Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка інформаційної системи виявлення шкідливих веб-ресурсів з використанням технологій машинного навчання»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Козлов Є.С.
(прізвище та ініціали)

Керівник к.т.н., доц. Ульяновська Ю. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри кібербезпеки та інформаційних технологій
(посада)

к.т.н., доц. Савченко Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Козлов Є.С. Розробка інформаційної системи виявлення шкідливих веб-ресурсів з використанням технологій машинного навчання.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єкт дослідження: процес виявлення шкідливих веб-ресурсів у мережі Інтернет. Предмет дослідження: методи та алгоритми машинного навчання для класифікації URL-адрес.

Мета роботи: розробка системи автоматизованого виявлення шкідливих веб-ресурсів із використанням методів машинного навчання для підвищення рівня інформаційної безпеки.

Дипломна робота присвячена розробці системи автоматизованого виявлення шкідливих веб-ресурсів із використанням алгоритмів машинного навчання. У роботі проведено аналіз основних загроз інформаційної безпеки у веб-середовищі, таких як фішингові атаки, шкідливе програмне забезпечення та небезпечні URL-адреси. Розглянуто методи виявлення загроз, зокрема лексичний аналіз, аналіз характеристик хосту та поведінкових ознак.

Для реалізації системи було використано бібліотеку Scikit-Learn, яка забезпечує простоту використання, широкий спектр алгоритмів для аналізу даних та підтримку моделей класифікації. Запропонований алгоритм успішно реалізований та протестований на реальних даних, що підтвердило його ефективність у виявленні потенційно шкідливих ресурсів.

Практична цінність роботи полягає у створенні автоматизованої системи для виявлення шкідливих веб-ресурсів, яка може бути використана у корпоративних системах кібербезпеки, для захисту мереж від фішингових атак.

Ключові слова: інформаційна безпека, машинне навчання, URL-адреси, Random Forest, Decision Tree, Scikit-Learn.

ABSTRACT

Kozlov Ye.S. Development of an information system for detecting malicious web resources using machine learning technologies.

Bachelor's qualification work for obtaining a bachelor's degree in specialty 122 "Computer Science." – University of Customs and Finance, Dnipro, 2025.

The object of the study is the process of detecting malicious web resources on the Internet. The subject of the study is machine learning methods and algorithms for URL classification.

The aim of the work is to develop an automated malicious web resource detection system using machine learning methods to improve information security.

This thesis focuses on the development of an automated system for detecting malicious web resources using machine learning algorithms. The research includes an analysis of key information security threats in the web environment, such as phishing attacks, malware, and dangerous URLs. Various threat detection methods are considered, including lexical analysis, host characteristics analysis, and behavioral feature analysis.

The Scikit-Learn library was used for the system's implementation due to its simplicity, wide range of data analysis algorithms, and support for classification models. The proposed algorithm was successfully implemented and tested on real data, confirming its effectiveness in detecting potentially harmful resources.

The practical significance of the work lies in the development of an automated system for detecting malicious web resources that can be used in corporate cybersecurity systems to protect networks from phishing attacks.

Keywords: information security, machine learning, URL addresses, Random Forest, Decision Tree, Scikit-Learn.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах веб-додатки завжди відкриті для широкого кола користувачів, і сьогодні важко уявити організацію, підприємство чи бізнес, які не мали б власного веб-ресурсу або не використовували б інтернет-технології. На відміну від внутрішніх мережевих програм, веб-браузер доступний для всіх, хто має підключення до інтернету, включаючи зловмисників. На жаль, розробники часто нехтують принципами безпеки веб-додатків. Команди здебільшого настільки зосереджені на функціональних аспектах, таких як програмування, дизайн інтерфейсу та зручність використання, що забувають приділити увагу стабільності та захищеності програмного продукту.

Стрімкий розвиток веб-технологій та поширення Інтернету значно збільшує кількість загроз для інформаційної безпеки, зокрема через наявність шкідливих веб-ресурсів, фішингових атак та поширення зловмисного програмного забезпечення. Традиційні методи виявлення загроз мають обмежену ефективність через статичність підходу та нездатність ідентифікувати ще не виявлені загрози. Це створює потребу у використанні сучасних методів аналізу даних, зокрема алгоритмів машинного навчання, які здатні автоматизувати процес ідентифікації небезпечних ресурсів шляхом аналізу структурних, лексичних та поведінкових ознак URL-адрес.

Метод виявлення шкідливих URL-адрес на основі машинного навчання (ML) є перспективним підходом для вирішення зазначених проблем. Він базується на використанні алгоритмів штучного інтелекту для автоматизованої класифікації веб-ресурсів як шкідливих або безпечних. Процес включає збір даних, виділення ключових ознак URL (лексичних, хостових, поведінкових), навчання моделей на основі цих ознак, а також тестування та подальше використання моделі для аналізу нових веб-адрес.

Серед алгоритмів машинного навчання, що застосовуються для цієї задачі, поширеними є логістична регресія, дерева рішень, Random Forest, SVM

та глибокі нейронні мережі. Вони дозволяють знаходити приховані закономірності у структурі URL-адрес та поведінці хостів, що ускладнює обходження системи кіберзахисту.

З урахуванням зазначеного вище тема кваліфікаційної роботи «Розробка інформаційної системи виявлення шкідливих веб-ресурсів з використанням технологій машинного навчання» є актуальною.

Інноваційність дослідження базується на використанні алгоритмів машинного навчання для автоматизованого виявлення шкідливих веб-ресурсів. Запропонований підхід ґрунтується на застосуванні ансамблевих методів: Random Forest та Decision Tree для підвищення точності класифікації та зниження кількості невірних результатів.

Мета роботи – автоматизація виявлення шкідливих веб-ресурсів із використанням методів машинного навчання для підвищення рівня інформаційної безпеки та зниження ризиків для користувачів мережі Інтернет.

Методи дослідження – методи машинного навчання, розробки програмного забезпечення, методи теорії інформації, методи обробки та аналізу даних.

У відповідності до поставленої мети у кваліфікаційній роботі вирішувались наступні завдання:

1. Провести аналіз наукових джерел щодо методів виявлення шкідливих веб-ресурсів.
2. Визначити критерії класифікації небезпечних URL-адрес та обґрунтувати вибір ключових ознак для аналізу.
3. Обрати платформу для збору навчальних даних, що забезпечить актуальність та якість інформації.
4. Обрати бібліотеку машинного навчання для реалізації системи.
5. Реалізувати програмний модуль для автоматизованої класифікації шкідливих веб-ресурсів.
6. Виконати тестування системи для перевірки її ефективності.

Об'єкт дослідження – процес виявлення шкідливих веб-ресурсів у мережі Інтернет.

Предмет дослідження – методи та алгоритми машинного навчання.

Практична цінність роботи полягає у створенні системи, здатної автоматично аналізувати веб-ресурси та виявляти потенційні загрози у режимі реального часу. Розроблений підхід може бути використаний у корпоративних системах кібербезпеки для захисту мереж від фішингових атак, а також у освітніх цілях для навчання фахівців у сфері інформаційної безпеки.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 60 сторінки основного тексту, 25 рисунків та 2 таблиць.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1. Аналіз публікацій щодо загроз в мережі Інтернет

У сучасному технологічному світі веб-додатки завжди відкриті для широкого кола користувачів, і сьогодні важко уявити організацію, підприємство чи бізнес, які не мали б власного веб-ресурсу або не використовували б інтернет-технології. На відміну від внутрішніх мережевих програм, веб-браузер доступний для всіх, хто має підключення до інтернету, включаючи зловмисників. На жаль, розробники часто нехтують принципами безпеки веб-додатків. Команди здебільшого настільки зосереджені на функціональних аспектах, таких як програмування, дизайн інтерфейсу та зручність використання, що забувають приділити увагу стабільності та захищеності програмного продукту.

Шкідливе програмне забезпечення (англ. Malware, від поєднання слів «malicious» – «зловмисний» та «software» – «програмне забезпечення») – це програмний засіб, створений для порушення нормальної роботи комп’ютерів, мереж або інших пристроїв, а також для отримання несанкціонованого доступу до конфіденційних даних, що зберігаються в системі [1]. Воно може існувати у формі файлів, коду, скриптів або послідовностей байтів. Зазвичай поширюється через мережу Інтернет.

Безпекова загроза – це сукупність подій, умов або факторів, які можуть спричинити втрату, пошкодження або несанкціонований доступ до даних і мережевих ресурсів, завдаючи шкоди інформаційній системі. Такі загрози можуть набувати різних форм, зокрема:

- Порушення цілісності або структури даних: знищення, несанкціоноване розголошення чи модифікація інформації.
- Відмова в обслуговуванні (DoS) – навмисне перевантаження системи трафіком.

- Фінансові махінації, зокрема шахрайство та вимагання коштів.

Клієнт-серверні системи, що активно використовуються у сучасних мережах, особливо вразливі до таких кібератак:

1. DoS/DDoS-атаки – спрямовані на перевантаження серверів з метою виведення їх з ладу.
2. MITM (Man-in-the-Middle) – атаки типу «людина посередині», коли зловмисник перехоплює трафік між клієнтом та сервером.
3. Сніфінг – несанкціонований моніторинг та перехоплення мережевого трафіку.

Небезпечні URL-адреси, або Uniform Resource Locator (URL), є глобальними ідентифікаторами ресурсів у мережі Інтернет, що використовуються для локалізації та доступу до веб-ресурсів. Вони представляють собою рядок символів, який указує на місцезнаходження конкретного ресурсу в мережі. Стандартна структура URL, як правило, включає кілька ключових компонентів (рис. 1.1) [6]:

- схема – визначає протокол передачі даних (наприклад, HTTP, HTTPS);
- хост – указує IP-адресу або зареєстроване доменне ім’я ресурсу;
- шлях – ідентифікує конкретний ресурс на сервері;
- параметри запиту – передають додаткові дані, необхідні для обробки запиту сервером;
- фрагмент – застосовується для позначення конкретної секції сторінки.

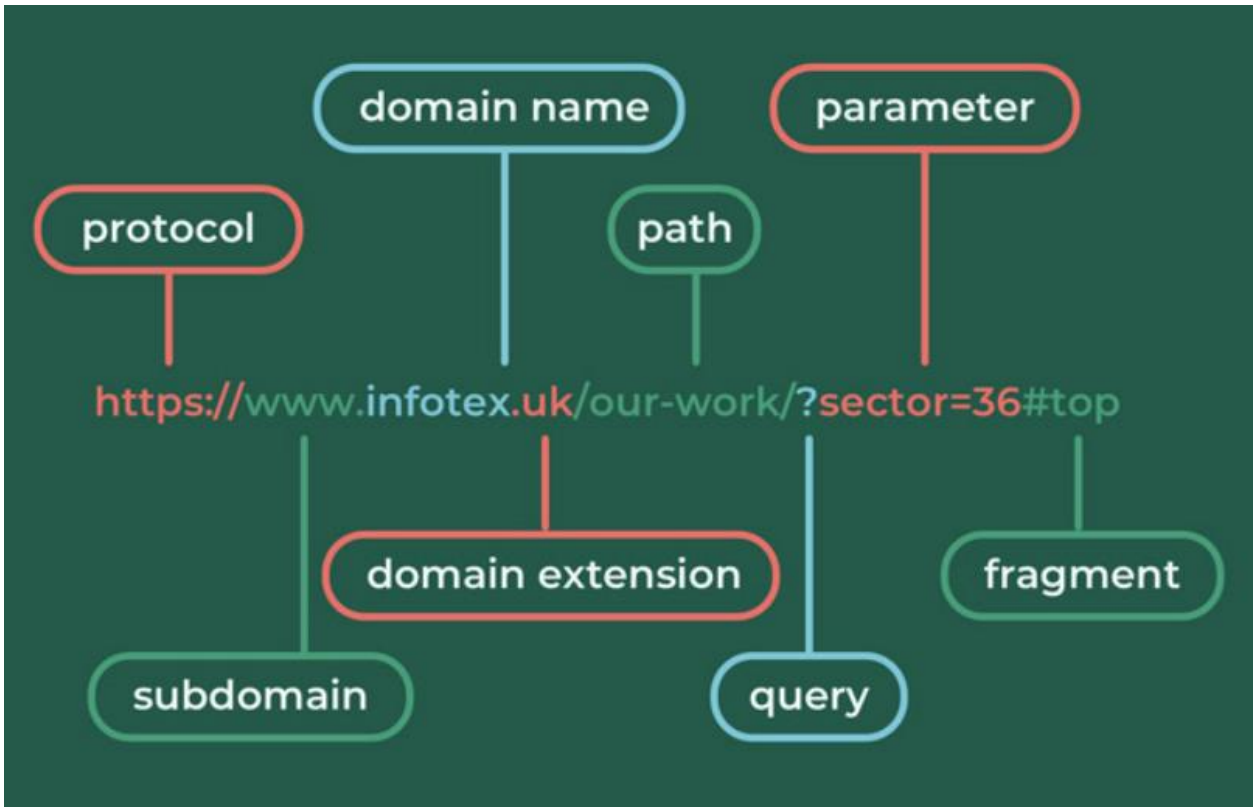

Рисунок 1.1 – Структура URL-адреси [11]

Термін «небезпечні URL-адреси» використовується для позначення веб-ресурсів, які містять загрозливий або небажаний контент. Вони становлять потенційну загрозу для користувачів, оскільки можуть містити шкідливий код, призначений для проведення різних типів кібератак. Такі URL-адреси можуть бути навмисно створеними зловмисниками або виникати внаслідок компрометації легітимних веб-сайтів.

Небезпечні URL-адреси класифікуються за характером загроз, які вони представляють, зокрема:

- спам-URL – використовуються для поширення небажаних рекламних або шахрайських повідомлень;
- фішингові URL – створені для введення користувача в оману з метою отримання конфіденційної інформації (наприклад, паролів або платіжних даних);

- URL-адреси з автоматичним завантаженням – спрямовані на примусове завантаження шкідливого програмного забезпечення на пристрій користувача.

Кожен із зазначених типів небезпечних URL-адрес характеризується специфічними методами реалізації та вимагає застосування різних підходів до їх виявлення та блокування. Ефективне виявлення загроз потребує застосування сучасних методів аналізу трафіку та машинного навчання, зважаючи на постійний розвиток технік маскування шкідливих посилань.

Веб-спам є формою недобросовісної оптимізації, спрямованої на маніпулювання алгоритмами пошукових систем задля штучного підвищення рейтингу веб-сайтів у результатах пошуку. Метою цієї техніки є збільшення відвідуваності ресурсу за рахунок отримання вищих позицій у пошуковій видачі. URL-адреси, що ведуть на такі сайти, називають спам-адресами. Незважаючи на їхній високий рейтинг, контент цих веб-ресурсів часто не відповідає запитам користувачів або взагалі є нерелевантним [2].

Існує два основних типи веб-спаму: контентний спам та посилальний спам.

Контентний спам передбачає навмисне спотворення або маніпуляцію вмістом веб-сторінки для підвищення її позицій у результатах пошуку. Ця практика базується на додаванні або зміні ключових елементів сторінки, що впливають на ранжування, включаючи:

- Перенасичення ключовими словами (keyword stuffing) – надмірне використання популярних ключових слів, яке робить текст неприродним і важким для читання.
- Маніпуляції з мета-тегами та заголовками – зміна заголовків сторінок, мета-описів та тегів для створення ілюзії релевантності запиту.
- Схований текст або контент (hidden text) – додавання тексту, який невидимий для користувача (наприклад, використання шрифту того ж кольору, що й фон сторінки), але індексується пошуковими системами.

- Дублювання контенту (duplicate content) – копіювання вмісту з інших ресурсів або створення кількох ідентичних сторінок із незначними змінами для маніпуляції результатами пошуку.

Посилальний спам полягає у маніпуляціях із зовнішніми посиланнями з метою покращення позицій у пошуковій видачі. Алгоритми пошукових систем, зокрема Google PageRank, враховують кількість і якість зовнішніх посилань як показник авторитетності ресурсу. Зловмисники використовують такі методи для штучного підвищення рейтингу:

- Купівля посилань (link buying) – придбання великої кількості зовнішніх посилань з метою збільшення показника довіри до сайту.
- Мережі приватних блогів (PBN – Private Blog Networks) – створення групи сайтів, які перехресно посилаються один на одного для штучного підвищення рейтингу.
- Коментарний спам (comment spam) – розміщення посилань у коментарях блогів, форумах чи інших інтерактивних платформах без жодної цінності для обговорення.
- Фармінг посилань (link farming) – створення веб-сторінок або сайтів, основна мета яких – лише розміщення великої кількості взаємних посилань.

Такі методи можуть тимчасово підвищити рейтинг сайту, проте сучасні пошукові системи, зокрема Google, активно вдосконалюють алгоритми для виявлення та покарання подібних практик.

Фішингові веб-сайти є поширеним засобом мережевого шахрайства, основною метою якого є незаконне отримання конфіденційної інформації користувачів шляхом введення їх в оману. Ці ресурси створюються зловмисниками для обманного спонукання до розкриття чутливих даних, таких як номери банківських карток, реквізити фінансових рахунків, паролі до облікових записів або особисті ідентифікаційні дані [3].

Фішингові веб-сайти, як правило, імітують дизайн та функціональність легітимних веб-ресурсів, зокрема інтернет-банкінгу, поштових сервісів,

платформ електронної комерції та соціальних мереж. Основною ознакою фішингових сайтів є схожість з оригінальними ресурсами, включаючи використання аналогічної колірної гами, логотипів, шрифтів та інших елементів дизайну.

Візуальна подібність доповнюється маніпуляціями з URL-адресами, які можуть містити незначні відмінності, що складно помітити неозброєним оком. Основні техніки модифікації URL включають:

- Підміна символів – заміна латинських символів схожими на кириличні або використання цифр замість літер (наприклад, bank.com → bank.com або rn замість m).
- Використання піддоменів – додавання неправдивих піддоменів, наприклад, secure.bank.com.fake-site.com.
- Незначні зміни у написанні домену – наприклад, використання bánk.com замість bank.com.

Фішингові URL-адреси зазвичай розповсюджуються через:

- Електронні листи із закликами до дії, наприклад оновлення паролю або підтвердження транзакції.
- SMS-повідомлення із посиланням на підроблену сторінку банківського сервісу.
- Особисті повідомлення у соціальних мережах із фальшивими акціями або виграшами.
- Спам-кампанії, що містять зловмисні посилання.

Основною метою фішингових атак є незаконне використання отриманих конфіденційних даних для:

- Отримання доступу до облікових записів у соціальних мережах, банківських системах або корпоративних мережах.
- Крадіжки особистих даних для подальшого використання у шахрайських діях, наприклад, для оформлення кредитів або поширення компрометуючої інформації.

- Продажу викрадених даних на нелегальних онлайн-майданчиках, зокрема в даркнеті.

Випадкове завантаження (англ. drive-by download) — це несанкціоноване та непомітне завантаження шкідливого програмного забезпечення (malware) на пристрій користувача під час відвідування скомпрометованих або зловмисних веб-ресурсів. Особливість цього виду загрози полягає в тому, що жертві не потрібно здійснювати жодних активних дій, таких як натискання посилань або завантаження файлів. Зараження може відбутися автоматично в момент відкриття веб-сторінки, оскільки шкідливий код вбудований у її структуру або мультимедійний контент [3].

Випадкове завантаження стає можливим через використання зловмисниками вразливостей у:

- Операційних системах;
- Веб-браузерах;
- Плагінах та додатках;
- Системах керування контентом.

Процес зараження зазвичай відбувається в кілька етапів:

1. Відвідування зараженого ресурсу.

Користувач відкриває веб-сайт, який містить вбудований шкідливий скрипт або елемент.

2. Експлуатація вразливості.

Шкідливий скрипт автоматично сканує систему користувача на наявність уразливостей.

3. Автоматичне завантаження.

Якщо знайдено вразливість, шкідливе програмне забезпечення завантажується на пристрій без явного сповіщення користувача.

4. Виконання коду.

Завантажений файл може автоматично запускатися у фоновому режимі, встановлюючи шкідливі програми або бекдори для подальшого контролю над пристроєм.

Шкідливе програмне забезпечення, яке може бути завантажене через випадкове завантаження, може включати:

- Троянські програми (Trojans) – створені для прихованого збору даних або створення бекдору.
- Програми-вимагачі (Ransomware) – шифрують файли користувача та вимагають викуп за їх розблокування.
- Програми-шпигуни (Spyware) – призначені для непомітного збору даних, таких як паролі, дані кредитних карток, історія браузера тощо.
- Ботнет-програми (Botnets) – перетворюють пристрій на частину мережі ботів для здійснення масових DDoS-атак [5].

1.2. Аналіз методів виявлення небезпечних веб-ресурсів

Використання повного рядка URL-адреси як єдиного джерела інформації для виявлення загроз може бути недостатньо ефективним. Тому першим важливим етапом у процесі ідентифікації шкідливих URL є вибір та вилучення ключових ознак, які здатні найкраще описати структуру та вміст URL-адреси. Застосування аналізу ознак замість обробки повного рядка дозволяє розробляти більш ефективні та точні алгоритми виявлення загроз.

Оскільки для різних типів шкідливих URL-адрес оптимальні набори ознак можуть відрізнятися, вибір відповідних параметрів є критично важливим для підвищення ефективності детекції. У наукових дослідженнях, присвячених виявленню шкідливих URL-адрес, запропоновано кілька категорій ознак, які можуть бути використані для цього завдання. До них належать:

- Чорні списки: перевірка URL-адреси на відповідність відомим базам даних шкідливих ресурсів.
- Лексичні ознаки: аналіз символів, довжини, доменних імен та інших текстових параметрів.

- Ознаки на основі хостів: використання інформації про IP-адресу, вік домену, наявність сертифіката безпеки тощо.

Метод виявлення шкідливих URL-адрес за допомогою чорних списків ґрунтується на порівнянні URL-адреси з попередньо створеними базами даних відомих шкідливих ресурсів. У таких базах містяться URL-адреси, які були ідентифіковані як потенційно небезпечні або вже залучені до кіберзлочинної діяльності, такої як фішинг, поширення шкідливого програмного забезпечення або обман користувачів [4, 7].

Принцип роботи:

1. URL-адреса перевіряється на відповідність записам у базі даних.
2. Якщо адреса збігається з елементом у списку, доступ до ресурсу блокується або користувач отримує попередження.

Приклади чорних списків:

- Google Safe Browsing;
- PhishTank;
- OpenPhish;
- Malware Domain List.

Переваги:

- Простота у впровадженні;
- Висока ефективність у виявленні відомих загроз;
- Мінімальні обчислювальні ресурси для перевірки URL.

Недоліки:

- Неможливість виявлення нових, раніше невідомих загроз;
- Залежність від регулярного оновлення баз даних;
- Обмеженість у випадках, коли URL-адреса не є повністю ідентичною до внесеної в список (наприклад, використання піддоменів або маскування символів).

Лексичний аналіз URL-адреси базується на дослідженні її текстових і символьних характеристик, які можуть вказувати на потенційну загрозу. Цей підхід фокусується на структурі URL-адреси, розділяючи її на компоненти,

такі як доменне ім’я, шлях, параметри запиту, довжина рядка тощо. Мета методу полягає у виявленні аномалій та шаблонів, притаманних шкідливим ресурсам.

Принцип роботи:

1. Токенізація URL: URL-адреса розділяється на компоненти (схема, хост, шлях, параметри).

2. Аналіз ключових ознак: Використовується набір текстових характеристик для оцінки ризиків.

3. Оцінка ризикових параметрів: На основі виявлених ознак проводиться оцінка потенційної загрози.

Основні лексичні ознаки:

- Довжина URL-адреси. Шкідливі URL часто мають надмірну довжину для маскування.
- Кількість піддоменів. Велика кількість піддоменів може свідчити про спробу приховати реальне джерело.
- Використання підозрілих символів. Часте застосування тире, дефісів або спеціальних символів (%, @, .).
- Повторення символів. Наприклад, secure-ssl-banking-login.com може вказувати на фішинговий сайт.
- Наявність числових IP-адрес у хості. Наприклад, http://192.168.1.1/malware.exe.

Метод аналізу на основі хостів спрямований на оцінку характеристик веб-хосту, до якого прив’язана URL-адреса. Він фокусується на технічних параметрах доменного імені, його історії та інфраструктури хостингу. Такий підхід дозволяє виявити підозрілі веб-ресурси, навіть якщо сама структура URL-адреси не містить явних ознак загрози.

Принцип роботи:

1. Витягнення даних про хост. Аналізуються такі параметри, як IP-адреса, доменне ім'я, дата реєстрації, геолокація сервера.

2. Перевірка репутації хосту. Порівняння з базами даних відомих шкідливих доменів та IP-адрес.

3. Оцінка віку та історії домену. Молоді домени часто використовуються для фішингу та шкідливих кампаній.

Основні ознаки для аналізу:

- IP-адреса хосту. Використання IP-адреси замість доменного імені (http://192.168.1.1).
- Вік домену. Молоді домени частіше створюються для фішингових атак.
- Розташування сервера. Географічне розташування може вказувати на підозрілі регіони з низькими стандартами кібербезпеки.
- Реєстратор домену. Використання анонімних або низькоякісних реєстраторів.
- Використання проксі або сервісів анонімності.
- Наявність сертифіката SSL/TLS. Відсутність https або використання безкоштовних сертифікатів для обману користувачів.

Метод виявлення шкідливих URL-адрес на основі машинного навчання (ML) базується на використанні алгоритмів штучного інтелекту для автоматизованої класифікації веб-ресурсів як шкідливих або безпечних. Процес включає збір даних, виділення ключових ознак URL (лексичних, хостових, поведінкових), навчання моделей на основі цих ознак, а також тестування та подальше використання моделі для аналізу нових веб-адрес.

Серед алгоритмів машинного навчання, що застосовуються для цієї задачі, поширеними є логістична регресія, дерева рішень, Random Forest, SVM та глибокі нейронні мережі. Вони дозволяють знаходити приховані закономірності у структурі URL-адрес та поведінці хостів, що ускладнює обходження системи кіберзахисту.

Основними перевагами підходу є здатність виявляти нові загрози без необхідності попередньо створених баз даних, ефективна автоматизація процесу та можливість обробки великих обсягів даних. Водночас метод має

обмеження, зокрема потребу в якісних даних для навчання, ресурсомісткість та ризик перенавчання.

Такі методи дозволяють створювати більш гнучкі та адаптивні системи виявлення, які можуть ефективно ідентифікувати шкідливі URL навіть за відсутності повного збігу з відомими загрозами.

1.3. Висновок до першого розділу

Метод виявлення шкідливих URL-адрес на основі машинного навчання (ML) є перспективним підходом для вирішення зазначених проблем. Він базується на використанні алгоритмів штучного інтелекту для автоматизованої класифікації веб-ресурсів як шкідливих або безпечних. Процес включає збір даних, виділення ключових ознак URL (лексичних, хостових, поведінкових), навчання моделей на основі цих ознак, а також тестування та подальше використання моделі для аналізу нових веб-адрес.

Серед алгоритмів машинного навчання, що застосовуються для цієї задачі, поширеними є логістична регресія, дерева рішень, Random Forest, SVM та глибокі нейронні мережі. Вони дозволяють знаходити приховані закономірності у структурі URL-адрес та поведінці хостів, що ускладнює обходження системи кіберзахисту.

З урахуванням зазначеного вище тема кваліфікаційної роботи «Розробка інформаційної системи виявлення шкідливих веб-ресурсів з використанням технологій машинного навчання» є актуальною.

У даному розділі було проведено дослідження загроз, пов’язаних із використанням веб-ресурсів, зокрема фішингових атак, поширення шкідливого програмного забезпечення та небезпечних URL-адрес. Проаналізовано основні типи загроз, їхні характеристики та способи реалізації кібератак, зокрема через маніпуляції з URL-адресами, використання піддоменів, спам-посилань та автоматичних завантажень зловмисного коду.

Особливу увагу приділено класифікації небезпечних URL-адрес, серед яких фішингові посилання, шкідливі скрипти та URL-адреси, що використовуються для розповсюдження зловмисного ПЗ. Розглянуто принципи дії таких загроз, їхній вплив на інформаційну безпеку користувачів та основні методи їхнього виявлення.

Метою дослідження є розробка системи автоматизованого виявлення шкідливих веб-ресурсів із використанням методів машинного навчання для покращення рівня кібербезпеки.

Для досягнення поставленої мети було визначено та виконано такі завдання дослідження:

1. Проведено аналіз наукових джерел щодо загроз у веб-середовищі та способів їх виявлення.
2. Визначено основні критерії для класифікації небезпечних веб-ресурсів.
3. Проаналізовано обмеження традиційних методів виявлення загроз.
4. Сформульовано вимоги до системи виявлення.
5. Розроблено систему виявлення шкідливих веб-ресурсів.

Вхідними даними для подальшого дослідження є вибірка URL-адрес, що включає як безпечні, так і шкідливі ресурси, які будуть використані для навчання та тестування моделей машинного навчання.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ ВИЯВЛЕННЯ ШКІДЛИВИХ ВЕБ-РЕСУРСІВ

2.1. Вибір методів реалізації

Методи машинного навчання (ML) активно застосовуються для виявлення шкідливих ресурсів шляхом навчання моделей класифікації на основі набору ознак. Основна ідея полягає у представленні кожного ресурсу як набору ознак, що характеризують його структуру та поведінкові аспекти, на основі яких модель навчається прогнозувати, чи є ресурс шкідливим або безпечним. Перевагою такого підходу є здатність виявляти нові потенційно небезпечні ресурси без необхідності попереднього внесення їх до чорних списків.

Процес виявлення шкідливих ресурсів за допомогою ML складається з двох основних етапів:

1. Вибір і представлення ознак

На цьому етапі виконується вибір інформативних ознак, які можуть характеризувати ресурс. Відбір ознак зазвичай базується на попередньому досвіді та евристичних методах, наприклад, довжина, кількість піддоменів, використання числових IP-адрес. Ознаки можуть бути як лексичними, так і на основі хостів або поведінкових факторів.

Процес підготовки даних є критично важливим для забезпечення ефективності моделей машинного навчання. Першим кроком є перетворення необроблених даних у числові вектори ознак, які можуть бути сприйняті алгоритмом. Зазвичай дані потребують нормалізації та очищення, оскільки якість вхідних даних безпосередньо впливає на точність навчання моделі.

Числове представлення тексту є ключовим аспектом підготовки ознак, оскільки текстові значення необхідно конвертувати у форму, придатну для обчислень. Нижче розглянуто основні методи кодування текстових даних:

- Bag-of-Words

Цей метод перетворює текстові дані у вектори, які відображають наявність або відсутність певних слів у наборі даних. Всі унікальні слова формують словник, а для кожного ресурсу створюється вектор, де 1 вказує на наявність слова, а 0 – на його відсутність. Недоліком методу є ігнорування порядку слів і створення розріджених матриць з великою кількістю нульових значень.

- One-Hot Encoding

Подібний до Bag-of-Words, але зберігає порядок слів. Кожне слово кодується в окремий вектор, де лише одна позиція має значення 1, а решта – 0. Метод зберігає більше структурної інформації, але все ще не фіксує зв’язків між словами.

- Кодування в унікальні числа (Integer Encoding)

У цьому підході кожне унікальне слово або токен перетворюється у числове значення. Це більш ефективний метод порівняно з попередніми, оскільки створює щільніші вектори, але не враховує семантичної подібності між словами.

- Вкладення слів

Вкладення слів, зокрема метод Word2Vec, є більш просунутим методом, що представляє слова у вигляді багатовимірних векторів з плаваючими значеннями, де схожі за значенням слова мають схожі вектори. Це дозволяє зберегти семантичні зв’язки між словами, наприклад, “банк” і “банківський”.

2. Навчання моделі

Після формування набору ознак дані перетворюються у вектор числових значень, придатних для обробки алгоритмами машинного навчання. Модель навчається на великому наборі даних, що містить як безпечні, так і шкідливі ресурси, з метою виявлення закономірностей, які дозволять розпізнавати загрози у нових даних.

Машинне навчання можна класифікувати за методом обробки даних під час навчання:

- Контрольоване навчання (Supervised Learning): Модель навчається на основі мічених даних, де для кожного ресурсу відомо, чи є він шкідливим або безпечним.
- Неконтрольоване навчання (Unsupervised Learning): Використовується, коли мітки відсутні, і модель виявляє аномалії або групує схожі ресурси без попереднього знання про їхню природу.
- Напівконтрольоване навчання (Semi-supervised Learning): Поєднує обидва підходи, використовуючи частково мічені дані для навчання моделі.

Також підхід може розділятися за типом подачі даних:

- Пакетне навчання (Batch Learning): Модель навчається на фіксованому наборі даних одноразово.
- Навчання в реальному часі (Online Learning): Модель постійно оновлюється, приймаючи нові дані потоками.

Методи машинного навчання у виявленні шкідливих URL-адрес є потужним інструментом для забезпечення кібербезпеки, особливо завдяки здатності виявляти нові загрози без необхідності попереднього додавання їх до чорних списків. Ефективність такого підходу залежить від якості даних, правильного вибору ознак та алгоритмів навчання. Комбінація методів представлення даних, таких як Bag-of-Words, One-Hot Encoding та Word2Vec, дозволяє створювати гнучкі та ефективні моделі для аналізу шкідливих URL-адрес. Однак для максимізації точності доцільно використовувати поєднання декількох підходів, зокрема контрольованого навчання та пакетного оновлення моделей.

2.2. Вимоги до реалізації системи виявлення шкідливих веб-ресурсів

Вимоги до системи виявлення небезпечних веб-ресурсів визначають набір ключових характеристик, які необхідно реалізувати для забезпечення ефективності, точності та стабільності системи. Чітке формулювання цих вимог є критично важливим етапом у розробці програмного забезпечення, оскільки воно гарантує відповідність функціональності технічним обмеженням та потребам користувачів.

Функціональні вимоги описують основний функціонал, який має виконувати алгоритм. Вони охоплюють обробку вхідних даних, виділення ключових ознак, класифікацію веб-ресурсів за допомогою моделей машинного навчання та забезпечення виведення результатів у зручній формі для користувача.

Нефункціональні вимоги зосереджені на таких аспектах, як продуктивність, масштабованість, надійність та зручність використання програмного забезпечення. Вони визначають критерії для швидкості обробки ресурсів, стабільності роботи алгоритмів, а також вимоги до якості обробки даних для мінімізації хибнопозитивних результатів.

1. Функціональні вимоги:

- Аналіз веб-ресурсів – система повинна аналізувати веб-ресурси для виявлення потенційних загроз шляхом обробки структурних, лексичних і поведінкових ознак.
- Обробка ознак – повинна бути реалізована можливість обробки ознак, включаючи довжину домену, кількість піддоменів, рейтинг хосту та використання IP-адрес.
- Класифікація ресурсів – система повинна визначати безпечні або небезпечні.
- Інтерфейс користувача, що дозволяє користувачам вводити URL для аналізу та отримувати результат класифікації.
- Виведення результатів

2. Нефункціональні вимоги:

- Продуктивність;
- Масштабованість;
- Надійність;
- Інтерфейс повинен бути простим та інтуїтивно зрозумілим.

3. Вимоги до даних:

- Використання даних із перевірених джерел.
- Необхідно передбачити механізми нормалізації та кодування ознак.
- Дані мають включати як безпечні, так і шкідливі ресурси для збалансованого навчання моделі.

2.3. Вибір набору даних для навчання моделі

Вибір відповідного набору даних є ключовим етапом у процесі розробки систем з використанням методів машинного навчання. Якість даних безпосередньо впливає на ефективність та точність класифікаційної моделі, зокрема на здатність системи ідентифікувати нові загрози та мінімізувати кількість хибно-позитивних спрацювань. Для забезпечення надійного навчання моделей були розглянуті кілька авторитетних платформ, що спеціалізуються на зборі та аналізі шкідливих ресурсів:

1. PhishTank,
2. URLhaus,
3. VirusTotal,
4. OpenPhish.

Ці платформи надають дані про небезпечні URL-адреси, фішингові сайти та ресурси, що поширюють шкідливе програмне забезпечення, забезпечуючи різні підходи до збору та оновлення інформації.

PhishTank — це відкрита платформа для виявлення, збору та перевірки фішингових веб-ресурсів, створена з метою підвищення безпеки користувачів у мережі Інтернет. Вона забезпечує централізоване сховище перевірених фішингових сайтів, які збираються спільнотою фахівців із кібербезпеки та добровольцями. Основною метою платформи є раннє виявлення фішингових атак і попередження користувачів про потенційні загрози.

PhishTank працює за принципом краудсорсингу, де користувачі можуть надсилати підозрілі URL-адреси для перевірки. Після подачі ресурс проходить перевірку модераторами платформи або автоматизованими інструментами. Якщо фішингову активність підтверджено, URL-адресу додають до відкритої бази даних. Ця база доступна для завантаження та інтеграції через API, що дозволяє автоматизувати процес блокування небезпечних ресурсів у корпоративних мережах та системах захисту [8].

Основними перевагами PhishTank є відкритий доступ до перевірених фішингових ресурсів та підтримка API для автоматизованої перевірки. Це дозволяє інтегрувати базу даних у системи кібербезпеки для зниження ризику фішингових атак. Однак оскільки дані частково перевіряються спільнотою, можливе включення хибнопозитивних результатів або затримки в оновленні бази.

Набір даних PhishTank включає:

- URL-адреси: Підозрілі ресурси, перевірені та підтверджені як фішингові.
- Дата виявлення: Час, коли ресурс був доданий до бази даних.
- Статус перевірки: Підтвердження факту фішингової активності.
- Тип загрози: Опис загрози (крадіжка облікових даних, банківські фішингові атаки).
- Ідентифікатор ресурсу: Унікальний ID для кожного запису у базі.

URLhaus — це платформа для збору та моніторингу шкідливих веб-ресурсів, зокрема фішингових сайтів, сторінок для поширення шкідливого програмного забезпечення (Malware) та інших загроз у реальному часі. Вона

спрямована на виявлення, документування та блокування URL-адрес, які використовуються для здійснення кібератак. URLhaus підтримується командою Abuse.ch та кіберспільнотою, що дозволяє швидко виявляти нові загрози завдяки колективним зусиллям.

Сервіс працює за принципом краудсорсингу: фахівці з кібербезпеки та добровольці надають URL-адреси, які перевіряються та додаються до бази даних після підтвердження загрози. Виявлені ресурси аналізуються за лексичними та поведінковими ознаками, включаючи використання шкідливих скриптів, завантаження вірусів та обхід систем безпеки. Платформа також автоматично перевіряє ресурси на актуальність, видаляючи застарілі загрози з бази даних. Дані можуть бути завантажені через API, що дозволяє інтегрувати їх у системи захисту інформації [9].

Основною перевагою URLhaus є доступність оновлюваної бази шкідливих ресурсів у відкритому форматі. Це дозволяє як дослідникам, так і фахівцям із кібербезпеки використовувати її для моніторингу загроз, навчання моделей машинного навчання та інтеграції в системи блокування шкідливих сайтів. Водночас, через краудсорсинговий принцип можливі хибнопозитивні результати, оскільки перевірка частково покладається на спільноту.

Набір даних URLhaus включає:

- URL-адреси: Посилання на шкідливі ресурси, зафіксовані у базі даних.
- Тип загрози: Вказівка, чи ресурс містить фішинг, троянські програми або інше шкідливе ПЗ.
- Дата виявлення: Час, коли ресурс був доданий до бази.
- Статус ресурсу: Вказує, чи є ресурс активним на момент перевірки.
- IP-адреса та домен: Технічні дані про хостинг шкідливого ресурсу.

VirusTotal — це багатофункціональна платформа для автоматизованого аналізу веб-ресурсів, файлів та програм на наявність шкідливого програмного забезпечення. Вона призначена для виявлення потенційних загроз, таких як

віруси, трояни, шпигунське ПЗ та фішингові сторінки. Платформа дозволяє перевіряти URL-адреси та файли одночасно за допомогою кількох антивірусних двигунів та систем аналізу веб-загроз, надаючи користувачеві об'єднаний звіт із результатами перевірок.

VirusTotal працює за принципом централізованої системи сканування. Платформа приймає завантажені користувачем файли або URL-адреси, передає їх на перевірку через більш ніж 70 антивірусних механізмів і веб-сканерів, а потім об'єднує результати в єдиний звіт. Ресурси аналізуються як за сигнатурами відомих загроз, так і за поведінковими характеристиками, включаючи виконання коду у віртуальних середовищах для виявлення прихованих шкідливих дій. Дані постійно оновлюються, що дозволяє платформі підтримувати високу актуальність інформації про загрози [10].

Основною перевагою VirusTotal є масштабність та глибина аналізу, адже він використовує одночасно декілька систем перевірки, забезпечуючи широку детекцію загроз. Платформа також підтримує API для автоматизації перевірок, що робить її корисною як для окремих користувачів, так і для компаній у сфері кібербезпеки. Проте, через використання статичних сигнатурних методів, можливі хибнопозитивні результати, особливо при аналізі нових або малоактивних файлів.

Набір даних VirusTotal включає:

- URL-адреси та файли: Перелік перевірених ресурсів та файлів.
- Дата перевірки: Час, коли файл або ресурс був проаналізований.
- Результати перевірок: Інформація від кількох антивірусних механізмів про виявлення загроз.
- Пов'язані мета-дані: Наприклад, хеші файлів, IP-адреси серверів, геолокаційні дані, цифрові підписи.

OpenPhish — це автоматизована платформа для виявлення фішингових ресурсів у реальному часі. Вона спрямована на ідентифікацію веб-сторінок, що використовуються для обманного отримання конфіденційної інформації, такої як паролі, реквізити платіжних карток та особисті дані. Платформа

збирає та аналізує веб-ресурси, перевіряючи їх на наявність характерних ознак фішингу, зокрема підроблених форм авторизації та схожості з легітимними сайтами.

OpenPhish працює за принципом автоматизованого сканування, що включає збір URL-адрес з відкритих джерел, баз даних та повідомлень користувачів. Виявлені ресурси аналізуються за допомогою лексичних та поведінкових ознак, а потім автоматично додаються до бази даних платформи. Ця база постійно оновлюється, забезпечуючи актуальність даних для захисту від нових фішингових кампаній [11].

Однією з ключових переваг OpenPhish є її здатність до автоматичного збору та оновлення інформації у реальному часі. Платформа пропонує API для інтеграції з корпоративними системами кібербезпеки, що дозволяє автоматизувати моніторинг загроз і блокування небезпечних ресурсів. Це робить OpenPhish ефективним інструментом для компаній, що прагнуть підвищити рівень захисту своїх мереж.

Однак платформа має і певні обмеження. Через повну автоматизацію можливе виникнення хибнопозитивних результатів, коли легітимний ресурс може бути помилково класифікований як фішинговий. Крім того, розширені функції, такі як доступ до історичних даних та повного набору виявлених ресурсів, доступні лише у платній версії платформи.

Набір даних OpenPhish є цінним ресурсом для дослідження фішингових атак, навчання моделей машинного навчання для виявлення шкідливих ресурсів та розробки систем попередження фішингових загроз у реальному часі.

Основними компонентами, включеними до набору даних OpenPhish, є:

- URL-адреси: Посилання на виявлені фішингові сторінки, що підозрюються у шахрайській діяльності.
- Дата виявлення: Час, коли ресурс був виявлений та доданий до бази даних.

- Статус загрози: Опис рівня загрози (підозрілий, підтверджений фішинговий ресурс).
- Тип фішингу: Інформація про ціль атаки, наприклад, фішинг банківських облікових записів або крадіжка паролів від соціальних мереж.

Набір даних, наданий платформою, може також містити додаткову технічну інформацію: IP-адресу сервера, геолокацію хоста, а також цифрові відбитки сторінки, які допомагають унікально ідентифікувати ресурс. Це дозволяє аналітикам використовувати ці дані для глибшого аналізу загроз.

Нижче представлено порівняння чотирьох популярних платформ — PhishTank, URLhaus, VirusTotal та OpenPhish. Таблиця 2.1 демонструє їхні ключові характеристики, зокрема методи збору даних, частоту оновлення, можливість інтеграції через API, а також сильні та слабкі сторони кожної з платформ.

Таблиця 2.1.

Порівняння платформ з відкритими данними

Критерій	PhishTank	URLhaus	VirusTotal	OpenPhish
Основне призначення	Виявлення та перевірка фішингових сайтів	Виявлення шкідливих та фішингових ресурсів	Аналіз файлів і URL-адрес за допомогою антивірусів	Виявлення фішингових ресурсів у реальному часі
Тип даних у базі	Фішингові URL-адреси	Шкідливі URL-адреси, фішингові сторінки	Шкідливі файли та URL-адреси	Фішингові URL-адреси
Метод збору даних	Краудсорсинг (користувачі додають ресурси)	Краудсорсинг + автоматизований моніторинг	Автоматизований аналіз через багатомодульні сканери	Автоматизоване сканування URL-адрес
API для інтеграції	Так (відкритий для публічного використання)	Так (відкритий для кіберспільноти)	Так (обмежений у безкоштовній версії)	Так (обмежений у безкоштовній версії)
Тип загроз, що виявляються	Фішингові атаки	Фішинг, поширення шкідливого ПЗ	Віруси, трояни, шпигунські програми, фішинг	Фішингові сайти, підроблені сторінки для крадіжки даних

Частота оновлення	Постійно оновлюється спільнотою	Постійно оновлюється (кілька разів на день)	У реальному часі	У реальному часі
Можливість завантаження даних	Так, CSV та через API	Так, через API та списки TXT	Так, через API (частково обмежений)	Так, через API (обмежений у безкоштовній версії)
Верифікація загроз	Перевірка модераторами після подачі	Частково автоматизована, частково модераторами	Повністю автоматизована перевірка	Повністю автоматизована перевірка
Основні переваги	Відкритий доступ, проста інтеграція через API	Актуальність даних, відкритий доступ	Велика база даних, багатомодульний аналіз	Висока швидкість оновлення, автоматизація
Основні недоліки	Можливі затримки у верифікації	Можливі хибнопозитивні спрацювання	Обмеження у безкоштовній версії, хибнопозитиви	Обмеження доступу до розширених функцій

OpenPhish було обрано серед інших платформ завдяки автоматизованому підходу до виявлення фішингових ресурсів у реальному часі, що забезпечує актуальність даних та мінімізує затримки в оновленні бази загроз. Порівняно з PhishTank та URLhaus, які покладаються на краудсорсинг та можуть мати затримки у перевірці, OpenPhish використовує повністю автоматизовану перевірку, що знижує людський фактор. У порівнянні з VirusTotal, який фокусується на комплексному аналізі файлів, OpenPhish спеціалізується виключно на фішингових атаках, що робить його більш ефективним у цій сфері.

2.4. Вибір бібліотеки машинного навчання

Бібліотеки машинного навчання є важливими інструментами для розробки, навчання та оцінки моделей, що застосовуються у сфері штучного інтелекту та аналізу даних. Вони надають готові реалізації алгоритмів, спрощують обробку великих обсягів даних та автоматизують ключові етапи розробки моделей. Використання бібліотек дозволяє дослідникам та

розробникам зосередитися на аналізі даних та підборі моделей, зменшуючи потребу у створенні алгоритмів з нуля.

Серед найпопулярніших бібліотек для машинного навчання можна виділити Scikit-Learn, TensorFlow, PyTorch та інші. Вони охоплюють різні аспекти машинного навчання: від класичних алгоритмів до складних нейронних мереж та методів глибокого навчання. Завдяки своїй універсальності та зручності, ці бібліотеки застосовуються у широкому спектрі завдань: від обробки текстових даних до виявлення кіберзагроз та медичної діагностики. Вибір конкретної бібліотеки залежить від складності завдання, обсягу даних та технічних вимог до системи. Розглянемо деякі з них:

Scikit-Learn — це потужна бібліотека машинного навчання для мови Python, призначена для реалізації класичних алгоритмів машинного навчання. Вона використовується для вирішення широкого спектра завдань, включаючи класифікацію, регресію, кластеризацію та зниження розмірності даних. Бібліотека забезпечує зручний інтерфейс для роботи з алгоритмами, такими як Random Forest, Decision Tree, SVM, K-Means, а також містить інструменти для підготовки даних, масштабування та крос-валідації.

Scikit-Learn працює на основі модульного підходу, де кожен етап побудови моделі є окремим компонентом. Це включає завантаження та обробку даних, вибір моделі, навчання, прогнозування та оцінку результатів. Бібліотека побудована поверх NumPy, SciPy та Matplotlib, що забезпечує її ефективність у роботі з табличними даними та зручну візуалізацію результатів. Вона також підтримує створення конвеєрів обробки даних, які автоматизують весь процес моделювання [13].

Однією з ключових переваг Scikit-Learn є простота використання, завдяки чому вона підходить як для новачків, так і для досвідчених фахівців. Вона має широку документацію, а також інтегрується з іншими бібліотеками для аналізу даних, такими як Pandas. Бібліотека дозволяє легко порівнювати декілька моделей машинного навчання, забезпечуючи інструменти для автоматизованої крос-валідації та вибору найкращих гіперпараметрів.

Проте Scikit-Learn має і деякі обмеження. Вона не призначена для обробки глибоких нейронних мереж, а також може бути менш ефективною при роботі з надзвичайно великими наборами даних у порівнянні з такими бібліотеками, як TensorFlow або PyTorch. Вона також не підтримує роботу з неструктурованими даними, такими як зображення або аудіо, фокусуючись переважно на табличних даних.

TensorFlow — це потужна бібліотека для машинного та глибокого навчання, розроблена компанією Google. Вона призначена для створення, навчання та розгортання моделей нейронних мереж, а також проведення складних обчислень із великими наборами даних. TensorFlow є відкритим програмним забезпеченням та підтримує як дослідницькі, так і комерційні проекти, надаючи широкі можливості для аналізу даних, комп'ютерного зору, обробки природної мови (NLP) та прогнозування.

TensorFlow побудований на основі концепції обчислювальних графів (data flow graphs), де кожна операція моделюється у вигляді вузла графа, а потоки даних — у вигляді ребер між ними. Ця архітектура дозволяє ефективно виконувати обчислення на різних апаратних платформах, включаючи CPU, GPU та TPU. TensorFlow підтримує як низькорівневий контроль через основний API, так і високорівневий модуль Keras, який значно спрощує створення моделей нейронних мереж [14].

Однією з ключових переваг TensorFlow є його масштабованість та підтримка паралельних обчислень, що робить його придатним для роботи з великими наборами даних і складними архітектурами нейронних мереж. Бібліотека також пропонує інструменти для автоматизованого пошуку гіперпараметрів, інтерпретації моделей (TensorBoard) та обробки даних. TensorFlow дозволяє експортувати готові моделі для використання на різних платформах, зокрема у мобільних додатках через TensorFlow Lite та у веб-застосунках через TensorFlow.js.

Водночас TensorFlow має і певні обмеження. Через свою багатофункціональність бібліотека є більш складною у використанні для

початківців у порівнянні з високорівневими бібліотеками, такими як Keras. Також робота з обчислювальними графами може вимагати глибшого розуміння архітектури нейронних мереж.

PyTorch — це гнучка та потужна бібліотека для машинного навчання, зосереджена на глибокому навчанні. Вона була розроблена компанією Meta (Facebook) та набула широкого поширення серед дослідників і розробників завдяки зручному API та підтримці динамічних обчислювальних графів. PyTorch використовується для створення, тренування та тестування нейронних мереж у задачах комп'ютерного зору, обробки природної мови (NLP) та інших галузях штучного інтелекту.

Головною архітектурною особливістю PyTorch є використання динамічних обчислювальних графів (dynamic computation graphs). Це означає, що структура графа створюється "на льоту" під час виконання програми, що дозволяє більш гнучко та зручно працювати з моделями, зокрема для дослідницьких експериментів. PyTorch також підтримує обчислення на GPU завдяки інтеграції з бібліотекою CUDA, що значно прискорює навчання моделей на великих наборах даних [15].

PyTorch має модульну структуру, яка включає високорівневі інструменти для спрощення розробки моделей. torch.nn використовується для створення шарів нейронних мереж, torch.optim — для налаштування алгоритмів оптимізації, а torch.utils.data — для завантаження та обробки даних. Також доступний модуль TorchVision для роботи із зображеннями та TorchText для обробки текстових даних.

Однією з ключових переваг PyTorch є інтуїтивний синтаксис та зручність у використанні, що робить бібліотеку популярною серед дослідників. Вона забезпечує зручний відладочний процес завдяки динамічним обчисленням, а також підтримує автоматичне диференціювання через функціонал Autograd.

Проте PyTorch має і певні обмеження. Вона менш оптимізована для продакшн-рішень порівняно з TensorFlow, хоча останні оновлення, такі як

TorchServe, значно покращили можливості для розгортання моделей. PyTorch також може вимагати більше налаштувань при роботі з великими наборами даних порівняно з TensorFlow.

Незважаючи на ці обмеження, PyTorch залишається одним із найкращих інструментів для дослідницьких проектів у сфері глибокого навчання, забезпечуючи гнучкість, ефективність та високу продуктивність у роботі з нейронними мережами.

Ці три бібліотеки широко використовуються у сфері аналізу даних, глибокого навчання та класичних алгоритмів ML. PyTorch та TensorFlow призначені для роботи з нейронними мережами, де PyTorch відзначається гнучкістю для дослідницьких проектів, а TensorFlow оптимізований для розгортання у виробничих середовищах. Scikit-Learn, своєю чергою, фокусується на класичних алгоритмах машинного навчання та підходить для прототипування та базового аналізу даних. Таблиця 2.2 демонструє ключові відмінності між цими бібліотеками за критеріями функціональності, продуктивності та зручності використання.

Таблиця 2.2.

Порівняння бібліотек машинного навчання

Критерій	PyTorch	TensorFlow	Scikit-Learn
Основне призначення	Глибоке навчання, дослідницькі експерименти	Глибоке навчання, виробничі середовища	Класичні алгоритми машинного навчання
Архітектура	Динамічний обчислювальний граф	Статичний (Graph) + динамічний (Eager) граф	Структурований підхід з модульністю
Типи моделей	Нейронні мережі, глибоке навчання	Нейронні мережі, глибоке навчання	Класичні алгоритми (SVM, Decision Tree, Random Forest)
Область застосування	Комп’ютерний зір, NLP, наукові дослідження	Комп’ютерний зір, NLP, виробничі системи	Класифікація, регресія, кластеризація

Інтерфейс користувача	Гнучкий та інтуїтивний для дослідників	Більш складний, але оптимізований для продакшн	Простий, модульний API
Підтримка GPU	Вбудована через CUDA	Вбудована через CUDA/TPU	Відсутня (підтримує лише CPU)
Модуль для створення моделей	torch.nn	tf.keras	sklearn.linear_model, sklearn.tree
Обробка даних	torch.utils.data	tf.data.Dataset	sklearn.preprocessing
Оптимізація	torch.optim	tf.optimizers	sklearn.model_selection
Візуалізація процесу навчання	Обмежені можливості	TensorBoard	Обмежені (через Matplotlib)
Автоматичне диференціювання	Вбудоване через autograd	Вбудоване через GradientTape	Відсутнє
Підтримка продакшн-рішень	Менш оптимізований для продакшну	Висока (TensorFlow Lite, TensorFlow Serving)	Низька, використовується для прототипів
Простота для новачків	Відносно простий для дослідників	Більш складний для початківців	Легкий у використанні
Продуктивність на великих даних	Висока	Дуже висока	Обмежена для великих наборів даних
Основні переваги	Гнучкість, динамічний граф, зручний API	Висока продуктивність, оптимізація для продакшну	Простий у використанні, великий набір алгоритмів
Основні недоліки	Менш оптимізований для продакшну	Складність для новачків	Відсутність підтримки глибокого навчання та GPU
Популярність у спільноті	Висока серед дослідників	Висока у виробничих рішеннях	Висока серед початківців та для класичного ML
Ліцензія	Відкрита (BSD)	Відкрита (Apache 2.0)	Відкрита (BSD)

Scikit-Learn було обрано для реалізації моделі машинного навчання через його простоту використання, широкий набір класичних алгоритмів та інструментів для обробки даних. Бібліотека забезпечує ефективні методи для класифікації, регресії та кластеризації, а також включає модулі для попередньої обробки даних, крос-валідації та оцінки моделей, що робить її універсальним інструментом для завдань з аналізу структурованих даних.

Завдяки інтуїтивному API та відсутності необхідності у складних налаштуваннях, Scikit-Learn є оптимальним вибором для задач, де важлива швидкість прототипування та інтерпретованість моделей, особливо для класичних алгоритмів машинного навчання.

2.5. Проектування методу виявлення небезпечних веб-ресурсів

Проектування методу виявлення небезпечних веб-ресурсів базується на використанні машинного навчання для аналізу URL-адрес та їх характеристик. Основний підхід складається з кількох етапів, що забезпечують ефективний збір, обробку даних та навчання моделей для класифікації ресурсів.

Запропонований метод складається з двох основних компонентів:

- Аналіз ознак ресурсу – вилучення ключових характеристик ресурсу, таких як довжина домену, кількість піддоменів, використання IP-адрес у URL, рейтинг хосту та інші структурні показники.
- Класифікація на основі ансамблевих моделей – паралельне використання двох алгоритмів для підвищення точності виявлення загроз та зниження рівня хибнопозитивних результатів.

Послідовність роботи методу

1. На першому етапі система ініціалізується:

- Завантажуються необхідні бібліотеки Python.
- Імпортується CSV-файл.
- Виконується обробка даних.

2. На другому етапі здійснюється вибір ключових ознак, найбільш інформативних для класифікації:

- Лексичні ознаки: довжина URL, кількість піддоменів.
- Ознаки хосту: наявність IP-адреси, тип домену.
- Поведінкові: кількість перенаправлень, використання скриптів.

Дані розділяються на навчальний та тестовий набори у пропорції 80% на 20% для забезпечення надійності оцінки.

3. Навчання моделей Random Forest та Decision Tree. Обидві моделі тренуються одночасно на одному наборі ознак.

4. Введення користувачем ресурсу через графічний інтерфейс програми

5. Нормалізація введених даних – перетворення текстових ознак у числові та масштабування даних.

6. Аналіз двома попередньо навченими моделями – Random Forest Classifier та Decision Tree Classifier

7. Виведення результату перевірки

Якщо хоча б одна з моделей виявляє підозрілі ознаки, ресурс отримує мітку «шкідливий», а система попереджає користувача про потенційну загрозу.

2.6. Висновок до другого розділу

У другому розділі було проведено комплексний аналіз методів та засобів для реалізації системи виявлення шкідливих веб-ресурсів із використанням машинного навчання. Було розглянуто можливості та функціональність різних підходів до класифікації URL-адрес, зокрема аналіз лексичних ознак, характеристик хосту та поведінкових факторів.

Було проаналізовано методи машинного навчання, такі як Random Forest та Decision Tree, які здатні ефективно працювати зі структурованими даними та забезпечувати високу точність класифікації.

Та обрано платформу з даним для навчання – OpenPhish, завдяки її автоматизованому підходу до збору даних, актуальності бази фішингових ресурсів у реальному часі та мінімізації затримок у верифікації нових загроз. Це дозволяє використовувати найбільш актуальні дані для навчання моделей, що критично важливо для виявлення сучасних загроз.

За результатами аналізу, для реалізації системи виявлення було обрано бібліотеку машинного навчання Scikit-Learn завдяки її простоті використання, широкому спектру інструментів для попередньої обробки даних, а також

наявності вбудованих алгоритмів класифікації: Random Forest та Decision Tree. Scikit-Learn забезпечує гнучкість у роботі з класичними моделями машинного навчання, зручні інструменти для оцінки продуктивності моделей та можливість інтеграції з іншими бібліотеками Python для аналізу даних.

Також було визначено наступні вимоги до системи:

1. Функціональних вимог – можливість аналізу структурних та поведінкових ознак URL-адрес, автоматична класифікація веб-ресурсів за допомогою алгоритмів машинного навчання, виведення результатів класифікації у зручному для користувача форматі.

2. Нефункціональних вимоги – висока продуктивність, масштабованість системи та здатність працювати із великими обсягами даних без втрати точності.

Таким чином, проведений аналіз методів, платформ та інструментів дозволив обґрунтовано обрати OpenPhish як основне джерело даних для навчання моделі та Scikit-Learn як оптимальний інструмент для реалізації системи виявлення шкідливих веб-ресурсів, що забезпечує баланс між простотою реалізації, точністю класифікації та ефективністю обробки даних.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ ШКІДЛИВХ ВЕБ-РЕСУРСІВ

3.1. Актуальність розробленої системи

Розробки системи виявлення шкідливих веб-ресурсів у сучасних умовах стрімкого розвитку Інтернету та цифрових технологій є надзвичайно важливою. Зі зростанням кількості веб-сайтів та глобалізацією доступу до інформації, проблема кіберзагроз, таких як фішинг, шкідливе програмне забезпечення, підроблені онлайн-магазини та сайти для крадіжки особистих даних, набуває все більшого масштабу. Зловмисники використовують веб-ресурси для обману користувачів, поширення вірусів, крадіжки конфіденційної інформації, а також для проведення фінансових шахрайств, що може призводити до значних фінансових збитків та загрози для персональних даних користувачів. Це створює потребу у розробці ефективних та автоматизованих систем для виявлення шкідливих ресурсів, здатних працювати в реальному часі та забезпечувати високий рівень захисту.

Традиційні підходи до захисту від шкідливих веб-ресурсів, такі як бази даних "чорних списків", де URL-адреси фіксуються як потенційно небезпечні, вже не можуть ефективно протидіяти сучасним загрозам. Основною проблемою таких систем є їхня статичність, оскільки вони потребують постійного оновлення баз даних, що може бути неефективним для нових, ще невідомих загроз. До того ж, зловмисники легко змінюють структуру URL, обходячи фільтри, засновані лише на сигнатурному аналізі. У цьому контексті актуальними стають системи, які базуються на методах машинного навчання, що дозволяють аналізувати не лише URL, але й низку інших характеристик, таких як структура посилання, довжина URL, ключові слова, рейтинг домену тощо.

Використання машинного навчання у системах виявлення шкідливих веб-ресурсів є сучасним підходом до забезпечення інформаційної безпеки.

Алгоритми аналізу даних, такі як Random Forest, Decision Tree, та інші, здатні навчатися на великих наборах даних, розпізнавати складні шаблони та здійснювати автоматичну класифікацію веб-ресурсів на шкідливі та безпечні. Це дозволяє системам працювати не лише з уже відомими загрозами, а й виявляти нові потенційно небезпечні ресурси на основі схожих ознак. Автоматизація такого підходу значно підвищує ефективність кіберзахисту, оскільки мінімізує потребу у ручному оновленні баз даних.

Актуальність розробки системи виявлення шкідливих веб-ресурсів також обумовлена зростаючою складністю загроз. Сучасні кібератаки часто мають складну багаторівневу структуру, де використовується маскування URL, шифрування даних, використання тимчасових доменів та підроблених сертифікатів безпеки. Тому системи, здатні враховувати комплекс ознак — від аналізу структури URL до перевірки його популярності в мережі, є необхідними для ефективного протистояння новим загрозам. Використання штучного інтелекту для аналізу великих обсягів даних дозволяє системам швидко реагувати на змінні загрози.

Ще одним важливим фактором, що підвищує актуальність даних систем, є посилення вимог до захисту персональних даних. Законодавчі акти, такі як GDPR (General Data Protection Regulation) в Європейському Союзі, зобов’язують компанії забезпечувати високий рівень захисту конфіденційної інформації. Шкідливі веб-ресурси становлять загрозу для витоку персональних даних користувачів, тому компанії та організації повинні впроваджувати автоматизовані системи для моніторингу та блокування підозрілих ресурсів.

Окрім захисту персональних даних, системи виявлення шкідливих веб-ресурсів є критично важливими для бізнесу та корпоративного сектору. Фінансові установи, онлайн-магазини та соціальні мережі часто стають мішенями для фішингових атак, що може призвести до втрати довіри клієнтів, фінансових збитків та репутаційних ризиків. Використання інтелектуальних систем захисту дозволяє автоматично аналізувати веб-трафік, блокувати

доступ до небезпечних ресурсів та попереджати користувачів про потенційні загрози.

Таким чином, розробка системи виявлення шкідливих веб-ресурсів є надзвичайно важливою у сучасних умовах зростання кіберзагроз. Вона дозволяє забезпечити надійний захист персональних даних, корпоративної інформації та фінансових активів. Інтеграція методів машинного навчання, таких як Random Forest та Decision Tree, підвищує ефективність та гнучкість системи, роблячи її здатною адаптуватися до нових викликів у сфері кібербезпеки.

3.2. Структура

Загальна структура проекту для розробки системи виявлення шкідливих веб-ресурсів поділяється на 3 частини:

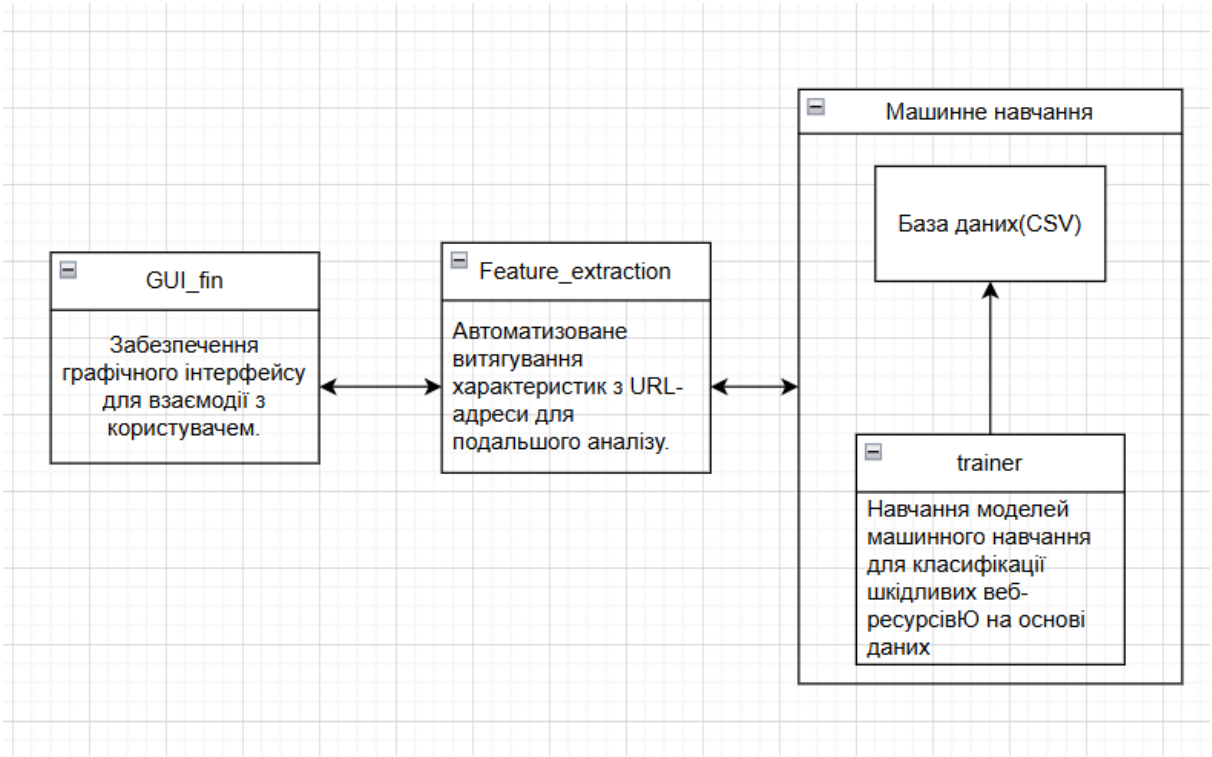

Рисунок 3.1 – Структура проекту

Перша частина відповідає за відображення інтерфейсу та його функціональність. Друга – відповідає за логіку переміщення даних, та за запити до компонентів, що відповідають за машинне навчання. Остання група це компоненти, що відповідають за машинне навчання, та навчають модель на основі бази даних.

3.3. Інструменти розробки

Для реалізації додатку використовувалась мова програмування Python, яка була обрана завдяки своїй універсальності, зручності у роботі з даними та наявності потужних бібліотек для машинного навчання та обробки текстової інформації.

Python забезпечує ефективний інструментарій для розробки як алгоритмічних компонентів, так і графічного інтерфейсу, що дозволило об'єднати всі етапи роботи системи — від збирання даних до візуалізації результатів аналізу. Різноманітні бібліотеки використовується для створення графічного інтерфейсу користувача, відображення результатів перевірки та взаємодії з компонентами машинного навчання [18].

Завдяки широкому набору бібліотек та активній підтримці спільноти, Python ідеально підходить для створення систем виявлення шкідливих ресурсів, дозволяючи розробляти як базові прототипи, так і масштабовані рішення для кібербезпеки, та створювати моделі для машинного навчання.

В даному проекті використовуються наступі алгоритми машинного навчання: Random Forest та Decision Tree.

Decision Tree Classifier — це алгоритм машинного навчання, який працює за принципом рекурсивного поділу даних на підгрупи для прийняття рішень. Він будує дерево, де кожен вузол представляє перевірку певної ознаки, а гілки — можливі значення цієї ознаки. Процес продовжується, поки дані не будуть повністю розділені на класи або не буде досягнуто обмеження за

глибиною дерева. Кожен листовий вузол відповідає прогнозованому класу [17].

Алгоритм працює на основі критеріїв поділу, таких як Gini Index або Entropy, які оцінюють ступінь "чистоти" вузлів. На кожному етапі вибирається ознака, яка найбільше зменшує невизначеність даних. Завдяки цьому Decision Tree добре справляється із задачами класифікації, де потрібно чітке розділення даних на категорії.

Набір параметрів Decision Tree Classifier включає:

- criterion: Функція для визначення поділу (gini, entropy).
- max_depth: Максимальна глибина дерева для контролю складності моделі.
- min_samples_split: Мінімальна кількість зразків для розбиття вузла.
- min_samples_leaf: Мінімальна кількість зразків у листовому вузлі.

Random Forest — це алгоритм машинного навчання, який використовує множину дерев рішень для вирішення задач класифікації та регресії. Принцип роботи алгоритму заснований на створенні великої кількості незалежних дерев рішень, кожне з яких навчається на випадковій підмножині даних із вибірковим набором ознак. В результаті прогнозування здійснюється шляхом голосування більшості дерев (у класифікації) або усереднення результатів (у регресії). Такий підхід підвищує стійкість до переобучення та дозволяє досягти більшої стабільності у порівнянні з окремими деревами рішень [16].

Основною перевагою Random Forest є здатність ефективно обробляти великі обсяги даних, включаючи набори із великою кількістю ознак, а також забезпечення високої точності прогнозування завдяки використанню ансамблю моделей. Алгоритм також здатен оцінювати важливість ознак, що дозволяє визначити, які фактори найбільше впливають на результат класифікації або регресії. Проте, через складність ансамблевої структури, модель може бути менш інтерпретованою порівняно з окремим деревом

рішень, а також вимагати значних обчислювальних ресурсів при роботі з великими наборами даних.

Набір параметрів Random Forest включає:

- n_estimators: Кількість дерев у лісі.
- max_depth: Максимальна глибина кожного дерева.
- criterion: Функція оцінки якості розбиття (gini або entropy).
- min_samples_split: Мінімальна кількість зразків для поділу вузла.

Набір даних, що використовується у даному проекті, був наданий платформою OpenPhish (рис. 3.2), яка спеціалізується на автоматичному виявленні фішингових ресурсів у реальному часі. OpenPhish є провідним сервісом у сфері кібербезпеки, який використовує сучасні алгоритми аналізу веб-контенту для моніторингу та виявлення ресурсів, що імітують легітимні веб-сайти. Основною метою таких ресурсів є отримання несанкціонованого доступу до конфіденційної інформації користувачів, зокрема облікових даних, паролів, банківських реквізитів та іншої приватної інформації. Виявлені шкідливі ресурси автоматично додаються до бази даних OpenPhish, що робить її цінним джерелом даних для систем виявлення кіберзагроз.

Рисунок 3.2 – Структура проекту

Однією з ключових переваг набору даних від OpenPhish є його актуальність. Дані про фішингові ресурси оновлюються щоденно, що забезпечує відображення поточних загроз у кіберпросторі. Це особливо важливо для систем машинного навчання, які потребують найсвіжіших даних для ефективного розпізнавання нових загроз. Фішингові атаки постійно

еволюціонують, і використання застарілих даних може знизити ефективність моделей класифікації. Завдяки регулярним оновленням, база даних OpenPhish забезпечує високу релевантність інформації, що використовується для навчання моделей.

Ще одним важливим аспектом є обширність набору даних. OpenPhish надає великий обсяг прикладів фішингових ресурсів, що дозволяє тренувати моделі машинного навчання на широкому спектрі зразків. Це включає як різноманітні типи URL-адрес, так і характеристики, які можуть свідчити про шкідливість ресурсу, такі як структура посилання, довжина URL, використання підозрілих ключових слів або наявність IP-адрес у домені. Використання різноманітних прикладів допомагає створити збалансований навчальний набір, який знижує ризик переобучення моделі на обмежених даних.

3.4. Розробка системи

Завантаження проекту відбувається з функції submitCallBack (рис. 3.3). Вона створює графічний інтерфейс користувача (GUI) для застосунку на Python, використовуючи бібліотеку tkinter разом із розширенням ttkthemes. Основне вікно програми створюється за допомогою ThemedTk з темою "radiance", що надає більш сучасний вигляд елементам інтерфейсу. Вікно отримує заголовок "URL Detector" і фіксований розмір 700x500 пікселів. Основна функція програми — перевірка введеного користувачем URL-адресу за допомогою локального файлу url_features.csv.

```
from tkinter import *
import tkinter as tk
import tkinter.messagebox as tkMessageBox
from ttkthemes import ThemedTk
from tkinter import messagebox
from tkinter import ttk
import trainer as tr
import webbrowser
import main
import Feature_extraction
import csv

def submitCallBack():
    url = entry_url.get()
    result_text.delete(1.0, END)
    result_text.insert(END, f"Запуск головного модуля...\nПрацюємо над: {url}\n")
```

Рисунок 3.3 – Створення інтерфейсу

Користувач може ввести URL (рис. 3.4) у текстове поле, яке розташоване у елементі frame. Мітка entry_label показує інструкцію "Введіть URL:", а саме текстове поле для введення (ttk.Entry) дозволяє вводити URL-адресу довжиною до 60 символів. Під текстовим полем розташоване багаторядкове текстове поле result_text для відображення результатів перевірки. Це поле має сірий фон (#F5F5F5), чорний текст та шрифт Courier New для зручнішого форматування виведених даних. Для підсвічування небезпечних URL використовується спеціальний тег "danger" з червоним кольором тексту.

```
entry_label = ttk.Label(frame, text="Введіть URL:", font=("Helvetica", 12))
entry_label.grid(row=0, column=0, padx=5, pady=5)

entry_url = ttk.Entry(frame, width=60, font=("Arial", 12))
entry_url.grid(row=0, column=1, padx=5, pady=5)

# Текстове поле для результатів
result_text = Text(root, height=15, width=80, font=("Courier New", 12), bg="#F5F5F5
result_text.pack(pady=10)

# Додаємо кольоровий тег для небезпечних URL
result_text.tag_configure("danger", foreground="red")

# Кнопки
button_frame = ttk.Frame(root)
button_frame.pack(pady=10)

submit_button = ttk.Button(button_frame, text="Опрацювати", command=submitCallBack,
```

Рисунок 3.4 – Елементи інтерфейсу

Функція submitCallBack (рис. 3.5) активується при натисканні кнопки "Опрацювати". Вона зчитує введений URL, очищає попередні результати у полі result_text і додає повідомлення про початок перевірки. Якщо URL містить «.ru», додається повідомлення про потенційно небезпечний домен країни-агресора. Програма відкриває файл url_features.csv і перевіряє, чи знайдено введений URL у базі даних. Якщо URL присутній у файлі, виводяться такі характеристики, як кількість токенів у домені (token_count), довжина URL (Length_of_url) та рейтинг хосту (rank_host). Якщо значення rank_host дорівнює -1, URL вважається небезпечним, і текст відображається червоним кольором із попереджувальним повідомленням. Якщо URL безпечний, з'являється зелене повідомлення про безпеку. Якщо введений URL не знайдено у файлі, програма повідомляє, що запис відсутній у базі даних.

```
def submitCallBack():
    url = entry_url.get()
    result_text.delete(1.0, END)
    result_text.insert(END, f"Запуск головного модуля...\nПрацюємо над: {url}\n")

    if '.ru' in url:
        result_text.insert(END, "⚠ Домен КРАЇНИ АГРЕСОРА.\n")

    try:
        with open('url_features.csv', 'r') as file:
            reader = csv.DictReader(file)
            is_dangerous = False
            for row in reader:
                if row['URL'] == url:
                    is_dangerous = row['rank_host'] == '-1'
                    result_text.insert(END, f"Token Count: {row.get('token_count', 'N/A')}\n")
                    result_text.insert(END, f"Length of URL: {row.get('Length_of_url', 'N/A')}\n")

                    if is_dangerous:
                        result_text.insert(END, "❌ НЕБЕЗПЕЧНА URL адреса!\n", "danger")
                    else:
                        result_text.insert(END, "✅ БЕЗПЕЧНА URL адреса\n")
                    break
            else:
                result_text.insert(END, "❓ URL не знайдено у базі даних\n")
    except FileNotFoundError:
        messagebox.showerror("Помилка", "Файл 'url_features.csv' не знайдено")
```

Рисунок 3.5 – Функція submitCallBack

Крім функції перевірки, передбачено також функцію about(), яка відкриває діалогове вікно з інформацією про розробника. Ця функція

активується при натисканні кнопки "Довідка". Ще одна кнопка "Вийти" завершує виконання програми через метод root.quit.

Кнопки та стилі оформлення налаштовані через бібліотеку ttk з використанням кастомного стилю "Accent.TButton". Цей стиль визначає білі літери на синьому фоні (#0078D7) із зміною кольору на темніший при наведенні (#005A9E). Функція ttk.Style() застосовується для створення цього стилю, а кнопки згруповані у фреймі button_frame для кращого управління простором.

```
submit_button = ttk.Button(button_frame, text="Опрацювати", command=subm
submit_button.pack(side=LEFT, padx=10)

about_button = ttk.Button(button_frame, text="Довідка", command=about)
about_button.pack(side=LEFT, padx=10)

exit_button = ttk.Button(button_frame, text="Вийти", command=root.quit)
exit_button.pack(side=LEFT, padx=10)
```

Рисунок 3.6 – Оформлення кнопок

Код організований, але включає базові принципи модульності та стилізації через ttkthemes для покращення візуального вигляду.

Наступний код виконує функції навчання та класифікації даних для виявлення шкідливих URL-адрес за допомогою алгоритмів машинного навчання Decision Tree (DST) та Random Forest. Основне призначення коду — аналіз даних, отриманих із CSV-файлів, проведення навчання моделей, їх оцінка та прогнозування результатів на нових даних. Він використовується як компонент у більшому додатку, де класифікатор викликається через графічний інтерфейс tkinter.

Спочатку імпортуються необхідні бібліотеки (рис. 3.7), включаючи pandas для обробки табличних даних, scikit-learn для машинного навчання, numpy для числових обчислень та matplotlib для візуалізації. Вмикається ігнорування попереджень для застарілих функцій pandas.

```
import pandas
import pandas as pd
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import numpy
from DecisionTreeClassifier import dst
from sklearn.model_selection import cross_validate as cv
import matplotlib.pylab as plt
import warnings
from sklearn.metrics import confusion_matrix

warnings.filterwarnings("ignore", category=DeprecationWarning, module="pandas", lineno=570)
```

Рисунок 3.7 – Імпорт бібліотек

Функція return_nonstring_col (рис. 3.8) обробляє вхідні стовпці даних, залишаючи лише числові дані, придатні для використання у моделях машинного навчання. Вона виключає текстові поля (URL, host, path) та окремо зберігає стовпці, які будуть використовуватись для навчання (train_cols).

Функція dst_classifier (рис. 3.8) реалізує класифікатор на основі DST (Decision Tree). Вона створює модель SVC, нормалізує дані для навчання (train) та тестування (query) за допомогою preprocessing.scale. Після навчання моделі вона виконує перехресну перевірку (cross_val_score) для оцінки якості моделі та прогнозує результати для тестових даних, зберігаючи результати в колонці result.

```
def return_nonstring_col(data_cols):
    cols_to_keep = []
    train_cols = []
    for col in data_cols:
        if col != 'URL' and col != 'host' and col != 'path':
            cols_to_keep.append(col)
            if col != 'malicious' and col != 'result':
                train_cols.append(col)
    return [cols_to_keep, train_cols]


def dst_classifier(train, query, train_cols):
    clf = dst.SVC()

    train[train_cols] = preprocessing.scale(train[train_cols])
    query[train_cols] = preprocessing.scale(query[train_cols])

    print (clf.fit(train[train_cols], train['malicious']))
    scores = cv.cross_val_score(clf, train[train_cols], train['malicious'], cv=30)
    print('Estimated score DST: %0.5f (+/- %0.5f)' % (scores.mean(), scores.std() / 2))

    query['result'] = clf.predict(query[train_cols])

    print(query[['URL', 'result']])
```

Рисунок 3.8 – Функції return_nonstring_col, dst_classifier

Функції forest_classifier та forest_classifier_gui (рис. 3.9) реалізують класифікацію за допомогою Random Forest. RandomForestClassifier використовується з параметром n_estimators=150 для створення ансамблю з 150 дерев рішень. Функція forest_classifier включає також перехресну перевірку для оцінки якості моделі, тоді як forest_classifier_gui спрощена для використання у графічному інтерфейсі і повертає лише результати передбачень.

Функція train здійснює основний процес навчання моделей. Вона зчитує дані з файлів CSV (train_db і test_db), виділяє числові стовпці за допомогою return_nonstring_col і передає їх до функцій dst_classifier та forest_classifier. Це дозволяє виконувати навчання та тестування одночасно з використанням обох моделей машинного навчання.

```
# Called from gui
def forest_classifier_gui(train, query, train_cols):
    rf = RandomForestClassifier(n_estimators=150)

    print(rf.fit(train[train_cols], train['malicious']))

    query['result'] = rf.predict(query[train_cols])

    print(query[['URL', 'result']].head(2))
    return query['result']


def forest_classifier(train, query, train_cols):
    rf = RandomForestClassifier(n_estimators=150)

    print(rf.fit(train[train_cols], train['malicious']))
    scores = cv.cross_val_score(rf, train[train_cols], tr
    print('Estimated score RandomForestClassifier: %0.5f

    query['result'] = rf.predict(query[train_cols])
    print(query[['URL', 'result']])
```

Рисунок 3.9 – Функції forest_classifier та forest_classifier_gui

Функція gui_caller використовується для спрощеного виклику моделі Random Forest із графічного інтерфейсу. Вона читає CSV-файли, обробляє дані та повертає результат класифікації, який потім може бути використаний у tkinter.

В кінці скрипту додається тестове повідомлення "trainer is running ...", яке свідчить про активацію файлу. В цілому, цей модуль відповідає за навчання та прогнозування шкідливих URL у застосунку, використовуючи два класифікатори — DST і Random Forest.

Цей Python-копмпонент (рис. 3.10) виконує функцію підготовки даних для навчання моделей машинного навчання у додатку для класифікації шкідливих URL-адрес. Він зчитує URL-адреси з файлів, обробляє їх, витягує ключові характеристики, зберігає ці характеристики у CSV-файлі та викликає модуль trainer для навчання моделей. Основною метою цього скрипту є попередня обробка даних для подальшого використання у класифікаторах DST та Random Forest.

```
import csv
import Feature_extraction as urlfeature
import trainer as tr


def resultwriter(rowdict, output_dest):
    with open(output_dest, 'w', newline='') as f:
        fieldnames = rowdict[0].keys()  # Вибираємо ключі
        writer = csv.DictWriter(f, fieldnames=fieldnames)
```

Рисунок 3.10 – Компонент main

Функція resultwriter (рис. 3.11) відповідає за збереження даних у CSV-форматі. Вона приймає два аргументи: список словників rowdict (де кожен словник містить URL і витягнуті характеристики) та output_dest — шлях до файлу для збереження результатів. Функція відкриває файл у режимі запису (w) і використовує csv.DictWriter для збереження ключів першого словника як заголовків CSV-файлу. Перед записом значень перевіряється, чи є вони рядками, і якщо так, то вони кодуються у байтовий формат для уникнення помилок при обробці текстових даних.

```
def resultwriter(rowdict, output_dest):
    with open(output_dest, 'w', newline='') as f:
        fieldnames = rowdict[0].keys()  # Вибираємо ключі п
        writer = csv.DictWriter(f, fieldnames=fieldnames)

        writer.writeheader()
        for row in rowdict:
            # Конвертуємо значення в байтовий об'єкт, якщо
            for key, value in row.items():
                if isinstance(value, str):
                    row[key] = value.encode()
            writer.writerow(row)
```

Рисунок 3.11 – Функція resultwriter

Функція process_URL_list (рис. 3.12) обробляє список URL-адрес із файлу file_dest, кожен з яких містить URL та мітку (malicious або benign). Вона

відкриває файл, читає кожен рядок, витягує URL та мітку шкідливості, після чого передає URL у функцію feature_extract з модуля Feature_extraction. Ця функція витягує характеристики URL-адреси (наприклад, довжину URL, кількість токенів, кількість крапок тощо). Потім отримані дані зберігаються у списку feature у вигляді словників, а результати записуються у CSV-файл за допомогою resultwriter.

```
def process_URL_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.split(',')[0].strip()
            malicious_bool = line.split(',')[1].strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                ret_dict['malicious'] = malicious_bool
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)
```

Рисунок 3.12 – Функція process_URL_list

Функція process_test_list (рис. 3.13) має аналогічний функціонал, але використовується для обробки тестових даних, де вхідний файл file_dest містить лише URL-адреси без міток. Вона також використовує модуль Feature_extraction для обробки кожного URL та зберігає результати у форматі CSV за допомогою resultwriter.

Функція process_test_url (рис. 3.13) є спрощеною версією попередніх функцій і використовується для обробки окремого URL-адресу. Вона приймає URL-адресу як аргумент, обробляє її за допомогою feature_extract і зберігає результат у CSV-файлі.

```
def process_test_list(file_dest, output_dest):
    feature = []
    with open(file_dest) as file:
        for line in file:
            url = line.strip()
            if url != '':
                print
                'working on: ' + url
                ret_dict = urlfeature.feature_extract(url)
                feature.append([url, ret_dict]);
    resultwriter(feature, output_dest)


# change
def process_test_url(url, output_dest):
    feature = []
    url = url.strip()
    if url != '':
        print('working on: ' + url)  # showoff
        ret_dict = urlfeature.feature_extract(url)
        feature.append({'URL': url, 'Features': ret_dict})
    resultwriter(feature, output_dest)
```

Рисунок 3.13 – Методи process_test_list, process_test_url

Головна функція main (рис. 3.14) виконує основний робочий процес програми. Вона може бути розкоментована для обробки навчального набору даних (process_URL_list) або тестового набору даних (process_test_list). Після підготовки CSV-файлів функція викликає модуль trainer (tr.train) для навчання моделей машинного навчання. Вона двічі викликає функцію train: перший раз для навчання на повному наборі даних, а другий раз для перевірки класифікаторів на тестовому наборі даних (query_features.csv).

```
def main():
    #process_URL_list('URL.txt', 'url_features.csv')
    process_test_list("query.txt",'query_features.csv')
    tr.train('url_features.csv', 'url_features.csv')  # 
    tr.train('url_features.csv', 'query_features.csv')
    # testing with urls in query.txt
```

Рисунок 3.14 – Функція main

У підсумку, цей скрипт виконує роль попередньої обробки даних у додатку. Він автоматизує процес збору характеристик URL, їх підготовку для навчання та подальше тестування моделей. Це важливий компонент додатка, оскільки якість витягнутих характеристик і правильна підготовка даних безпосередньо впливають на ефективність класифікаторів машинного навчання.

Наступний компонент виконує функцію витягування характеристик URL-адрес для їх подальшого використання у машинному навчанні з метою класифікації шкідливих веб-ресурсів. Він є ключовим компонентом додатку, оскільки саме на основі цих характеристик навчаються моделі Random Forest та DST для визначення потенційно небезпечних URL.

Код починається з імпорту необхідних бібліотек (рис. 3.15), включаючи urllib для роботи з URL-адресами, re для регулярних виразів, pygeoip для визначення геолокації за IP-адресами, а також csv для обробки даних у табличному форматі. Встановлюється користувацький User-Agent, який дозволяє програмі імітувати запит від веб-браузера, щоб уникнути блокувань серверів при зверненні до зовнішніх ресурсів, наприклад, Alexa Rank.

```
from urllib.parse import urlparse
import re
import urllib.request
from xml.dom import minidom
import csv
import pygeoip as pygeoip


opener = urllib.request.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
print(opener)

nf = -1
```

Рисунок 3.15 – Імпорту необхідних бібліотек

Функція Tokenise (рис. 3.16) відповідає за розбивку URL-адреси на окремі токени (слова), обчислення їхньої середньої довжини, кількості токенів та довжини найдовшого токена. Це дозволяє використовувати ці дані як характеристики для навчання моделі, оскільки довжина URL та складність його структури можуть бути ознаками шкідливості.

```
def Tokenise(url):
    if url == '':
        return [0, 0, 0]
    token_word = re.split('\W+', url)
    no_ele = sum_len = largest = 0
    for ele in token_word:
        l = len(ele)
        sum_len += l
        if l > 0:
            no_ele += 1
        if largest < l:
            largest = l
    try:
        return [float(sum_len) / no_ele, no_ele, largest]
    except:
        return [0, no_ele, largest]
```

Рисунок 3.16 – Функція Tokenise

Функція sitepopularity (рис. 3.17) виконує перевірку популярності сайту через Alexa Rank. Вона надсилає запит до http://data.alexa.com і отримує XML-відповідь із ранжуванням сайту у глобальному та локальному масштабі. Для парсингу XML використовується бібліотека minidom. Якщо рейтинг недоступний, функція повертає значення -1 як індикатор відсутності даних.

```
def sitepopularity(host):
    xmlpath = 'http://data.alexa.com/data?cli=10&dat=snbamz&url=' + host
    try:
        xml = urllib.request.urlopen(xmlpath)
        dom = minidom.parse(xml)
        rank_host = find_ele_with_attribute(dom, 'REACH', 'RANK')
        rank_country = find_ele_with_attribute(dom, 'COUNTRY', 'RANK')
        return [rank_host, rank_country]
    except:
        return [nf, nf]
```

Рисунок 3.17 – Функція sitepopularity

Security_sensitive перевіряє, чи містить URL потенційно небезпечні ключові слова, такі як login, secure, account, banking, що можуть свідчити про фішингові атаки або підроблені сторінки для збору конфіденційної інформації.

Функція exe_in_url (рис. 3.18) перевіряє, чи містить URL файл із розширенням .exe, що може бути показником шкідливого ПЗ. Check_IPaddress визначає, чи містить URL IP-адресу замість доменного імені, оскільки використання IP часто характерне для шкідливих сайтів.

getASN (Autonomous System Number) використовує бібліотеку pygeoip для отримання автономної системи (ASN), до якої належить домен. ASN дозволяє визначити, до якого провайдера або мережі належить ресурс, що може бути індикатором ризику (рис. 3.18).

```
def exe_in_url(url):
    if url.find('.exe') != -1:
        return 1
    return 0

def getASN(host):
    try:
        g = pygeoip.GeoIP('GeoIPASNum.dat')
        asn = int(g.org_by_name(host).split()[0][2:])
        return asn
    except:
        return nf
```

Рисунок 3.18 – Функції exe_in_url, getASN

Функція safebrowsing здійснює перевірку URL за допомогою API Google Safe Browsing, який може ідентифікувати URL-адреси, що містять фішингові або шкідливі елементи. У разі підозрілої активності, API повертає код 200, що сигналізує про небезпеку. Код 204 свідчить про безпеку сайту.

Основною функцією коду є feature_extract (рис. 3.19). Вона приймає URL як вхідний параметр, розбиває його на компоненти за допомогою urlparse, зберігає хост та шлях, а потім викликає інші допоміжні функції для збору

характеристик: довжина URL, кількість крапок, рейтинг сайту, наявність IP-адреси, небезпечні ключові слова, ASN та результат перевірки Google Safe Browsing. Якщо домен належить країні-агресору (містить .ru), це також фіксується у результатах. Усі характеристики збираються у словник Feature та повертаються для подальшого використання у процесі навчання моделей.

```
def feature_extract(url_input):
    Feature = {}
    tokens_words = re.split('\W+', url_input)

    obj = urlparse(url_input)
    host = obj.netloc
    path = obj.path

    Feature['URL'] = url_input
    Feature['rank_host'], Feature['rank_country'] = sitepopularity(host)
    Feature['host'] = obj.netloc
    Feature['path'] = obj.path
    Feature['Length_of_url'] = len(url_input)
    Feature['Length_of_host'] = len(host)
    Feature['No_of_dots'] = url_input.count('.')
    Feature['avg_token_length'], Feature['token_count'], Feature['larges
    Feature['avg_domain_token_length'], Feature['domain_token_count'], F
    Feature['avg_path_token'], Feature['path_token_count'], Feature['lar
    Feature['sec_sen_word_cnt'] = Security_sensitive(tokens_words)
    Feature['IPaddress_presence'] = Check_IPaddress(tokens_words)

    # Перевіряємо, чи є .ru в хості
    if '.ru' in host:
        print("Домен КРАЇНИ АГРЕСОРА")
```

Рисунок 3.19 – Функція feature_extract

Даний код виконує автоматизоване збирання характеристик для подальшої класифікації веб-ресурсів як шкідливих або безпечних.

3.5. Тестування додатку

Для ефективного тестування додатка, який класифікує URL-адреси за рівнем безпеки, необхідно перевірити його здатність коректно обробляти вхідні дані, виконувати витягування характеристик URL та забезпечувати

надійність моделей машинного навчання при виявленні потенційно шкідливих ресурсів.

Спочатку потрібно завантажити додаток. Перед користувачем повинне відритися наступне вікно (рис. 3.20):

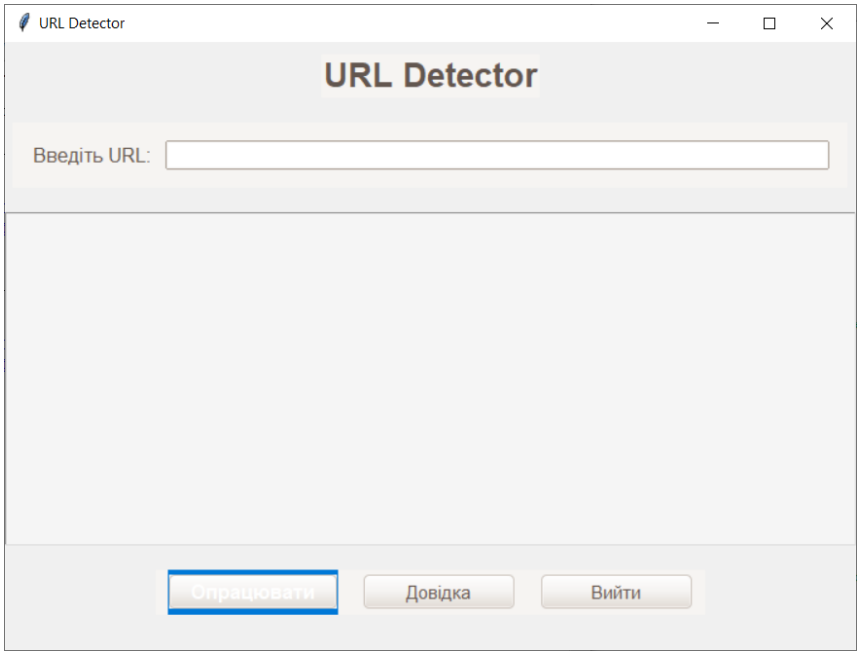

Рисунок 3.20 – Вікно додатку

Далі користувачу потрібно ввести будь-яку URL-адресу, для перевірки сайту. Для прикладу перевіримо адрес Google (рис. 3.21):

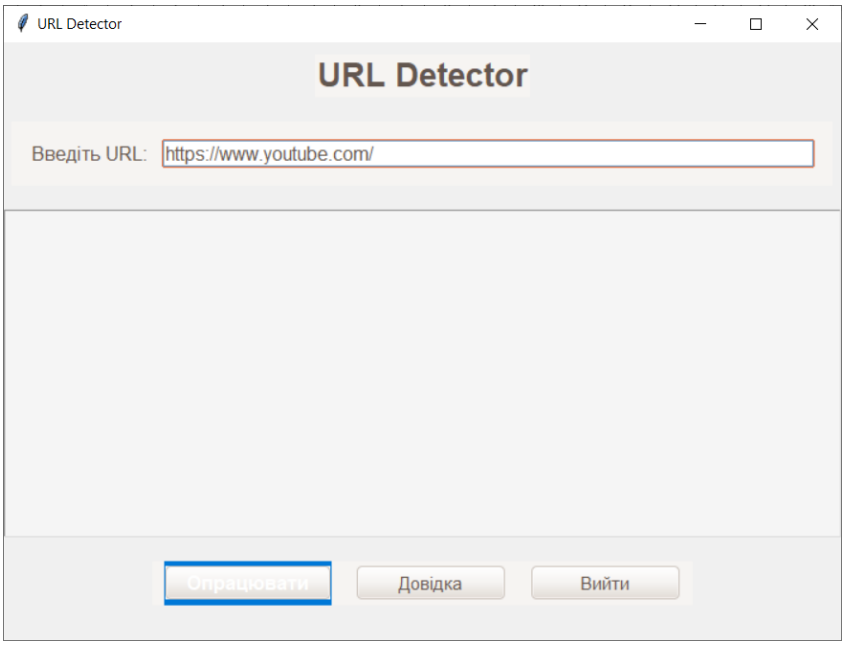

Рисунок 3.21 – Введення Url-адреси

Після цих дій потрібно натиснути на кнопку «Опрацювати». Перед користувачем на екрані, з’явиться загальний висновок про цей адрес (рис. 3.22).

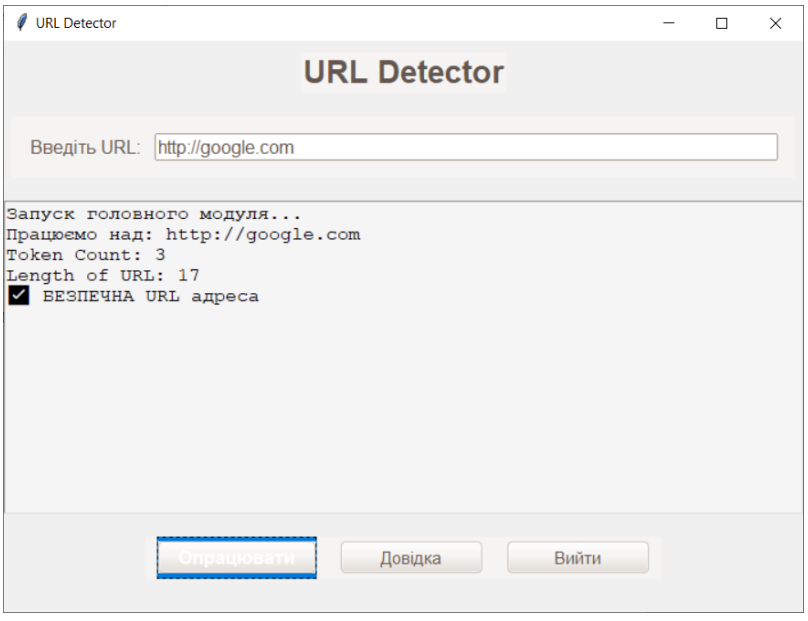

Рисунок 3.22 – Виведення висновку після перевірки

Після опрацювання адреси, програма виводить висновок в якому зазначається, що дана адреса безпечна для перегляду. Далі спробуємо адресу, що за кінчується на «.ru» (рис. 3.23).

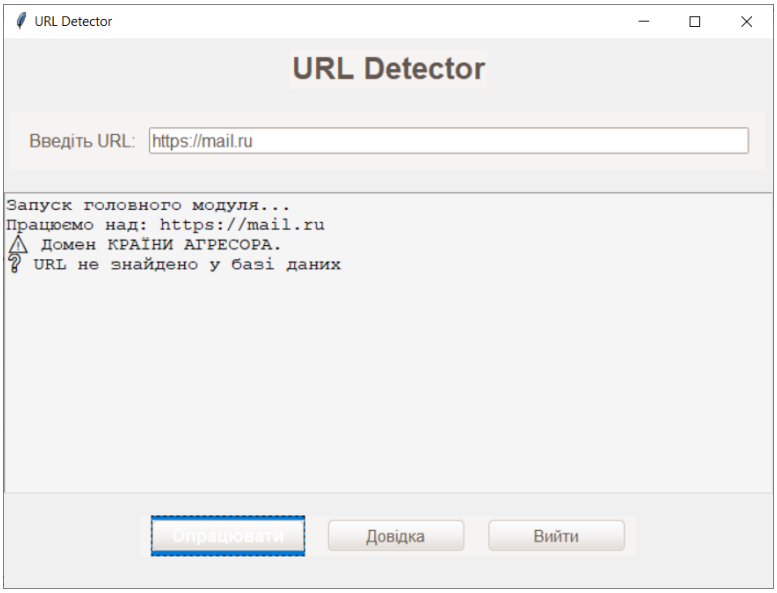

Рисунок 3.23 – Виведення повідомлення про небезпеку

В результаті отримано повідомлення про небезпеку, адже домен відноситься до країни агресора, це може бути небезпечним тому не варто переглядати данні сайти.

3.6. Висновок до третього розділу

Актуальність розробки системи виявлення шкідливих веб-ресурсів обумовлена зростаючою складністю загроз. Сучасні кібератаки часто мають складну багаторівневу структуру, де використовується маскування URL, шифрування даних, використання тимчасових доменів та підроблених сертифікатів безпеки. Тому системи, здатні враховувати комплекс ознак — від аналізу структури URL до перевірки його популярності в мережі, є необхідними для ефективного протистояння новим загрозам. Використання штучного інтелекту для аналізу великих обсягів даних дозволяє системам швидко реагувати на змінні загрози.

В результаті розроблений комплексний програмний продукт для виявлення шкідливих веб-ресурсів, який поєднує модулі графічного інтерфейсу, збору та обробки даних, а також машинного навчання. Система дозволяє автоматично аналізувати URL-адреси, витягувати з них ключові характеристики, такі як довжина URL, кількість токенів, наявність підозрілих ключових слів та рейтинги домену, а також навчати та тестувати моделі Random Forest і DST для класифікації веб-ресурсів як шкідливих або безпечних. Використання актуального набору даних із платформи OpenPhish забезпечує високу якість навчання моделей, а модульний підхід до розробки сприяє легкому масштабуванню та оновленню системи. Створена система є ефективним інструментом для аналізу загроз у сфері кібербезпеки, який може бути використаний як для навчальних, так і для дослідницьких цілей.

ВИСНОВОК

Веб-браузер доступний для всіх, хто має підключення до інтернету, включаючи зловмисників. На жаль, розробники часто нехтують принципами безпеки веб-додатків. Команди здебільшого настільки зосереджені на функціональних аспектах, таких як програмування, дизайн інтерфейсу та зручність використання, що забувають приділити увагу стабільності та захищеності програмного продукту.

Стрімкий розвиток веб-технологій та поширення Інтернету значно збільшує кількість загроз для інформаційної безпеки, зокрема через наявність шкідливих веб-ресурсів, фішингових атак та поширення зловмисного програмного забезпечення.

Традиційні методи виявлення загроз мають обмежену ефективність через статичність підходу та нездатність ідентифікувати ще не виявлені загрози. Це створює потребу у використанні сучасних методів аналізу даних, зокрема алгоритмів машинного навчання, які здатні автоматизувати процес ідентифікації небезпечних ресурсів шляхом аналізу структурних, лексичних та поведінкових ознак URL-адрес.

Актуальність розробки системи виявлення шкідливих веб-ресурсів також обумовлена зростаючою складністю загроз. Сучасні кібератаки часто мають складну багаторівневу структуру, де використовується маскування URL, шифрування даних, використання тимчасових доменів та підроблених сертифікатів безпеки. Тому системи, здатні враховувати комплекс ознак — від аналізу структури URL до перевірки його популярності в мережі, є необхідними для ефективного протистояння новим загрозам. Використання штучного інтелекту для аналізу великих обсягів даних дозволяє системам швидко реагувати на змінні загрози.

У результаті проведеного дослідження було розроблено та реалізовано систему виявлення шкідливих веб-ресурсів із використанням методів машинного навчання. Було проведено комплексний аналіз загроз у мережі

Інтернет, включаючи фішингові атаки, шкідливі URL-адреси та поширення зловмисного програмного забезпечення. Розглянуто існуючі методи захисту, зокрема використання чорних списків, та обґрунтовано їх обмеження при виявленні нових загроз.

Особливу увагу було приділено використанню алгоритмів машинного навчання для автоматизації процесу класифікації веб-ресурсів. У рамках дослідження проведено порівняльний аналіз платформ для збору навчальних даних (PhishTank, URLhaus, VirusTotal, OpenPhish). Обґрунтовано вибір платформи OpenPhish завдяки її автоматизованому оновленню даних у реальному часі, що дозволяє зберігати актуальність інформації.

Для побудови системи було використано бібліотеку Scikit-Learn, що забезпечує зручний інструментарій для роботи з класичними алгоритмами машинного навчання. Обрані моделі Random Forest та Decision Tree показали високу точність у виявленні шкідливих веб-ресурсів за рахунок можливості обробки великих обсягів даних та ефективності роботи зі структурованими ознаками URL-адрес.

Розроблений комплексний програмний продукт для виявлення шкідливих веб-ресурсів, який поєднує модулі графічного інтерфейсу, збору та обробки даних, а також машинного навчання. Система дозволяє автоматично аналізувати URL-адреси, витягувати з них ключові характеристики, такі як довжина URL, кількість токенів, наявність підозрілих ключових слів та рейтинги домену, а також навчати та тестувати моделі Random Forest і DST для класифікації веб-ресурсів як шкідливих або безпечних. Використання актуального набору даних із платформи OpenPhish забезпечує високу якість навчання моделей, а модульний підхід до розробки сприяє легкому масштабуванню та оновленню системи. Створена система є ефективним інструментом для аналізу загроз у сфері кібербезпеки, який може бути використаний як для навчальних, так і для дослідницьких цілей.

У результаті виконано такі етапи дослідження:

1. Проведено аналіз методів виявлення шкідливих веб-ресурсів.

2. Визначено вимоги до функціональності та ефективності системи.

3. Обрано джерело даних для навчання моделей та бібліотеку для реалізації алгоритму.

4. Реалізовано програмну модель на основі алгоритмів Random Forest та Decision Tree.

5. Виконано тестування системи, яке підтвердило її ефективність у виявленні потенційно небезпечних ресурсів.

Вхідними даними для системи є набір URL-адрес, який автоматично аналізується на основі структурних, лексичних та поведінкових ознак. Результатом роботи системи є автоматична класифікація ресурсів як шкідливих або безпечних.

Таким чином, розроблена система забезпечує ефективне виявлення загроз у веб-просторі, відповідає сучасним вимогам кібербезпеки та може бути використана для подальшого вдосконалення алгоритмів аналізу та розробки комплексних рішень для захисту інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J.Jang-Jaccard and S. Nepal, “A survey of emerging threats in cybersecurity,” Journal of Computer and System Sciences, vol. 80, no. 5, pp. 973–993, 2014.

2. Rajapaksha, S., Senanayake, J., Kalutarage, H., & Al-Kadri, M. O. (2024). Enhancing security assurance in software development: AI-based vulnerable code detection with static analysis. In S. Katsikas et al. (Eds.), Computer Security. ESORICS 2023 International Workshops (Vol. 14399, p. 20). Cham: Springer. doi:10.1007/978-3-031-54129-2_20

3. Seshapriyan, T., Dinesh, S. M., & Godwin Ponsam, J. (2024). SentinelScan: Advanced network scanner and packet detection suite. In 2024 8th International Conference on Inventive Systems and Control (ICISC) (pp. 253–258). Coimbatore, India. doi:10.1109/ICISC62624.2024.00050

4. Singh, R., Kumar Gupta, M., Patil, D. R., & Patil, S. M. (2024). Analysis of web application vulnerabilities using dynamic application security testing. In 2024 IEEE 9th International Conference for Convergence in Technology (I2CT) (pp. 1–6). Pune, India. doi:10.1109/I2CT61223.2024.10543484

5. Barocsai, M., Can, J., Karresand, M., & Nadjm-Tehrani, S. (2024). Mapping and analysis of common vulnerabilities in popular web servers. In S. Pickl, B. Hämmerli, P. Mattila, & A. Sevillano (Eds.), Critical Information Infrastructures Security. CRITIS 2023 (Vol. 14599, p. 1). Cham: Springer. doi:10.1007/978-3-031-62139-0_1

6. DoS Attack vs. DDoS Attack. – [Електронний ресурс] – Режим доступу: https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos

7. Packet Sniffing: Types, Methods, Examples, and Best Practices. – [Електронний ресурс] – Режим доступу: https://www.knowledgehut.com/blog/security/packet-sniffing

8. Phishtank. Developer Information – [Електронний ресурс] – Режим доступу: https://phishtank.org/developer_info.php

9. URLhaus Database – [Електронний ресурс] – Режим доступу: https://urlhaus.abuse.ch/browse/

10. VirusTotal API v3 Overview – [Електронний ресурс] – Режим доступу: https://docs.virustotal.com/reference/overview

11. Websites 101: Anatomy of a URL – [Електронний ресурс] – Режим доступу: https://www.infotex.uk/websites-101-anatomy-of-a-url/

12. OpenPhish Database – https://openphish.com/phishing_database.html

13. Documentation of scikit-learn – [Електронний ресурс] – Режим доступу: https://scikit-learn.org/0.21/documentation.html

14. TensorFlow. Models & datasets – [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/resources/models-datasets

15. PyTorch documentation – https://pytorch.org/docs/stable/index.html

16. Random Forest – [Електронний ресурс] – Режим доступу: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html

17. DecisionTreeClassifier – [Електронний ресурс] – Режим доступу: https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeClassifier.html

18. Python 3.13.1 documentation – [Електронний ресурс] – Режим доступу: https://docs.python.org/3/