gephiltephish: A Crowdsourced Phishing Detection Platform

Dan Vaccaro

Capstone Design Document

Project Overview

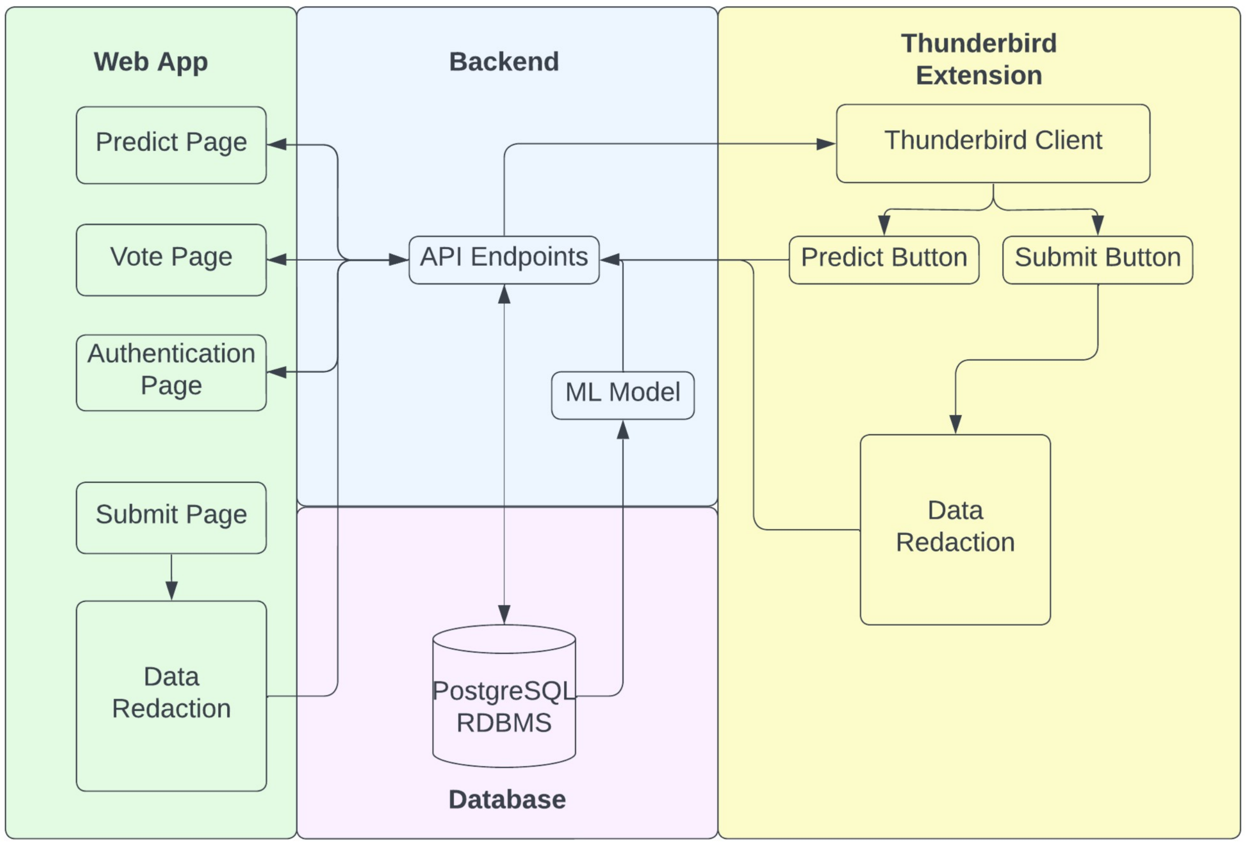
gephiltephish is a web application and Thunderbird extension designed to crowdsource phishing email detection. Users can submit emails via a web form or directly through the Thunderbird extension. The application processes and stores these emails, allowing the community to vote on whether they are phishing or legitimate. The data collected is used to train a machine learning model to improve phishing detection over time.

System Architecture

There are 4 major components of the system.

1. A web application, which provides interfaces for user authentication, submission, voting, and prediction. Any submissions are stripped of Personally Identifiable Information (PII) using NLP methods before submitting to the backend.
2. A Thunderbird extension, which provides an interface for direct submission and prediction from the email client. It is also able to strip PII before transit.
3. A backend Python application, which provides API endpoints for registration, authentication, submission, voting, and prediction. It also handles periodically training the model and managing the database.
4. A PostgreSQL database to store users, emails, and votes.

High-level Architecture Diagram



Component Design

A. Web Application

The web application serves as the user interface for gephiltephish, providing users with a platform to submit emails, vote on the legitimacy of emails submitted by others, and view the results of the analysis. The web application is built using React, with the following components:

1. **Login/Registration Form**. Users must be registered and authenticated to submit and vote on emails. This form will make calls to the /register and /login API endpoints.
2. **Submission Form.** The submission form is the entry point for users to submit suspected phishing emails. The form is designed to be simple, requiring only the email sender and body content to be pasted into a text area. On submission, the content is validated to ensure it is not empty, then undergoes a data redaction process before being sent to the /submit endpoint.
3. **Voting Interface.** The voting interface displays a list of emails submitted by other users, allowing the community to vote on whether each email is phishing or legitimate. The interface is designed to be intuitive, with clear buttons for casting votes. The votes are then sent to the backend for aggregation and analysis via the /vote endpoint.
4. **Prediction Form**. The prediction form provides an interface for users to test the model on a given email. As with the submission form, the user pastes the sender and body content, which undergoes data redaction and is then sent to the /predict endpoint. The model prediction is then displayed to the user.

B. Thunderbird Extension

The Thunderbird extension allows users to submit and evaluate emails directly from their email client. Built using the WebExtensions API, the extension integrates with Thunderbird to add functionality for phishing detection without requiring users to leave their email client. The extension consists of the following components:

1. **Email Submission Button:** This button is added to the Thunderbird UI, allowing users to quickly submit the currently selected email to gephiltephish. When clicked, the email content is processed, redacted, and sent to the backend for analysis via the /submit endpoint. The session must have a login token stored for the user to be able to submit.
2. **Prediction Button:** The prediction button enables users to get an immediate assessment of whether the selected email is likely to be a phishing attempt by sending the content to the /predict endpoint.
3. **Email Processing:** Before any data is sent to the backend, the Thunderbird extension processes the email to redact sensitive information to ensure that user privacy is maintained while still allowing for effective analysis.

C. Backend

The backend is responsible for handling API requests, managing data storage, and integrating the machine learning model. Developed using Python using the Django framework, the backend will be designed for robustness and scalability. It consists of the following components:

1. **API Endpoints:** The backend provides various API endpoints for different functionalities, including login/registration, email submission, voting, and predictions. APIs can be implemented using Django REST Framework.

2. **Machine Learning Model:** The machine learning model is central to gephiltephish's ability to detect phishing emails. The model is trained on the data collected from user submissions and votes, allowing it to improve over time. The backend handles both the training process and the serving of predictions through API endpoints.
    - While a basic simple bag-of-words model has been constructed from a CountVectorizer → LogisticRegression pipeline from an existing dataset, there is a desire to experiment with more complex models such as transformers using Hugging Face's transformers library.
    - Natural language processing tasks such as tokenization will be handled using spaCy.
3. **Domain Analysis:** As part of the email processing, the backend performs WHOIS lookups and domain reputation checks. These checks help assess the legitimacy of the email's sender domain, contributing to the overall phishing detection.

D. Database

The PostgreSQL database houses all user data, emails, and domain analysis results. It stores the following schema:

1. **Users:**
    - id: Primary Key, uniquely identifies each user.
    - email: Text, Stores the user's email address, which is used for authentication and communication.
    - password_hash: String, stores a salted and hashed version of the user's password.
    - created_at: Timestamp of when the user registered.
2. **Emails:**
    - id: Primary Key, uniquely identifies each submitted email.
    - user_id: Foreign Key, links the email to the submitting user.
    - content: Text, The redacted text content of the email.
    - created_at: Timestamp of when the email was submitted.
    - votes_phishing: Integer, the number of votes that the email is a phishing attempt.
    - votes_legitimate: Integer, the number of votes that the email is legitimate.
3. **Domain_Analyses:**
    - id: Primary Key, uniquely identifies each domain analysis entry.
    - email_id: Foreign Key, links the analysis to the associated email.
    - domain: Text, stores the domain name being analyzed.
    - registration_date: Date, indicates when the domain was registered.
    - reputation_score: Integer, reflects the domain's reputation based on third-party checks from VirusTotal.
    - created_at: Timestamp of when the analysis was performed.

Deployment and Management

The platform will need to be adaptable and scalable. The backend will be deployed as a Docker container to maintain a consistent environment. It will be hosted on AWS, which has features to automatically handle scaling, load balancing, and key management. The database can be run as a PostgreSQL instance on AWS, as well. Version control will be handled with git.

Testing Strategy

Each individual component will need to be unit tested to ensure it works in isolation, and then integrated testing between different components will need to be developed to ensure the web application, extension, backend, and database all interact seamlessly. This will involve identifying edge cases and developing a suite of tests to be run before each deployment to ensure bugs don’t recur. Static analysis tools such as SonarQube can also be utilized to detect bugs in code before deployment.

Security and Privacy

It is essential to maintain trust between the platform and its community for gephiltephish to work as intended. Since users are submitting their own personal emails, it is crucial to ensure all emails are stripped of PII before being sent to the model for training. Therefore, we need to make sure all data redaction takes place on the client side, and only post-processed text is sent over the network. In addition, all communication between the client and server will be encrypted by default. User authentication will be secured with salted and hashed passwords and the use of session management. The database will be protected from SQL injection attacks thanks to Django’s built-in use of prepared statements.

Stretch Goals

The plan for the platform thus far is comprehensive, but there are some additional considerations to make down the line. Ultimately, the program aims to solve a problem that is currently profitable for bad actors. While these actors may attempt to take down the platform via DDOS or other common attacks, these can be managed. More concerning would be an attempt to corrupt the platform by feeding it bad data and/or generating fake votes.

One idea for handling this is to make it profitable to be a good actor. The economics and game theory behind developing such a system are beyond the current scope while we work on developing the base platform, but one simple idea is to develop a web3 wallet integration to require a user to maintain a small stake to vote, and to reward good users with some yield built off of that stake. There are some low-fee, low-latency networks that may work for this such as Flare, which offers built-in yield mechanisms.

The development of this rewards system will take place after all other components are built out and tested extensively. It is not considered essential to the platform’s base functionality, but would be an interesting feature to tack on and use to attract users.

Target Dates for Deliverables

Week 1
Target: Sept. 13

- Set up project structure: Create initial repository with version control, set up Docker and AWS environments.
- Define database schema in PostgreSQL.
- Begin setting up the Django backend with basic API endpoints for login/registration.

Week 2
Target: Sept. 20

- Complete the core backend API endpoints for user registration, authentication, and email submission.
- Set up the PostgreSQL instance on AWS and connect it with the Django backend.

• Create unit tests for the login/registration API endpoints.

Week 3
Target: Sept. 27

• Start building the Thunderbird extension (email submission button and prediction button).
• Develop email processing and redaction functionality for the extension.
• Ensure proper integration of the extension with the backend for email submission and predictions.

Week 4
Target: Oct. 4

• Continue developing the Thunderbird extension, focusing on refining the user interface.
• Finalize email redaction and security processes (data redaction before transmission).
• Test submission and prediction functionality within Thunderbird.

Week 5
Target: Oct. 11

• Complete the Thunderbird extension development, ensuring all features (submission, prediction, redaction) work seamlessly.
• Set up integration between the Thunderbird extension and the backend for user login and token management.
• Conduct thorough end-to-end testing for the extension.

Week 6
Target: Oct. 18

• Begin developing the web application's login/registration forms (React).
• Create the basic submission form for the web app that interacts with the backend.
• Start implementing the voting interface in the web frontend.

Week 7
Target: Oct. 25

• Complete the frontend submission form with validation.
• Finalize the voting interface on the frontend (React) with backend integration.
• Set up the prediction form on the web app for testing emails.

Week 8
Target: Nov. 1

• Integrate the existing basic machine learning model (CountVectorizer → Logistic Regression) into the backend for predictions.
• Experiment with more complex models (e.g., transformers) using Hugging Face.
• Ensure both the web app and Thunderbird extension are interacting correctly with the machine learning model.

Week 9
Target: Nov. 8

- Conduct load testing and optimize the backend for scalability using AWS features (scaling, load balancing).
- Finalize email redaction methods and domain reputation analysis.
- Start writing unit tests for both the backend and Thunderbird extension.

Week 10
Target: Nov. 15

- Finalize all testing (unit, integration, security) for both the web application and the Thunderbird extension.
- Deploy the web app and Thunderbird extension to production on AWS.
- Begin final QA testing to identify any last-minute bugs or issues.

Week 11
Target: Nov. 22

- Final bug fixes and optimizations based on QA testing.
- Document the system architecture, components, and deployment process.
- Prepare for final submission and presentation.

Week 12
Target: Nov. 29

- Use this week to catch up on any remaining tasks or work on stretch goals.
- Handle unexpected issues, polish final details, or optimize performance.
- Conduct additional QA testing or final walkthroughs to ensure everything is working smoothly.