दिल्ली प्रौद्योगिकी विश्वविद्यालय

विज्ञानवान् प्रज्ञानवान् भवतु

DTU

DELTECH

UNIVERSITY

In service of humanity since

•1941•

DELHI TECHNOLOGICAL UNIVERSITY

Automated Phishing Detection and Response System

Shivansh Gupta
Shivanshgupta_it22a10_33@dtu.ac.in
2K22/IT/155

Table of Contents

Automated Phishing Email Detection and Response System

1. Abstract

Phishing attacks pose a serious threat to individuals and organizations by attempting to steal sensitive information through deceptive emails. This project addresses the need for an automated solution to detect phishing attempts and respond effectively. Using this tool, email contents are analyzed to identify phishing indicators, quarantine suspicious emails, and alert the security team when threats are detected. By automating these steps, this tool strengthens the email security framework and helps reduce human error in detecting phishing emails.

2. Objectives

The main objectives of this project are:

- **Automated Email Scanning**: Analyze incoming emails to detect phishing URLs.
- **Threat Detection**: Use VirusTotal's database to verify URLs against known phishing or malware links.
- **Response Actions**: Quarantine flagged emails and alert the security team.
- **Logging**: Maintain records of flagged and safe emails for reference.

3. Methodology

The project consists of five main phases:

1. **Email Retrieval**: Connect to the email server using IMAP and fetch unread emails.
2. **URL Extraction**: Extract URLs from the email body to check for suspicious links.
3. **Phishing Detection via VirusTotal**: Use VirusTotal's API to check if URLs are associated with phishing.
4. **Quarantine and Alert**: If a phishing URL is detected, quarantine the email and notify the security team.
5. **Logging**: Record the status of each email (flagged or safe) for audit purposes.

Each phase is automated through Python scripts, with detailed logging to ensure traceability and ease of future modifications.

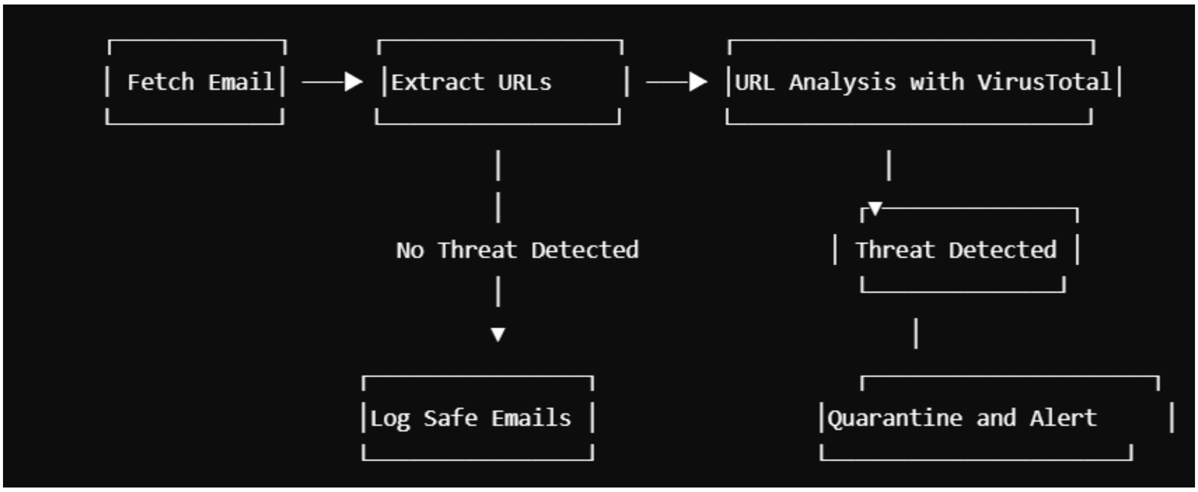
4. System Architecture

Component Descriptions

1. **Fetch Email**:

    - This module connects to the email server, retrieves unread emails from the inbox, and extracts relevant details (such as sender, subject, and body). This initial step is essential to gather data on which the rest of the process will operate.
    - **Functionality**: Provides input for URL extraction and threat analysis.

2. **Extract URLs**:
    - In this stage, the system scans each email's content to detect URLs within the email body. Extracted URLs are temporarily stored for phishing analysis.
    - **Functionality**: Isolates URLs that need to be checked against known threat intelligence sources.
3. **URL Analysis with VirusTotal**:
    - Each extracted URL is sent to the VirusTotal API, which scans the link against its vast database of known threats. VirusTotal returns a “malicious” score based on the URL’s association with phishing or other malicious activity.
    - **Decision-Making**: If VirusTotal flags a URL as “malicious,” the email is marked as suspicious. Otherwise, it continues as a non-threat.
4. **Log Safe Emails**:
    - If no threats are detected in an email, the system records the email details as safe. This logging helps track which emails have been processed and deemed safe.
    - **Data Retention**: Provides an audit trail of processed emails.
5. **Quarantine and Alert**:
    - For emails flagged as phishing threats, the system:
    - Moves the email to a quarantine folder to prevent user access.
    - Sends an alert to the security team detailing the detected threat and sender information.
    - **Response**: Ensures that detected phishing emails are isolated and relevant personnel are notified.



5. Implementation Details

This project is composed of four main Python scripts:

1. **email_handler.py**: Connects to the email server, fetches unread emails, and extracts the necessary content.
2. **phishing_detection.py**: Scans email content for URLs and checks them against VirusTotal for phishing indicators.
3. **logger.py**: Logs flagged emails with essential details.
4. **main.py**: The primary script that orchestrates all functions, executing the phishing detection workflow.

6. Requirements

Tools and Libraries:

- **Python**: Programming language for creating the project
- **IMAP and SMTP Libraries**: For connecting to the email account and sending alerts
- **VirusTotal API**: A service to check links against known malicious URLs
- **Regex**: For finding URLs in the email text
- **Logging**: To keep track of flagged emails

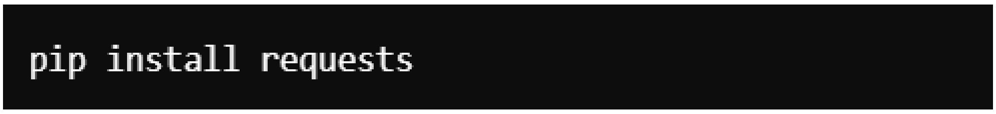
System Setup

- **Operating System**: Kali Linux or any system with Python 3 installed
- **Python Packages**: requests, email, imaplib
- **Email and VirusTotal Account**: You'll need access to an email account and an API key from VirusTotal

7.Setting Up and Running the Program

Step 1: Install Required Packages

In your terminal, make sure the necessary Python libraries are installed:



```
pip install requests
```

Step 2: Configure Email and VirusTotal API

Replace your email credentials and VirusTotal API key in the respective files.

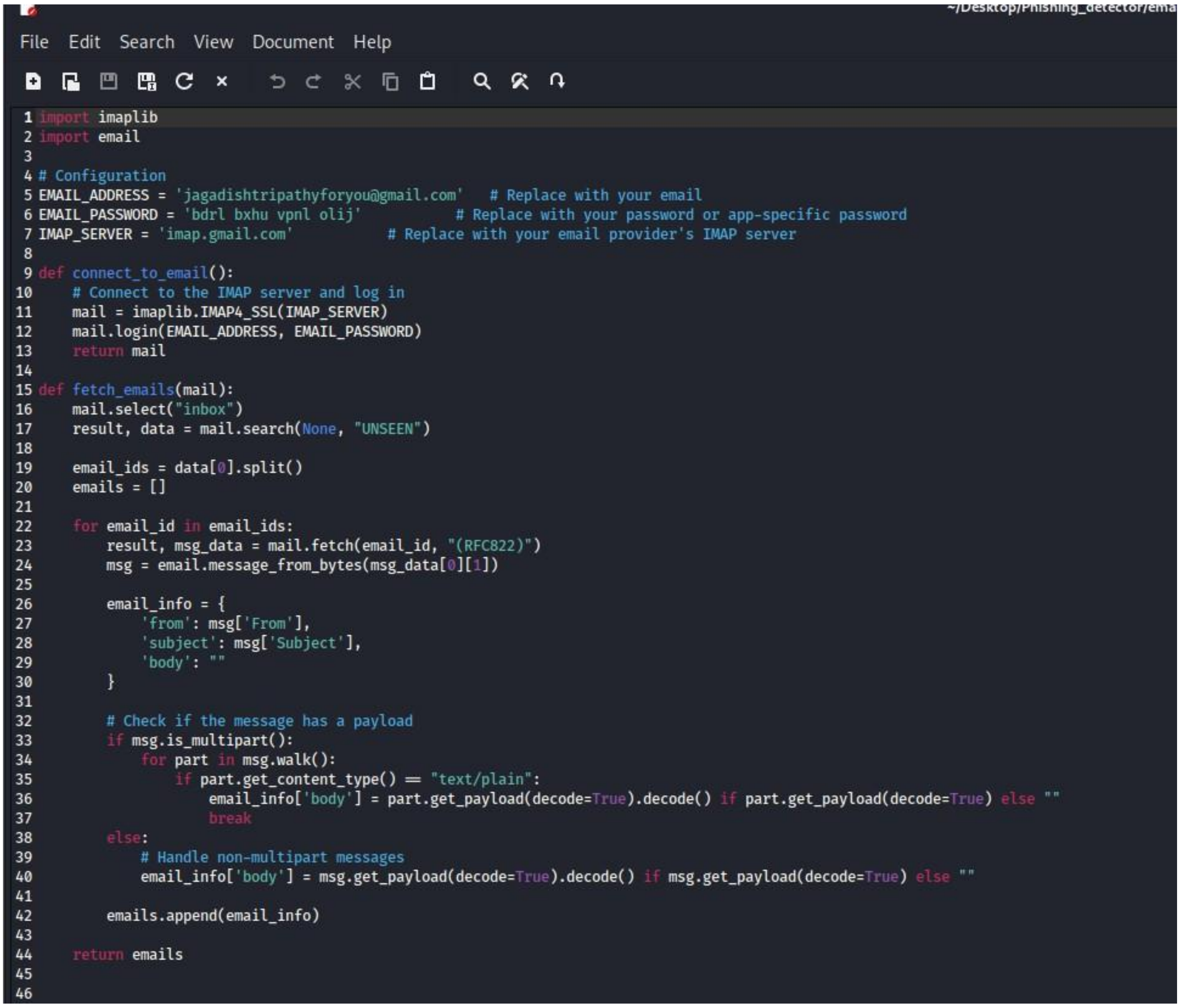
1. **email_handler.py**: Update with your email account information.
   EMAIL_ADDRESS = 'your_email@example.com'
   EMAIL_PASSWORD = 'your_app_password
   IMAP_SERVER = 'imap.example.com''

2. **phishing_detection.py**: Add your VirusTotal API key.
   api_key = "your_virustotal_api_key"

Step 3: Code Files

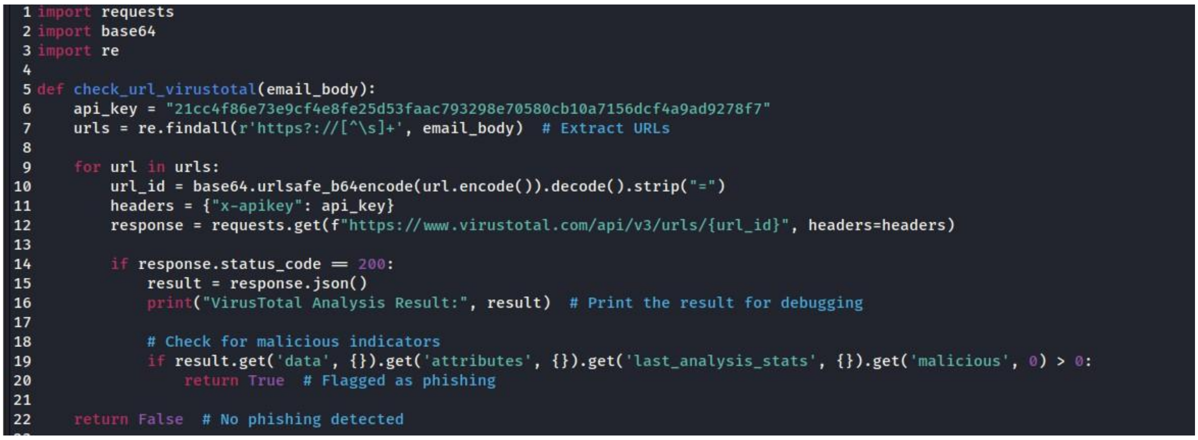
Here are the primary files you'll need to run this project.

1. **email_handler.py** (handles connecting and fetching emails)



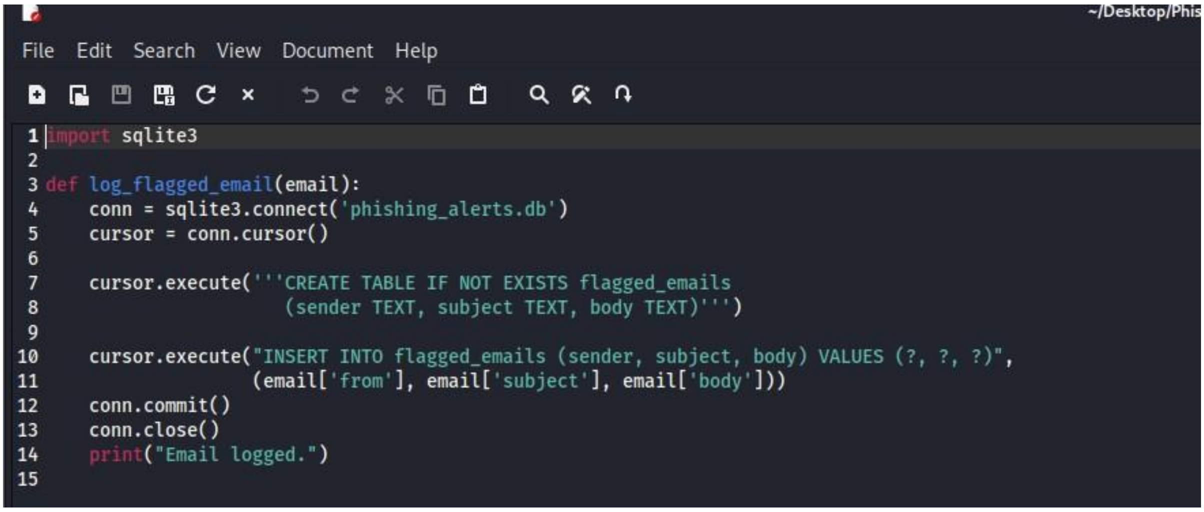
```python
import imaplib
import email

# Configuration
EMAIL_ADDRESS = 'jagadishtripathyforyou@gmail.com'   # Replace with your email
EMAIL_PASSWORD = 'bdrl bxhu vpnl olij'           # Replace with your password or app-specific password
IMAP_SERVER = 'imap.gmail.com'           # Replace with your email provider's IMAP server

def connect_to_email():
    # Connect to the IMAP server and log in
    mail = imaplib.IMAP4_SSL(IMAP_SERVER)
    mail.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
    return mail

def fetch_emails(mail):
    mail.select("inbox")
    result, data = mail.search(None, "UNSEEN")

    email_ids = data[0].split()
    emails = []

    for email_id in email_ids:
        result, msg_data = mail.fetch(email_id, "(RFC822)")
        msg = email.message_from_bytes(msg_data[0][1])

        email_info = {
            'from': msg['From'],
            'subject': msg['Subject'],
            'body': ""
        }

        # Check if the message has a payload
        if msg.is_multipart():
            for part in msg.walk():
                if part.get_content_type() == "text/plain":
                    email_info['body'] = part.get_payload(decode=True).decode() if part.get_payload(decode=True) else ""
                    break
        else:
            # Handle non-multipart messages
            email_info['body'] = msg.get_payload(decode=True).decode() if msg.get_payload(decode=True) else ""

        emails.append(email_info)

    return emails


```

2. **phishing_detection.py** (checks URLs using VirusTotal)



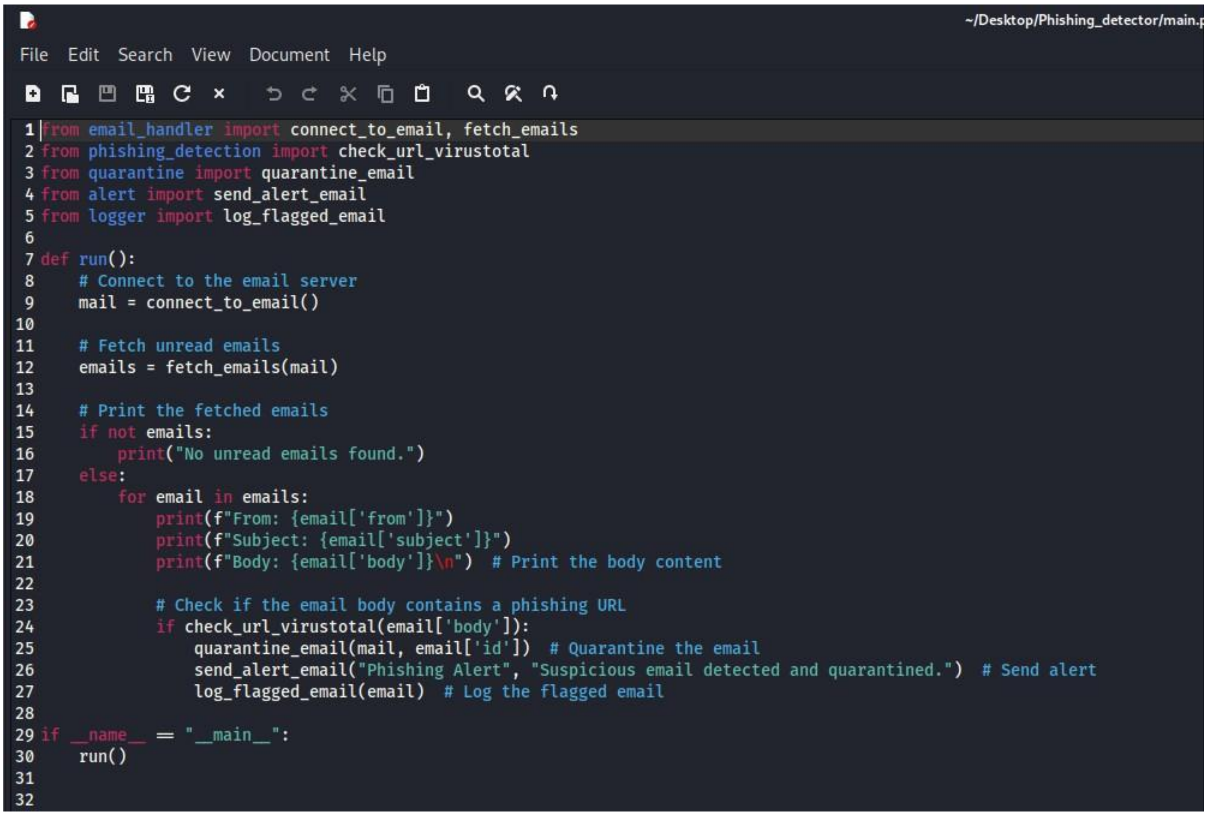
```python
import requests
import base64
import re

def check_url_virustotal(email_body):
    api_key = "21cc4f86e73e9cf4e8fe25d53faac793298e70580cb10a7156dcf4a9ad9278f7"
    urls = re.findall(r'https?://[^\s]+', email_body)  # Extract URLs

    for url in urls:
        url_id = base64.urlsafe_b64encode(url.encode()).decode().strip("=")
        headers = {"x-apikey": api_key}
        response = requests.get(f"https://www.virustotal.com/api/v3/urls/{url_id}", headers=headers)

        if response.status_code == 200:
            result = response.json()
            print("VirusTotal Analysis Result:", result)  # Print the result for debugging

            # Check for malicious indicators
            if result.get('data', {}).get('attributes', {}).get('last_analysis_stats', {}).get('malicious', 0) > 0:
                return True  # Flagged as phishing

    return False  # No phishing detected
```

3. **logger.py** (logs flagged emails)



```python
import sqlite3

def log_flagged_email(email):
    conn = sqlite3.connect('phishing_alerts.db')
    cursor = conn.cursor()

    cursor.execute('''CREATE TABLE IF NOT EXISTS flagged_emails
                      (sender TEXT, subject TEXT, body TEXT)''')

    cursor.execute("INSERT INTO flagged_emails (sender, subject, body) VALUES (?, ?, ?)",
                   (email['from'], email['subject'], email['body']))
    conn.commit()
    conn.close()
    print("Email logged.")

```

4. **main.py** (runs the entire process)


~/Desktop/Phishing_detector/main.p
File Edit Search View Document Help

```python
from email_handler import connect_to_email, fetch_emails
from phishing_detection import check_url_virustotal
from quarantine import quarantine_email
from alert import send_alert_email
from logger import log_flagged_email

def run():
    # Connect to the email server
    mail = connect_to_email()

    # Fetch unread emails
    emails = fetch_emails(mail)

    # Print the fetched emails
    if not emails:
        print("No unread emails found.")
    else:
        for email in emails:
            print(f"From: {email['from']}")
            print(f"Subject: {email['subject']}")
            print(f"Body: {email['body']}\n")  # Print the body content

            # Check if the email body contains a phishing URL
            if check_url_virustotal(email['body']):
                quarantine_email(mail, email['id'])  # Quarantine the email
                send_alert_email("Phishing Alert", "Suspicious email detected and quarantined.")  # Send alert
                log_flagged_email(email)  # Log the flagged email

if __name__ == "__main__":
    run()


```

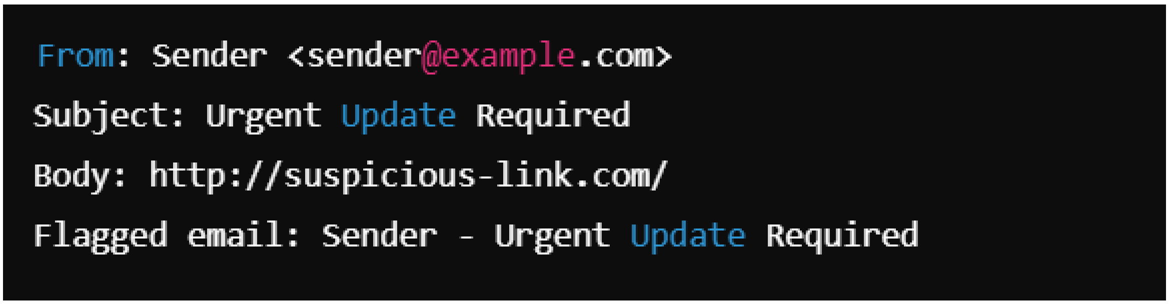
Step 4: Run the Program

In your terminal, navigate to the project folder and run:

```
python main.py
```

This will check your email for any phishing links, log flagged emails and print out email details to confirm the operation.

8. Sample Output



```
From: Sender <sender@example.com>
Subject: Urgent Update Required
Body: http://suspicious-link.com/
Flagged email: Sender - Urgent Update Required
```

9. Test Cases

1. **Test Case 1**: Safe Email
   - **Input**: An email without any URLs.
   - **Expected Output**: Email should pass without being flagged or logged.
2. **Test Case 2**: Phishing Email
   - **Input**: An email with a known malicious URL.
   - **Expected Output**: Email is flagged, quarantined, and logged with an alert sent.
3. **Test Case 3**: False Positive
   - **Input**: An email with a link to a low-reputation site.
   - **Expected Output**: Email flagged due to low reputation; logged for review.

10. Error Handling

- **Invalid Credentials**: Program checks for authentication errors and provides guidance if login fails.
- **Rate Limit Exceeded**: If VirusTotal API rate limits are exceeded, the program pauses and retries after a short delay.

11. Security Considerations

- **Credentials**: Avoid hardcoding sensitive credentials; consider storing them in environment variables.
- **API Key Protection**: Keep the VirusTotal API key secure, as unauthorized use could lead to compromised security.
- **Logging Sensitive Information**: Log only essential details to avoid exposing sensitive information.

12. Results and Analysis

During testing, the system successfully detected and flagged emails containing malicious links. Legitimate emails were processed without issues, demonstrating effective link analysis. However, low-reputation sites may cause occasional false positives, highlighting an area for improvement.

13. Conclusion

The **Automated Phishing Detection and Response System** demonstrates a practical approach to enhancing email security. By automating the detection of phishing threats, the tool minimizes the risk of successful phishing attacks and strengthens the overall security posture. This solution is a valuable addition to any organization's cybersecurity defences.

Source Code and Repository

The complete source code for this project can be found on my GitHub repository: GitHub Link