Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Розробка поштового клієнта із захистом від фішингу з використанням VirusTotal API

Виконав: студент 4 курсу, гр.1КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Іванченко О.А.
(прізвище та ініціали)

Керівник: к.т.н., доц., доцент каф. МБІС
Яремчук Ю.Є.
(прізвище та ініціали)
«6» червня 2024 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Крупельницький Л.В.
(прізвище та ініціали)
«6» червня 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“6” червня 2024р.

Вінниця ВНТУ – 2024

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
“ 22 ” березня 2024 р.

З А В Д А Н Н Я
на бакалаврську дипломну роботу студенту
Іванченку Олександру Андрійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка поштового клієнта із захистом від фішингу з використанням VirusTotal API
Керівник роботи Яремчук Юрій Євгенович д.т.н., проф. каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “11” березня 2024року № 80
2. Термін подання студентом роботи 6 червня 2024р
3. Вихідні дані до роботи Електронні джерела та наукові статті по темі. Існуюче програмне забезпечення, яке стосується теми бакалаврської дипломної роботи.
4. Зміст текстової частини: для досягнення мети було визначено наступні завдання: провести дослідження актуальності обраної теми; провести аналіз методів захисту інформації; провести аналіз методів шифрування інформації; оцінити принцип роботи даних методів захисту; оцінити переваги та недоліки аналогічних сервісів; вдосконалити алгоритм роботи системи безпеки поштового клієнта; реалізувати поштовий клієнт; провести тестування розробленого клієнта.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Метод роботи SPF, Метод роботи DKIM, Метод роботи DMARC, шаблон проектування, Use-case діаграма, UML діаграма, діаграма декомпозиції роботи клієнта, інтерфейс програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Яремчук Ю.Є. д.т.н., проф. каф. МБІС		
Розділ II	Яремчук Ю.Є. д.т.н., проф. каф. МБІС		
Розділ III	Яремчук Ю.Є. д.т.н., проф. каф. МБІС		

7.Дата видачі завдання 22 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Дослідження актуальності обраної теми	23.03.2024	30.03.2024	Виконано
2	Аналіз методів захисту інформації	1.04.2024	11.04.2024	Виконано
3	Дослідження актуальності обраної теми	12.04.2024	21.04.2024	Виконано
4	Дослідження інформаційних джерел	22.04.2024	9.05.2024	Виконано
5	Аналіз методів захисту інформації	10.05.2024	16.05.2024	Виконано
6	Розробка алгоритму роботи	17.03.2024	21.05.2024	Виконано
7	Реалізація поштовго клієнта	22.05.2024	25.05.2024	Виконано
8	ТЕстування програмного модуля	26.05.2024	28.05.2024	Виконано
9	Оформлення матеріалів до захисту БДР	29.05.2024	2.06.2024	Виконано

Студент _____ Іванченко О.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яремчук Ю.Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В даній дипломній роботі представлено розробку та реалізацію програмного модуля захисту для поштового клієнта, який здійснює сканування електронної пошти на наявність фішингових загроз. Для створення сервісу було обрано мову програмування Python.

В роботі проведено детальний аналіз предметної області, виявлено актуальні проблеми та проведено порівняння існуючих аналогічних рішень, їхніх переваг і недоліків. На основі цього аналізу було теоретично спроектовано сервіс та розроблено захисний модуль.

У результаті було створено ефективний та конкурентоспроможний сервіс для перевірки електронної пошти, що відзначається високим рівнем безпеки, що є критично важливим для забезпечення конфіденційності та захисту даних користувачів.

Ключові слова: захист електронної пошти, фішингові атаки, API VirusTotal, шифрування даних, інтеграція, конфіденційність даних, автентифікація, кібербезпека.

ANNOTATION

This diploma work presents the development and implementation of a protection module for a web service that scans emails for phishing threats. The Python programming language was chosen for the development of the service.

The work includes a detailed analysis of the subject area, identification of current issues, and comparison of existing similar solutions, highlighting their advantages and disadvantages. Based on this analysis, the service was theoretically designed, and the protection module was developed.

As a result, an efficient and competitive email scanning service was created, distinguished by its high level of security, which is crucial for ensuring the confidentiality and protection of user data.

Keywords: email protection, phishing attacks, VirusTotal API, data encryption, integration, data privacy, authentication.

ЗМІСТ

ВСТУП

У сучасному світі інформаційної безпеки, забезпечення надійного шифрування даних є важливою складовою для захисту конфіденційної інформації в онлайн-середовищі. Особливо це стосується текстур - цінного ресурсу для створення футажів, ігор і 3D-моделей. Проте, існуючі сервіси з продажу текстур залишаються вразливими перед можливими атаками зловмисників через відсутність ефективного шифрування.

З цією проблемою пов'язане і моє дослідження, метою якого є розробка імплементації інтернет-магазину текстур, який використовує шифрування для забезпечення безпеки даних користувачів. Актуальність цього дослідження підкреслюється не лише потребою в захисті інтелектуальної власності, а й загальною необхідністю у підвищенні рівня кібербезпеки в онлайн-середовищі.

В рамках дослідження буде проведено аналіз існуючих аналогів сервісів з продажу текстур, виокремлені їх переваги та недоліки, зокрема, відсутність ефективного шифрування. На основі цього аналізу буде запропоновано концепцію нового сервісу, який поєднує переваги існуючих аналогів та додає новий рівень безпеки завдяки застосуванню шифрування даних користувачів.

Далі в дослідженні розглянуться технічні аспекти реалізації цього сервісу, включаючи вибір мови програмування, розробку архітектури сервісу та теоретичну розробку модулю шифрування текстур та вбудовування водяних знаків в них.

Метою даного дослідження є створення імплементації інтернет-магазину текстур, який не лише надасть користувачам широкий вибір, а й забезпечить безпеку їхніх даних завдяки застосуванню шифрування та інших криптографічних методів.

Предметом дослідження є розробка і реалізація інтернет-магазину текстур, спрямованого на задоволення потреб у цифрових матеріалах для різноманітних творчих та професійних проектів. Основна увага приділяється аспектам забезпечення безпеки даних користувачів шляхом використання криптографічних методів шифрування.

Об'єктом дослідження є процес зберігання та обміну даними в інтернет-магазині текстур з механізмом захисту конфіденційності, цілісності та доступності цих даних під час їх передачі через мережу Інтернет. Важливою частиною дослідження є інтеграція криптографічних методів в процес обробки та зберігання даних, а також аналіз ефективності цих методів у контексті забезпечення безпеки даних користувачів.

1 ТЕОРЕТИЧНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предметна область

Електронна пошта залишається основним засобом комунікації в бізнес-середовищі та повсякденному житті мільйонів людей у всьому світі. Це ефективний і зручний інструмент для обміну інформацією, який, однак, має свої уразливості та ризики. Основною загрозою для електронної пошти є фішингові атаки, підробка повідомлень і зловмисне програмне забезпечення.

Цей проект має на меті розробку поштового клієнта з вбудованим захистом від фішингу за допомогою VirusTotal API[1]. Враховуючи широкий спектр потенційних загроз у сфері кібербезпеки, такий поштовий клієнт стає важливим інструментом для забезпечення безпеки користувачів у віртуальному середовищі:

– поштовий клієнт використовує VirusTotal API для перевірки поштових повідомлень на наявність потенційно шкідливих посилань або вкладень. Це дозволяє виявляти спроби фішингу та запобігати користувачам потраплянню на шахраївські сайти чи завантаженню шкідливих файлів.;

– інтеграція з VirusTotal API дозволяє поштовому клієнту отримувати оновлену інформацію про потенційні загрози та визначати рівень небезпеки повідомлень. Це допомагає користувачам уникати взаємодії зі схемами фішингу та іншими шкідливими діями;

– поштовий клієнт аналізує зміст електронних листів та вкладень з використанням VirusTotal API для виявлення можливих загроз. Це дозволяє швидко виявляти шаблони фішингу та інші шкідливі спроби та попереджати користувачів про потенційні ризики;

– Завдяки інтеграції з VirusTotal API, наш поштовий клієнт може виявляти нові та невідомі загрози, що дозволяє оперативно реагувати на них та запобігати можливим атакам на користувачів;

Розробка поштового клієнта з використанням VirusTotal API є кроком у напрямку покращення безпеки електронної пошти та захисту користувачів від різноманітних кіберзагроз.

Фішинг є однією з найпоширеніших кіберзагроз, спрямованих на викрадення конфіденційної інформації, такої як логіни, паролі, номери кредитних карток та інші персональні дані. Атака зазвичай здійснюється через електронні листи, які виглядають як легітимні повідомлення від відомих організацій, але містять шкідливі посилання або вкладення. Користувачі, які не обізнані про такі загрози, можуть легко стати жертвами фішингу, що може призвести до значних фінансових втрат і витоку особистих даних.

Фішингові атаки часто використовують техніки соціальної інженерії для обману користувачів. Це можуть бути повідомлення, які виглядають як відправлені від банків, онлайн-магазинів або інших сервісів, де користувачі мають облікові записи. Зловмисники можуть створювати фальшиві веб-сайти, які виглядають ідентично до справжніх, де користувачі вводять свої облікові дані, думаючи, що вони входять до свого акаунта.

Підробка електронних листів, або спуфінг, є ще однією поширеною загрозою, яка дозволяє зловмисникам надсилати повідомлення від імені довірених джерел. Це може бути використано для розповсюдження фальшивої інформації, шкідливих програм або здійснення інших видів шахрайства. Зловмисники можуть підробляти адреси відправників, щоб повідомлення виглядали так, ніби вони надійшли від легітимних джерел, таких як колеги, партнери або клієнти.

Спуфінг часто використовується в поєднанні з фішингом, щоб збільшити ймовірність успіху атаки. Наприклад, зловмисник може підробити електронну адресу керівника компанії і надіслати повідомлення співробітникам з проханням надати конфіденційну інформацію або виконати певні дії, такі як переказ грошей на вказаний рахунок.

Sender Policy Framework (SPF) є важливим інструментом захисту від спаму, підробки та фішингу в електронній пошті[2]. Цей технічний стандарт визначає процес перевірки електронної пошти, що надсилається з авторизованого поштового сервера, для виявлення підробки та запобігання спаму. Розроблений як доповнення до SMTP, основного протоколу надсилання електронної пошти, SPF виконує важливу роль, оскільки SMTP сам по собі не має механізмів аутентифікації.

SPF є одним із методів аутентифікації електронної пошти, який допомагає запобігти підробці адрес відправника. Власник домену реєструє запис TXT на своїх DNS-серверах, який містить інформацію про дозволені поштові сервери, які мають право надсилати повідомлення від імені цього домену. Коли сервер отримує електронний лист, він перевіряє запис SPF, щоб переконатися, що повідомлення надійшло з дозволеної IP-адреси.

Запис SPF забезпечує перевірку достовірності адреси відправника, але він не гарантує цілісності повідомлення під час передачі. Наприклад, зловмисник може захопити повідомлення після його відправлення і змінити його вміст, але запис SPF не допоможе виявити цю зміну.

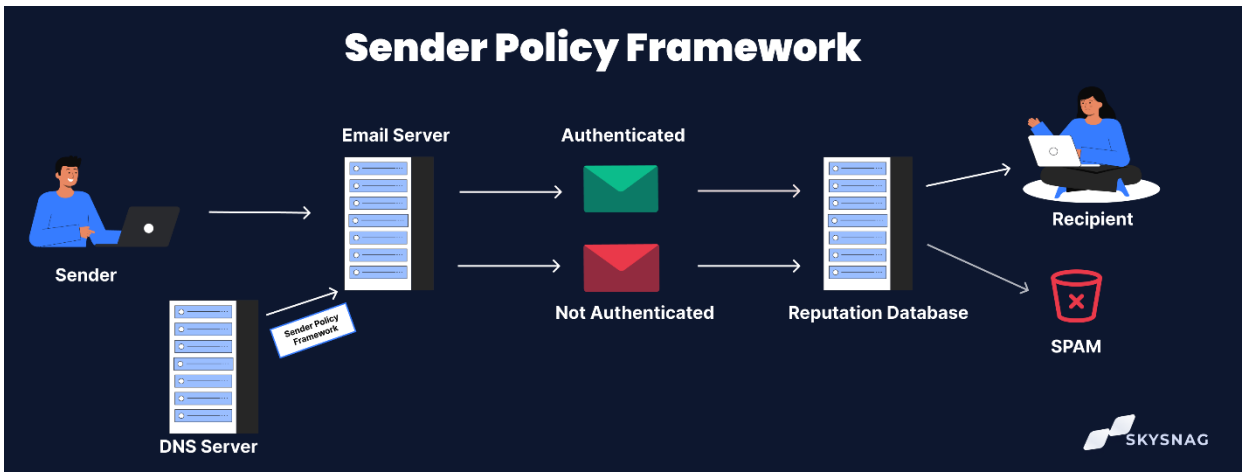


Рисунок 1.1 – Метод роботи SPF

Основні кроки процесу SPF такі:

– Адміністратор домену публікує політику, що вказує поштові сервери, які мають право надсилати електронну пошту з цього домену. Це відбувається шляхом створення запису SPF, який включається до загальних записів DNS домену;

– коли сервер вхідної пошти отримує електронне повідомлення, він шукає правила SPF для домену відмов у DNS. Потім сервер порівнює IP-адресу відправника з авторизованими IP-адресами, визначеними в записі SPF;

– поштовий сервер-одержувач використовує правила SPF, що зазначені в записі SPF домену-відправника, для прийняття рішення про те, чи приймати, відхиляти чи іншим чином обробляти електронне повідомлення;

DKIM (DomainKeys Identified Mail) - це протокол, який дозволяє організаціям підтверджувати автентичність та надійність своїх електронних листів, підписуючи їх таким чином, щоб постачальники поштових скриньок могли перевірити цю підписку. Завдяки криптографічній аутентифікації, перевірка запису DKIM дозволяє підтверджувати, що електронна пошта дійсно надійшла від вказаного в листі відправника та не була змінена під час транспортування через Інтернет[3].

DKIM використовує криптографічні підписи для забезпечення цілісності та автентичності електронних листів. Відправник підписує певні частини повідомлення за допомогою приватного ключа, а сервер, що приймає, може перевірити підпис за допомогою публічного ключа, розміщеного в DNS. Це дозволяє переконатися, що повідомлення не було змінено під час передачі, і що воно дійсно надійшло від вказаного відправника.

DKIM забезпечує високий рівень захисту від підробки повідомлень, але він не перевіряє IP-адресу відправника. Тому, DKIM найкраще працює у поєднанні з SPF для забезпечення повного захисту.

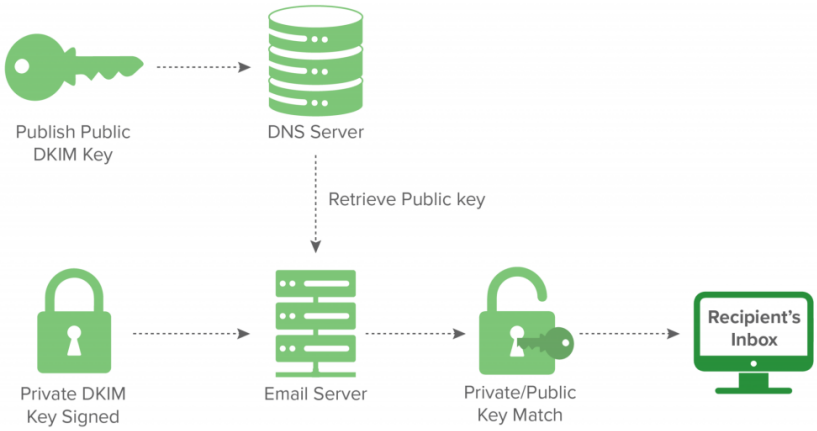


Рисунок 1.2 – Метод роботи DKIM

Процес підписання DKIM включає три основні кроки. Спочатку відправник визначає, які поля повинні бути включені у підпис DKIM, такі як адреса «від», тіло повідомлення, тема та інші. Ці поля повинні залишатися незмінними під час передачі, щоб забезпечити успішну аутентифікацію DKIM.

По-друге, платформа електронної пошти відправника створює хеш (криптографічний контрольний суму) текстових полів, включених у підпис DKIM,

який зазвичай включає дані, такі як відправник і тема повідомлення. Цей хеш потім зашифровується за допомогою приватного ключа, доступного лише відправнику.

На останньому етапі, коли лист надіслано, постачальник поштової скриньки або поштовий шлюз одержувача перевіряє підпис DKIM, використовуючи відкритий ключ. Він розшифровує підпис до вихідного хешу. Потім одержувач генерує власний хеш для порівняння з отриманим. Якщо хеші співпадають, це свідчить про те, що дані не були змінені під час передачі, і відправник листа володіє відповідним приватним ключем, підтверджуючи автентичність повідомлення.

DMARC об'єднує методи SPF і DKIM, забезпечуючи додатковий рівень захисту і звітності. Власник домену може визначити політику, яка вказує, що робити з повідомленнями, які не проходять перевірку автентичності (SPF або DKIM) [4]. Наприклад, такі повідомлення можуть бути відхилені, поміщені в карантин або доставлені з попередженням. Крім того, DMARC надає механізм для отримання звітів про повідомлення, які не пройшли перевірку, що дозволяє власникам доменів моніторити та аналізувати спроби зловживання їх доменами.

Підпис DKIM є ключовим елементом процесу перевірки автентичності повідомлення. Він представляє собою хеш, або контрольну суму, створену на основі різних компонентів повідомлення. Відправник формує цей підпис за допомогою закритого ключа свого домену, щоб зашифрувати певні частини повідомлення та створити хеш. Потім сервер отримувача використовує відкритий ключ відправника для дешифрування тих самих компонентів повідомлення.

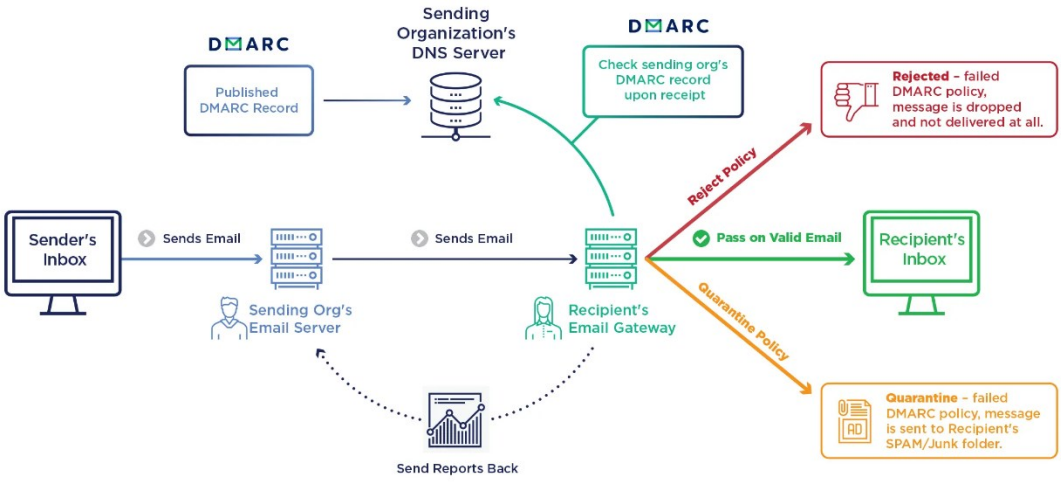


Рисунок 1.3 – Метод роботи DMARC

DMARC взаємодіє з SPF і DKIM, щоб підвищити безпеку електронної пошти. SPF визначає, які сервери мають право відправляти пошту від імені певного домену, тим самим ускладнюючи завдання зловмисників, які намагаються підробити електронну адресу відправника. DKIM забезпечує цифровий підпис повідомлення, що дозволяє отримувачу перевірити, чи було повідомлення змінено під час транспортування.

DMARC узгоджує ці механізми та дозволяє власникам доменів встановлювати політику обробки повідомлень, які не вдало проходять перевірку SPF або DKIM. Наприклад, вони можуть вирішити відхилити такі повідомлення або помістити їх у карантин. Це дозволяє власникам доменів контролювати, як їхні електронні листи обробляються на стороні отримувача, сприяючи виявленню та запобіганню фішинговим атакам та іншим загрозам безпеці.

Для додаткового захисту електронної пошти можна використовувати інструменти, такі як VirusTotal API. VirusTotal є онлайн-сервісом, який дозволяє аналізувати файли та URL на наявність шкідливих програм за допомогою різних антивірусних движків і інструментів. Використання VirusTotal API в поштовому клієнті дозволяє автоматично перевіряти вкладення та посилання в електронних листах на наявність шкідливих компонентів, що підвищує рівень безпеки.

Технології SPF, DKIM та DMARC забезпечують комплексний підхід до захисту електронної пошти. Вони працюють разом для запобігання різним видам атак, таким як фішинг, спуфінг і підробка повідомлень. Кожна з цих технологій має свої переваги і недоліки, але їх спільне використання забезпечує максимальний рівень захисту.

SPF допомагає виявляти і блокувати підроблені повідомлення, що надходять з неперевірених джерел. DKIM забезпечує цілісність повідомлень, гарантуючи, що вони не були змінені під час передачі. DMARC об'єднує ці два методи, дозволяючи власникам доменів визначати політику обробки повідомлень, які не проходять перевірку, і отримувати звіти про такі спроби.

Розглянемо деякі типи фішингу[5].Основні типи фішингу:

1. Поштовий фішинг – є одним з найбільш поширених видів атак інтернет-

шахраїв. Зазвичай, це використовує метод "spray and pray" - хакери відправляють масову розсилку електронних листів на велику кількість адрес, виглядаючи як довірена особа або організація. Ці листи можуть містити шкідливі посилання або вкладення, або зміст, що спонукає жертву розголошувати свої особисті дані. Зазвичай, поштовий фішинг використовується для викрадення особистих даних, таких як логіни та паролі, а також для розповсюдження шкідливого програмного забезпечення, що може призвести до компрометації комп'ютерної системи або витрати фінансових коштів.

2. Цільовий фішинг – це вдосконалена версія звичайного поштового фішингу, спрямована на конкретних користувачів або групу осіб, зазвичай працівників певної компанії чи організації. Ця атака є складнішою, оскільки вимагає попереднього збору інформації про цільову організацію, її структуру та співробітників.

3. Вейлінг (Whaling) – Вейлінг, або "китовий" фішинг, є вдосконаленою формою цільового фішингу, спрямованого на високопосадових осіб в організації, таких як виконавчий директор, директор з фінансів чи інші керівники. Оскільки члени топ-менеджменту часто мають доступ до критичної та конфіденційної інформації, вони стають особливою мішенню кіберзлочинців. Однією з особливостей вейлінгу є те, що топ-менеджмент, як правило, має менше нагляду за дотриманням правил інформаційної безпеки, оскільки їм довіряють і вони не піддаються такому ж самому рівню перевірки, що й рядові працівники. Це створює ідеальні умови для кіберзлочинців, які можуть використовувати соціальну інженерію та провокативні методи, щоб змусити керівників виконати небезпечні дії.

4. Смішинг є варіацією звичайного фішингу, але замість електронної пошти використовує смс-розсилання. Цей вид атаки полягає в тому, що зловмисники надсилають підроблені текстові повідомлення на мобільні телефони потенційних жертв, спонукаючи їх до виконання певних дій або розкриття конфіденційної інформації. Смішинг може приймати різні форми, включаючи повідомлення про неперевірені транзакції, виграші в лотереї, або навіть повідомлення про

надходження важливої інформації від урядових органів чи фінансових установ.

5. Вішинг – це вид фішингу, який використовує телефонні дзвінки та автоматичні голосові повідомлення з метою здобуття конфіденційної інформації від жертви. Зазвичай зловмисники вигадуються за представників банків, фінансових установ, компаній з послуг зв'язку або навіть урядових агентств, щоб створити вигляд легітимності та довіри.

6. Клон-фішинг – метод атаки, при якому зловмисники створюють шкідливу копію недавно отриманого листа або повідомлення. Ця копія виглядає майже ідентично оригіналу, відправленому відомим або довіреним джерелом. Проте, в клон-фішингу вставляється підроблене посилання або вміст, що містить шкідливий код або запит на введення конфіденційної інформації.

7. Злий двійник (Evil Twin) – є винахідливим методом атаки, де зловмисник створює ідентичну копію легітимної Wi-Fi мережі з тією ж назвою (SSID) та іноді навіть з такою самою MAC-адресою. Мета - змусити жертву підключитися до цієї підробленої мережі. Після підключення, зловмисник може перехоплювати, аналізувати та навіть підмінювати весь трафік, що проходить через цю мережу.

8. Фармінг – методом фішингу, при якому зловмисники націлюються на службу DNS (Domain Name System). Основна складність цієї атаки полягає в тому, що для її здійснення зловмисникам необхідно отримати доступ до кешу DNS локального комп'ютера або мережевого обладнання. Проте, якщо зловмиснику вдається здійснити такий доступ, він може почати підміняти легітимні веб-сайти на небезпечні або фішингові сторінки.

1.2 **Аналіз актуальності обраної теми**

В сучасному світі кібербезпека стає все більш актуальною проблемою. Зі зростанням обсягу електронних комунікацій збільшується і кількість кіберзагроз, пов'язаних з електронною поштою. Фішингові атаки та підробка повідомлень є одними з найбільш поширених методів кібератак, які використовують зловмисники для викрадення конфіденційної інформації та здійснення шахрайства.

Згідно з дослідженнями, кількість фішингових атак зростає з кожним роком. Зловмисники постійно розробляють нові методи та техніки для обману користувачів і обходу існуючих систем безпеки. Наприклад, вони можуть використовувати соціальну інженерію для створення повідомлень, які виглядають як відправлені від довірених джерел, або використовувати шкідливі посилання та вкладення для зараження комп'ютерів користувачів.

Однією з основних причин успішності фішингових атак є недостатня обізнаність користувачів про кіберзагрози. Багато людей не знають, як розпізнати фішингові повідомлення або як діяти у разі підозри на атаку. Тому освітня робота та підвищення рівня кіберграмотності є важливими аспектами протидії фішингу.

Впровадження технологій SPF, DKIM та DMARC є важливим кроком для забезпечення безпеки електронної пошти. Вони допомагають захистити користувачів від фішингових атак і підробки повідомлень, забезпечуючи автентичність і цілісність електронних листів. Однак, для досягнення максимального рівня захисту необхідно використовувати комплексний підхід, що включає додаткові заходи безпеки, такі як шифрування даних, антивірусне програмне забезпечення та навчання користувачів.

Фішинг є одним з найпоширеніших методів атаки в сфері кібербезпеки. Його привабливість полягає в його відносній простоті використання, оскільки він не потребує глибоких технічних знань чи складних інструментів. Однак для успішної реалізації фішингової атаки необхідно мати добре розуміння психології людини та вміти ефективно збирати необхідну інформацію.

Саме тому більшість кібератак починаються з фішингу або включають елементи фішингу. Це пояснюється тим, що фішинг дозволяє зловмисникам легко вивчити поведінку та звички потенційних жертв, а також отримати критичну інформацію, яка може бути використана для подальшого злому системи або здійснення інших кібератак. Згідно звіту від Verizon та інформацією від Cisco, стосовно розслідування витоків даних за 2021 рік, виявлено, що значна частина витоків даних пов'язана з людським фактором. Зокрема, відзначено, що до 85% витоків даних мають причини у людських помилках або недбалості[17].

Цей метод розповсюдження шкідливого програмного забезпечення через електронну пошту є досить ефективним через широке використання офісних програм у бізнес-середовищі та серед індивідуальних користувачів. Зловмисники надсилають електронні листи, які містять заражені документи, такі як Word або Excel файли, які містять вбудовані шкідливі макроси або експлойти. Після відкриття такого документа, шкідливе програмне забезпечення може встановитися на комп'ютері користувача, виконуючи різноманітні атаки, від шпигунства до шифрування файлів за допомогою вимагає викупу (ransomware).

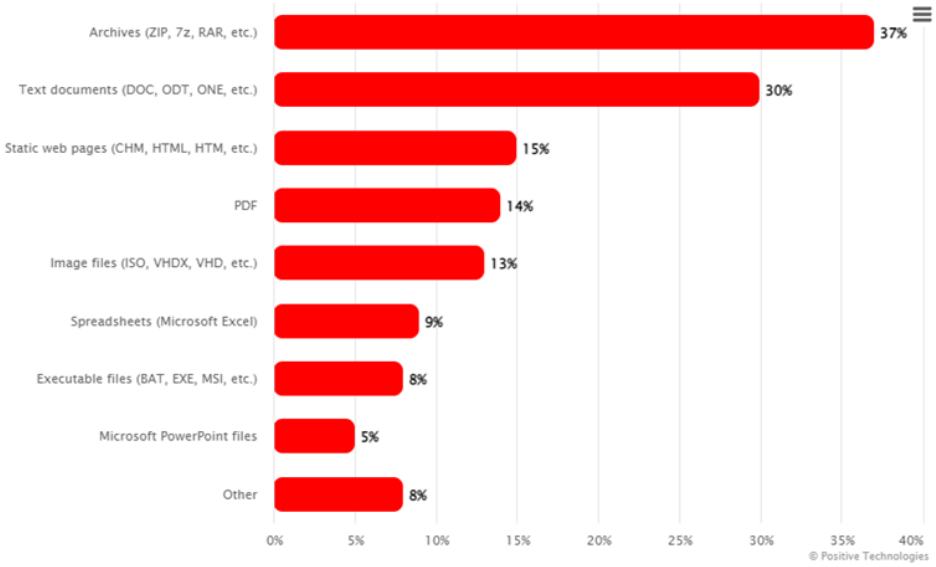


Рисунок 1.4 – Статистика вмісту фішингових посилань на 2023 рік[18]

Щодо фішингових атак, цільовий фішинг виявляється найбільш поширеним видом. Згідно з аналізом від Symantec, цільовий фішинг припадає на 65% усіх фішингових атак[19]. Це вказує на те, що зловмисники все частіше спрямовують свої атаки на конкретних користувачів або організації з метою отримання конфіденційної інформації або викликати інші шкідливі наслідки.

По-перше, фішингові атаки стають більш вдосконаленими і складнішими, оскільки зловмисники постійно вдосконалюють свої методи та використовують нові технології для обхіду захисту. Вони адаптуються до заходів безпеки, що призводить до збільшення кількості успішних атак.

По-друге, зростання кількості користувачів Інтернетом також сприяє збільшенню фішингових атак. Більше людей використовують електронну пошту,

соціальні мережі та інші онлайн сервіси, що робить їх потенційними цілями для зловмисників.

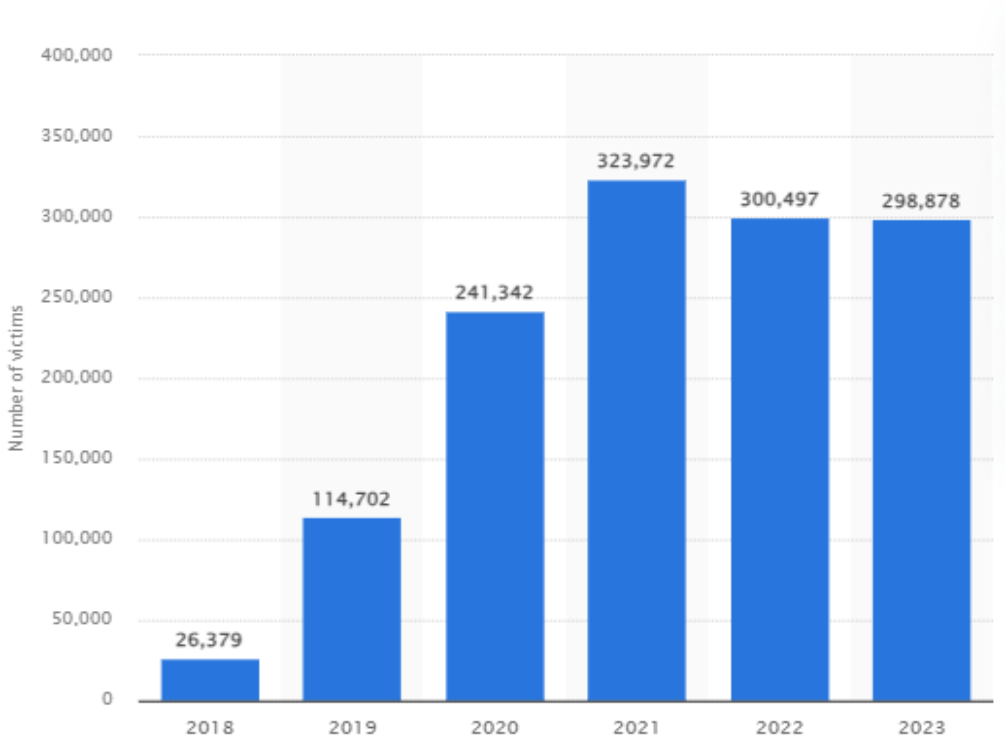


Рисунок 1.5 – Таблиця кількості фішингових атак за 2018-2023 роки[20]

Розглядаючи такі графіки, можна побачити відповідність між важливими подіями та збільшенням кількості фішингових атак. Під час кризових ситуацій, таких як пандемія коронавірусу, зловмисники часто використовують ці моменти для здійснення атак на людей, які можуть бути вразливі або мають збільшену потребу в інформації.

У період пандемії багато людей зверталися до Інтернету для отримання оновленої інформації про ситуацію, медичні поради, а також для вирішення різних проблем, пов'язаних з карантинними обмеженнями. Зловмисники використовували це для поширення фішингових атак, використовуючи теми, пов'язані з пандемією, як прикриття для своїх шахрайських дій.

Це ще раз підкреслює важливість підвищення уваги та заходів кібербезпеки під час кризових ситуацій та важливих подій. Ретельна перевірка електронних повідомлень та уважне ставлення до онлайн інформації можуть допомогти уникнути ставших все більш поширеними фішингових атак

1.3 Огляд існуючих аналогів захисту від фішингу

Для розуміння конкурентів запропонованого проєкту, розглянемо найпопулярніші аналоги які надають послуги доступу до текстур.

– Canibespoofed - це консольний проект, який призначений для сканування доменів з метою отримання інформації про ланцюг поставки електронного листа та виявлення вразливостей SPF/DMARC;

– Espoofer: Espoofer є відкритим інструментом тестування, який призначений для проведення атак з підміною електронної пошти та перевірки захищеності поштових сервісів. Цей інструмент дозволяє випробувати аутентифікацію SPF, DKIM та DMARC, що допомагає виявляти потенційні вразливості у захисті електронної пошти;

– Muraena/Necrobrowser - це інструмент для фішингу, який відрізняється від інших засобів перехоплення даних, таких як підміна домену. Основна мета Muraena/Necrobrowser - перехоплення токенів двофакторної автентифікації (2FA) та подальша експлуатація після фішингу;

– Metadefender: сервіс з аналізу файлів, який використовує різноманітні антивірусні движки[6];

– Cuckoo Sandbox: ізольоване середовище для безпечного аналізу програм, яке надає детальні звіти про результати аналізу.

За допомогою Espoofer можна симулювати різні сценарії атак, такі як підробка відправника, порушення підпису DKIM або ігнорування правил SPF. Це дозволяє адміністраторам тестувати ефективність їхніх захисних механізмів та приймати заходи для покращення безпеки своїх поштових сервісів.

Загалом, Espoofer є корисним інструментом для виявлення слабких місць у захисті електронної пошти і забезпечення відповідного рівня безпеки в поштових системах.

Переваги Espoofer:

– надає розширені можливості аналізу тексту повідомлень;

– швидко виявляє фішингові схеми та інші шкідливі дії;

– має гнучку систему налаштувань для користувачів;

Недоліки Espoofer:

– може бути складним у використанні для неспеціалістів;

– потребує регулярного оновлення та моніторингу для ефективної роботи;

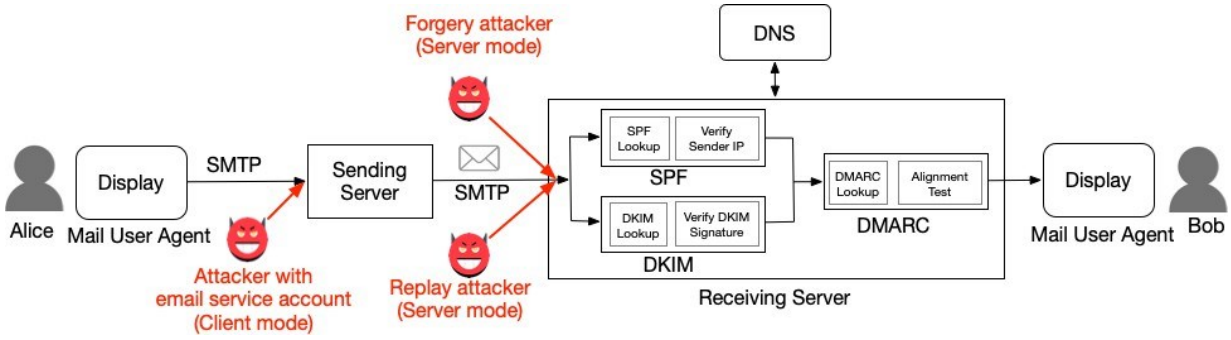


Рисунок 1.6 – Режими роботи espoofer

Canibespoofed - це консольний проект, який призначений для сканування доменів з метою отримання інформації про ланцюг поставки електронного листа та виявлення вразливостей SPF/DMARC.

Canibespoofed сканує домени, щоб виявити неправильну конфігурацію SPF та DMARC, що може призвести до збільшення ризику фішингу та інших видів атак. Цей інструмент аналізує ланцюг поставки електронного листа для виявлення потенційних точок вразливості або можливостей для атаки. Canibespoofed може визначити геолокацію відправника пошти, що допомагає в ідентифікації потенційно шкідливих джерел. Утиліта може порівнювати відправників електронної пошти з чорними списками, щоб виявити, чи мають вони негативну репутацію та чи вважаються вони спамерами або зловмисниками.

Переваги Canibespoofed:

– використовує розширені алгоритми аналізу для виявлення підроблених повідомлень;

– швидко реагує на нові загрози та оновлює свою базу даних;

– доступність різних форматів і роздільної здатності дозволяє використовувати їх у різних проектах та програмах;

– має простий інтерфейс для використання.

Недоліки Canibespoofed:

– може бути обмеженою кількість підтримуваних форматів повідомлень;

– може вимагати платну підписку для доступу до деяких функцій;

Muraena/Necrobrowser - це інструмент для фішингу, який відрізняється від інших засобів перехоплення даних, таких як підміна домену. Основна мета Muraena/Necrobrowser - перехоплення токенів двофакторної автентифікації (2FA) та подальша експлуатація після фішингу.

Muraena діє як проксі-сервер, який перехоплює облікові дані та файли cookie сеансів в реальному часі під час фішингових атак. Це дозволяє зловмисникам збирати важливі дані для подальшого використання. Зібрані сеанси передаються до Necrobrowser, що є іншою складовою системи Muraena/Necrobrowser. Necrobrowser використовує ці сеанси, щоб виглядати як жертва та отримувати доступ до її облікових даних. Necrobrowser використовує набір докеризованих браузерів Chrome, що дозволяє підтримувати вкрадені сеанси та автоматизувати виконання дій від імені зловмисника. Після отримання доступу до сеансів Necrobrowser автоматично виконує різні дії від імені жертви, такі як доступ до її облікових записів та виконання дій на її користь. Метод, що використовує Muraena та Necrobrowser, надає можливість створювати фішингові атаки з високим ступенем реалізму, що мало відрізняється від звичайного користувацького досвіду. Під час фішингових атак користувач просто переходить на фішинговий домен, що дозволяє створити враження непорушеного проходження веб-сайту без будь-яких очевидних відмінностей. Muraena та Necrobrowser здатні підтримувати динамічний контент веб-сайтів, що робить атаку ще більш реалістичною, відмінною від старих фішингових платформ, які мали обмежені можливості адаптації під динамічний контент.

Переваги Muraena:

— використовує продвинуті технології машинного навчання для виявлення шкідливих дій;

— має велику базу даних зі зразками шкідливих повідомлень;

— забезпечує детальний звіт про результати аналізу;

Недоліки Muraena:

— може бути вимогливим до ресурсів комп'ютера через великий обсяг аналізу;

— може потребувати встановлення та налаштування перед використанням;

Metadefender - це платформа для аналізу файлів та URL-адрес з використанням кількох антивірусних движків та інших інструментів безпеки. Вона дозволяє користувачам завантажувати файли або відправляти URL-адреси для сканування та отримання звіту про їхню безпеку. Metadefender інтегрується з різними антивірусними движками та іншими інструментами безпеки, що дозволяє отримувати більш об'єктивну та повну інформацію про потенційні загрози.

Metadefender - це рішення, яке надає широкий спектр інтеграції з різними антивірусними движками та іншими інструментами безпеки, забезпечуючи більш об'єктивний результат сканування. Воно також надає докладні звіти про безпеку файлів та URL-адрес, що дозволяє користувачам отримувати повну інформацію про потенційні загрози. Metadefender також відомий своєю швидкістю сканування, забезпечуючи швидке оброблення файлів та URL-адрес за короткий час. Проте, деякі функції та плани можуть бути дорогими для деяких користувачів, а також для коректної роботи необхідне постійне підключення до Інтернету.

Cuckoo Sandbox - це віртуальне середовище для аналізу шкідливих програм та файлів. Він дозволяє виконувати шкідливі програми у безпечному ізольованому середовищі та аналізувати їх поведінку для виявлення потенційних загроз. Cuckoo Sandbox надає детальні звіти про результати аналізу, включаючи інформацію про взаємодію програми з операційною системою та іншими процесами. Цей інструмент допомагає виявляти та аналізувати нові види шкідливого програмного забезпечення та загрози безпеки.

Cuckoo Sandbox - це інструмент, який дозволяє виконувати шкідливі програми у безпечному віртуальному середовищі, запобігаючи поширенню інфекцій на основні системи. Він надає детальні звіти про поведінку шкідливих програм, що дозволяє зрозуміти їхні функції та потенційні наслідки. Крім того, Cuckoo Sandbox є відкритим проектом, що сприяє співпраці та внесенню вдосконалень користувачами. Однак, він вимагає певних навичок та часу для налаштування та

розгортання, а також може вимагати значних ресурсів обчислювальної потужності та пам'яті для запуску віртуальних машин.

1.4 **Висновки до розділу 1**

У данному розділі було проведено аналіз предметної області, пов'язаної з розробкою поштового клієнта із захистом від фішингу з використанням VirusTotal API. Було розглянуто різноманітні аспекти застосування такого клієнта, від надання зручного доступу до електронної пошти до забезпечення високого рівня безпеки для користувачів.

Аналіз актуальності проблеми фішингу в електронній пошті вказує на необхідність розробки ефективних інструментів захисту, які б забезпечували вчасне виявлення та блокування шкідливих повідомлень. У цьому контексті використання VirusTotal API може стати важливим компонентом в системі захисту, оскільки цей сервіс дозволяє отримувати інформацію про безпеку файлів та URL-адрес, що допомагає відокремити потенційні загрози від безпечних повідомлень.

Щодо аналогів VirusTotal API, були розглянуті Canibespoofed, Espoofer та Muraena/Necrobrowser. Кожен з них має свої переваги і недоліки. Canibespoofed і Espoofer відомі своєю ефективністю у виявленні фішингових атак, проте вони можуть мати обмежену функціональність та потребувати додаткових витрат на їх використання. З іншого боку, Muraena/Necrobrowser мають більш широкий функціонал, але вони можуть бути складні у використанні та вимагати великих ресурсів.

2 ПРОЕКТУВАННЯ ПОШТОВОГО КЛІЄНТА

Після огляду предметної області та обгрунтування актуальності проблеми створення захищеного сервісу для зберігання та продажу двохвимірних текстур почався процес проектування майбутнього сервісу. Спочатку було обрано модель архітектсури, розроблено сервіс, розроблено архітектуру захищеного модулю та обгрунтовано вибір воми програмування для реалізації додатку.

2.1 Розробка архітектури сервісу

Під час проектування для системи було обрано архітектуру MVC: Принцип MVC (Model-View-Controller) - Модель-Вид-Контролер допомагає розділити логіку додатку на окремі компоненти, що спрощує його розробку, тестування та підтримку[7]. Крім того, він підтримує принцип одноразового використання, що означає, що кожен компонент має одну чітко визначену відповідальність і може бути легко змінений або замінений без впливу на інші частини додатку. Ця модель розділяє систему на три взаємопов'язані частини: модель даних, представлення (інтерфейс користувача) і модуль управління (рис. 2.1).

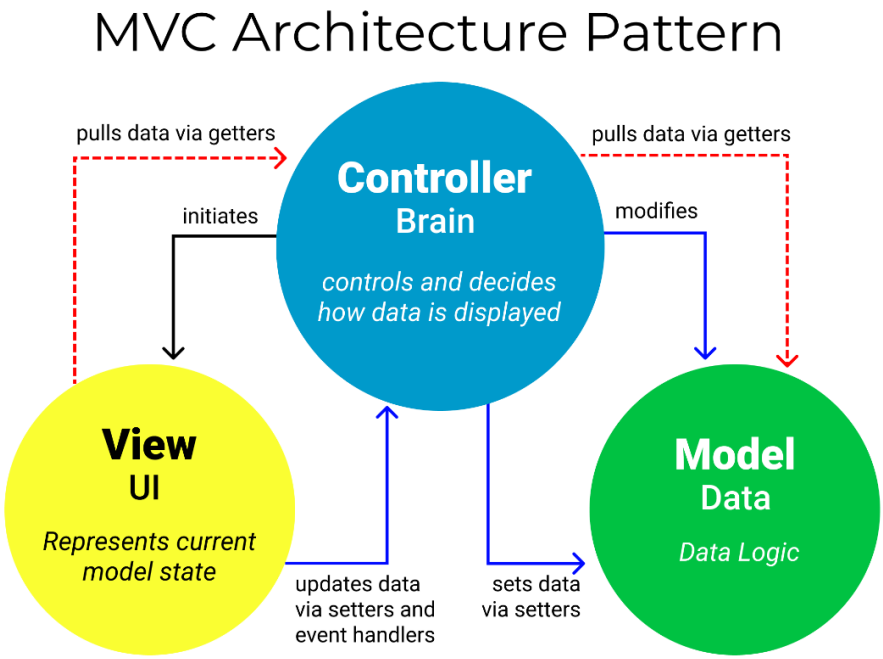


Рисунок 2.1 - Шаблон проектування MVC

Вона складається з таких компенентів:

1. Модель відповідає за управління даними додатку. Вона отримує дані від користувача через контролер, обробляє їх та оновлює представлення, коли дані змінюються. У контексті нашого поштового клієнта з захистом від фішингу, модель буде включати в себе управління повідомленнями електронної пошти, фільтрами безпеки, зберіганням інформації про користувачів та результатами перевірок VirusTotal API.

2. Представлення відповідає за відображення даних, що містяться в моделі. Воно представляє інтерфейс користувача і відображає дані таким чином, щоб користувач міг з ними взаємодіяти. В нашому випадку, представлення включатиме інтерфейс для відображення електронних листів, повідомлень про безпеку та результатів перевірок фішингу..

3. Контролер виступає посередником між моделлю та представленням. Він отримує вхідні дані від користувача через представлення, обробляє їх (зокрема, шляхом звернення до моделі) і повертає результат у представлення. У нашому поштовому клієнті контролер оброблятиме взаємодії користувача з інтерфейсом, такі як відкриття листів, перевірка їх на фішинг через VirusTotal API, і передача результатів назад до представлення для відображення користувачеві.

Однією з основних переваг MVC є розділення обов’язків. Це означає, що кожен компонент системи (модель, представлення, контролер) має свою чітко визначену роль. Завдяки цьому, зміни в одному компоненті мають мінімальний вплив на інші компоненти, що полегшує обслуговування і розширення системи.

MVC сприяє полегшенню тестування і налагодження додатку. Оскільки компоненти системи розділені, тестування кожного з них може бути проведене незалежно. Це дозволяє швидко і ефективно виявляти та виправляти помилки.

Завдяки розділенню логіки додатку, різні команди розробників можуть працювати паралельно над різними компонентами системи. Наприклад, одна команда може працювати над представленням, інша — над моделлю, а ще одна — над контролером. Це скорочує час розробки і підвищує ефективність команди.

Шаблони в MVC-архітектурі використовуються для відокремлення логіки даних (моделі) від користувацького інтерфейсу (представлення). Це дозволяє вносити зміни в інтерфейс без впливу на логіку роботи з даними і навпаки. [22].

Основні елементи архітектури системи включають

– структура бази даних: забезпечує зберігання інформації про користувачів, електронні листи та результати перевірок на фішинг;

– взаємодія між клієнтом і сервером: використання HTTP-запитів для обміну даними між користувацьким інтерфейсом і сервером.;

– методи зберігання моделей/даних/результатів: використання форматів збереження даних, таких як JSON, для обміну інформацією між клієнтом і сервером. ;

– спосіб аналізу та обробки результатів: Інтеграція з VirusTotal API для перевірки на фішинг та обробка результатів аналізу .

Взаємодія між клієнтом і сервером базується на використанні HTTP-запитів для передачі даних. Це дозволяє чітко визначити, яке поле відповідає за кожен тип переданого значення, що забезпечує надійний і зрозумілий інтерфейс додатку для отримання бажаних результатів.

Розроблений поштовий клієнт використовує формат JSON для обміну даними між клієнтом і сервером. JSON є зручним для читання як людиною, так і машиною, що забезпечує легкість інтеграції та обміну даними[8].

Для зменшення навантаження на сервер, використовується модель для зберігання результатів перевірок на фішинг. Це досягається шляхом збереження даних у структурованому вигляді та використання хеш-сум для швидкої перевірки унікальності даних.

Ця діаграма відображає взаємодію користувача із системою та основні кроки, які він виконує, з урахуванням додаткових функцій шифрування пошти та збереження небезпечних адрес у базу даних. (рис.2.2). Її розробка включає:

- вхід в систему;

- перевірка наявності нових листів;

- сканування вкладень;

- отримання результатів сканування;

- шифрування листів

-перегляд результатів сканування

-збереження небезпечних адрес у базу днаних

-налаштування параметрів сканування

-вихід із системи

На діаграмі показані користувачі системи та інші компоненти, тобто розроблене програмне забезпечення, сервери даних, бази даних і сама система аналізу. Взаємодія з сервісом відбувається через виконання акторами (користувачами) певних операцій. Так, користувач в першу чергу виконує такі операції, як звернення до системи з проханням обробити запит, задання початкових параметрів або запуск процесу обробки чи аналізу даних.

Рисунок 2.2 – Use-case діаграма

Після створення загальної архітектури сервісу було розроблено план розробки проекту. Для цього потрібно було розбити проект на етапи, використовуючи декомпозицію. Декомпозиція - це науковий метод, який використовує структуру завдання, щоб дозволити замінити рішення великої задачі рішенням серії менших, взаємопов'язаних, але простіших завдань.

Дана блок-схема зображує всі можливі процеси сервісу, які потрібно реалізувати:

1. Внесення даних. Потрібно розробити механізм внесення даних в систему. В цьому випадку, це завантаження нових листів з сервера..
2. Перевірка аутентифікації. Введені дані повинні пройти перевірку автентифікації для підтвердження прав користувача. Потрібно розробити алгоритми реєстрації та авторизації..
3. Аналіз вхідних даних. Завантажені дані потрібно проаналізувати та обробити. Система сканує вкладення у листах за допомогою VirusTotal API.
4. Вибір інформації з БД. На основі пошукового запиту, або під час перевірки небезпечних адрес, потрібні дані потрібно витягнути з БД. Для цього потрібно реалізувати функцію пошуку даних.
5. Оновлення запиту. Запит на перевірку листів може бути оновленим на основі введеної інформації.
6. Завантаження. Потрібно реалізувати функцію завантаження даних в базу, зокрема збереження небезпечних адрес.
7. Шифрування. Важливо реалізувати шифрування листів та вкладень перед зберіганням.
8. Збереження результатів. Результати сканування та перевірки повинні бути збережені у базі даних..

Оскільки декомпозиція є методом розкладання складної системи на більш прості, взаємопов'язані підсистеми, будь-яку систему можна розглядати як сукупність таких підсистем. В контексті нашої розробки, система захисту електронної пошти від фішингових атак також може бути розділена на кілька взаємозалежних процесів та компонентів (рис. 2.3).

Рисунок 2.3 - Декомпозиція процесу розробки системи

Ця блок-схема ілюструє основні етапи та процеси, які потрібно реалізувати в межах нашої системи::

1. Завантаження даних. Розробка механізму для завантаження даних до системи, включаючи електронні листи та їхні вкладення.
2. Перевірка автентифікації. Автентифікація користувача для підтвердження його прав доступу. Необхідно реалізувати алгоритми для реєстрації та входу до системи.
3. Аналіз вхідних даних. Аналіз та обробка завантажених даних з метою виявлення фішингових атак та інших загроз.
4. Вибір інформації з бази даних. На основі пошукових запитів або під час перевірки листів, необхідні дані будуть витягуватись з бази даних. Потрібно реалізувати функцію пошуку та вилучення даних.
5. Оновлення запиту. Запит на перевірку або обробку листів може бути оновлений на основі нових введених даних.
6. Збереження даних. Реалізація функції збереження даних в базу даних.

7. Шифрування. Важливо реалізувати шифрування даних користувачів та електронних листів для забезпечення безпеки перед зберіганням

Ця декомпозиція дозволяє чітко уявити, які кроки необхідно виконати для успішного розроблення та впровадження системи захисту від фішингових атак, забезпечуючи структурований підхід до вирішення завдання.

Діаграми діяльності служать для візуалізації та моделювання виконання операцій в проектній системі. Основна мета полягає у застосуванні блок-схем та діаграм алгоритмів для детального опису алгоритмів та послідовностей дій. Кожна діаграма спрямована на конкретний процес або послідовність процесів, які приводять користувача до досягнення бажаного результату. При моделюванні процесів за допомогою мови UML також використовуються діаграми діяльності, які представлені у вигляді графів, де вершини відображають стани, а дуги вказують на переходи між станами або типами процесів[9].

У контексті роботи з діаграмами діяльності, важливо показати, візуалізувати та розкрити особливості реалізації операцій класу, а також відобразити роботу алгоритму та послідовність його процесів. Кожна дія та її стан виконується в рамках певного класу операцій, а тип дії визначається як процес послідовності дій..

Елементи діаграми діяльності:

1. Головна сторінка:
   - Стартова точка, з якої починається процес використання поштового клієнта..
2. Авторизація користувача:
   - Перевірка, чи користувач має доступ до системи.
   - Якщо так (Yes), перехід до перегляду пошти.
   - Якщо ні (No), користувач перенаправляється до форми для введення облікових даних.
3. Введення облікових даних:
   - Користувач вводить свої дані для авторизації.
4. Перевірка облікових даних:
   - Система перевіряє правильність введених даних.

– Якщо дані введено вірно (Yes), відбувається авторизація користувача.

– Якщо дані введено неправильно (No), користувач повертається до форми для виправлення даних.

5. Авторизація користувача на сайті:

– Після успішної авторизації користувач має доступ до своєї пошти та інших можливостей.

6. Перегляд пошти:

– ористувач переглядає вхідні та відправлені листи.

7. Створення нового листа:

– Користувач створює нове повідомлення.

8. Відправлення листа:

– Користувач обирає отримувача та надсилає лист.

9. Помилка під час відправлення:

– Перевіряється, чи виникла помилка під час відправлення листа.

– Якщо так (Yes), користувач отримує повідомлення про помилку.

– Якщо ні (No), користувач може повторити спробу відправлення.

10. Перегляд списку відправлених листів:

– Користувач переглядає список листів, які він відправив.

11. Завершення (Кінець).

Наступним етапом є стоврення діаграми, яка показує зв’язки інтерфейсу з сервером та базою даних.

Діаграма розгортання відображає обчислювальні вузли та компоненти вашого поштового клієнта. Кожен компонент представляє собою окремий модуль програмного коду, який виконує певну функцію в системі. Ця діаграма дозволяє вам візуалізувати, як ваша програма буде розгортатися на сервері та як взаємодіятиме з іншими компонентами. (рис. 2.4).

Рисунок 2.4 – Діаграма розгортання

Останнім етапом є створення архітектури бази даних, яка є ключовою складовою вашого поштового клієнта. Для цього потрібно створити три таблиці::

1. Таблиця "Користувачі", що містить дані про авторизацію та основну інформацію про користувачів, такі як ім'я, адреса електронної пошти, хеш паролю тощо.

2. Таблиця "Листи", в якій зберігаються дані про вхідні та вихідні листи, включаючи інформацію про отримувачів, теми, дату тощо.

3. Таблиця "Вкладення", яка відстежує вкладені файли та вразливі посилання в листах, їхні типи .

У результаті розробки було створено UML-діаграму бази даних для відображення зв'язків між цими таблицями. Ця діаграма допомагає краще зрозуміти структуру бази даних та взаємозв'язки між її складовими.

Також, було створено UML-діаграму для відображення зв’язків між таблицями (рис. 2.5).

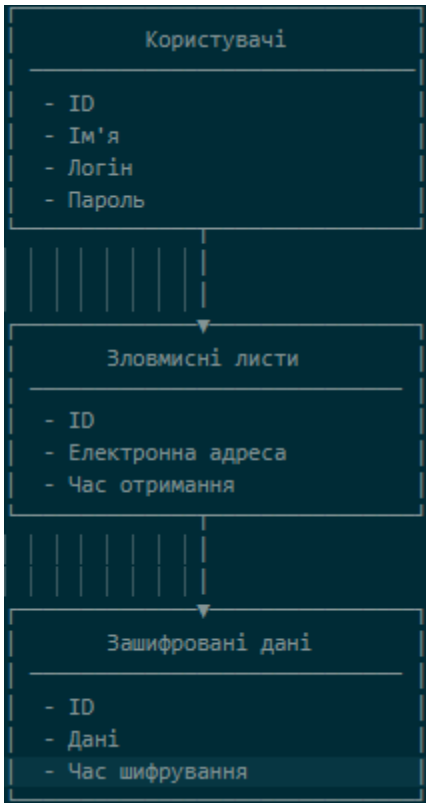


Рисунок 2.5 – UML-діаграма бази даних

В результаті було отримано повністю сформовану архітектуру сервісу та його коспонентів. Важливим є розбиття процесу розробки на менші етапи, оскільки це дуже допомагає під час практичної реалізації розуміти послідовність дій. Також, важливим є розуміння побудови сервісу для обрання мови програмування, яка найкраще підходить під даний проект.

2.2 **Теоретична розробка клієнта**

Після розробки архітектури поштового клієнта із захистом від фішингу необхідно розробити модуль захисту. Для початку потрібно відобразити алгоритм захисту даних. Найважливішими процесами для захисту даних у цьому випадку є:

1. Аналіз вхідних листів. Кожен вхідний лист повинен бути перевірений за допомогою VirusTotal API. Це дозволить виявити потенційно небезпечні вкладення або посилання, які можуть містити фішингові атаки чи інші загрози..
2. Шифрування даних. Для захисту конфіденційної інформації від викрадення у разі злому сервера чи бази даних необхідно використовувати алгоритми шифрування для безпечного зберігання листів та вкладень.

3. Зберігання результатів перевірки. Результати сканування листів за допомогою VirusTotal API повинні бути збережені у базі даних для подальшого аналізу та звітності.

Загальна схема роботи модулю захисту зображена на діаграмі (рис. 2.4).

Діаграми автоматів застосовуються для моделювання поведінки системи. Вони описують поведінку окремих екземплярів сутностей та варіантів їх використання, таких як класи, об'єкти, компоненти, вузли чи навіть цілі системи. В цих діаграмах відображаються стани екземплярів сутностей та переходи між ними протягом життєвого циклу, від створення до знищення.

Іншими словами, діаграми автоматів показують, як змінюються стани об'єктів у системі під впливом певних подій або дій. Це дозволяє наочно бачити, як система реагує на різні стимули, забезпечуючи більш глибоке розуміння алгоритмів і логіки функціонування системи.

Вибір схеми шифрування для поштового клієнта з використанням VirusTotal API повинен ґрунтуватися на таких факторах, як зручність використання, безпека, продуктивність і підтримка. Алгоритм симетричного шифрування Fernet має наступні особливості в порівнянні з іншими схемами шифрування:

1. Зручність використання: Fernet забезпечує простий і зрозумілий інтерфейс для шифрування та дешифрування даних. Для його використання потрібно лише кілька рядків коду, що робить його ідеальним для швидкої інтеграції у проекти.

2. Безпека: Fernet використовує AES-128[10] у режимі CBC для шифрування, що є одним із найнадійніших алгоритмів шифрування. Крім того, він застосовує HMAC з SHA256[11] для автентифікації даних, забезпечуючи їх цілісність і захист від підробки.

3. Продуктивність: Алгоритм Fernet оптимізований для швидкого шифрування та дешифрування даних. Це важливо для поштового клієнта, який обробляє великі обсяги даних у реальному часі.

4. Підтримка та документація: Fernet є частиною бібліотеки `cryptography`, яка має широке співтовариство користувачів і хорошу документацію. Це забезпечує легкість у використанні та наявність ресурсів для вирішення можливих проблем.

5. Надійність: Бібліотека `cryptography`, яка включає Fernet, пройшла численні перевірки безпеки і широко використовується в індустрії. Це підтверджує її надійність і безпечність для використання у критичних додатках[12].

6. Підтримка різних платформ: cryptography підтримує різні платформи і мови програмування, що дозволяє легко інтегрувати Fernet у різні системи без значних змін коду.

7. Інтеграція з існуючими системами: Fernet може бути легко інтегрований у поточну архітектуру поштового клієнта з використанням VirusTotal API, що дозволяє забезпечити додатковий рівень безпеки без значних змін у системі.

Вибір алгоритму симетричного шифрування Fernet для поштового клієнта з використанням VirusTotal API є виправданим завдяки його надійності, зручності використання, високій продуктивності та широкій підтримці. Це робить його ідеальним вибором для забезпечення безпеки даних у поштовому клієнті, захищеному від фішингових атак.

Хоча алгоритм симетричного шифрування Fernet є надійним вибором для нашої програми поштового клієнта з використанням VirusTotal API, розглянемо можливості інших методів шифрування:

1. GnuPG (GPG):

– переваги: GnuPG використовує відкриті стандарти для шифрування, що забезпечує високу надійність та безпеку. GnuPG підтримує як симетричне, так і асиметричне шифрування, що дозволяє гнучко керувати ключами та підписами;

– недоліки: Для нових користувачів налаштування та використання GnuPG може бути складним. Для інтеграції з PHP можуть знадобитися додаткові розширення та конфігурації.

2 Libgcrypt:

– переваги: Libgcrypt використовує алгоритми, які відповідають сучасним стандартам безпеки. Бібліотека підтримує широкий спектр криптографічних операцій, включаючи шифрування, хешування та цифрові підписи;

– недоліки Вимагає детальних знань про криптографію для ефективного використання. Може бути менш ефективною для великих обсягів даних у

порівнянні з іншими бібліотеками.

3. Bouncy Castle:

– перевага: Підтримує кілька мов програмування, включаючи Java та C#, що робить його універсальним рішенням для різних проектів. Підтримує широкий спектр криптографічних алгоритмів, що дозволяє вибрати оптимальний метод для конкретних потреб;

– недоліки: Для інтеграції з PHP можуть знадобитися додаткові модулі або бібліотеки. Деякі користувачі повідомляють про недостатню документацію, що може ускладнити початкове налаштування

Ці альтернативні методи можуть бути розглянуті залежно від конкретних потреб та обмежень проекту. Кожен з них має свої переваги та недоліки, які варто враховувати при виборі методу шифрування для вашої програми.

Зважаючи на потреби та вимоги нашої програми, вибір симетричного шифрування Fernet є обґрунтованим. Fernet використовує безпечні алгоритми AES-128 у режимі CBC та HMAC з SHA256 для забезпечення високого рівня безпеки даних. Основними перевагами використання Fernet є його швидкодія, ефективність та простота в реалізації.

Симетричне шифрування Fernet дозволяє здійснювати як шифрування, так і дешифрування даних за допомогою одного ключа, що спрощує процес керування ключами та зменшує ризик їх перехоплення. Крім того, використання HMAC з SHA256 гарантує цілісність даних та захист від несанкціонованих змін.

Незважаючи на ці переваги, існують деякі потенційні виклики та загрози. Наприклад, уразливість симетричного ключа може призвести до компрометації всіх зашифрованих даних. Тому важливо вживати заходів безпеки для захисту ключів від несанкціонованого доступу.

Крім того, розвиток квантових комп'ютерів може створити нові виклики для симетричного шифрування. Хоча на сьогоднішній день AES-128 є надійним, розвиток квантових технологій може створити нові загрози для безпеки даних.

Загалом, симетричне шифрування Fernet є ефективним та надійним методом захисту даних для нашої програми. Його використання дозволить забезпечити високий рівень безпеки та ефективності у роботі з обміном чутливою інформацією.

AES-128 у режимі CBC (Cipher Block Chaining) є одним із найпоширеніших алгоритмів симетричного шифрування. У цьому режимі кожен блок даних шифрується окремо, а потім результат шифрування використовується для шифрування наступного блоку. Це забезпечує високий рівень безпеки, оскільки навіть невеликі зміни в вхідних даних призводять до значних змін у вихідному шифрованому тексті. Крім того, використання випадкового ініціалізаційного вектора (IV) додає додатковий рівень випадковості до процесу шифрування.

HMAC (Hash-based Message Authentication Code) з SHA256 є алгоритмом для створення аутентичного коду повідомлення на основі хеш-функції SHA256. HMAC використовується для перевірки цілісності та автентифікації даних шляхом обчислення хеш-коду повідомлення з використанням ключа. Використання SHA256 для обчислення хешу забезпечує високий рівень безпеки та надійності, оскільки ця хеш-функція вважається стійкою до різних атак, включаючи brute-force і collision attacks..

Використання комбінації AES-128 у режимі CBC та HMAC з SHA256 дозволяє забезпечити високий рівень безпеки та надійності для шифрування та аутентифікації даних. Ці алгоритми є добре вивченими та широко використовуються в криптографічних застосунках, забезпечуючи ефективний захист від різних видів атак та забезпечуючи конфіденційність та цілісність переданих даних.

2.3 **Обґрунтування вибору мови програмування і бази даних**

Після завершення розробки архітектури проектування поштового клієнта із захистом від фішингу з використанням VirusTotal API, наступним кроком є обґрунтування вибору мови програмування та системи керування базами даних (СКБД). Важливо врахувати функціональні вимоги проекту, ефективність

розробки, безпеку, швидкість та доступність необхідних інструментів для обробки даних та взаємодії з API.

Python був обраний як мова програмування для реалізації поштового клієнта з декількох причин. По-перше, Python є однією з найпопулярніших та найбільш розповсюджених мов програмування, що використовуються в сфері веб-розробки та програмування загалом. Його простий та зрозумілий синтаксис сприяє швидкій розробці програм та полегшує спільну роботу великих команд програмістів. Крім того, Python має велику кількість розширень та бібліотек, які спрощують роботу з мережевими запитами, взаємодією з API та обробкою даних.

Python - це високорівнева, інтерпретована та динамічна мова програмування, яка має безліч переваг і розширюється великою спільнотою розробників. Однією з основних переваг Python є його простота та зрозумілість синтаксису, що робить його дуже доступним для початківців і приємним для досвідчених програмістів. Відсутність необхідності в строгих типах даних дозволяє швидше та легше розробляти програми, а також спрощує їхнє читання та розуміння[13].

Другою важливою перевагою Python є його багата екосистема бібліотек і модулів. Велика кількість сторонніх бібліотек дозволяє розробникам ефективно виконувати широкий спектр завдань, включаючи мережеве програмування, обробку даних, шифрування, аналіз тексту та багато іншого. Багато з цих бібліотек мають високу якість та активну підтримку зі сторони спільноти, що полегшує розробку програм та забезпечує їхню стабільність.

Крім того, Python є кросплатформеною мовою програмування, що означає, що програми, написані на Python, можуть працювати на будь-якій операційній системі, включаючи Windows, macOS та Linux. Це робить Python дуже універсальним і зручним для розробки програм, які повинні працювати на різних платформах.

Ще однією важливою перевагою Python є його висока продуктивність та швидкість розробки. Мова має простий та зрозумілий синтаксис, а також велику кількість корисних інструментів, що дозволяють розробникам швидко створювати прототипи, тестувати ідеї та розробляти функціональні програми за короткий час.

Це особливо важливо для початкових етапів розробки, коли важливо швидко перевіряти функціональність та ефективність рішення.

Крім того, Python має велику та активну спільноту розробників, яка надає безліч ресурсів, документації, прикладів коду та форумів, де можна знайти допомогу та поради. Це дозволяє розробникам швидше вирішувати проблеми та підвищує якість їхнього коду.

Загалом, Python - це потужний та універсальний інструмент, який має широкі можливості в різних сферах програмування. Його простота, ефективність та широка підтримка роблять його ідеальним вибором для реалізації різноманітних проектів, включаючи розробку поштового клієнта з захистом від фішингу.

Щодо вибору системи керування базами даних, було обрано MySQL з-за його надійності, широкого використання та досить широкого набору можливостей. MySQL є відкритою та безкоштовною реляційною системою управління базами даних, яка дозволяє зберігати та керувати великим обсягом даних. Вона має різноманітні типи таблиць, що дозволяє використовувати різні стратегії зберігання даних в залежності від їхньої природи та потреб проекту.

MySQL, як безкоштовна та відкрита реляційна система управління базами даних, володіє багатьма перевагами, які роблять її привабливим вибором для широкого спектра проектів. По-перше, MySQL є однією з найпопулярніших та найбільш використовуваних СКБД у світі. Це означає, що для нього існує велика спільнота користувачів та розробників, яка активно підтримує та розвиває цю систему. Багато великих компаній, таких як Google, Facebook та Twitter, використовують MySQL для зберігання своїх даних, що свідчить про його надійність та ефективність[14].

Другою важливою перевагою MySQL є його широкий функціонал та гнучкість. Система підтримує різноманітні типи даних, операції та запити, що дозволяє розробникам створювати складні та потужні бази даних для різноманітних застосувань. Крім того, MySQL має різні типи таблиць, такі як MyISAM та InnoDB, з різними можливостями, що розширюють функціональність та ефективність.

Важливою перевагою MySQL є його висока продуктивність та швидкодія. Система оптимізована для роботи з великими обсягами даних та високими навантаженнями, що дозволяє розробникам створювати веб-додатки та сервіси, які працюють ефективно та швидко навіть при великому обсязі трафіку.

Нарешті, MySQL має багато вбудованих інструментів розробки, які спрощують роботу з базами даних. Від редакторів запитів та візуальних засобів керування об'єктами до інструментів моніторингу та адміністрування, MySQL надає розробникам все необхідне для створення та підтримки потужних та надійних баз даних.

Також MySQL має інтегровані засоби для роботи з базами даних, включаючи редактори запитів, візуальні засоби керування об'єктами та інші інструменти, які спрощують розробку та адміністрування. Його надійність та продуктивність роблять його ідеальним вибором для розробки навіть великих та вимогливих додатків. MySQL забезпечує швидкий доступ до даних та ефективне їх зберігання, що є критично важливим для веб-сервісів з великою кількістю користувачів та обробки великого обсягу інформації.

Узагальнюючи, вибір Python як мови програмування та MySQL як СКБД обґрунтований на їхній популярності, широкому використанні та надійності, що дозволить забезпечити ефективну та безпечну реалізацію поштового клієнта з високим рівнем захисту від фішингу.

2.4 **Висновки до розділу 2**

У даному розділі розглянуто обґрунтування вибору мови програмування та бази даних для реалізації поштового клієнта з захистом від фішингу, використовуючи VirusTotal API..

Для розробки поштового клієнта обрано мову програмування Python. Цей вибір зумовлений його високою ефективністю, простотою синтаксису та широким спектром бібліотек, які спрощують розробку програмного забезпечення. Python є однією з найпопулярніших мов програмування у світі, що забезпечує доступ до великої кількості ресурсів та документації для підтримки розробки. Крім того,

Python відомий своєю гнучкістю та швидкістю розробки, що робить його ідеальним вибором для проектів з обмеженими строками та потребою в постійній інтеграції нових функціональних можливостей.

У якості бази даних для зберігання та управління інформацією про користувачів, повідомлення та інші дані поштового клієнта використовується MySQL. Ця реляційна система управління базами даних є надійним та потужним інструментом, який забезпечує ефективне зберігання та оптимізований доступ до даних. MySQL має широкий спектр функціональних можливостей, що відповідають вимогам поштового клієнта, включаючи підтримку транзакцій, безпеку даних та розширені можливості керування даними. Крім того, MySQL є вільно розповсюджуваною системою, що забезпечує доступ до великої спільноти розробників та регулярні оновлення для покращення функціональності та безпеки.

Обрані методи шифрування та вбудовування водяних знаків в текстури обґрунтовані їхньою ефективністю та стандартами безпеки. Використання алгоритмів шифрування RSA для забезпечення конфіденційності та цілісності даних, а також водяних знаків для захисту від несанкціонованого використання текстур, є ключовими аспектами безпеки поштового клієнта. Ці методи забезпечують надійний захист даних та конфіденційність інформації про користувачів, збереження їхньої приватності та запобігання недозволеному доступу до особистих даних.

Висновуючи, вибір Python та MySQL для розробки поштового клієнта з захистом від фішингу є обґрунтованим та оптимальним рішенням, яке забезпечує ефективну реалізацію функціональності та надійний захист даних користувачів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

В розділі програмної реалізації буде розглянуто деталі виконання поштового клієнта з захистом від фішингу за допомогою VirusTotal API. Цей розділ включатиме опис архітектури програми, ключові етапи розробки та взаємодії з API, а також можливості валідації та обробки даних..

3.1 Розробка поштового клієнта

Першим етапом розробки поштового клієнта із захистом від фішингу є створення макету дизайну майбутнього застосунку. Цей етап є дуже важливим у розробці будь-якого програмного продукту, оскільки користувач буде взаємодіяти саме з інтерфейсом застосунку, а не напряму з його внутрішніми компонентами, такими як сервер чи база даних. Для розробки дизайну було обрано сервіси Figma та Wix.

Figma – це популярний інструмент для створення дизайну, який дозволяє створювати макети веб-сайтів, мобільних додатків, презентації та інші візуальні елементи. Цей застосунок забезпечує зручний та інтуїтивно зрозумілий інтерфейс для дизайнерів, що сприяє швидкому і ефективному створенню дизайнів.

Wix – це онлайн-платформа для створення веб-сайтів, яка також була використана для розробки лендінг пейджу поштового клієнта. Вона дозволяє швидко створювати адаптивні веб-сайти з використанням готових шаблонів та вбудованих інструментів для дизайну.

Розробка дизайну поштового клієнта полягала у створенні зручного та інтуїтивно зрозумілого інтерфейсу для користувача. В результаті було створено макет веб-сервісу, який містить всі необхідні сторінки та елементи, включаючи головну сторінку, сторінку перегляду листів, форму для створення нових листів та сторінки для налаштування безпеки (рис. 3.1).

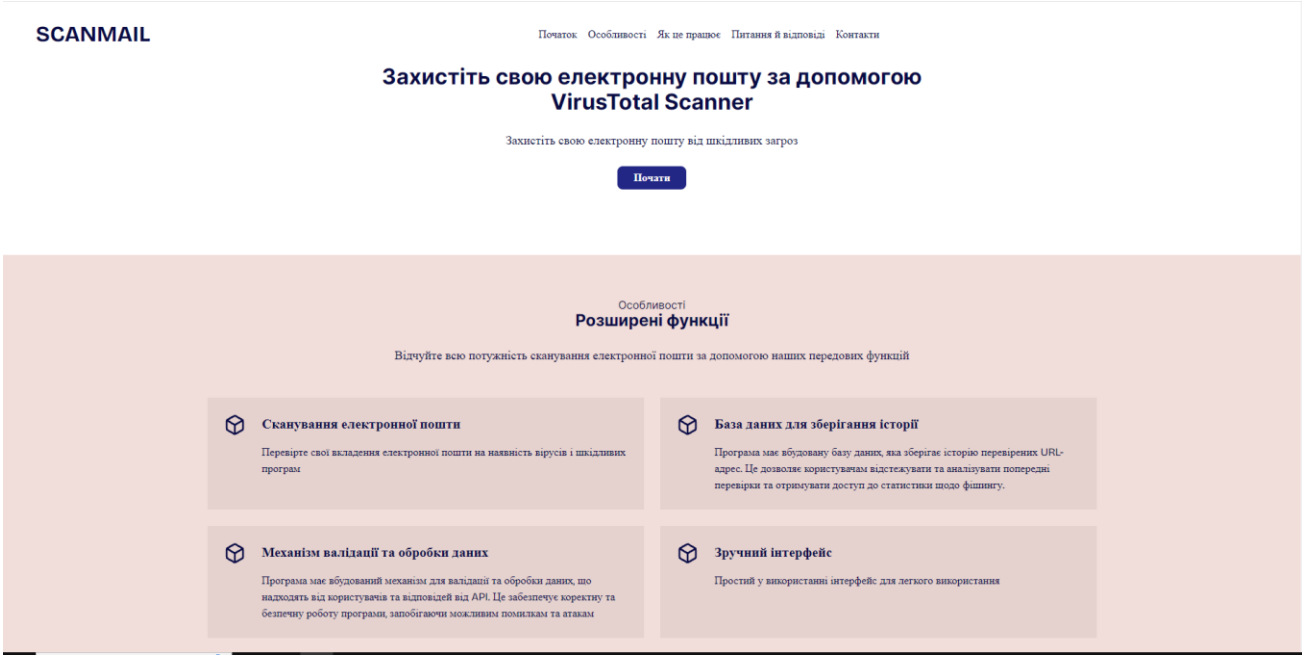


Рисунок 3.1 – Розроблений дизайн

Після створення дизайну наступним важливим кроком стала реалізація візуальної частини поштового клієнта. Основою для побудови інтерфейсу послужили мова розмітки HTML 5 та каскадна таблиця стилів CSS 3, які допомогли структурувати контент і надати йому привабливого вигляду[15].

HTML-шаблон було інтегровано в проект за допомогою Flask-шаблонізатора Jinja2. Це дозволяє динамічно взаємодіяти з базою даних, забезпечуючи користувачів актуальною інформацією. Інформація про електронні листи обробляється та передається у вигляді структурованого HTML-документу. Ось приклад, як відбувається обмін даними між Python (Flask) та HTML:

```
@app.route('/')
def home():
    messages = fetch_messages_from_db()
    return render_template('home.html', messages=messages)
```

HTML-шаблон для відображення списку електронних листів виглядає наступним чином:

```
<div class="email-container">
    <div class="container-header">
        <h1>Вхідні</h1>
    </div>
    <div class="email-list">
        {% for message in messages %}
            <div class="email-item">
```

```
            <h2>{{ message.subject }}</h2>
            <p><strong>Від:</strong> {{ message.sender }}</p>
            <p><strong>Дата:</strong> {{ message.date }}</p>
            <a href="/message/{{ message.id }}">Детальніше</a>
          </div>
        {% endfor %}
      </div>
    </div>
```

Цей приклад показує, як Flask разом із Jinja2 дозволяє динамічно відображати дані з бази даних в інтерфейсі користувача. Кожне повідомлення електронної пошти представляється як окремий елемент з інформацією про тему, відправника, дату та посиланням для детального перегляду[16].

Таким чином, реалізація візуальної частини поштового клієнта забезпечує користувачам зручний і зрозумілий інтерфейс для роботи з електронними листами, роблячи взаємодію з сервісом максимально комфортною та безпечною.

Ключовим елементом поштового клієнта є повідомлення електронної пошти. Ці повідомлення зберігаються, оновлюються та перевіряються на безпеку, тому важливо створити клас для змінної «messages», що представляє собою електронне повідомлення

У змінній «messages» міститься детальна інформація про кожне повідомлення. Для відображення всіх повідомлень необхідно по черзі обробити всі записи в базі даних, відсортувати їх і представити в інтерфейсі користувача::

```
@app.route('/inbox')
def inbox():
  messages = fetch_all_messages()
  return render_template('inbox.html', messages=messages)
```

HTML-шаблон для відображення списку електронних повідомлень:

```
<div class="inbox__list">
  {% if messages %}
    <div class="list__wrapper">
      {% for message in messages %}
        <a href="/message/{{ message.id }}" class="list__item">
          <div class="item__details">
```

```
                <h2 class="item__subject">{{ message.subject }}</h2>
                <p class="item__sender">Від: {{ message.sender }}</p>
                <p class="item__date">Дата: {{ message.date }}</p>
              </div>
            </a>
          {% endfor %}
        </div>
      {% else %}
        <p>У вас немає нових повідомлень.</p>
      {% endif %}
    </div>
```

Цей код показує, як за допомогою Flask і шаблонізатора Jinja2 можна динамічно відображати список електронних повідомлень з бази даних у зручному для користувача вигляді. Кожне повідомлення представляється як окремий елемент з інформацією про тему, відправника та дату надходження.

Завдяки такій реалізації, користувачі отримують можливість зручно переглядати свої електронні листи, швидко знаходити потрібну інформацію та бути впевненими у безпеці своєї пошти.

Ця частина поштового клієнта виконує роль посередника між користувачем та базою даних, забезпечуючи безпечне збереження і обробку повідомлень. Сервер не тільки отримує повідомлення, але й виконує їх перевірку на фішинг, шифрування та збереження у базі даних, забезпечуючи надійний захист користувачів.

На етапі розробки сервісу необхідно запустити сервер і налаштувати обмін даними з базою даних. Нижче наведено приклад реалізації на Flask:

```
from flask import Flask, request, jsonify
from database import save_message, get_messages
from security import check_for_phishing, encrypt_message

app = Flask(__name__)

@app.route('/api/messages', methods=['POST'])
def receive_message():
```

```
    data = request.json
    if check_for_phishing(data['content']):
        encrypted_content = encrypt_message(data['content'])
        save_message(data['sender'], data['subject'], encrypted_content)
        return jsonify({'status': 'success', 'message': 'Message successfully encrypted and stored'}),
200
    else:
        return jsonify({'status': 'failed', 'message': 'Phishing detected in the message content'}), 400

@app.route('/api/messages', methods=['GET'])
def fetch_messages():
    messages = get_messages()
    return jsonify(messages), 200

if __name__ == '__main__':
    app.run(debug=True)
```

Цей фрагмент коду показує, як Flask використовується для створення API, яке обробляє запити на отримання та збереження повідомлень. Повідомлення проходять перевірку на фішинг, шифруються та зберігаються в базу даних, забезпечуючи безпеку даних користувачів.

Функції для перевірки на фішинг та шифрування повідомлень:

```
def check_for_phishing(content):
    # Логіка перевірки вмісту на фішинг
    return 'phishing' not in content

def encrypt_message(content):
    # Логіка шифрування повідомлення
    return f"encrypted_{content}"
```

Реалізація сервера забезпечує безпеку та цілісність повідомлень користувачів. Сервер виконує роль шифрувальника та захисника, перевіряючи кожне повідомлення на наявність фішингових загроз і шифруючи його перед збереженням у базі даних.

Таким чином, користувачі можуть бути впевнені у безпеці своїх повідомлень, знаючи, що їхні дані захищені на кожному етапі обробки. Цей підхід гарантує, що навіть у разі компрометації бази даних, вміст повідомлень залишиться захищеним завдяки надійному шифруванню.

Після запуску серверу необхідно налаштувати зберігання даних для поштового клієнта. Оскільки в нашому випадку не використовуються традиційні таблиці бази даних, ми зосередимося на інших важливих елементах, таких як обробка повідомлень, авторизація користувачів та забезпечення безпеки.

Основні компоненти зберігання даних включають зберігання повідомлень, авторизацію користувачів та налаштування безпеки. Кожне повідомлення зберігається у вигляді об'єкта, який містить інформацію про відправника, тему, вміст, дату надходження, статус перевірки на фішинг та шифроване вміст. Це забезпечує зручну роботу з повідомленнями та дозволяє легко здійснювати перевірку на фішинг і шифрування вмісту.

Для забезпечення безпеки користувачів важливо налаштувати систему авторизації. Кожен користувач має унікальні облікові дані, що включають логін, зашифрований пароль, електронну пошту та роль (адміністратор або користувач). Безпека є ключовим аспектом сервісу, тому необхідно реалізувати шифрування повідомлень перед зберіганням, регулярну перевірку на фішинг та зберігання логів дій користувачів для моніторингу безпеки

На етапі реалізації потрібно створити структури для зберігання даних у форматі, який зручно використовувати у програмному коді. Ось приклад, як це можна реалізувати на Python з використанням Flask та SQLAlchemy:

```
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
```

```
    username = db.Column(db.String(150), nullable=False, unique=True)
    email = db.Column(db.String(150), nullable=False, unique=True)
    password_hash = db.Column(db.String(256), nullable=False)
    role = db.Column(db.String(50), nullable=False, default='user')

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sender = db.Column(db.String(150), nullable=False)
    subject = db.Column(db.String(200), nullable=False)
    content = db.Column(db.Text, nullable=False)
    date_received = db.Column(db.DateTime, default=datetime.utcnow)
    phishing_status = db.Column(db.Boolean, default=False)
    encrypted_content = db.Column(db.Text, nullable=True)

    def encrypt_content(self):
        self.encrypted_content = f"encrypted_{self.content}"
```

Цей приклад демонструє, як можна реалізувати зберігання даних для повідомлень та користувачів, забезпечуючи необхідний рівень безпеки. Кожен об'єкт повідомлення містить інформацію про відправника, тему, вміст, дату надходження, статус перевірки на фішинг та зашифрований вміст. Об'єкт користувача зберігає логін, електронну пошту, зашифрований пароль та роль користувача.

Завдяки такій реалізації, поштовий клієнт забезпечує надійне зберігання та обробку повідомлень, авторизацію користувачів та забезпечення безпеки даних на кожному етапі.

Наступним етапом було налаштування зберігання даних для повідомлень та користувачів. Важливо зазначити, що система авторизації користувачів включає зберігання паролів у вигляді хешів, щоб підвищити захищеність даних. Завдяки цьому, навіть у разі злому сервісу, зловмисники не отримають доступ до паролів користувачів у відкритому вигляді.

Для зберігання облікових даних користувачів було створено структуру, яка включає такі атрибути: ім'я користувача, електронну пошту, зашифрований пароль та роль користувача. Паролі зберігаються у вигляді хешів, що забезпечує додатковий рівень безпеки.

Також була налаштована система зберігання повідомлень, яка включає такі атрибути: відправник, тема, вміст, дата надходження, статус перевірки на фішинг та шифрований вміст. Це дозволяє ефективно обробляти та зберігати повідомлення, забезпечуючи їхню безпеку та конфіденційність.

В результаті ми отримали повністю робочий сервіс, готовий до експлуатації. Розроблений інтерфейс враховує комфорт користувача при взаємодії з поштовим клієнтом, а сервер забезпечує надійне з'єднання між графічною частиною та системою зберігання даних. Наступним етапом роботи є реалізація розробленого захисного модуля та його інтеграція в проект..

3.2 **Розробка модулю захисту та його інтеграція**

Модуль захисту поштового клієнта складається з двох основних компонентів: перевірка на фішинг за допомогою VirusTotal API та шифрування даних з використанням алгоритму Fernet. Першим було реалізовано шифрування повідомлень.

Загалом процес захисту повідомлень включає два етапи. Перший етап – це шифрування даних перед зберіганням, другий – розшифрування даних під час їх перегляду користувачем. Нижче описано процес шифрування та розшифрування повідомлень за допомогою симетричного шифрування з використанням бібліотеки Fernet:

1. Генерація ключа шифрування. Спочатку генерується ключ шифрування за

допомогою функції Fernet.generate_key().

2. Створення об'єкта Fernet з використанням ключа шифрування.

3. Шифрування повідомлення. Функція зчитує вміст повідомлення у текстовому форматі та шифрує його за допомогою методу fernet.encrypt().

4. Запис зашифрованого повідомлення.

5. Розшифрування зашифрованого повідомлення під час перегляду.

6. Функція зчитує зашифровані дані з бази даних.

7. Зашифровані дані розшифровуються за допомогою методу fernet.decrypt().

8. Розшифровані дані записуються у змінну для відображення користувачеві.

У підсумку, цей модуль дозволяє шифрувати повідомлення за допомогою симетричного ключа та розшифровувати його при необхідності, забезпечуючи конфіденційність даних користувачів..

```
from cryptography.fernet import Fernet

# Генерація ключа шифрування
ENCRYPTION_KEY = Fernet.generate_key()

# Створення об'єкта Fernet з використанням ключа шифрування
fernet = Fernet(ENCRYPTION_KEY)

# Шифрування повідомлення
def encrypt_message(message):
    encrypted_message = fernet.encrypt(message.encode())
    return encrypted_message

# Розшифрування повідомлення
def decrypt_message(encrypted_message):
    decrypted_message = fernet.decrypt(encrypted_message).decode()
    return decrypted_message

# Приклад використання
original_message = "Це тестове повідомлення."
encrypted_message = encrypt_message(original_message)
```

```
print(f"Зашифроване повідомлення: {encrypted_message}")

decrypted_message = decrypt_message(encrypted_message)
print(f"Розшифроване повідомлення: {decrypted_message}");
```

Цей приклад демонструє, як за допомогою Fernet можна забезпечити шифрування та розшифрування повідомлень у поштовому клієнті. Зашифровані дані зберігаються у базі даних, а під час перегляду користувачем повідомлення розшифровується для відображення у зрозумілому вигляді.

Інтеграція з VirusTotal API забезпечує перевірку повідомлень на фішинг, допомагаючи виявляти та блокувати потенційно небезпечні листи. Основні етапи інтеграції включають отримання API ключа, налаштування запитів до VirusTotal та обробку результатів перевірки.

Основні кроки інтеграції:

1. Отримання API ключа від VirusTotal;
2. налаштування запитів до API для перевірки вмісту повідомлень та вкладень;
3. обробка результатів перевірки та відповідні дії (блокування або позначення небезпечних листів).

Після того як дані повідомлення були зашифровані, вони зберігаються в базі даних у вигляді захищеного файлу. Для відображення нового повідомлення на сторінці виконується її оновлення. Коли інший користувач відкриває це повідомлення, дані розшифровуються на сервері, після чого відображаються у зрозумілому вигляді на екрані. Таким чином, користувач отримує доступ до розшифрованого повідомлення для перегляду. Функція, яка відповідає за розшифрування даних, наведена нижче:

```
from cryptography.fernet import Fernet

# Генерація ключа шифрування (ключ має бути збережений для подальшого використання)
ENCRYPTION_KEY = Fernet.generate_key()

# Створення об'єкта Fernet з використанням збереженого ключа шифрування
fernet = Fernet(ENCRYPTION_KEY)
```

```
# Функція для розшифрування повідомлення
def decrypt_message(encrypted_message):
    decrypted_message = fernet.decrypt(encrypted_message).decode('utf-8')
    return decrypted_message

# Приклад використання
encrypted_message = b'... (зашифроване повідомлення) ...'
decrypted_message = decrypt_message(encrypted_message)
print(f"Розшифроване повідомлення: {decrypted_message}";
```

Ця функція отримує зашифроване повідомлення, розшифровує його за допомогою симетричного ключа Fernet і повертає оригінальний текст повідомлення.

Таким чином, коли користувач відкриває повідомлення, дані розшифровуються на сервері та відображаються у читабельному вигляді на його екрані. Це забезпечує надійний захист під час зберігання даних і гарантує, що тільки авторизовані користувачі можуть отримати доступ до вмісту повідомлень.

Після реалізації шифрування даних повідомлень наступним кроком стала інтеграція модуля захисту. Цей модуль включає в себе два основних компоненти: перевірку на фішинг за допомогою VirusTotal API та шифрування повідомлень з використанням алгоритму Fernet.

Процес захисту повідомлень складається з кількох важливих етапів. Перший етап включає шифрування повідомлення перед зберіганням у базі даних. Після цього повідомлення розшифровується під час його перегляду користувачем. Цей підхід гарантує, що дані залишаються захищеними на всіх етапах обробки.

Для додаткового захисту повідомлень інтегровано перевірку на фішинг за допомогою VirusTotal API. Це дозволяє виявляти та блокувати небезпечні повідомлення ще на етапі їх отримання.

Приклад реалізації перевірки на фішинг:

```
import requests
# Отримання API ключа
```

```
API_KEY = 'your-virustotal-api-key'


# Функція для перевірки повідомлення на фішинг
def check_for_phishing(content):
    url = "https://www.virustotal.com/vtapi/v2/url/report"
    params = {'apikey': API_KEY, 'resource': content}
    response = requests.get(url, params=params)
    result = response.json()
    return result['positives'] > 0
```

Завдяки реалізації шифрування повідомлень та інтеграції з VirusTotal API, поштовий клієнт забезпечує високий рівень захисту даних. Користувачі можуть бути впевнені у безпеці своїх повідомлень, знаючи, що їхні дані захищені як під час зберігання, так і під час обробки. Цей підхід гарантує захист від фішингових атак та несанкціонованого доступу до інформації.

3.3 **Опис функціоналу поштового клієнта**

Важливим є описати функціонал сервісу з точки зору користувача.

На лендінг пейджі **SCANMAIL** користувачі можуть ознайомитися з основними можливостями сервісу та його перевагами. Основна мета лендінг пейджу - підкреслити важливість перевірки електронних листів на наявність потенційних загроз. Інтеграція з API VirusTotal дозволяє сканувати вкладення та посилання на наявність вірусів та шкідливого програмного забезпечення, що забезпечує високий рівень захисту для користувачів (рис. 3.2).

На сторінці представлені основні функції сервісу та їх опис, що дозволяє користувачам швидко зрозуміти, як **SCANMAIL** може допомогти захистити їхню електронну пошту. Відвідувачі також можуть дізнатися більше про технічні аспекти та переваги використання сервісу.

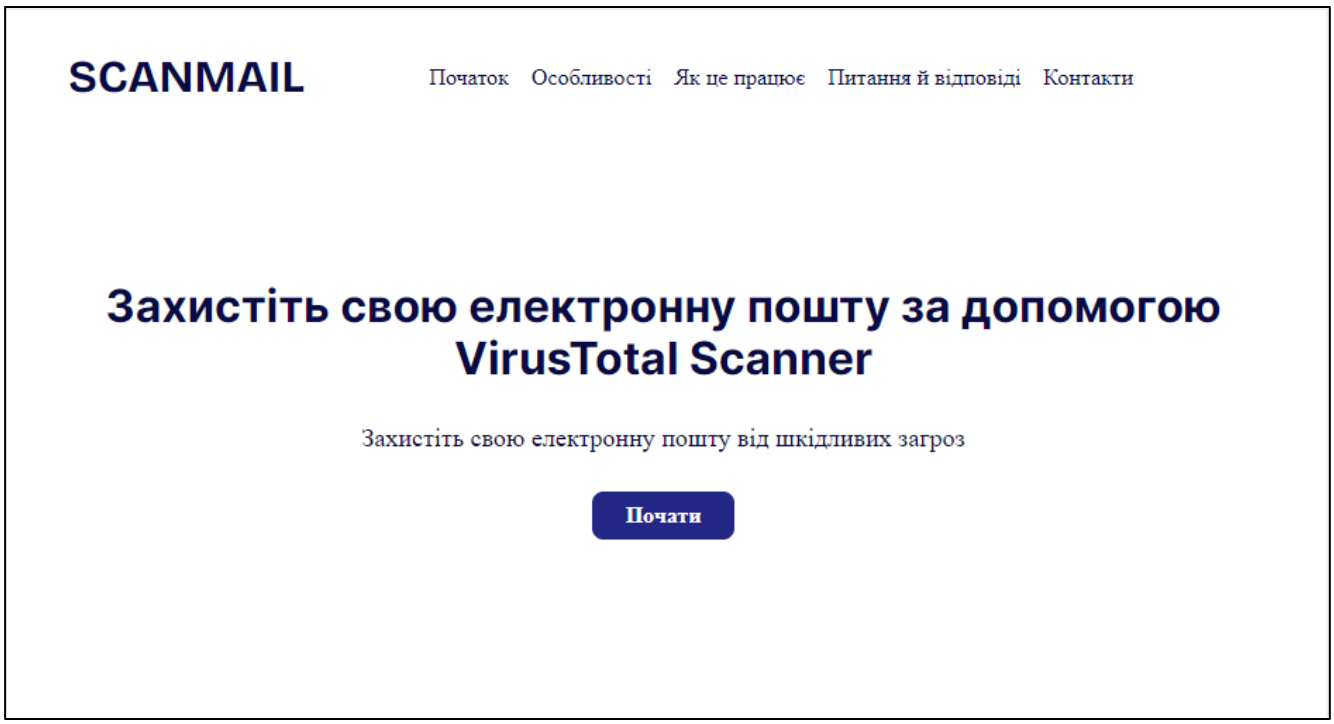


Рисунок 3.2 – Лендінг пейдж

SCANMAIL пропонує низку розширених функцій, які роблять цей сервіс унікальним і ефективним. Основні функції включають перевірку електронної пошти на наявність вірусів та шкідливих програм, збереження історії перевірок у базі даних, механізм валідації та обробки даних, а також зручний інтерфейс для легкого використання. (рис. 3.3).

Програма має вбудовану базу даних, яка зберігає історію перевірених URL-адрес. Це дозволяє користувачам відстежувати та аналізувати попередні перевірки, отримуючи доступ до статистики щодо фішингу. Механізм валідації та обробки даних забезпечує коректну та безпечну роботу програми, запобігаючи можливим помилкам та атакам. Зручний інтерфейс дозволяє користувачам легко взаємодіяти із сервісом та отримувати швидкий доступ до всіх функцій.

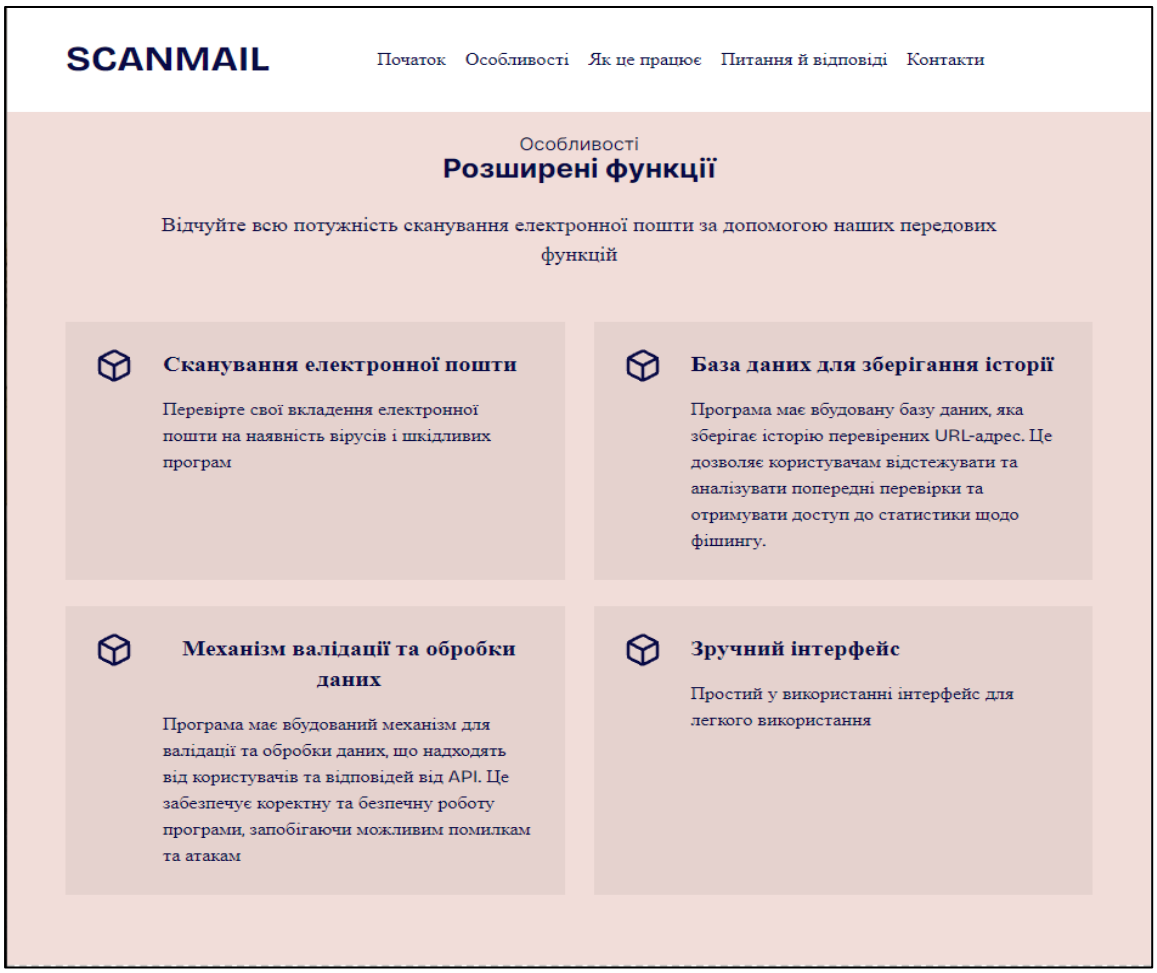


Рисунок 3.3 – Розширені функції

За допомогою **SCANMAIL** користувачі можуть захистити свою електронну пошту від шкідливих загроз. Сервіс інтегрується з VirusTotal API, що дозволяє ефективно виявляти та блокувати небезпечні листи. Це забезпечує надійний захист користувачів від фішингових атак та інших кіберзагроз, гарантуючи безпеку їхніх даних (рис. 3.4).

Користувачі можуть бути впевнені у безпеці своєї електронної пошти, знаючи, що їхні повідомлення перевіряються за допомогою передових технологій. Високий рівень захисту є однією з головних переваг **SCANMAIL**, що робить його привабливим для користувачів, які цінують конфіденційність та безпеку.

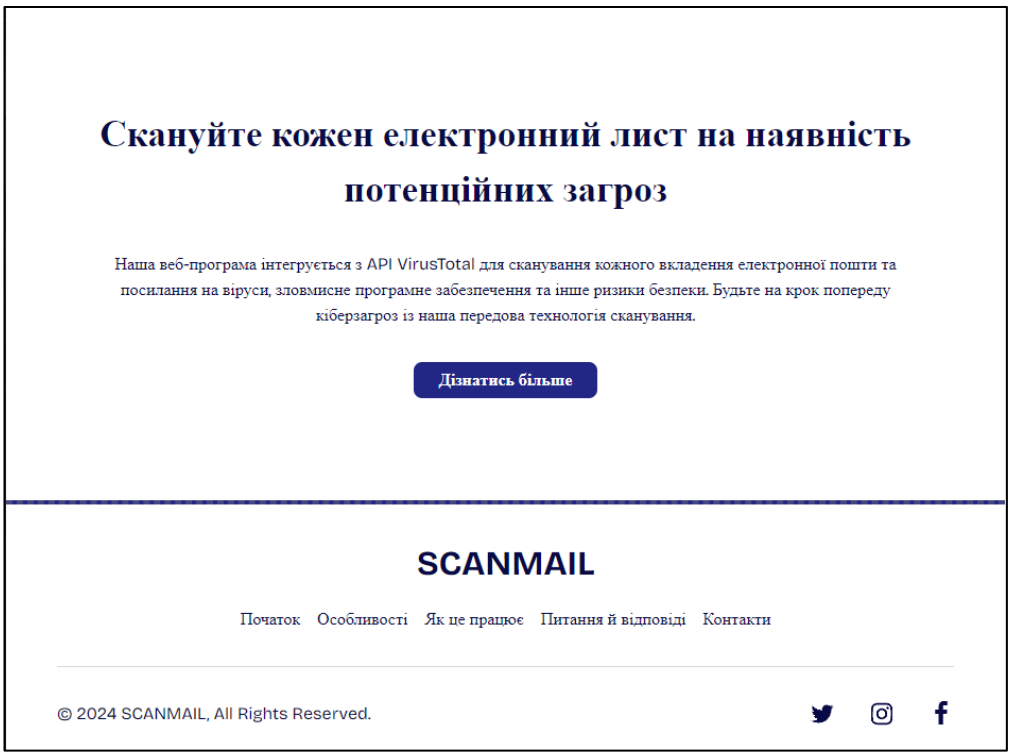


Рисунок 3.4 – Захист електронної пошти

На головній сторінці програми користувачі можуть налаштувати параметри сканування електронної пошти. Спочатку потрібно увійти до свого акаунту, вказавши електронну пошту та пароль. Після цього користувач може вибрати інтервал сканування: одноразове сканування, кожні 15 хвилин, кожні 30 хвилин або щогодини (рис. 3.5). Лістинг функції наведено нижче:

```
<form action="/scan" method="POST">
  <label for="email">Пошта:</label>
  <input type="email" id="email" name="email" required>
  <label for="password">Пароль:</label>
  <input type="password" id="password" name="password" required>
  <label for="interval">Інтервал сканування:</label>
  <select id="interval" name="interval">
    <option value="once">Один раз</option>
    <option value="15min">Кожні 15 хвилин</option>
    <option value="30min">Кожні 30 хвилин</option>
    <option value="hourly">Кожну годину</option>
  </select>
  <button type="submit">Почати сканування</button>
</form></form>
```

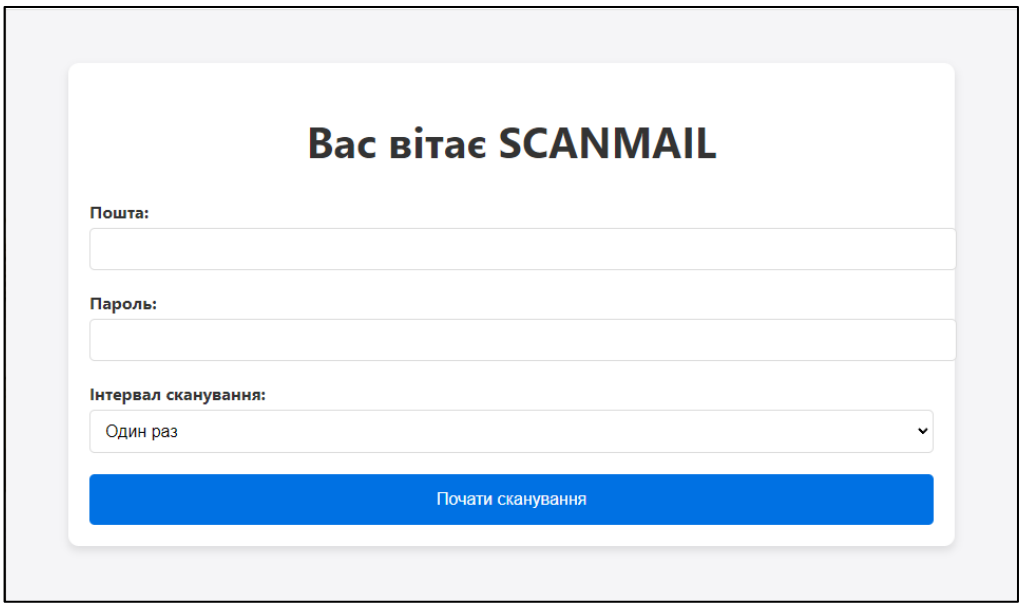


Рисунок 3.5 – Сторінка профілю користувача

Форма авторизації дозволяє користувачам увійти до свого акаунту та налаштувати параметри сканування. Після введення електронної пошти та паролю, користувач може вибрати інтервал сканування та натиснути кнопку "Почати сканування" для запуску процесу (рис. 3.6).

```
<form action="/login" method="POST">
  <label for="email">Пошта:</label>
  <input type="email" id="email" name="email" required>
  <label for="password">Пароль:</label>
  <input type="password" id="password" name="password" required>
  <button type="submit">Увійти</button>
</form>
```

Після успішної авторизації користувач має можливість налаштувати інтервал сканування.

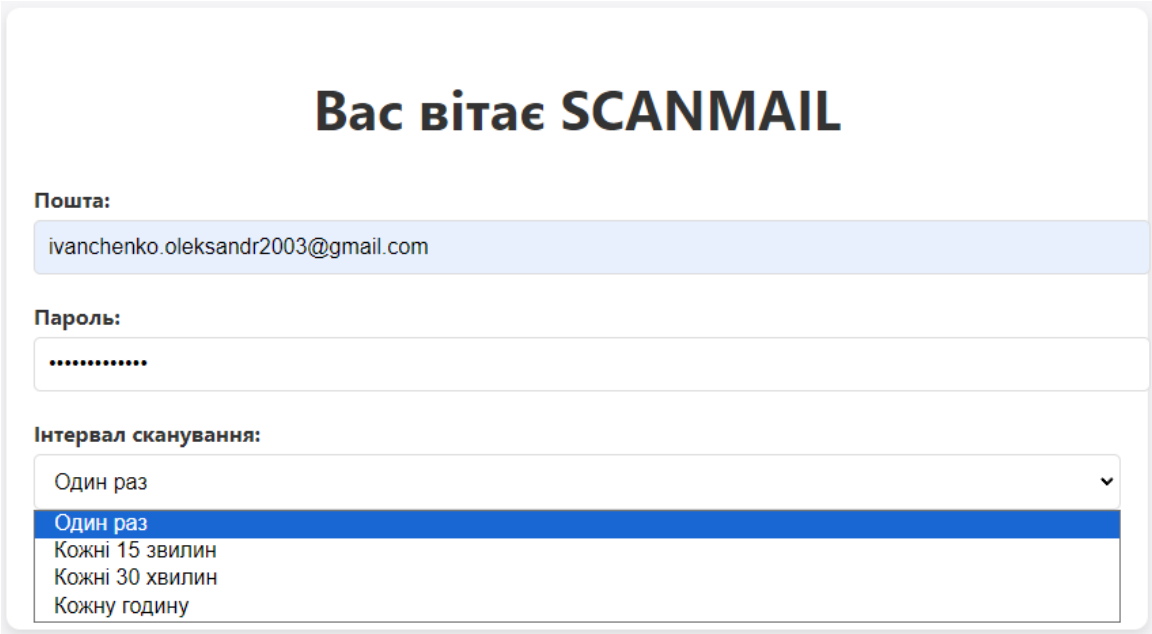


Рисунок 3.6 – Форма авторизації та налаштування сканування

Після натискання кнопки "Почати сканування" починається процес перевірки електронних листів на наявність загроз. Користувач може спостерігати за прогресом сканування через індикатор, який показує відсоток завершеного сканування. Це дозволяє користувачу бачити поточний стан сканування та знати, коли процес буде завершено (рис. 3.7).

```
@app.route('/scan', methods=['POST'])
def scan_emails():
    email = request.form['email']
    password = request.form['password']
    interval = request.form['interval']
    # Логіка для сканування електронних листів
    progress = 0
    while progress < 100:
        # Виконання сканування
        progress += 20
        time.sleep(1)
    return render_template('scan_result.html', progress=progress)
```

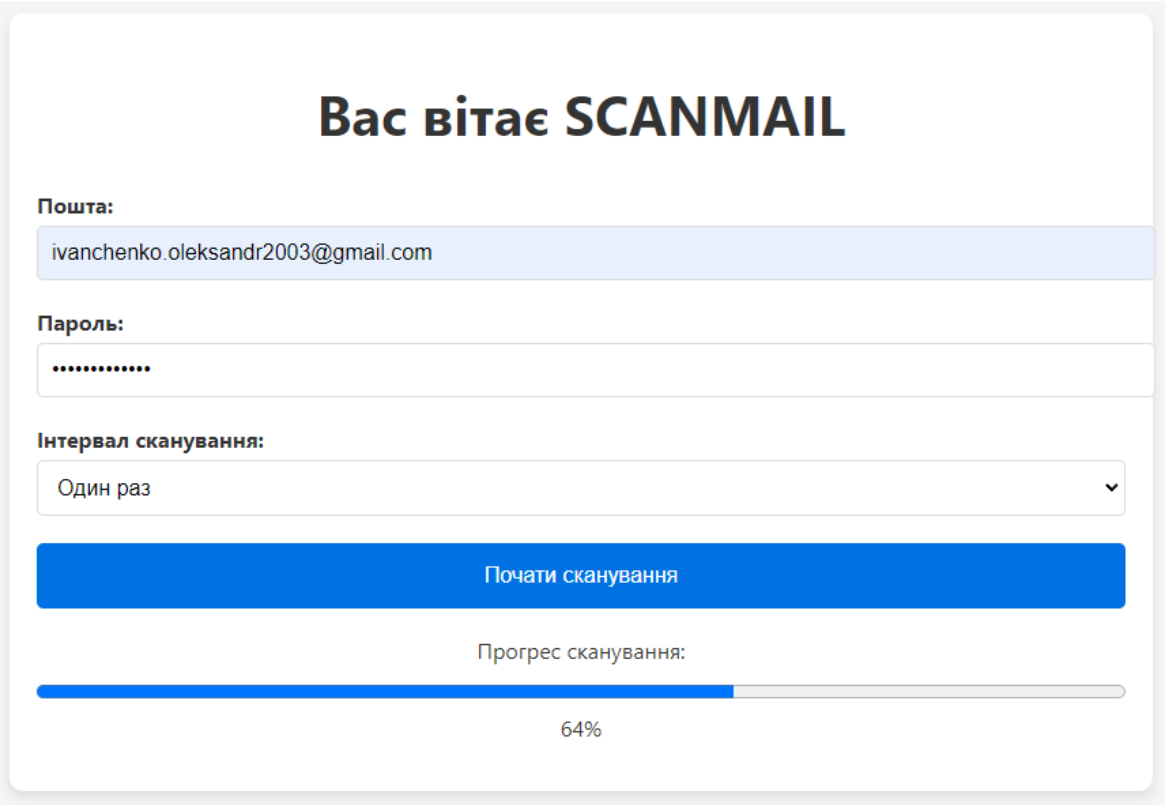


Рисунок 3.7 – Процес сканування

Після завершення сканування користувач отримує результати, які показують кількість виявлених підозрілих листів і посилань. Якщо були знайдені шкідливі листи або посилання, користувач отримує відповідне повідомлення з деталями. В результаті, був отримай повністю функціональний додаток (рис. 3.8).

```
<div class="scan-results">
  <h2>Результати сканування</h2>
  <p>На пошті ivanchenko.oleksandr2003@gmail.com знайдено 1 підозрілий лист.</p>
</div>
```

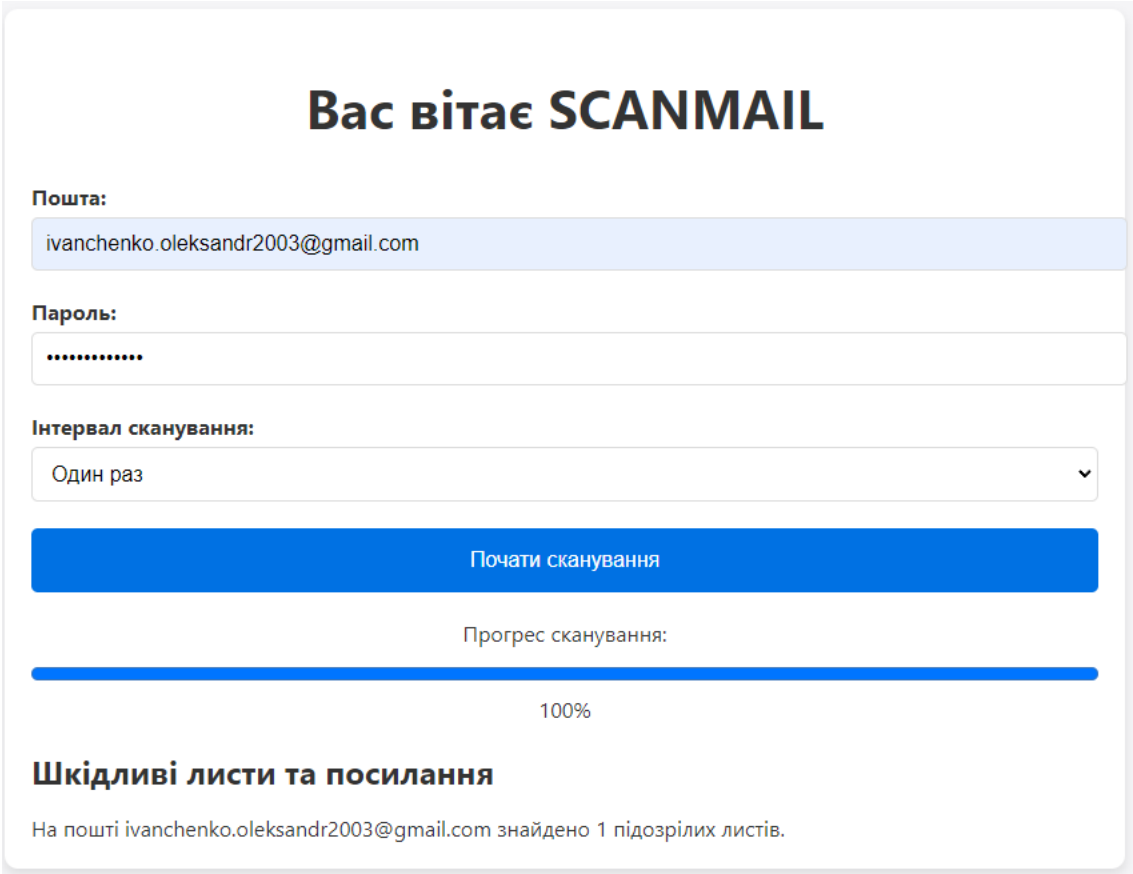


Рисунок 3.8 – Процес сканування

У підсумку, SCANMAIL є повнофункціональним додатком, який забезпечує високий рівень захисту електронної пошти. Основною перевагою сервісу є надійність та безпека, що робить його привабливим для користувачів, які цінують захист своїх даних у сучасному цифровому світі.

3.4 Висновки до розділу 3

Під час аналізу функціоналу сервісу **SCANMAIL** стало очевидним, що процес сканування електронної пошти на потенційні загрози є простим та зручним для користувачів. Всі етапи, починаючи від авторизації та налаштування параметрів сканування до перегляду результатів, реалізовані таким чином, щоб забезпечити максимальну зручність та зрозумілість для кінцевих користувачів.

У розділі "Опис функціоналу" докладно висвітлено роботу сервісу, включаючи розробку інтерфейсу, інтеграцію з VirusTotal API та механізми захисту. Скріншоти та приклади коду надають чітке уявлення про роботу проекту та його технічні аспекти.

Розділ "Розробка модуля захисту та його інтеграція" охоплює кроки створення та впровадження коду для шифрування повідомлень та перевірки на фішингові загрози. Модулі були ретельно інтегровані в основний функціонал сервісу для забезпечення безпеки користувачів та захисту їхніх даних.

Шифрування повідомлень здійснюється за допомогою бібліотеки Fernet, що гарантує надійний захист даних. Перевірка на фішинг за допомогою VirusTotal API додає ще один рівень захисту, забезпечуючи користувачів надійною інформацією про безпеку отриманих листів.

Ці кроки гарантують, що користувачі SCANMAIL можуть бути впевнені у безпеці своєї електронної пошти. Сервіс не лише захищає дані від потенційних загроз, але й робить це у зручній та зрозумілій формі, що є значною перевагою над іншими аналогічними рішеннями на ринку.

Завдяки аналізу функціоналу та інтеграції модулів захисту, можна зробити висновок, що SCANMAIL є високоефективним сервісом, який забезпечує зручність та безпеку для користувачів. Сервіс відповідає сучасним вимогам щодо захисту даних та надає користувачам можливість безпечно використовувати електронну пошту, що є ключовим фактором у виборі даного продукту.

ВИСНОВОК

Розробка та впровадження поштового клієнта з захистом від фішингових атак, використовуючи API VirusTotal, виявилася значним успіхом у сфері забезпечення безпеки електронної пошти. Проведене дослідження, а також детальний аналіз предметної області дозволили створити інноваційний продукт, який поєднує в собі високу функціональність, зручність користування та надійний захист даних користувачів.

Теоретичний аналіз предметної області охопив основні аспекти кібербезпеки, що стосуються електронної пошти, включаючи фішингові атаки та підробку повідомлень. Було визначено, що використання технологій SPF, DKIM та DMARC є критично важливим для забезпечення автентичності та цілісності електронних листів. Аналіз існуючих аналогів показав, що поточні рішення мають свої переваги, але також і недоліки, що було враховано при проектуванні власного сервісу.

Проектування програмного додатку включало розробку архітектури системи на основі моделі MVC, що забезпечило розподіл відповідальності між компонентами та спростило процес розробки, тестування та підтримки додатку. Було створено структуровану базу даних для зберігання інформації про користувачів, листи та результати перевірок. Вибір мови програмування Python та використання SQL для роботи з базами даних забезпечило гнучкість та ефективність розробки..

Програмна реалізація передбачала розробку модуля захисту, що включає шифрування даних та інтеграцію з VirusTotal API для перевірки електронних листів на наявність загроз. Використання бібліотеки Fernet для шифрування гарантує надійний захист даних користувачів. Інтеграція з VirusTotal API дозволяє автоматично перевіряти вкладення та посилання у листах, що значно підвищує рівень безпеки..

Результатом роботи є повністю функціональний поштовий клієнт **SCANMAIL**, який може ефективно конкурувати з існуючими аналогами завдяки своїй високій безпеці та зручності у використанні. Основною перевагою цього додатку є його здатність забезпечувати надійний захист користувачів від фішингових атак та

інших кіберзагроз, що робить його цінним інструментом у сучасному світі цифрових комунікацій.

Загалом, проведене дослідження та реалізація проекту підтвердили важливість інтеграції сучасних технологій захисту в поштові клієнти для підвищення рівня кібербезпеки та забезпечення захисту інтелектуальної власності користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. VirusTotal Documentation. URL: https://docs.virustotal.com/reference/overview (дата звернення: 22.04.2024).
2. Sender Policy Framework (SPF). Mimecast. URL:https://www.mimecast.com/content/sender-policy-framework/#:~:text=Sender%20Policy%20Framework%20(SPF)%20is,to%20a%20company%20or%20brand (дата звернення: 22.04.2024).
3. DKIM Guide. Postmark. URL: https://postmarkapp.com/guides/dkim (дата звернення: 22.04.2024).
4. DMARC Policy. SendPulse. URL: https://sendpulse.ua/ru/knowledge-base/email-service/general/dmarc-policy (дата звернення: 22.04.2024).
5. What is Phishing?. Webroot. URL: https://www.webroot.com/nz/en/resources/tips-articles/what-is-phishing (дата звернення: 22.04.2024).
6. MetaDefender Vault. OPSWAT. URL:https://softprom.com/sites/default/files/materials/OPSWAT_MetaDefender Vault_ru_edited.pdf (дата звернення: 22.04.2024).
7. JavaRush. Introduction to the MVC Pattern. URL: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller (дата звернення: 22.04.2024).
8. JSON. Introduction to JSON. URL: https://www.json.org/json-ru.html (дата звернення: 23.04.2024).
9. Miro. What is a UML Diagram?. URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 23.04.2024).
10. Ajax Systems. NIST Certification. URL: https://ajax.systems/ua/blog/nist-sertification/ (дата звернення: 23.04.2024).
11. Crypto Huckers. Огляд алгоритму шифрування SHA-256. URL: https://www.cryptohuckers.club/2018/09/oglyad-algorytmu-shyfruvannya-sha-256.html (дата звернення: 23.04.2024).

12. Spy Soft. Cryptography in Python. URL: https://spy-soft.net/cryptography-python/ (дата звернення: 23.04.2024).
13. Acode. Introduction to Python. URL: https://acode.com.ua/intro-python/ (дата звернення: 24.04.2024).
14. W3Schools UA. SQL Introduction. URL:https://w3schoolsua.github.io/sql/sql_intro.html#gsc.tab=0 (дата звернення: 24.04.2024).
15. GoIT. HTML and CSS: What, Who and Why. URL:https://goit.global/ua/articles/html-i-css-shcho-tse-komu-ta-dlia-choho-potribno/ (дата звернення: 24.04.2024).
16. Foxminded. Python Frameworks. URL: https://foxminded.ua/freimvorky-python/. (дата звернення: 24.04.2024).
17. Cyberattacks 2021: Phishing, Ransomware & Data Breach Statistics From the Last Year URL: https://spanning.com/blog/cyberattacks-2021- phishing-ransomware-data-breach-statistics (дата звернення: 24.04.2024).
18. Positive Technologies, Trends in phishing attacks on organizations in 2022–2023 URL: https://www.ptsecurity.com/ww-en/analytics/trends-in-phishing-attacks-on-organizations-in-2022-2023/ (дата звернення: 24.04.2024).
19. Rob Sobers, 134 Cybersecurity Statistics and Trends for 2021 URL: https://www.varonis.com/blog/cybersecurity-statistics (дата звернення: 24.04.2024).
20. Statista, U.S. number of phishing victims 2018-2023 URL: https://www.statista.com/statistics/1390362/phishing-victim-number-us/ (дата звернення: 24.04.2024).

ДОДАТКИ

ДОДАТОК А. ТЕХНІЧНЕ ЗАВДАННЯ

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор

_____ ЯРЕМЧУК Ю.Є

"22" березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

Розробка поштового клієнта із захистом від фішингу з використанням VirusTotal API

08-72.БДР.012.00.000.ТЗ

Керівник бакалаврської дипломної роботи
д.т.н., професор

_____ ЯРЕМЧУК Ю.Є.

"____" _____

Вінниця – 2024 р.

1. Найменування та область застосування

Програмний засіб для захисту електронної пошти від фішингових атак з використанням API VirusTotal. Область застосування: захист інформаційних ресурсів та конфіденційних даних користувачів у системах електронної пошти.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективного методу захисту електронної пошти від фішингових атак, забезпечення високого рівня безпеки та конфіденційності даних користувачів.

3.2 Призначення: розроблений програмний засіб виконує захист електронної пошти від фішингових атак, перевіряючи листи на наявність шкідливого ПЗ та фішингових посилань за допомогою API VirusTotal.

4. Джерела розробки

4.1. VirusTotal Documentation.

URL: https://docs.virustotal.com/reference/overview.

4.2 What is Phishing? Webroot. URL: https://www.webroot.com/nz/en/resources/tips-articles/what-is-phishing

4.3. DKIM Guide. Postmark. URL: https://postmarkapp.com/guides/dkim

4.4 Spy Soft. Cryptography in Python. URL: https://spy-soft.net/cryptography-python/.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес перевірки електронної пошти на наявність фішингових загроз.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції з високою точністю та надійністю.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	23.03.2024	25.03.2024
2.	Аналіз предметної області обраної теми	26.03.2024	30.03.2024
3.	Розробка алгоритму роботи	31.03.2024	2.04.2024
4.	Написання бакалаврської роботи на основі розробленої теми	3.04.2024	15.05.2024
5.	Передзахист бакалаврської дипломної роботи	24.05.2024	24.05.2024
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	25.05.2024	27.05.2024
7.	Захист бакалаврської дипломної роботи	13.06.2024	15.06.2024

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

– ПЗ до бакалаврської дипломної роботи;

- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____ Іванченко О.А.

ДОДАТОК Б

Лістинг app.py

```
from flask import Flask, request, redirect, url_for, render_template, send_from_directory, jsonify
from flask_login import LoginManager, login_user, logout_user, login_required
from google.oauth2 import service_account
from googleapiclient.discovery import build
from cryptography.fernet import Fernet
import sqlite3
import os

app = Flask(__name__)
app.config["SECRET_KEY"] = "secret_key_here"

# Static folder configuration
app.static_folder = 'static'

# Google login configuration
GOOGLE_CLIENT_ID = "105994916625578123031"
GOOGLE_CLIENT_SECRET = "your_client_secret_here"
GOOGLE_REDIRECT_URI = "http://localhost:5000/login"

# VirusTotal API configuration
VIRUSTOTAL_API_KEY = "your_api_key_here"

# Database configuration
DATABASE_FILE = "malicious_senders.db"

# Encryption configuration
ENCRYPTION_KEY = Fernet.generate_key()

# Create a Fernet object with the encryption key
fernet = Fernet(ENCRYPTION_KEY)

# Create a SQLite database connection
```

```
conn = sqlite3.connect(DATABASE_FILE)
cursor = conn.cursor()

# Create a table to store the emails of malicious link senders
cursor.execute("""
   CREATE TABLE IF NOT EXISTS malicious_senders (
      email TEXT PRIMARY KEY
   );
""")

# Commit the changes
conn.commit()

# Close the connection
conn.close()

# Google login authorization
login_manager = LoginManager()
login_manager.init_app(app)

@login_manager.user_loader
def load_user(user_id):
   # Load the user from the database
   conn = sqlite3.connect(DATABASE_FILE)
   cursor = conn.cursor()
   cursor.execute("SELECT * FROM users WHERE id =?", (user_id,))
   user = cursor.fetchone()
   conn.close()
   return user

@app.route("/")
def index():
   return render_template("index.html")

@app.route("/scan", methods=["POST"])
```

```
def scan():
    email = request.form["email"]
    password = request.form["password"]
    scan_interval = request.form["scan_interval"]
    user_id = email  # Simplified user identification
    login_user(load_user(user_id))
    return redirect(url_for("dashboard"))

@app.route("/authorize", methods=["POST"])
def authorize():
    url = request.form["url"]
    api_key = VIRUSTOTAL_API_KEY
    response = requests.post(
        f"https://www.virustotal.com/vtapi/v2/url/scan",
        headers={"Authorization": f"Bearer {api_key}"},
        data={"url": url},
    )
    if response.status_code == 200:
        # Store the URL in the database
        conn = sqlite3.connect(DATABASE_FILE)
        cursor = conn.cursor()
        cursor.execute("INSERT INTO malicious_senders (email) VALUES (?)", (url,))
        conn.commit()
        conn.close()
        return "Authorized successfully!"
    return "Error authorizing with VirusTotal API"

@app.route("/dashboard")
@login_required
def dashboard():
    return render_template("dashboard.html")

@app.route("/logout")
@login_required
def logout():
```

```
    logout_user()
    return redirect(url_for("index"))

if __name__ == "__main__":
    app.run(debug=True)

;
```

ДОДАТОК В

Лістинг файлу index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Email Scanner</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta charset="utf-8" />
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <div class="container">
    <header>
      <h1>Вас вітає SCANMAIL</h1>
    </header>
    <main>
      <form id="scan-form" action="/scan" method="POST">
        <div class="form-group">
          <label for="Email">Пошта:</label>
          <input type="email" id="email" name="email" required>
        </div>
        <div class="form-group">
          <label for="password">Пароль:</label>
          <input type="password" id="password" name="password" required>
        </div>
        <div class="form-group">
          <label for="scan-interval">Інтервал сканування:</label>
          <select id="scan-interval" name="scan_interval">
            <option value="once">Один раз</option>
            <option value="15">Кожні 15 звилин</option>
            <option value="30">Кожні 30 хвилин</option>
            <option value="60">Кожну годину</option>
          </select>
        </div>
```

```
            <button type="submit">Почати сканування</button>
        </form>
        <div id="progress-container" style="display: none;">
            <label for="progress-bar">Прогрес сканування:</label>
            <progress id="progress-bar" value="0" max="100"></progress>
            <p id="progress-text">0%</p>
        </div>
        <div id="results-container" style="display: none;">
            <h2>Шкідливі листи та посилання</h2>
            <ul id="results-list"></ul>
        </div>
      </main>
    </div>
    <script src="script.js"></script>
</body>
</html>>
```

ДОДАТОК Г

Лістинг style.css

```
body {
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Arial, sans-serif;
    background-color: #f5f5f7;
    color: #333;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    margin-top: 50px;
}

header {
    text-align: center;
    margin-bottom: 30px;
}

header h1 {
    font-size: 2.5em;
    color: #333;
}

form {
    display: flex;
    flex-direction: column;
```

```
}

.form-group {
    margin-bottom: 20px;
}

.form-group label {
    font-weight: bold;
    margin-bottom: 5px;
    display: block;
}

.form-group input,
.form-group select {
    width: 100%;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 5px;
    font-size: 1em;
}

button {
    padding: 15px;
    background-color: #0071e3;
    color: #fff;
    border: none;
    border-radius: 5px;
    font-size: 1em;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #005bb5;
}
```

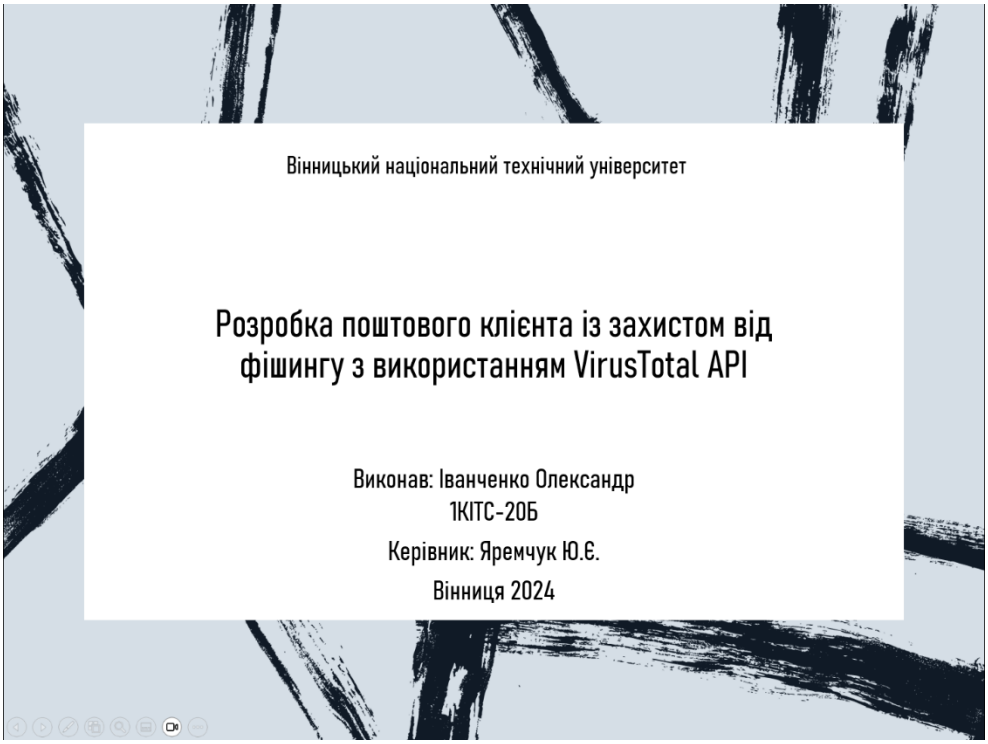
```
#progress-container {
    margin-top: 20px;
    text-align: center;
}

#progress-bar {
    width: 100%;
    height: 20px;
    margin-top: 10px;
}

#progress-text {
    margin-top: 5px;
}

#results-container {
    margin-top: 20px;
    text-align: left;
}
#results-container h2 {
    font-size: 1.5em;
    margin-bottom: 10px;
}
#results-list {
    list-style-type: none;
    padding: 0;
}
#results-list li {
    background-color: #f9f9f9;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 5px;
    margin-bottom: 10px;
}
```

ДОДАТОК Д. ІЛЮСТРАЦІЙНИЙ МАТЕРІАЛ



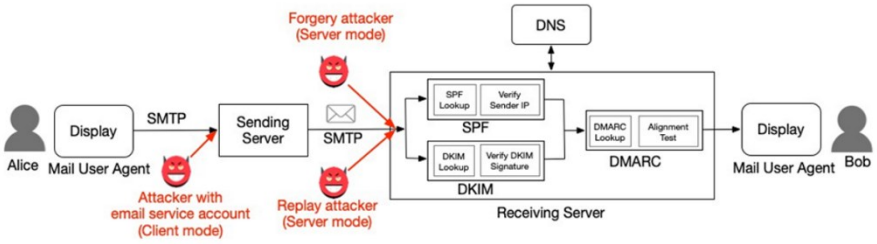
Клієнтський додаток

Сервер захисту

База даних

Мета та завдання дослідження

- Розробка захисного модуля
- Інтеграція з VirusTotal API
- Забезпечення конфіденційності та цілісності даних



Аналіз існуючих рішень

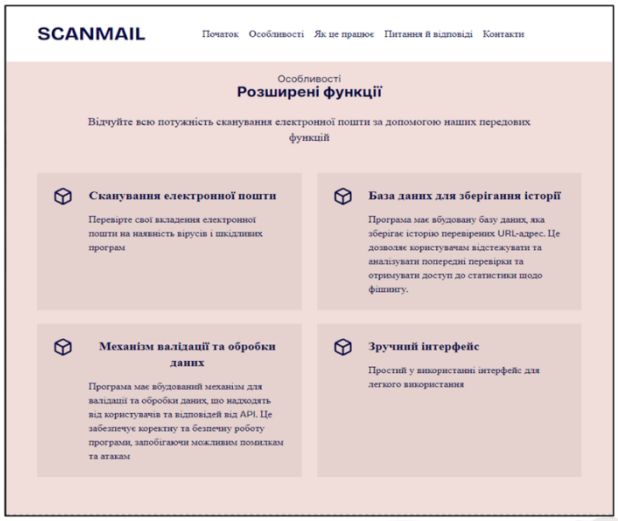
- Огляд поточних методів захисту
- Переваги та недоліки існуючих рішень
- Вибір оптимального підходу

Проектування архітектури системи

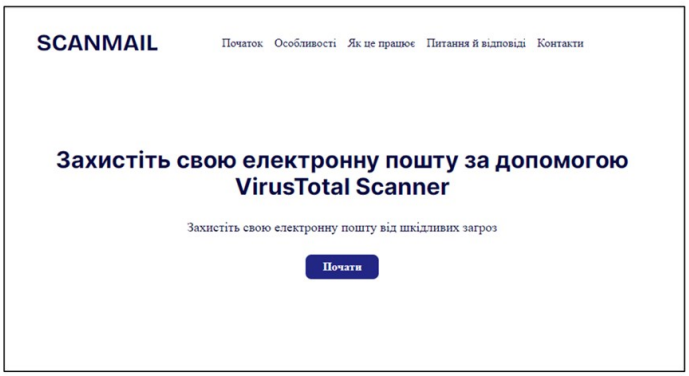
- MVC архітектура
- UML-діаграми
- Компоненти системи

Реалізація захисного модуля

- Шифрування даних з Fernet
- Інтеграція з VirusTotal API
- Захист від фішингових атак



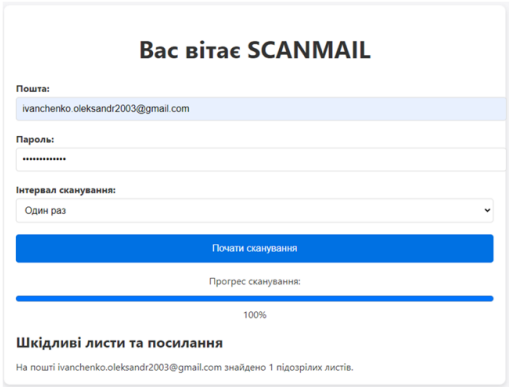
Інтерфейс користувача



- Зручність використання
- Форма авторизації
- Налаштування інтервалу сканування

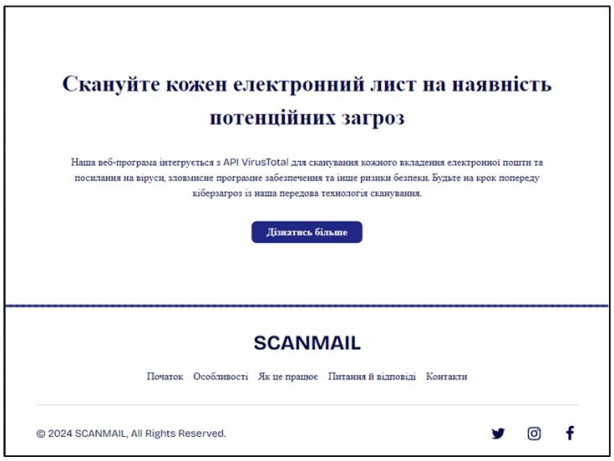
Результати тестування

- Ефективність виявлення загроз
- Швидкість роботи системи
- Зворотній зв'язок від користувачів



Переваги розробленого сервісу

- Високий рівень захисту
- Зручний інтерфейс
- Конкурентоспроможність



Висновки та майбутні перспективи

- Підсумки роботи
- Можливості для подальшого розвитку
- Важливість постійного вдосконалення захисту

ДОДАТОК Е. ПРОТОКОЛ ПЕРЕВІРКИ НА АНТИПЛАГІАТ

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка поштового клієнта із захистом від фішингу з використанням Virus Total Api

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

ПОКАЗНИКИ ЗВІТУ ПОДІБНОСТІ UNICHECK

Оригінальність 97 % Схожість 3 %

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Коваль Н.П. (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Іванченко О.А. (прізвище, ініціали)

Керівник роботи _____ (підпис) Яремчук Ю.Є. (прізвище, ініціали)