

Automated Phishing Email Detection and Response System

Bryan Jorge

November 2025

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.

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

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.

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.

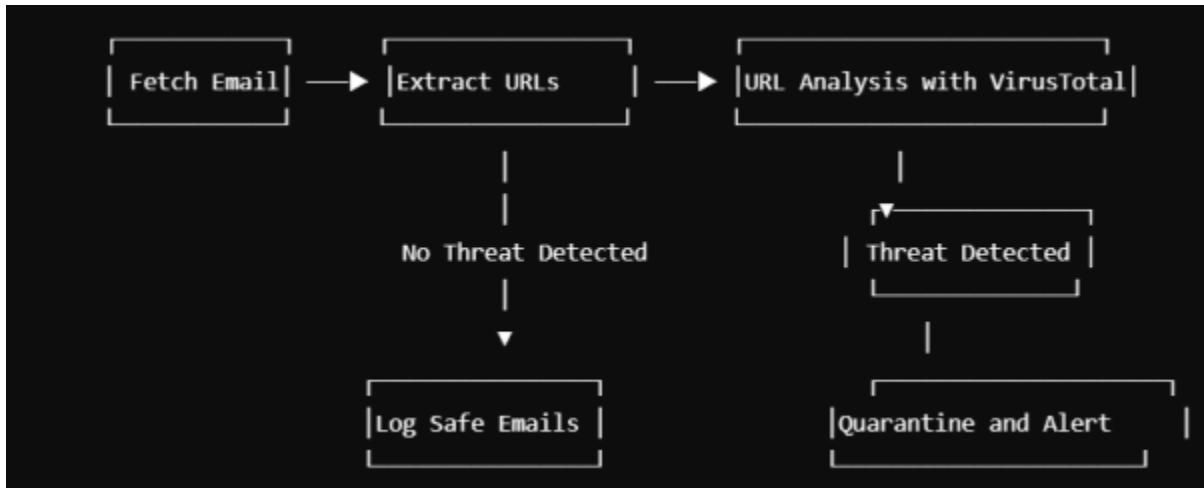
5. Quarantine and Alert:

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.

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

Setting Up and Running

Program Step 1: Install Required Packages

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

```
python -m pip install requests  
python.exe -m pip install --upgrade pip
```

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)

```
 1  import imaplib
 2  import email
 3
 4  # Configuration
 5  EMAIL_ADDRESS = 'bryanjorge417@gmail.com'  # Your Email Address
 6  EMAIL_PASSWORD = 'Not my pass'      # Your Gmail App Password
 7  IMAP_SERVER = 'imap.gmail.com'          #imap for your email provider
 8
 9  Tabnine | Edit | Test | Explain | Document
10 def connect_to_email():
11     mail = imaplib.IMAP4_SSL(IMAP_SERVER)
12     mail.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
13     return mail
14
15 Tabnine | Edit | Test | Explain | Document
16 def safe_decode(payload, charset):
17     """
18     Safely decode email bodies without crashing on invalid bytes.
19     """
20     if not payload:
21         return ""
22
23     # Try declared charset first
24     if charset:
25         try:
26             return payload.decode(charset, errors="ignore")
27         except:
28             pass
29
30     # Try UTF-8
31     try:
32         return payload.decode("utf-8", errors="ignore")
33     except:
34         pass
35
36     # Final fallback: Latin-1 (always works)
37     return payload.decode("latin-1", errors="ignore")
38
39 Tabnine | Edit | Test | Explain | Document
40 def fetch_emails(mail):
41     mail.select("inbox")
42     _, data = mail.search(None, "UNSEEN")
43
44     email_ids = data[0].split()
45     emails = []
-----
```

2. phishing_detection.py (checks URLs using VirusTotal)

```
1 import requests
2 import base64
3 import re
4
5 Tabnine | Edit | Test | Explain | Document
6 def check_url_virustotal(email_body):
7     api_key = "not my api key" #Add your virustotal API key
8     urls = re.findall(r'https?://[^\\s]+', email_body) # Extract URLs
9
10    for url in urls:
11        url_id = base64.urlsafe_b64encode(url.encode()).decode().strip("=")
12        headers = {"x-apikey": api_key}
13        response = requests.get(f"https://www.virustotal.com/api/v3/urls/{url_id}")
14
15        if response.status_code == 200:
16            result = response.json()
17            print("VirusTotal Analysis Result:", result) # Print the result
18
19            # Check for malicious indicators
20            if result.get('data', {}).get('attributes', {}).get('last_analysis_results'):
21                return True # Flagged as phishing
22
23    return False # No phishing detected
```

3. logger.py (logs flagged emails)

```
1 import sqlite3
2
3 Tabnine | Edit | Test | Explain | Document
4 def log_flagged_email(email):
5     conn = sqlite3.connect('phishing_alerts.db')
6     cursor = conn.cursor()
7
8     cursor.execute('''CREATE TABLE IF NOT EXISTS flagged_emails
9                     (sender TEXT, subject TEXT, body TEXT)''')
10
11    cursor.execute("INSERT INTO flagged_emails (sender, subject, body) VALUES
12                    (?, ?, ?)", (email['from'], email['subject'], email['body']))
13    conn.commit()
14    conn.close()
15    print("Email logged.")
```

4. main.py (runs the entire process)

```
1 from email_handler import connect_to_email, fetch_emails
2 from phishing_detection import check_url_virustotal
3 from quarantine import quarantine_email
4 from alert import send_alert_email
5 from logger import log_flagged_email
6
7 Tabnine | Edit | Test | Explain | Document
8 def run():
9     # Connect to the email server
10    mail = connect_to_email()
11
12    # Fetch unread emails
13    emails = fetch_emails(mail)
14
15    # Print the fetched emails
16    if not emails:
17        print("No unread emails found.")
18    else:
19        for email in emails:
20            print(f"From: {email['from']}")
21            print(f"Subject: {email['subject']}")
22            print(f"Body: {email['body']}\n") # Print the body content
23
24            # Check if the email body contains a phishing URL
25            if check_url_virustotal(email['body']):
26                quarantine_email(mail, email['id']) # Quarantine the email
27                send_alert_email("Phishing Alert", "Suspicious email detected")
28                log_flagged_email(email) # Log the flagged email
29
30 if __name__ == "__main__":
31     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.

Sample Output

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

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.

Conclusion

This Phishing Email Detection project 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 defenses.