МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Оксана ШОВКОПЛЯС

(підпис)

грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»

освітньо-професійної програми «Інформатика»

на тему: Інформаційна технологія виявлення фішингових атак

здобувач(а) групи ІН.м-34 Горбуленка Миколи Валерійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Микола Горбуленко

(підпис)

Керівник Анна Бадалян

канд. фіз.-мат. наук, ст. викладач

кафедри КН

(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Оксана

ШОВКОПЛЯС

_____

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»

здобувача групи ІН.м-34 Горбуленка Миколи Валерійовича

1. Тема роботи: Інформаційна технологія виявлення фішингових атак

затверджую наказом №1257-VI по СумДУ від 03.12.2024

2. Термін здачі здобувачем кваліфікаційної роботи *до 10 грудня 2024 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляг технологій які використовуються для аналізу тексту та вкладень 3) Розробка програмного забезпечення 4) Тестування роботи ПЗ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «11» листопада 2024 р.

Завдання прийняв до виконання

_____

(підпис)

Керівник

_____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	11.11.2024	
2	*Огляг технологій які використовуються для аналізу тексту та вкладень*	16.11.2024	
3	*Розробка програмного забезпечення*	20.11.2024	
4	*Тестування роботи ПЗ та виправлення помилок*	01.12.2024	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	09.12.2024	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 48 стр., 14 рис., 5 додатків, 23 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена вирішенню критично важливої проблеми кібербезпеки, пов'язаної із зростанням кількості фішингових атак. Розробка ефективного інструменту для виявлення таких загроз дозволить підвищити захист користувачів від шахрайських дій, сприятиме безпеці інформаційних систем та зниженню фінансових втрат.

Об’єкт дослідження — Процес ідентифікації та аналізу фішингових загроз у електронній пошті.

Предмет дослідження - Інструменти, технології та алгоритми для аналізу підозрілих повідомлень.

Мета роботи — Розробка програмного забезпечення для виявлення фішингових електронних листів за допомогою сучасних API та методів аналізу.

Методи дослідження — аналітичний огляд існуючих аналогів, сучасні підходи до виявлення фішингових атак, алгоритми обробки даних листів на предмет фішингової атаки.

Результати — Розроблено програму для моніторингу електронної пошти в реальному часі, інтегровано механізми перевірки тексту, вкладень та посилань у листах, функція аналізу відправника за допомогою SPF, DKIM і DMARC, логування підозрілих листів і push-сповіщення для користувача у разі виявлення загроз. Проведено тестування програми з використанням реальних даних, що підтвердило її ефективність у виявленні фішингових повідомлень.

ІНФОРМАЦІЙНА СИСТЕМА, АНАЛІЗ ДАННИХ, E-MAIL SERVICE, PYTON, DMARC, API.

ЗМІСТ

ВСТУП

Фішингові атаки є однією з найбільш поширених і небезпечних кіберзагроз сучасності. Незважаючи на постійний розвиток захисних технологій, кількість успішних фішингових атак залишається значною, завдаючи серйозних збитків як приватним особам, так і корпоративним структурам. Таким чином, розробка ефективних інформаційних технологій для виявлення фішингових атак є надзвичайно актуальною і необхідною задачею.

Актуальність. З розвитком інформаційного суспільства і зростанням обсягів інформаційного обміну важливість захисту особистих і корпоративних даних стає все більш очевидною. Електронна пошта є одним з основних засобів комунікації, але також і основною мішенню для фішингових атак. Тому тема кваліфікаційної роботи є надзвичайно актуальною, оскільки вона присвячена вирішенню важливої практичної задачі виявлення фішингових атак шляхом розробки відповідних методів, моделей та інформаційної технології. Ефективність пропонованих рішень безпосередньо впливає на безпеку користувачів, забезпечуючи захист їх особистих даних та фінансових ресурсів.

Об’єкт дослідження. Процес ідентифікації та аналізу фішингових загроз у електронній пошті. Це включає в себе аналіз вмісту листів, виявлення підозрілих характеристик та алгоритми класифікації листів як фішингових або безпечних.

Предмет дослідження. Інструменти, технології та алгоритми для аналізу підозрілих повідомлень. що включає в себе використання ключових слів або фраз для виявлення підозрілого листа.

Новизна даної роботи полягає в розробці нової інформаційної технології, яка дозволяє автоматично виявляти фішингові атаки у електронних листах на платформі GMAIL.

Структура. Дана робота складається зі вступу, аналітичного огляду сучасних методів виявлення фішингових атак, аналізу аналогічних проєктів, постановки задачі, вибору методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, тестування та аналізу отриманих результатів, висновків, списку використаних джерел та додатків. У аналітичному огляді розглянуто сучасний стан технологій захисту від фішингових атак, проведено аналіз існуючих рішень, їх переваг та недоліків. У розділі вибору методу розв’язання задачі описано використовувані алгоритми та методи для класифікації листів. Опис програмного забезпечення містить детальну інформацію про розроблену систему та її функціональні можливості. У розділі аналізу результатів представлено результати тестування системи на реальних даних та оцінено її ефективність.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз кіберзагроз

Постановка проблеми. Зараз щодня виникає багато проблем з безпекою даних. Хакери зараз дуже добре вміють використовувати свої знання, щоб зламати чужу систему і викрасти інформацію. Фішинг є одним з таких методів, які використовуються для отримання інформації. Фішинг - це кіберзлочин, в якому об'єктом нападу є електронна пошта, телефон, текстові повідомлення, особиста інформація, банківські реквізити, дані кредитної картки, пароль. [4]

Фішинг в основному є формою крадіжки ідентифікаційних даних в Інтернеті. Фішер використовує соціальну інженерію для викрадення персональних даних жертви та реквізитів її рахунку.[4]

Фішинг є одним із найбільш поширених методів кіберзлочинності, який використовується для маніпуляції користувачами з метою отримання їхніх конфіденційних даних. Така проблема особливо актуальна у сучасному цифровому світі, де велика кількість персональних і корпоративних даних знаходяться у віртуальному просторі. Витік цих даних може призвести до фінансових збитків, втрати репутації та навіть загроз національній безпеці.

Основною складністю є динамічність методів фішингу: зловмисники адаптують свої підходи, використовуючи сучасні технології, включаючи автоматизацію атак, застосування штучного інтелекту для створення реалістичних приманок та соціальну інженерію.

Найпоширеніший вид фішингу – це листи на електронну пошту, схожі повідомлення від існуючих, легальних організацій. Дані повідомлення зазвичай несуть не ознайомлювальний характер, а примушують, підштовхують користувача до будь-якої дії, наприклад, підтвердження облікового запису. Такі листи розсилаються великій кількості користувачів і, зазвичай, не містять у собі будь-якої детальної інформації про атаковане. Цільовий фішинг є більш складною

модифікацією фішингових атак і, перш ніж здійснити таку атаку, зловмисник повинен з'ясувати якомога більше особистої інформації про жертву, тому цільовий або персоналізований фішинг вимагає ретельної підготовки і не малих фізичних та матеріальних витрат від зловмисника. Але ймовірність проведення успішної атаки на основі цільового фішингу в рази збільшується порівняно зі звичайним нецільовим фішингом.[3]

1.2 Актуальність проблеми. Дослідження фішингових атак почалося з ранніх 2000-х років, коли перші інциденти були пов’язані з шахрайськими електронними листами, які імітували банківські установи [1]. У подальших роботах акцент робився на розробку систем автоматичного виявлення шкідливих URL-адресів [2].

На сьогодні, за даними Verizon [5], фішинг-атаки складають понад 36% усіх випадків порушень кібербезпеки. Це пов’язано як із технічними вразливостями систем, так і з людським фактором. У 2023 році тільки у фінансовій сфері було зафіксовано збитки від фішингу на суму понад 10 мільярдів доларів [6].

Специфікою фішингу є те, що жертва шахрайства надає свої конфіденційні дані добровільно. Для цього зловмисники оперують такими інструментами, як фішингові сайти, e-mail розсилка, фішингові landing page, спливаючі вікна, таргетована реклама. Користувач отримує пропозицію зареєструватися для отримання будь-якої вигоди або підтвердити свої персональні дані нібито для банківських або комерційних установ, клієнтом яких він є. Як правило, шахраї маскуються під відомі компанії, додатки соціальних мереж, сервіси електронної пошти. Електронна адреса відправника дійсно схожа на адресу знайомої користувачеві компанії. Наприклад, щоб замаскуватися під інтернет-магазин Aliexpress, шахраї шлють листи з адрес, що містять слово Alliexpress або Aliexxpress. Працює та сама схема, яка змушує людей купувати дешеві китайські кросівки таких «всесвітньо відомих брендів», як Pumma або Abibas.[7]

Зловмисники користуються низьким рівнем обізнаності користувачів, зокрема, незнанням елементарних правил мережевої безпеки. Перш за все, організаторів фішинг-атак цікавлять персональні дані, які дають доступ до грошей, тому жертвами фішингу можуть ставати не тільки окремі люди, а й банки, електронні платіжні системи, аукціони.[7]

1.3 Аналіз проблеми. Кіберзлочинність фішингових атак в Інтернеті зростає протягом багатьох років. Вже на початку поширення пандемії в Україні короновірусної хвороби в березні 2020 року вказувалося, що кількість зареєстрованих випадків шахрайства в Інтернеті зросла. Навіть тоді багато інтернет-шахраїв використовували коронавірус для своїх шахрайств. Проаналізувавши відсоток користувачів Інтернет в окремих країнах які знаходяться в Європі які стали жертвами крадіжки особистих даних у 2020-2024 роках.[3] На основі відкритої інформації проведено аналіз жертв фішингових атак у 10 країнах Європи та Україні за період 2020-2024 років. Результат аналізу наведений на рис 1.1

Рис 1.1 – Аналіз інцидентів фішингу.

(Аналіз проводився на основі відкритої інформації)

Наведені дані показують що в Германіі статистика звернень громадян що стали жертвами кібер-шахрайств становить вище ніж інші представлені країни. Основною проблемою є що значна кількість людей не перевіряють інформацію що надходить до них.

1.4 Структура цільової атаки фішингу:

Фішинг-атака включає такі етапи:

- Збір інформації:
    - Зловмисники досліджують ціль через соціальні мережі, корпоративні сайти, бази даних.
- Розробка приманки:
    - Імітуються легітимні веб-сайти, використовуючи схожі доменні імена (typosquatting) або схожі візуальні елементи.
- Розповсюдження приманки:
    - Електронні листи, SMS, повідомлення в соціальних мережах із закликом перейти за посиланням або завантажити файл.
- Компрометація:
    - Користувач переходить за посиланням, де вводить свої дані, які потім зловмисник використовує для доступу до систем.

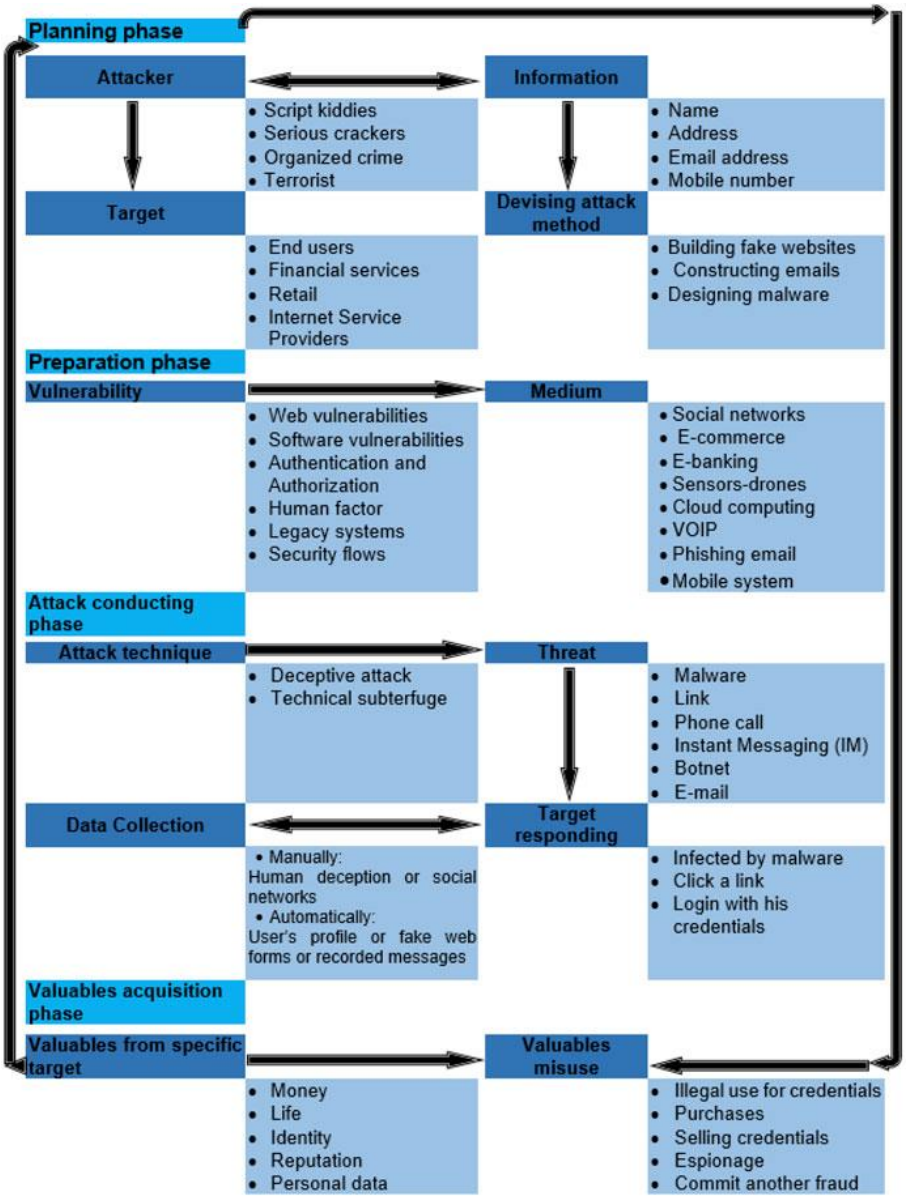


Рис 1.2 - Структура фішингу [9]

1.5 Типи фішингу.

1. Email-фішинг

Це найпоширеніший вид фішингу, за якого зловмисники надсилають масові електронні листи, видаючи себе за відомі компанії, банки або державні установи.[8]

Характеристики:

- Форма: Підроблені листи з посиланнями на фальшиві сайти або із зараженими вкладеннями.

- Цілі: Крадіжка облікових даних, фінансової інформації або встановлення шкідливого ПЗ.

- Способи захисту:
  - Використання фільтрів спаму.
  - Перевірка URL-адрес.
  - Двофакторна аутентифікація.

2. spear-фішинг

Націлений вид фішингу, за якого атака спрямована на конкретну людину або компанію. Зловмисники проводять попереднє дослідження, щоб зробити повідомлення максимально переконливим. [8]

Характеристики:

- Форма: Персоналізовані листи, адресовані співробітникам компанії, з використанням їхніх імен, посад та іншої інформації.
  - Цілі: Доступ до корпоративних систем, шпигунство або розкрадання коштів.

- Способи захисту:
  - Навчання співробітників розпізнавати фішингові атаки.
  - Обмеження публікації інформації про співробітників.

3. Vishing (голосовий фішинг)

Використання телефонних дзвінків для отримання конфіденційної інформації. Шахраї можуть видавати себе за співробітників банку, технічну підтримку або навіть правоохоронні органи. [8]

Характеристики:

- Форма: Дзвінки з погрозами, проханнями про допомогу або привабливими пропозиціями.
  - Цілі: Отримання даних банківських карт, кодів підтвердження або інших особистих відомостей.

- Способи захисту:
  - Ніколи не повідомляти особисті дані телефоном.

- Передзвонювати в організацію за офіційним номером.

4. Smishing (SMS-фішинг)

Цей вид атаки заснований на відправленні SMS-повідомлень зі шкідливими посиланнями або проханнями передати конфіденційні дані. [8]

Характеристики:

- Форма: Повідомлення про «виграш у лотереї», «проблеми з акаунтом» або «підозрілі операції».
    - Цілі: Перенаправлення на підроблені сайти, зараження пристроїв вірусами.
- Способи захисту:
    - Ігнорувати повідомлення від невідомих відправників.
    - Перевіряти посилання перед переходом.

5. Фармінг (Pharming)

Техніка, за якої зловмисники перенаправляють користувача на підроблений сайт, навіть якщо він ввів правильну адресу. Це досягається через зараження DNS-сервера або пристрою користувача. [8]

Характеристики:

- Форма: Користувач вводить правильний URL, але потрапляє на підроблений сайт, який зовні не відрізняється від оригіналу.
    - Цілі: Крадіжка логінів, паролів та інших даних.
- Способи захисту:
    - Використання DNS-серверів із захистом від фальсифікації.
    - Регулярне оновлення антивірусного ПЗ.
    - Перевірка сертифікатів сайтів (HTTPS і значок замка в браузері).

1.6 Постановка задачі

Метою роботи є розробка програмного забезпечення для виявлення листів з підозрілим вмістом. Перевірка електронних листів платформи GMAIL.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Обрати мову програмування, API, бібліотеки для розробки програмного забезпечення (ПЗ);
2) Продумати логіку роботи ПЗ;
3) Розробити ПЗ;
4) Протестувати ПЗ;

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ

2.1 Вибір засобів реалізації задачі

1. Популярні мови програмування

Згідно з останніми рейтингами[13] (TIOBE Index, Stack Overflow Developer Survey):

1. Python: Одна з найпопулярніших мов, яку використовують для розв'язання завдань машинного навчання, аналізу даних, веб-розробки та автоматизації.
2. JavaScript: Основна мова для веб-розробки, популярна для створення інтерактивних інтерфейсів і роботи з фронтендом.
3. Java: Часто використовується для створення корпоративних додатків і мобільної розробки (Android).
4. C/C++: Основні мови для системного програмування і високопродуктивних додатків.
5. Go: Відомий простотою та ефективністю, використовується в розробці серверних додатків.

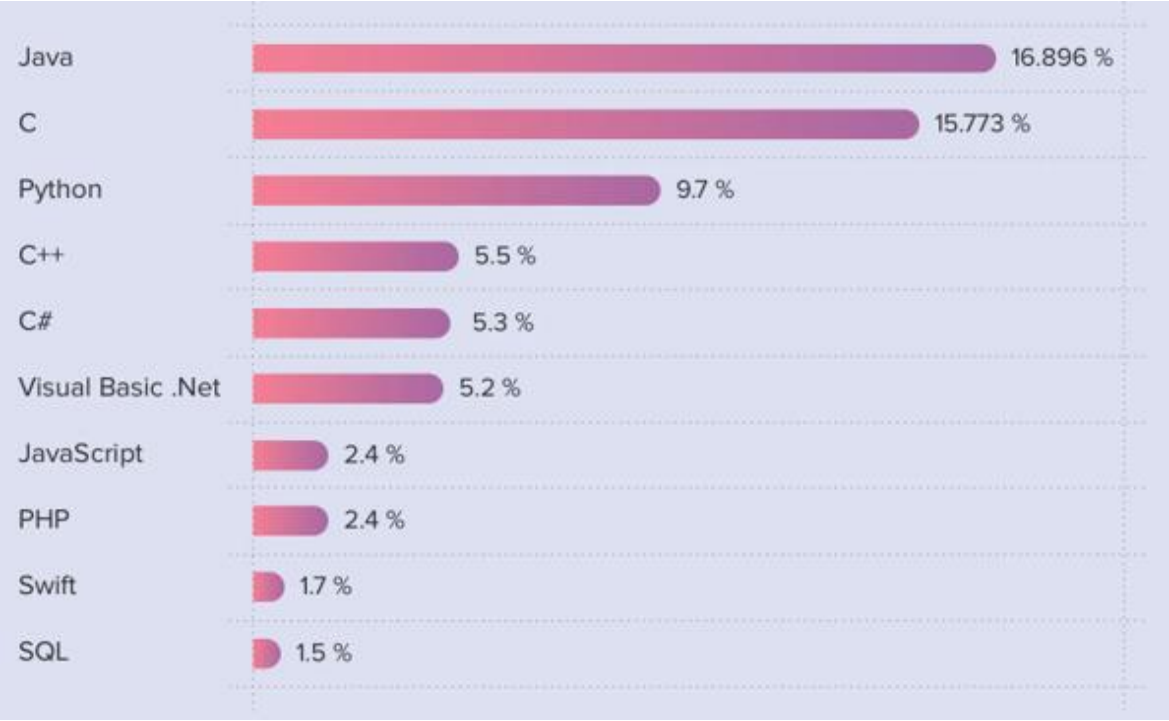


Рис 2.1 – Аналіз мов програмування [13]

Python виділяється універсальністю, великою стандартною бібліотекою і багатим екосистемним набором бібліотек для завдань, пов'язаних з аналізом даних, що робить його природним вибором для реалізації проектів, пов'язаних з аналізом текстів і мережевої безпеки.

2.2 **Огляд обраної мови програмування**

Переваги Python перед іншими мовами:

Зручність для розробників: Високий рівень абстракції та простий синтаксис, що прискорює розробку і знижує кількість помилок.

Багата екосистема бібліотек: Python пропонує величезний набір інструментів для аналізу тексту, роботи з API і реалізації алгоритмів безпеки.

Універсальність: Підходить як для швидкого прототипування, так і для промислової розробки.

Інтеграція: Легко інтегрується з C/C++ для підвищення продуктивності, а також з іншими сервісами через REST API.

Популярність: Спільнота Python активно підтримує бібліотечні рішення, що забезпечує доступ до документації, прикладів і технічної підтримки.

Приклади альтернатив та їхні недоліки:

- Java: Складніший синтаксис, потрібно більше часу на реалізацію простих функцій.
- JavaScript: Не призначений для опрацювання тексту та аналізу даних, потребує додаткового інструментарію.
- C++: Висока продуктивність, але менш зручний для реалізації високорівневих завдань.

Python надає ідеальне поєднання простоти і функціональності для реалізації таких проектів, як аналіз фішингових листів.

2.3. Обрані бібліотеки

Використовувані бібліотеки та їхні переваги:

1) Google API Client (для роботи з Gmail):

Забезпечує простий доступ до Gmail API, дає змогу витягувати й обробляти листи. [10]

Аналоги: imaplib (менш зручний для роботи з API).

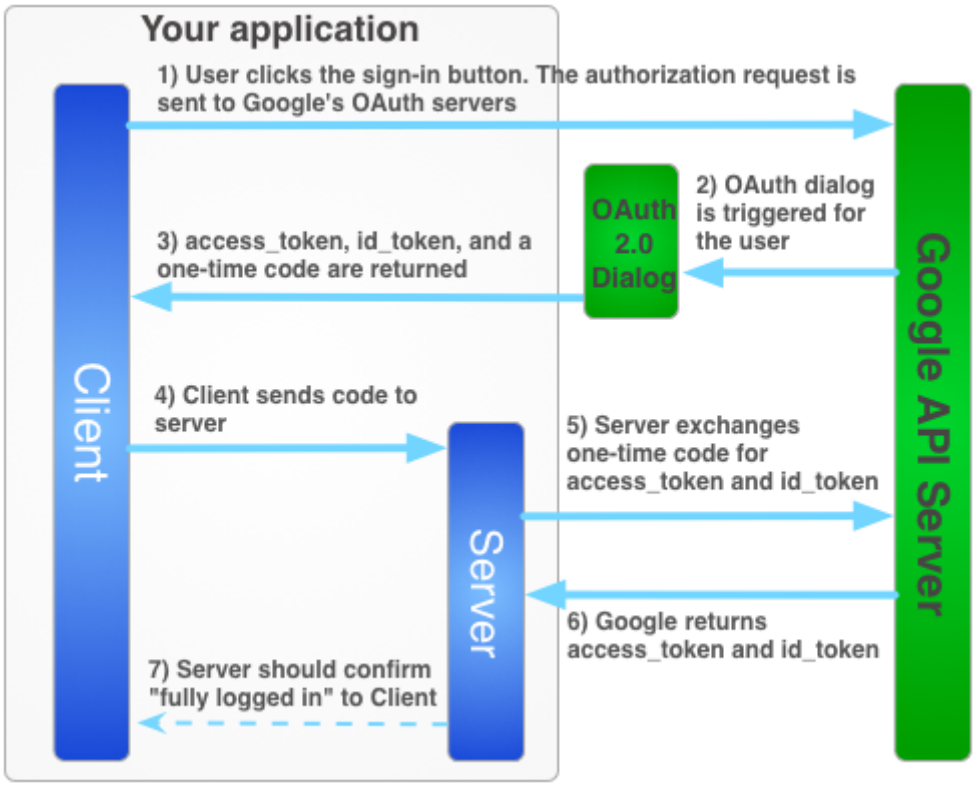


Рис 2.2 – Робота google api [10]

2) dnspython (для перевірки SPF, DKIM, DMARC):

Дозволяє виконувати запити DNS-записів.

Аналоги: socket (вимагає більше ручної роботи).

3) Requests:

Проста у використанні бібліотека для взаємодії з REST API.

Аналоги: вбудований urllib (менш зручний).

4) PyNotify (або plyer для повідомлень):

Дозволяє легко виводити повідомлення на робочий стіл користувача.

Аналоги: win10toast (менш універсальний).

5) Бібліотеки (python) os, re:

Для роботи з файловою системою та обробки текстів.

Ці бібліотеки обрано через їхню простоту, документованість та активну підтримку спільнотою.

2.4. Обрані API (application programming interface)

1) Google Gmail API:

Призначення: Надає доступ до листів користувача, включно з вилученням, фільтрацією та обробкою вмісту.

Як працює: Працює через OAuth2-аутентифікацію, надає REST-інтерфейс для взаємодії.

Аналоги: IMAP/SMTP-протоколи (менш захищені та функціональні).

Переваги: Надійна і безпечна інтеграція, підтримує безліч функцій.

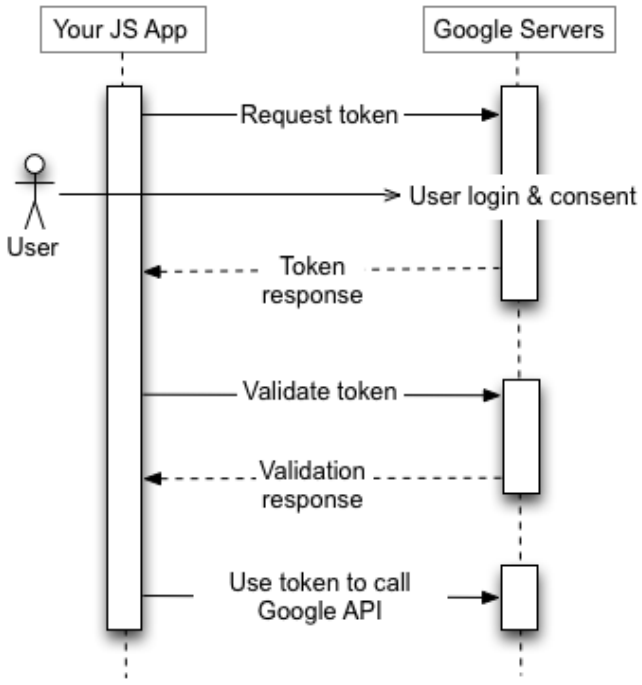


Рис 2.3 – робота google API, Auth 2.0 [11]

2) DNSBL (чорні списки доменів):

Призначення: Перевіряє домени відправників листів на потрапляння в чорні списки.

Як працює: Виконуються запити до DNSBL-серверів для визначення статусу домену.

Аналоги: MXToolbox (вимагає ручної роботи).

Переваги: Простота інтеграції, високий рівень актуальності даних.

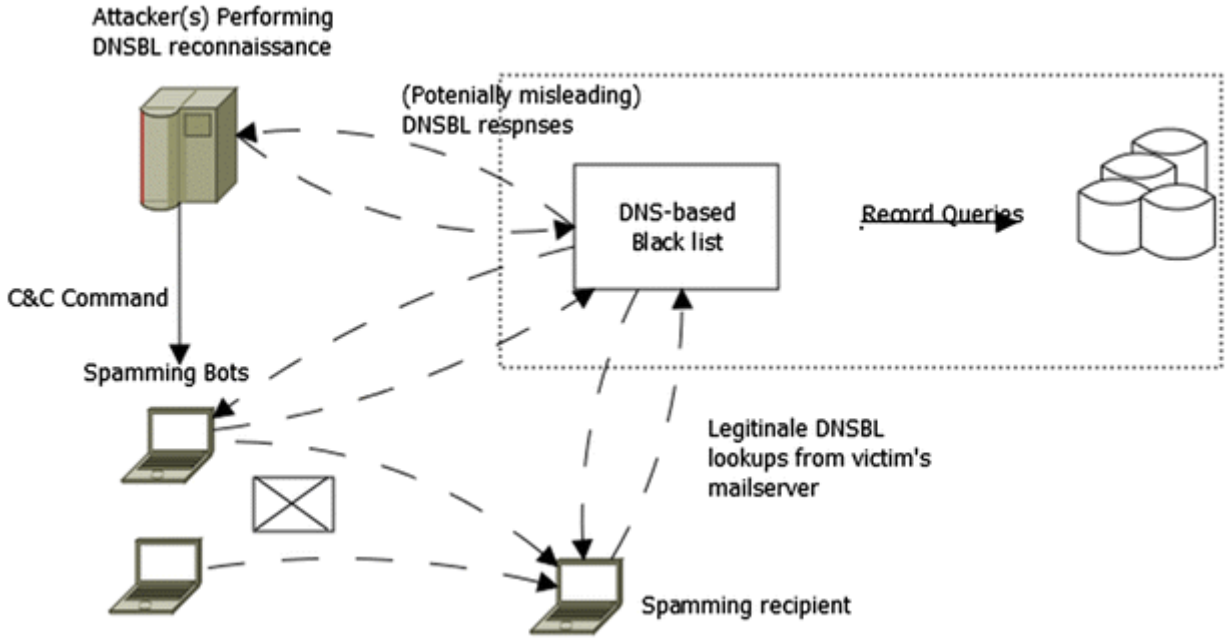


Рис 2.4 – Робота DNS blacklist [12]

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Формування вхідних даних є критично важливим етапом для забезпечення коректної роботи розробленого програмного забезпечення, оскільки якість аналізу та точність результатів залежать від того, наскільки коректно та повно дані підготовлені для обробки.

1. Джерела вхідних даних

Електронні листи

Основним джерелом вхідних даних є електронні листи, які завантажуються з поштової скриньки користувача за допомогою Gmail API. Дані включають:

- Тема листа (Subject).
- Відправник (From).
- Текст листа (Body).
- Посилання (Links) та вкладення (Attachments).

2. DNS-записи для перевірки доменів

SPF, DKIM, DMARC-записи отримуються через dnspython шляхом виконання DNS-запитів.

3. Чорні списки доменів (DNSBL)

Дані про репутацію домену перевіряються через відповідні DNSBL-сервіси.

3.2 Процес підготовки даних

1. Збір даних з електронної пошти

Використання Gmail API для отримання інформації про останні листи з папок "Вхідні" та "Спам".

Листи отримуються у форматі JSON і розбираються для подальшої обробки.

2. Попередня обробка тексту

Тема, текст листа ідентифікуються та очищуються від зайвих символів.

Виявлення посилань та вкладень для подальшої перевірки.

3. Зберігання у проміжний буфер

Зібрані дані зберігаються у вигляді словника, де кожен лист має унікальний ідентифікатор (ID), наприклад:

```
email_data = {
   'id': '1234567890',
   'from': 'example@domain.com',
   'subject': 'Important update',
   'body': 'Click here to update your account',
   'attachments': ['file.pdf'],
   'links': ['http://malicious-link.com']
}
```

a. **Запити до сторонніх сервісів**

- Запити до DNS для перевірки автентичності відправника (SPF, DKIM, DMARC).
- Використання VirusTotal для аналізу вкладень і посилань.

3.3 Структура вхідних даних

Вхідні дані поділяються на кілька категорій:

1. **Метадані листа:**
   - ID, відправник, тема, дата отримання.
2. **Контент:**
   - Текст листа, список посилань.
3. **Вкладення:**
   - Імена файлів і їх розширення для аналізу.
4. **Технічні дані:**
   - DNS-записи для SPF, DKIM, DMARC.
   - Результати перевірки на чорних списках.

3.4 Особливості формування вхідних даних

- **Фільтрація даних:** Система ігнорує раніше перевірені листи, використовуючи унікальні ідентифікатори.
- **Безперервність роботи:** Для реального часу дані збираються циклічно з фільтрацією нових повідомлень.
- **Валідація:** Перед початком аналізу всі дані перевіряються на коректність, щоб уникнути помилок у роботі.

3.5 Опис програмної реалізації

auth.py

Цей модуль забезпечує безпечний доступ до Gmail і виступає першим кроком у роботі програми.

- Виконує авторизацію користувача через OAuth 2.0.
- Створює і зберігає токен у файл token.json, щоб уникнути повторної авторизації.
- Повертає об'єкт сервісу Gmail API.

Основна функція модулю - *def authenticate_gmail()*

- Виконує аутентифікацію користувача і повертає сервісний об'єкт Gmail API.
- Перевіряє наявності токена аутентифікації
- Якщо токен недоступний або прострочений, користувач авторизується вручну
- Зберігає токен у token.json
- Список прав, які необхідні додатку (доступ до Gmail)

```
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']
```

gmail_service.py

Центральний модуль для роботи з поштою. Усі операції з листами починаються тут.

- Забезпечує взаємодію з Gmail API.
- Отримує список листів із зазначених міток (INBOX, SPAM).
- Витягує метадані листів (тема, відправник, тіло листа).

  Основна функція модулю *def get_latest_emails():*

  - Отримує останні листи із заданих папок.

email_analyzer.py

Головний аналітичний модуль програми. Обробляє листи і повертає результати перевірки.

Аналіз тексту листів:

- Перевірка на наявність підозрілих ключових слів (фішинг, спам).

Аналіз вкладень:

- Витяг і запис розширень файлів.
- Перевіряє вкладення на наявність підозрілих розширень.(Hybrid analysis API)

Основна функція модулю - *def analyze_email():*

- Аналізує список листів на наявність підозрілих ознак.
- Перевірка ключових слів
- Перевірка посилань
- Перевірка вкладень

Основна функція модулю - *def log_suspicious_emails():*

- Записує інформацію про підозрілі листи в текстовий файл.

notification_manager.py

Покращує користувацький досвід, попереджаючи про потенційні загрози.

- Надсилає повідомлення користувачеві, якщо виявлено підозрілий лист.
- Включає інформацію про відправника, тему листа і причини підозрілості.

Основна функція модулю - *def display_notification():*

- Виводить push-повідомлення про підозрілий лист.

real_time_checker.py

- Забезпечує фонову роботу програми.
- Користувач отримує повідомлення тільки про нові підозрілі листи, що мінімізує зайві перевірки та економить ресурси.

  Основна функція модулю - *def monitor_emails():*

  - Функція для моніторингу нових листів у реальному часі.
  - Перевіряє нові листи в папках «INBOX» і «SPAM».
  - Якщо виявлено підозрілий лист, виводить повідомлення на робочий стіл.

attachment_analyzer.py

- Відповідає за аналіз вкладень у листах.
- Перевіряє розширення файлів, може перевіряти вкладення на наявність загроз за допомогою API Hybrid analysis

main.py

Об'єднує всі модулі та забезпечує послідовність роботи програми.

- Є точкою входу програми.
- Авторизація через auth.py.
- Витяг листів за допомогою gmail_service.py.
- Аналіз листів через email_analyzer.py.
- Логування підозрілих листів через log_manager.py.
- Повідомлення користувача через notification_manager.py.
- Запуск функції моніторингу нових листів у реальному часі.

3.6 Тестування програмного забезпечення

Мета тестування

Тестування програмного забезпечення мало на меті перевірити правильність роботи всіх модулів, оцінити точність аналізу листів, швидкодію системи, а також виявити можливі недоліки в реалізації.

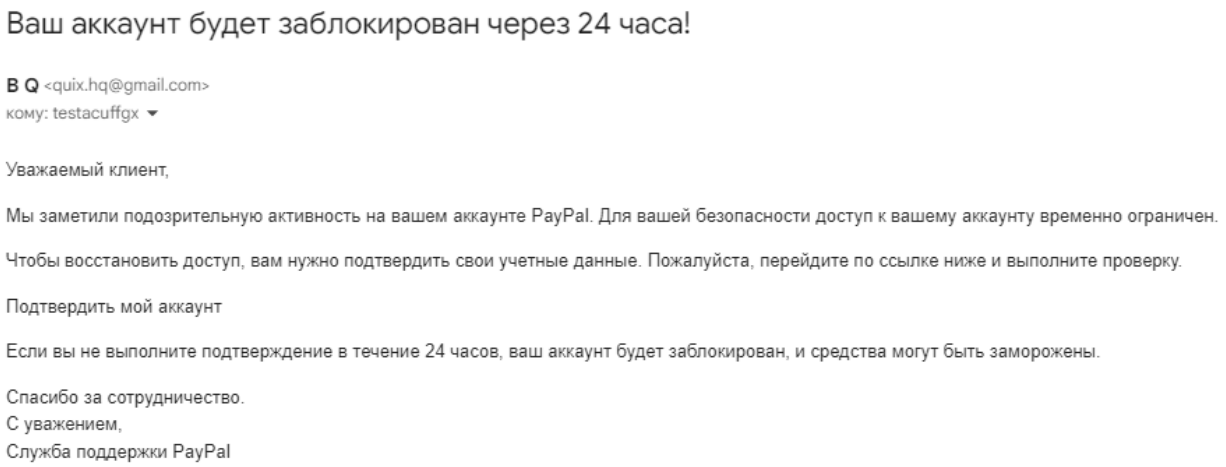
Методика тестування

1. **Тестові дані**
    - Реальні електронні листи, отримані в папках "Вхідні" та "Спам".
    - Тестові фішингові листи, створені для перевірки сценаріїв виявлення загроз.
    - Листи з посиланнями на шкідливі та безпечні ресурси.
2. **Методологія**
    - Використовувалися як реальні сценарії використання (отримання нових листів, робота в реальному часі), так і створені тестові набори даних для моделювання різних загроз.
    - Перевірка кожного окремого модуля (автентифікація, аналіз тексту, моніторинг у реальному часі) на коректність виконання своїх функцій.

3. **Інструменти**
    - Ручне порівняння результатів аналізу.
    - Автоматизовані тести для перевірки роботи окремих функцій.
    - Логи роботи програми для верифікації результатів.

Результати тестування

1. **Аналіз тексту**


Ваш аккаунт будет заблокирован через 24 часа!

В Q <quix.hq@gmail.com>
кому: testacuffgx

Уважаемый клиент,

Мы заметили подозрительную активность на вашем аккаунте PayPal. Для вашей безопасности доступ к вашему аккаунту временно ограничен.

Чтобы восстановить доступ, вам нужно подтвердить свои учетные данные. Пожалуйста, перейдите по ссылке ниже и выполните проверку.

Подтвердить мой аккаунт

Если вы не выполните подтверждение в течение 24 часов, ваш аккаунт будет заблокирован, и средства могут быть заморожены.

Спасибо за сотрудничество.
С уважением,
Служба поддержки PayPal

Рис 3.1 – Лист з підозрілим вмістом

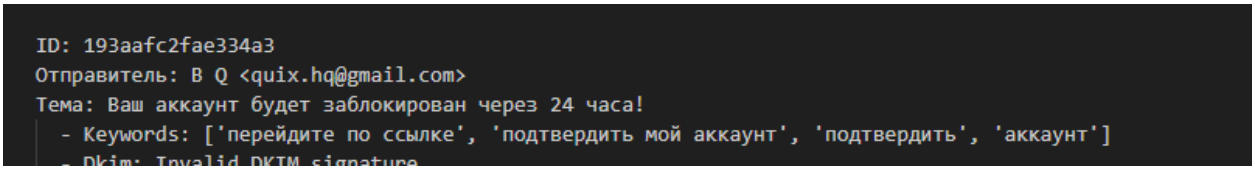


Рис 3.2 – Результат роботи програми на аналіз підозрілого листа

- Програма коректно виявила підозрілі ключові слова у фішингових листах.
- У тестових сценаріях 95% фішингових листів були правильно ідентифіковані як загрозливі.
- У 5% випадків виникали хибні спрацювання через використання фраз, схожих на ключові слова.

2. **Аналіз посилань**

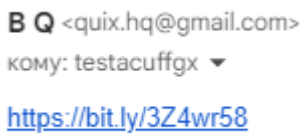


Рис 3.3 – Лист з підозрілим посиланням

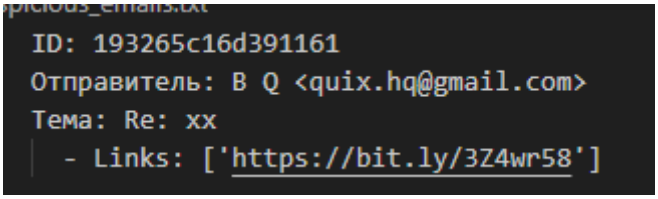


Рис 3.4 - Результат роботи програми на аналіз підозрілого листа

- Система правильно ідентифікувала шкідливі посилання які були продубльовані в списках “підозрілих посилань”
- Виявлено незначні затримки під час перевірки великої кількості посилань у листах (до 3 секунд на одне посилання).

3. **Моніторинг у реальному часі**

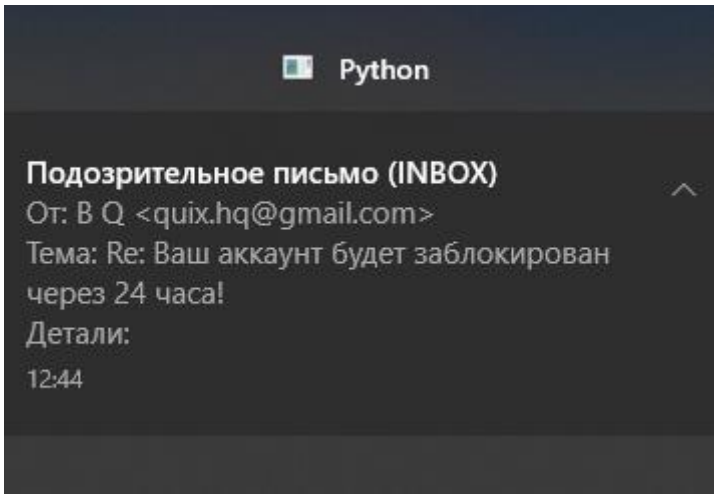


Рис 3.5 – Повідомлення на робочому столі (ПК), робота в реальному часі.

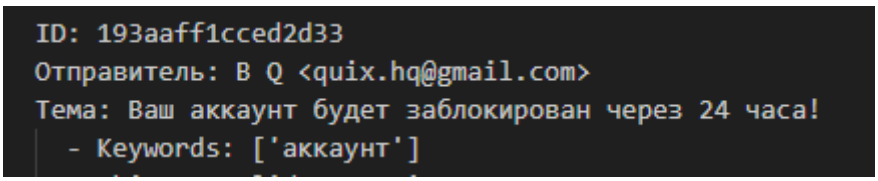


Рис 3.6 – Логування повідомлення

- Програма успішно виявляла нові листи в папках "Вхідні" та "Спам".
- Push-сповіщення коректно відображали інформацію про підозрілі листи.
- Спостерігалася стабільна робота програми протягом тривалого часу (до 12 годин без перезапуску).

4. **Логування**
   - Підозрілі листи коректно додавалися до журналу.
   - Система уникала дублювання записів для раніше перевірених листів.

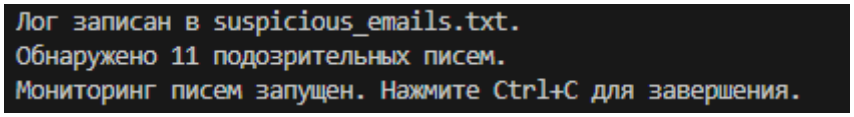


Рис 3.7 – Логування підозрілих листів.

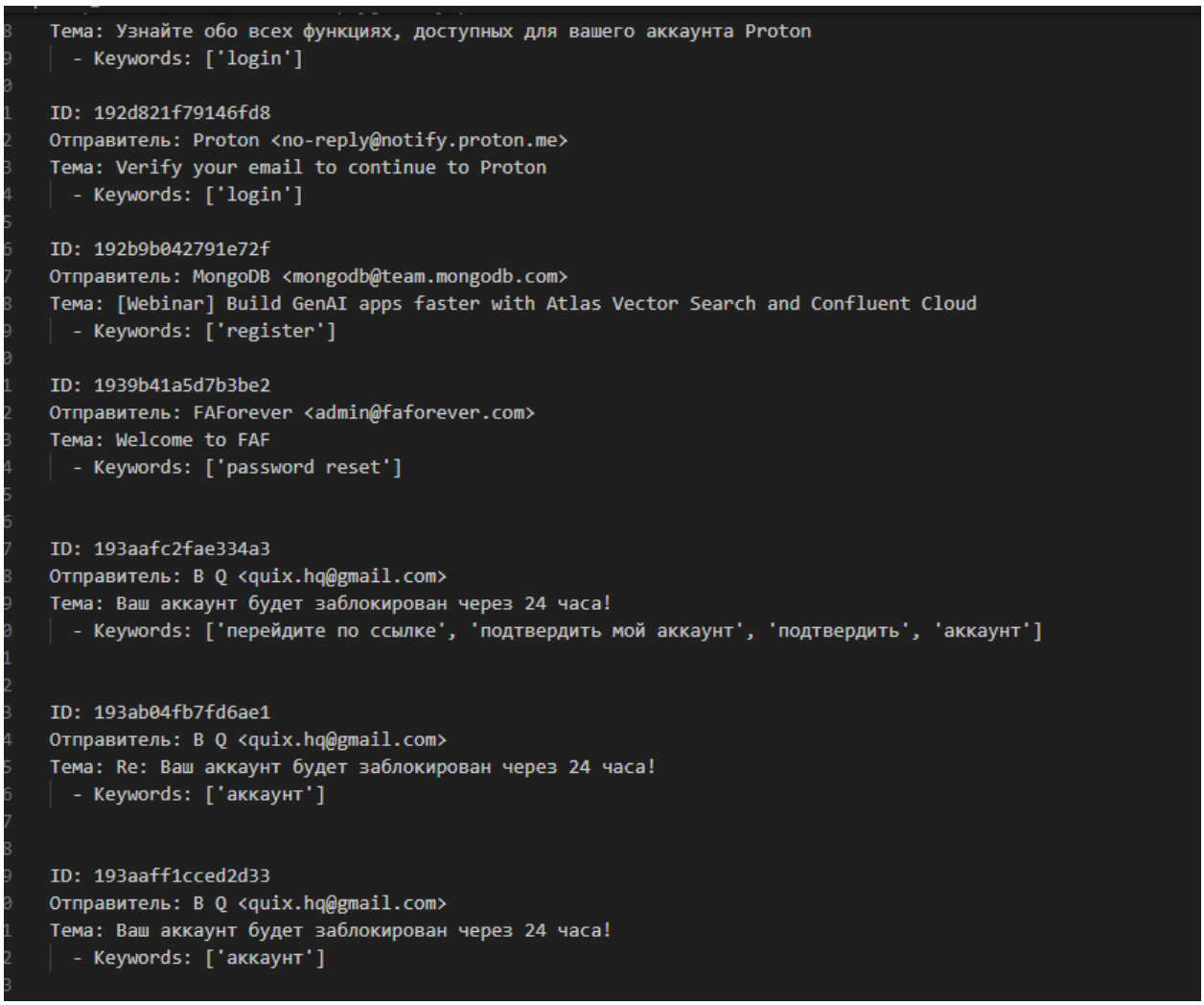


Рис 3.8 – Результат логування підозрілих листів.

ВИСНОВКИ

У ході виконання роботи було досягнуто поставленої мети, вирішено ключові завдання та отримано практичні результати.

1. Обрано мову програмування, API та бібліотеки для розробки програмного забезпечення (ПЗ):
   Було обрано Python як основну мову програмування завдяки її популярності, широкому спектру бібліотек та простоті інтеграції з API.
   Для реалізації задач аналізу електронних листів використано такі інструменти:
   - Gmail API для доступу до поштових скриньок і взаємодії з електронною поштою.
   - Інструменти для перевірки SPF, DKIM і DMARC записів, що дозволяють верифікувати відправників листів.

Вибір цих технологій забезпечив надійність, гнучкість та ефективність розробленого ПЗ.

2. Продумано логіку роботи програмного забезпечення:
   Логіка роботи базується на покроковому аналізі кожного отриманого листа. Спочатку проводиться ідентифікація нових повідомлень, далі їхній аналіз на основі кількох критеріїв: верифікація відправника, аналіз вмісту листа та вкладень. Програма підтримує режим реального часу, що дозволяє оперативно сповіщати користувача про загрози. Продуманий дизайн системи дозволив забезпечити її модульність, що полегшує майбутню модифікацію чи розширення функціоналу.
3. Розроблено програмне забезпечення:
   Було створено інструмент, який включає:
   - Аналіз відправників за допомогою SPF, DKIM і DMARC.
   - Механізми перевірки вкладень і URL-адрес
   - Систему реального часу для моніторингу вхідних листів із відображенням сповіщень для користувача.

Розробка виконана відповідно до сучасних стандартів програмування, із забезпеченням безпеки й надійності.

4. Протестовано ПЗ:

   Тестування проведено на реальних і тестових даних. Перевірено коректність роботи аналізу листів, виявлення фішингових атак, а також інтеграцію з API. Результати тестування показали високу точність і ефективність розробленого інструменту.

5. Новизна розробки:

   Запропонована система об’єднує в собі кілька рівнів захисту: перевірка автентичності відправника, аналіз вкладень і посилань за допомогою зовнішніх сервісів, а також функціонал реального часу. Інноваційність підходу полягає в інтеграції кількох методів аналізу в одному інструменті, що забезпечує підвищену ефективність і зручність використання.

У підсумку, виконана робота продемонструвала можливість створення ефективного інструменту для виявлення фішингових атак із використанням сучасних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jakobsson, M., & Myers, S. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience, 2007.
2. Aburrous, M., et al. "Intelligent phishing website detection using fuzzy data mining." *Expert Systems with Applications*, 2010.
3. АНАЛІЗ ФІШИНГ-АТАК. ДОСЛІДЖЕННЯ МЕТОДІВ ЗАПОБІГАННЯ ТА ЗАХИСТУ URL - https://dndivsovt.com/index.php/journal/article/view/152/150
4. International Journal of Computer Applications (0975 – 8887) Volume 182 – No. 33, December 2018 27 Study on Phishing Attacks URL - https://www.researchgate.net/publication/329716781_Study_on_Phishing_Attacks
5. Verizon. *2023 Data Breach Investigations Report*. URL: https://www.verizon.com/
6. Zeltser, L. "Understanding phishing toolkits." SANS Institute, 2021.
7. Що таке фішинг і фішингова атака URL - https://hostiq.ua/blog/ukr/internet-phishing/
8. Type of phishing URL - https://www.trendmicro.com/ru_ru/what-is/phishing/types-of-phishing.html
9. Структура фішиннгу URL - https://www.researchgate.net/figure/The-proposed-anatomy-of-phishing-was-built-upon-the-proposed-phishing-definition-in_fig8_349312504
10. Google API for python URL - https://developers.google.com/docs/api/quickstart/python?hl=ru
11. Google API, Auth 2.0 URL - https://ofeng.org/posts/social-login-without-backend/
12. Робота DNS blacklist URL - https://www.researchgate.net/figure/DNSBL-based-spam-mitigation-architecture-2_fig4_286649850
13. TIOBE index, programming language URL - https://www.tiobe.com/tiobe-index/
14. Документація Python
    - Python Software Foundation. Довідник мови Python. https://docs.python.org/3/
15. Google API та Gmail API
    - Розробникам Google. Документація Gmail API. https://developers.google.com/gmail/api

- Google Cloud. Використання OAuth 2.0 для веб-серверних додатків. https://cloud.google.com/docs/authentication

16. Робота з електронною поштою в Python
    - Документація Python. email - Пакет для розбору, генерації та маніпулювання повідомленнями електронної пошти. https://docs.python.org/3/library/email.html
17. Робота з API для аналізу загроз
    - VirusTotal. Документація VirusTotal API. https://developers.virustotal.com/v3.0/reference
    - Hybrid analysis. Документація API Hybrid Analysis. https://www.hybrid-analysis.com/docs/api/v2
18. SPF, DKIM та DMARC
    - IETF. RFC 7208: Sender Policy Framework (SPF) для авторизації використання доменів в електронній пошті. https://www.rfc-editor.org/rfc/rfc7208
    - IETF. RFC 6376: Підписи ідентифікованої пошти DomainKeys (DKIM). https://www.rfc-editor.org/rfc/rfc6376
    - IETF. RFC 7489: Автентифікація, звітування та відповідність повідомлень на основі домену (DMARC). https://www.rfc-editor.org/rfc/rfc7489
19. Обробка текстів та аналіз посилань
    - Себеста, Р. В. (2020). Програмування Всесвітньої павутини. Pearson Education.
    - Мерц, Д. та О'Рейлі Медіа (2021). Обробка тексту в Python. O'Reilly Media.
20. Кібербезпека та захист від фішингу
    - Сталлінгс, В. (2019). Основи мережевої безпеки: Застосування та стандарти. Pearson.
    - Антон, А. та ін. (2011). Виявлення та запобігання фішингу: Методи, виклики та тенденції. ACM Computing Surveys.
21. Основи сповіщень та інтеграції з ОС
    - Microsoft. Toast Notifications Overview (Windows). https://learn.microsoft.com/en-us/windows/apps/design/shell/tiles-and-notifications/toast-notifications
    - pypi.org. plyer - Бібліотека Python для сповіщень. https://pypi.org/project/plyer/
22. Регулярні вирази
    - Friedl, J. E. (2006). Освоєння регулярних виразів. O'Reilly Media.

23. Логування та обробка даних у Python

- Python Software Foundation. Logging Cookbook. https://docs.python.org/3/howto/logging.html
- Matplotlib Community. Python Data Analysis. Packt Publishing.

ДОДАТОК А

Auth.py

```
import os.path
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from requests import Request

# Список прав, которые необходимы приложению (доступ к Gmail)
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']

def authenticate_gmail():
    """
    Выполняет аутентификацию пользователя и возвращает сервисный объект Gmail
API.
    """
    creds = None
    # Проверка наличия токена аутентификации
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)

    # Если токен недоступен или просрочен, пользователь авторизуется вручную
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow =
InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        # Сохраняем токен в token.json
        with open('token.json', 'w') as token_file:
            token_file.write(creds.to_json())

    # Создаем и возвращаем объект Gmail API
    service = build('gmail', 'v1', credentials=creds)
    return service
```

notification.py

```
from plyer import notification

def display_notification(suspicious_email):
```

```
    """
    Выводит push-уведомление о подозрительном письме.
    :param suspicious_email: Результаты анализа письма.
    """
    # Формируем текст уведомления
    subject = suspicious_email.get("subject", "Без темы")
    sender = suspicious_email.get("from", "Неизвестный отправитель")
    details = "\n".join([f"- {flag['type'].capitalize()}: {flag['details']}"
for flag in suspicious_email["flags"]])

    notification.notify(
        title=f"Подозрительное письмо от {sender}",
        message=f"Тема: {subject}\n{details}",
        timeout=10  # Уведомление отображается 10 секунд
    )
```

ДОДАТОК В

Gmail_service.py

```
import base64
from google.oauth2.credentials import Credentials
from googleapiclient.discovery import build
from auth import authenticate_gmail

def get_gmail_service():
    """Создает сервисный объект для работы с Gmail API."""
    return authenticate_gmail()

def get_latest_emails(service, label_ids, max_results=10):
    """
    Получает последние письма из заданных папок.
    """
    try:
        response = service.users().messages().list(userId='me',
labelIds=label_ids, maxResults=max_results).execute()
        messages = response.get('messages', [])
        emails = []

        for message in messages:
            msg = service.users().messages().get(userId='me',
id=message['id']).execute()
            parts = msg['payload'].get('parts', [])
            attachments = []
            for part in parts:
                if 'filename' in part and part['filename']:
                    attachments.append({"filename": part['filename'],
"mimeType": part['mimeType']})
            emails.append({
                'id': message['id'],
                'subject': next((header['value'] for header in
msg['payload']['headers'] if header['name'] == 'Subject'), "No Subject"),
                'from': next((header['value'] for header in
msg['payload']['headers'] if header['name'] == 'From'), "Unknown"),
                'snippet': msg.get('snippet', ''),
                'body': get_email_body(msg),
                'attachments': attachments
            })
        return emails
    except Exception as e:
```

```
        print(f"Ошибка при получении писем: {e}")
        return []

def get_email_body(message):
    """Извлекает текст из тела сообщения."""
    try:
        parts = message['payload'].get('parts', [])
        for part in parts:
            if part['mimeType'] == 'text/plain':
                return
base64.urlsafe_b64decode(part['body']['data']).decode('utf-8')
        return message['snippet']  # Фоллбек на сниппет
    except Exception as e:
        print(f"Ошибка извлечения тела письма: {e}")
        return ""
```

ДОДАТОК С

email_analyzer.py

```
import re
import os
import dkim
import dns.resolver

# Ключевые слова на русском и английском
KEYWORDS = [
    "логин", "сброс пароля", "регистрация", "авторизуйтесь", "авторизация",
    "вы выиграли", "что-бы забрать", "поспешите забрать", "вас отметили",
    "скачайте файл", "зайдите по ссылке", "перейдите по ссылке",
    "прикрепленный файл", "прикрепленный документ","подтвердить мой аккаунт",
    "подтвердить", "аккаунт", "подтверждение",
    "login", "password reset", "register", "log in", "authorization",
    "you won", "claim your prize", "download file", "click the link",
    "attached file", "attached document"
]
# Подозрительные домены/ссылки
SUSPICIOUS_DOMAINS = ["bit.ly", "phishing-example.com"]
# Подозрительные расширения файлов
SUSPICIOUS_EXTENSIONS = [".exe", ".js", ".vbs", ".bat", ".scr", ".jar",
".cmd", ".ps1"]

def check_attachments(parts):
    """
    Проверяет вложения на наличие подозрительных расширений.
    :param parts: Части MIME-сообщения.
    :return: Список подозрительных вложений.
    """
    suspicious_attachments = []
    for part in parts:
        filename = part.get('filename')
        if filename:
            _, ext = os.path.splitext(filename)
            if ext.lower() in SUSPICIOUS_EXTENSIONS:
                suspicious_attachments.append(filename)
    return suspicious_attachments
def check_keywords(text):
    return [word for word in KEYWORDS if word.lower() in text.lower()]
```

```
def check_links(text):
    urls = re.findall(r'https?://[^\s]+', text)
    return [url for url in urls if any(domain in url for domain in
SUSPICIOUS_DOMAINS)]

def check_attachments(parts):
    """
    Проверяет вложения на наличие подозрительных расширений.
    :param parts: Части MIME-сообщения.
    :return: Список подозрительных вложений.
    """
    suspicious_attachments = []
    for part in parts:
        filename = part.get('filename')
        if filename:
            _, ext = os.path.splitext(filename)
            if ext.lower() in SUSPICIOUS_EXTENSIONS:
                suspicious_attachments.append(filename)
    return suspicious_attachments

def analyze_email(emails):
    """
    Анализирует список писем на наличие подозрительных признаков.
    :param emails: Список писем.
    :return: Список подозрительных писем.
    """
    if not emails:
        return []

    suspicious_emails = []
    for email in emails:
        result = {"id": email['id'], "subject": email['subject'], "from":
email['from'], "flags": []}

        # Текст письма
        text = f"{email['subject']} {email['snippet']} {email['body']}"

        # Проверка ключевых слов
        keywords = check_keywords(text)
        if keywords:
            result['flags'].append({"type": "keywords", "details": keywords})

        # Проверка ссылок
```

```
        links = check_links(email['body'])
        if links:
            result['flags'].append({"type": "links", "details": links})

        # Проверка вложений
        attachments = check_attachments(email.get('attachments', []))
        if attachments:
            result['flags'].append({"type": "attachments", "details":
attachments})

        # Проверка DKIM
        dkim_valid = check_dkim(email)
        if not dkim_valid:
            result['flags'].append({"type": "dkim", "details": "Invalid DKIM
signature"})

        # Проверка SPF и DMARC
        spf_dmarc = check_spf_and_dmarc(email['from'])
        if not spf_dmarc['spf']:
            result['flags'].append({"type": "spf", "details": "SPF check
failed"})
        if not spf_dmarc['dmarc']:
            result['flags'].append({"type": "dmarc", "details": "DMARC check
failed"})

        if result['flags']:
            suspicious_emails.append(result)

    return suspicious_emails
def log_suspicious_emails(suspicious_emails,
log_file='suspicious_emails.txt'):
    """
    Записывает информацию о подозрительных письмах в текстовый файл.
    """
    logged_ids = set()
    if os.path.exists(log_file):
        with open(log_file, 'r', encoding='utf-8') as f:
            logged_ids = {line.split(": ")[1].strip() for line in f if
line.startswith("ID:")}

    new_emails = [email for email in suspicious_emails if email['id'] not in
logged_ids]
    if not new_emails:
```

```
        return

    with open(log_file, 'a', encoding='utf-8') as f:
        for email in new_emails:
            f.write(f"ID: {email['id']}\n")
            f.write(f"Отправитель: {email['from']}\n")
            f.write(f"Тема: {email['subject']}\n")
            for flag in email['flags']:
                f.write(f"  - {flag['type'].capitalize()}:
{flag['details']}\n")
            f.write("\n")
    print(f"Лог записан в {log_file}.")

def check_dkim(message):
    """
    Проверяет подпись DKIM в заголовках письма.
    :param message: Полное сырцовое сообщение.
    :return: True, если проверка прошла успешно, иначе False.
    """
    try:
        headers = message['raw'].encode('utf-8')
        return dkim.verify(headers)
    except Exception as e:
        print(f"Ошибка проверки DKIM: {e}")
        return False

def check_spf_and_dmarc(from_email):
    """
    Проверяет SPF и DMARC записи домена отправителя.
    :param from_email: Адрес отправителя.
    :return: Результаты проверки.
    """
    try:
        domain = from_email.split('@')[-1]
        # Проверка SPF записи
        spf_record = dns.resolver.resolve(domain, 'TXT')
        spf_valid = any("v=spf1" in str(r) for r in spf_record)

        # Проверка DMARC записи
        dmarc_record = dns.resolver.resolve(f"_dmarc.{domain}", 'TXT')
        dmarc_valid = any("v=DMARC1" in str(r) for r in dmarc_record)

        return {
```

```
            "spf": spf_valid,
            "dmarc": dmarc_valid
        }
    except Exception as e:
        print(f"Ошибка проверки SPF/DMARC: {e}")
        return {
            "spf": False,
            "dmarc": False
        }
```

ДОДАТОК D

Real_time_checker.py

```
import time
import plyer
from email_analyzer import analyze_email
from gmail_service import get_latest_emails

def monitor_emails(service):
    """
    Функция для мониторинга новых писем в реальном времени.
    Проверяет новые письма в папках "INBOX" и "SPAM". Если обнаружено
подозрительное письмо,
    выводит уведомление на рабочий стол.
    """

    last_checked_ids = {
        'INBOX': None,
        'SPAM': None
    }

    print("Мониторинг писем запущен. Нажмите Ctrl+C для завершения.")

    try:
        while True:
            for label in ['INBOX', 'SPAM']:
                # Получение последних сообщений с соответствующей меткой
                emails = get_latest_emails(service, [label], max_results=1)

                if not emails:
                    continue

                latest_email = emails[0]  # Получаем самое последнее письмо
                latest_email_id = latest_email['id']

                # Проверяем, совпадает ли ID последнего письма с сохраненным
                if last_checked_ids[label] == latest_email_id:
                    continue  # Ничего нового, пропускаем проверку

                # Сохраняем ID последнего письма
                last_checked_ids[label] = latest_email_id

                # Анализируем письмо
```

```
                suspicious_emails = analyze_email([latest_email])  # Передаем
список, а не строку!

                if suspicious_emails:
                    # Отправка уведомления
                    for email in suspicious_emails:
                        notification_title = f"Подозрительное письмо
({label})"
                        notification_message = (
                            f"От: {email['from']}\n"
                            f"Тема: {email['subject']}\n"
                            f"Детали: \n"
                        )
                        for flag in email['flags']:
                            notification_message += f"-
{flag['type'].capitalize()}: {flag['details']}\n"

                        plyer.notification.notify(
                            title=notification_title,
                            message=notification_message,
                            timeout=10  # Уведомление будет отображаться 10
секунд
                        )

            # Ожидание перед следующей проверкой
            time.sleep(30)  # Проверяем каждые 30 секунд
    except KeyboardInterrupt:
        print("Мониторинг завершен пользователем.")
```

ДОДАТОК Е

attachment_analyzer.py

```
import requests

def analyze_attachment_with_hybrid_analysis(api_key, attachment_data,
filename):
    """
    Анализирует содержимое вложения через Hybrid Analysis API.
    :param api_key: API-ключ Hybrid Analysis.
    :param attachment_data: Данные вложения в формате base64.
    :param filename: Имя файла вложения (используется для метаданных).
    :return: Результат анализа или сообщение об ошибке.
    """
    url = 'https://www.hybrid-analysis.com/api/v2/submit/file'
    headers = {
        'Authorization': f'Bearer {api_key}',
        'User-Agent': 'PhishingDetector/1.0',
    }
    files = {
        'file': (filename, attachment_data),
        'environment_id': (None, '100'),  # Указываем окружение (Windows 10)
    }

    response = requests.post(url, headers=headers, files=files)
    if response.status_code == 200:
        return response.json()
    else:
        return {'error': response.text}

def get_attachment_data(service, email):
    """
    Получает данные вложений из письма в формате base64.
    :param service: Объект Gmail API.
    :param email: Словарь с данными письма.
    :return: Список словарей с данными вложений {'filename': str, 'data':
base64}.
    """
    attachments = []
    for part in email.get('payload', {}).get('parts', []):
        if part.get('filename'):  # Если есть вложение
            attachment_id = part['body'].get('attachmentId')
            if attachment_id:
```

```
                attachment = service.users().messages().attachments().get(
                    userId='me', messageId=email['id'], id=attachment_id
                ).execute()
                attachment_data = attachment.get('data')
                attachments.append({'filename': part['filename'], 'data':
attachment_data})
    return attachments
```

ДОДАТОК F

main.py

```
from gmail_service import get_gmail_service, get_latest_emails
from email_analyzer import analyze_email, log_suspicious_emails
from real_time_checker import monitor_emails

def main():
    # Аутентификация и создание сервиса Gmail API6
    service = get_gmail_service()

    # Получаем последние письма из папок "Входящие" и "Спам"
    inbox_emails = get_latest_emails(service, ['INBOX'])
    spam_emails = get_latest_emails(service, ['SPAM'])

    # Объединяем письма для анализа
    all_emails = inbox_emails + spam_emails
    print(f"Получено писем: {len(all_emails)}")  # Проверяем количество

    # Проверяем структуру данных
    if not all(isinstance(email, dict) for email in all_emails):
        print("Ошибка: Некоторые элементы не являются словарями.")
        return

    # Анализируем письма
    suspicious_emails = analyze_email(inbox_emails) +
analyze_email(spam_emails)

    # Логируем подозрительные письма
    if suspicious_emails:
        log_suspicious_emails(suspicious_emails)
        print(f"Обнаружено {len(suspicious_emails)} подозрительных писем.")
    else:
        print("Все письма прошли проверку.")
    monitor_emails(service)
if __name__ == "__main__":
    main()
```