CYBERSECURITY ASSIGNMENT- 1

Report on Phishing Detection using Google Safe Browsing API

Name: Harshita Vuthaluru.

Roll No:160123737091

Class:IT-2

Project Title: Phishing Website Detection using Google Safe Browsing API

Date: 01/09/2025

Github Repository: HarshitaVu/Phishing-Website-Detection-using-Google-Safe-Browsing-API

Project Overview

This project explores the **Google Safe Browsing API** as a tool for automated phishing detection.
The objective is to build a phishing detection tool that verifies whether a URL is safe or malicious.

With the growing number of phishing websites targeting banking, social media, and e-commerce portals, using a trusted API helps **automate detection** and protect users from identity theft.

The tool simulates a real-world phishing detection system where URLs are submitted to Google Safe Browsing API, which checks against its database of unsafe URLs and returns results indicating whether the URL is flagged as dangerous.

This project demonstrates how to integrate external cybersecurity APIs into Python for real-world problem-solving.

Technologies & Tools Used

- **Programming Language:** Python (VS Code)
- **Library:** requests (for API calls)
- **API:** Google Safe Browsing API v4
- **Environment:** Local execution in VS Code

System Architecture

1. User provides one or more URLs to check.
2. Python script submits the URLs to Google Safe Browsing API.
3. API checks URLs against phishing, malware, and unwanted software threats.
4. The script fetches the results as JSON.
5. Results are displayed as either **Safe** or **Dangerous**, with threat type if flagged.

Security Features

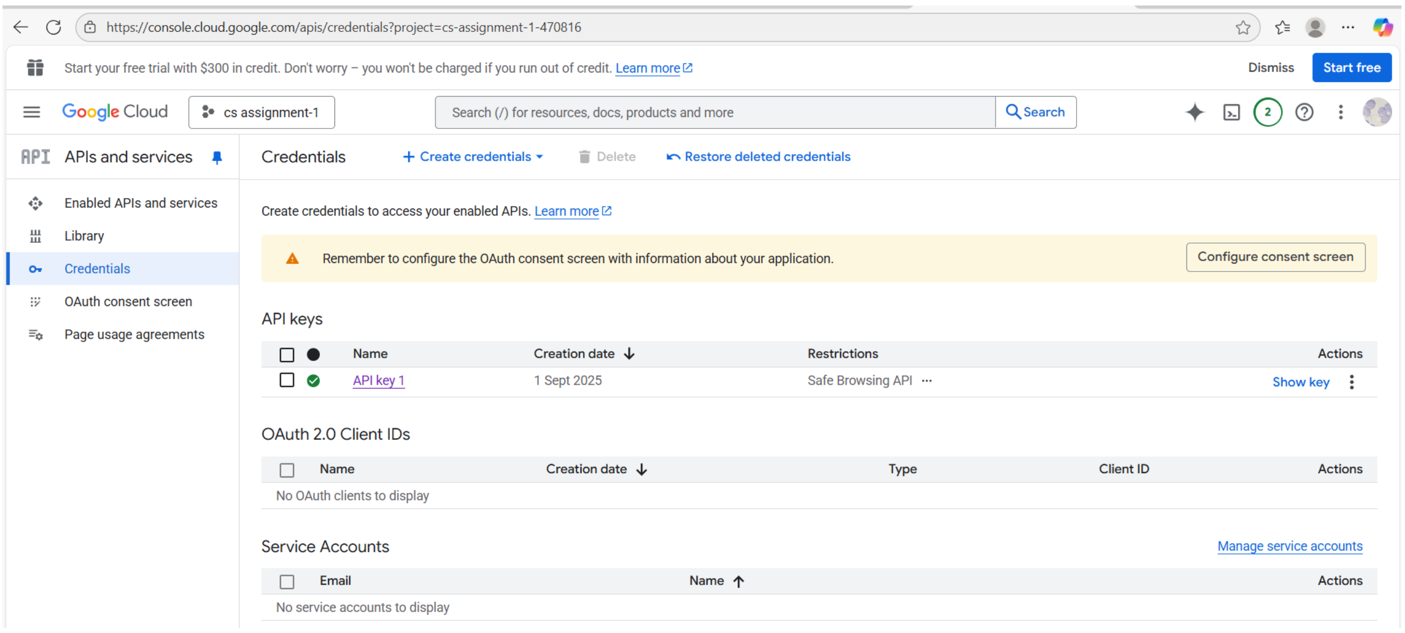
- **Trusted API Source:** Google Safe Browsing provides a reliable database of unsafe URLs.
- **Automated Detection:** Eliminates manual verification of websites.
- **Real-time Analysis:** Newly submitted URLs are checked immediately.
- **Clear Results:** Returns JSON containing threat type information for flagged URLs.

Folder Structure

```
Phishing-Website-Detection-using-Google-Safe-Browsing-API/
│
├── safe_check.py        # Main Python script
├── urls.txt             # Sample test URLs (safe + phishing)
├── Outputs/             # Screenshots of results
│   ├── safe.png         # Example of safe URL
│   ├── malicious.png    # Example of flagged URL
├── report/
│   ├──CYBERSECURITY ASSIGNMENT.pdf  # Final 2-page project report
└── README.md            # Documentation
```

Screenshots:

1. Tool explored: Google-Safe-Browsing-API



https://console.cloud.google.com/apis/credentials?project=cs-assignment-1-470816

Start your free trial with $300 in credit. Don't worry – you won't be charged if you run out of credit. Learn more

Dismiss | Start free

Google Cloud | cs assignment-1 | Search (/) for resources, docs, products and more | Search

APIs and services: Enabled APIs and services, Library, Credentials, OAuth consent screen, Page usage agreements

Credentials | + Create credentials | Delete | Restore deleted credentials

Create credentials to access your enabled APIs. Learn more

Remember to configure the OAuth consent screen with information about your application. | Configure consent screen

API keys

Name	Creation date	Restrictions	Actions
API key 1	1 Sept 2025	Safe Browsing API	Show key

OAuth 2.0 Client IDs

Name	Creation date	Type	Client ID	Actions
No OAuth clients to display				

Service Accounts | Manage service accounts

Email	Name	Actions
No service accounts to display		

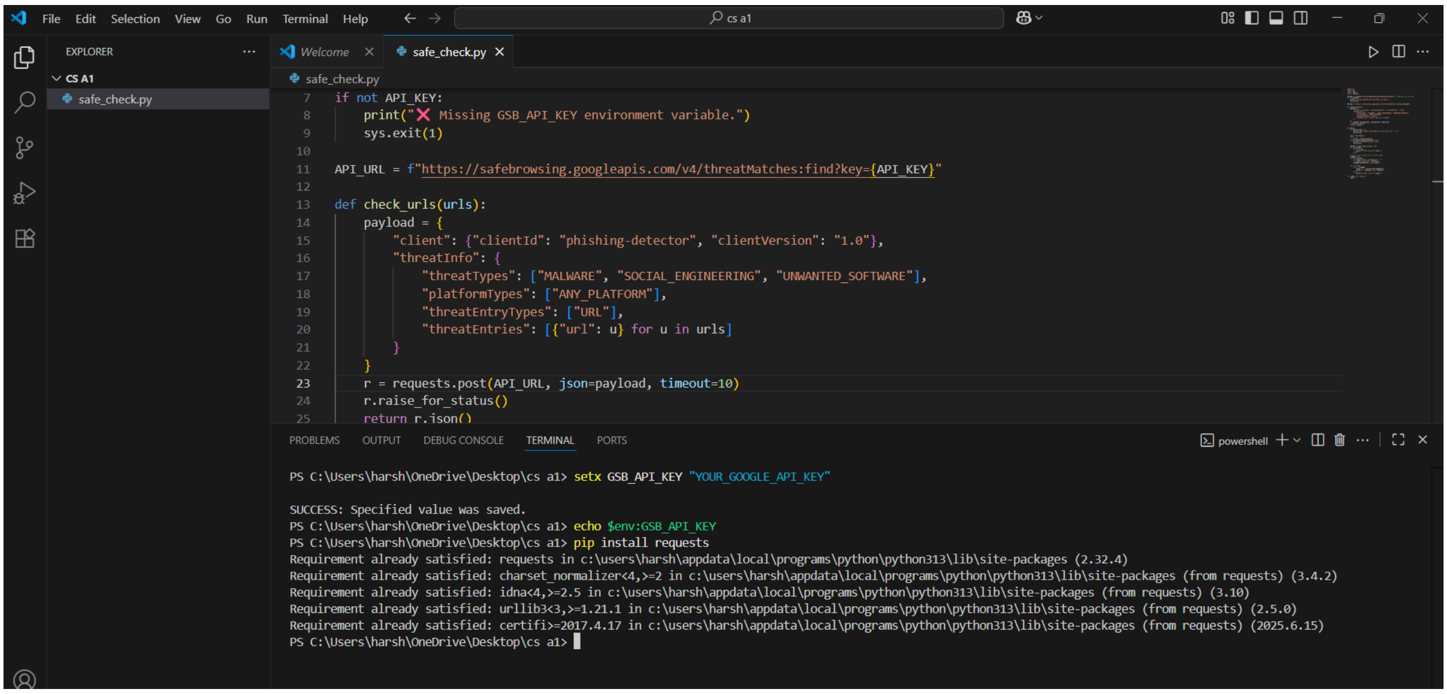
To interact with the **Google Safe Browsing API**, an API key is required. This key acts as a unique identifier that authenticates the user and allows controlled access to Google's Safe Browsing services.

- I registered on **Google Cloud Platform** and created a new project to enable the Google Safe Browsing API.
- I generated a personal API key from the project dashboard.
- The API key was securely stored and not shared publicly (only referenced in the Python script as a variable).

- In the Python script, the API key is included in the **API request URL** (?key=API_KEY) to authorize all API calls.
- Using this key, the script can submit URLs to Google Safe Browsing and fetch real-time analysis results to determine whether the URLs are safe or flagged as malicious (phishing, malware, or unwanted software).

This step ensured that the script could communicate with Google Safe Browsing reliably while following **security best practices**, without exposing the API key publicly.

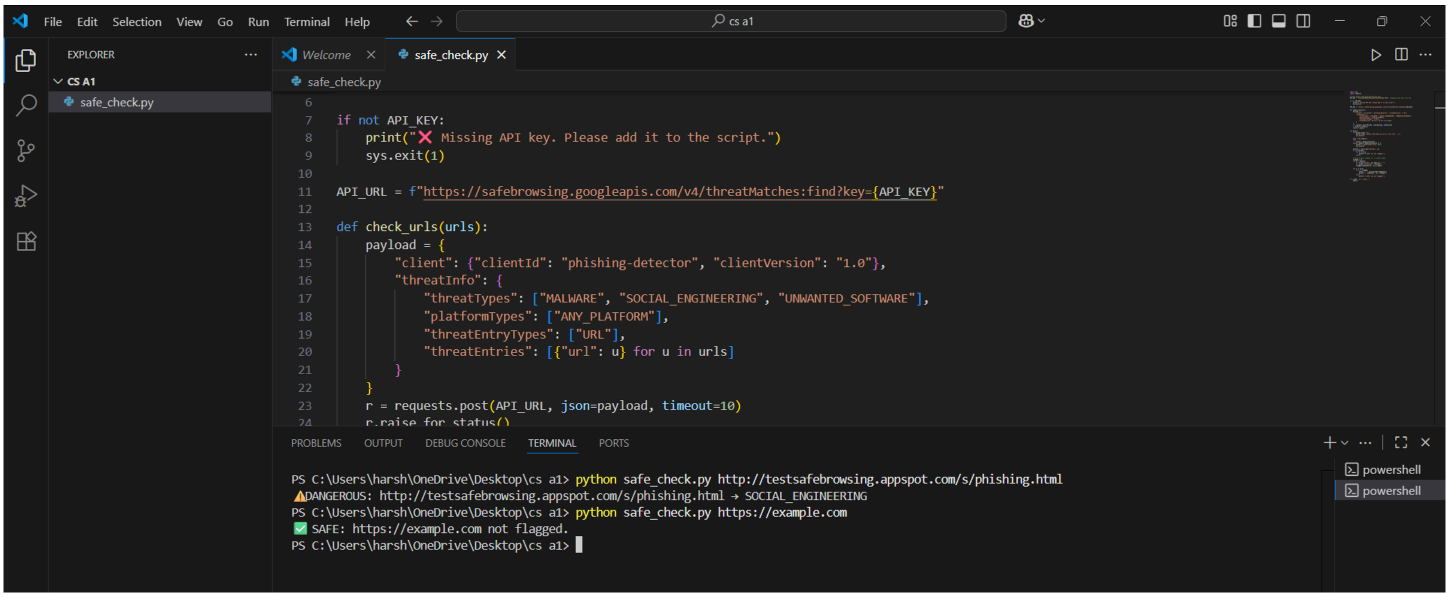
2. Installing Dependencies (pip install requests, upgrade pip, etc.)



```python
if not API_KEY:
    print("❌ Missing GSB_API_KEY environment variable.")
    sys.exit(1)

API_URL = f"https://safebrowsing.googleapis.com/v4/threatMatches:find?key={API_KEY}"

def check_urls(urls):
    payload = {
        "client": {"clientId": "phishing-detector", "clientVersion": "1.0"},
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE"],
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": u} for u in urls]
        }
    }
    r = requests.post(API_URL, json=payload, timeout=10)
    r.raise_for_status()
    return r.json()
```

```
PS C:\Users\harsh\OneDrive\Desktop\cs a1> setx GSB_API_KEY "YOUR_GOOGLE_API_KEY"

SUCCESS: Specified value was saved.
PS C:\Users\harsh\OneDrive\Desktop\cs a1> echo $env:GSB_API_KEY
PS C:\Users\harsh\OneDrive\Desktop\cs a1> pip install requests
Requirement already satisfied: requests in c:\users\harsh\appdata\local\programs\python\python313\lib\site-packages (2.32.4)
Requirement already satisfied: charset_normalizer<4,>=2 in c:\users\harsh\appdata\local\programs\python\python313\lib\site-packages (from requests) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\harsh\appdata\local\programs\python\python313\lib\site-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\harsh\appdata\local\programs\python\python313\lib\site-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\harsh\appdata\local\programs\python\python313\lib\site-packages (from requests) (2025.6.15)
PS C:\Users\harsh\OneDrive\Desktop\cs a1>
```

3. Running script with Safe URL

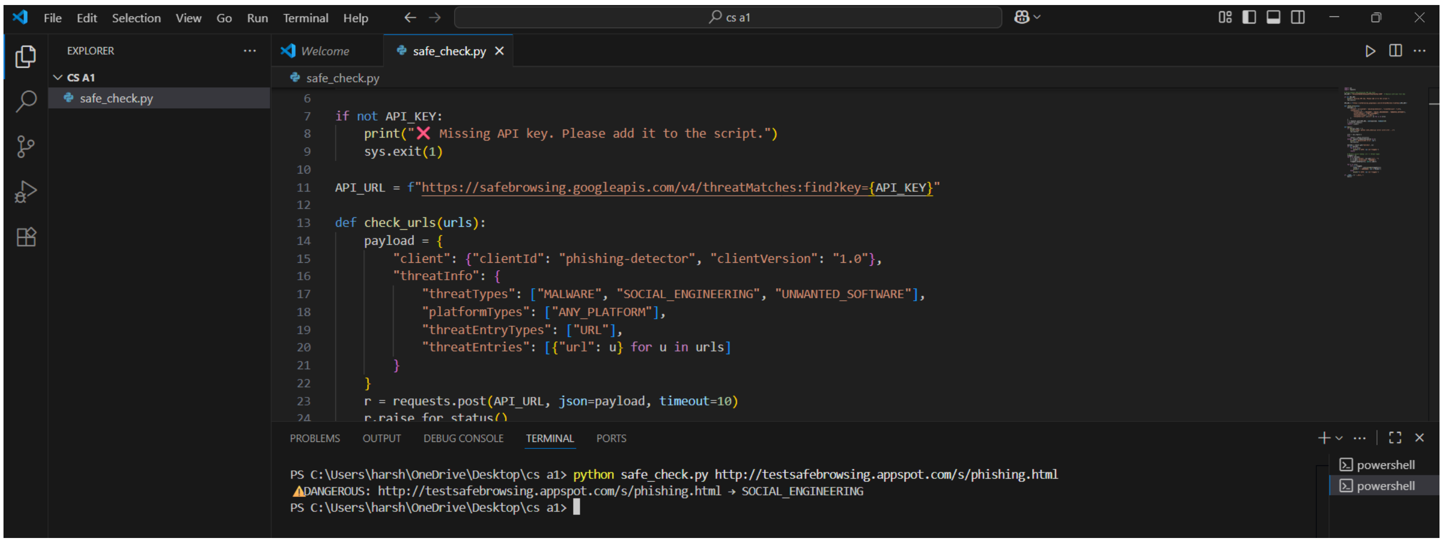


```python
if not API_KEY:
    print("❌ Missing API key. Please add it to the script.")
    sys.exit(1)

API_URL = f"https://safebrowsing.googleapis.com/v4/threatMatches:find?key={API_KEY}"

def check_urls(urls):
    payload = {
        "client": {"clientId": "phishing-detector", "clientVersion": "1.0"},
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE"],
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": u} for u in urls]
        }
    }
    r = requests.post(API_URL, json=payload, timeout=10)
    r.raise_for_status()
```

```
PS C:\Users\harsh\OneDrive\Desktop\cs a1> python safe_check.py http://testsafebrowsing.appspot.com/s/phishing.html
⚠DANGEROUS: http://testsafebrowsing.appspot.com/s/phishing.html → SOCIAL_ENGINEERING
PS C:\Users\harsh\OneDrive\Desktop\cs a1> python safe_check.py https://example.com
✅ SAFE: https://example.com not flagged.
PS C:\Users\harsh\OneDrive\Desktop\cs a1>
```

Output showing harmless(not flagged/malicious)

4. Running script with Phishing URL



```python

if not API_KEY:
    print("❌ Missing API key. Please add it to the script.")
    sys.exit(1)

API_URL = f"https://safebrowsing.googleapis.com/v4/threatMatches:find?key={API_KEY}"

def check_urls(urls):
    payload = {
        "client": {"clientId": "phishing-detector", "clientVersion": "1.0"},
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING", "UNWANTED_SOFTWARE"],
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": u} for u in urls]
        }
    }
    r = requests.post(API_URL, json=payload, timeout=10)
    r.raise_for_status()
```

```
PS C:\Users\harsh\OneDrive\Desktop\cs a1> python safe_check.py http://testsafebrowsing.appspot.com/s/phishing.html
⚠DANGEROUS: http://testsafebrowsing.appspot.com/s/phishing.html → SOCIAL_ENGINEERING
PS C:\Users\harsh\OneDrive\Desktop\cs a1> 
```

Output showing dangerous(flagged/malicious)

Deliverables

- GitHub repository with Python script (safe_check.py), sample URLs (urls.txt), and outputs.
- Outputs folder containing screenshots of results (safe vs malicious).
- Final project report PDF documenting methodology, implementation, and results.
- README.md with setup instructions, usage, and repository structure.

Learning Outcomes

- Learned phishing detection using API-based threat intelligence.
- Gained hands-on experience with Google Safe Browsing API v4.
- Learned how to authenticate with API keys and handle JSON responses in Python.
- Developed skills in integrating cybersecurity tools into Python workflows.
- Practiced using GitHub for project version control and structured deliverables.

Conclusion

This project demonstrates the use of Google Safe Browsing API for phishing website detection.By submitting URLs and analyzing results from Google's threat database, the system can accurately classify websites as **Safe** or **Dangerous**.The solution is lightweight, automated, and practical, avoiding the need to build machine learning models from scratch.

Future enhancements:

- Integrate the script into a browser extension for real-time protection.
- Automate batch URL analysis from emails or logs.
- Combine with ML-based detection models for hybrid phishing detection.