PhishShield : Real-time Phishing URL Detector

Phishing : Phishing is a type of cyberattack where criminals try to trick as into revealing sensitive information like passwords, credit card details, or personal data. Legitimate refers to something genuine, authentic, and trustworthy, like a reputable company or a safe website. Phishing attacks exploit this trust by mimicking legitimate entities to steal your information.

HERE ARE THE LOSSES CAUSED BY PHISHING WEBSITES AND URLS

1. **Financial loss:** Victims may lose money directly from their bank accounts or through fraudulent transactions initiated by attackers.

2. Theft of sensitive information: Phishing sites capture personal information, logins, passwords, and credit card data, which criminals can use for identity theft.

3. Data breaches: Organizations experience unauthorized access to confidential data, leading to further losses. Loss of trust: Customers lose confidence in brands or services that have been targeted, harming business reputation.

4. Malware infection: Phishing URLs can distribute malware that damages devices or steals more information. Legal and compliance issues: Companies may face penalties or lawsuits if phishing leads to privacy violations.

5. Personal data : in using phishing website we lost our personal data like image , vedion , email , password and so many thing

OUR PROJECT ABOUT

❑ Project Overview :

❖ we are developed a browser extension designed to enhance online security.

❑What the Extension Does:

❖ When a user clicks on any URL, the extension instantly checks the link.

❖ It analyzes the URL for signs of phishing or malicious activity. The extension then provides a numeric safety score for the website.

❖ It also displays a color-coded alert (such as green, yellow, or red) to visually indicate the risk level.

1—25	Safe
25—50	Intermediate
50—80	Risky
80—100	Danger

All the process step by step for our project

(Backend + Frontend Flow)

1.Model Training (ML – Backend)

- Collect phishing + safe URLs dataset
- Extract lexical & host features (length, digits, subdomains, etc.)
- Train ML model (Scikit-learn: RandomForest/XGBoost)
- Save trained model

2.Backend (Flask API)

- Loads trained ML model
- Receives URL from extension
- Processes features → Predicts risk score (0–100)
- Optionally checks VirusTotal for extra validation
- Sends JSON response:

3.Frontend (Browser Extension)

- Captures current tab's URL
- Calls Flask API ..
- Shows **badge color** (Green/Yellow/Red)
- If Red (High Risk) → Display **block page** with warning

Machine Learning Model for URL Detection

- Dataset : collect dataset of Phishing URL s and Legitimate URLs

- IMPORT ALL IMPORTANT LAIBRARY

- Import pandas for read our csv file
- **NumPy:** is used for efficient numerical operations
- **Pandas:** provides data structures like DataFrames for easy data cleaning, manipulation, and analysis.
- **Matplotlib's and seaborn :**module is used to create static, interactive, and animated visualizations
- **Urlparse:** helps break down URLs into components (e.g., scheme, netloc, path), useful for web data analysis.
- **Re:** is Python's built-in module for working with regular expressions
- **LabelEncoder** converts categorical text data into numbers for machine learning algorithms
- **SMOTE (Synthetic Minority Over-sampling Technique)** is used to balance class distribution
- **train_test_split** splits data into training and testing sets
- **RandomizedSearchCV** finds optimal hyperparameters for models through randomized search.
- **DecisionTreeClassifier** creates a tree-based model for classification tasks.
- **RandomForestClassifier** is an ensemble learning method combining many decision trees for improved predictive performance.
- **accuracy_score** measures how often predictions match actual values.
- **joblib** is used for saving and loading Python objects efficiently, such as trained machine learning models.

- **Extract feature :** Raw **URLs** are just text strings → Machine Learning models can't use them directly.
- We need to **convert URLs into numeric features** (like length, number of digits, suspicious words, e
- Read our csv for performing task in csv fille
- Find null value and fill them by fillna function
- The we Converts the list of extracted feature dictionaries into a new dataframe called **X**.
- y is our **target vector** (the answer column for training the model)
- Then we use ploting for our data to clear visualization and analysis
- Then we use boxplot forn visualize the spread, central tendency, and variability of a column.
- The we use **DecisionTreeClassifier** , Simple, fast, easy to interpret ,Good baseline. ,But: can easily **overfit** on training data
- **RandomForestClassifier :** More accurate & stable than a single tree , Handles noisy data better.
- **XGBClassifier :** Often achieves **higher accuracy** than RandomForest , Handles imbalanced datasets well , But more
- complex and slower than RandomForest.
- Train_test_ split : then we split our data into train and test data using smote laibrary
- **Hyperparameter :** Because the default parameters of RandomForest may not give the **best accuracy**.
  So we search for the **best combination of parameters** automatically instead of guessing.
- Accuracy is calculated to measure the overall correctness of predictions.
- A classification report gives precision, recall, and F1-score for each class.
- The confusion matrix shows counts of correct and incorrect predictions

❑ Save our model

- ❖ Our File "url_phishshield.joblib" will be used in your API, web app, or extension
- ❖ Our File “**label_encoder.joblib"** Saves the **label encoder** (e.g., mapping phishing → 1, legitimate → 0
- ❖ Our File **"feature_names.joblib“** Saves the list of **feature names** used during training.

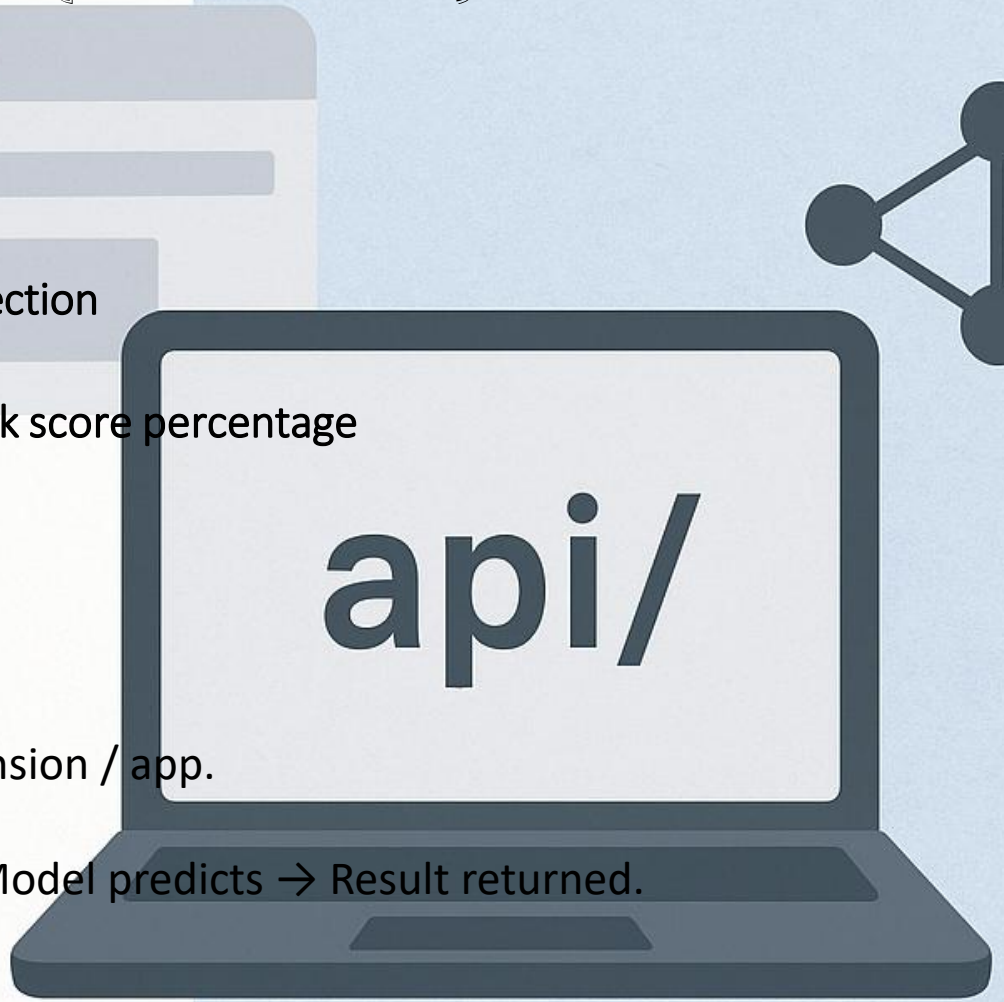
❑ Load and predict

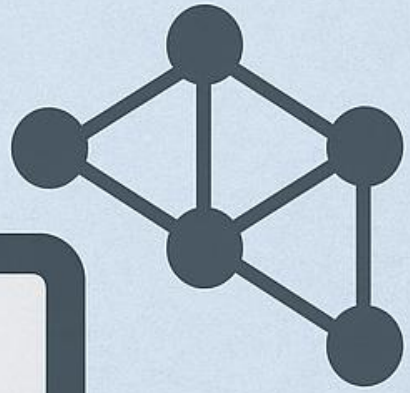
- ❖ First we load our all joblib file
- ❖ Then we give inpurt URL,s manualy
- ❖ Then our matchine lernign model predict the result

BACKEND (FLASK API)

❑ Basic overview

- ❖ **Filename:** app.py
- ❖ **Purpose:** Flask Web API for phishing URL detection
- ❖ **Input:** A URL ({"url": "http://example.com"})
- ❖ **Output**: Prediction (legitimate or phishing) + risk score percentage





❑ We use Flask API for :

- ❖ **Bridge between ML model & frontend**
  → Connects trained model with browser extension / app.
- ❖ **Serves predictions via HTTP**
  → Example: Extension sends URL → Flask → Model predicts → Result returned.
- ❖ **Lightweight & easy**
  → Simple Python framework, fast to set up.
- ❖ **Flexible deployment**
  → Can run locally (localhost) or on cloud (Heroku, AWS, etc.).

❑ Laibrary explanation

- **Flask:** Creates the web application
- **request:** Reads incoming data
- **jsonify:** Returns response in JSON format
- **CORS:** Allows cross-domain usage (e.g., frontend React app calls API)
- **os:** Handles paths (cross-platform)
- **joblib:** Loads ML model and encoders
- **pandas:** For tabular data handling
- **urlparse:** Extracts components of input URL (domain, scheme, etc.)

❑ Load our all joblib file

- Loads the trained ML model + encoder + feature set only when needed.
- Uses global variables so objects don’t load multiple times.
- Prevents failures if NumPy/Scikit-learn versions differ (lazy loading)

❑ Using featutre extract

- Converts URL into numbers ML can process:
  - Length, dots, hyphens, digits
  - HTTPS used or not
  - Subdomain count
  - Presence of suspicious keywords like "login", "bank", "paypal"
- Output: Dictionary of numeric features.

❑ Model Prediction

- ❖ If model supports probabilities → use them.
- ❖ Else → try decision_function → convert to probability with logistic sigmoid.
- ❖ Else → fallback to binary prediction

❑ Class Handling

- ❖ If model has "phishing" class → map directly.
- ❖ If "legit" found → assume other is phishing.
- ❖ Else fallback: last class = phishing.

❑ Final Output

- ❖ Probability of phishing → convert to risk %
- ❖ Human-readable predicted label (legit vs. phishing)
- ❖ Returns final JSON result with:
  - URL
  - Classes
  - Probability + Risk %
  - Prediction label
- ❖ Extracted features

❑ Run server

- ❖ Runs the Flask API server at http://127.0.0.1:5000.
- ❖ debug=False → no auto reload / safer for production

FRONTEND + EXTENSION

❑ Purpose of the Program

- This is a **Chrome Extension background script.**
- It checks if the currently opened website is **safe or risky** using a local API (http://127.0.0.1:5000/predict).
- Helps protect users from **phishing or unsafe sites.**

❑ How It Works

❖ Monitors Active Tab

- Detects when you open or switch to a new website.
- Automatically scans the site's URL.

❖ Connects to Local Server

- Sends the website URL to a local server (/predict API).
- The server replies with a **risk percentage** (0–100%).

- **Shows Risk Level**
  - Displays the risk as a **badge number on the extension icon**.
  - Uses different colors:
    - 🟢 Green (safe)
    - 🟡 Yellow (medium)
    - 🟠 Orange (risky)
    - 🔴 Red (dangerous)

- **Blocks Dangerous Sites**

  - If the risk is **80% or higher**, it automatically blocks the page and redirects to a **“blocked” warning page**.

- **Popup Support**

  - Saves scan results for quick display in the popup interface.
  - Users can manually trigger scans.

❑ Overview of manifest.json

- **Manifest Version:** 3 (latest Chrome Extension format).
- **Name & Version:** "PhishShield URL Rater", version 1.0.0.
- **Description:** Explains that the extension **rates the current page's phishing risk (1–100)** and shows it with badge colors.

❑ Key Parts

- **action**
  - Defines popup (popup.html) and title shown when you click the extension.
- **permissions**
  - "activeTab", "tabs", "storage" → lets extension read current tab, manage tabs, and save temporary data.
- **host_permissions**
  - Allows the extension to send requests to the **Flask backend** (http://127.0.0.1:5000/ and http://localhost:5000/).
- **background**
  - Runs background.js as a **service worker** to monitor tabs and do phishing checks.
- **web_accessible_resources**
  - Makes blocked.html accessible so it can be shown when a dangerous site is blocked.

❑ Purpose of popup.html

- ❖ Provides the **user interface** of the Chrome extension.
- ❖ Lets the user **see the phishing risk score** for the current site.
- ❖ Allows the user to **manually scan** the site.

❑ What We Do in popup.html

❖ Score Display

- ➢ Shows a big circular badge (-- → updated with risk %).
- ➢ Color changes based on risk level (green, yellow, orange, red).

❖ Website Info

- ➢ Displays the current site's **URL**.
- ➢ Shows the **predicted label** (e.g., safe / phishing).

❖ Scan Button

- ❖ Button "Scan this site" → sends a message to background.js → triggers a new scan via Flask server.

❖ Legend / Guide

- ❖ Explains what each score range means (0–24 safe, 25–49 caution, 50–74 risky, 75–100 phishing).

❖ Hint / Status

- ❖ A small text area to show additional messages (like errors or tips).

❑ Overview of popup.js

- ❖ Handles the **logic** behind the popup interface (popup.html).
- ❖ Connects the **UI** with the **background script** and **Flask server.**

❑ What It Does

1.Color Mapping

- •Chooses the badge color (green, yellow, orange, red) based on phishing risk score.

2.UI Updates

- •Fills in the popup with:
  - •Current website URL
  - •Risk percentage
  - •Prediction label (Safe / Phishing)

3.Fetch Results

- •Reads the cached scan results from background.js.
- •Shows tips if the Flask server is not running.

4.Manual Scan

- •On button click "Scan this site" → sends a request to re-scan current tab.
- •Updates UI with new results.

5.Initialization

- •Runs automatically when the popup opens (DOMContentLoaded)

Overview of blocked.html

- Acts as the **warning page** when a website is too risky (≥ 80% phishing score).
- Replaces the dangerous website to **protect the user**.

What It Does

Warning Message

Shows a big alert: “ ⚠️ Blocked for Your Safety”.

Explains the site was flagged as a **phishing risk**.

Design

Full-page centered card with dark background.

Red highlight for danger message.

Clear and user-friendly layout.

Action Button

Provides a **“Go Back”** button → takes user back to previous safe page.