Developing AI-Driven Solutions for Password Fatigue and Phishing Vulnerability

Objective

To demonstrate the development and implementation of AI-driven solutions to combat password fatigue and phishing vulnerabilities at TechEdu Solutions: This project emphasizes deploying a sophisticated AI-powered password manager and a highly effective AI-based phishing detection system. The primary goal is to mitigate the risks associated with password management and phishing attacks by leveraging advanced AI technologies. This integration aims to enhance user security and streamline interaction on the platform, ensuring a safer and more reliable digital experience for all users.

Problem Statement:

At TechEdu Solutions, increasing incidents of password fatigue among students and staff have led to security vulnerabilities, particularly concerning phishing attacks. These incidents often arise from the repetitive need to remember and manage numerous complex passwords across different educational tools and platforms.

Solution: To tackle these issues, TechEdu Solutions introduces an AI-powered password manager and phishing detection system. This approach automates secure password management and enhances phishing threat detection, improving security and user convenience

Tasks

The following tasks outline the process of developing an AI-driven solution to address password fatigue and phishing vulnerabilities for TechEdu Solutions:

Task 1. Develop an AI Solution for Password Management and Phishing Detection:

To address the issue of password fatigue and phishing attacks at TechEdu Solutions, we can design an AI-powered Password Management and Phishing Detection System that automates and enhances password management while improving security through real-time phishing detection.

Here's a high-level design and steps to implement such a solution:

Solution Overview

The solution will consist of two primary components:

1. **Password Management System:**
    - Automate the generation and storage of secure passwords.
    - Implement a password vault that encrypts and stores passwords securely.
    - Provide automatic password filling across various platforms.
2. **Phishing Detection System:**
    - Integrate real-time phishing detection to analyse email content and URLs.
    - Use AI/ML to identify phishing attempts based on patterns and features from emails and websites.
    - Alert users of potential phishing attacks.

Detailed Breakdown

1. AI-Powered Password Management System

Features:

- Password Generation: Secure, unique passwords are automatically generated for every platform used.
- Password Autofill: Integrate with browsers and platforms to auto-fill login credentials securely.
- Secure Storage: Use AES encryption to store passwords locally or in a secure cloud-based vault.
- Password Strength Checking: Alert users if passwords are weak or reused.

Steps:

1. Password Generation: Use randomization techniques with special characters, digits, and uppercase letters to create strong passwords.
2. Secure Storage: Store encrypted passwords in a vault (local or cloud). Use a hashing algorithm (e.g., bcrypt) to store the vault's master password.
3. Auto-fill: Integrate with browser extension APIs (e.g., Chrome extension API) to auto-fill the login forms.
4. Password Strength: Implement a strength checker that evaluates passwords based on length, complexity, and uniqueness.

2. AI-Based Phishing Detection

Features:

- Email Phishing Detection: Use machine learning to detect phishing attempts based on email body, subject line, and metadata.
- URL Analysis: Real-time URL analysis to flag suspicious websites.
- User Alerts: Notify users in case of potential phishing threats.

Steps:

1. Email Analysis: Train a model on email data (phishing vs. legitimate) to classify incoming emails as phishing or legitimate. The features used can include the presence of certain keywords (e.g., "urgent", "account suspension"), email sender information, and common phishing tactics.
2. URL Analysis: For links in emails, analyze the URL for suspicious characteristics (domain names, SSL/TLS certificate validity, etc.).
3. Phishing Detection Model:
    - Train a machine learning model using a dataset of phishing and non-phishing emails. Features might include subject line, email content, presence of attachments, and URL patterns.
4. Real-Time Detection: Continuously monitor incoming emails and alert the user when a phishing attempt is detected.

Tech Stack and Tools

1. **Backend:**
    - Flask/Django for web services (if building an integrated web solution).
    - Python for the backend logic and model training.

- scikit-learn for building the phishing detection models.
- cryptography for password encryption.

2. **AI and Machine Learning:**
   - Natural Language Processing (NLP): For analysing email content and detecting phishing attempts.
   - Logistic Regression / Random Forest / SVM for classification of phishing vs legitimate emails.
   - TensorFlow / PyTorch for building more advanced models if needed (e.g., neural networks).
3. **Frontend / Browser Extension:**
   - HTML/CSS/JavaScript for creating the frontend interface (if web-based).
   - Chrome Extensions API: For integrating password management and phishing detection directly into the browser.
4. **Database:**
   - SQLite (local) or PostgreSQL/MySQL (cloud-based) for storing passwords and user data securely.
   - Redis for caching real-time alerts or logs.

Solution Implementation:

1. Building the Password Management System

Generate Strong Passwords:

python

Copy

```
import random
import string

def generate_strong_password(length=16):
    """Generate a random, strong password"""
    characters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(characters) for i in range(length))
```

```
# Example usage
password = generate_strong_password()
print("Generated Password:", password)
```

Store and Retrieve Passwords (Encrypted):

python

Copy

```
from cryptography.fernet import Fernet

# Function to generate encryption key
def generate_key():
    return Fernet.generate_key()

# Encrypting password
def encrypt_password(password, key):
    f = Fernet(key)
    encrypted_password = f.encrypt(password.encode())
    return encrypted_password

# Decrypting password
def decrypt_password(encrypted_password, key):
    f = Fernet(key)
    decrypted_password = f.decrypt(encrypted_password).decode()
    return decrypted_password

# Example usage
key = generate_key()  # Store this key securely (use a secure vault)
encrypted = encrypt_password(password, key)
decrypted = decrypt_password(encrypted, key)
```

```
print("Encrypted Password:", encrypted)
print("Decrypted Password:", decrypted)
```

2. Phishing Detection System

Train Machine Learning Model for Phishing Detection:

python

Copy

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import joblib

# Load phishing email dataset
df = pd.read_csv('phishing_emails.csv')

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['email_body'])
y = df['label']  # 1 for phishing, 0 for legitimate

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
```

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# Save the model and vectorizer
joblib.dump(model, 'phishing_model.pkl')
joblib.dump(vectorizer, 'vectorizer.pkl')
```

Predict Phishing in Real-time:

python

Copy

```
import joblib

# Load the trained model and vectorizer
model = joblib.load('phishing_model.pkl')
vectorizer = joblib.load('vectorizer.pkl')

def predict_phishing(email_body):
    email_vector = vectorizer.transform([email_body])
    prediction = model.predict(email_vector)
    return 'Phishing' if prediction == 1 else 'Legitimate'

# Example usage
email = "Dear user, click the link to reset your password: http://phishingsite.com"
print(predict_phishing(email))
```

3. Integration with Browser (Chrome Extension)

- Chrome Extension will allow users to automatically fill passwords, generate new ones, and alert them of phishing emails in real-time.
- Use the Chrome chrome.identity API to securely handle user login.
- Use chrome.storage to store and retrieve passwords locally in the extension.

- Phishing Detection: The extension can send email contents to the backend for phishing analysis, or integrate real-time phishing detection into the frontend.

4. Future Enhancements

1. Multi-Factor Authentication (MFA): Integrate MFA for added security when logging in.
2. AI-Powered Threat Analysis: Integrate machine learning models that detect suspicious activity in email or web traffic patterns.
3. Cross-Platform Sync: Allow users to access their passwords and phishing alerts across multiple devices securely.

Task 2. Generate a Code for a Phishing Email Detection System Using GenAI

Below is a step-by-step guide to create a **Phishing Email Detection System** in **VS Code**. This system will use **machine learning** techniques to classify emails as **phishing** or **legitimate** based on their content. The process includes data preparation, text cleaning, feature extraction, model training, and prediction.

Steps to Create a Phishing Email Detection System:

1. Set up your project structure

Create a new folder for your project, e.g., PhishingEmailDetection, and open it in **VS Code**. Inside this folder, create the following structure: PhishingEmailDetection/

lua

|-- data/

| |-- phishing_emails.csv # Sample dataset (phishing and non-phishing emails)

|-- models/

| |-- email_model.pkl # Saved model after training

|-- requirements.txt

|-- app.py

|-- utils.py

|-- preprocess.py

2. Install Dependencies

You'll need the following Python libraries:

- **pandas**: To load and manipulate the email data.
- **nltk**: To clean and process the text.
- **scikit-learn**: For machine learning models.
- **joblib**: To save and load the model.
- **beautifulsoup4**: For parsing HTML content (if any).

Create a requirements.txt file with the following contents:

txt

Copy

pandas

nltk

scikit-learn

joblib

beautifulsoup4

Install the dependencies by running:

bash

Copy

pip install -r requirements.txt

3. Preprocess Email Data

You need to load the email dataset and preprocess the text (cleaning, tokenization, etc.). We'll assume that the dataset contains two columns: email_body (the text of the email) and label (1 for phishing, 0 for legitimate).

Create a file called **preprocess.py** for this:

python

Copy

```
# preprocess.py


import pandas as pd

import re

import nltk

from bs4 import BeautifulSoup

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize


nltk.download('stopwords')

nltk.download('punkt')


# Function to clean email text

def clean_email_text(text):

    # Remove HTML tags

    soup = BeautifulSoup(text, "html.parser")      text = soup.get_text()
```

```
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)

    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Convert text to lowercase
    text = text.lower()

    # Tokenize and remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)
    filtered_tokens = [word for word in tokens if word not in stop_words]

    return ' '.join(filtered_tokens)

# Function to load and preprocess the dataset
def load_and_preprocess_data(file_path):
    df = pd.read_csv(file_path)
    df['cleaned_body'] = df['email_body'].apply(clean_email_text)
    return df
```

4. Train a Machine Learning Model

The next step is to train a machine learning model using the cleaned text data. We'll use **Logistic Regression** (but you can experiment with other models like Random Forest, Naive Bayes, etc.).

Create a file called **app.py**:

python

Copy

```
# app.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.externals import joblib
from preprocess import load_and_preprocess_data

# Load and preprocess the data
df = load_and_preprocess_data('data/phishing_emails.csv')

# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['cleaned_body'])
y = df['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict the test set
y_pred = model.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

```
# Save the model and vectorizer to disk
joblib.dump(model, 'models/email_model.pkl')
joblib.dump(vectorizer, 'models/vectorizer.pkl')
```

5. Load the Model and Make Predictions

Once the model is trained and saved, we can load it to make predictions on new, unseen email data. Create a **utils.py** file to handle the prediction:

python

Copy

```
# utils.py

import joblib
import pandas as pd
from preprocess import clean_email_text

# Load the saved model and vectorizer

model = joblib.load('models/email_model.pkl')
vectorizer = joblib.load('models/vectorizer.pkl')

# Function to make predictions on a new email

def predict_email_phishing(email_body):
    cleaned_email = clean_email_text(email_body)
    email_vector = vectorizer.transform([cleaned_email])
    prediction = model.predict(email_vector)

    return 'Phishing' if prediction[0] == 1 else 'Legitimate'

# Test the function
```

```
if __name__ == "__main__":
    email = "Click here to claim your reward! http://phishingsite.com"
    print(predict_email_phishing(email))
```

6. Running the System

- First, make sure you have your **email dataset** (phishing_emails.csv) in the data/ folder. The dataset should have at least two columns: email_body (email content) and label (1 for phishing, 0 for legitimate).
- Run the model training script:

bash

Copy

```
python app.py
```

This will train the model, evaluate its performance, and save the trained model and vectorizer.

- After training, you can use the predict_email_phishing function in **utils.py** to classify new emails. You can also build a simple user interface (UI) for users to input an email and get a prediction.

Final Structure of the Project Folder

Lua

```
PhishingEmailDetection/
|-- data/
|   |-- phishing_emails.csv    # Email dataset (phishing vs legitimate)
|-- models/
|   |-- email_model.pkl        # Trained machine learning model
|   |-- vectorizer.pkl         # Trained vectorizer for text transformation
|-- requirements.txt           # List of dependencies
|-- app.py                     # Script for training the model
|-- utils.py                   # Functions for making predictions
|-- preprocess.py              # Functions for cleaning and preprocessing the data
```

Explanation

- **Data Preprocessing** (preprocess.py): Cleans the email body text by removing HTML tags, URLs, non-alphanumeric characters, and stopwords.
- **Model Training** (app.py): Trains a machine learning model using TF-IDF features and a Logistic Regression classifier to detect phishing emails.
- **Prediction** (utils.py): Loads the trained model and vectorizer, and predicts whether a new email is phishing or legitimate.

Task 3. Generate a Chrome Extension for Password Management Using GenAI

Creating a **Chrome Extension** for **Password Management** that tackles the issues of **password fatigue** and **phishing** is an excellent solution for TechEdu Solutions. The extension would automate secure password management, generate strong passwords, and provide protection against phishing threats, making it easier for users to manage multiple accounts while ensuring security.

Here's an overview and step-by-step guide to implementing this Chrome extension.

Overview of the Chrome Extension

1. **Password Management:**
    - **Password Generation**: Generate strong and unique passwords for each website or platform.
    - **Autofill**: Autofill login credentials for websites automatically to reduce the need for manual input.
    - **Password Storage**: Securely store passwords and keep them synchronized across devices.
    - **Password Strength Check**: Check the strength of passwords and alert users if a password is weak or reused.
2. **Phishing Protection:**
    - **Phishing Detection**: Analyze URLs and website content to detect phishing attempts.
    - **Alert System**: Notify users if they visit a site that is potentially a phishing site.
    - **Warn Users**: Display a warning when an insecure or suspicious website is detected.

Steps to Create the Chrome Extension

1. Create the Chrome Extension Folder Structure

Create a new folder for your Chrome Extension with the following structure:

pgsql

Copy

password-manager-extension/

|-- manifest.json

|-- background.js

|-- popup.html

|-- popup.js

|-- content.js

|-- style.css

|-- assets/

  |-- icon.png

2. manifest.json – The Extension Configuration File

This file will define the metadata, permissions, and resources required by the extension.

json

Copy

```
{
 "manifest_version": 3,
 "name": "TechEdu Password Manager",
 "version": "1.0",
 "description": "A secure password manager and phishing protection system.",
 "permissions": [
  "storage",
  "tabs",
  "activeTab",
  "identity",
  "webNavigation"
 ],
 "background": {
  "service_worker": "background.js"
 },
 "action": {
  "default_popup": "popup.html",
  "default_icon": {
```

```
    "16": "assets/icon.png",
    "48": "assets/icon.png",
    "128": "assets/icon.png"
   }
 },
 "content_scripts": [
  {
   "matches": ["<all_urls>"],
   "js": ["content.js"]
  }
 ],
 "host_permissions": [
  "http://*/*",
  "https://*/*"
 ]
}
```

- **Permissions**:
  - storage: Allows saving passwords locally.
  - tabs: Allows interacting with browser tabs to autofill credentials.
  - identity: To authenticate users if using an OAuth service for login.
  - webNavigation: Enables monitoring navigation to detect phishing sites.

3. background.js – Managing Password Storage & Phishing Detection

This JavaScript file is responsible for managing password storage and phishing protection logic.

javascript

Copy

```
// background.js


// Function to handle phishing detection
```

```
function checkPhishing(url) {
  // Simple phishing check based on known phishing domain list (can be extended)
  const phishingDomains = ['malicious.com', 'phishingsite.com', 'fakebank.com'];

  for (let domain of phishingDomains) {
    if (url.includes(domain)) {
      return true;
    }
  }
  return false;
}

// Listen for tab updates and check for phishing
chrome.webNavigation.onCompleted.addListener(function (details) {
  const url = details.url;

  if (checkPhishing(url)) {
    chrome.scripting.executeScript({
      target: { tabId: details.tabId },
      func: function () {
        alert('Warning: Phishing site detected!');
      }
    });
  }
});
// Store password data securely
chrome.storage.local.set({ passwordData: {} });
```

- **Phishing Detection**: The function checkPhishing() checks whether the visited URL matches a predefined list of malicious domains.

- **Password Storage**: Uses chrome.storage.local to store passwords securely. (In a production system, you'd likely use encryption for this.)

4. popup.html – The Extension Popup UI

This file defines the extension's UI in the popup when the user clicks on the extension icon.

html

Copy

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Password Manager</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Password Manager</h1>
    <div>
      <input type="text" id="passwordInput" placeholder="Enter password" />
      <button id="generatePassword">Generate Strong Password</button>
    </div>
    <div id="passwordStrength"></div>
    <div>
      <button id="savePassword">Save Password</button>
    </div>
  </div>

  <script src="popup.js"></script>
</body>
```

```
</html>
```

5. popup.js – Handling UI Interactions

This file handles user interactions, such as generating a password and saving it.

javascript

Copy

```
// popup.js

document.getElementById('generatePassword').addEventListener('click', function () {
  const password = generateStrongPassword();
  document.getElementById('passwordInput').value = password;
  document.getElementById('passwordStrength').innerText = "Password Strength: Strong";
});

document.getElementById('savePassword').addEventListener('click', function () {
  const password = document.getElementById('passwordInput').value;
  if (password) {
    chrome.storage.local.set({ password: password }, function () {
      alert('Password saved securely!');
    });
  }
});

// Function to generate a strong password
function generateStrongPassword() {
  const chars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()';
  let password = '';
  for (let i = 0; i < 16; i++) {
    password += chars.charAt(Math.floor(Math.random() * chars.length));
  }
```

```
  return password;
}
```

6. content.js – Monitoring Webpage for Phishing Attempts

This content script is injected into every webpage the user visits, checking for phishing attempts.

javascript

Copy

```
// content.js

// Check if the page URL contains a phishing keyword
const suspiciousDomains = ['phishingsite.com', 'malicious.com'];

const currentUrl = window.location.href;

for (let domain of suspiciousDomains) {
  if (currentUrl.includes(domain)) {
    document.body.innerHTML = '<h1>Warning: This is a suspected phishing site!</h1>';
    break;
  }
}
```

This content script looks at the page URL and checks for suspicious domains (this can be extended with more advanced phishing detection algorithms).

7. style.css – Styling the Popup

This file will style the popup UI of the extension.

css

Copy

```
/* style.css */

body {
```

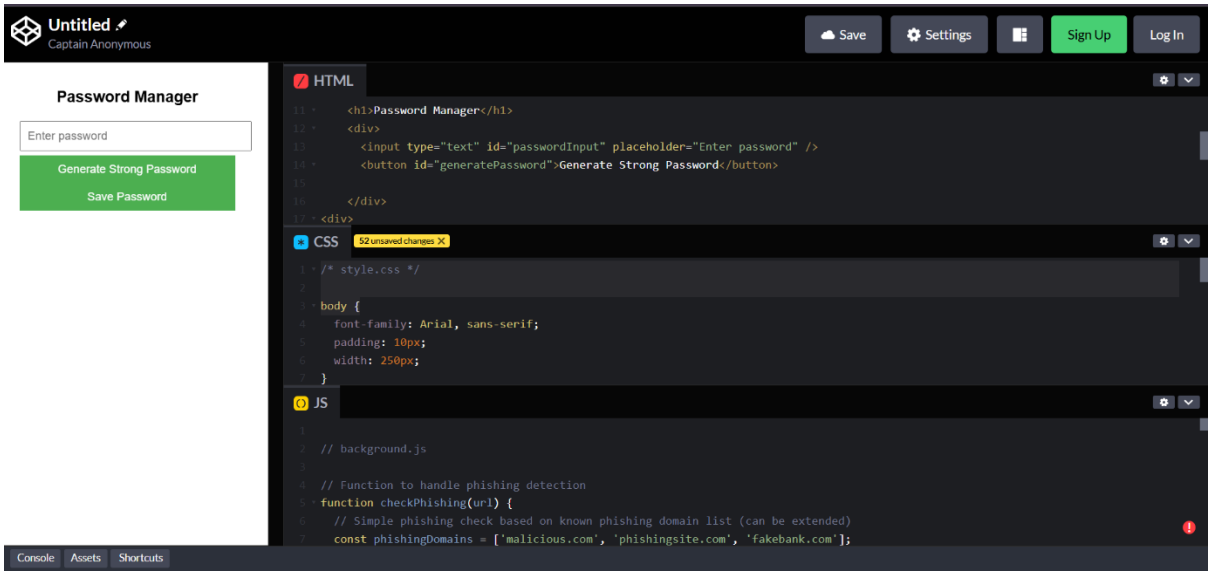
```
  font-family: Arial, sans-serif;
  padding: 10px;
  width: 250px;
}

h1 {
  font-size: 18px;
  text-align: center;
}

input[type="text"] {
  width: 100%;
  padding: 8px;
  margin: 5px 0;
}

button {
  width: 100%;
  padding: 8px;
  background-color: #4CAF50;
  color: white;
  border: none;
  cursor: pointer;
}

button:hover {
  background-color: #45a049;
}
```



Conclusion:

This Chrome extension addresses password fatigue and phishing threats by:

- Automating the generation of strong passwords.
- Storing and managing passwords securely.
- Providing phishing protection by warning users about suspicious websites.
- Offering an easy-to-use interface to store and generate passwords on the fly.

END