Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» травня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

освітня програма *Кібербезпека*
(назва освітньо-професійної програми)

на тему: «Метод виявлення фішингових адрес в доменній зоні .onion»

Виконавець: студентка ІІ курсу, групи КБм-22

_____ Аліна КИСЕЛЬОВА
(підпис) (ім'я прізвище)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Сергій БУЧИК	
Нормоконтроль	Сергій ДАКОВ	

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» жовтня 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*
(назва освітньої програми)

Здобувачки **КБм-22** (група) **Кисельової Аліни Павлівни** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи «Метод виявлення фішингових адрес в доменній зоні .onion»

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 4 від 24.10.2024 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес виявлення фішингових імітацій вебсайтів у Dark Web.
Предмет досліджень	Методи та засоби виявлення фішингових адрес в домені .onion.
Мета	Розробка та реалізація вдосконаленого комплексного методу, який поєднає в собі техніки детектування фішингових адрес в домені .onion
Вихідні дані для проведення роботи	Методи захисту від фішингових кампаній в мережі Dark Web.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна вдосконалення процесу виявлення фішингових сайтів у домені .onion шляхом інтеграції в розроблений метод багаторівневого аналізу технік маніпуляції користувачами

Практична цінність захист користувачів Tor від фішингових атак та автоматизований моніторинг підроблених сервісів у Dark Web.

4. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	25.10.2024 – 29.12.2024
Аналіз літературних джерел за темою дослідження	10.01.2024 – 12.02.2024
Вивчення нормативно-правової бази в галузі фішингу	13.02.2024 – 21.02.2024
Дослідження загроз та вразливостей до фішингових атак користувачів Dark Web	22.02.2024 – 26.02.2024
Аналіз технік, що використовуються зловмисниками для введення користувачів в оману шляхом імперсонейтингу	27.02.2024 – 04.03.2024
Порівняльна характеристика існуючих досліджень щодо виявлення фішингу в Dark Web	05.03.2024 – 10.03.2024
Планування реалізації практичної частини	11.03.2024 – 13.03.2024
Розробка власного комплексного методу виявлення фішингових посилань в домені .onion	14.03.2024 – 05.04.2024
Тестування розробленого методу	06.04.2024 – 10.04.2024
Оцінювання ефективності розробленого методу	11.04.2024 – 25.04.2024
Оформлення пояснювальної записки згідно методичних рекомендацій	26.04.2024 – 18.05.2025
Подача пакету документів на розгляд ЕК	19.05.2025

Завдання видав _____ Сергій БУЧИК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Аліна КИСЕЛЬОВА
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 25.10.2024 р.
Термін подання кваліфікаційної роботи до ЕК 19.05.2025 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод виявлення фішингових адрес в доменній зоні .onion» складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків, має 85 сторінок основного тексту, включає в себе зміст, вступ, чотири розділи роботи, висновки та список джерел. Робота містить 2 додатки із загальною кількістю сторінок 20. У пояснювальній записці кваліфікаційної роботи міститься 43 рисунки і 5 таблиць. Список використаних джерел містить 53 найменування і займає 6 сторінок.

Мета дослідження: розробка та реалізація вдосконаленого комплексного методу, який поєднає в собі техніки детектування фішингових адрес в домені .onion.

Об’єкт дослідження: процес виявлення фішингових імітацій вебсайтів у Dark Web.

Предмет дослідження: методи та засоби виявлення фішингових адрес в домені .onion.

Методи дослідження: аналіз, порівняння, проєктування, тестування, оцінювання ефективності.

Наукова новизна кваліфікаційної роботи полягає в інтеграції в розроблений метод багаторівневого аналізу для вдосконалення процесу виявлення фішингових сайтів у домені .onion. У роботі реалізовано комплексний підхід, використовуючи техніки та методи для порівняння HTTP-заголовків, HTML-коду, текстового вмісту, схожості зображень та URL-адрес. Представлена комбінація критеріїв дозволяє істотно покращити точність визначення фішингових вебсторінок в мережі Dark Web, оскільки охоплює ключові аспекти, що використовуються для імітацій автентичних ресурсів та маніпулювання користувачами.

Практична цінність: запропонований метод може бути використаний для захисту користувачів Tor від фішингових атак, а також для автоматизованого моніторингу підроблених сервісів у Dark Web.

Ключові слова: Dark Web, браузер Tor, фішинг, імперсонація, клонування, onion.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTTP – Hypertext Transfer Protocol

HTML – HyperText Markup Language

TOR – The Onion Router

URL – Uniform Resource Locator

SSL – Secure Sockets Layer

CSS – Cascading Style Sheets

PHash – Perceptual Hash

ORB – Oriented FAST and Rotated BRIEF

SSIM – Structural Similarity Index Measure

SHA-256 – Secure Hash Algorithm 256-bit

COAV – Compression-based Orthogonal Approximate Vectorization

BoW – Bag of Words

ЗМІСТ

ВСТУП

Dark Web (прихована мережа) – це частина мережі Інтернет, яка не індексується традиційними пошуковими системами і доступна лише за допомогою спеціального програмного забезпечення, такого як браузер Tor (The Onion Router), який надає підвищену анонімність користувачам за рахунок багатошарового шифрування та маршрутизації трафіку через розподілену мережу вузлів, на відміну від загальнодоступної мережі (Clear Web), де вебсайти є загальнодоступними та контролюються пошуковими системами та регуляторними органами. Тому Dark Web став центром для діяльності, орієнтованої на конфіденційність, а також для незаконних операцій, таких як чорні ринки, хакерські форуми та фішингові атаки.

Актуальність роботи пояснюється зростаючою складністю та поширеністю фішингових атак у мережі Tor, особливо націлених на домени .onion. На відміну від традиційного фішингу в загальнодоступній мережі, фішинг у Dark Web використовує унікальні механізми довіри та функції анонімності Tor, що значно ускладнює виявлення та запобігання. Це дослідження сприяє виявленню та протидії фішинговим загрозам у прихованих службах.

Домен .onion, доступний виключно через мережу Tor, забезпечує безпечне та приватне середовище для користувачів, однак ця властивість Dark Web використовується зловмисниками для створення шахрайських вебсайтів, які імітують легітимні сервіси. На відміну від Clear Web, де передові методи виявлення фішингу постійно вдосконалюються, Dark Web представляє унікальні проблеми. Традиційні механізми виявлення фішингу, які покладаються на репутацію домену, сертифікати SSL і моніторинг пошукової системи, менш ефективні в екосистемі .onion. Це вимагає розробки альтернативних методів та інструментів, здатних ідентифікувати фішингові загрози на основі властивих характеристик прихованих служб.

Запропонований метод використовує техніки виявлення на основі тексту, аналіз схожості зображень, HTTP-заголовків, HTML-коду сайтів, а також порівняння на основі адрес для оцінки ймовірності спроб фішингу, реалізуючи комплексний аналіз

посилань на вебсайти в домені .onion. Включаючи косинусний аналіз подібності, перцептивне хешування та показники структурної подібності, система виявлення підвищує точність ідентифікації шахрайських доменів .onion.

Забезпечуючи ефективний механізм виявлення спроб фішингу, це дослідження та розроблений інструмент, в свою чергу, сприяє підвищенню безпеки та захисту користувачів від фішингових імітацій та шахрайства в Dark Web.

Підсумовуючи, це дослідження є актуальним у сучасному цифровому середовищі, сприяє зростанню обізнаності про розвиток загроз фішингу в прихованій мережі, покращуючи можливості виявлення та протидії шахрайським маніпуляціям в Dark Web.

Тому метою дослідження є розробка комплексного програмного рішення, яке поєднає в собі методи детектування фішингових адрес в домені .onion.

Об’єктом дослідження є процес виявлення фішингових імітацій вебсайтів у Dark Web.

Предметом дослідження є методи та засоби виявлення фішингових адрес в домені .onion.

Методи дослідження включають в себе аналіз, порівняння, проєктування, тестування.

Наукова новизна кваліфікаційної роботи полягає в інтеграції в розроблене програмне рішення багаторівневого аналізу для вдосконалення процесу виявлення фішингових сайтів у домені .onion. У даній роботі реалізовано комплексний підхід, використовуючи методи для порівняння HTTP заголовків, HTML коду, схожості зображень та URL-адрес. Представлена комбінація методів дозволяє істотно покращити точність визначення фішингових вебсторінок в мережі Dark Web, оскільки охоплює ключові аспекти, що використовуються для імітацій автентичних ресурсів та маніпулювання користувачами.

Практична цінність дослідження полягає в тому, що запропонований алгоритм може бути використаний для захисту користувачів Tor від фішингових атак, а також для автоматичного моніторингу підроблених сервісів у Darkweb.

РОЗДІЛ 1
ОСНОВНІ ПОНЯТТЯ ПРО ФІШИНГ У DARK WEB

1.1 Анонімність в мережі Dark Web

Визначення Dark Web

Dark Web – це прихований сегмент інтернету і недоступний через стандартні веббраузери та пошукові системи. Він функціонує в оверлейних мережах, таких як Tor, призначених для анонімізації активності та місцезнаходження користувачів. Це середовище сприяє як законній, так і незаконній діяльності, починаючи від комунікацій, орієнтованих на конфіденційність, і закінчуючи нелегальними маркетплейсами [1].

Інтернет складається з декількох рівнів (рис. 1.1):

Surface Web (Clear Web) – загальнодоступні вебсайти, проіндексовані пошуковими системами;

Deep Web – зміст, який не індексується пошуковими системами, включаючи приватні бази даних і внутрішні мережі;

Dark Web – підмножина Deep Web, навмисно прихована, для доступу до якої потрібне спеціалізоване програмне забезпечення [2].

Рисунок 1.1 – Рівні мережі Інтернет

Архітектура Dark Web забезпечує анонімність шляхом маршрутизації зв'язку через низку зашифрованих вузлів, що ускладнює відстеження дій користувачів. Ця анонімність приваблює різноманітних користувачів і цілі [3].

Визначення мережі Tor

Доступ до Dark Web здійснюється в першу чергу через мережу Tor. Tor анонімізує трафік користувачів, шифруючи дані і маршрутизуючи їх через кілька серверів, що працюють на волонтерських засадах, відомих як ретранслятори. Кожен ретранслятор розшифровує шар шифрування, щоб виявити наступний пункт призначення – процес, схожий на зняття шарів цибулі, звідси і назва «The Onion Router» (рис. 1.2) [4]. Цей метод гарантує, що жоден ретранслятор не знає ні походження, ні призначення даних, зберігаючи анонімність користувача.

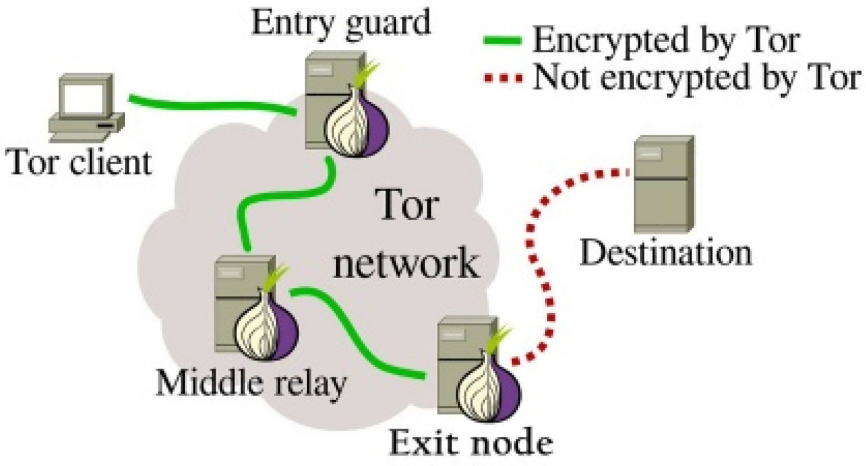

Рисунок 1.2 – Принцип роботи onion маршрутизації

Сайти в Dark Web використовують домен верхнього рівня .onion і доступні лише через браузер Tor. На цих сайтах, які називають «onion сервісами», може розміщуватися різний контент – від форумів і блогів до торгових майданчиків. Децентралізований і зашифрований характер Dark Web робить його стійким до цензури і зовнішнього моніторингу [5].

Tor є невід’ємною частиною функціонування Dark Web. Розроблений спочатку Військово-морською дослідницькою лабораторією США для захисту урядових

комунікацій, Tor перетворився на інструмент для звичайних користувачів, які прагнуть анонімності в Інтернеті. Неприбуткова організація Tor Project підтримує мережу та пов’язане з нею програмне забезпечення [4]. Використовуючи браузер Tor, люди можуть анонімно отримувати доступ як до Surface Web, так і до прихованих сервісів у Dark Web. Браузер розроблений таким чином, щоб запобігти відстеженню вебсайтами фізичного місцезнаходження користувачів та їхньої активності в Інтернеті. Крім того, Tor дозволяє користувачам публікувати вебсайти і сервіси, не розкриваючи місцезнаходження сервера, забезпечуючи приватність [1].

Застосування Dark Web

Dark Web слугує багатьом цілям, зокрема для забезпечення конфіденційності і свободи слова – люди в репресивних режимах використовують цю мережу, щоб обійти цензуру і безпечно спілкуватися. Журналісти та викривачі використовують його, щоб ділитися інформацією, не розкриваючи свою особистість. Наприклад, такі платформи, як SecureDrop, дозволяють викривачам анонімно передавати документи журналістам [3].

Але також, в Dark Web набула великого поширення незаконна діяльність, наприклад чорні ринки, на яких торгують незаконними товарами та послугами. Яскравим прикладом є вже неіснуючий ринок «Silk Road», який сприяв продажу наркотиків та інших нелегальних товарів. Кіберзлочинці також використовують приховану мережу для продажу викрадених даних, інструментів для злому та підроблених документів тощо [2].

Крім того, Dark Web є центром шахрайських схем, включаючи фішингові атаки, які мають на меті видати себе за легальні сервіси, щоб викрасти облікові дані користувачів Кіберзлочинці вдаються до імперсонейтингу, використовуючи анонімність сайтів .onion для створення клонів надійних ресурсів – ринків, банківських сервісів і криптовалютних бірж тощо [1]. Отримані внаслідок фішингу викрадені персональні (ім’я, дата народження, номер телефону, домашня адреса, дані паспорту, соціального страхування тощо), фінансові (дані кредитних карток, приватні ключі від криптогаманців тощо), медичні (дані про план лікування, рецепти, медичне страхування), корпоративні (комерційна таємниця, клієнтська інформація),

автентифікаційні (логіни, паролі, секретні ключі тощо) дані в подальшому можуть бути використані для продажу на хакерських маркетплейсах, створення фальшивої особистості (identity theft), корпоративного шпіонажу, вимагання грошового викупу (ransomware), фінансового та медичного шахрайства тощо (рис. 1.3).

Рисунок 1.3 – Наслідки фішингових атак

1.2 Технічні аспекти створення фішингових сайтів

Фішинг – це поширена кібератака, під час якої зловмисники маскуються під легітимні організації, щоб обманом змусити людей розкрити конфіденційну інформацію, таку як імена користувачів, паролі та фінансові дані тощо.

1.2.1 Техніки імперсонації у фішингових атаках

Зловмисники використовують різні стратегії для створення підроблених вебсайтів, які виглядають автентичними. Одним із поширених методів є тайпсквоттинг (typosquatting), який передбачає реєстрацію доменних імен, що містять

незначні помилки в написанні легітимних сайтів (рис. 1.4). Це використовує друкарські помилки користувачів, спрямовуючи їх на шкідливі сайти [6].

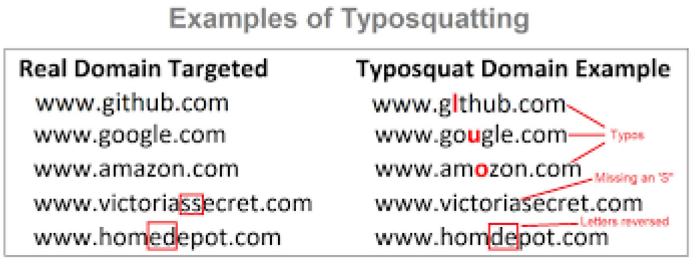

Рисунок 1.4 – Приклади тайпсквоттингу доменного імені

Інший підхід – омографічні атаки, коли символи з різних алфавітів використовуються для створення візуально ідентичних URL-адрес з метою обману користувачів [7]. Наприклад, зловмисники можуть замінити латинську літеру «о» на грецьку літеру «о», яка здається ідентичною для користувачів, які нічого не підозрюють.

Крім того, підміна субдоменів є ще однією широко використовуваною технікою (рис. 1.5). Зловмисники створюють субдомени, які містять назву законного бренду, вводячи користувачів в оману і змушуючи їх повірити, що вони знаходяться на офіційному сайті [8].

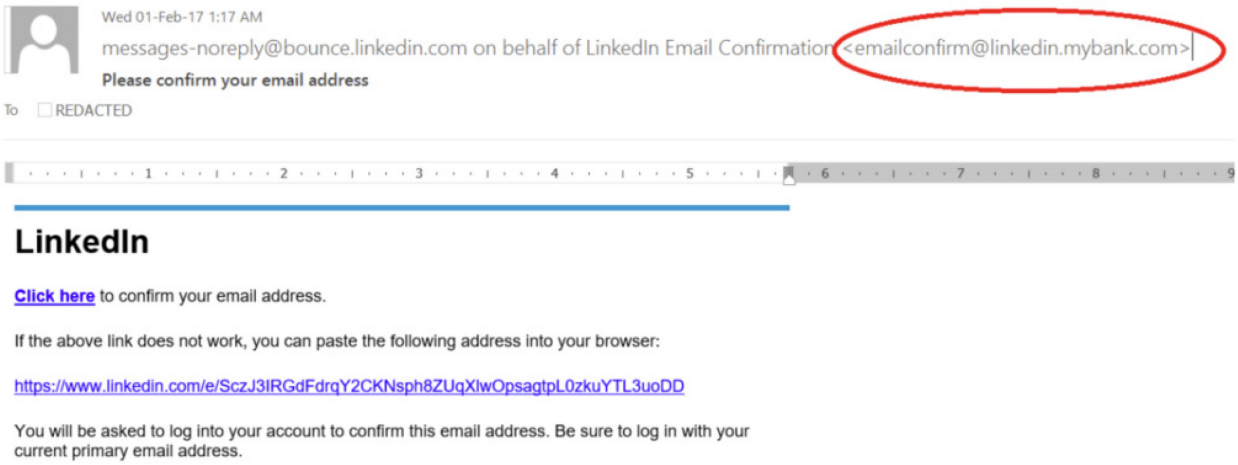

Рисунок 1.5 – Приклад підміни субдомену

1.2.2. Створення фішингових сайтів

Розробка фішингових сайтів складається з кількох технічних етапів. Перший крок – клонування сайту, коли зловмисники копіюють HTML, CSS та JavaScript легітимного сайту, щоб створити його копію, яку неможливо відрізнити від оригіналу. Потім цей клонований сайт розміщується на підробленому домені, щоб отримати облікові дані користувачів [6].

Складність клонування може бути різною – від простого копіювання зовнішнього вигляду сайту до складної реплікації функціональних можливостей як зовнішнього, так і внутрішнього інтерфейсу.

Просте клонування передбачає копіювання видимих компонентів вебсайту без копіювання його основних функцій. Зловмисники часто використовують інструменти для завантаження та відтворення HTML, CSS та зображень сайту, створюючи поверхневу копію [9]. Однак динамічні функції, які потребують обробки на стороні сервера, такі як аутентифікація при вході та запити до бази даних, зазвичай не функціонують у цих клонах.

Комплексне клонування сайту передбачає копіювання як фронтенду, так і певних функціональних можливостей бекенду для створення більш переконливого шахрайського сайту. Цей підхід вимагає просунутих технічних навичок і часто включає в себе складні методи атаки [9].

Методи, що використовуються при комплексному клонуванні:

Клонування на основі проксі-серверів – зловмисники встановлюють проксі-сервер, який перехоплює зв’язок між користувачем і легітимним вебсайтом, що дозволяє в режимі реального часу дублювати контент і маніпулювати ним [10]. Таким чином хакер реалізовує атаку «людина посередині» (MITM) – розташовуючись між користувачем і легітимним сайтом, зловмисники можуть перехоплювати і змінювати обмін даними в реальному часі, полегшуючи клонування динамічного контенту і взаємодій [11].

Впровадження шкідливих скриптів JavaScript у код клонованого сайту – техніка, що дозволяє зловмисникам перехоплювати вхідні дані користувачів і передавати їх на

свої сервери, фактично викрадаючи конфіденційну інформацію [9]. Фішингові сайти часто вбудовують такі скрипти для перехоплення даних користувача, таких як облікові дані для входу в систему, і передають їх на сервер зловмисника. У деяких випадках зловмисники використовують кейлоггери JavaScript або перехоплювачі полів для викрадення конфіденційної інформації [8]. Після того, як облікові дані зібрані, жертву можуть перенаправити на легальний сайт, щоб мінімізувати підозри.

Використання фішингових наборів, що надають шаблони та інструменти для створення оманливих вебсайтів, які точно імітують легальні, що спрощує процес клонування для зловмисників [10].

Ще однією поширеною фішинговою практикою є використання безкоштовних хостингових послуг, щоб зменшити витрати та уникнути виявлення. Фішери часто використовують безкоштовні хостингові платформи, які дозволяють швидко розгортати фішингові сайти, не вимагаючи реєстраційних даних, і таким чином зберігають анонімність [7]. Ці сервіси дозволяють кіберзлочинцям проводити кілька фішингових кампаній одночасно з мінімальними зусиллями.

Ще один шахрайський метод – зловживання SSL-сертифікатами. Фішингові домени можуть отримати SSL-сертифікати для відображення в браузерах значка замка, який часто асоціюється у користувачів з безпекою. Це підвищує довіру до шахрайського сайту, ускладнюючи користувачам можливість відрізнити його від легітимного [7].

1.2.3 Фішингові імітації onion сервісів

Фішингові атаки в мережі Tor особливо небезпечні через складну природу onion адрес і відсутність традиційних механізмів перевірки. Onion адреси – це довгі рядки символів (16 в старій або 56 в новій версії алгоритму генерації), отримані з криптографічних ключів, що ускладнює запам'ятовування або розпізнавання користувачами легітимних сервісів. Цим користуються кіберзлочинці, створюючи шахрайські сервіси, які імітують надійні платформи, обманом змушуючи

користувачів надавати облікові дані, криптовалютні транзакції або інші конфіденційні дані [12].

Зловмисники використовують різні методи для імітації сервісів, що ускладнюють користувачам розрізнення справжніх і підроблених вебсайтів.

Підробка адрес onion. Оскільки адреси onion генеруються як хешовані версії криптографічних ключів, вони не можуть бути прочитані людиною, як традиційні доменні імена. Зловмисники користуються цим, генеруючи схожі на вигляд onion адреси за допомогою генерації адрес методом грубої сили або за допомогою інструментів, які дозволяють створювати кілька onion адрес, поки не буде знайдено візуально схожу [13]. Ця методика схожа на тайпсквоттинг у відкритому Інтернеті, але її значно складніше виявити через рандомізовану природу адрес onion [14]

Клонування легітимних вебсайтів. Зловмисники копіюють весь інтерфейс легітимного сервісу, включаючи HTML, CSS та JavaScript, і розміщують його на фальшивій адресі. Це схоже на традиційні фішингові атаки в інтернеті, коли шахрайські вебсайти точно імітують справжні (просте клонування). У багатьох випадках користувачі не помічають різниці, оскільки шахрайські сайти часто не мають чіткого брендингу або впізнаваних доменних імен.

Під час клонування зловмисники можуть застосовувати згадані раніше атаки зі зворотним проксі-сервером (зворотний проксі пересилає вхідні дані жертви, наприклад, імена користувачів, паролі або дані криптовалютних гаманців, до легітимного сервісу, викрадаючи дані під час передачі) та впровадження шкідливих скриптів JavaScript у код клонованого сайту (кейлоггери JavaScript або перехоплювачі полів для викрадення конфіденційної інформації)[15].

Використання безкоштовного хостингу та прихованих сервісів. Зловмисники часто використовують послуги безкоштовного хостингу, доступні в мережі Tor, для швидкого розгортання шахрайських сайтів в домені onion. Оскільки приховані сервіси можуть бути створені з мінімальними ресурсами і без перевірки особи, зловмисники можуть часто створювати і видаляти фішингові сайти, що ускладнює боротьбу з ними [13].

Проблеми з виявленням фішингу в onion сервісах

Фішингові атаки в Clear Web часто можна виявити за допомогою служб перевірки доменів, SSL-сертифікатів та централізованого моніторингу. Однак, onion сервіси створюють кілька проблем, які ускладнюють виявлення фішингу.

Однією з проблем є відсутність перевірки доменних імен. На відміну від традиційних вебсайтів, onion домени не покладаються на довірені центри сертифікації (ЦС) для перевірки. Користувачі не можуть легко перевірити, чи належить onion адреса легальному сервісу [15].

Також ускладнює виявлення фішингу децентралізація та анонімність мережі, оскільки сервіси onion працюють на піринговій основі, не існує жодного керівного органу, який би перевіряв законні сервіси або вносив шахрайські в чорний список [14].

Складнощі додаються через часті зміни адрес onion, більшість сервісів регулярно змінюють свої адреси для підвищення безпеки, що ускладнює запам'ятовування та перевірку автентичності користувачами. Зловмисники користуються цим, реєструючи адреси-двійники щоразу, коли популярний сервіс змінює домен [13].

1.3 Нормативно-правова база в сфері кібершахрайства

Кібершахрайство, зокрема фішинг, є серйозною загрозою у сфері інформаційної безпеки, тому його регулювання здійснюється через національні та міжнародні нормативно-правові акти. Вони встановлюють кримінальну відповідальність за кіберзлочини, визначають заходи захисту персональних даних та передбачають механізми міжнародної співпраці для боротьби з кіберзлочинністю.

Європейський Союз має комплексний підхід до боротьби з кіберзлочинністю, який охоплює кримінальне законодавство, захист персональних даних та заходи з кібербезпеки.

Директива про мережеві та інформаційні системи (NIS2) є ключовим нормативним актом Європейського Союзу, який встановлює вимоги до забезпечення кібербезпеки у країнах-членах, включаючи заходи щодо запобігання фішинговим

атакам як одному із способів несанкціонованого доступу до інформаційних систем. Ця директива визначає обов'язкові стандарти для організацій критичної інфраструктури, зобов'язує впроваджувати заходи кіберзахисту, посилює відповідальність за порушення безпеки даних, а також передбачає механізми ефективного обміну інформацією між державами-членами ЄС для спільної боротьби з кіберзлочинністю, включаючи фішинг та інші види атак [16].

Загальний регламент про захист даних (GDPR) визначає обов'язки компаній щодо обробки та захисту персональних даних громадян ЄС. Якщо фішингова атака призводить до витоку особистої інформації, організація, яка не забезпечила належний рівень безпеки, може бути оштрафована відповідно до GDPR. Це стимулює компанії впроваджувати ефективні механізми захисту від кібератак, включаючи фішингові кампанії [17].

Агентство Європейського Союзу з кібербезпеки (ENISA) здійснює моніторинг кіберзагроз, аналізує нові фішингові атаки та розробляє рекомендації для державних органів та бізнесу щодо протидії кіберзлочинності.

У США боротьба з кібершахрайством здійснюється через кілька ключових федеральних законів. Закон про комп'ютерне шахрайство та зловживання (Computer Fraud and Abuse Act, CFAA) криміналізує несанкціонований доступ до комп'ютерних систем, що охоплює фішингові атаки, якщо вони використовуються для отримання персональних або фінансових даних [18].

Закон про ідентифікаційне шахрайство (Identity Theft and Assumption Deterrence Act) передбачає кримінальну відповідальність за використання викрадених особистих даних у шахрайських цілях. Фішинг вважається основним методом отримання таких даних, тому цей закон є одним із ключових у боротьбі з подібними атаками [19].

У США питання захисту прав споживачів у сфері кібершахрайства, регулюється діяльністю Федеральної торгової комісії (FTC) відповідно до положень Federal Trade Commission Act (1914). FTC здійснює нагляд за дотриманням норм щодо добросовісної електронної комерції, веде боротьбу зі споживчим шахрайством, а також активно розслідує випадки фішингових атак. Окрім цього, комісія займається інформаційною підтримкою громадян, публікуючи практичні рекомендації щодо

виявлення та запобігання фішинговим схемам, а також впроваджує штрафи для компаній, які не забезпечили належного рівня кіберзахисту або допустили витік персональних даних внаслідок фішингових атак [20, 21].

Акт про електронні комунікації (Electronic Communications Privacy Act, ECPA) регулює захист конфіденційної інформації в електронній комунікації, що включає захист від викрадення даних через фішингові атаки [22].

США також розвивають співпрацю між державним сектором та приватними компаніями у сфері кібербезпеки. Наприклад, програма CISA (Cybersecurity and Infrastructure Security Agency) здійснює моніторинг загроз та надає рекомендації щодо боротьби з фішингом для підприємств та державних установ.

В Україні боротьба з фішингом та іншими формами кібершахрайства регулюється кількома законодавчими актами. Кримінальний кодекс України передбачає відповідальність за кіберзлочини. Стаття 190 ККУ ("Шахрайство") може застосовуватися до випадків фішингу, якщо доведено факт обману з метою незаконного отримання майнової вигоди. Також стаття 361 ККУ ("Несанкціоноване втручання в роботу електронно-обчислювальних машин, автоматизованих систем, комп'ютерних мереж чи мереж електрозв'язку") передбачає покарання за хакерські атаки, які можуть включати фішингові атаки на IT-системи компаній та державних установ [23].

Закон України "Про основні засади забезпечення кібербезпеки України" встановлює правові та організаційні основи захисту кіберпростору. У ньому закріплені механізми державного контролю за кіберзагрозами [24].

Закон України "Про захист персональних даних" регулює обробку та захист персональної інформації громадян. Оскільки фішинг часто спрямований на викрадення особистих даних, цей закон є важливим елементом захисту від таких атак. Відповідно до нього, організації, що обробляють персональні дані, повинні забезпечувати їхній захист від несанкціонованого доступу, що включає й фішингові атаки [25].

Закон України "Про електронну комерцію" також регулює безпеку електронних транзакцій, передбачає відповідальність за шахрайські дії в інтернеті та встановлює вимоги до ідентифікації продавців у сфері онлайн-торгівлі [26].

1.4 Аналіз існуючих досліджень в сфері виявлення фішингу

Фішинг у Dark Web являє собою складну еволюцію традиційної тактики фішингу, яка використовує анонімність і нерегульовану природу прихованих сервісів. У цьому підрозділі розглядаються дослідження, які аналізують фішингову діяльність у Dark Web, зосереджуючись на методологіях, механізмах виявлення та унікальних викликах, що виникають у цьому середовищі.

1.4.1. Механізми виявлення фішингу

Виявлення фішингових вебсайтів у Dark Web створює унікальні виклики через притаманну їм анонімність і відсутність традиційних механізмів перевірки. Вентурі та інші автори [27] в своєму дослідженні запропонували класифікацію наборів фішингу для раннього виявлення постачальниками платформ. У своєму дослідженні вони проаналізували понад дві тисячі фішингових наборів, зосередившись на функціях обходу систем виявлення та обфускації коду, і продемонстрували потенціал керованих моделей машинного навчання для класифікації наборів, що застосовують нові методи обходу антифішингових механізмів браузерів, систем виявлення хостингів, CAPCHA, систем машинного навчання для виявлення фішингу.

Абуадбба та інші автори [28] дослідили обмеження сучасних систем виявлення фішингу у загальнодоступній мережі та запропонували Anti-SubtlePhish – модель, що включає горизонтальні простори ознак для підвищення точності виявлення. Їхні експерименти зі ста тисячами фішингових і нешкідливих сайтів показали точність, підкресливши необхідність вдосконалених механізмів виявлення для боротьби з витонченими фішинговими тактиками.

Розуміння стратегій, які використовують зловмисники для імітації URL-адрес у Clear Web, має вирішальне значення для розробки систем виявлення. Тарані та Араччілаге [29] дослідили вибір ознак фішингових URL-адрес, використовуючи методи машинного навчання Information Gain та Chi-Squared для вибору ознак. Їхні результати виявили десять методів, які фішери використовують для імітації URL-адрес, що дають уявлення про те, як кіберзлочинці створюють оманливі посилання.

1.4.2 Фішингові стратегії в Dark Web

Зловмисники в Dark Web використовують різні тактики для обману користувачів, часто імітуючи легальні сервіси для збору конфіденційної інформації. Барр-Сміт і Райт [30] провели глибокий аналіз зловмисної імітації та клонування ринків у даркнеті, висвітливши, як зловмисники копіюють популярні сервіси .onion для реалізації фішингових схем. У статті показано, як зловмисники копіюють популярні сервіси для обману користувачів, що нагадує традиційний тайпсквоттинг Clear Web. У їхньому дослідженні використовувався інструмент модульного скрепера для виявлення мереж зловмисно клонованих ринків у даркнеті, що свідчить про складність довірчих відносин у сервісах Tor. Крім того, автори звернули увагу на труднощі у виявленні фішингових сайтів через складні довірчі відносини в сервісах Tor. Їхній аналіз показав, що децентралізована та анонімна природа Темного Інтернету ускладнює виявлення та пом'якшення фішингових загроз.

Аналогічно, Юн та інші [31] провели комплексний аналіз фішингових прихованих веб-сервісів у Dark Web, виявивши значну кількість дублікатів вебсайтів, націлених на користувачів. Їхні висновки показали, що невелика кількість окремих груп вебсайтів була відтворена в численних доменах, що підкреслює поширеність фішингової діяльності в екосистемі Dark Web.

Також у дослідженні про захист користувачів onion від фішингу [32] авторами було проаналізовано поширеність фішингових сайтів серед сервісів Tor. Дослідники виявили, що зловмисники користуються складністю розрізнення автентичних доменних імен onion від фішингових сайтів, використовуючи складність адрес onion

для обману користувачів. У дослідженні було проаналізовано попередні дослідження, присвячені цій проблемі, і виявлено, що лише два відомі підходи – хеш-візуалізація та обмін ключами з парольною аутентифікацією (PAKE) – здатні вирішити цю проблему. Однак ці методи мають обмеження, наприклад, створюють сліди про відвідані сервіси, що може бути небажаним для особливо вразливих користувачів. Дослідники розробили розширення для браузеру, але допомагало генерувати візуальний хеш з посилань .onion, щоб покращити користувачам задачу розпізнавання автентичних посилань на сайти від їх фішингових копій.

Штайнебах та інші [33] дослідили виявлення фішингу на прихованих сервісах Tor, проаналізувавши методи, які зловмисники використовують, щоб видавати себе за легітимні сервіси. Вони розробили метрики для автоматичного виявлення спроб фішингу, зосередившись на заходах схожості контенту та аналізі заголовків HTTP. Їхні висновки підкреслили поширеність клонованих сайтів, призначених для обману користувачів у мережі Tor. Автори запропонували методологію, яка передбачає сканування сторінок onion та застосування методів порівняння на основі адрес, вбудованих зображень і текстового контенту.

Ще одним важливим дослідженням у сфері виявлення фішингових атак у Dark Web є робота Бергмана та Попова [34], у якій дослідники розглянули методи аналізу схожості контенту прихованих сервісів для ідентифікації фішингових ресурсів і дублікатів. Для аналізу текстового вмісту вебсторінок використовувався підхід на основі символів із застосуванням інструменту, який перетворює текст у числові вектори шляхом підрахунку частоти входження символів. Це дало змогу стандартизовано представити вміст сторінок та надалі порівнювати його за допомогою популярних метрик схожості, зокрема косинусної подібності, що визначає кут між векторами та показує рівень схожості текстів, також жаккарової подібності, яка оцінює відношення спільних символів до загальної кількості унікальних символів та евклідової відстані, що вимірює фактичну відстань між векторами у багатовимірному просторі. Також у дослідженні було реалізовано аналіз графічного контенту сторінок – для цього використовувався інструмент, який обчислює хеші зображень за алгоритмом хешування SHA-1. Це дозволяло визначати ідентичні або

схожі зображення, що могли бути скопійовані зі справжніх onion-сайтів на фішингові копії.

1.4.3. Фішингові набори та інструменти в Dark Web

Поширення фішингових наборів в Dark Web значно спрощує завдання для кіберзлочинців. Нещодавнє відкриття висвітлило фішинговий набір FishXProxy, який назвали одним з найпотужніших наборів інструментів для фішингу. Цей набір починається з унікально згенерованих посилань, які можуть уникнути початкової підозри, надаючи кіберзлочинцям складні інструменти [35].

Наявність таких наборів підкреслює необхідність постійного моніторингу та аналізу нових фішингових інструментів для розробки ефективних заходів протидії.

1.5 Порівняльна характеристика існуючих рішень виявлення фішингу в домені .onion

У сфері виявлення фішингових атак у домені .onion дослідники пропонують різні підходи, кожен із яких орієнтований на окремі аспекти виявлення копій onion-сервісів. Серед найвизначніших робіт у цій галузі можна виділити дослідження Барр-Сміта, Гюльдерінга та Штейнебаха, які застосовують різні стратегії боротьби з фішинговими сайтами.

У роботі Барр-Сміта і Райта [30] основна увага зосереджена на аналізі onion-доменів із використанням підходу тайпсквоттингу – пошуку доменів, які мають мінімальні відмінності у назві від популярних ресурсів і потенційно є фішинговими копіями. Дослідження включало порівняння цих сайтів для виявлення спроб фішингу, зокрема шляхом аналізу заголовків HTTP, які часто виявляли невідповідності між законними та фішинговими сайтами. Цей метод дозволив дослідникам кластеризувати сайти на основі спільної інформації заголовків, що вказує на те, що ними керує одна організація. Крім того, HTML-вміст порівнювався за допомогою бібліотеки для оцінки подібності, хоча це було менш ефективним проти більш

складних фішингових сайтів. Також було проведено ручний аналіз, щоб виявити відмінності між легітимними сайтами та їх клонами, підкресливши розбіжності в архітектурі сайту та методах генерації вмісту. Загалом методи були спрямовані на ефективну кількісну оцінку та характеристику фішингової діяльності в темній мережі.

Дослідження Гюльденрінга [32] пропонує підхід захисту безпосередньо для користувачів Tor, шляхом застосування системи візуалізації onion-адрес на основі хеш-функцій. Ідея полягає в тому, щоб допомогти користувачам розпізнавати автентичні адреси onion-сервісів за допомогою графічних позначень. Однак дослідники вказують на низку обмежень цього підходу. По-перше, необхідна додаткова оцінка, щоб визначити, чи дійсно користувачі можуть надійно розпізнавати такі відбитки. Також виникають труднощі при додаванні або видаленні доменів із розпізнаного набору, що може призводити до втрати інформації про попередні елементи. Як відомо, у Dark Web адреси вебсайтів часто можуть змінюватись з метою збереження анонімності. Іншою проблемою є те, що засіб розпізнавання не здатний визначити, коли один домен імітує інший, що потенційно можна вирішити через додавання хешу заголовка сторінки, хоча це створює ризики колізій. Окрім цього, автори відзначають проблему масштабованості: зі збільшенням кількості розпізнаних доменів зростає й довжина ключів, що ускладнює використання системи. Таким чином, цей підхід підвищує безпеку на рівні користувача, але його ефективність залишається обмеженою в умовах активного розвитку фішингових атак і технічних обмежень самої системи.

Найбільш комплексний підхід запропоновано в роботі Штейнебаха [33], де розроблено систему автоматизованого виявлення фішингових сайтів шляхом глибокого аналізу контенту onion-сервісів. У межах цього підходу здійснюється збір сторінок через запропонований інструмент для вебсканування, а далі проводиться порівняння за кількома параметрами. Для виявлення фішингу на основі тексту (HTML, CSS і JavaScript) використовуються методи порівняння вмісту вебсторінок шляхом обчислення криптографічних хешів або застосування алгоритму COAV, що оцінює подібність через рівень стиснення документів. Для виявлення фішингових сторінок на основі зображень використовуються хеш-функції, зокрема алгоритми

бібліотеки PHash, що описувався в роботі Клінгера [36] і стійкий хеш ForBild згаданий в більш ранній роботі Штейнебаха [37], які дозволяють виявляти схожі графічні елементи. Також виконується аналіз onion-адрес: враховується, що адреса формується з хешу відкритого ключа, тому схожі префікси можуть свідчити про навмисну імітацію. Такий підхід дозволяє виявляти не лише прямі копії, але й модифіковані версії фішингових сайтів. Цей підхід є найповнішим, оскільки охоплює всі рівні перевірки: від текстового аналізу до перевірки технічних параметрів сторінок, проте його недоліком є складність реалізації та висока потреба в обчислювальних ресурсах.

У табл. 1.1 наведено характеристику застосованих методів кожному з досліджень.

Таблиця 1.1

Застосовані методи в розглянутих дослідженнях Барр-Сміта, Гюльденрінга та Штайнебаха

Застосовані методи	**Дослідження Барр-Сміта**	**Дослідження Гюльденрінга**	**Дослідження Штайнебаха**
Порівняння на основі тексту	HTML-вміст порівнювався за допомогою бібліотеки html-similarity для оцінки подібності	Не застосовано	Порівняння HTML, CSS та Javascript коду: обчислення криптографічних хешів для виявлення точних копій, застосування алгоритму COAV, що оцінює подібність сайтів
Порівняння на основі зображень	Не застосовано	Не застосовано	Алгоритми бібліотеки PHash та стійкий хеш ForBild для виявлення

продовження таблиці 1.1

			схожості графічних елементів
Порівняння на основі URL-адреси	Ручний аналіз схожості доменних імен	Програмне рішення обчислює візуальний хеш відвідуваного домену, і візуалізує його у вигляді зображення	Порівняння загального префіксу сторінок
Порівняння на основі метаданих сайту	Виконувалось порівняння HTTP заголовків	Не застосовано	Не застосовано

Таким чином, кожен із підходів має свою сферу застосування: рішення Барр-Сміта ефективне для моніторингу підозрілих доменів, метод Гюльденрінга орієнтований на підвищення обізнаності користувачів щодо достовірності адрес, а система від Штайнебаха забезпечує найглибший аналіз, орієнтований на виявлення фішингу на основі контенту та структури сторінок.

Висновки до розділу 1

У першому розділі було проведено аналіз основних понять, пов’язаних з фішинговими атаками в Dark Web, специфічних особливостей мережі Tor, а також технічних аспектів створення фішингових вебсайтів у домені .onion. Особливу увагу було приділено вивченню методів імітації та клонування onion-сервісів, які становлять значну загрозу для користувачів прихованих сервісів через складність перевірки автентичності ресурсів та високий рівень анонімності.

У розділі також проаналізовано нормативно-правову базу боротьби з кібершахрайством в Україні, Європейському Союзі та Сполучених Штатах Америки. Було визначено, що хоча законодавство активно реагує на кіберзагрози, включаючи

фішингові атаки, все ще існує недостатнє регулювання, спеціально орієнтоване на середовище Dark Web. Ця прогалина створює можливості для зловмисників використовувати анонімність Tor для проведення фішингових кампаній з мінімальним ризиком переслідування.

Крім того, було проведено огляд актуальних наукових досліджень у сфері виявлення фішингу в прихованих сервісах. Ці дослідження охоплюють різні підходи до аналізу контенту вебсайтів та виявлення дублікатів і фішингових копій, включаючи використання методів машинного навчання, векторизації тексту та методів хешування зображень. Дослідження підтверджують, що фішинг у Dark Web має унікальні характеристики, які вимагають спеціалізованих рішень, адаптованих до специфіки цибулевих доменів і механізму довіри.

Таким чином, у цьому розділі створено теоретичне підґрунтя для подальшої розробки ефективних методів виявлення фішингових вебсайтів у домені .onion, що є основною метою цього дослідження. Результати аналізу демонструють необхідність комплексного підходу до виявлення фішингу в Dark Web з урахуванням технічних особливостей, правового контексту і тактики кіберзлочинців.

РОЗДІЛ 2
МЕТОД ВИЯВЛЕННЯ ФІШИНГОВИХ АДРЕС В ДОМЕННІЙ ЗОНІ .ONION

В даному розділі будуть представлені обрані техніки виявлення фішингу в домені .onion, базуючись на існуючих дослідженнях, розглянутих в попередньому розділі. Також буде описана план реалізації методу виявлення фішингу в домені .onion шляхом впровадження цих технік у програмний інструмент, його функціонал та алгоритм роботи програми.

2.1 Обґрунтування застосованих технік виявлення фішингу

Базуючись на огляді існуючих досліджень, було визначено список технік, які слугуватимуть для визначення фішингових посилань в домені .onion. Нижче наведений їх перелік та детальна характеристика.

1. Аналіз заголовків HTTP

HTTP-заголовки є важливою частиною вебкомунікації, оскільки вони передають метадані між клієнтом (браузером) і сервером під час HTTP-запитів і відповідей. Ці заголовки включають різні технічні параметри, такі як тип сервера, тип контенту, політики кешування, файли cookie та налаштування безпеки. У контексті виявлення фішингу в Dark Web аналіз HTTP-заголовків є ефективним методом виявлення імітацій сайтів з доменом .onion, оскільки фішингові клони часто копіюють тільки візуальні аспекти вебсайту, але конфігурація сервера може відрізнятись від оригінальнго ресурсу.

Метод аналізу HTTP-заголовків починається з надсилання запитів до порівнюваних вебсайтів через мережу Tor, яка необхідна для доступу до сервісів .onion. Після встановлення з’єднання інструмент отримує повний набір заголовків HTTP-відповідей з кожного сайту.

Після вилучення заголовків виконується порівняння спільних полів заголовків між двома сайтами. Для кожного поля, присутнього в обох відповідях, перевіряється,

чи значення ідентичні. На основі цього обчислюється показник схожості, який зазвичай виражається як частка співпадаючих полів заголовків у загальній кількості порівнюваних полів.

Подібні підходи були застосовані в таких дослідженнях Штейнебаха [33], де метадані та технічні індикатори допомогли виявити підробку.

2. Подібність HTML-коду.

Одним з найпоширеніших методів, який зловмисники використовують для створення фішингових сайтів, особливо в Dark Web, є клонування структур легальних сайтів. Ці клони часто повністю копіюють HTML-код цільового сервісу .onion, щоб ввести в оману користувачів, представляючи візуально ідентичну і функціонально схожу копію.

Метод схожості HTML-коду передбачає двоетапний аналіз:

- точне виявлення дублікатів за допомогою хешування – обчислення криптографічного хешу повного вихідного коду HTML в даному випадку виконуватиметься з використаттям алгоритму SHA-256. Криптографічна хеш-функція генерує унікальний дайджест фіксованої довжини з вхідних даних (в даному випадку з усього HTML-коду). Навіть найменша зміна вмісту (наприклад, додатковий пробіл або зміна одного символу) призводить до абсолютно іншого хеш-значення. Якщо дві сторінки мають однакові хеші SHA-256, це означає, що їхній HTML-код абсолютно однаковий, що підтверджує факт клонування. Однак фішингові оператори часто вносять невеликі зміни, щоб уникнути виявлення за допомогою точного хешування, наприклад, модифікують незначний текст, змінюють URL-адреси або вставляють невидимі елементи. Таким чином, хоча порівняння хешів ефективне для виявлення ідеальних копій, його недостатньо для виявлення модифікованих клонів, що призводить до переходу на наступний рівень аналізу.

– виявлення часткової схожості за допомогою алгоритмів пошуку найдовшої суміжної підпослідовності між текстами. В розроблюваному методі використовуватиметься алгоритм Ratcliff/Obershelp, який знаходить найдовшу суміжну підпослідовність між двома текстами, а потім рекурсивно порівнює решту секцій. Цей алгоритм реалізований у функції SequenceMatcher з бібліотеки difflib мови

Python. SequenceMatcher порівнює послідовності символів HTML-коду. Він виводить коефіцієнт схожості в діапазоні від 0 до 1, де 1.0 вказує на повністю ідентичні HTML-структури. Значення ближче до 0 вказують на дуже низьку схожість. SequenceMatcher особливо ефективний, коли зловмисники намагаються модифікувати частини HTML, щоб обійти перевірку на основі хешу, зберігаючи при цьому більшу частину оригінальної структури і макета. Цей метод допомагає виявити фішингові сторінки, які не є точними копіями, але мають значну структурну і текстову схожість з оригінальним сайтом.

Отже, порівняння хешів і схожість коду дозволяють виявити дублікати сайтів, як зазначено в роботах Гадієнта [9] і Вонга [10], які досліджували структурну та текстову подібність у фішингових атаках.

3. Подібність текстового контенту.

Фішингові сторінки в Dark Web часто дублюють текстовий контент легітимних сайтів. Заголовки, описи, списки товарів, спливаючі вікна та інші текстові елементи часто копіюються майже без змін, щоб зберегти ілюзію автентичності. Навіть якщо HTML-код або зображення трохи змінені, основний текст часто залишається незмінним, що робить аналіз текстового контенту ефективним методом виявлення фішингу. Цей метод аналізу фокусується лише на змістовному тексті, видимому користувачеві, ігноруючи код, скрипти та метадані.

Виявлення текстової схожості складається з двох основних етапів:

– векторизація тексту – процес перетворення тексту в різноманітні векторні контури та геометричні форми. Для виконання цієї задачі застосовуватиметься метод CountVectorizer з бібліотеки Python scikit-learn, що працює за принципом «мішка слів» (Bag of Words, BoW), де кожен текст представлений у вигляді вектора частот слів. CountVectorizer перетворює текстовий вміст на числові дані, які можна математично порівняти. Видимий текст кожного вебсайту перетворюється на набір слів або символів (залежно від конфігурації). Векторизатор створює матрицю підрахунку токенів, де кожне унікальне слово або символ в об'єднаному текстовому корпусі стає ознакою, а його значенням є кількість разів, коли він зустрічається в тексті. CountVectorizer добре працює для виявлення фішингу, оскільки навіть коли

зловмисники змінюють дрібні деталі або порядок елементів сторінки, загальне вживання і частота слів залишаються схожими, особливо на таких сторінках, як маркетплейси, екрани входу в систему і контактні форми;

– вимірювання схожості за допомогою алгоритму косинусної подібності. Після векторизації тексту інструмент обчислює косинусну подібність між двома текстовими векторами. Косинусна схожість вимірює косинус кута між двома векторами в багатовимірному просторі. Якщо два тексти ідентичні, їхні вектори спрямовані в одному напрямку, і косинус подібності дорівнює 1.0 (максимальна схожість). Якщо вони взагалі не мають спільних термінів, косинусна схожість наближається до 0 (відсутність схожості). Цей метод є особливо надійним для виявлення фішингу, оскільки він нормалізує порівняння, роблячи його менш чутливим до довжини тексту і більш орієнтованим на фактичний розподіл слів. Нижче наведена формула розрахунку косинусної подібності (2.1).

$$cos(\Theta) = \frac{A * B}{||A|||B||} \quad (2.1)$$

де A та B – вектори кількості слів у двох текстах.

Цей метод ґрунтується на роботі Бергмана і Попова [34], де дослідники представили методи для визначення схожості текстів за допомогою векторизованого текстового аналізу, використовуючи алгоритми косинусної подібності, Жаккарової подібності та евклідової відстані. Їхнє дослідження продемонструвало, що текстові ознаки, зокрема частота та розподіл слів, є надійними індикаторами фішингу.

Жаккарова подібність порівнює дві множини (наприклад, набори символів або слів) і визначає відношення перетину до об’єднання множин.

Евклідова відстань вимірює пряму геометричну відстань між двома точками (векторами) у багатовимірному просторі.

Однак, у задачі порівняння текстового вмісту onion-сайтів для виявлення фішингу найефективнішою є саме косинусна подібність, оскільки:

не залежить від довжини текстів – у Tor сторінки можуть бути різними за обсягом через вставки додаткових фраз, банерів чи змін, але важлива не довжина, а схожість у структурі контенту;

враховує частоти символів чи слів, що важливо для виявлення повторюваних патернів тексту в фішингових копіях;

показує високі результати на розріджених векторах, які є типовими для текстових даних після векторизації (оскільки більшість символів або слів трапляються рідко);

менш чутлива до незначних змін у тексті, які фішингові сайти можуть використовувати для обходу простих детекторів (на відміну від Жаккарової подібності, яка різко падає, якщо відмінності унікальних символів зростають).

4. Аналіз схожості зображень.

Зображення, такі як логотипи або банери, завантажуються з вебсайтів і порівнюються за допомогою декількох методів:

– перцептивне хешування (pHash) призначене для порівняння зображень на основі їх візуального сприйняття, а не точного двійкового представлення. На відміну від криптографічних алгоритмів хешування (таких як SHA-256), які створюють абсолютно різні хеші навіть при найменшій зміні зображення, перцептивне хешування генерує схожі хеші для зображень, які візуально виглядають однаково, навіть якщо вони були змінені через зміну розміру, незначні зміни кольору, артефакти стиснення або шум. Алгоритм pHash спочатку змінює розмір зображення до стандартного розміру і перетворює його у відтінки сірого, щоб усунути вплив кольорів. Потім застосовується дискретне косинусне перетворення (ДКП) для виявлення частотних патернів на зображенні. Вибираються найбільш значущі коефіцієнти ДКП і перетворюються на бінарний хеш, порівнюючи їх з медіанним значенням набору. Потім визначається схожість між зображеннями шляхом обчислення відстані Хеммінга між їхніми хешами. У контексті виявлення фішингу pHash є дуже ефективним у виявленні повторюваних елементів брендингу, таких як логотипи, навіть якщо в них були внесені незначні зміни, щоб уникнути прямого виявлення копіювання. Цей підхід відповідає методам, описаним у дослідженні

Штейнебаха [33], де для виявлення клонованих прихованих сервісів описується підхід, що базується на аналізі зображень Dark Web;

– індекс структурної подібності (SSIM) – це ще одна сучасна методика, що застосовується для виявлення візуальної подібності між зображеннями на структурному рівні. На відміну від базових методів порівняння пікселів, SSIM враховує, як людське око сприймає структурну інформацію в зображеннях, аналізуючи такі аспекти, як яскравість, контрастність і просторову структуру. Алгоритм SSIM ділить зображення на невеликі блоки і порівнює відповідні ділянки, щоб виміряти яскравість, дисперсію контрасту і схожість патернів. Цей метод виводить оцінку від -1 до 1, де 1 означає ідентичні зображення. SSIM особливо корисний для виявлення фішингових сайтів, які вносять незначні зміни в зображення, наприклад, регулюють яскравість або контрастність, зберігаючи при цьому основну структуру скопійованої графіки. Цей метод дозволяє виявляти зображення, які навмисно змінені, щоб обійти просте виявлення, але при цьому зберігають достатню візуальну схожість для обману користувачів. Використання SSIM було описано в дослідженні [38] для визначення схожості зображень.

– алгоритм ORB (Oriented FAST and Rotated BRIEF) забезпечує додатковий рівень захисту, зосереджуючись на порівнянні специфічних особливостей зображення. ORB – це метод зіставлення на основі особливостей, який визначає і порівнює характерні ключові точки на двох зображеннях, такі як кути, краї і текстури, які залишаються незмінними навіть під час таких перетворень, як обертання або масштабування. Метод починається з алгоритму FAST для виявлення ключових точок і призначення орієнтації, щоб зробити їх інваріантними до повороту. Потім він використовує дескриптори BRIEF для кодування локальних патернів навколо цих ключових точок у двійкові рядки. Щоб оцінити схожість, ORB обчислює відстань Хеммінга між дескрипторами з різних зображень і зіставляє їх з відповідними ключовими точками. Цей метод особливо ефективний, коли зловмисники вносять геометричні спотворення або ледь помітні зміни в скопійовані зображення, щоб уникнути виявлення. У виявленні фішингу в Dark Web ORB відіграє вирішальну роль у виявленні зображень, які вже не є ідентичними по пікселям, але мають ідентичні

структури та шаблони на рівні елементів. Такі підходи були висвітлені в дослідженні Гадієнта [9], де ідентифікація фішингових клонів спиралася на узгодженість візуальних елементів, незважаючи на зусилля із затушовування.

У поєднанні ці три методи – pHash, SSIM та ORB – забезпечують комплексну основу для виявлення фішингу на основі зображень у Dark Web. У той час як pHash фіксує загальну візуальну схожість, SSIM оцінює структурну схожість, що має відношення до людського сприйняття, а ORB забезпечує стійкість до геометричних перетворень. Разом вони дозволяють ідентифікувати скопійовані візуальні елементи в сервісах onion, навіть якщо модифікації навмисно вносяться в них, щоб обійти прості механізми виявлення. Їх інтеграція в системи виявлення фішингу вирішує одну з ключових проблем захисту користувачів від шахрайських прихованих сервісів, які покладаються на візуальну імітацію, щоб завоювати довіру користувачів.

5. Порівняння префіксів URL-адрес .onion

Оскільки адреси .onion генеруються як криптографічні хеші, а не як доменні імена, які читаються людиною, користувачі часто покладаються на розпізнавання знайомих префіксів при перевірці автентичності сервісу. Зловмисники користуються цим, застосовуючи метод грубого перебору адрес – метод, який генерує кілька адрес, доки не буде знайдено візуально схожу. Цей метод дозволяє створювати фішингові сайти, які виглядають майже ідентично легальним сервісам, що збільшує ймовірність обману користувачів, які нічого не підозрюють.

Щоб виявити цю форму атаки, будуть порівнюватись перші шість символів адрес .onion і підраховуватиметься, скільки з них збігаються. Буде визначений поріг збігу 35%, тобто якщо більше двох з шести символів ідентичні, тоді адреси позначаються як підозріло схожі. Цей поріг заснований на припущенні, що зловмисник, який намагається видати себе за службу, згенерує велику кількість адрес, щоб знайти одну зі збігом префікса, тим самим збільшуючи ймовірність введення в оману користувачів, які лише побіжно подивилися на домен.

Цей метод концептуально схожий на тайпсквоттинг у Clear Web, коли зловмисники реєструють доменні імена з незначними змінами (наприклад, «paypa1.com» замість «paypal.com»), щоб ввести в оману користувачів. Однак

виявлення спроб фішингу в мережі Tor є більш складним завданням через рандомізовану природу адрес .onion. На відміну від традиційних доменів, URL-адреси .onion не мають чіткої структури і важко запам'ятовуються, що робить обман на основі префіксів особливо ефективним.

Схожі підходи застосовувались в дослідженнях Барр-Сміта [30] та Штейнебаха [33], де дослідили, як фішингові кампанії в мережі Tor використовують методи імітації засновані, зокрема, на базі схожості префіксів посилань.

6. Комбінована оцінка ймовірності фішингу

Інструмент обчислює зважену оцінку ймовірності фішингу, об’єднуючи оцінки схожості з усіх попередніх методів з визначеними коефіцієнтами. Такий підхід відображає необхідність багатофакторного аналізу, що є критично важливим для виявлення складних фішингових атак. Як зазначено в дослідженні [39], ефективне розпізнавання фішингових сайтів вимагає одночасного аналізу текстових та візуальних характеристик, оскільки зловмисники часто змінюють лише один із цих аспектів, намагаючись уникнути автоматичного виявлення.

2.2 Використані бібліотеки Python

Оскільки обраною мовою програмування для написання програмного коду для інструменту виявлення фішингових адрес в домені .onion є Python, відповідно бібліотеки застосовувались для цієї мови. Програма виявлення фішингу використовує набір спеціалізованих бібліотек, які надають необхідні інструменти для мережевої взаємодії, обробки даних, аналізу зображень, порівняння текстів і машинного навчання. Нижче наведено детальний опис кожної бібліотеки.

Бібліотека requests – одна з найпопулярніших бібліотек Python для створення HTTP-запитів. Вона спрощує надсилання HTTP/1.1 запитів, таких як GET, POST, PUT і DELETE, обробляючи такі завдання, як автентифікація, сеанси, файли cookie, заголовки та завантаження файлів. Вона може бути використана для надсилання запитів до onion-сервісів через мережу Tor і отримання HTML-контенту, HTTP-

заголовків та зображень цільових веб-сайтів. Це спрощує процес взаємодії з вебресурсами, управління сеансами та надсилання запитів через проксі-сервери [40].

difflib – це стандартна бібліотека Python для порівняння послідовностей, таких як рядки або списки. Вона зазвичай використовується для таких завдань, як визначення відмінності між файлами або текстами, коефіцієнтів схожості та зіставлення послідовностей. Однією з її ключових особливостей є клас SequenceMatcher, який знаходить найдовшу суміжну підпослідовність між двома вхідними даними [41].

Бібліотека hashlib пропонує алгоритми для криптографічного хешування, такі як MD5, SHA-1, SHA-256 та інші. hashlib широко використовується для створення контрольних сум або цифрових відбитків даних, забезпечення цілісності даних, хешування паролів і криптографічних програм.

numpy – це основна наукова обчислювальна бібліотека на Python, яка забезпечує підтримку великих багатовимірних масивів і матриць разом із математичними функціями для ефективної роботи з ними. Може застосовуватись для обробки масивів при порівнянні зображень та обчислення матриць подібності, підтримуючи ефективні математичні операції [42].

imagehash – бібліотека для створення перцептивних хешів зображень. Ці хеші представляють візуальні характеристики зображення, що дозволяє швидко порівнювати зображення на основі схожості вмісту, навіть якщо зображення були змінені, обрізані або трохи змінені [43].

cv2, або OpenCV (Open Source Computer Vision Library) – це комплексна бібліотека візуального розпізнавання та машинного навчання з відкритим вихідним кодом. Вона використовується для обробки зображень, аналізу відео, виявлення об’єктів у реальному часі, зіставлення ознак, розпізнавання обличь тощо [44].

scikit-image – це бібліотека з відкритим кодом для мови програмування Python, яка призначена для обробки та аналізу зображень. Вона базується на бібліотеці NumPy і є частиною екосистеми scikit-learn, що робить її сумісною з іншими інструментами для машинного навчання та наукових обчислень. ssim – це функція з бібліотеки scikit-image, яка вимірює схожість між двома зображеннями. Вона порівнює структурну

інформацію, яскравість і контраст, забезпечуючи перцептивну метрику якості або відмінності зображень, що часто використовується в таких задачах, як оцінка стиснення зображень і виявлення дублікатів [45].

Python Imaging Library (PIL) – це одна з найстаріших та найпопулярніших бібліотек Python для роботи із зображеннями. Вона забезпечує прості та ефективні інструменти для відкривання, обробки та збереження графічних файлів різних форматів. PIL дозволяє працювати із зображеннями як з об’єктами, що дає можливість легко їх редагувати, змінювати або аналізувати [46].

io – це стандартна бібліотека Python, яка забезпечує інтерфейси для роботи з потоками введення та виведення (I/O). Вона дозволяє працювати з файлами, текстами та бінарними даними як із потоками, що значно спрощує обробку даних у пам’яті без збереження на диск. Бібліотека io об’єднує низькорівневі та високорівневі інструменти для роботи з потоками, включаючи файлові об’єкти, текстові буфери, бінарні буфери та обгортки для роботи з різними типами даних. BytesIO є частиною вбудованого модуля io в Python і забезпечує двійковий потік в пам’яті. Він діє як файл, відкритий у двійковому режимі, дозволяючи читати дані (наприклад, зображення або файли) з пам’яті або записувати в пам’ять, а не з диска, що корисно для тимчасової обробки даних [47].

BeautifulSoup – це бібліотека Python для синтаксичного аналізу та навігації по HTML та XML документам. Вона надає потужні методи для пошуку, модифікації та вилучення даних з веб-сторінок, що робить її широко використовуваною в задачах вебскрепінгу, вилучення даних та маніпулювання контентом [48].

urllib – це стандартна бібліотека Python для роботи з URL-адресами та мережевими запитами. Вона дозволяє взаємодіяти з вебресурсами, здійснювати HTTP-запити, обробляти URL, а також виконувати кодування та декодування даних для передачі через Інтернет. Модуль urllib.parse слугує для розбору та обробки URL. urljoin та urlparse – утиліти для роботи з URL-адресами. urlparse розкладає URL-адреси на складові (схема, домен, шлях тощо), тоді як urljoin конструює абсолютні URL-адреси з базових та відносних шляхів [49].

scikit-learn – це одна з найпопулярніших бібліотек Python для машинного навчання. Вона забезпечує широкий набір інструментів для роботи з даними, включаючи алгоритми класифікації, регресії, кластеризації, зменшення розмірності, підготовки даних та оцінювання моделей. Бібліотека побудована на основі NumPy, SciPy та matplotlib, що робить її зручною для інтеграції в наукові та інженерні проєкти. CountVectorizer – це інструмент для вилучення особливостей з scikit-learn, який перетворює текстові дані в числові вектори на основі підрахунку токенів. Він широко використовується в обробці природної мови (NLP) для підготовки тексту для моделей машинного навчання шляхом перетворення слів або символів у числові представлення ознак. cosine_similarity – метрика з scikit-learn, яка вимірює косинус кута між двома ненульовими векторами. Вона зазвичай використовується в текстовому аналізі, рекомендаційних системах і кластеризації, щоб оцінити, наскільки схожі дві точки даних (наприклад, документи або набори ознак), незалежно від їхньої довжини [50].

2.3 Метод детектування фішингових адрес в доменній зоні .onion

Розроблений метод виконує виявлення фішингових сайтів шляхом порівняння двох onion-сервісів за низкою критеріїв, що охоплюють технічні, текстові та візуальні характеристики.

Метод включає наступні етапи:

1. Збір вхідних даних для роботи програми.

– користувач вводить два onion-посилання для перевірки;

– відбувається підключається до мережі Tor через проксі-сервер (SOCKS5 на localhost:9050).

2. Отримання даних з сайтів

Для кожного з двох сайтів:

– отримуються HTTP-заголовки – набір метаданих, що описують відповіді серверів;

– завантажується HTML-код сторінки;

– отримуються зображення (обирається перше знайдене на сторінці зображення).

3. Порівняння сайтів за ключовими параметрами

Виконуються порівняння за п’ятьма групами ознак:

а) HTTP-заголовки:

– визначення спільних заголовків;

– обчислення частки однакових значень для цих заголовків.

б) HTML-код:

– обчислення схожості HTML-кодів за допомогою алгоритму SequenceMatcher;

– перевірка ідентичності хешів HTML (SHA-256).

в) Текстовий вміст:

– пошук текстового вмісту – заголовків та абзаців;

– векторизація текстів за допомогою CountVectorizer;

– обчислення косинусної подібності між текстами.

г) Зображення

Якщо на сторінках є зображення, застосовуються три алгоритми:

– SSIM (Structural Similarity Index) – порівняння структури зображень;

– PHash (Perceptual Hash) – порівняння візуальних хешів;

– ORB (Oriented FAST and Rotated BRIEF) – пошук та порівняння ключових точок.

ґ) Порівняння префіксів URL-адрес .onion

– інструмент витягує перші шість символів обох доменів .onion і підраховує, скільки з них збігаються;

– якщо принаймні 35% перших шести символів збігаються, адреси позначаються як підозріло схожі.

4. Узагальнення результатів

Обчислення підсумкової ймовірності фішингової імітації шляхом зваженого підсумовування показників за всіма категоріями.

5. Повернення результату

Як результат роботи методу, програма виводить у консоль значення ймовірності того, що другий сайт є фішинговою копією першого у відсотках.

На рис. 2.1 наведена структурно-логічна схема, що графічно демонструє етапи роботи методу.

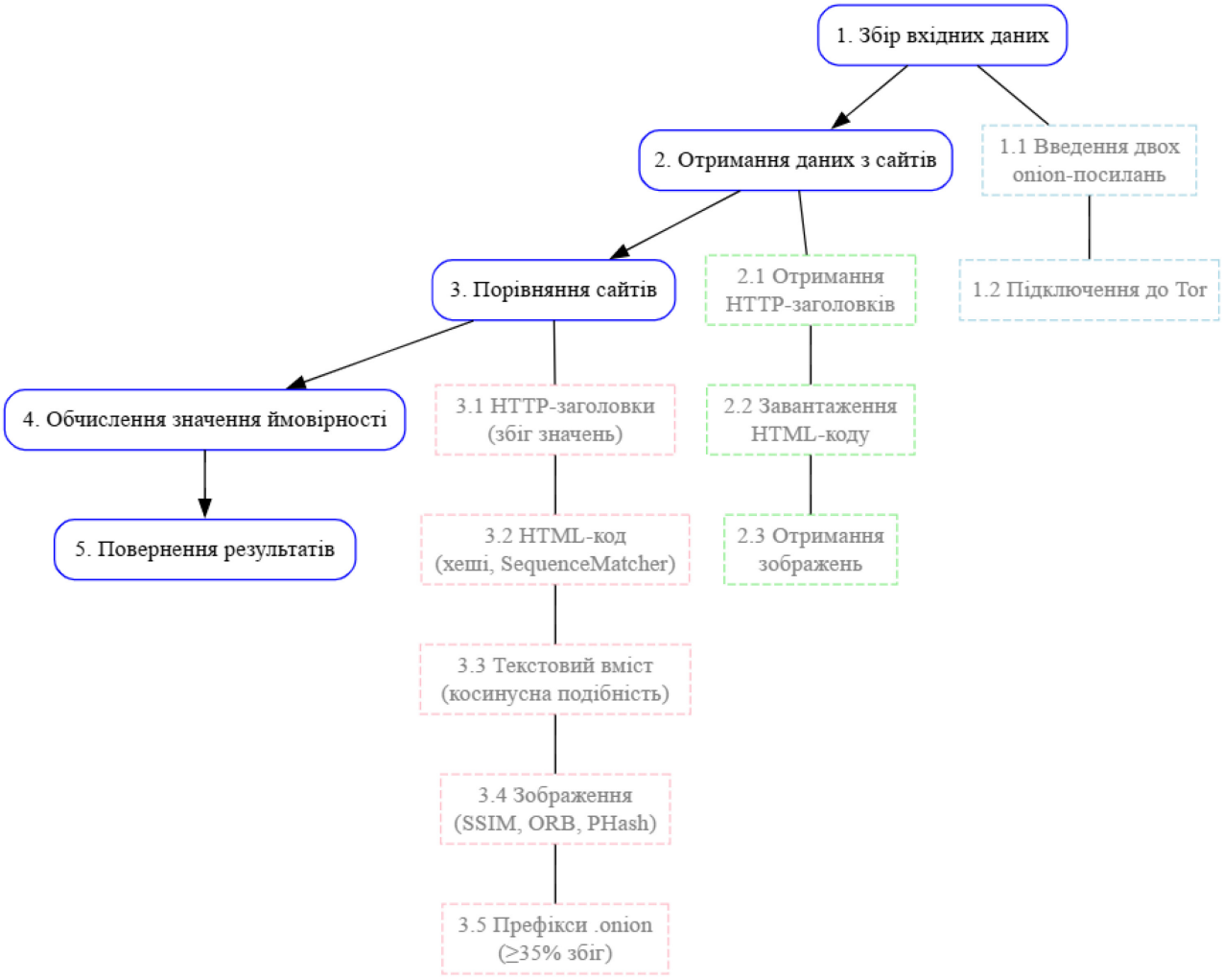

Рисунок 2.1 – Структурно-логічна схема роботи методу

2.4 Математична модель виявлення фішингових .onion-адрес

Математична модель буде представлена у формальному вигляді як аналітична модель з елементами імітаційного моделювання. Метою побудови математичної моделі є формалізація процесу оцінки ймовірності фішингової імітації прихованого сервісу в мережі Tor на основі множини ознак схожості між двома вебсайтами.

2.4.1 Формальна постановка задачі

Вхідні параметри – кожен сайт оцінюється за шістьма ознаками:

H – подібність HTTP-заголовків ($H \in [0,1]$)

T_{html} – схожість HTML-коду ($T_{html} \in [0,1]$)

T_{hash} – схожість текстових хешів ($T_{hash} \in \{0,1\}$)

T_{cos} – косинусна подібність тексту ($T_{cos} \in [0,1]$)

I – середня схожість зображень (SSIM, PHash, ORB) ($I \in [0,1]$)

I=(SSIM+PHash+ORB)/3

A – частка збігу перших символів доменів .onion ($A \in [0,1]$)

Кожен із цих параметрів впливає на кінцевий результат з певною вагою.

Вихідна величина P – оцінка ймовірності фішингової імітації сайту.

2.4.2. Побудова математичної моделі

1. Функція оцінки фішингової атаки.

Ймовірність фішингової атаки визначається за формулою 3.1 як лінійна комбінація окремих характеристик з відповідними ваговими коефіцієнтами:

$$P=w_H \cdot H+w_{html} \cdot T_{html}+w_{hash} \cdot T_{hash}+w_{cos} \cdot T_{cos}+w_I \cdot I+w_A \cdot A \quad (3.1)$$

де $w_H, w_{html}, w_{hash}, w_{cos}, w_I, w_A$ – вагові коефіцієнти, які визначають значимість кожного параметра;

$w_H+w_{html}+w_{hash}+w_{cos}+w_I+w_A = 1$ (нормалізовані ваги).

Приклад вагових коефіцієнтів (за емпіричним підходом):

$w_H=0.1, w_{html}=0.2, w_{hash}=0.2, w_{cos}=0.2, w_I=0.2, w_A=0.1$

2. Інтерпретація результату.

$P<0.5$ – сайт не є фішинговим.

$0.5 \leq P<0.75$ – потенційно підозрілий сайт.

$P \geq 0.75$ – висока ймовірність фішингу.

2.4.3. Функціональна схема математичної моделі

Функціональна схема як графічне представлення математичної моделі використовується для візуалізації структури процесів, що описуються математичною моделлю. Вона демонструє взаємозв'язки між вхідними даними, обробкою інформації, основними етапами аналізу та вихідними результатами.

Математична модель представлена у вигляді функціональної схеми на рис. 2.2.

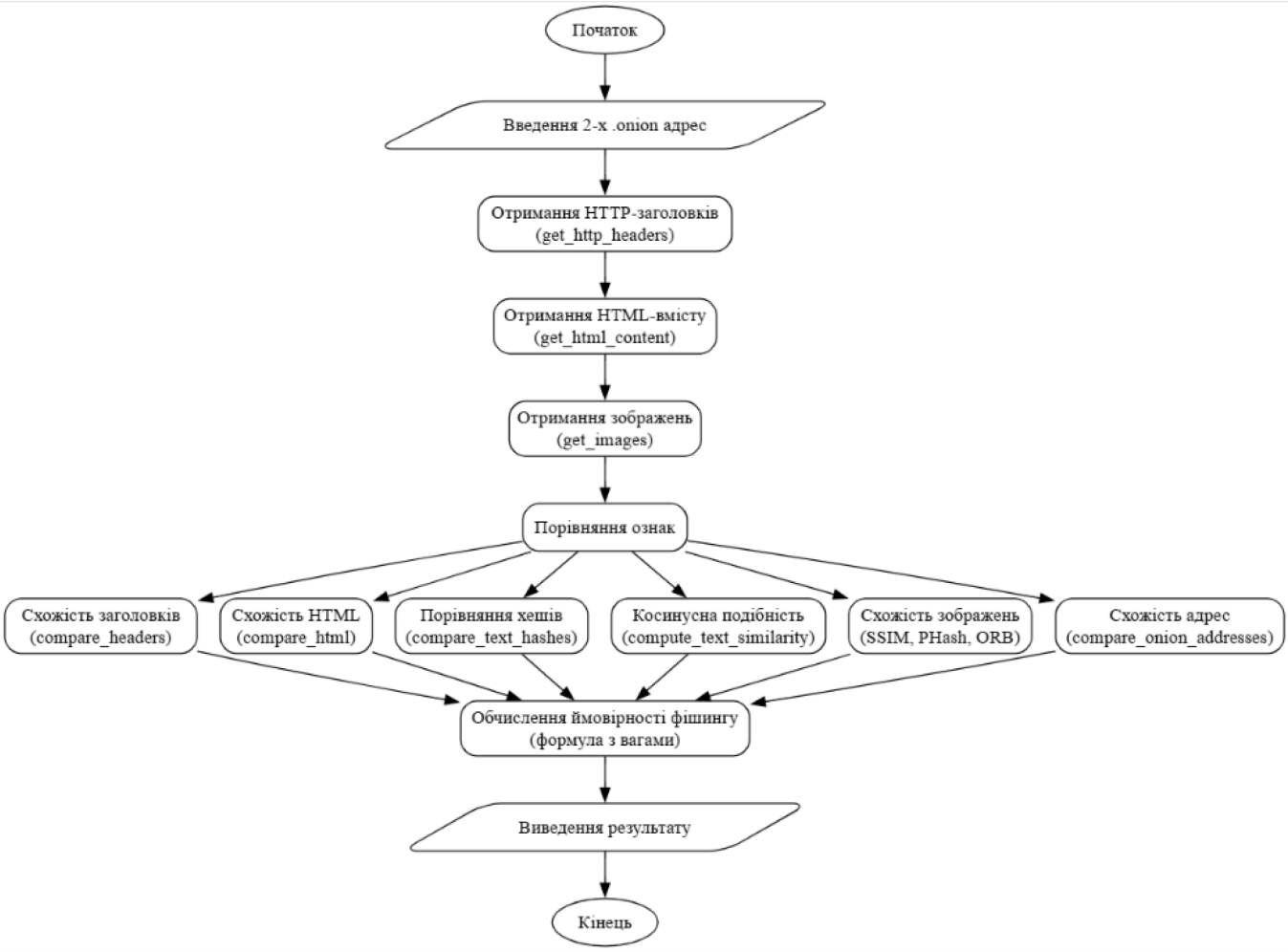

Рисунок 2.2 – Функціональна схема математичної моделі

Висновки до розділу 2

У цьому розділі було описано системний підхід до розробки інструменту виявлення фішингу для домену .onion. Дослідження та реалізація базувалися на попередніх дослідженнях та методологіях, проаналізованих у розділі 1. Було

окреслено методи виявлення, обрані для ідентифікації фішингових вебсайтів, та обґрунтувано їхню застосовність у середовищі Dark Web. Ці методи включають аналіз заголовків HTTP, схожість HTML-коду, порівняння текстового контенту та виявлення схожості зображень, кожен з яких відіграє важливу роль у розпізнаванні шкідливих веб-сайтів, що видають себе за легальні приховані сервіси.

Було досліджено аналіз заголовків HTTP як основний метод виявлення невідповідностей у конфігураціях серверів між фішинговими та легальними сайтами. Аналізуючи заголовки HTTP-відповідей, ми можемо виявити спроби імітації, коли зловмисники намагаються повторити технічну структуру справжнього onion сервісу. Підхід подібності HTML був реалізований з використанням хешування SHA-256 для точних дублікатів і SequenceMatcher для виявлення часткових модифікацій в структурах HTML, що є важливим у випадках, коли зловмисники вносять незначні зміни, щоб уникнути виявлення.

Крім того, в розділі представлено виявлення фішингу на основі тексту із застосуванням CountVectorizer для векторизації тексту та косинусної схожості для порівняння текстів. Цей підхід гарантує розпізнавання клонованого контенту, навіть якщо в нього внесено незначні зміни. Також було обговорено виявлення фішингу на основі зображень з використанням SSIM (Structural Similarity Index), перцептивного хешування (pHash) та ORB (Oriented FAST and Rotated BRIEF) для виявлення схожості у візуальних елементах, таких як логотипи та банери. Ці методи в сукупності підвищують здатність виявляти фішингові сайти, які імітують оригінальні елементи брендингу з метою обману користувачів.

У розділі також описано архітектуру інструменту, включаючи його функціональні компоненти, бібліотеки, використані для реалізації, та алгоритм, що керує його роботою. Робочий процес алгоритму включає отримання вхідних даних від користувача, вилучення та аналіз вебконтенту, порівняння різних ознак за допомогою вищезгаданих методів виявлення та обчислення остаточної оцінки ймовірності фішингу. Результати представлені в зрозумілому і доступному для інтерпретації форматі, що дозволяє користувачам оцінити ризик того, що певний сервіс на цибулі є фішинговим клоном.

Підсумовуючи, у розділі створено багаторівневу модель виявлення, яка інтегрує технічні, текстові та візуальні індикатори фішингу. Таке поєднання методів підвищує точність виявлення та стійкість до тактик ухилення, що використовуються зловмисниками в Dark Web. Розроблена методологія пропонує надійну основу для безпеки цибулевих сервісів з потенційним застосуванням в автоматизованому моніторингу фішингу, дослідженнях кібербезпеки та розвідці загроз в Dark Web.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ

В даному розділі будуть описані необхідні налаштування середовища для коректної роботи інструменту, наведений детальний опис функцій програмного коду, очікувані результати роботи розробленої програми, оцінка її ефективності у відповідності до очікуваних результатів. Також представлені шляхи застосування представленого методу, а також переваги та недоліки програмного рішення.

3.1 Налаштування середовища для роботи програми

Під час розробки програми використовувалась операційна система Debian GNU/Linux версії 12. Нижче буде наведене налаштування середовища для цієї ОС.

1. Оновлення пакетів та налаштування хосту.

По-перше, необхідно оновити список пакетів:

sudo apt update && sudo apt upgrade -y

По-друге, треба відредагувати файл /etc/hosts, додавши в нього ім’я хосту, щоб відбувалось підключення по мережі (рис. 3.1).

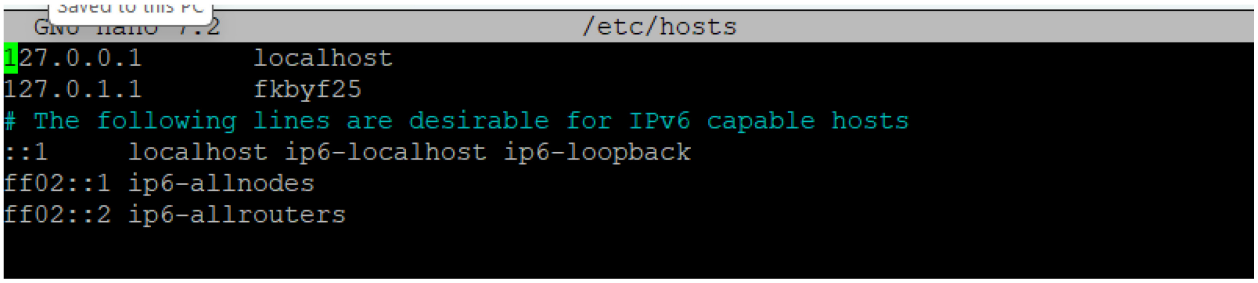

Рисунок 3.1 – Редагування файлу /etc/hosts

2. Встановлення Python та бібліотек для роботи з Tor

Встановлення мови програмування Python та необхідні утиліти для завантаження інших потрібних пакетів

sudo apt install python3 python3-pip python3-venv -y

Встановлення браузеру Tor

sudo apt install tor -y

Запуск Tor

sudo systemctl start tor

Перевірка роботи Tor (рис. 3.2)

systemctl status tor

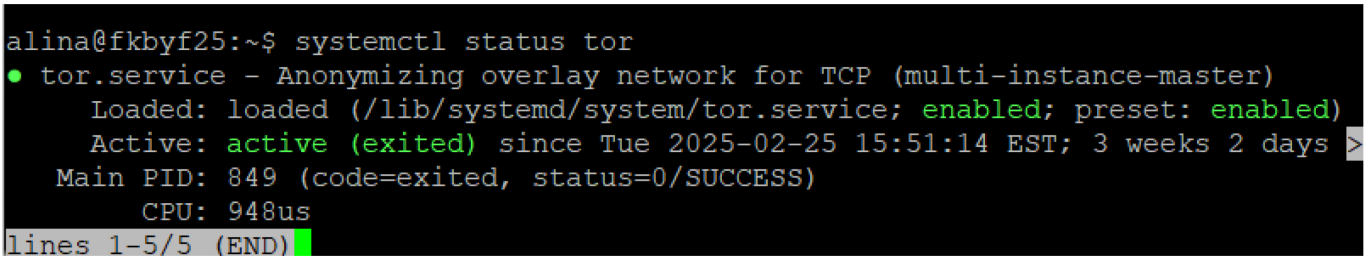

Рисунок 3.2 – Перевірка роботи Tor

Налаштувати автозапуск браузеру Tor

sudo systemctl enable tor

Перевірка роботи браузера через утиліту curl (рис. 3.3)

torsocks curl -I http://check.torproject.org

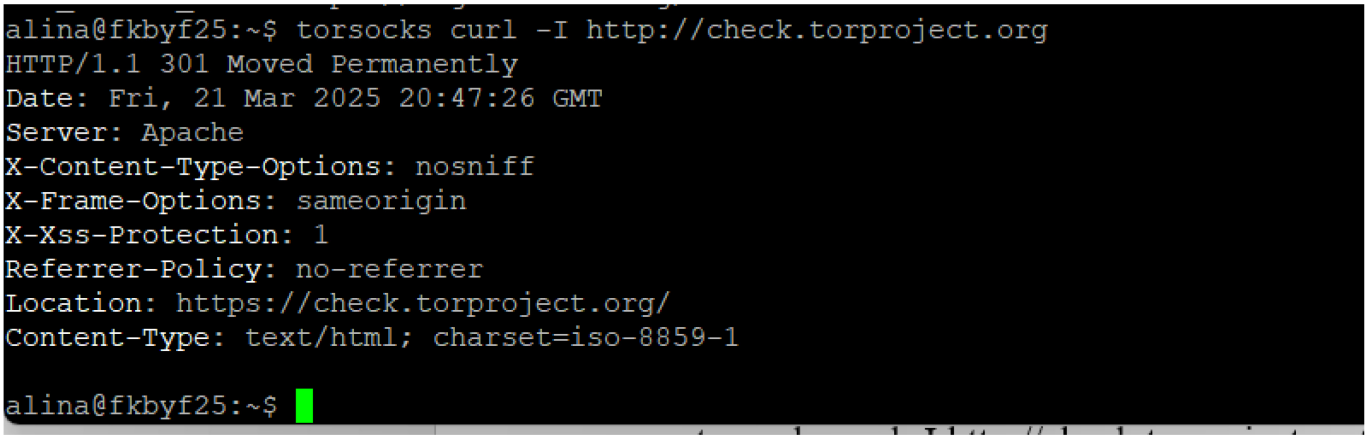

Рисунок 3.3 – Перевірка роботи Tor

Отже, браузер Tor налаштований для роботи.

3. Створення віртуального середовища.

Щоб коректно встановились всі пакети та для запобігання конфліктам із глобальними бібліотеками, створюємо віртуальне середовище та активуємо його.

python3 -m venv myenv

source myenv/bin/activate

4. Встановлення бібліотек Python для роботи застосунку

Наступним кроком буде встановлення необхідних Python-бібліотек (рис. 3.4)

pip install requests beautifulsoup4 opencv-python-headless scikit-image imagehash numpy scikit-learn pillow

```
(myenv) alina@fkbyf25:~$ pip3 list | grep -E "requests|beautifulsoup4|opencv-pyt
hon-headless|scikit-image|imagehash|numpy|scikit-learn|pillow"
beautifulsoup4          4.13.3
numpy                   2.2.3
opencv-python-headless 4.11.0.86
pillow                  11.1.0
requests                2.32.3
scikit-image            0.25.2
scikit-learn            1.6.1
(myenv) alina@fkbyf25:~$
```

Рисунок 3.4 – Перевірка інсталювання бібліотек

У табл. 3.1 наведене призначення кожної з встановлених бібліотек.

Таблиця 3.1

Призначення встановлених бібліотек

Бібліотека	**Призначення**
requests	Отримання HTML-сторінок та зображень по HTTP
beautifulsoup4	Парсинг HTML-документів
opencv-python-headless	Обробка зображень, виявлення ключових точок (ORB)
scikit-image	Порівняння зображень за SSIM
imagehash	Генерація хешів зображень (pHash)
numpy	Робота з матрицями, масивами
scikit-learn	Векторизація тексту, косинусна подібність
pillow	Завантаження та обробка зображень

Отже, середовище для роботи застосунку успішно налаштоване.

3.2 Опис функцій програмного коду

В підрозділі буде наведено детальний опис усіх функцій програмного коду інструменту phishing_detector.py, розробленого в рамках кваліфікаційної роботи, що є програмною реалізацією запропонованого методу виявлення фішингових адрес у доменній зоні .onion.

Функція *get_http_headers(url)* отримує заголовки HTTP із заданої URL-адреси, використовуючи мережу Tor через проксі-сервер SOCKS5 (рис. 3.5).

```
def get_http_headers(url):
    proxies = {
        "http": "socks5h://127.0.0.1:9050",
        "https": "socks5h://127.0.0.1:9050"
    }
    try:
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()
        return dict(response.headers)
    except requests.RequestException as e:
        print(f"Error fetching headers from {url}: {e}")
        return None
```

Рисунок 3.5 – Код функції get_http_headers(url)

Першим кроком налаштовується HTTP- і HTTPS-трафік для маршрутизації через локальний проксі-сервер Tor (socks5h:// означає, що дозвіл імен хостів також маршрутизується через Tor). Tor працює локально на порту 9050.

proxies = {

"http": "socks5h://127.0.0.1:9050",

"https": "socks5h://127.0.0.1:9050"

}

Далі виконується GET-запит до цільової адреси через Tor і встановлюється 30-секундний таймаут, щоб уникнути зависання програми.

response = requests.get(url, proxies=proxies, timeout=30)

Обробка помилок – згенерується HTTPError, якщо сервер повертає код стану ≥ 400 (наприклад, 404, 500). Це запобігає тихому завершенню роботи при невдалих відповідях.

response.raise_for_status()

Повертаються заголовки відповіді у вигляді словника Python.

return dict(response.headers)

Тут перехоплюється будь-яка помилка, пов'язана з мережею або запитом, і повертається значення None.

except requests.RequestException as e:

print(f"Error fetching headers from {url}: {e}")

return None

Функція *get_html_content(url)* призначена для отримання вихідного коду HTML веб-сторінки через мережу Tor, використовуючи проксі-сервер SOCKS5 (рис. 3.6).

```
def get_html_content(url):
    proxies = {
        "http": "socks5h://127.0.0.1:9050",
        "https": "socks5h://127.0.0.1:9050"
    }
    try:
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()
        return response.text
    except requests.RequestException as e:
        print(f"Error fetching HTML from {url}: {e}")
        return None
```

Рисунок 3.6 – Код функції get_html_content(url)

Налаштовується HTTP- і HTTPS-трафік для маршрутизації через локальний проксі-сервер Tor:

proxies = {

"http": "socks5h://127.0.0.1:9050",

"https": "socks5h://127.0.0.1:9050"

}

GET-запит надсилається на цільовий URL, використовуючи Tor як проксі-сервер. 30-секундний таймаут гарантує, що програма не зависне на невизначений час, якщо сайт повільний або не відповідає.

response = requests.get(url, proxies=proxies, timeout=30)

Викликає помилку, якщо код статусу HTTP вказує на помилку (наприклад, 404, 500).

response.raise_for_status()

Повертає необроблений HTML вміст сторінки у вигляді рядка.

return response.text

Перехоплює будь-який тип помилки, пов'язаної із запитом, записує повідомлення в журнал і безпечно повертає None.

except requests.RequestException as e:

print(f"Error fetching HTML from {url}: {e}")

return None 3.3 Інтерпретація результату роботи програми

Функція *hash_text(text)* приймає на вхід текстовий рядок і повертає SHA-256 хеш цього тексту у шістнадцятковому форматі (рис. 3.7).

```
def hash_text(text):
    return hashlib.sha256(text.encode()).hexdigest()

def compare_text_hashes(text1, text2):
    return hash_text(text1) == hash_text(text2)
```

Рисунок 3.7 – Код функцій hash_text(text) та compare_text_hashes(text1, text2)

Перетворюється вхідний рядок у байти, оскільки криптографічні хеш-функції працюють з байтовими даними. За замовчуванням використовує кодування UTF-8.

text.encode()

Застосовується алгоритм хешування SHA-256 з вбудованого модуля hashlib Python, криптографічна хеш-функція, яка створює 256-бітний (32-байтовий) хеш.

hashlib.sha256(...)

Вихідний двійковий хеш перетворюэться у читабельний шістнадцятковий рядок (довжиною 64 символи).

hexdigest()

Функція *compare_text_hashes(text1, text2)* використовується для визначення того, чи є два фрагменти тексту абсолютно однаковими, шляхом порівняння їхніх хешів SHA-256 (рис. 3.7).

Спочатку отримуються два рядкові аргументи: text1 та text2.

Викликається функція *hash_text()* для обчислення хешу SHA-256 кожного рядка. Цей крок перетворює вхідний текст в унікальний відбиток фіксованої довжини (64-символьний шістнадцятковий рядок).

Далі відбувається перевірка, чи ідентичні отримані хеші. Якщо тексти повністю збігаються, їхні хеші однакові.

Функція повертає булеве значення True, якщо тексти повністю ідентичні і False, якщо хоча б один символ відрізняється.

Функція *compare_html(html1, html2)* порівнює два HTML-документи і обчислює оцінку схожості між ними на основі їхнього необробленого (raw) текстового вмісту у вигляді рядків (рис. 3.8).

```
def compare_html(html1, html2):
    if not html1 or not html2:
        return 0
    return difflib.SequenceMatcher(None, html1, html2).ratio()
```

Рисунок 3.8 – Код функції compare_html(html1, html2)

Якщо не вдалося отримати HTML код у функції *get_html_content*, і як результат, принаймні один з вхідних даних є None або порожнім рядком, функція повертає 0.

if not html1 or not html2:

return 0

Використовується вбудований в Python інструмент difflib.SequenceMatcher. Вie обчислює коефіцієнт схожості між двома HTML-рядками. Інструмент базується на основі алгоритму найдовшої суміжної підпослідовності.

return difflib.SequenceMatcher(None, html1, html2).ratio()

Функція *compare_headers(headers1, headers2)* порівнює два набори HTTP-заголовків, щоб оцінити їхню схожість і виявити розбіжності, які можуть свідчити про спробу фішингу (рис. 3.9).

```
def compare_headers(headers1, headers2):
    if not headers1 or not headers2:
        return (0.0, True)

    mismatched_headers = []

    h1 = {k.lower(): v.strip() for k, v in headers1.items()}
    h2 = {k.lower(): v.strip() for k, v in headers2.items()}

    for key in h1:
        if key not in h2 or h1[key] != h2[key]:
            mismatched_headers.append(key)

    if mismatched_headers:
        print("⚠ Mismatched or missing headers detected:")
        for h in mismatched_headers:
            print(f" - {h}")

    common_keys = set(headers1.keys()) & set(headers2.keys())
    same_values = sum(1 for key in common_keys if headers1[key] == headers2[key])

    if common_keys:
        similarity = same_values / len(common_keys)
        return (similarity, similarity < 0.7)
    else:
        return (0.0, True)
```

Рисунок 3.9 – Код функції compare_headers(headers1, headers2)

Якщо хоча б один з наборів заголовків відсутній або немає жодного, повертається значення схожості 0.

Приведення заголовків до однакового формату для коректного порівняння:

h1 = {k.lower(): v.strip() for k, v in headers1.items()}

h2 = {k.lower(): v.strip() for k, v in headers2.items()}

Пошук різних заголовків – збираються всі ключі, які або не існують в обох заголовках, або мають різні значення:

for key in h1:

if key not in h2 or h1[key] != h2[key]:

mismatched_headers.append(key)

Якщо була знайдена невідповідність, ці ключі виводяться в консоль:

if mismatched_headers:

print(" Mismatched or missing headers detected:")

for h in mismatched_headers:

print(f" - {h}")

Обчислення відсотку спільних заголовків, які мають однакові значення:

common_keys = set(headers1.keys()) & set(headers2.keys())

same_values = sum(1 for key in common_keys if headers1[key] == headers2[key])

similarity = same_values / len(common_keys)

Функція повертає кортеж з двох значень – оцінка схожості (з плаваючою комою) та True, якщо менше 0.7 (тобто підозра на фішинг), інакше False

if common_keys:

return (similarity, similarity < 0.7)

else:

return (0.0, True)

Функція *extract_text_from_html(html)* приймає на вхід HTML-документ і витягує видимий текстовий вміст з вибраних тегів: <p>, <h1>, <h2> і <h3>. Вона повертає єдиний текстовий рядок, створений шляхом конкатенації цього вмісту (рис. 3.10).

```
def extract_text_from_html(html):
    soup = BeautifulSoup(html, "html.parser")
    text = ' '.join([p.get_text() for p in soup.find_all(['p', 'h1', 'h2', 'h3'])])
    return text
```

Рисунок 3.10 – Код функції extract_text_from_html(html)

Створюється об'єкт (soup) з вхідного HTML за допомогою вбудованого в Python html.parser:

soup = BeautifulSoup(html, "html.parser")

Знаходяться всі елементи в документі з вказаними тегами (<p>, <h1>, <h2>, <h3>). Для кожного з них викликається get_text(), щоб витягти вміст, придатний для читання людиною:

[p.get_text() for p in soup.find_all(['p', 'h1', 'h2', 'h3'])]

Об'єднання текстів в один рядок:

' '.join([...])

Всі витягнуті фрагменти тексту об'єднуються в один рядок, розділений пробілами.

І останнім кроком, функція повертає остаточний, очищений рядок з текстовим вмістом сторінки:

return text

Функція *compute_text_similarity(html1, html2)* обчислює косинусну подібність між двома вебсторінками, порівнюючи їх видимий текстовий вміст, витягнутий з HTML. Косинусна подібність – це значення між 0.0 і 1.0, де 1.0 означає, що тексти ідентичні, 0.0 означає відсутність схожості (рис. 3.11).

```
def compute_text_similarity(html1, html2):
    text1 = extract_text_from_html(html1)
    text2 = extract_text_from_html(html2)

    if not text1.strip() or not text2.strip():
        print("⚠ One or both pages have no usable text content.")
        return 0.0

    try:
        vectorizer = CountVectorizer().fit([text1, text2])
        vectors = vectorizer.transform([text1, text2])
        cos_sim = cosine_similarity(vectors[0], vectors[1])[0][0]
        return cos_sim
    except ValueError as e:
        print(f"⚠ Vectorization failed: {e}")
        return 0.0
```

Рисунок 3.11 – Код функції compute_text_similarity(html1, html2)

Першим кроком виконується видобування тексту. Використовує раніше визначену функцію *extract_text_from_html(html)* для витягування видимого тексту з тегів HTML, таких як <p>, <h1> і т.д.

text1 = extract_text_from_html(html1)

text2 = extract_text_from_html(html2)

Перевірка вмісту: якщо одна зі сторінок порожня (після видалення пробілів), повертає 0.0 і виводить попередження:

if not text1.strip() or not text2.strip(): ...

Векторизація тексту виконується за допомогою інструменту CountVectorizer з бібліотеки Python scikit-learn перетворює текстовий вміст на числові дані, які можна математично порівняти. Видимий текст кожного вебсайту перетворюється на набір слів або символів (залежно від конфігурації). Векторизатор створює матрицю підрахунку токенів, де кожне унікальне слово або символ в об'єднаному текстовому корпусі стає ознакою, а його значенням є кількість разів, коли він зустрічається в тексті. CountVectorizer добре працює для виявлення фішингу, оскільки навіть коли зловмисники змінюють дрібні деталі або порядок елементів сторінки, загальне вживання і частота слів залишаються схожими, особливо на таких сторінках, як маркетплейси, екрани входу в систему і контактні форми;згенерувати виключення на порожньому вході.

vectorizer = CountVectorizer().fit([text1, text2])

vectors = vectorizer.transform([text1, text2])

Обидва тексти перетворюються у числові вектори частоти слів, використовуючи модель «bag-of-words». Кожен вектор представляє кількість слів у відповідному тексті.

Розрахунок косинусної подібності відбувається за допомогою метрики cosine_similarity з бібліотеки scikit-learn, яка вимірює косинус кута між двома ненульовими векторами. Вона зазвичай використовується в текстовому аналізі, рекомендаційних системах і кластеризації, щоб оцінити, наскільки схожі дві точки даних (наприклад, документи або набори ознак), незалежно від їхньої довжини.

cos_sim = cosine_similarity(vectors[0], vectors[1])[0][0]

Вимірюється кут між двома текстовими векторами – чим менший кут, тим більш схожі тексти.

Обробка помилок спрацьовує у випадку, якщо CountVectorizer зазнає невдачі (наприклад, через видалення стоп-слова, що призвело до порожнього словника), помилка буде перехоплена і буде повернуто 0.0.

except ValueError as e: ...

Функція *get_images(url)* підключається до URL-адреси .onion через мережу Tor за допомогою проксі-сервера SOCKS5, розбирає HTML-сторінку, щоб знайти перший тег <img>, завантажує зображення та повертає об'єкт зображення PIL.Image (рис. 3.12).

```
def get_images(url):
    try:
        proxies = {
            "http": "socks5h://127.0.0.1:9050",
            "https": "socks5h://127.0.0.1:9050"
        }
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()

        content_type = response.headers.get("Content-Type", "")
        if not content_type.startswith("text/html"):
            print(f"Skipping non-HTML content from {url} (Content-Type: {content_type})")
            return None, None

        soup = BeautifulSoup(response.text, "html.parser")
        img_urls = [img["src"] for img in soup.find_all("img") if "src" in img.attrs]

        if not img_urls:
            print(f"No images found on {url}")
            return None, None

        img_url = urljoin(url, img_urls[0])
        print(f"Fetching image: {img_url}")

        img_response = requests.get(img_url, proxies=proxies, timeout=30)
        img_response.raise_for_status()

        img_content_type = img_response.headers.get("Content-Type", "")
        if not img_content_type.startswith("image/"):
            print(f"Skipping non-image content from {img_url} (Content-Type: {img_content_type})")
            return None, None

        image = Image.open(BytesIO(img_response.content))
        return image, imagehash.phash(image)
    except Exception as e:
        print(f"Error fetching images from {url}: {e}")
        return None, None
```

Рисунок 3.12 – Код функції get_images(url)

Першим кроком налаштовується HTTP- і HTTPS-трафік для маршрутизації через локальний проксі-сервер Tor (socks5h:// означає, що дозвіл імен хостів також маршрутизується через Tor). Tor працює локально на порту 9050.

proxies = {

"http": "socks5h://127.0.0.1:9050",

"https": "socks5h://127.0.0.1:9050"

}

GET-запит надсилається на цільовий URL, використовуючи Tor як проксі-сервер. 30-секундний таймаут гарантує, що програма не зависне на невизначений час, якщо сайт повільний або не відповідає.

response = requests.get(url, proxies=proxies, timeout=30)

Викликає помилку, якщо код статусу HTTP вказує на помилку (наприклад, 404, 500).

response.raise_for_status()

Виконується перевірка на наявність вмісту HTML. Якщо HTML відсутній, функція повертається раніше:

content_type = response.headers.get("Content-Type", "")

if not content_type.startswith("text/html"): ...

HTML розбирається за допомогою BeautifulSoup, щоб зібрати всі теги <img>, які мають дійсний атрибут src.

soup = BeautifulSoup(response.text, "html.parser")

img_urls = [img["src"] for img in soup.find_all("img") if "src" in img.attrs]

Якщо зображень не знайдено:

if not img_urls:

return None, None

Далі в коді виконується завантаження першого зображення. Для цього розпізнається повна URL-адреса зображення (обробляються відносні шляхи).

img_url = urljoin(url, img_urls[0])

img_response = requests.get(img_url, proxies=proxies, timeout=30)

Наступним кроком здійснюється перевірка зображень

if not img_content_type.startswith("image/"):

return None, None

Фінальний крок функції – це повернення зображення та значення його перцептивного хешу, яке буде застосовуватись далі для обчислення схожості.

image = Image.open(BytesIO(img_response.content))

return image, imagehash.phash(image)

Зображення відкривається з пам'яті та обчислюється його перцептивний хеш за допомогою алгоритму pHash. Цей хеш можна використовувати для виявлення візуально схожих зображень, навіть якщо вони дещо змінені.

Додатково реалізована обробка помилок – будь-яка помилка мережі або зображення перехоплюється. Функція виведе помилку і повертає None, None.

Функція *compare_images_phash(hash1, hash2)* порівнює два перцептивні хеші зображень (pHash) і повертає нормалізовану оцінку подібності між 0.0 і 1.0, де 1.0 → зображення ідентичні або майже ідентичні, 0.0 → зображення абсолютно різні. Він призначений для виявлення візуальної схожості між зображеннями, навіть якщо є незначні зміни, такі як масштабування, артефакти стиснення або зміна кольору (рис. 13).

```
def compare_images_ssim(img1, img2):
    size = (300, 300)
    img1_resized = img1.resize(size)
    img2_resized = img2.resize(size)

    img1_gray = cv2.cvtColor(np.array(img1_resized), cv2.COLOR_RGB2GRAY)
    img2_gray = cv2.cvtColor(np.array(img2_resized), cv2.COLOR_RGB2GRAY)

    similarity, _ = ssim(img1_gray, img2_gray, full=True)
    return similarity

def compare_images_phash(hash1, hash2):
    return 1 - (hash1 - hash2) / len(hash1.hash) ** 2 if hash1 and hash2 else 0

def compare_images_orb(img1, img2):
    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(np.array(img1), None)
    kp2, des2 = orb.detectAndCompute(np.array(img2), None)
    if des1 is None or des2 is None:
        return 0
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    return len(matches) / max(len(kp1), len(kp2))
```

Рисунок 3.13 – Код функції compare_images_phash(hash1, hash2)

Перевіряється, чи обидва хеш-значення не дорівнюють None. Якщо хоч одне з них відсутнє, повертає 0 (тобто немає схожості).

if hash1 and hash2:...

Бібліотека imagehash визначає оператор – для обчислення відстані Хеммінга між двома хеш-об'єктами (тобто кількості бітів, що відрізняються).

hash1 - hash2

Нормалізація показнику схожості:

*1 - (hash1 - hash2) / len(hash1.hash) ** 2*

len(hash1.hash) повертає довжину сторони матриці pHash (наприклад, 8x8).

*len(hash1.hash) ** 2* дає загальну кількість бітів у хеші (зазвичай 64).

Результат нормалізується до діапазону 0.0-1.0.

Функція *compare_images_ssim(img1, img2)* обчислює індекс структурної подібності (SSIM) між двома зображеннями – перцептивну метрику, яка кількісно оцінює візуальну подібність між зображеннями з точки зору яскравості, контрасту і структури. Результатом є значення з плаваючою комою між 1.0 – зображення ідентичні і 0.0 – зображення абсолютно різні (рис. 3.13).

Стандартизація обидвох зображень до фіксованого розміру для коректного порівняння. Використовується функція resize() з PIL :

size = (300, 300)

img1_resized = img1.resize(size)

img2_resized = img2.resize(size)

Конвертація у відтінки сірого з RGB за допомогою OpenCV. Необхідний крок, оскільки SSIM зазвичай працює з одноканальними зображеннями.

img1_gray = cv2.cvtColor(np.array(img1_resized), cv2.COLOR_RGB2GRAY)

img2_gray = cv2.cvtColor(np.array(img2_resized), cv2.COLOR_RGB2GRAY)

Обчислює SSIM за допомогою функції з бібліотеки skimage.metrics.structural_similarity:

similarity, _ = ssim(img1_gray, img2_gray, full=True)

return similarity

Повертається скалярне значення структурної подібності.

Функція *compare_images_orb(img1, img2)* використовує алгоритм ORB (Oriented FAST and Rotated BRIEF) для виявлення і порівняння локальних особливостей на двох

зображеннях. Вона обчислює оцінку схожості на основі кількості точок, що збігаються (рис. 3.13).

Ініціалізація ORB-детектора

orb = cv2.ORB_create()

Виявлення ключових точок та дескрипторів

kp1, des1 = orb.detectAndCompute(np.array(img1), None)

kp2, des2 = orb.detectAndCompute(np.array(img2), None)

kpX – ключові точки (такі як кути, ребра)

desX – дескриптори (числові вектори, що представляють локальні патерни)

Перевірка допустимих дескрипторів:

if des1 is None or des2 is None:

return 0

Якщо дескрипторів не знайдено (наприклад, порожні або однорідні зображення), повертається оцінка схожості зі значенням 0.

Зіставлення дескрипторів – використовується метод перебору з відстанню Хеммінга (оскільки ORB використовує двійкові дескриптори). crossCheck=True гарантує взаємну найкращу відповідність.

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

matches = bf.match(des1, des2)

Обчислення показнику схожості за алгоритмом ORB

return len(matches) / max(len(kp1), len(kp2))

Показник схожості – це відношення кількості ключових точок, що збігаються, до максимальної кількості ключових точок, виявлених на обох зображеннях.

Функція *compare_onion_addresses(addr1, addr2)* перевіряє, чи є дві адреси .onion візуально схожими, порівнюючи їх перші 6 символів (префікс), і повертає значення True, якщо принаймні 35% з них збігаються, інакше – False (рис. 3.14).

```
def compare_onion_addresses(addr1, addr2):
    min_length = min(len(addr1), len(addr2), 6)
    common_prefix = sum(1 for i in range(min_length) if addr1[i] == addr2[i])
    return common_prefix / 6 >= 0.35
```

Рисунок 3.14 – Код функції compare_onion_addresses(addr1, addr2)

Обчислюється довжина адрес. Забезпечується порівняння до 6 символів, або менше, якщо одна з адрес коротша. Обмежується область порівняння лише префіксом, який зазвичай використовується користувачами для візуального розпізнавання сервісу.

min_length = min(len(addr1), len(addr2), 6)

Підрахунок співпадінь символів у префіксі:

common_prefix = sum(1 for i in range(min_length) if addr1[i] == addr2[i])

Порівнюється кожен символ в однаковій позиції у префіксі обох адрес та лічильник збільшується для кожного збігу.

Обчислення частки збігів символів серед перших шести.

return common_prefix / 6 >= 0.35

Якщо 35% (або більше) збігів, повертається True (підозріла схожість).

Функція *phishing_probability(url1, url2)* отримує дві URL-адреси .onion і обчислює оцінку ймовірності фішингу між ними. Оцінка є зваженою комбінацією декількох метрик схожості (HTTP-заголовки, HTML та текстовий вміст, зображення, префікс адреси). Вищий бал (ближче до 1) вказує на більшу ймовірність того, що один сайт імітує інший з метою фішингу (рис. 3.15).

Отримання даних з URL-адрес: HTTP-заголовки, HTML-код і перше зображення з обох сайтів.

headers1, headers2 = get_http_headers(url1), get_http_headers(url2)

html1, html2 = get_html_content(url1), get_html_content(url2)

img1, img1_phash = get_images(url1)

img2, img2_phash = get_images(url2)

Достроково завершити роботу, якщо не вдалось отримати дані з URL-адрес

if headers1 is None or headers2 is None or html1 is None or html2 is None:

print("Error: One or both sites could not be accessed.")

return 0

Якщо жодна з URL-адрес не відповідає належним чином, повертається нульове значення ймовірності.

Обчислення всіх оцінок схожості

header_similarity = compare_headers(headers1, headers2)

html_similarity = compare_html(html1, html2)

text_hash_similarity = compare_text_hashes(html1, html2)

text_vector_similarity = compute_text_similarity(html1, html2)

address_similarity = compare_onion_addresses(...)

Порівняння зображення (якщо такі є):

ssim_sim = compare_images_ssim(img1, img2)

phash_sim = compare_images_phash(img1_phash, img2_phash)

orb_sim = compare_images_orb(img1, img2)

image_similarity = (ssim_sim + phash_sim + orb_sim) / 3

Поєднуються три методи порівняння зображень: SSIM за піксельною структурою, pHash за візуальним відбитком, ORB за характерними точками.

Наступним кроком визначається вага для кожного з порівнюваних компонентів

weight_headers = 0.1

....

weight_address = 0.1

Обчислення зваженої оцінки

```
probability = (
    header_similarity[1] * weight_headers +  # suspicious=True = 1
    html_similarity * weight_html +
    text_hash_similarity * weight_text_hash +  # True = 1, False = 0
    text_vector_similarity * weight_text_vector +
    image_similarity * weight_images +
    address_similarity * weight_address
)
```

Повернення результуючого значення оцінки ймовірності фішингової імітації – число з плаваючою точкою від 0.0 до 1.0, що представляє ймовірність фішингу.

return probability

```python
def phishing_probability(url1, url2):
    print("Getting HTTP Headers...")
    headers1, headers2 = get_http_headers(url1), get_http_headers(url2)
    print("Getting HTML Content...")
    html1, html2 = get_html_content(url1), get_html_content(url2)
    print("Getting images...")
    img1, img1_phash = get_images(url1)
    img2, img2_phash = get_images(url2)

    if headers1 is None or headers2 is None or html1 is None or html2 is None:
        print("Error: One or both sites could not be accessed.")
        return 0

    header_similarity = compare_headers(headers1, headers2)
    html_similarity = compare_html(html1, html2)
    text_hash_similarity = compare_text_hashes(html1, html2)
    text_vector_similarity = compute_text_similarity(html1, html2)
    address_similarity = compare_onion_addresses(onion_url1.split('//')[-1], onion_url2.split('//')[-1])
    print("HTTP Headers Similarity:", header_similarity[0], header_similarity[1])
    print("HTML Similarity:", html_similarity)
    print("HTML Hash Similarity:", text_hash_similarity)
    print("Cosinus Text Similarity:", text_vector_similarity)
    print("URL Prefixes Similarity:", address_similarity)

    image_similarity = 0
    if img1 and img2:
        ssim_sim = compare_images_ssim(img1, img2)
        phash_sim = compare_images_phash(img1_phash, img2_phash)
        orb_sim = compare_images_orb(img1, img2)
        image_similarity = (ssim_sim + phash_sim + orb_sim) / 3

        print("SSIM:", ssim_sim)
        print("PHash:", phash_sim)
        print("ORB:", orb_sim)
        print("Image Similarity average:", image_similarity)

    weight_headers = 0.1
    weight_html = 0.2
    weight_text_hash = 0.2
    weight_text_vector = 0.2
    weight_images = 0.2
    weight_address = 0.1

    probability = (
        header_similarity[1] * weight_headers +
        html_similarity * weight_html +
        text_hash_similarity * weight_text_hash +
        text_vector_similarity * weight_text_vector +
        image_similarity * weight_images +
        address_similarity * weight_address
    )
    print("Phishing probability evaluation...")
    return probability
```

Рисунок 3.15 – Код функції phishing_probability(url1, url2)

Алгоритм роботи інструменту phishing_detector.py реалізується у функції *phishing_probability(url1, url2)*. На рис. 3.16 наведена блок-схема, яка відображає послідовність викликів всіх функцій, які забезпечують отримання вмісту та подальший його аналіз.

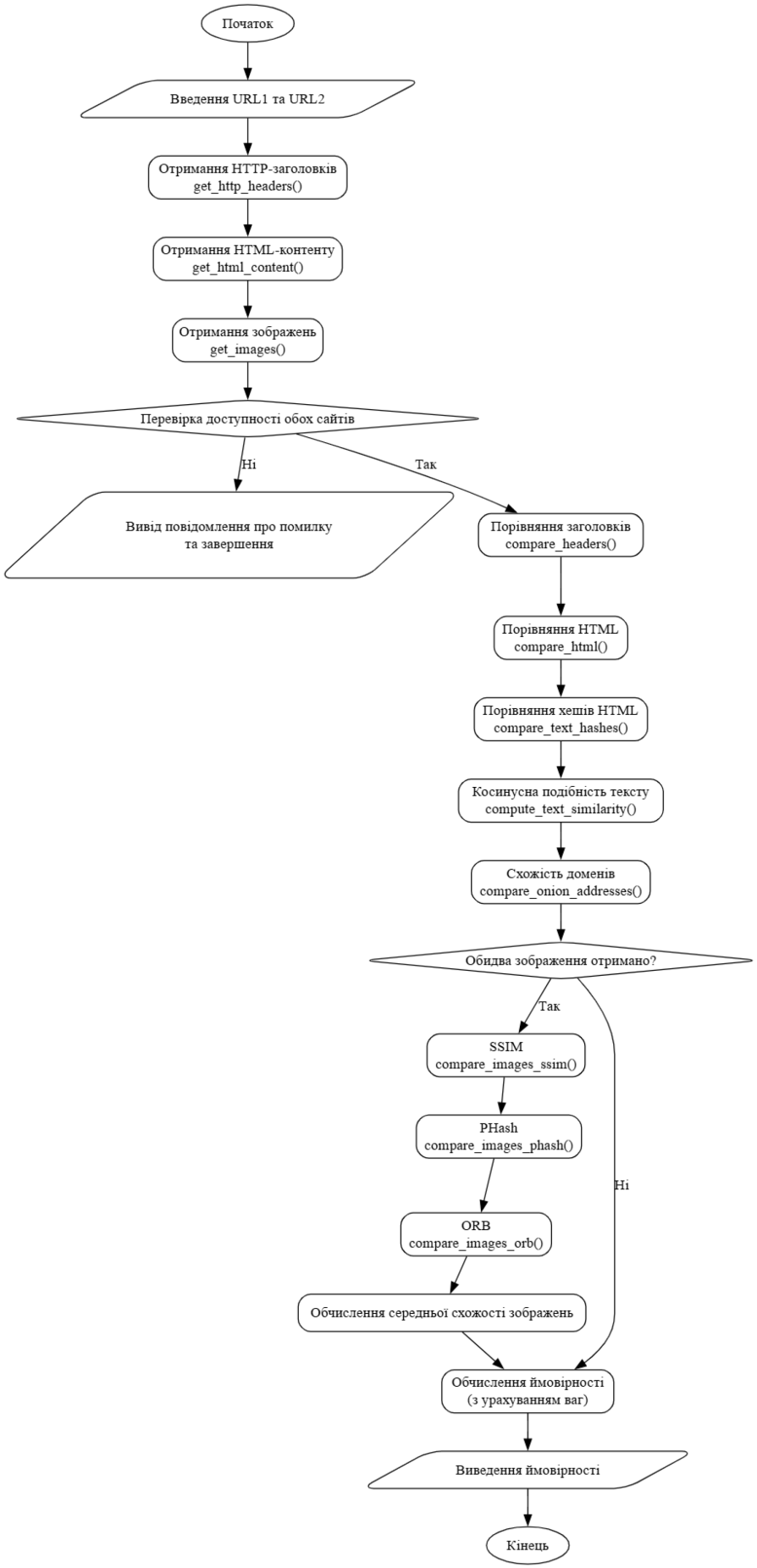

Рисунок 3.16 – Блок-схема функції phishing_probability (url1, url2).

Код на рис. 3.17 – точка входу в програму

```
if __name__ == "__main__":
    onion_url1 = input("Enter 1st .onion link: ")
    onion_url2 = input("Enter 2nd .onion link: ")

    probability = phishing_probability(onion_url1, onion_url2)
    print(f"Phishing imitation probability: {probability:.2%}")
```

Рисунок 3.17 – Точка входу в програму

Тут виконується запит користувача на ввід двох .onion-посиланнь. Вони будуть використовуватись як аргументи для порівняння.

onion_url1 = input("Enter 1st .onion link: ")

onion_url2 = input("Enter 2nd .onion link: ")

І після відпрацювання основної функції програми *phishing_probability(url1, url2)* значення оцінки ймовірності фішингової імітації виводиться в консоль у відсотках

probability = phishing_probability(onion_url1, onion_url2)

print(f"Phishing imitation probability: {probability:.2%}")

3.3 Інтерпретація результатів роботи програми

В ході роботи алгоритму для аналізу посилання на предмет фішингу, застосовані методи порівняння HTTP заголовків, HTML коду, текстового вмісту, зображень та префіксів посилань. Кожен з цих параметрів повертає значення, яке враховується під час розрахунку потенційної ймовірності фішингової імітації за порівнюваним посиланням.

Під час порівняння індексу схожості заголовків HTTP повертається кортеж, що містить два значення: типу float в діапазоні 0.0 – 1.0, що надає кількісну оцінку частки заголовків, що збігаються, де 1.0 трактується як те, що заголовки повністю збігаються; 0.0 – всі заголовки відсутні або відрізняються; значення < 0.7 може вважатися підозрілим, оскільки конфігурації серверів відрізняються. Друге значення кортежу

типу bool, що повертає False, якщо заголовки мають високу схожість (розцінюється як те, що сервери двох сайтів мають однакові налаштування і скоріш за все обидва є легітимними) і True, якщо індекс схожості менше 0.7, що говорить про різні налаштування серверів і з'являється підозра на фішингову імітацію.

Після порівняння HTML коду повертається число типу даних float, має діапазон значень 0.0 – 1.0, де значення 1.0 означає, що HTML-коди ідентичні, ~0.5 – часткова подібність, 0.0 – повністю різні структури.

Результат порівняння текстового вмісту з використанням хешування повертає значення типу bool, з діапазоном True або False, де True – хеші ідентичні, тобто вміст точно такий самий, а False – є хоча б незначна відмінність у тексті.

Після порівняння текстового вмісту за допомогою алгоритму косинусної подібності, для оцінки схожості двох текстів, повертається тип даних float з діапазоном значень 0.0 – 1.0, де 1.0 – тексти ідентичні; ~0.7–0.9 – дуже схожий текст; 0.1 – 0.6 – є незначна схожість; 0.0 – зовсім інший вміст.

Порівняння зображень з викристанням алгоритму SSIM повертає тип даних float у діапазоні -1.0 – 1.0, де значення слід розуміти як 1.0 – зображення ідентичні; >0.7 – майже ідентичні, ймовірно застосовувались певні перетворення; <0.4 – зовсім різні.

Вимірювання схожості зображень на основі pHash повертає float в діапазоні 0.0 – 1.0, де результати мають значення 1.0 – візуально схожі зображення; ~0.5 – частково схожі; 0.0 – різні зображення.

Результат порівняння зображень з алгоритмом ORB повертає float значення від 0.0 до 1.0, де 1.0 означає, що всі ключові точки збігаються; ~0.5 – частковий збіг структури; 0.0 – відсутність схожості.

Зіставлення префіксів адрес onion повертаэ тип даних bool, тобто True або False, де True – збіг префіксу (перших шести символів) ≥ 35%; False – префікс значно відрізняється.

В результаті роботи програма повертає відстоткове значення ймовірності того, що одне з посилань є фішинговою копією іншого.

У табл. 3.2 наведено діапазон значень у відсотках, які відображають значення ймовірності фішингової імітації, розрахований за критеріями порівняння HTTP заголовків, HTML коду, текстового вмісту, зображень та префіксів посилань; а також інтерпретація значення ймовірності у відсотках та рекомендовані подальші дії з сайтами, що перевірялись

Таблиця 3.2

Інтерпретація результатів роботи програми

Ймовірність (%)	**Інтерпретація**	**Рекомендовані дії**
0 – 24%	Низька ймовірність фішингу	Ймовірно безпечний сайт, можна використовувати
25 – 49%	Помірна схожість, потенційний ризик	Рекомендується додаткова ручна перевірка
50 – 74%	Висока ймовірність фішингової поведінки	Сайт підозрілий, не рекомендується взаємодія
75 – 100%	Дуже висока ймовірність фішингу	Імовірно фішинговий сайт, не використовувати

Отже, визначено приблизні діапазони та очікувані значення, які допоможуть орієнтуватись користувачам розробленого інструменту у кількісних показниках кожного з критеріїв, що оцінюються під час роботи алгоритму.

3.4 Запуск програми в термінальному середовищі

Для того, щоб запустити на виконання детектор з виявлення фішингу, необхідно виконати наступну команду (рис. 3.18)

```
(myenv) alina@fkbyf25:~/testdir$ python3 phishing_detector_updated.py
```

Рисунок 3.18 – Запуск програми в консолі

При запуску програма запитує у користувача два аргументи – посилання на сайти, які порівнюватимуться та визначатиметься ймовірність фішингової імітації між ними (рис. 3.19)

```
Enter 1st .onion link: http://rhysidafohrhyy2aszi7bm32tnjat5xri65fopcxkdfxhi4tic
sg7cad.onion/
Enter 2nd .onion link: http://rhysidafc6lm7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm
6yegad.onion/
```

Рисунок 3.19 – Введення аргументів

Перший етап виконання програми – це надсилання запитів до сайтів на отримання даних (HTTP-заголовки, HTML-код, зображення) (рис. 3.20).

```
Getting HTTP Headers...
Getting HTML Content...
Getting images...
Fetching image: http://rhysidafohrhyy2aszi7bm32tnjat5xri65fopcxkdfxhi4tidsg7cad.
onion/logos/logo.png
Fetching image: http://rhysidafc6lm7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm6yegad.
onion/logos/logo.png
⚠ Mismatched or missing headers detected:
 - date
```

Рисунок 3.20 – Отримання даних з сайтів

Другий етап – оцінювання за визначеними критеріями (рис. 3.21).

```
***Criteria evaluation...
HTTP Headers Similarity: 0.8333333333333334
HTML Similarity: 1.0
HTML Hash Similarity: True
Cosinus Text Similarity: 1.0000000000000004
URL Prefixes Similarity: True
SSIM: 1.0
PHash: 1.0
ORB: 1.0
Image Similarity average: 1.0
```

Рисунок 3.21 – Оцінювання отриманих значень

Останнім етапом є, власне, розрахунок ймовірності фішингової імітації між порівнюваними вебсайтами (рис. 3.22).

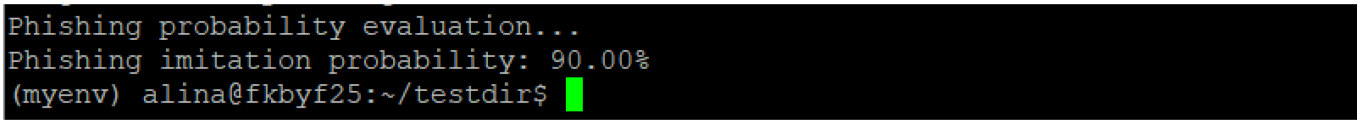

Рисунок 3.22 – Розрахунок ймовірності фішингової імітації

Як результат виконання програма повертає відсоткове значення ймовірності фішингу між двома посиланнями, наданими користувачем. Крім того, в процесі оцінювання критеріїв, в консоль виводиться значення оцінки кожного з них для більш детальної характеристики.

Висновки до розділу 3

У цьому розділі розроблено комплексний метод виявлення фішингових доменів .onion у Dark Web. Запропонований підхід інтегрує декілька методів виявлення, включаючи аналіз схожості на основі тексту, зображень, заголовків та адрес. Методологія була реалізована у вигляді програмного інструменту, який автоматизує процес виявлення фішингу шляхом отримання та порівняння ключових характеристик веб-сайтів.

Підхід до виявлення на основі тексту включав хеш-порівняння HTML-контенту та косинусну подібність для текстового аналізу. Ці методи дозволили виявити фішингові сайти, які копіюють легальні сервіси, змінюючи лише незначні текстові елементи.

Виявлення на основі зображень використовувало кілька алгоритмів, зокрема Perceptual Hash (PHash), ORB (Oriented FAST and Rotated BRIEF) та SSIM (Structural Similarity Index – індекс структурної схожості). Цей метод виявився ефективним у виявленні візуальної схожості між фішинговими та легальними веб-сайтами, навіть якщо на них були внесені незначні зміни.

Виявлення на основі адрес аналізувало схожість префіксів доменів .onion, використовуючи статистичні порогові значення для виявлення випадків, коли фішингові домени імітували адреси легальних сервісів.

РОЗДІЛ 4
ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ МЕТОДУ

В даному розділі буде проведено оцінювання розробленого методу, також можливі шляхи його практичного застосування, наведено характерику переваг, недоліків та напрями майбутніх досліджень.

4.1 Оцінювання ефективності методу виявлення фішингових посилань

Оцінювання ефективності розробленого методу виявлення фішингу має вирішальне значення для забезпечення його надійності у виявленні зловмисних доменів .onion і мінімізації помилкових спрацювань.

4.1.1 Визначення методу та метрик оцінювання

Для вимірювання точності системи виявлення фішингу було використано стандартний підхід на основі матриці невідповідностей [51], який поділяє результати класифікації на чотири типи:

1. Істинно позитивні результати (True Positives – TP) – фішинговий домен правильно ідентифіковано як фішинговий.

2. Хибнопозитивні (False Positives – FP) – легальний домен помилково класифікується як фішинговий.

3. Істинно негативні (True Negatives – TN) – легітимний домен правильно класифікується як легітимний.

4. Хибнонегативні (False Negatives – FN) – фішинговий домен помилково класифікується як легітимний.

Матриця невідповідностей показана у табл. 4.1. Вона відображає підсумки класифікації – число хибнопозитивних, хибнонегативних, істинно позитивних та істинно негативних результатів

Таблиця 4.1

Матриця невідповідностей

		Прогрозоване	
		Позитивне (P)	Негативне
Фактичне	Позитивне (P)	Істинно позитивне (TP)	Істинно негативне (TN)
	Негативне (N)	Хибнопозитивне (FP)	Хибнонегативне (FN)

На основі цих значень розраховуються ключові показники ефективності – точність, ефективність, оцінка загальної ефективності та продуктивність.

Показник точності вимірює, скільки з доменів, ідентифікованих як фішингові, виявлені правильно та обчислюється за формулою 4.1. Високий показник точності вказує на низький рівень помилкових спрацьовувань, що гарантує, що легітимні веб-сайти не будуть помилково позначені як фішингові.

$$precision = \frac{TP}{TP+FP} \tag{4.1}$$

Ефективність показує, наскільки добре інструмент виявляє фішингові домени серед усіх фішингових сайтів з тестового датасету. Розраховується за формулою 4.2 як відношення кількості істино-позитивних випадків до суми істино позитивних і хибнонегативних випадків.

$$recall = \frac{TP}{TP+FN} \tag{4.2}$$

Оцінка загальної ефективності – це середнє гармонійне значення точності та ефективності моделі, яка розраховується за формулою 4.3. Ця метрика особливо корисна, коли потрібно одночасно мінімізувати помилкові спрацьовування і помилкові негативні результати. Чим вище значення оцінки загальної ефективності, тим краща точність системи.

$$F1\text{-}Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4.3)$$

Продуктивність моделі розраховується за формулою 4.4 як відношення загальної кількості правильних випадків до загальної кількості випадків.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.4)$$

4.1.2 Алгоритм роботи інструменту оцінювання ефективності

Для оцінки інструменту виявлення фішингу було використано набір даних, що містить пари доменів .onion, позначені як фішингові або легальні. Інструмент було протестовано на цьому наборі даних, і на основі результатів класифікації було обчислено матрицю невідповідностей.

Щоб оцінити ефективність розробленого інструменту, було реалізовано програмний код, що розраховує вищезазначені показники ефективності. Було встановлено порогове значення 0,5. Якщо розрахований показник схожості між двома доменами .onion перевищував цей поріг, інструмент класифікував пару доменів як фішингову; в іншому випадку вона класифікувалася як легітимна.

Функція *phishing_probability* в програмній реалізації методу *phishing_detector.py* є основою для оцінки ймовірності фішингу між двома .onion-доменами. Вона отримує дві .onion-адреси та аналізує їх схожість за різними параметрами і повертає певний показник (ступінь ймовірності фішингу), який потім порівнюється з встановленим порогом (0.5), щоб зробити фінальне рішення.

Тобто, якщо функція *phishing_probability(url1, url2)* повертає значення ≥ 0.5, то сайт вважається фішинговим, а якщо значення < 0.5, сайт вважається легітимним.

Програма, що виконує оцінку ефективності має назву evaluate_phishing_detector.py. Першим кроком імпортується основна функція

phishing_probability, яка розраховує показник ймовірності фішингу, та визначений поріг, перевищення значення якого прийнято трактувати як фішинг (рис. 4.1).

```
from phishing_detector_updated import phishing_probability

threshold = 0.5
```

Рисунок 4.1 – Імпорт функції та визначення порогу оцінювання

evaluate_phishing_detector.py містить в собі тестовий датасет (рис. 4.2)

```
# test dataset
test_data = [
    # phishing links
    {"url1": "http://arkanabb66ee4nsdji61a2bu6bwqe3dbtsyf3rxrv6vhiehod7utagad.onion/", "url2":
    "http://ransomwvbabemdnwl7lzgeenyfmmhskaed6jcruwhkvapsia76vttzyd.onion/", "label": "phishing"},
    {"url1": "http://rhysidafohrhyy2aszi7bm32tnjat5xri65fopcxkdfxhi4tidsg7cad.onion/", "url2":
    "http://rhysidafc61m7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm6yegad.onion/", "label": "phishing"},
    {"url1": "http://lynxblogxstgzsarfyk2pvhdv45igghb4zmthnzmsipzeoduruz3xwqd.onion/leaks", "url2":
    "http://lynxblogco7r37jt7p5wrmfxzqze7ghxw6rihzkqc455qluacwotciyd.onion/leaks", "label": "phishing"},
    {"url1": "http://mbrlkbtq5jonaqkurjwmxftytyn2ethqvbxfu4rgjbkkknndqwae6byd.onion/", "url2":
    "http://k7kg3jqxang3wh7hnmaiokchk7qoebupfgoik6rha6mjpzwupwtj25yd.onion/", "label": "phishing"},
    {"url1": "http://basheqtvzqwz4vp6ks5lm2ocq7i6tozqgf6vjcasj4ezmsy4bkpshhyd.onion/page_company.php?id=124", "url2":
    "http://basherq53eniermxovo3bkduw5qqq5bkqcml3qictfmamgvmzovykyqd.onion/page_company.php?id=124", "label": "phishing"},
    {"url1": "http://weg7sdx54bevnvulapqu6bpzwztryeflq3s23tegbmnhkbpqz637f2yd.onion/", "url2":
    "http://c7jpc6h2ccrdwmhofuij7kz6sr2fg2ndtbvvqy4fse23cf7m2e5hvqid.onion/", "label": "phishing"},

    # legitimate links
    {"url1": "http://arkanabb66ee4nsdji61a2bu6bwqe3dbtsyf3rxrv6vhiehod7utagad.onion/", "url2":
    "http://mbrlkbtq5jonaqkurjwmxftytyn2ethqvbxfu4rgjbkkknndqwae6byd.onion/", "label": "legitimate"},
    {"url1": "http://basherq53eniermxovo3bkduw5qqq5bkqcml3qictfmamgvmzovykyqd.onion/", "url2":
    "http://lynxblogxstgzsarfyk2pvhdv45igghb4zmthnzmsipzeoduruz3xwqd.onion/leaks", "label": "legitimate"},
    {"url1": "http://rhysidafc61m7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm6yegad.onion/", "url2":
    "http://arkanabb66ee4nsdji61a2bu6bwqe3dbtsyf3rxrv6vhiehod7utagad.onion/", "label": "legitimate"},
    {"url1": "http://k7kg3jqxang3wh7hnmaiokchk7qoebupfgoik6rha6mjpzwupwtj25yd.onion/", "url2":
    "http://c7jpc6h2ccrdwmhofuij7kz6sr2fg2ndtbvvqy4fse23cf7m2e5hvqid.onion/", "label": "legitimate"},
    {"url1": "http://weg7sdx54bevnvulapqu6bpzwztryeflq3s23tegbmnhkbpqz637f2yd.onion/", "url2":
    "http://lynxblogco7r37jt7p5wrmfxzqze7ghxw6rihzkqc455qluacwotciyd.onion/leaks", "label": "legitimate"},
]
```

Рисунок 4.2 – Тестовий датасет

Визначена логіка класифікації результатів виконання оцінки ймовірності фішингу порівнюваних посилань у відповідності до порогового значення (рис. 4.3)

```
for item in test_data:
    print("=" * 60)
    print(f"🔍 Comparing:\n{item['url1']}  ⇔  {item['url2']}")

    try:
        score = phishing_probability(item["url1"], item["url2"])
    except Exception as e:
        print(f"❌ Error during phishing_probability: {e}")
        continue

    is_phishing = score >= threshold
    print(f"🎯 Score: {score:.3f} → {'Phishing' if is_phishing else 'Legitimate'} (expected: {item['label']})")

    if is_phishing and item["label"] == "phishing":
        TP += 1
    elif is_phishing and item["label"] == "legitimate":
        FP += 1
    elif not is_phishing and item["label"] == "legitimate":
        TN += 1
    elif not is_phishing and item["label"] == "phishing":
        FN += 1
```

Рисунок 4.3 – Логіка класифікації результатів виконання оцінки ймовірності фішингу порівнюваних посилань

Прописані формули для розрахунку метрик оцінювання ефективності програми та вивід результатів оцінки ефективності у консоль (рис. 4.4)

```
precision = TP / (TP + FP) if (TP + FP) else 0
recall = TP / (TP + FN) if (TP + FN) else 0
f1 = 2 * precision * recall / (precision + recall) if (precision + recall) else 0
accuracy = (TP + TN) / (TP + FP + TN + FN) if (TP + FP + TN + FN) else 0

print("\n=== 📊 Evaluation Results ===")
print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
print(f"Precision: {precision:.2f}")
print(f"Recall:    {recall:.2f}")
print(f"F1 Score:  {f1:.2f}")
print(f"Accuracy:  {accuracy:.2f}")
```

Рисунок 4.4 – Формули для розрахунку метрик оцінювання ефективності програми

4.1.3 Виконання тестування ефективності розробленого інструменту

Тестовий датасет складається з одинадцяти пар посилань (6 фішингових, 5 легітимних). Запустивши *evaluate_phishing_detector.py*, було отримано розраховані значення ймовірності фішингової імітації та порівняння прогнозованого результату з фактичними для всіх одинадцяти пар. Результат оцінки однієї з фішингових пар (рис. 4.5 – 4.6) посилань показано на рис. 4.7.

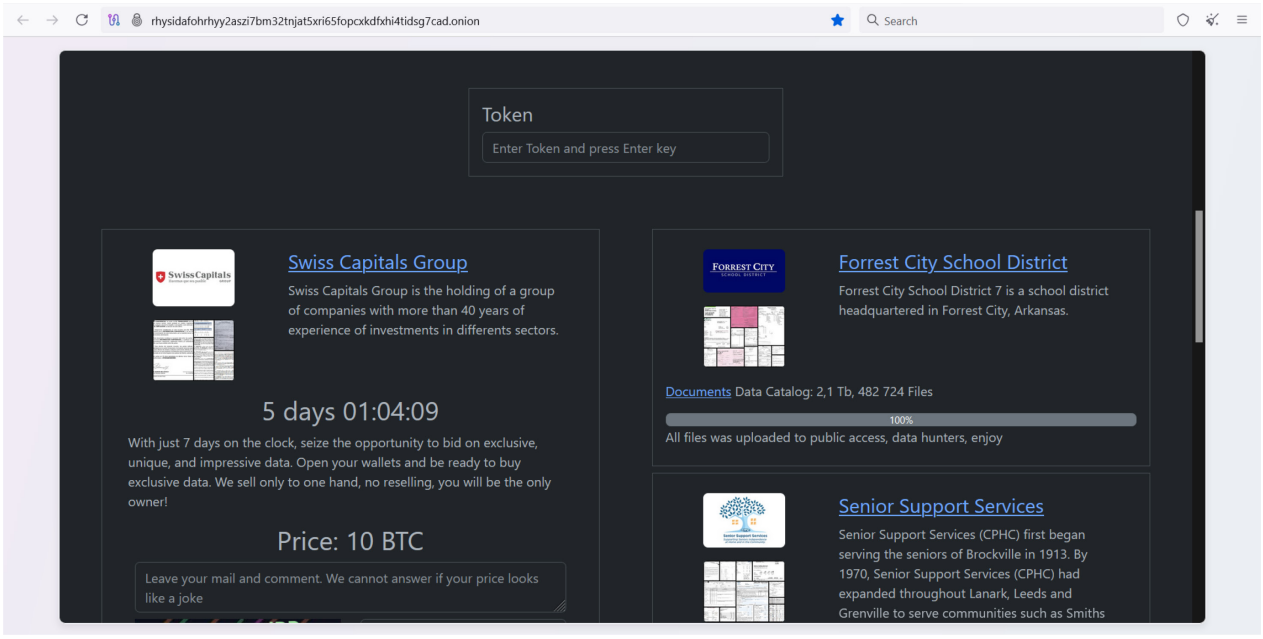

Рисунок 4.5 – Скріншот автентичного порівнюваного сайту з фішингової пари

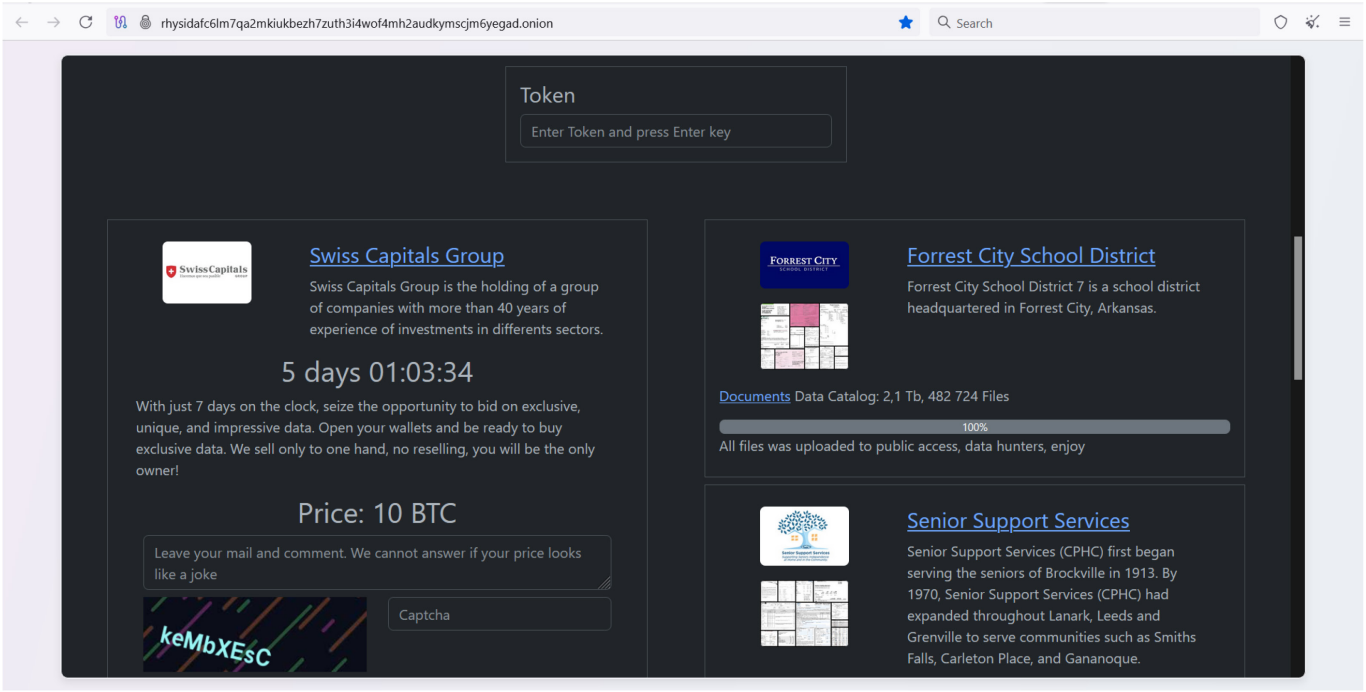

Рисунок 4.6 – Скріншот імітованого порівнюваного сайту з фішингової пари

```
============================================================
🔍 Comparing:
http://rhysidafohrhyy2aszi7bm32tnjat5xri65fopcxkdfxhi4tidsg7cad.onion/ ↔ http:
//rhysidafc6lm7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm6yegad.onion/
Getting HTTP Headers...
Getting HTML Content...
Getting images...
Fetching image: http://rhysidafohrhyy2aszi7bm32tnjat5xri65fopcxkdfxhi4tidsg7cad.
onion/logos/logo.png
Fetching image: http://rhysidafc6lm7qa2mkiukbezh7zuth3i4wof4mh2audkymscjm6yegad.
onion/logos/logo.png
***Criteria evaluation...
HTTP Headers Similarity: 1.0
HTML Similarity: 1.0
HTML Hash Similarity: True
Cosinus Text Similarity: 1.0000000000000004
URL Prefixes Similarity: True
SSIM: 1.0
PHash: 1.0
ORB: 1.0
Image Similarity average: 1.0
Phishing probability evaluation...
💡 Score: 0.900 → Phishing (expected: phishing)
============================================================
```

Рисунок 4.7 – Результат оцінки однієї з фішингових пар

Візуальний вигляд однієї з легітимних пар посилань показано на рис 4.8 – 4.9.

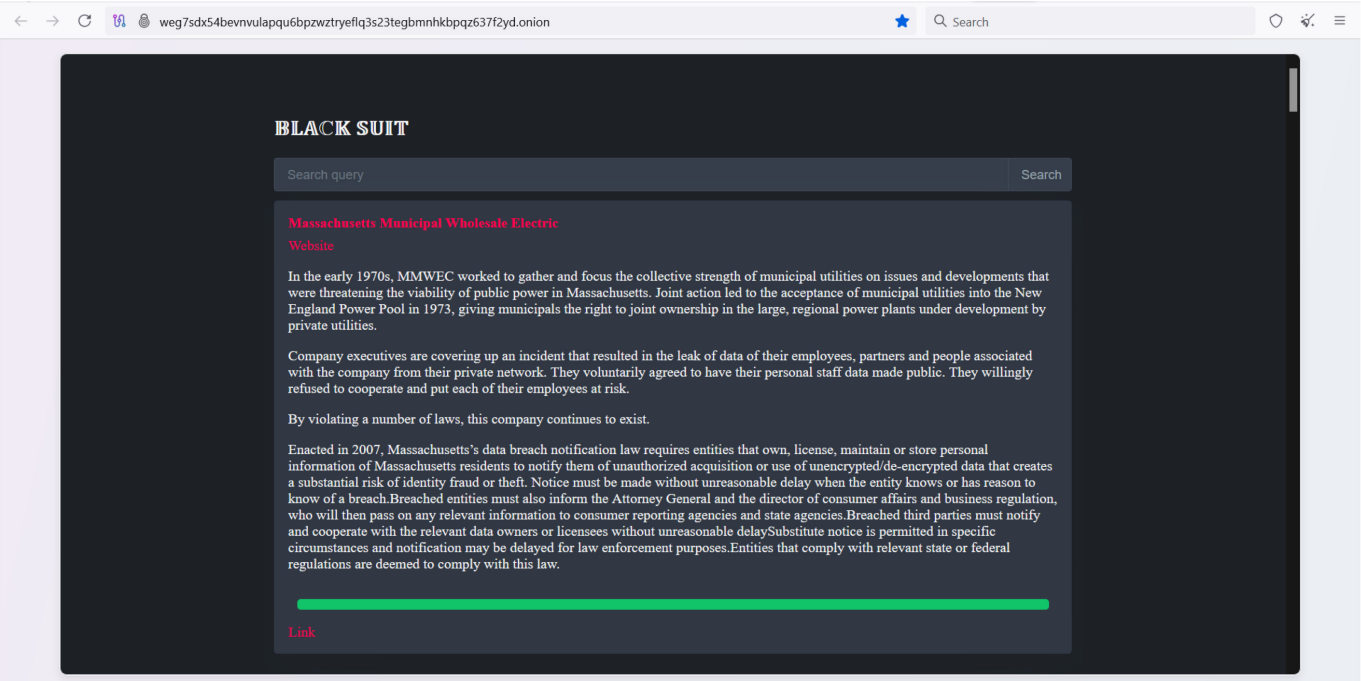

Рисунок 4.8 – Скріншот першого автентичного порівнюваного сайту з легітимної пари

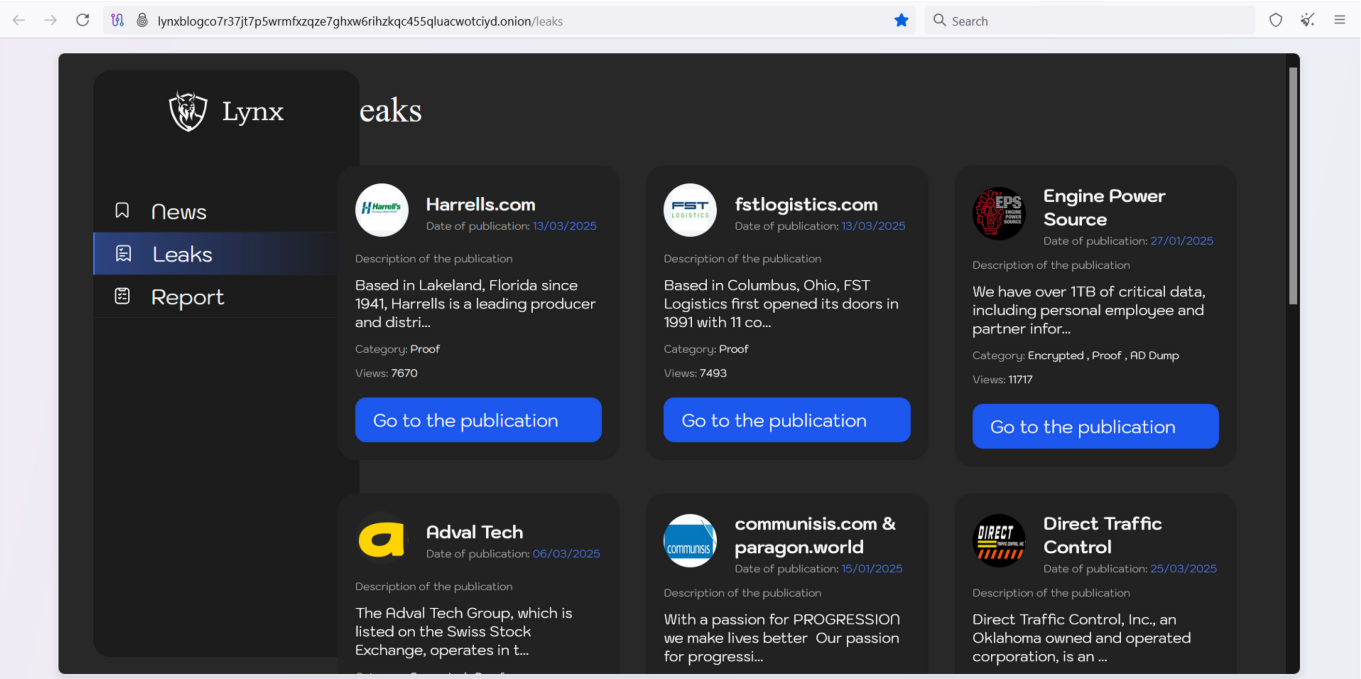

Рисунок 4.9 – Скріншот другого автентичного порівнюваного сайту з легітимної пари

Результат оцінки однієї з легітимних пар посилань показано на рис. 4.10.

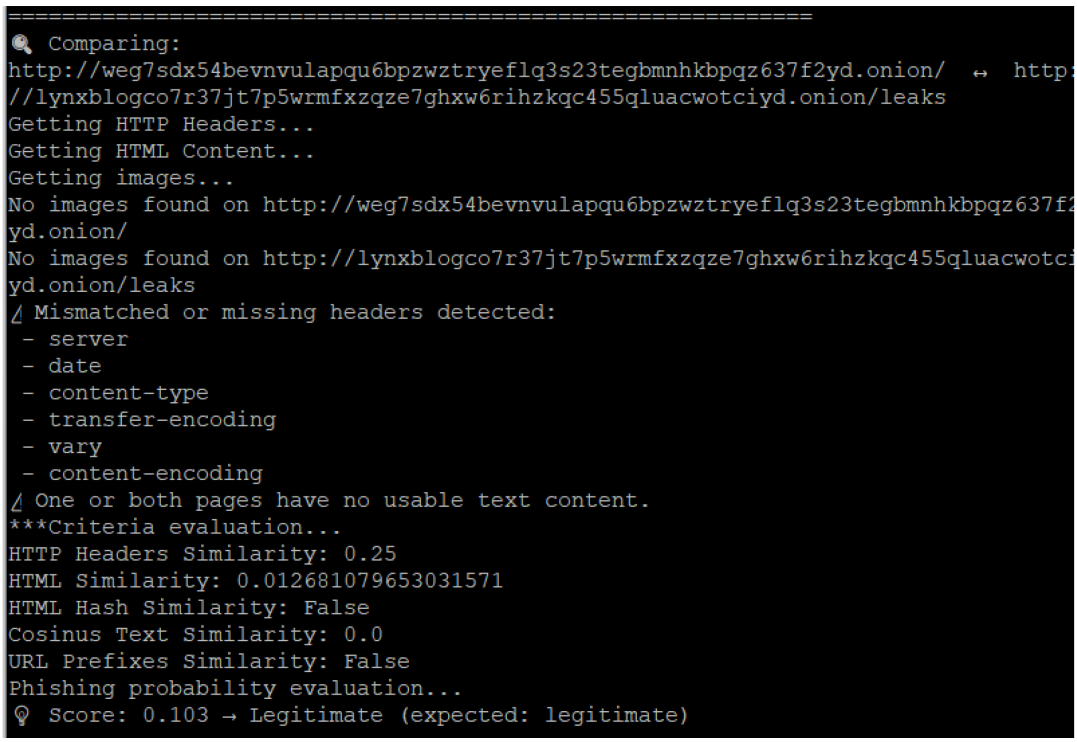

Рисунок 4.10 – Результат оцінки однієї з легітимних пар

Було зафіксоване одне помилкове спрацювання детектора – хибнонегативне (рис. 4.11)

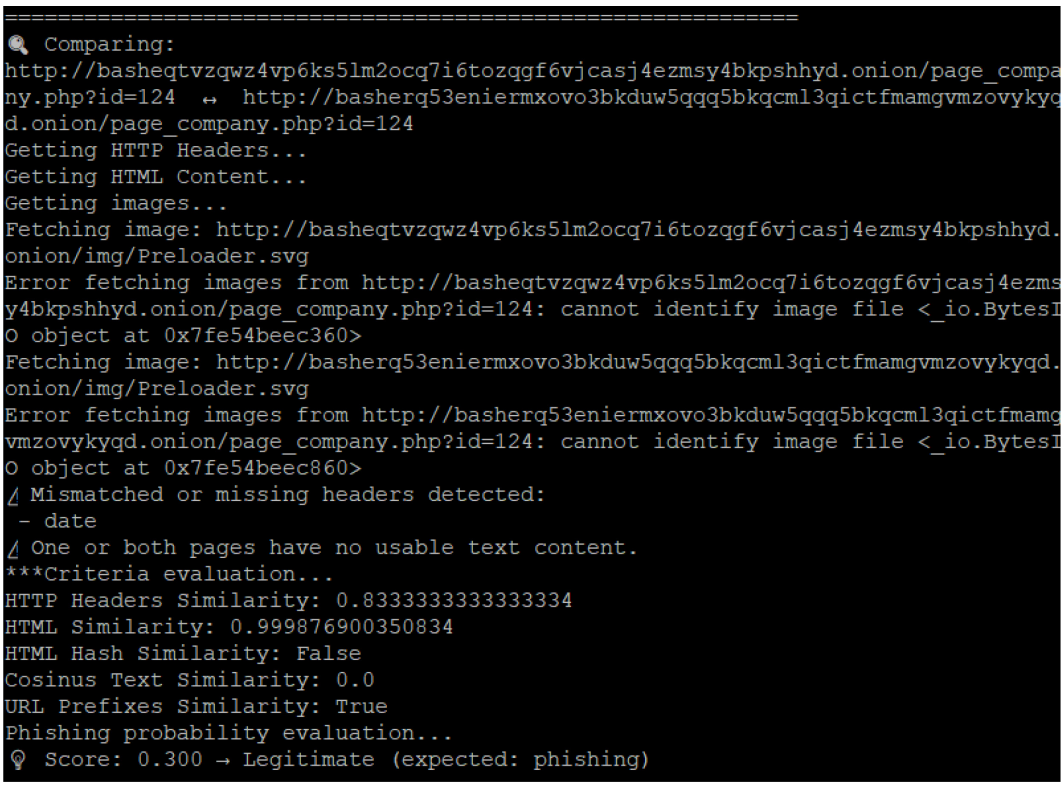

Рисунок 4.11 – Хибнонегативне значення

Враховуючи це, загальні результати оцінки ефективності показують високу точність розробленого інструменту (рис. 4.12)

Рисунок 4.12 – Результати оцінки ефективності

Отримане значення Precision = TP / (TP+FP) = 5/ (5+0) = 1.00 означає, що всі сайти, які були класифіковані як фішингові, справді є такими. Відсутність хибних позитивних спрацювань (FP = 0) підтверджує, що система не помилково блокує легітимні сайти.

Результат розрахунку показника Recall = TP / (TP + FN) =5 / (5+1) = 0.83 означає, що 83% усіх фішингових сайтів були правильно виявлені. Водночас наявність одного випадку False Negative (FN = 1) свідчить про те, що система може не ідентифікувати деякі фішингові сайти. В даній тестовій парі причиною цього могла стати помилка при завантаженні зображень з сайтів, а також відсутність текстових тегів HTML. Оскільки це два критерії з п’яти оцінюваних, це значно вплинуло на результуючий показник ймовірності.

Значення продуктивності Accuracy = (TP+TN)/(TP+FP+TN+FN) = (5+5) / (5+0+5+1) = 0.91 означає, що 91% усіх перевірених доменів були правильно класифіковані. Високе значення продуктивності підтверджує загальну ефективність системи.

Розрахунок показника F1 Score = 2* ((Presision * Recall) / (Presision + Recall))= 2 * ((1*0.83)/(1+0.83)) = 0.91 свідчить про хорошу збалансованість між точністю та повнотою, тобто система не лише добре виявляє фішингові сайти, а й майже не робить помилкових спрацювань.

Наочне представлення результатів оцінювання ефективності програмного засобу наведене у табл. 4.2.

Таблиця 4.2

Результати оцінки ефективності програмного засобу

Посилання на автентичний сайт	Посилання на фішингову копію	Посилання на сайт, який не є фішинговою копією	Прогнозований результат	Розраховане значення	Фактичний результат
http://arkanabb66ee.....onion/	http://ransomwvbab....onion/		phishing	0.6	phishing
http://rhysidafohrhyy2as....onion	http://rhysidafc6l...gad.onion		phishing	0.9	phishing
http://lynxblogxstgzsar....onion	http://lynxblogco...iyd.onion		phishing	0.5	phishing
http://mbrlkbtq5jonaqkur....onion/	http://k7kg3jqxang...5yd.onion		phishing	0.5	phishing
http://basheqtvzqwz4vp....onion	http://basherq53enier...yqd.onion		phishing	0.3	legitimate
http://weg7sdx54bevnyd....onion	http://c7jpc6h2ccrdwmho...qid.onion		phishing	0.6	phishing
http://arkanabb6...gad.onion		http://mbrlkbtq5jonaqku...byd.onion	legitimate	0.102	legitimate
http://basherq53enier...yqd.onion		http://lynxblogxstgzsar...wqd.onion	legitimate	0.101	legitimate
http://rhysidafc6lm7qa...gad.onion		http://arkanabb66ee4ns...gad.onion	legitimate	0.114	legitimate
http://k7kg3jqxang3...5yd.onion		http://c7jpc6h2ccrdwm...qid.onion	legitimate	0.104	legitimate
http://weg7sdx54bevn...2yd.onion		http://lynxblogco7r37jt...iyd.onion	legitimate	0.103	legitimate

Загалом, отримані результати оцінки підтверджують ефективність використаних методів, а також окреслюють можливі напрями подальшого розвитку системи.

4.2 Застосування розробленого засобу для виявлення фішингу

Розроблений метод орієнтований на виявлення фішингових атак у мережі Tor шляхом комплексного порівняння двох onion-сайтів.

Застосування запропонованого методу є доцільним у таких випадках (рис. 4.13)

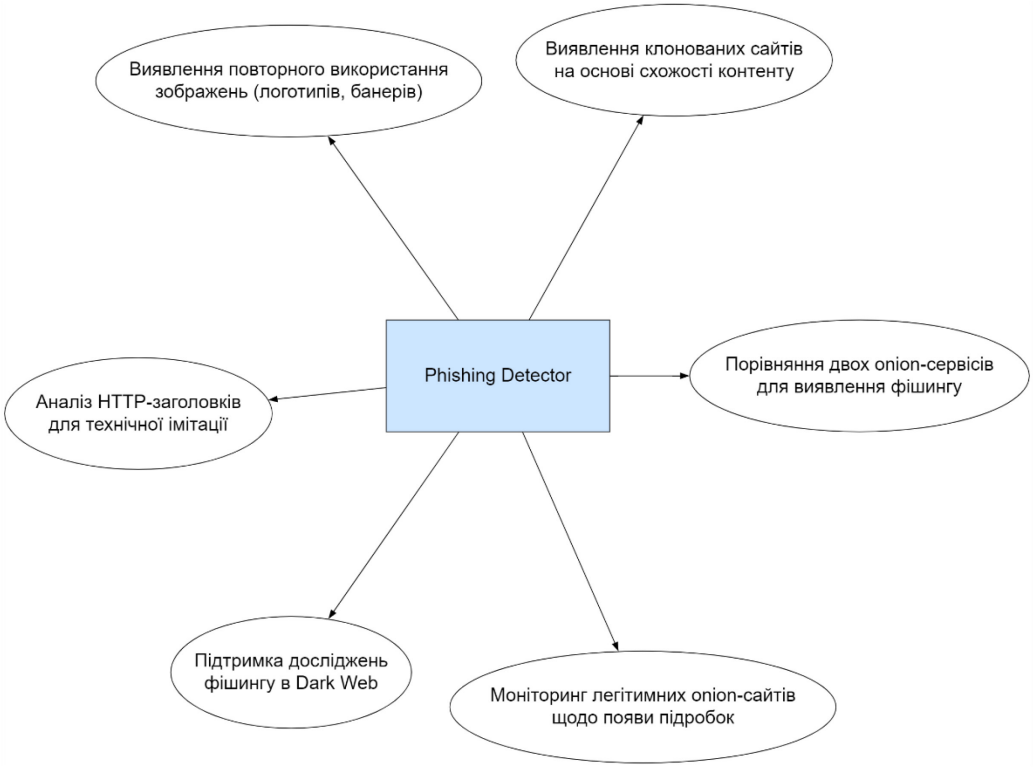

Рисунок 4.13 – Шляхи застосування методу

1. Порівняння двох onion-сервісів для виявлення можливого фішингу. Розроблена програма дозволяє перевірити, чи є другий сайт фішинговою копією першого, шляхом аналізу технічних і контентних характеристик.

2. Виявлення клонованих сайтів на основі подібності вмісту. Метод визначає схожість HTML-коду, текстового наповнення та структури сторінок, що допомагає розпізнати фішингові копії, навіть якщо вони частково модифіковані.

3. Виявлення повторного використання графічних матеріалів. Програма аналізує зображення (логотипи, банери), щоб знайти схожі або ідентичні графічні елементи, які часто копіюють фішингові сайти для підвищення довіри користувачів.

4. Аналіз HTTP-заголовків для виявлення технічної імітації. За допомогою порівняння HTTP-заголовків програма виявляє схожість у налаштуваннях серверів, що може свідчити про клонування сервісів.

5. Підтримка досліджень фішингу в Dark Web. Інструмент може використовуватися для наукових досліджень та збору статистики щодо масових фішингових кампаній у домені .onion, дозволяючи вивчати типові патерни фішингових атак.

6. Моніторинг легітимних onion-сервісів щодо появи несанкціонованих копій. Представлений метод може бути корисним для власників onion-сайтів, які хочуть перевіряти, чи не з’явилися в мережі підробки їхніх ресурсів.

Таким чином, розроблена метод є універсальним інструментом для аналізу onion-сервісів, що дозволяє підвищити рівень безпеки користувачів Dark Web і полегшує виявлення фішингових атак через багаторівневий аналіз сайтів.

4.3 Переваги, недоліки та майбутні напрямки досліджень

4.3.1 Переваги розробленого методу

До переваг методу, представленого в роботі, можна віднести:

1. Комплексний підхід. Метод використовує кілька рівнів аналізу: аналіз HTTP-заголовків (ідентифікація подібних серверних конфігурацій), HTML-код та текстовий вміст (виявлення схожості у структурі сторінок), аналіз доменного імені (подібність .onion-адрес), обробка зображень (визначення схожості логотипів та інших елементів). Це зменшує ймовірність помилкових спрацьовувань, оскільки атака може бути реалізована на різних рівнях.

2. Гнучкість та адаптивність. Використання вагової моделі дозволяє регулювати важливість окремих ознак. Можна легко інтегрувати додаткові ознаки, такі як аналіз стилю оформлення або поведінкові особливості сторінки.

3. Підтримка Dark Web. Багато існуючих фішингових детекторів орієнтовані на Clear Web. Представлений метод спеціально адаптований для .onion-сайтів, що рідко розглядаються у традиційних підходах.

4. Використання ефективних алгоритмів. Методи, засновані на порівнянні тексту дають швидкі результати для текстового аналізу. Методи, засновані на порівнянні зображень (PHash, SSIM, ORB) працюють навіть у випадку незначних модифікацій зображень (наприклад, змін у розмірі або контрасті). Аналіз доменних імен допомагає ідентифікувати атаки, що базуються на генерації схожих адрес (typosquatting).

5. Автономність та анонімність. Метод не потребує централізованої бази фішингових сайтів, що важливо для роботи у дарквебі, де домени часто змінюються або працюють обмежений час.

4.3.2 Недоліки розробленого методу

До недоліків методу, представленого в роботі, можна віднести:

1. Висока чутливість до змін контенту. Фішингові сайти можуть навмисно змінювати структуру HTML-коду та HTTP-заголовків, що знижує ефективність методу.

2. Проблеми з доступністю контенту. Деякі .onion-сайти працюють лише для певного списку користувачів або вимагають авторизації. Блокування ботів та наявність CAPCHA може ускладнити аналіз HTML-коду та зображень.

3. Обмежена стійкість до генеративних атак. Сучасні фішингові сайти можуть використовувати штучний інтелект для динамічного створення контенту, що ускладнює виявлення на основі подібності.

4. Витрати ресурсів. Аналіз HTML, тексту та зображень може бути ресурсоємним, особливо для великих сторінок або при обробці великої кількості сайтів.

4.3.3 Майбутні напрямки досліджень

Нижче наведений перелік можливих покращень методу в майбутніх дослідженнях:

1. Використання машинного навчання. Замість ручного підбору вагових коефіцієнтів можна використовувати нейромережі або ансамблеві методи для автоматичного навчання моделі на основі реальних фішингових атак.

2. Аналіз поведінкових ознак. Додаткові перевірки, пов’язані з часом роботи .onion-сайту (фішингові домени часто існують недовго), частотою змін контенту (динамічні зміни можуть свідчити про шкідливу активність), аналізом мета-тегів та редиректів (наприклад, приховані перенаправлення на фішингові ресурси).

3. Використання мережевого аналізу. Впровадити дослідження зв’язків між .onion-домена через графи взаємопосилань, аналіз вхідного/вихідного трафіку для виявлення підозрілих схем взаємодії.

4. Покращення обробки зображень. Використання глибоких нейромереж (CNN, Vision Transformers) для виявлення схожих логотипів або елементів дизайну.

Висновки до розділу 4

В даному розділі було проведено оцінку ефективності розробленого методу виявлення фішингових адрес в домені .onion з використанням матриці невідповідностей. Експериментальна оцінка продемонструвала ефективність розробленого методу. Система була протестована на наборі даних сайтів в домені .onion. Результати оцінювання ефективності показали, що інструмент ефективно виявляє фішингові .onion-домени, демонструючи високу точність (1.00), повноту (0.83) та загальну правильність класифікації (0.91). Однак були виявлені деякі обмеження. Основною проблемою є ризик того, що деякі з оцінюваних критеріїв, на яких базується метод детектування, можуть бути некоректно оцінені з причини їх відсутності (зображення, текстовий вміст у HTML-коді) або недоступності (HTTP-заголовки, HTML-код, затримки при підключенні до вебсайту). В такому випадку

результати програми можуть піддаватись сумніву і потребуватимуть додаткової ручної перевірки. Така проблема може бути виправлена шляхом додавання додаткових методів аналізу та зміни ваги для кожного з критеріїв оцінки. Загалом, отримані результати оцінки підтверджують ефективність використаних методів, а також окреслюють можливі напрями подальшого розвитку системи.

Розроблений метод забезпечує міцну основу для автоматизованого виявлення фішингу в Dark Web з потенційним застосуванням для моніторингу загроз кібербезпеки та збору розвідувальної інформації.

ВИСНОВКИ

У кваліфікаційній роботі було представлено метод виявлення фішингових адрес у домені .onion. Дослідження включало аналіз методів виявлення фішингу, розробку підходу на основі множинних метрик схожості та проведення експериментальної оцінки ефективності запропонованого методу.

У першому розділі було проаналізовано фундаментальні аспекти фішингових атак у Dark Web, включаючи структуру та особливості анонімності мережі Tor. Проведено огляд методів, які використовують кіберзлочинці для створення фішингових сайтів у домені .onion. Крім того, було проаналізовано нормативно-правову базу щодо запобігання фішингу. Аналіз підтвердив, що фішинг у Dark Web створює унікальні проблеми через анонімність і складність перевірки автентичності сервісів у домені .onion. Крім того, огляд літератури підкреслив необхідність створення спеціалізованих механізмів виявлення, пристосованих до специфічних характеристик мережі Tor. Проаналізовано переваги та недоліки існуючих рішень виявлення фішингу в Dark Web, що дозволило визначити прогалини та розробити метод, що включатиме багаторівневий аналіз для виконання поставленої задачі.

Другий розділ був присвячений проєктуванню багаторівневої методології виявлення фішингу в домені .onion. Було запропоновано кілька ключових методів виявлення, включаючи аналіз заголовків HTTP, схожість структури HTML, порівняння текстового контенту, виявлення схожості зображень та зіставлення URL-адрес. Аналіз HTTP-заголовків допоміг виявити невідповідності в конфігураціях серверів між легальними та фішинговими вебсайтами. Для виявлення дубльованих або злегка змінених вебструктур використовувалися методи на основі хешування та зіставлення послідовностей. Аналіз тексту використовував векторизацію тексту і косинусну подібність для виявлення клонування контенту, а порівняння на основі зображень включало SSIM, перцептивне хешування (PHash) і ORB для виявлення візуальної схожості логотипів, банерів та інших візуальних елементів. Розроблений метод виявлення об'єднав зазначені критерії для підвищення точності ідентифікації

фішингу. Крім того, було детально описано архітектуру системи, функціональні компоненти та робочий процес.

У третьому розділі було програмно реалізовано метод виявлення фішингових вебсайтів у домені .onion. Було детально описано процес налаштування середовища, реалізацію функціоналу, включаючи методи збору HTML-коду, HTTP-заголовків, зображень, а також їх подальшого аналізу. Інтегровано алгоритми для порівняння текстового контенту на основі хешування та векторизації, методи для оцінки візуальної схожості зображень (SSIM, ORB, pHash) і аналіз схожості onion-адрес. Було побудовано математичну модель оцінки ймовірності фішингу, яка поєднує результати з усіх методів і надає комплексну оцінку ризику. Програму протестовано на реальних прикладах onion-сайтів у середовищі Tor, що підтвердило її здатність виявляти фішингові імітації навіть у випадках часткової модифікації контенту. Отримані результати підтверджують працездатність методів і доцільність їхньої інтеграції в єдину систему для виявлення фішингових ресурсів у Dark Web.

У четвертому розділі оцінено ефективність розробленого методу виявлення фішингу шляхом експериментальної перевірки. Для автоматизації виявлення фішингу було використано аналіз схожості коду, тексту, зображень, HTTP-заголовків та адрес. Експериментальні результати продемонстрували високу ефективність у виявленні фішингових доменів .onion. Метрики оцінювання показали високу точність (1,00), ефективність (0,83) та продуктивність класифікації (0,91). Однак були виявлені деякі обмеження, зокрема випадки, коли певні критерії оцінки були відсутні або недоступні, що впливало на точність виявлення. Щоб вирішити ці проблеми, майбутні вдосконалення можуть включати додавання додаткових методів аналізу та динамічне коригування вагових коефіцієнтів параметрів оцінювання. Отже, були окреслені сильні сторони та обмеження запропонованого методу та рекомендації щодо покращення у майбутніх напрямках досліджень.

Підбиваючи підсумки, в результаті дослідження було успішно розроблено ефективний метод виявлення фішингу в Dark Web, що поєднує технічні, текстові та візуальні індикатори. Запропонована система має значний потенціал для інтеграції в систему моніторингу кібербезпеки та розвідки загроз у мережі Tor. Майбутні

дослідження можуть бути зосереджені на підвищенні адаптивності моделі до еволюції тактик фішингу та розширенні її застосування до ширших сценаріїв виявлення загроз у Dark Web.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bernaschi, M., Celestini, A., Cianfriglia, M., Guarino, S., Lombardi, F., & Mastrostefano, E. (2022). Onion under Microscope: An in-depth analysis of the Tor Web. World Wide Web, 25(3), 1287-1313.

2. Pastor-Galindo, J., Sandlin, H. Â., Mármol, F. G., Bovet, G., & Pérez, G. M. (2024). A Big Data architecture for early identification and categorization of dark web sites. Future Generation Computer Systems, 157, 67-81.

3. Adewopo, V., Gonen, B., Varlioglu, S., & Ozer, M. (2019). Plunge into the underworld: A survey on emergence of darknet. In 2019 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 155-159). IEEE.

4. Dingledine, R., Mathewson, N., & Syverson, P. F. (2004, August). Tor: The second-generation onion router. In USENIX security symposium (Vol. 4, pp. 303-320).

5. Cilleruelo, C., De-Marcos, L., Junquera-Sánchez, J., & Martinez-Herraiz, J. J. (2020). Interconnection between darknets. IEEE Internet Computing, 25(3), 61-70.

6. Das, A., Baki, S., El Aassal, A., Verma, R., & Dunbar, A. (2019). SoK: a comprehensive reexamination of phishing research from the security perspective. IEEE Communications Surveys & Tutorials, 22(1), 671-708.

7. Aleroud, A., & Zhou, L. (2017). Phishing environments, techniques, and countermeasures: A survey. Computers & Security, 68, 160-196.

8. Kalaharsha, P., & Mehtre, B. M. (2021). Detecting Phishing Sites--An Overview. arXiv preprint arXiv:2103.12739.

9. Gadient, P., Gerig, P., Nierstrasz, O., & Ghafari, M. (2021). Phish What You Wish. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS) (pp. 1048-1059). IEEE.

10. Wong, A., Abuadbba, A., Almashor, M., & Kanhere, S. (2022). PhishClone: measuring the efficacy of cloning evasion attacks. arXiv preprint arXiv:2209.01582.

11. Roy, S. S., Karanjit, U., & Nilizadeh, S. (2022). A large-scale analysis of phishing websites hosted on free web hosting domains. arXiv preprint arXiv:2212.02563.

12. Jaggard, A. D., & Syverson, P. (2017). Onions in the crosshairs: When The Man really is out to get you. – Режим доступу: https://dl.acm.org/doi/abs/10.1145/3139550.3139553 (дата звернення 10.03.2025)

13. Biryukov, A., Pustogarov, I., & Weinmann, R. P. (2013). Trawling for tor hidden services: Detection, measurement, deanonymization. In 2013 IEEE Symposium on Security and Privacy (pp. 80-94). IEEE.

14. Jagerman, R., Sabee, W., Versluis, L., de Vos, M., & Pouwelse, J. (2014). The fifteen year struggle of decentralizing privacy-enhancing technology. arXiv preprint arXiv:1404.4818.

15. Haughey, H., Epiphaniou, G., Al-Khateeb, H., & Dehghantanha, A. (2018). Adaptive traffic fingerprinting for darknet threat intelligence. Cyber Threat Intelligence, 193-217.

16. Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148 (NIS 2 Directive) (Text with EEA relevance). Official Journal of the European Union, L 333, 27.12.2022, p. 80–152. – Режим доступу: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022L2555 (дата звернення 15.03.2025)

17. General Data Protection Regulation (GDPR): Регламент (ЄС) 2016/679 Європейського Парламенту і Ради від 27 квітня 2016 року про захист фізичних осіб у зв'язку з обробкою персональних даних і про вільне переміщення таких даних. [Офіційний переклад] / Верховна Рада України. – Режим доступу: https://zakon.rada.gov.ua/laws/show/984_008-16#Text (дата звернення 15.03.2025)

18. Computer Fraud and Abuse Act (CFAA). – Режим доступу: https://www.law.cornell.edu/uscode/text/18/1030 (дата звернення 15.03.2025)

19. Identity Theft and Assumption Deterrence Act. – Режим доступу: https://www.govinfo.gov/content/pkg/PLAW-105publ318/pdf/PLAW-105publ318.pdf (дата звернення 15.03.2025)

20. Federal Trade Commission (FTC). (2025a). Fighting Fraud and Protecting Consumers. – Режим доступу: https://www.ftc.gov/about-ftc/mission/enforcement-authority (дата звернення 15.03.2025)

21. Federal Trade Commission (FTC). (2025b). How to Recognize and Avoid Phishing Scams. – Режим доступу: https://consumer.ftc.gov/articles/how-recognize-and-avoid-phishing-scams (дата звернення 15.03.2025)

22. Electronic Communications Privacy Act (ECPA), 18 U.S.C. §§ 2510–2523 (1986). – Режим доступу: https://www.law.cornell.edu/uscode/text/18/part-I/chapter-119 (дата звернення 15.03.2025)

23. Кримінальний кодекс України. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2341-14 (дата звернення 15.03.2025)

24. Закон України «Про основні засади забезпечення кібербезпеки України» – Режим доступу: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення 15.03.2025)

25. Закон України «Про захист персональних даних» – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення 15.03.2025)

26. Закон України «Про електронну комерцію» – Режим доступу: https://zakon.rada.gov.ua/laws/show/675-19#Text (дата звернення 15.03.2025)

27. Andrea Venturi, Michele Colajanni, Marco Ramilli, Giorgio Valenziano Santangelo (2022). Classification of Web Phishing Kits for early detection by platform providers. – Режим доступу: https://arxiv.org/abs/2210.08273 (дата звернення 16.03.2025)

28. Abuadbba, A., Wang, S., Almashor, M., Ahmed, M. E., Gaire, R., Camtepe, S., & Nepal, S. (2022). Towards web phishing detection limitations and mitigation. arXiv preprint arXiv:2204.00985.

29. Tharani, J. S., & Arachchilage, N. A. (2020). Understanding phishers' strategies of mimicking uniform resource locators to leverage phishing attacks: A machine learning approach. Security and Privacy, 3(5), e120.

30. Barr-Smith, C., & Wright, J. (2020). Phishing With A Darknet: Imitation of Onion Services. eCrime Researchers Summit (APWG).

31. Yoon, C., Kim, K., Kim, Y., Shin, S., & Son, S. (2019). Doppelgängers on the dark web: A large-scale assessment on phishing hidden web services. In The World Wide Web Conference (pp. 2225-2235).

32. Güldenring, B., & Roth, V. (2024). Protecting onion service users against phishing. arXiv preprint arXiv:2408.07787.

33. Steinebach, M., Zenglein, S., & Brandl, K. (2021). Phishing detection on tor hidden services. Forensic Science International: Digital Investigation, 36, 301117.

34. Bergman, J., & Popov, O. B. (2023). Exploring dark web crawlers: a systematic literature review of dark web crawlers and their implementation. IEEE Access, 11, 35914-35933.

35. Researchers discovered a new phishing kit on the Dark Web. (2022). Security Magazine. – Режим доступу: https://www.securitymagazine.com/articles/100863-researchers-discovered-a-new-phishing-kit-on-the-dark-web (дата звернення 17.03.2025)

36. Klinger, E., & Starkweather, D. (2013). pHash: The open source perceptual hash library. – Режим доступу: http://www.phash. org/apps (дата звернення 17.03.2025)

37. Steinebach, M., Liu, H., & Yannikos, Y. (2012). Forbild: Efficient robust image hashing. In Media Watermarking, Security, and Forensics 2012 (Vol. 8303, pp. 195-202). SPIE.

38. Sara, U., Akter, M., & Uddin, M. S. (2019). Image quality assessment through FSIM, SSIM, MSE and PSNR–a comparative study. Journal of Computer and Communications, 7(3), 8-18.

39. Bram van Dooremaal, Pavlo Burda, Luca Allodi, Nicola Zannone (2021). Combining Text and Visual Features to Improve the Identification of Cloned Webpages for Early Phishing Detection

40. Requests: HTTP for Humans. – Режим доступу: https://requests.readthedocs.io/en/latest/# (дата звернення 20.03.2025)

41. difflib – Helpers for computing deltas. – Режим доступу: https://docs.python.org/uk/3/library/difflib.html (дата звернення 20.03.2025)

42. hashlib – Secure hashes and message digests. – Режим доступу: https://docs.python.org/3/library/hashlib.html (дата звернення 20.03.2025)

43. ImageHash 4.3.2. – Режим доступу: https://pypi.org/project/ImageHash/ (дата звернення 20.03.2025)

44. Open Source Computer Vision. – Режим доступу: https://docs.opencv.org/4.x/index.html (дата звернення 20.03.2025)

45. scikit-image’s documentation. – Режим доступу: https://scikit-image.org/docs/stable/ (дата звернення 20.03.2025)

46. Pillow (PIL Fork) 11.1.0 documentation. – Режим доступу: https://pillow.readthedocs.io/en/stable/# (дата звернення 20.03.2025)

47. io – Core tools for working with streams. – Режим доступу: https://docs.python.org/3/library/io.html (дата звернення 20.03.2025)

48. Beautiful Soup Documentation. – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення 20.03.2025)

49. urllib – URL handling modules. – Режим доступу: https://docs.python.org/3/library/urllib.html (дата звернення 20.03.2025)

50. scikit-learn: machine learning in Python. – Режим доступу: https://scikit-learn.org/stable/ (дата звернення 20.03.2025)

51. Марчук, Д. К., & Граф, М. С. (2023). Методи оцінки ефективності моделей виявлення об’єктів у комп’ютерному зорі. Вісник Херсонського національного технічного університету, (2 (85)), 181-186.

52. Buchyk S., Shutenko D., Toliupa S., (2022) Phishing Attacks Detection. IX International Scientific Conference “Information Technology and Implementation" (IT&I-2022), Workshop Proceedings, Kyiv, Ukraine, November 30 - December 02, 2022., Kyiv, Ukraine, pp. 193–201. – Режим доступу: https://ceur-ws.org/Vol-3384/Short_7.pdf (дата звернення 01.04.2025).

53. Toliupa S., Buchyk S., Shabanova A., Buchyk O. (2023) The Method for Determining the Degree of Suspiciousness of a Phishing Url. X International Scientific Conference "Information Technology and Implementation" (IT&I-2023), Workshop Proceedings (IT&I-WS 2023), Kyiv, Ukraine, November 20-21, 2023, pp. 239-247. – Режим доступу: https://ceur-ws.org/Vol-3646/Short_5.pdf (дата звернення 01.04.2025).

54. Бучик С.С., Кисельова А.П. Методи виявлення фішингу в Dark Web. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем (PCSICS)». – Київ 2025. – С. 17-20

ДОДАТОК А
СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Копія наукової публікації

Бучик С.С., Кисельова А.П. Методи виявлення фішингу в Dark Web. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем (PCSICS)». – Київ 2025. – С. 17-20

Методи виявлення фішингу в Dark Web

С. С. Бучик[1] , А. П. Кисельова[2]

1. Кафедра кібербезпеки та захисту інформації
Київський національний університет імені Тараса Шевченка, УКРАЇНА, м. Київ, вул. Б. Гаврилишина, 24
E-mail: buchyk@knu.ua

2. Кафедра кібербезпеки та захисту інформації
Київський національний університет імені Тараса Шевченка, УКРАЇНА, м. Київ, вул. Б. Гаврилишина, 24
E-mail: kyselovaa@fit.knu.ua

Abstractum – the article analyses the basic concepts related to phishing attacks and describes the technical aspects of creating phishing websites on the Dark Web. This article presents an overview of modern methods of detecting phishing pages in the .onion domain zone. The expediency of these methods application as a comprehensive system for detecting phishing in hidden services is substantiated.

Ключові слова: фішинг, Dark Web, onion, імперсонейтинг, шахрайство.

Вступ

Dark Web (прихована мережа) – це частина мережі Інтернет, яка не індексується традиційними пошуковими системами і доступна лише за допомогою спеціального програмного забезпечення, такого як браузер Tor (The Onion Router), який надає підвищену анонімність користувачам за рахунок багатошарового шифрування та маршрутизації трафіку через розподілену мережу вузлів, на відміну від загальнодоступної мережі (Clear Web), де вебсайти є загальнодоступними та контролюються пошуковими системами та регуляторними органами.

Домен .onion, доступний виключно через мережу Tor, забезпечує безпечне та приватне середовище для користувачів, однак ця властивість Dark Web використовується зловмисниками для створення шахрайських вебсайтів, які імітують легітимні сервіси. На відміну від Clear Web, де передові методи виявлення фішингу постійно вдосконалюються, Dark Web представляє унікальні проблеми.

Традиційні механізми виявлення фішингу, які покладаються на репутацію домену, сертифікати SSL і моніторинг пошукової системи, не є ефективними в екосистемі .onion.

На це є кілька причин:

onion домени не покладаються на довірені центри сертифікації для перевірки;

вузли в мережі Tor працюють на піринговій основі і не існує жодного керівного органу, який би перевіряв законні сервіси або вносив шахрайські в чорний список.

Також складнощі виникають через часті зміни адрес onion, тому що більшість сервісів регулярно вдаються до цього для підвищення безпеки, що ускладнює запам’ятовування та перевірку автентичності користувачами. Враховуючи перелічені особливості мережі Tor, постає необхідність розробки альтернативних методів та інструментів, здатних ідентифікувати фішингові загрози на основі властивих характеристик прихованих служб.

Основний текст

Dark Web, завдяки своїй анонімній природі, окрім основного призначення – забезпечення конфіденційності та свободи слова – стала осередком незаконної діяльності.

Dark Web є центром шахрайських схем, включаючи фішингові атаки, які мають на меті видати себе за легітимні організації, щоб обманом змусити людей розкрити конфіденційну інформацію. Кіберзлочинці вдаються до імперсонейтингу, використовуючи анонімність сайтів .onion для створення клонів надійних ресурсів – ринків, банківських сервісів і криптовалютних бірж тощо. Отримані внаслідок фішингу викрадені персональні, фінансові, медичні, корпоративні, автентифікаційні дані в подальшому можуть бути використані для продажу на хакерських маркетплейсах, створення фальшивої особистості (identity theft), корпоративного шпіонажу, вимагання грошового викупу (ransomware), фінансового шахрайства тощо (Рис 1).

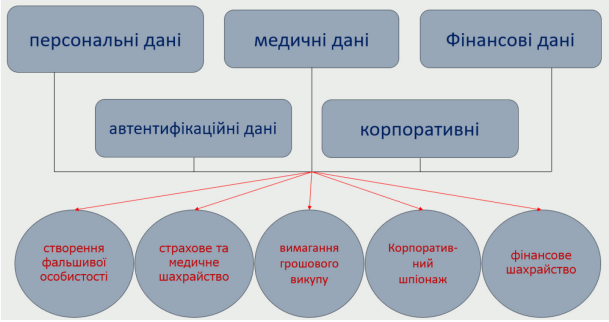

Рисунок 1 - Наслідки фішингових атак

Технічні аспекти створення фішингових сайтів

Фішингові атаки в мережі Tor особливо небезпечні через складну природу onion адрес і відсутність традиційних механізмів перевірки.

Onion адреси – це довгі рядки символів (16 в старій або 56 в новій версії алгоритму генерації), отримані з криптографічних ключів, що ускладнює запам’ятовування або розпізнавання користувачами легітимних сервісів. Цим користуються кіберзлочинці, створюючи шахрайські сервіси, які імітують надійні платформи, обманом змушуючи користувачів надавати облікові дані, криптовалютні транзакції або інші конфіденційні дані.

Зловмисники використовують різні методи для імітації сервісів, що ускладнюють користувачам розрізнення справжніх і підроблених вебсайтів.

1. Підробка адрес onion.

Оскільки адреси onion генеруються як хешовані версії криптографічних ключів, вони не можуть бути прочитані людиною, як традиційні доменні імена. Зловмисники

користуються цим, генеруючи схожі на вигляд onion адреси за допомогою генерації адрес методом грубої сили або за допомогою інструментів, які дозволяють створювати кілька onion адрес, поки не буде знайдено візуально схожу [1]. Приклад спеціально згенерованого префіксу адреси показаний на Рис. 2

Ця методика схожа на тайпсквоттинг у відкритому Інтернеті, але її значно складніше виявити через рандомізовану природу адрес onion [2].

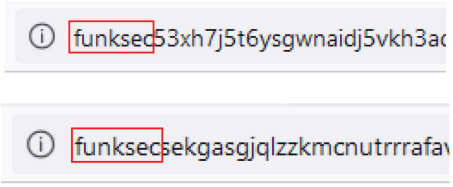

Рисунок 2 – Навмисно створені візуально схожі адреси .onion

2. Клонування легітимних вебсайтів.

Зловмисники копіюють весь інтерфейс легітимного сервісу, включаючи HTML, CSS та JavaScript, і розміщують його на фальшивій адресі. Це схоже на традиційні фішингові атаки в інтернеті, коли шахрайські вебсайти точно імітують справжні (просте клонування). Запити та дані замість автентичного серверу йдуть напряму на сервер зловмисників. На Рис. 3 показана архітектура простого клонування.

У багатьох випадках користувачі не помічають різниці, оскільки шахрайські сайти часто не мають чіткого брендингу або впізнаваних доменних імен.

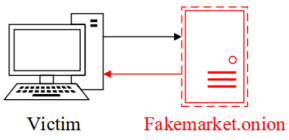

Рисунок 3 – архітектура простого клонування

3. Атаки зі зворотним проксі-сервером.

Більш складні фішингові кампанії використовують зворотні проксі-сервери, які виступають посередниками між жертвами та реальним сервісом (Рис. 4). Зворотний проксі пересилає вхідні дані жертви (наприклад, імена користувачів, паролі або дані криптовалютних гаманців) до легітимного сервісу, викрадаючи дані під час передачі [3].

Цей метод особливо ефективний, оскільки користувачі бачать вміст справжнього сайту в режимі реального часу, що робить майже неможливим виявлення фішингової атаки.

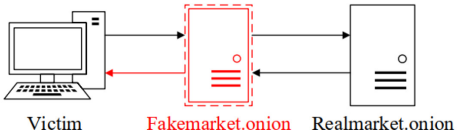

Рисунок 4 – клонування на базі проксі-сервера

Складність клонування може бути різною – від простого копіювання зовнішнього вигляду сайту (просте клонування) до складної реплікації функціональних можливостей як зовнішнього, так і внутрішнього інтерфейсу (комплексне клонування) [4].

Огляд існуючих рішень виявлення фішингу в домені .onion

У сфері виявлення фішингових атак у домені .onion дослідники пропонують різні підходи, кожен із яких орієнтований на окремі аспекти виявлення копій onion-сервісів.

Серед найвизначніших робіт у цій галузі можна виділити дослідження Барр-Сміта, Гюльдерінга та Штейнебаха, які застосовують різні стратегії боротьби з фішинговими сайтами.

У роботі Барр-Сміта і Райта [5] основна увага зосереджена на аналізі onion-доменів із використанням підходу тайпсквоттингу – пошуку доменів, які мають мінімальні відмінності у назві від популярних ресурсів і потенційно є фішинговими копіями.

Дослідження Гюльденрінга [6] пропонує підхід захисту безпосередньо для користувачів Tor, шляхом застосування системи візуалізації onion-адрес на основі хеш-функцій.

Найбільш комплексний підхід запропоновано в роботі Штейнебаха [7], де розроблено систему автоматизованого виявлення фішингових сайтів шляхом глибокого аналізу контенту onion-сервісів на основі тексту, зображень та URL-адрес.

Методи виявлення фішингу в домені .onion

Базуючись на огляді існуючих досліджень, було визначено список методів, які слугуватимуть для визначення фішингових посилань в домені .onion. Нижче наведений перелік обраних методів.

1. Аналіз заголовків HTTP.

HTTP-заголовки є важливою частиною вебкомунікації, оскільки вони передають метадані між клієнтом (браузером) і сервером під час HTTP-запитів і відповідей. Ці заголовки включають різні технічні параметри, такі як тип сервера, тип контенту, політики кешування, файли cookie та налаштування безпеки. У контексті виявлення фішингу в Dark Web аналіз HTTP-заголовків є ефективним методом виявлення імітацій сайтів з доменом .onion, оскільки фішингові клони часто копіюють не тільки візуальні аспекти вебсайту, але й намагаються імітувати конфігурацію сервера, щоб виглядати більш автентичними [7].

2. Подібність HTML-коду.

Одним з найпоширеніших методів, який зловмисники використовують для створення фішингових сайтів, особливо в Dark Web, є клонування структур легальних сайтів. Ці клони часто повністю копіюють HTML-код цільового сервісу .onion, щоб ввести в оману користувачів, представляючи візуально ідентичну і функціонально схожу копію.

Метод схожості HTML-коду може містити в собі два підходи, в залежності від потреб користувача: точне виявлення дублікатів за допомогою алгоритмів хешування (наприклад, SHA-256) та виявлення часткової схожості, використовуючи алгоритми приблизного порівняння тексту, наприклад Ratcliff/Obershelp [4, 8].

3. Подібність текстового контенту.

Фішингові сторінки в Dark Web часто дублюють текстовий контент легітимних сайтів. Заголовки, описи, списки товарів, спливаючі вікна та інші текстові елементи часто копіюються майже без змін, щоб зберегти ілюзію автентичності. Навіть якщо HTML-код або зображення трохи змінені, основний текст часто залишається незмінним, що робить аналіз текстового контенту ефективним методом виявлення фішингу. Цей метод аналізу фокусується лише на змістовному тексті, видимому користувачеві, ігноруючи код, скрипти та метадані.

Виявлення текстової схожості складається з двох основних етапів:

- векторизація тексту, внаслідок якої текстовий вміст перетворюється на числові дані, які можна математично порівняти;

- вимірювання схожості між двома текстовими векторами за допомогою алгоритмів косинусної подібності, Жаккарової подібності, евклідової відстані [9].

4. Аналіз схожості зображень.

Зображення, такі як логотипи або банери тощо, можна порівнювати за допомогою декількох методів, щоб визначити міру їх подібності:

- перцептивне хешування (pHash) призначене для порівняння зображень на основі їх візуального сприйняття, а не точного двійкового представлення. Цей алгоритм генерує схожі хеші для зображень, які візуально виглядають однаково, навіть якщо вони були змінені через зміну розміру, незначні зміни кольору, артефакти стиснення або шум. pHash спочатку змінює розмір зображення до стандартного розміру і перетворює його у відтінки сірого, щоб усунути вплив кольорів. Потім застосовується дискретне косинусне перетворення (ДКП) для виявлення частотних патернів на зображенні. Вибираються найбільш значущі коефіцієнти ДКП і перетворюються на бінарний хеш, порівнюючи їх з медіанним значенням набору. Потім визначається схожість між зображеннями шляхом обчислення відстані Хеммінга між їхніми хешами [7].

- індекс структурної подібності (SSIM) ділить зображення на невеликі блоки і порівнює відповідні ділянки, щоб виміряти яскравість, дисперсію контрасту і схожість патернів. Цей метод виводить оцінку від -1 до 1, де 1 означає ідентичні зображення. SSIM особливо корисний для виявлення фішингових сайтів, які вносять незначні зміни в зображення, наприклад, регулюють яскравість або контрастність, зберігаючи при цьому основну структуру скопійованої графіки. Цей метод дозволяє виявляти зображення, які навмисно змінені, щоб обійти просте виявлення, але при цьому зберігають достатню візуальну схожість для обману користувачів [10].

- алгоритм ORB (Oriented FAST and Rotated BRIEF) – це метод зіставлення на основі особливостей, який визначає і порівнює характерні ключові точки на двох зображеннях, такі як кути, краї і текстури, які залишаються незмінними навіть під час

таких перетворень, як обертання або масштабування. Метод починається з алгоритму FAST для виявлення ключових точок і призначення орієнтації, щоб зробити їх інваріантними до повороту. Потім він використовує дескриптори BRIEF для кодування локальних патернів навколо цих ключових точок у двійкові рядки. Щоб оцінити схожість, ORB обчислює відстань Хеммінга між дескрипторами з різних зображень і зіставляє їх з відповідними ключовими точками. Цей метод особливо ефективний, коли зловмисники вносять геометричні спотворення або ледь помітні зміни в скопійовані зображення, щоб уникнути виявлення. ORB відіграє вирішальну роль у виявленні зображень, які вже не є ідентичними по пікселям, але мають ідентичні структури та шаблони на рівні елементів [4].

У поєднанні ці три методи – pHash, SSIM та ORB – забезпечують комплексну основу для виявлення фішингу на основі зображень у Dark Web. У той час як pHash фіксує загальну візуальну схожість, SSIM оцінює структурну схожість, що має відношення до людського сприйняття, а ORB забезпечує стійкість до геометричних перетворень.

5. Порівняння префіксів URL-адрес .onion

Оскільки адреси .onion генеруються як криптографічні хеші, а не як доменні імена, які читаються людиною, користувачі часто покладаються на розпізнавання знайомих префіксів при перевірці автентичності сервісу. Зловмисники користуються цим, застосовуючи метод грубого перебору адрес – метод, який генерує кілька адрес, доки не буде знайдено візуально схожу. Цей метод дозволяє створювати фішингові сайти, які виглядають майже ідентично легальним сервісам, що збільшує ймовірність обману користувачів, які нічого не підозрюють [5].

Щоб виявити цю форму атаки, найдоцільніше буде взяти для порівняння перші шість символів адрес .onion і визначити, скільки з них збігаються. Визначити поріг збігу (рекомендується 35%), і якщо значення схожості перевищуватиме його, тоді адреси позначаються як підозріло схожі. Цей поріг заснований на припущенні [7], що зловмисник, який намагається видати себе за службу, згенерує велику кількість адрес, щоб знайти одну зі збігом префікса, тим самим збільшуючи ймовірність введення в оману користувачів, які лише побіжно подивилися на домен.

Отже, якщо використовуючи всі методи виявлення фішингу, зазначені вище, то буде створена багаторівнева модель виявлення, яка інтегрує технічні, текстові та візуальні індикатори фішингу (Рис. 5).

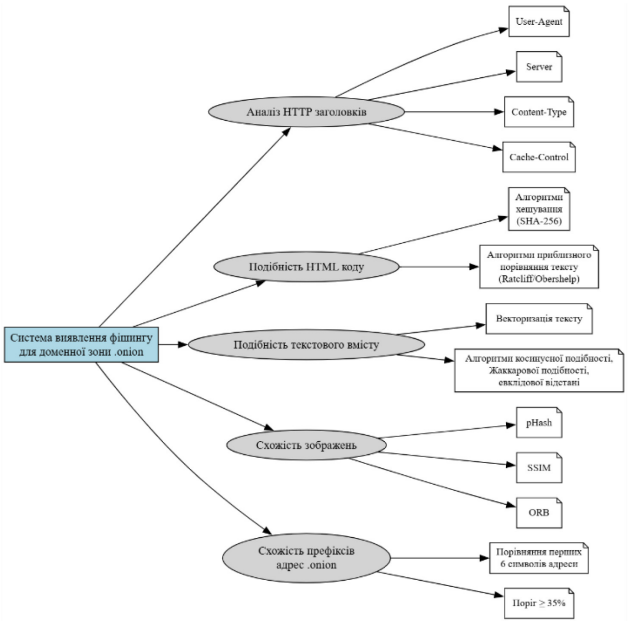

Рисунок 5 – Узагальнена схема системи виявлення фішингу

Таке поєднання методів підвищує точність виявлення та стійкість до тактик введення в оману користувачів, що використовуються зловмисниками в Dark Web [11].

Висновок

Підбиваючи підсумки, у статті було проведено аналіз основних понять, пов’язаних з фішинговими атаками в Dark Web, надано опис технічних аспектів створення фішингових вебсайтів у домені .onion. Особливу увагу було приділено вивченню методів імітації та клонування onion-сервісів, які становлять значну загрозу для користувачів прихованих сервісів через складність перевірки автентичності ресурсів та високий рівень анонімності.

У статті надано детальний огляд методів виявлення фішингових вебсайтів у домені .onion, що в перспективі дає змогу побудувати багаторівневу модель виявлення, яка інтегрує технічні, текстові та візуальні індикатори фішингу. Розроблена методологія пропонує надійну основу для безпеки прихованих сервісів з потенційним застосуванням в автоматизованому моніторингу фішингу, дослідженнях кібербезпеки та розвідці загроз в Dark Web.

Література

1. Biryukov, A., Pustogarov, I., & Weinmann, R. P. (2013, May). Trawling for tor hidden services: Detection, measurement, deanonymization. In 2013 IEEE Symposium on Security and Privacy (pp. 80-94). IEEE.

2. Jagerman, R., Sabee, W., Versluis, L., de Vos, M., & Pouwelse, J. (2014). The fifteen year struggle of decentralizing privacy-enhancing technology. arXiv preprint arXiv:1404.4818.

3. Haughey, H., Epiphaniou, G., Al-Khateeb, H., & Dehghantanha, A. (2018). Adaptive traffic fingerprinting for darknet threat intelligence. Cyber Threat Intelligence, 193-217.

4. Gadient, P., Gerig, P., Nierstrasz, O., & Ghafari, M. (2021). Phish What You Wish. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS) (pp. 1048-1059). IEEE.

5. Barr-Smith, C., & Wright, J. (2020). Phishing With A Darknet: Imitation of Onion Services. eCrime Researchers Summit (APWG).

6. Güldenring, B., & Roth, V. (2024). Protecting onion service users against phishing. arXiv preprint arXiv:2408.07787.

7. Steinebach, M., Zenglein, S., & Brandl, K. (2021). Phishing detection on tor hidden services. Forensic Science International: Digital Investigation, 36, 301117.

8. Wong, A., Abuadbba, A., Almashor, M., & Kanhere, S. (2022). PhishClone: measuring the efficacy of cloning evasion attacks. arXiv preprint arXiv:2209.01582.

9. Bergman, J., & Popov, O. B. (2023). Exploring dark web crawlers: a systematic literature review of dark web crawlers and their implementation. IEEE Access, 11, 35914-35933.

10. Sara, U., Akter, M., & Uddin, M. S. (2019). Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study. Journal of Computer and Communications, 7(3), 8-18.

11. Bram van Dooremaal, Pavlo Burda, Luca Allodi, Nicola Zannone (2021). Combining Text and Visual Features to Improve the Identification of Cloned Webpages for Early Phishing Detection

ДОДАТОК Б
ЛІСТИНГ ПРОГРАМНОГО КОДУ

Файл «phishing_detector.py»

```
import requests
import difflib
import hashlib
import numpy as np
import imagehash
import cv2
from skimage.metrics import structural_similarity as ssim
from PIL import Image
from io import BytesIO
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def get_http_headers(url):
    proxies = {
        "http": "socks5h://127.0.0.1:9050",
        "https": "socks5h://127.0.0.1:9050"
    }
    try:
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()
        return dict(response.headers)
    except requests.RequestException as e:
        print(f"Error fetching headers from {url}: {e}")
        return None

def get_html_content(url):
```

```
    proxies = {
        "http": "socks5h://127.0.0.1:9050",
        "https": "socks5h://127.0.0.1:9050"
    }

    try:
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()
        return response.text
    except requests.RequestException as e:
        print(f"Error fetching HTML from {url}: {e}")
        return None

def hash_text(text):
    return hashlib.sha256(text.encode()).hexdigest()

def compare_text_hashes(text1, text2):
    return hash_text(text1) == hash_text(text2)

def compare_html(html1, html2):
    if not html1 or not html2:
        return 0
    return difflib.SequenceMatcher(None, html1, html2).ratio()

def compare_headers(headers1, headers2):
    if not headers1 or not headers2:
        return (0.0, True)

    mismatched_headers = []

    h1 = {k.lower(): v.strip() for k, v in headers1.items()}
    h2 = {k.lower(): v.strip() for k, v in headers2.items()}

    for key in h1:
        if key not in h2 or h1[key] != h2[key]:
            mismatched_headers.append(key)
```

```
    if mismatched_headers:
        print(" Mismatched or missing headers detected:")
        for h in mismatched_headers:
            print(f" - {h}")

    common_keys = set(headers1.keys()) & set(headers2.keys())
    same_values = sum(1 for key in common_keys if headers1[key] ==
headers2[key])

    if common_keys:
        similarity = same_values / len(common_keys)
        return (similarity, similarity < 0.7)
    else:
        return (0.0, True)

def extract_text_from_html(html):
    soup = BeautifulSoup(html, "html.parser")
    text = ' '.join([p.get_text() for p in soup.find_all(['p', 'h1',
'h2', 'h3'])])
    return text

def compute_text_similarity(html1, html2):
    text1 = extract_text_from_html(html1)
    text2 = extract_text_from_html(html2)

    if not text1.strip() or not text2.strip():
        print(" One or both pages have no usable text content.")
        return 0.0

    try:
        vectorizer = CountVectorizer().fit([text1, text2])
        vectors = vectorizer.transform([text1, text2])
        cos_sim = cosine_similarity(vectors[0], vectors[1])[0][0]
        return cos_sim
    except ValueError as e:
```

```
        print(f"Vectorization failed: {e}")
        return 0.0


def get_images(url):
    try:
        proxies = {
            "http": "socks5h://127.0.0.1:9050",
            "https": "socks5h://127.0.0.1:9050"
        }
        response = requests.get(url, proxies=proxies, timeout=30)
        response.raise_for_status()

        content_type = response.headers.get("Content-Type", "")
        if not content_type.startswith("text/html"):
            print(f"Skipping non-HTML content from {url} (Content-Type:
{content_type})")
            return None, None

        soup = BeautifulSoup(response.text, "html.parser")
        img_urls = [img["src"] for img in soup.find_all("img") if "src"
in img.attrs]

        if not img_urls:
            print(f"No images found on {url}")
            return None, None

        img_url = urljoin(url, img_urls[0])
        print(f"Fetching image: {img_url}")

        img_response    =    requests.get(img_url,    proxies=proxies,
timeout=30)
        img_response.raise_for_status()

        img_content_type = img_response.headers.get("Content-Type", "")
        if not img_content_type.startswith("image/"):
```

```
            print(f"Skipping non-image content from {img_url} (Content-
Type: {img_content_type})")
            return None, None

        image = Image.open(BytesIO(img_response.content))
        return image, imagehash.phash(image)
    except Exception as e:
        print(f"Error fetching images from {url}: {e}")
        return None, None

def compare_images_ssim(img1, img2):
    size = (300, 300)
    img1_resized = img1.resize(size)
    img2_resized = img2.resize(size)

    img1_gray= cv2.cvtColor(np.array(img1_resized), cv2.COLOR_RGB2GRAY)
    img2_gray= cv2.cvtColor(np.array(img2_resized), cv2.COLOR_RGB2GRAY)

    similarity, _ = ssim(img1_gray, img2_gray, full=True)
    return similarity

def compare_images_phash(hash1, hash2):
    return 1 - (hash1 - hash2) / len(hash1.hash) ** 2 if hash1 and hash2
else 0

def compare_images_orb(img1, img2):
    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(np.array(img1), None)
    kp2, des2 = orb.detectAndCompute(np.array(img2), None)
    if des1 is None or des2 is None:
        return 0
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    return len(matches) / max(len(kp1), len(kp2))

def compare_onion_addresses(addr1, addr2):
```

```
    min_length = min(len(addr1), len(addr2), 6)
    common_prefix = sum(1 for i in range(min_length) if addr1[i] ==
addr2[i])
    return common_prefix / 6 >= 0.35

def phishing_probability(url1, url2):
    print("Getting HTTP Headers...")
    headers1, headers2 = get_http_headers(url1), get_http_headers(url2)
    print("Getting HTML Content...")
    html1, html2 = get_html_content(url1), get_html_content(url2)
    print("Getting images...")
    img1, img1_phash = get_images(url1)
    img2, img2_phash = get_images(url2)

    if headers1 is None or headers2 is None or html1 is None or html2
is None:
        print("Error: One or both sites could not be accessed.")
        return 0

    header_similarity = compare_headers(headers1, headers2)
    html_similarity = compare_html(html1, html2)
    text_hash_similarity = compare_text_hashes(html1, html2)
    text_vector_similarity = compute_text_similarity(html1, html2)
    address_similarity                                              =
compare_onion_addresses(onion_url1.split('//')[-1],
onion_url2.split('//')[-1])
    print("HTTP Headers Similarity:", header_similarity[0])
    print("HTML Similarity:", html_similarity)
    print("HTML Hash Similarity:", text_hash_similarity)
    print("Cosinus Text Similarity:", text_vector_similarity)
    print("URL Prefixes Similarity:", address_similarity)

    image_similarity = 0
    if img1 and img2:
        ssim_sim = compare_images_ssim(img1, img2)
        phash_sim = compare_images_phash(img1_phash, img2_phash)
```

```
        orb_sim = compare_images_orb(img1, img2)
        image_similarity = (ssim_sim + phash_sim + orb_sim) / 3

        print("SSIM:", ssim_sim)
        print("PHash:", phash_sim)
        print("ORB:", orb_sim)
        print("Image Similarity average:", image_similarity)

    weight_headers = 0.1
    weight_html = 0.2
    weight_text_hash = 0.2
    weight_text_vector = 0.2
    weight_images = 0.2
    weight_address = 0.1

    probability = (
        header_similarity[1] * weight_headers +
        html_similarity * weight_html +
        text_hash_similarity * weight_text_hash +
        text_vector_similarity * weight_text_vector +
        image_similarity * weight_images +
        address_similarity * weight_address
    )
    print("Phishing probability evaluation...")
    return probability


if __name__ == "__main__":
    onion_url1 = input("Enter 1st .onion link: ")
    onion_url2 = input("Enter 2nd .onion link: ")

    probability = phishing_probability(onion_url1, onion_url2)
    print(f"Phishing imitation probability: {probability:.2%}")
```

Файл «evaluate_phishing_detector.py»

```
from phishing_detector import phishing_probability
threshold = 0.5

# test dataset
test_data = [
    # phishing links
    {"url1":    "http://arkanabb66ee4nsdji6labwqe...hod7utagad.onion/",
"url2":
"http://ransomwvbabemdnwl7lzgeenyfmmhs...kvapsia76vttzyd.onion/",
"label": "phishing"},
    {"url1":
"http://rhysidafohrhyy2aszi7bm32tn...opcxkdfxhi4tidsg7cad.onion/",
"url2":
"http://rhysidafc6lm7qa2mkiukbezh7zut...2audkymscjm6yegad.onion/",
"label": "phishing"},
    {"url1":
"http://lynxblogxstgzsarfyk2pvhdv45iggh...sipzeoduruz3xwqd.onion/leaks
",                                                          "url2":
"http://lynxblogco7r37jt7p5wrmfxzqze7ghxw6r...uacwotciyd.onion/leaks",
"label": "phishing"},
    {"url1":
"http://mbrlkbtq5jonaqkurjwmxftytyn2et...kknndqwae6byd.onion/", "url2":
"http://k7kg3jqxang3wh7hnmaiokchk7qoebupfgoi...mjpzwupwtj25yd.onion/",
"label": "phishing"},
    {"url1":
"http://basheqtvzqwz4vp6ks5lm2ocq7i6tozqgf...4ezmsy4bkpshhyd.onion/",
"url2":
"http://basherq53eniermxovo3bkduw5q...qictfmamgvmzovykyqd.onion/",
"label": "phishing"},
    {"url1":
"http://weg7sdx54bevnvulapqu6bp...tegbmnhkbpqz637f2yd.onion/",  "url2":
"http://c7jpc6h2ccrdwmhofuij7kz6sr2fg2...23cf7m2e5hvqid.onion/",
"label": "phishing"},
```

```
    # legitimate links
    {"url1":
"http://arkanabb66ee4nsdji6la2bu...f3rxrv6vhiehod7utagad.onion/",
"url2":
"http://mbrlkbtq5jonaqkurjwmxftytyn2eth...kkknndqwae6byd.onion/",
"label": "legitimate"},
    {"url1":
"http://basherq53eniermxovo3bkduw5qq...mamgvmzovykyqd.onion/",  "url2":
"http://lynxblogxstgzsarfyk2pvhdv4...msipzeoduruz3xwqd.onion/leaks",
"label": "legitimate"},
    {"url1":
"http://rhysidafc6lm7qa2mkiukbezh7zu...udkymscjm6yegad.onion/", "url2":
"http://arkanabb66ee4nsdji6la2bu6bwqe3db...vhiehod7utagad.onion/",
"label": "legitimate"},
    {"url1":
"http://k7kg3jqxang3wh7hnmaiok...fgoik6rha6mjpzwupwtj25yd.onion/",
"url2":
"http://c7jpc6h2ccrdwmhofuij7kz...tbvvqy4fse23cf7m2e5hvqid.onion/",
"label": "legitimate"},
    {"url1":
"http://weg7sdx54bevnvulapqu6bp...s23tegbmnhkbpqz637f2yd.onion/",
"url2":
"http://lynxblogco7r37jt7p5wrmfxz...ihzkqc455qluacwotciyd.onion/leaks"
, "label": "legitimate"},
]

TP = FP = TN = FN = 0

for item in test_data:
    print("=" * 60)
    print(f" Comparing:\n{item['url1']}  ↔  {item['url2']}")

    try:
        score = phishing_probability(item["url1"], item["url2"])
    except Exception as e:
```

```
        print(f" Error during phishing_probability: {e}")
        continue

    is_phishing = score >= threshold
    print(f"  Score:  {score:.3f}  →  {'Phishing'  if  is_phishing  else
'Legitimate'} (expected: {item['label']})")

    if is_phishing and item["label"] == "phishing":
        TP += 1
    elif is_phishing and item["label"] == "legitimate":
        FP += 1
    elif not is_phishing and item["label"] == "legitimate":
        TN += 1
    elif not is_phishing and item["label"] == "phishing":
        FN += 1

#metrics
precision = TP / (TP + FP) if (TP + FP) else 0
recall = TP / (TP + FN) if (TP + FN) else 0
f1 = 2 * precision * recall / (precision + recall) if (precision +
recall) else 0
accuracy = (TP + TN) / (TP + FP + TN + FN) if (TP + FP + TN + FN) else
0

print("\n===  Evaluation Results ===")
print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
print(f"Precision: {precision:.2f}")
print(f"Recall:    {recall:.2f}")
print(f"F1 Score:  {f1:.2f}")
print(f"Accuracy:  {accuracy:.2f}")
```