Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики

Факультету інформатики

Текстова частина до кваліфікаційної роботи за спеціальністю 122 «Комп'ютерні науки»

Автоматизована система аналізу захищеності Web застосунків за допомогою OSINT

Керівник курсової роботи
ст. викл. Вознюк Я. І.

_____

(підпис)

"____"_____2025 р.

Виконав студент
4-го року навчання спеціальності
122 "Комп'ютерні науки"
Галиць Владислав
Олександрович
"____"_____2025 р

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
доц., канд.ф.-м.н.
____С. С. Гороховський

„_____” ____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу

студенту 4-го курсу факультету інформатики курсу
Галицю Владиславу Олександровичу
ТЕМА: **Автоматизована система аналізу захищеності Web застосунків за допомогою OSINT**

Зміст ТЧ до кваліфікаційної роботи:
Індивідуальне завдання

Календарний план

Анотація

Вступ

1. Огляд основних типів вразливостей

2. Опис основних технологій і понять предметної області неконтактної

безпеки

3. Проектування системи аналізу захищеності на основі OSINT

4. Реалізація системи аналізу захищеності веб-застосунків

Висновки

Список літератури

Дата видачі «__»_____2025 р.

Керівник _____(підпис)

Завдання отримав_____(підпис)

Тема: Автоматизована система аналізу захищеності Web-застосунків за допомогою OSINT

Календарний план виконання роботи:

Номер	Назва етапу курсової	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу.	жовтень	
2.	Огляд технічної літератури за темою роботи.	листопад	
3.	Аналіз методів для виявлення вразливостей.	грудень-січень	
4.	Аналіз типів вразливостей.	лютий	
5.	Вибір інструментів для побудови застосунку.	березень	
6.	Створення модулів для тестування різних типів вразливостей на сервері.	кінець березня	
8.	Текстове оформлення результатів роботи.	квітень	
9.	Попередній аналіз курсової. Виправлення помилок.	травень	
10.	Захист кваліфікаційної роботи.	травень	

Студент Галиць В. О.

Керівник Вознюк Я. І.

“_____” _____

Зміст

Перелік використаних скорочень

API - Application Programming Interface

OSINT – Open Source Intelligence

HTTP – Hypertext Transfer Protocol

TCP - Transmission Control Protocol

APT – Advanced Persistent Threat

IaaS – Infrastructure as a Service

PaaS – Platform as a Service

DNS – Domain Name System

SSR – Server-Side Rendering

Анотація

Метою цієї роботи є дослідження характеристик інфраструктури, які можуть вказувати на можливі вразливості у системах. Такі індикатори, зазвичай, використовуються для побудови векторів атаки на інфраструктури компаній та є першим кроком до компрометації. Дипломна робота базується на аналізі індикаторів та побудові системи для сканування своїх ресурсів з метою виявлення вразливих компонентів. За допомогою отриманої інформації , буде побудовано систему для відслідковування вразливостей як у мережевому секторі ,так і моніторинг витоку даних користувачів досліджуваної системи.

Вступ

В сучасних умовах світу, коли більшість приватних та державних установ надають перевагу автоматизованим системам для виконання задач, зростає попит на захист своєї інфраструктури. За дослідженням Verizon DBIR, статистика вказує на те, що більше половини інцидентів стаються саме через відомі вразливості, які не були вчасно усунуті, а більше третини через помилки у конфігурації та ненавмисне розкриття даних про інфраструктуру.
Зі зростом залежності від інтернет-сервісів, зростають і загрози, починаючи з BotNet сканерів та завершуючи цільовим спрямуванням APT (Advanced Persistent Threat) угрупувань.

Попри те, що більшість організацій надають перевагу IaaS (Infrastructure as a Service) та PaaS (Platform as a Service) рішенням, полягаючись на готові налаштування, цього зазвичай може бути недостатньо, та завжди є потреба у проведенні розвідки своєї інфраструктури за допомогою методологій OSINT (Open Source Intelligence).

Класичний аудит безпеки агресивним скануванням може розкрити дії зловмисників та привернути увагу ще на ранніх стадіях атаки, саме тому першим кроком є пасивний збір інформації у відкритих джерелах, не генеруючи підозрілих запитів до цілі.

Хоча збір інформації з відкритих джерел є пасивним та зазвичай прихованим, у багатьох випадках, навіть для досвідченого розробника може бути проблематично провести якісну розвідку, адже техніки та інструменти є різні. Саме розробка Автоматизованої системи аналізу захищеності має бути спрямована для збору даних про інфраструктуру для виявлення слабких місць.

1 Опис основних технологій і понять предметної області мережевої безпеки

1.1 Що таке OSINT ?

Розвідка з відкритих джерел (OSINT) – це акт збору та аналізу загальнодоступних даних для розвідувальних цілей. Хоча до більшості даних з відкритим вихідним кодом можна отримати доступ через відкритий інтернет і їх можна проіндексувати за допомогою пошукової системи, до них також можна отримати доступ через більш закриті форуми, які не індексуються пошуковими системами. Хоча більшість контенту глибокого вебу недоступна для звичайних користувачів, оскільки вона знаходиться за платним доступом або вимагає входу для доступу, вона все ще вважається частиною суспільного надбання.

Важливо також зазначити, що часто існує величезна кількість вторинних даних, які можна використовувати з кожного відкритого джерела інформації. Наприклад, облікові записи соціальних мереж можна аналізувати на предмет особистої інформації, такої як ім'я користувача, дата народження, члени сім'ї та місце проживання. Однак метадані файлів з певних публікацій також можуть розкривати додаткову інформацію, таку як місце створення публікації, пристрій, який використовувався для створення файлу, та автор файлу.

У сфері кібербезпеки дослідники та аналітики розвідки використовують дані з відкритих джерел, щоб краще зрозуміти ландшафт загроз та допомогти захистити організації та окремих осіб від відомих ризиків у їхньому IT-середовищі.

Гарним початком буде етап збору DNS записів. У них багато корисного , від ймовірної IP адреси , до відсутніх SPF політик. Ці записи є «візитною карткою» будь-якого ресурсу, але бувають випадки , коли в них немає

1.2 Використання OSINT у кібербезпеці

У кібербезпеці існує два поширених випадки використання OSINT – вимірювання ризику для власної організації та розуміння виконавця, тактики та цілей. В рамках роботи, буде розглянуто саме тактики вимірювання ризиків для власної організації.

Аналіз мережевої інфраструктури в OSINT стосується систематичного дослідження та картографування мережевих систем організації з використанням загальнодоступної інформації та неінвазивних методів. Цей метод включає ідентифікацію, документування та розуміння різних компонентів, що складають цифрову інфраструктуру організації, включаючи апаратні пристрої, топологію мережі, протоколи зв'язку та взаємоз'єднання.

По суті, аналіз мережевої інфраструктури має на меті створити повну картину того, як системи організації структуровані та підключені до Інтернету. Це включає картування:

1. Фізичне та віртуальне обладнання : сервери, маршрутизатори, комутатори, брандмауери, балансувальники навантаження та хмарна інфраструктура
2. Топологія мережі : як ці компоненти взаємопов'язані та взаємодіють один з одним
3. Зовнішнє підключення : інтернет-сервіси, точки входу та канали зв'язку
4. Механізми захисту : Засоби контролю безпеки та захисні технології на місці
5. Межі мережі : периметр, що відділяє внутрішні системи від загальнодоступного Інтернету.

На відміну від традиційного картування мережі, яке виконують внутрішні IT-команди з повним доступом, аналіз мережевої інфраструктури на основі OSINT спирається на дані, що спостерігаються ззовні. Цей підхід поєднує пасивну розвідку (збір інформації без безпосередньої взаємодії з цільовими системами) та ретельно відібрані активні методи, які не порушують операцій та законодавчих меж.

Для фахівців з кібербезпеки цей метод забезпечує критично важливий контекст для розуміння потенційних вразливостей, векторів атак та прогалин у безпеці без необхідності привілейованого доступу. Розглядаючи мережу з зовнішньої точки зору — подібно до того, як її бачили б потенційні зловмисники — команди безпеки можуть виявити сліпі зони у своєму захисті та відповідно посилити свою безпеку.

Розвідувальні дані, зібрані за допомогою аналізу мережевої інфраструктури, слугують основою для різних заходів безпеки, включаючи оцінку вразливостей, тестування на проникнення, пошук загроз та реагування на інциденти. Це також допомагає організаціям зрозуміти свій цифровий слід та вразливість до потенційних зловмисників, що дозволяє приймати більш обґрунтовані рішення щодо безпеки та розподіляти ресурси.

1.3 Domain OSINT

Ви коли-небудь замислювалися, як ваш комп’ютер знає, як знайти веб-сайт, коли ви вводите його назву у браузері? Це завдяки системі доменних імен, або скорочено DNS. Уявіть собі DNS як гігантську телефонну книгу для Інтернету. Так само, як ви використовуєте телефонну книгу для пошуку чийогось номера телефону, DNS використовується для пошуку IP-адреси веб-сайту, коли ви вводите його доменне ім’я.

Коли ви вводите адресу веб-сайту, таку як www.example.com , у свій веб-браузер, ваш комп’ютер надсилає запит до DNS-сервера, щоб знайти IP-адресу,

пов’язану з цим доменним іменем. Потім резолвер запитує кореневий сервер, щоб знайти сервер імен домену верхнього рівня (TLD), відповідальний за TLD .com. Потім сервер TLD спрямовує резолвер до авторитетного сервера імен для www.example.com, який повертає IP-адресу, пов’язану з цим доменним іменем. Резолвер кешує цю інформацію, щоб він міг швидко отримати її в майбутньому, і ваш комп’ютер міг використовувати цю IP-адресу для підключення до запитуваного вами веб-сайту.

Записи DNS схожі на записи в телефонній книзі. Вони містять інформацію про доменні імена та пов’язані з ними IP-адреси. Існують різні типи записів DNS, кожен з яких має своє призначення.

1.4 A, AAAA та CNAME записи DNS

Найпоширеніше використання записів A — це пошук IP-адрес: зіставлення доменного імені (наприклад, «**ukma.edu.ua»** з IPv4-адресою. Це дозволяє пристрою користувача підключатися до веб-сайту та завантажувати його, без необхідності запам’ятовувати та вводити фактичну IP-адресу. Веб-браузер користувача автоматично виконує це, надсилаючи запит до DNS-резолвера .

Записи DNS A також використовуються для роботи зі списком чорних дір на основі системи доменних імен (DNSBL). DNSBL можуть допомогти поштовим серверам ідентифікувати та блокувати електронні листи від відомих доменів спамерів.

Як і записи A, записи AAAA дозволяють клієнтським пристроям дізнатися IP-адресу доменного імені. Потім клієнтський пристрій може підключитися до веб-сайту та завантажити його.

Записи AAAA використовуються лише тоді, коли домен має адресу IPv6 на додаток до адреси IPv4, і коли відповідний клієнтський пристрій налаштовано на використання IPv6. Хоча всі домени мають одну або кілька адрес IPv4 та супутні записи A, не всі домени мають адреси IPv6, і не всі пристрої користувачів налаштовано на використання IPv6.

Однак, IPv6 набуває все більшого поширення. Ймовірно, це триватиме й надалі, оскільки кількість доступних IPv4-адрес швидко зменшується, що часто змушує кілька пристроїв використовувати одну IPv4-адресу

Запис CNAME вказує з псевдоніма домену на «канонічний» домен. Запис CNAME використовується замість запису A , коли домен або піддомен є псевдонімом іншого домену. Усі записи CNAME повинні вказувати на домен, а не на IP-адресу.

Наприклад, припустимо, що ***main.ukma.edu.ua*** має запис CNAME зі значенням ***ukma.edu.ua*** (без "**main**"). Це означає, що коли DNS- сервер звертається до DNS-записів для ***main.ukma.edu.ua***, він фактично запускає інший DNS-запит до ***ukma.edu.ua***, повертаючи **його** IP-адресу **через A** запис. У цьому випадку ми б сказали, що ***ukma.edu.ua*** – це канонічне ім'я (або справжнє ім'я) ***main.ukma.edu.ua.***

1.5 NS запис DNS

NS розшифровується як Name Server, а запис сервера імен вказує, який DNS сервер є авторитетним для цього домен. Це значить , що будь-який запис зі значенням адреси DNS сервера, буде пріоритетнішим ніж DNS сервери з міжнародним визнанням як Google чи Cloudflare, які часто використовуються для направлення запитів.

При проведенні розвідки , NS записи можуть вказувати на DNS провайдера, наприклад ns.awsdns.com чи ns.cloudflare.com. Зазвичай, при користуванні додатками чи сервісами, такі деталі непомітні, проте гарна розвідка розповідає дуже багато інформації. Звідси, наприклад, ми можемо

зробити висновки про те, який Web Application Firewall використовується та підкорегувати майбутні дії.

1.6 Під-домени

У будь-якій інфраструктурі є багато сервісів, як окремих, так і тих, що працюють один з одним. Саме такий підхід і може стати «Ахіллесовою п’ятою» зовнішнього периметра. Коли нападник планує свої операції, ймовірно, він не буде одразу атакувати основні ресурси організації чи компанії, а зверне увагу на ті ресурси, які розробники будуть вважати менш критичними, а саме під-домени, які не використовуються клієнтом.

При проведенні тестування на проникнення , окрема увага приділяється саме стороннім сервісам. У кожної зрілої команди є свої інструменти для CI/CD , збору метрик та логів чи звичайний портал для управління персоналом. Виходячи з такої логіки, буде природньо шукати слабкі місця в таких інструментах, саме тому одним з перших кроків оцінки безпеки системи має бути пошук під-доменів. Їх можливо шукати як самостійно , так і за допомогою сторонніх сервісів. Одним з способів – метод грубої сили, а саме перебір символів у шляху до ресурсу (Unified Resource Location), коли зловмисник перебирає усі варіанти послідовності літер у посиланні на ресурс. Хоча цей метод і дієвий, проте дуже довгий, як альтернативу можливо використати перебір по словнику. Надалі, до «перебору по словнику» буде приділена окрема увага, адже ця техніка є одним з базових прийомів як при повноцінних атаках, так і під час розвідки, а заключається вона в тому, що у світі програмування та розробки є правила , або «гарний тон» , якому слідують більшість спеціалістів. Хоч це і вважається гарною практикою, проте саме такий шлях і є причиною багатьох прогалин у системах. Саме через таку поведінку розробників, вистачає вже готового словника , або набору слів, для того щоб перебрати найвірогідніші варіанти під-доменів. Не секрет, що майже кожна організація має під-домени «dev», «grafana» чи «jekins», дуже часто

вони є незахищеними та працюють у штатному режимі, а враховуючи, що в багатьох розміщених таким чином інструментів є неусунуті вразливості, то це є дуже солодкою ціллю для хакерів та має стати пріоритетом для відділу безпеки компанії.

1.7 Перехоплення під-доменів

Перехоплення під-доменів , або subdomain take-over, це дуже проста по своїй суті техніка, підступність якої проявляється у тому, що її реалізація та перевірка на можливість її реалізації проходить повністю непомітно. На відміну від багатьох інших типів розвідки та атаки, нам не потрібно на пряму взаємодіяти з зовнішнім периметром цілі. Як згадувалося раніше, під-домени дуже часто є не самим надійним елементом саме через неуважність розробників.

Розглянемо ситуацію, коли у компанії був під-домен «marketing» для перегляду статистики по користувачам, яка надавалась у вигляді веб-сторінки на Github Pages. У такому випадку буде DNS запис зі значенням CNAME, яке буде вказувати на «marketing-test.guthubpages.com». На перший погляд усе вірно, але тільки до тих пір поки ресурс «marketing-test.githubpages.com» належить саме компанії та її розробникам. При розвідці, ми можемо пасивно збирати усі DNS запис та бачити таку інформацію. У випадку, якщо під-домен ще зареєстрований та має валідний сертифікат, а сторонній сервіс вже покинутий та не активний, у нападника є можливість створити повну копію такого ресурсу з тією ж назвою, тоді будь-яка людина, що зайшла на «marketing» під-домен буде перенаправлена на ресурс з вразливим кодом. Таким чином проводяться атаки на браузери, завантаження файлів, соціальна інженерія та інше.

1.8 HTTP з’єднання та TLS

HTTP (Hypertext Transfer Protocol) - це мережевий протокол транспортного рівня, який використовується для з’єднання клієнта та сервера,

який працює по схемі «запит-відповідь». При з’єднанні через HTTP, браузер передає усю інформацію , яка пов’язана з конкретним посиланням, а саме тіло запиту, заголовки та cookie, які зберігались у браузері. Недоліком є те, що таке з’єднання зовсім не захищене, а запит можливо перехопити та побачити увесь зміст. Такі атаки були поширені до масовості HTTPS, де з’явилась нова літера, яка значить Secure, або захищений. Захист накладає саме TLS, додаючи шифрування каналу, автентифікацію сервера і, при правильному налаштуванні, механізм недопущення підміни трафіку.

Під час з’єднання клієнта та сервера через HTTPS, стається «рукостискання», коли дві сторони обмінюються заголовками запиту Syn, Syn-Ack та Ack, саме у такій послідовності вони дають одне одному знати, що з’єднання відбулось. Під час рукостискання клієнт і сервер погоджують найвищу спільну версію протоколу, алгоритм обміну ключами та набір шифрів. Сервер доводить свою справжність сертифікатом, підписаним кореневим центром довіри; клієнт перевіряє ланцюг підписів і відповідність Common Name або SAN запитуваному хостнейму. З цього моменту всі пакети шифруються сесійним ключем, відомим лише двом сторонам.

На практиці безпека каналу залежить не від факту того чи ввімкнений HTTPS, а від конкретних параметрів. Підтримка TLS 1.0 чи 1.1відкриває шлях даунґрейд-атакам, дозволені шифри на кшталт RC4 або 3DES дають можливість дешифрувати трафік пост-фактум. Відсутність механізмів HSTS і OCSP Stapling робить користувача вразливим до перехоплення трафіку, зміни шифрування та підміни вмісту запитів. Саме тому в сучасних реаліях, сервер повинен приймати мінімум TLS 1.2, мати список шифрів, очищений від слабких комбінацій CBC, і повертати заголовок Strict-Transport-Security з прапором preload, аби браузер ніколи не спробував небезпечний та незашифрований протокол спілкування.

Уся ця інформація доступна будь-кому, а компанії, в яких недосвідчені працівники, можуть допускатись схожих помилок, тому такі перевірки мають стати обов’язковими та регулярними.

2 Опис основних технологій і понять предметної області неконтактної безпеки

2.1 Threat Intelligence

Threat intelligence, або розвідка загроз — це перетворення необроблених даних про події, артефакти й тенденції в кіберпросторі на знання, що допомагають організації приймати обґрунтовані рішення щодо захисту. Йдеться не лише про перелік IP-адрес зі шкідливим трафіком а про контекст: хто стоїть за атакою, яку мотивацію має група, які інструменти вона використала, які слабкі місця були задіяні і якова вірогідність повторного інциденту. У корпоративному світі до класичної розвідки загроз додаються такі аспекти, як моніторинг бренду та моніторинг глибокого вебу. В той час, коли бренд моніторинг – зазвичай включає в себе пошук та парсинг даних з публічних джерел як форуми, сайти відгуків та пошук вебсайтів, які порушують правила поширення контенту, то моніторинг глибокого вебу включає в себе моніторинг ресурсів, які є відкритими для публічного доступу, проте вони не з’являються у топі запитів пошукових систем , а доступ до них зазвичай передається в рамках соціальних груп та не виходить за їх межі.

В нашому випадку, розвідка загроз має бути використана для пошуку неконтактних способів атаки, а саме пошук втрачених даних для авторизації користувачів.

2.2 Моніторинг Глибокого Вебу

Зовсім нещодавно, були популярні різні форуми, куди користувачі могли потрапити тільки по запрошенню або за гроші, але в сучасному світі все змінюється досить швидко, тому на горизонті з’являються нові платформи з новими можливостями. Однією з таких платформ є Telegram. Це крос-платформенний мессенджер з відкритим доступом до API та двостороннім шифруванням. Така гнучкість, з однієї сторони є дуже привабливою для

користувачів, а з іншої надає величезні переваги для створення, організації та зберігання контенту для ентузіастів.

В контексті моніторингу, варто зосередитись саме над роботою з каналами, де користувачі можуть публікувати свій контент у різних форматах, у тому числі в форматах файлів. Серед великої кількості блогів, каналів новин та гумору, є досить прихована від звичайних пересічних користувачів сфера, а саме канали, в яких публікують витоки паролів, імейлів та логінів. Саме ця інформація і вважається критично важливою, адже набагато простіше взяти логін та пароль від сайту, ніж проникнути в нього іншими способами.

2.3 Неконтактні загрози

Уявімо компанію з ідеальним захистом, де все налаштовано ідеально та, здавалося б, що прогалина у безпеці неможлива. У таких ситуаціях, як і завжди, найслабшим елементом є людина. Наявність витоків у відкритих джерелах – не значить, що виток стався саме у таргетованої компанії.

Дослідження Statcounter показало, що впродовж усього 2024 року, приблизно 65% користувачів операційної системи Windows обирають 10-ту версію, при тому, що її підтримка від Microsoft завершується у жовтні цього року, а значить і підтримка найпопулярнішої системи захисту – Windows Defender теж буде завершена. По Україні, цей показник дещо вище – 67%, враховуючи, що сертифікацію від ДССЗЗІ Windows 10 отримала лишу минулого року, через 9 років після її появи на ринку, можемо з впевненістю сказати, що державні установи будуть під ризиком ще досить довго після завершення підтримки сертифікованої системи.

В чому ж загроза? За дослідженням deepstrike.io, через популярність ШІ, поріг входу до написання вірусів досить сильно знизився, що привело до зросту кількості інцидентів, особливо на Windows 10 та 11. Враховуючи, що доля ринку Windows – більше 80% , а писати віруси для нього стає все легше через появу таких мов програмування як Nim, дуже велика кількість людей стає під загрозою. То до чого ж тут витоки даних? Ці витоки, зазвичай, не є результатом

зламу веб сайтів, це інформація, яка надходить від вірусів на операційних системах користувачів. Уся інформація з браузерів, з сховищ паролю та інших можливих місць надходить у руки зловмисникам. Так як такі віруси працюють на масовість, розповсюджуються вони самими різними способами, у програмах, у файлах скачаних з торент, піратським іграм та інших варіантах. Усі ці віруси збирають інформацію та передають на C2 (Command & Control) сервер, звідки вони парсяться та далі з'являються на публічних ресурсах. Людина з найкращим антивірусом на корпоративному комп'ютері та з самими кращими стандартами безпеки, всього лише може використовувати однаковий пароль вдома, на інфікованому комп'ютері та на роботі. Збираючи словник паролей співробітників, хакер може спрямувати це на перебір грубою силою та потенційне проникнення у систему.

3 Проектування системи аналізу захищеності на основі OSINT

3.1 Огляд існуючих систем та технологій

У сфері кібербезпеки, методологіям OSINT приділяється особлива увага. Існує безліч рішень, від автоматизованого пошуку пошуку нікнейму серед соціальних мереж, до GeoINT (Geografical Intelligence) фреймворків для пошуку локацій по місцевостям. Одним з рішень, яке має найбільшу схожість з реалізованим проектом є Rengine – Система націлена на проведення операцій у зовнішній мережі. Ця система гарно побудована, але має ряд недоліків. Розглянемо спочатку її переваги:

1. Гарний інтерфейс – система має дійсно дуже інформативний інтерфейс, який включає в себе карту розташування цілей, статистику по знайденим вразливостям та декілька різних сторінок з більш детальним описом проведених маніпуляцій.
2. Масштабованість – цей пункт є дуже сумнівним, адже попри те, що у додаток було інтегровано «двигун» для запуску різних утиліт, це сильно ускладнило архітектуру всього проекту. Було протестовано

встановлення деяких додаткових модулів та ні один з них не працював належним чином. Так як встановлення йде через надання команди для встановлення на ОС Linux, яка розгорнута у Docker, дуже часто встановлення переривалось через нестачу додаткових залежностей у системі, які потрібно було до встановлювати через Docker консоль. Навіть якби вдалось все вірно запустити, труднощі тільки починались.

Варто далі розглянути недоліки системи:

1. Складність використання – як було сказано раніше, встановлення нових модулів потребує окремих знань роботи з операційними системами на низькому ріні, а також встановлення багатьох додаткових залежностей у систему. Навіть якщо усі кроки були виконані, є ймовірність що ново встановлений модуль не буде відпрацьовувати як потрібно, адже налаштування його роботи йдуть через YML файл, інструкції для написання якого не вистачає , та потрібно експериментувати.
2. Нестача OSINT модулів – попри те, що цей інструмент гарно масштабується , в базі він не має базових OSINT інструментів та покладється на результати більш агресивної програми Nuclei.

Попри всі недоліки та переваги, у цієї системи дуже цікава базова ідея, тому варто розглянути, які технології були використані. Увесь код застосунку написаний на мові Python, що є позитивним елементом, через те, що ця мова вважається досить високорівневою та більш простою у вивченні базового синтаксису, досить велика кількість людей її використовує для розробки різних інструментів , особливо для OSINT. Для відпрацювання багатьох процесів паралельно, була використана черга Celery разом з Django фреймворком для серверної частини. До речі, застосунок не має клієнтської частини, а використовує Server-Side Rendering для надання користувачу інтерфейсу.

У висновку можемо сказати, що проект досить успішний та в нього є своя аудиторія, але найбільшим його недоліком є те, що він надто складний для

більшості користувачів, враховуючи що мало кого цікавить витратити понад годину на те, щоб розібратись як працює додаток.

3.2 Проектування своєї системи аналізу захищеності

У цьому розділі буде розглянуто інструменти та підходи до побудови архітектури застосунку. Основними критеріями для побудови були горизонтальна масштабованість, максимальна можливість інтегрувати відомі інструменти та розділення залежностей по контейнерам для уникнення конфліктів при скануваннях.

Так як загально відомо, що Python є мовою, у якої дуже велика кількість прихильників , було обрано саме її, так як для неї було написано багато корисних бібліотек та вона є фаворитом у кібербезпеці та підтримує дуже просту інтеграцію різних компонентів.

Розгортати застосунок було вирішено у докер, так як він є найкращим рішенням для крос-платформної розробки, а також легко запускається на будь-якому типі процесорних чипів, як ARM, так і AMD. За допомогою Docker Network дуже просто організувати роботу між різними сервісами.

Щоб розподілити навантаження на різні оброники , або Worker, було використано Arq Worker, який побудований на базі Asyncio, та підтримує сотні, або навіть тисячі паралельних задач, які можуть виконуватись та не блокувати один одного. Цей інструмент також підтримує відкладені задачі, декілька спроб виконати задачу та налаштування з’єднань з базою даних, що дає впевненість в тому, що усі покладені задачі будуть виконані. Дуже важливим є те, що при своїй функціональності ,він є досить компактним, всього 700 рядків коду, та простим у використанні.

Для сплікування з Telegram API, була використана бібліотека Telethon, яка побудована за допомогою Asyncio та підтримує з’єднання через MTProto – спеціальний протокол , який працює поверх TCP, та шифрує дані клієнта. У цієї бібліотеки є багато переваг, наприклад вона підтримує декілька варіантів взаємодії з MTProto, як користувач та як бот, а також вона є досить простою у використанні, для початку роботи потрібно всього лише отримати токен на

офіційному сайті Telegram та підключитися до апі за допомогою всього лише 5 рядків коду. Саме через можливість імітації юзера, сервери телеграм сприймають усю активність звернень як легітимну та не блокують запити, що ідеально підходить для парсингу даних з різних каналів.

Основний збір DNS записів реалізовано двома шляхами: прямі запити авторитетним серверам виконуються бібліотекою dnspython, щоб отримати повну відповідь разом із DNSSEC-підписами. Історичні дані надходять із API SecurityTrails у форматі JSON. Це поєднання дає нам можливість майже миттєво отримати інформацію цілком безкоштовно без потреби піднімати свою базу даних та оновлювати її, що займало б дуже багато часу та ресурсів.

Для перевірки TLS використовується testssl.sh— це безкоштовний інструмент командного рядка, який перевіряє службу сервера на будь-якому порту на підтримку шифрів TLS/SSL, протоколів, а також деяких криптографічних недоліків. Основним недоліком цього інструменту є те, що він задумувався як інструмент командного рядку, але якщо його встановити у Docker контейнер та викликати за допомогою Python, з’являється можливість зпарсити JSON результат та зберегти його у базу даних. Операції по аналізу криптографії є досить тривалими, тому будуть виконуватись в окремому контейнері обробника події.

Одним з найвідоміших та найдієвіших інструментів, без якого не обходиться жодна операція з кібербезпеки це Nmap. Саме за допомогою Nmap, відбувається сканування відкритих портів та ідентифікація технологій та їх версій за банерами. Nmap може просканувати усі TCP та UDP порти, а велика кількість заготовлених скриптів дає можливість одразу виявляти загрози у зовнішньому периметрі мережі. Саме з цього зазвичай починається побудова векторів атаки. У контексті застосунку, буде використане API для nmap, який в свою чергу, буде встановлено у Docker контейнер та викликаний з одного з обробників подій.

Збір імейл адрес буде відбуватись за допомогою сторонніх API. У той час, як робити розвідку по адресам можливо й через спеціальні запити до

пошукових систем, для точності та більш прогнозованої роботи, варто скористатись такими сервісами як hunter.io чи snov.io, адже в них є ліміт безкоштовних пошуків, що підійде для некомерційного використання.

Для побудови API застосунку буде використано FastAPI , який на відміну від Django, задумувався як асинхронний фреймворк, побудований на базу Starlette, що ідеально підходить для задачі. Такий функціонал, як генерація Swagger документації на базі моделей Pydantic та Background Task, дозволяє спростити розробку та тестування, а також ефективно розподіляти задачі між Worker-ами. Так як уся бізнес логіка винесена до Arq. Worker, задача FastAPI – слугувати швидким та надійним маршрутизатором запитів.

Для розробки клієнтського інтерфейсу, було використано React бібліотеку, так як вони є досить простою для опанувати ,але при тому досить надійною та масштабованою. Саме для роботи з даними з різних сервісів, React показує себе прекрасно. Завдяки односторінковому підходу, розділенню на компоненти зі своїм станом, є можливість їх тестувати, масштабувати та змінювати, згідно змін на серверній частині. Для обробки запитів та стану застосунку, був використаний тандем з Redux Toolkit та Axios, який дозволяє здійснювати запити та одразу зберігати результати у стан застосунку та розподіляти його між компонентами.

4 Реалізація системи аналізу захищеності веб-застосунків

4.1 Архітектура застосунків

Система аналізу контактних та неконтактних загроз поділена на два окремих застосунка з API , які використовуються для різних задач, але прислідують спільну мету. Перший – націлений на розвідку та збір інформації про інфраструктуру, другий – націлений на збір витоків з платформи Telegram та пошуку потрібних даних. Попри те, що це да різні застосунки, вони маюьб

спільну архітектуру. На рисунку 1.1 можемо спостерігати основну схему застосунку.

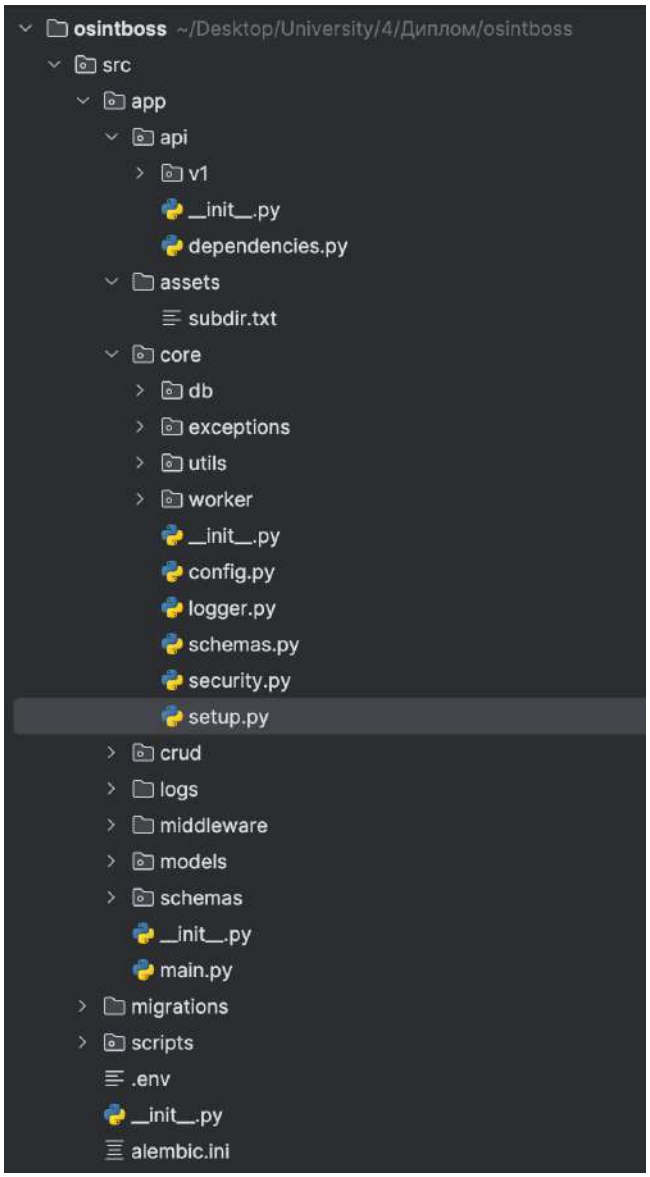
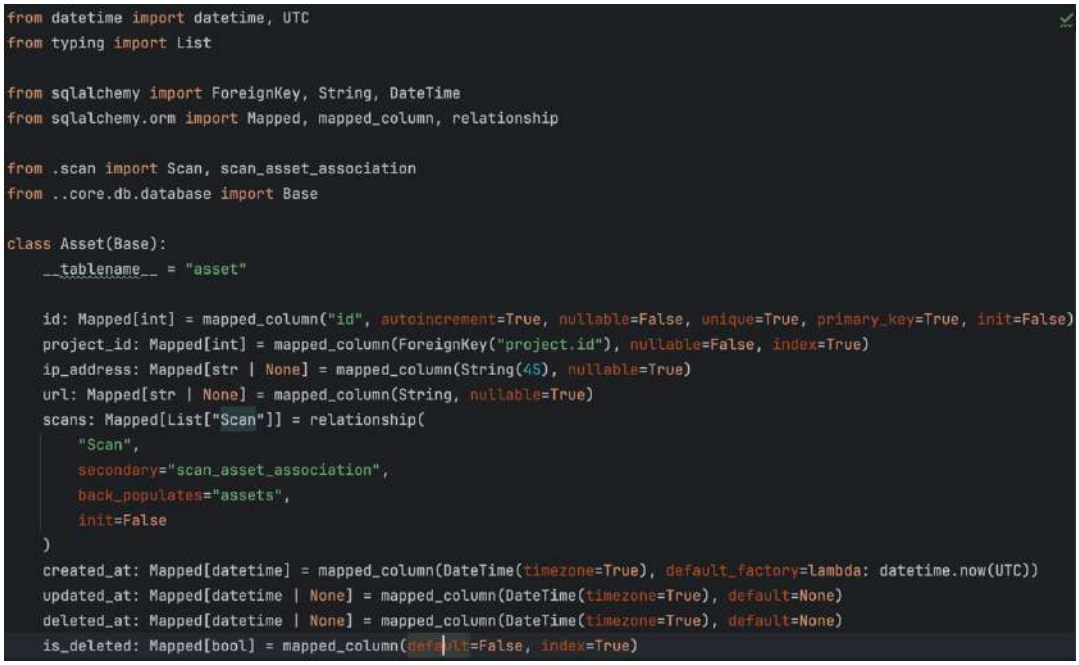


Рисунок 1.1

У шляху *api/v1* зберігаються файли усіх кінцевих точок застосунку, у шляху *src/core* – налаштування Worker та з'єднання з базою даних, також є setup.py файл з ініціалізацією застосунка та налаштувань. У шляху *src/models* знаходяться моделі для бази даних, у яких прописана структура усіх таблиць, їх значень та зав'язків. Шлях *src/schemas* відповідальний за схеми даних для Pydantic, за допомогою якого відбувається валідація даних при записі до БД.

4.2 Компоненти системи мережевої розвідки

У цьому розділі буде розглянуто окремі компоненти та їх логіку. Кожен застосунок починається з бази даних, а саме з опису залежностей.

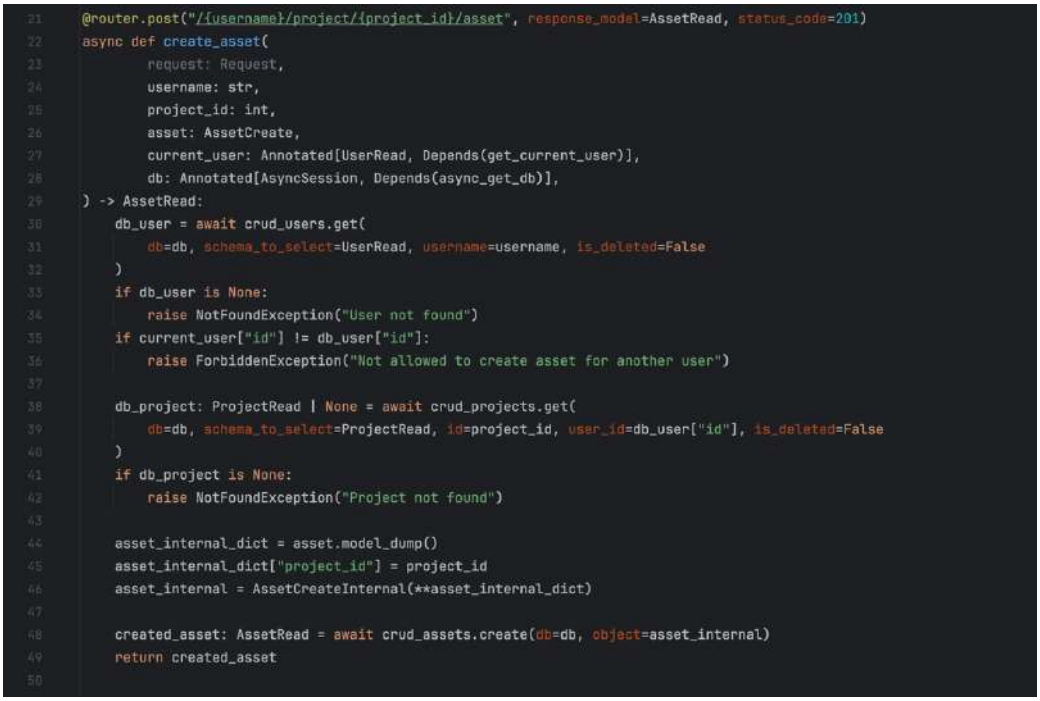


```python
from datetime import datetime, UTC
from typing import List

from sqlalchemy import ForeignKey, String, DateTime
from sqlalchemy.orm import Mapped, mapped_column, relationship

from .scan import Scan, scan_asset_association
from ..core.db.database import Base

class Asset(Base):
    __tablename__ = "asset"

    id: Mapped[int] = mapped_column("id", autoincrement=True, nullable=False, unique=True, primary_key=True, init=False)
    project_id: Mapped[int] = mapped_column(ForeignKey("project.id"), nullable=False, index=True)
    ip_address: Mapped[str | None] = mapped_column(String(45), nullable=True)
    url: Mapped[str | None] = mapped_column(String, nullable=True)
    scans: Mapped[List["Scan"]] = relationship(
        "Scan",
        secondary="scan_asset_association",
        back_populates="assets",
        init=False
    )
    created_at: Mapped[datetime] = mapped_column(DateTime(timezone=True), default_factory=lambda: datetime.now(UTC))
    updated_at: Mapped[datetime | None] = mapped_column(DateTime(timezone=True), default=None)
    deleted_at: Mapped[datetime | None] = mapped_column(DateTime(timezone=True), default=None)
    is_deleted: Mapped[bool] = mapped_column(default=False, index=True)
```

Рисунок 1.2

На рисунку 1.2 можемо побачити схему таблиці “Asset” , яка є відповідальною за основні властивості сутності як ідентифікатори, часові відбитки, значення та зв’язок з таблицею “scan” за допомогою створення спільної таблиці значень “scan_asset_association” , яка відповідає за збереження цілісності зв’язків та логіки застосунку.

Коли є база даних для наповнення, повинна бути і точка входу для запису даних. У архітектурі застосунку використаний природний ступінь ієрархії, тобто, у користувача є проекти, в яких є цілі, в яких є сканування та у сканування може бути багато результатів , так як і цілей у одного сканування. Кожна сутність має свої точки входу та кожна наступна не може бути створена без попередньої.



```python
@router.post("/{username}/project/{project_id}/asset", response_model=AssetRead, status_code=201)
async def create_asset(
        request: Request,
        username: str,
        project_id: int,
        asset: AssetCreate,
        current_user: Annotated[UserRead, Depends(get_current_user)],
        db: Annotated[AsyncSession, Depends(async_get_db)],
) -> AssetRead:
    db_user = await crud_users.get(
        db=db, schema_to_select=UserRead, username=username, is_deleted=False
    )
    if db_user is None:
        raise NotFoundException("User not found")
    if current_user["id"] != db_user["id"]:
        raise ForbiddenException("Not allowed to create asset for another user")

    db_project: ProjectRead | None = await crud_projects.get(
        db=db, schema_to_select=ProjectRead, id=project_id, user_id=db_user["id"], is_deleted=False
    )
    if db_project is None:
        raise NotFoundException("Project not found")

    asset_internal_dict = asset.model_dump()
    asset_internal_dict["project_id"] = project_id
    asset_internal = AssetCreateInternal(**asset_internal_dict)

    created_asset: AssetRead = await crud_assets.create(db=db, object=asset_internal)
    return created_asset

```

Рисунок 1.3

На рисунку 1.3 зображена одна з кінцевих точок для взаємодії з застосунком. Вона відповідає за перевірку авторизації , вірності введених даних та запис інформації про ціль у базу даних.

Одним з найбільш відповідальних частин є розподілення задач після створення запиту на сканування. Основною проблемою є те, що у випадку, коли цілей багато, цей запит робить дуже важкі звернення до бази даних ,тому було прийнято рішення використати вбудований модуль FastAPI для розподілення навантаження та обробки даних «за кулісами» та без затримки відповіді сервера. Однією з особливостей такого модуля є те, що передавати з’єднання до бази даних не можна, адже створюється новий процес і з’єднання може конфліктувати з наявними надалі, так як після завершення роботи, процес закривається, а з’єднання ні, тому створювати та закривати його потрібно саме вже під час початку роботи модуля.

```python
async def start_scan_jobs(scan_id: int,api_keys:dict) -> None:
    async with async_get_db_session() as db:
        logging.info(f"Starting scan {scan_id}")
        scan = await crud_scans.get(db=db, id=scan_id, schema_to_select=None)
        if not scan:
            logging.error(f"Scan with id {scan_id} not found in database.")
            return

        asset_ids = getattr(scan, "asset_ids", [])
        asset_list = []
        for asset_id in asset_ids:
            asset = await crud_assets.get(db=db, schema_to_select=AssetRead, id=asset_id)
            if not asset:
                logging.info(f"Asset with id {asset_id} not found.")
                continue
            target = asset.get("ip_address") or asset.get("url")
            if target:
                asset_list.append(target)

        logging.info(f"Asset targets: {asset_list}")

        for flag, job_name in job_flags.items():
            if getattr(scan, flag, False):
                if flag == "run_securitytrails":
                    try:
                        job = await queue.pool.enqueue_job(
                            "run_securitytrails_job",
                            scan_id=scan_id,
                            asset_list=asset_list,
                            api_keys=api_keys,
                            _queue_name=f"arq:run_securitytrails"
                        )
                        logging.info(f"Enqueued securitytrails job {job} for scan {scan_id}")
                    except Exception as e:
                        logging.error(f"Failed to enqueue SECURITYTRAILS job on scan {scan_id}: {e}")
                    continue
```

Рисунок 1.4

На рисунку 1.4 можемо побачити, що задача не є складною і заключається в отриманні інформації про запущене сканування та передати параметри у відповідний Arq Worker.

```python
class NmapWorkerSettings:
    functions = [
        nmap_scan_task
    ]
    redis_settings = RedisSettings(host=REDIS_QUEUE_HOST, port=REDIS_QUEUE_PORT)
    on_startup = startup
    on_shutdown = shutdown
    max_jobs = 1
    queue_name = "arq:run_nmap"
    handle_signals = False
    job_timeout= 150000
```

Рисунок 1.5

Саме за допомогою Arq Worker і можливе розподілення задач у системі. На рисунку 1.5 продемонстровані налаштування одного з таких обробників задач. Можемо побачити, що для черги задач використовується черга Redis , у кожного обробника є функція активації та деактивації , які відпрацбовують на початку, та відповідно, в кінці роботи. Також вказана кількість максимальної кількості паралельних задач, в нашому випадку значення 1 для простішої розробки. Також потрібно задати назву черги та час до закінчення очікування результату.

```
# --------- final image build ---------
FROM python:3.11

WORKDIR /code

COPY --from=requirements-stage /tmp/requirements.txt /code/requirements.txt

RUN pip install --upgrade pip setuptools wheel

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

COPY ./src/app /code/app

RUN apt-get update && apt-get install -y nmap \
    bash \
    git \
    curl \
    coreutils \
    bsdmainutils \
    dnsutils

RUN git clone --depth 1 https://github.com/testssl/testssl.sh.git /opt/testssl.sh && \
    ln -s /opt/testssl.sh/testssl.sh /usr/local/bin/testssl.sh && \
    chmod +x /usr/local/bin/testssl.sh

ENV JSONFILE=/tmp/testssl_output.json
ENV TERM_WIDTH=80

# -------- replace with comment to run with gunicorn --------
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

Рисунок 1.6

Що стосується робочої середи, то застосунок працює у Docker на операційній системі Debian Linux, а деякі важливі компоненти встановлюються на етапі ініціалізації , як прописано у Dockerfile на рисунку 1.6 вище.

4.3 Компоненти системи неконтактної розвідки

Так як два застосунки поділяють схожу архітектуру, тут варто загострити увагу саме на унікальних для цієї системи компонентах, а саме модуль роботи з Telegram. Загалом, уся робота побудована на взаємодії з MTProto через Telethon бібліотеку. Для роботи нам потрібні декілька значень: API_ID, API_HASH та Session. В той час коли, API_ID та API_HASH є значеннями, які надаються офіційним сайтом Telegram, то Session це значення, яке користувач має отримати самостійно. Саме значення Session і є тим ідентифікатором, який розділяє людину і бот з точки зору API Telegram. Це значення можливо отримати тільки після проходження авторизації, це такий замінник класичного токену JWT. На відміну від JWT , Session зашифрований дещо сильніше ніж Base64, тому і інформації про нього трошки менше, адже про можливість використання такого токену мало де згадується , адже у веб версії та у версії для комп’ютерів використовується дещо інший формат. Секрет цього токену в тому, що це просто набір значень по яким Telegram вирішує до якого ЦОД під’єднається клієнт та за яким секретним значенням буде проведена авторизація. Здавалося б, що цей токен зашифрований та більш найдійний ніж JWT, але він не підписаний, тому фактично користувач може сам вирішувати до якого ЦОД йому під’єднатись.

```python
async def download_media(self, message):
    if message.media:
        if self.download_dir is None:
            self.download_dir = os.path.join(self.base_download_dir, self.scrape_id)
            os.makedirs(self.download_dir, exist_ok=True)
        if isinstance(message.media, MessageMediaDocument):
            file_name = message.media.document.attributes[0].file_name
            self.logger.info(f'{self.scraper_name}: Downloading {file_name} from message {message.id}')
            file_path = await self.client.download_media(message, self.download_dir)
            if file_path:
                file_name = os.path.basename(file_path)
                self.logger.info(f'{self.scraper_name}: Downloaded {file_name} from message {message.id}')
                return True
            else:
                self.logger.warning(f'{self.scraper_name}: Failed to download media from message: {message.id}')
    return False
```

Рисунок 2.1

На рисунку 2.1 зображена одна з основних функцій , а саме завантаження файлів. Ця функція є частиною більшого класу, в якому закладені основні

функції роботи з API. У класа також є своя функція логування, яка приймає назву класа.

```python

logger = logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")

2 usages
async def run_scrapers(channel_identifier, start_date, end_date, scrape_id, logger=logger):
    """Run the Telegram scraping process with two scrapers in parallel."""

    channel_identifier = extract_channel_identifier(channel_identifier)

    logger.info(f"Starting scrape for channel: {channel_identifier} from {start_date} to {end_date}")

    try:
        scraper1 = TelegramScraper(API_ID, API_HASH, SESSION1, DOWNLOAD_DIRECTORY, scrape_id, scraper_name: 'scraper-1',
                                   logger=logger)
        scraper2 = TelegramScraper(API_ID, API_HASH, SESSION2, DOWNLOAD_DIRECTORY, scrape_id, scraper_name: 'scraper-2',
                                   logger=logger)

        media_message_ids = await scraper1.get_media_message_ids(channel_identifier, start_date, end_date)

        mid_index = len(media_message_ids) // 2
        scraper1_ids = media_message_ids[:mid_index]
        scraper2_ids = media_message_ids[mid_index:]

        await asyncio.gather(
            scraper1.run(channel_identifier, scraper1_ids),
            scraper2.run(channel_identifier, scraper2_ids)
        )

        logger.info(f"Scraping completed for both sessions.")
        return {"message": "Scraping completed successfully."}
    except Exception as e:
        logger.error(f"Error during scraping process: {str(e)}")
        raise e

```

Рисунок 2.2

На рисунку 2.2 продемонстрована досить унікальна особливість роботи цього парсеру. Він використовує дві сущності класу TelegramScraper та дві окремі сесії користувача. Цей функціонал може бути використаний для багатьох випадків, а саме для під’єднання до різних ЦОД, щоб пришвидшити завантаження чи нприклад для масштабування горизонтально для збільшення швидкості завантаження. Хоча, Telethon має вбудований функціонал перемикання центрів даних, проте цей процес займає багато часу та може бути розцінений як підозріла активність. Наведена вище функція приймає усі відмічені ID повідомлень та починає їх завантаження у два потоки за допомогою *asyncio.gather.*

Так як ця система все ж досить дотична до розвідки загроз, був доданий функціонал парсингу веб-сайтів за допомогою Selenium для демонсрації можливостей. Це бібліотека, яка дає можливість працювати з ядром браузеру та

з докер контейнеру їх відкривати та зчитувати наповнення. Недоліком є те, що для кожного сайту потрібно розробляти окремий модуль парсингу. У цілях тестування, було додано один веб-сайт на якому розміщені вразливості та злочинний код для експлуатації вразливостей. Для тестування потрібно ввести довільну дату для перевірки публікацій за той день та отримане посилання передати у наступний ендпоінт у розділі tester у Swagger документації.

4.4 Огляд відповіді системи

Основна інформація від системи, буде показана при перегляді проекту у розділі Results при натисканні на одне з сканувань у рамках проекту. Вся інформація буде розподілена по розділам, як показано на рисунку 3.1.

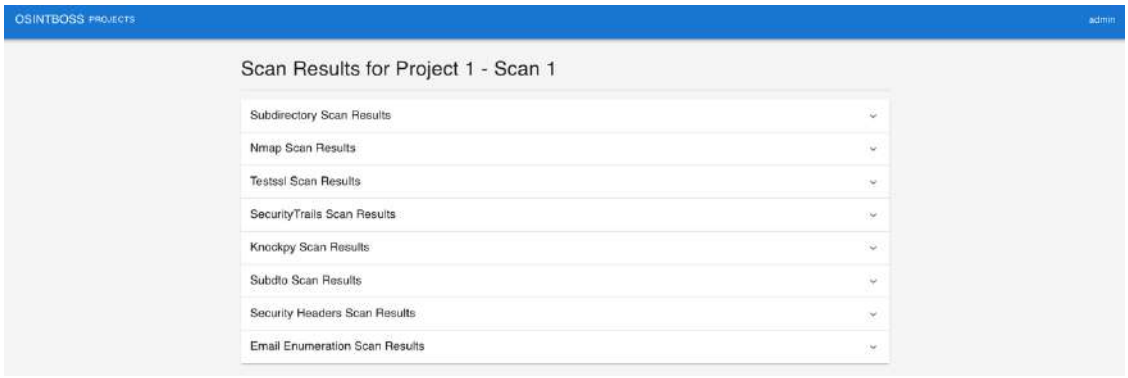


Рисунок 3.1

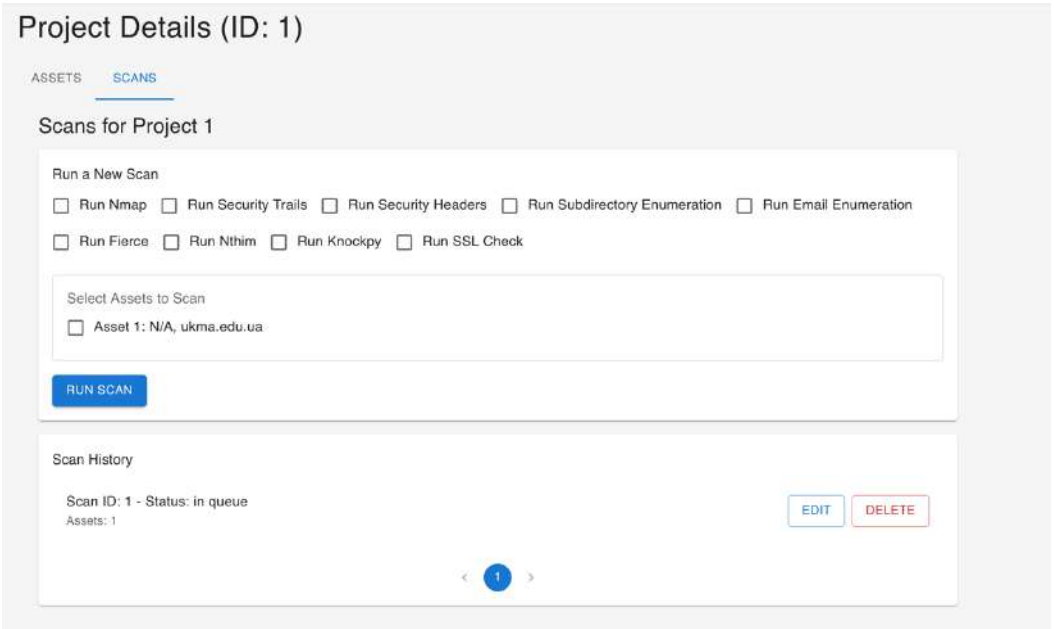


Рисунок 3.2

Для початку сканування потрібно обрати ціль зі списку та обрати параметри сканування, які будуть потрібні для отримання очікуваного результату як показано на рисунку 3.2.

Для прикладу результатів сканування, візьмемо сканування TLS.

Result ID: 1
Scan ID: 1
Host: ukma.edu.ua
IP Address: 135.181.9.30
Port: 443
Service: HTTP
RDNS: web.ukma.edu.ua.

Rating

ID	Finding	Severity
rating_spec	SSL Labs's 'SSL Server Rating Guide' (version 2009q from 2020-01-30)	INFO
rating_doc	https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide	INFO
protocol_support_score	100	INFO
protocol_support_score_weighted	30	INFO
key_exchange_score	90	INFO
key_exchange_score_weighted	27	INFO
cipher_strength_score	90	INFO
cipher_strength_score_weighted	36	INFO
final_score	93	INFO
overall_grade	A	OK
grade_cap_reason_1	Grade capped to A. HSTS is not offered	INFO

Рисунок 3.3

Як можемо побачити на рисунку 3.3 , сканування пройшло успішно, ми отримали інформацію про перевірку рейтингу шифрів веб сайту та можемо бачити, що у даному випадку немає HSTS, що призвело до зниження оцінки.

Висновки

У процесі виконання дипломної роботи було реалізовано повноцінну систему автоматизованого OSINT-аналізу захищеності веб-ресурсів, що поєднує пасивне збирання даних, мінімально активні перевірки та безперервний моніторинг витоків. Запропонована архітектура — контейнерна, мікросервісна, з асинхронним FastAPI-backend, чергою ARQ, окремими сканерами в Docker контейнерах і React веб сайтом для керування. Такий поділ дає змогу гнучко масштабувати виробничі навантаження та оновлювати окремі модулі без зупинки всієї платформи.

Практична цінність розробки полягає в тому, що користувач отримує карту периметра: актуальні DNS-записи, історію змін, карту під-доменів, ризики перехоплення під-доменів, відкриті порти, конфігурацію TLS і заголовки безпеки HTTPS, а також можливість пошку витоків корпоративних облікових записів. Завдяки цьому час між появою нової уразливості й її усуненням скорочується з тижнів до годин, а отже зменшується ймовірність успішної атаки та пов’язаних фінансових і репутаційних втрат.

Список використаної літератури

1. https://www.verizon.com/business/resources/reports/dbir/
2. https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/open-source-intelligence-osint/
3. https://www.cyberquizzer.com/blog/osint-network-infrastructure-analysis#what
4. https://www.cloudflare.com/en-gb/learning/dns/dns-records
5. https://www.makeuseof.com/windows-11-market-share-is-dropping-again/
6. https://deepstrike.io/blog/Malware-Attacks-and-Infections-2025