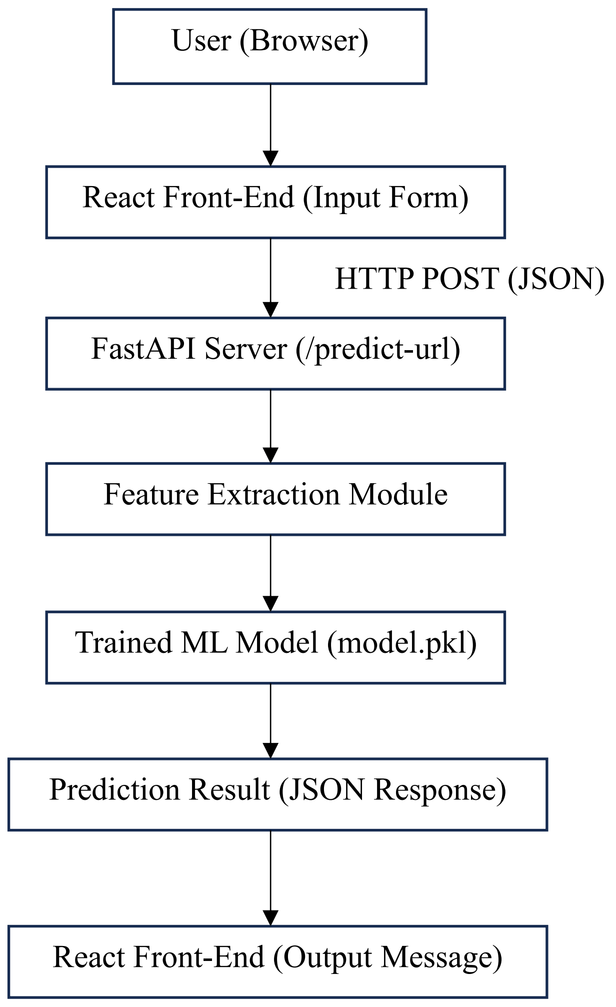
Project Description

This project is about creating a complete AI-powered phishing URL detection system by combining both frontend and backend technologies. The main aim is to build a practical cybersecurity tool that can help identify whether a website is safe to visit or potentially harmful. The system is made up of three major components a React.js frontend, FastAPI backend, and a machine learning model. When user enters URL, the backend extracts more than 85 lexical, structural, and technical features from the link including character level patterns, number of dots and hyphens, presence of suspicious tokens, domain age, redirection behaviour, subdomain structure, and more. These handcrafted features are then processed by a Random Forest classifier, which was chosen because of its ability by handle large feature sets and its strong performance in detecting complex phishing behaviors. During model training, the Random Forest achieved around 96% accuracy, proving that it can reliably distinguish between legitimate and phishing URLs .

In this project, the model is integrated directly into the FastAPI backend, allowing the system to perform predictions in real time. The FastAPI service communicates with the React front-end through REST API calls, ensuring a smooth user experience . Once the prediction complete, the result either “Phishing Website” or “Legitimate Website” is returned to the user with very fast response time. Although the frontend can display charts and deeper analytics , the focus of this submission is on achieving a working URL to prediction pipeline due to time limitations.

Overall, this project helped me understand how AI models can deployed in real applications and not just in isolated Jupyter notebooks. I also learned how frontend and backend systems interact, how to design secure API communication , and how machine learning can play major role in modern cybersecurity solutions. Developing this tool gave me hands on experience in integrating multiple technologies React, FastAPI, Python , and a trained ML model into a single functional web application capable of detecting phishing websites in real world scenarios.

System Architecture


User (Browser)
React Front-End (Input Form)
HTTP POST (JSON)
FastAPI Server (/predict-url)
Feature Extraction Module
Trained ML Model (model.pkl)
Prediction Result (JSON Response)
React Front-End (Output Message)

Components

1. React Front-End
    - Captures user URL
    - Sends payload to FastAPI via Fetch API
    - Displays prediction response
    - Includes validation for correct URL format
2. FastAPI Back-End
    - Provides health check route /health
    - Prediction endpoint /predict-url
    - Converts URL → numerical feature vector
    - Loads ML model from model.pkl
    - Returns prediction as JSON
3. ML Model
    - Trained using phishing dataset (85+ features)
    - Label:
        - 0 = Phishing
        - 1 = Legitimate

Frontend Implementation (React.js)

The frontend of the phishing detection system developed using React.js, which allowed me to build a clean, interactive, and responsive user interface. The main goal of the interface is to provide users with a simple way to test whether a URL is phishing or legitimate . React was chosen because of its component based structure, fast rendering through the virtual DOM, and ease of managing UI states such as loading messages, predictions, and form inputs. In addition to React, I used CSS to style the interface and give it modern, professional appearance with centered layouts, smooth colors, and a clearly visible input section .

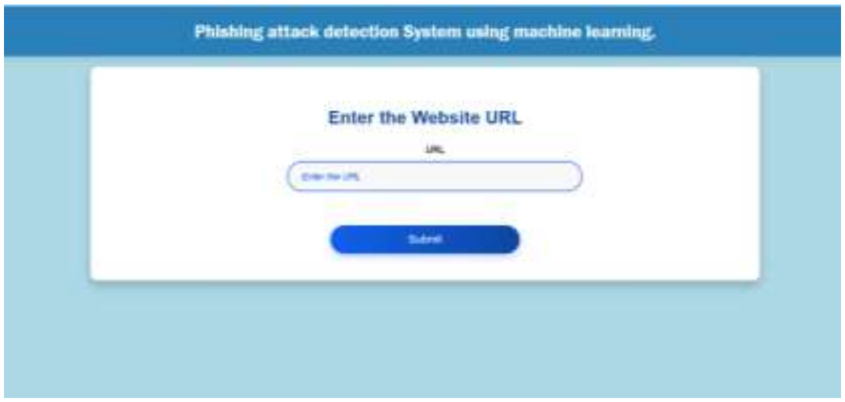
The core feature of the frontend the URL input form. This form allows users to enter website link, which then validated before submission . Basic validation ensures that the URL always begins with either http:// or https://, because machine learning models require properly formatted URLs for accurate feature extraction. If the user enters an invalid URL, error message is displayed immediately. This makes the application more user friendly and prevents unnecessary requests to the server.

Once the user submits a valid URL, the system uses the Fetch API to send the data the FastAPI backend. The URL is packaged JSON and sent through a POST request. The main logic responsible for this communication is

```
const res = await fetch("http://127.0.0.1:8000/predict-url", { method: "POST", headers: {
"Content-Type": "application/json" }, body: JSON.stringify({ url }),});
```

This code ensures that the backend receives the user's input correctly so can extract features and run the machine learning model . While the request is being processed , loading message such as "Checking…" is displayed. This gives users feedback , showing that the system is working instead of appearing frozen. When the backend responds with the prediction, the result either Phishing Website or Legitimate Website is shown clearly on the page .

The front end also handles different user feedback scenarios , including unreachable server errors . If the backend is offline or cannot be accessed, the interface displays an appropriate error message to help users understand the issue. This improves the robustness the application and avoids confusion. Overall, React front end plays essential role by ensuring smooth, clear communication between the user and the AI model through an intuitive, visually appealing interface.



Backend Implementation (FastAPI)

The backend of the phishing detection web application was built using FastAPI, modern Python framework designed high performance APIs. FastAPI was selected because it fast, lightweight, and easy to integrate with machine learning models . The server runs on Python 3 and uses additional libraries such as Joblib (for loading the trained Random Forest model), Pandas (for handling feature vectors), and tldextract, urllib, and regex for extracting URL based features. Uvicorn is used the ASGI server run the application, providing fast request handling and smooth interaction experience between the frontend and the model.

In this implementation, I created two main endpoints. The first one, /health, is simple GET method that returns {"status": "ok"}, which is useful for checking whether backend is running correctly. This helps during development and testing, especially when connecting it with the React frontend. The second endpoint, /predict-url, is the core API of the system . It accepts a POST request containing a JSON body with a URL, for example: { "url": "https://example.com/login" }. Once the URL is received, FastAPI processes it and returns a JSON response with the prediction label (Phishing or Legitimate) and the probability score .

The prediction workflow begins when the backend receives the URL from the frontend. The system first sanitizes the URL to avoid invalid or harmful inputs. After this, the backend extracts more than 85 handcrafted features from the URL, such as its length, number of special characters, count of digits, presence of suspicious keywords, and whether the domain belongs to a risky TLD. Libraries like urllib.parse, re, and tldextract help collect these features accurately. These extracted values are then arranged according to the exact feature order that the Random Forest model was trained on. This step is very important because machine learning models can only make correct predictions if the input features follow the same structure as the training dataset.

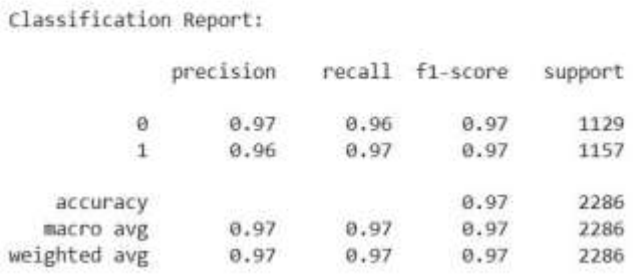
Once the feature vector is ready, it is passed into the loaded Random Forest model. The model analyzes the input and calculates the probability of URL being phishing. FastAPI then returns the output to the front-end in JSON form, such { "label": "Phishing", "probability": 0.92 }. The entire workflow is optimized to be fast and responsive, allowing users to receive predictions within seconds. Overall, the FastAPI back-end handles all data processing, feature engineering , and model computation, making it a critical part fullstack phishing detection system.

AI Model Integration

For AI model integration, I used Random Forest Classifier. This model was chosen because it performed best during evaluation, achieving high accuracy of 97%, and also because it handles complex, non-linear patterns in URL data very effectively . After training the model, I exported it model.pkl using Joblib. In FastAPI backend, the model is loaded automatically at server startup, which means the system is ready to make predictions as soon as the API begins running. This avoids repeated loading and helps improve the response time of the application .

The prediction pipeline starts with feature extraction, which is essential part of this project. Whenever user enters a URL, the backend dynamically calculates all the handcrafted features that were used during model training. These include URL length, number of dots and hyphens, symbol counts, ratio of digits, suspicious patterns, TLD based checks, hostname analysis, and several subdomain related behaviours . Since the accuracy the prediction depends heavily on feature consistency, the backend ensures that all 80+ features are generated in the exact same order as they appear training dataset. This allows the Random Forest model to understand the input correctly and produce reliable results .

Once features are prepared, they are converted into numerical vector and passed into the model. The model uses predict_proba() function to compute the probability of each class. In our dataset, the model was trained with two labels 0 for Phishing and 1 for Legitimate. This means that higher probability towards class 0 indicates phishing attempt, while higher probability towards class 1 signals safe URL . The backend returns both the prediction label and its associated probability score, which helps the user understand not only the result but also the confidence level of the model. Overall, AI model integration ensures intelligent and data driven decision making in the web application, enabling real time phishing detection based on machine learning .

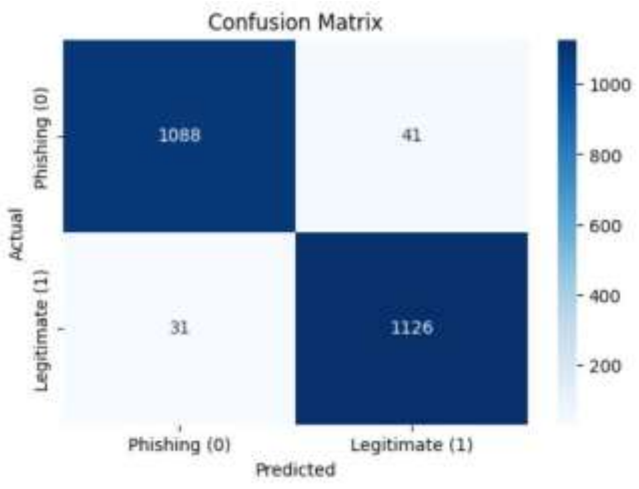


```
Classification Report:

              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1129
           1       0.96      0.97      0.97      1157

    accuracy                           0.97      2286
   macro avg       0.97      0.97      0.97      2286
weighted avg       0.97      0.97      0.97      2286
```



Conclusion

In conclusion, this project brought together multiple technologies to build fully functioning AI-powered cybersecurity tool . By integrating a React.js frontend, FastAPI backend, and Random Forest machine learning model, the system provides users with simple, fast, and effective way to evaluate whether a given URL is safe or potentially malicious. The web interface allows users to submit URLs easil , while the backend ha ndles all processing tasks, including advanced feature extraction and AI based prediction .

Throughout the development process, I able to apply several concepts learned during the course, such as API communication, model deployment, and full-stack integration . The project demonstrates how real time interaction between the frontend and the backend can be achieved through clean JSON based communication. It also highlights the importance of validation, exception handling, and secure preprocessing when dealing with potentially harmful user input such as phishing URLs .

The AI model worked effectively in predicting phishing behavior , mainly because it trained using large feature set and achieved strong accuracy score. This project also showed how feature engineering plays major role in cybersecurity, especially when dealing with phishing attacks where small URL patterns can indicate fraudulent intent. By applying these features during runtime, the system becomes capable of analyzing suspicious URLs in a smart and automated way.

Overall, this project helped me gain practical experience in deploying machine learning models within a real web application environment . It also strengthened my understanding of how AI can be used to solve cybersecurity problems in real life scenarios . The completed system provides solid foundation more advanced phishing detection tools, and in the future, it could be expanded with additional features such as visual URL analysis, browser extensions, or deeper network level threat detection.

References

- Mohammad, R., Thabtah, F. & McCluskey, L. (2015). *Predicting phishing websites using classification models.* Applied Computing and Informatics.
- FastAPI Documentation. (2024). *https://fastapi.tiangolo.com/*
- React Documentation. (2024). *https://react.dev/*
- Scikit-Learn Documentation. (2024). *https://scikit-learn.org/*
- TLDExtract Library. (2024). *https://github.com/john-kurkowski/tldextract*