Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ІНСТРУМЕНТІВ ТЕСТУВАННЯ ВРАЗЛИВОСТЕЙ НА ПРИКЛАДІ ВІРТУАЛЬНИХ МАШИН
(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2
Прокоп’єв С. А.
(прізвище, ініціали)

Спеціальності 122 Комп’ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Прокоп'єву Степану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів та інструментів тестування вразливостей на прикладі віртуальних машин

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії 25 грудня 2024 р.

3. Вихідні дані до роботи взламані віртуальні машини (віртуальні машини, на яких проводилась експлуатація вразливостей й на яких було на практичному досвіді доведенно вразливість), отримані флаги (де це можливо).

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд принципів, за якими працює Інтернет.
2. Огляд поширених вразливостей.
3. Огляд методів експлуатації вразливостей.
4. Експлуатація вразливостей.
5. Огляд життєвого циклу тестування на проникнення.
6. Ознайомлення та практичне використання OSINT методів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми недостатньої уваги до кібербезпеки, аналіз вразливостей, схеми вразливостей та їх принципи роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	26.11.24-27.11.24	
3	Аналіз літератури з досліджуваної проблеми	27.11.24-28.11.24	
4	Аналіз вразливостей	28.11.24-30.11.24	
5	Експлуатація вразливостей	01.12.24-03.12.24	
6	Програмна реалізація	03.12.24-04.12.24	
7	Оформлення пояснювальної записки	04.12.24-05.12.24	
8	Перевірка на плагіат	10.12.2024	
9	Рецензування	16.12.2024	
10	Підготовка презентації та доповіді	20.12.2024	
11	Занесення роботи в електронний архів	02.01.2025	
12	Попередній захист кваліфікаційної роботи	02.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Тітова О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 68 с., 1 табл., 57 рис., 41 джерело.

ТЕСТУВАННЯ БЕЗПЕКИ, ВРАЗЛИВІСТЬ, API АТАКИ, FUZZING ТЕСТУВАННЯ, ВІРТУАЛЬНА МАШИНА, ПОЛАМАНА АВТОРИЗАЦІЯ, SQL ІН'ЄКЦІЯ, МЕЖСАЙТОВИЙ СКРИПТИНГ, ДОСЛІДЖЕННЯ МЕРЕЖ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

Об'єктом дослідження є процес тестування тестових сайтів на різноманітні вразливості (міжсайтовий скриптинг, ін'єкції, атаки на стороні API, проблеми з аутентифікацією та авторизацією, знаходження непублічного контенту) з наданням усієї необхідної теоретичної інформації та демонстрацією вразливостей на практичному досвіді за допомогою віртуальних машин, які надаються платформою HackTheBox.

Метою дослідження є розробка комплексного підходу до тестування безпеки веб-ресурсу, а саме створення інструкції для тестувальників на проникнення, що стане у нагоді перед введенням веб-ресурсу у експлуатацію.

Очікуваним результатом дослідження є демонстрація згаданих вище вразливостей та отримання флагів (імітація конфіденційної інформації) там, де це передбачено, а також відображення повного процесу експлуатації кожної з вразливостей.

PENETRATION TESTING, VULNERABILITY, API ATTACKS, FUZZING TESTING, VIRTUAL MACHINE, BROKEN AUTHORIZATION, SQL INJECTION, CROSS-SITE SCRIPTING, NETWORK MAPPER, CLIENT-SERVER ARCHITECTURE.

The object of the research is the process of testing test sites for various vulnerabilities (cross-site scripting, injections, attacks on the API side, problems with authentication and authorization, finding non-public content) with the provision of all necessary theoretical information and demonstration of vulnerabilities on practical experience using virtual machines provided by the HackTheBox platform.

The purpose of the research is to develop a comprehensive approach to testing the security of a web resource, namely the creation of a manual for penetration testers which will be useful before putting a web resourse into operation.

The expected result of the research is the demonstration of the above-mentioned vulnerabilities and receiving flags (simulation of confidential information) where it is provided, as well as the display of the full process of exploiting each of the vulnerabilities.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

SDLC – Software Development Life Cycle (життєвий цикл розробки ПЗ)

STLC – Software Testing Life Cycle (життєвий цикл тестування ПЗ)

API – Application Programming Interface (програмний інтерфейс)

Ендпоінт – URL-адреса, через яку клієнт має змогу отримувати дані та обробляти їх

CRUD – Create Read Update Delete (операції, які можна робити з даними)

ID – Identifier (унікально ідентифікуючий номер, за яким однозначно можна дізнатись про що йде мова)

DOM – Document Object Model (об’єкт документу, який генерується після завантаження сторінки)

HTML – Hyper Text Markup Language (мова верстки та написання структури вебсайтів)

XSS – Cross-Site Scripting (міжсайтовий скриптинг)

OWASP – Open Web Application Security Project (некомерційна група ентузіастів з Інтернету, які цікавляться безпекою)

PoC – Proof of Concept (доказ, який наводиться для того, щоб показати працездатність чогось у програмуванні)

Токен – набір символів, що унікально ідентифікує користувача, надсилається з кожним запитом

ВМ – віртуальна машина

Флаг – секрет, що сховано на вразливому сайті

CTF – Capture The Flag, формат практичного застосування засвоєної інформації з ціллю отримання секрету (флагу)

ВСТУП

З моменту появи Інтернету технології розвивались з неймовірною швидкістю. Дану тенденцію можна спостерігати й на сьогодення. Але разом з цим розвивались й кібератаки.

Наразі у майже кожної людини є принаймні один девайс, за допомогою якого можна доєднатись до мережі Інтернет, й у більшості випадків, таких девайсів на одну людину припадає близько 3-4. Усі ми користуємось Інтернетом кожного дня: для того, щоб переглянути новини, написати друзям, подивитись серіал, знайти відповідь на питання, проходити курси та навчатись, тощо. Попри таку шалену популярність Інтернету та велику кількість часу, який люди проводять в Інтернеті здавалося би, що у нових проектів просто неможливо буде знайти якусь вразливість, але авжеж це не так.

Хакери та розробники наче грають в теніс, почергово знаходячи нові вразливості та впроваджуючи нові патчи, в яких дана вразливість буде усунена. У розрізі кібербезпеки завжди потрібно «тримати руку на пульсі» та слідкувати за останніми тенденціями.

За досить короткий час існування Інтернету були скомпрометовані тисячі веб-сайтів, мобільних додатків, програмних забезпечень, тощо. Серед яких були й крупні бренди – Google, Twitter, RockYou та інші [1]. Навіть Microsoft мала критичні вразливості, які дозволяли користувачам віддалено отримати доступ до будь-якого іншого девайсу з операційною системою Windows.

Важливо зауважити, що хоча первинна відповідальність у забезпеченні кібербезпеки полягає на розробників, проте користувачі також потрібно пам’ятати про власну кібер-гігієну, адже як всім відомо, людина – це найслабкіша ланка, яку найлегше експлуатувати. Таким чином були зламані акаунти відомих діячів, серед яких Ілон Маск, Дональд Трамп та багато інфлюєнсерів й блогерів на таких платформах як YouTube та Twitter.

Що може бути більш спокусливим для хакера, як не повністю захищений ресурс, яким користуються мільйони користувачів щодня? Кібер-шахраї щодня вигадують нові вектори атак, деякі з яких є успішними й несуть шкоду. Саме для цього при розробці та тестуванні ПЗ потрібно дуже велику увагу приділяти безпеці, адже один інцидент може спричинити колосальним втратам: від особистих даних користувачів до засекречених документів компанії.

Важливість кібербезпеки неможливо переоцінити, деякі люди навіть вважають, що війни майбутнього будуть суто у кібер-просторі. Українці вже це відчували частково на власному досвіді, ще з 2015 року країна-терорист впроваджувала численні кібер-атаки, націлені на виведення енергосистеми України з ладу (що вважається кібертероризмом) та на руйнування та повне припинення роботи усіх Інтернет-сервісів (як вірус сімейства NotPetya) [2].

З нещодавніх новин так само за допомогою кіберспеціалістів була проведена операція Ізраїля по самознищенню девайсів, що мали літій-іонні акумулятори на території Лівану. Ситуації описані вище лише підтверджують концепцію майбутніх війн.

Актуальність дослідження полягає у розробці комплексного методу тестування безпеки веб-ресурсу. Даний метод буде відповідати рекомендаціям OWASP та покривати значну частину найбільш розповсюджених вразливостей. Наведений матеріал буде супроводжуватись усіма необхідними теоретичними знаннями та підкріплено на практичному досвіді, що збільшить розуміння матеріалу та зробить інформацію більш ефективною та корисною.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ. ПІДГОТОВКА ДО ТЕСТУВАННЯ НА ПРОНИКНЕННЯ.

1.1 Життєвий цикл тестування на проникнення

Тестування безпеки, як й будь-який інший процес розробки чи тестування можна нормалізувати, стандартизувати та описати у вигляді життєвого циклу, як це зроблено з SDLC та STLC [3]. Процес тестування безпеки може відрізнятись в залежності від компанії, де ви працюєте чи від вимог замовника, але зазвичай будь-яке тестування безпеки можна розподілити на наступні фази та кроки [4]:

Перша фаза – збір інформації, містить у собі наступні кроки:

– стадія 1: мережева та технічна розвідка. На даній стадії збирається інформація про ресурс, яка є публічно доступною, збір відбувається як пасивно (без прямої взаємодії з ресурсом) так й активно (включає пряму взаємодію з ресурсом). Головним інструментом на даному етапі є Nmap (Network Mapper), за допомогою якого можна легко просканувати порти на системі. Отримана інформація може бути дуже корисною у формуванні вектору атаки. Потрібно звертати увагу на архітектуру ПЗ та API, застосовані фреймворки та інструменти, структури запитів, як зберігаються дані користувача (токен), тощо [5]. Також, хоча й здається очевидним, але бувають випадки, коли розробники при тестуванні залишають у коді дані від акаунтів та іншу важливу інформацію у якості коментарів (наприклад API-ключ), які потім при релізі функціоналу забувають прибрати, що дає потенційному хакеру пропуск у систему;

– стадія 2: комплексна розвідка. Дана стадія логічно продовжує попередню й спрямована на подальший збір інформації про систему. Сюди входить OSINT (Open System Intelligence) розвідка, що включає в себе пошук інформації про систему чи її співробітників у відкритих джерелах. Прикладом може бути перегляд LinkedIn сторінки працівника компанії на

предмет переліку технологій, з якими даний працівник працює. Також на даному кроці проводиться фаззінг (пошук URL-параметрів та контенту, який зберігається на сервері);

– стадія 3: аналіз та обробка зібраної інформації. На даному кроці, ґрунтуючись на даних, отриманих у попередніх кроках, спеціаліст приступає до пошуку та ідентифікації вразливостей. Існують певні бази даних, які містять в собі усі відомі вразливості. Серед найбільш популярних – база даних CVE (Common Vulnerabilities and Exploitations), в якій за отриманим фреймворк та його версією наприклад можна подивитись чи наявні в ньому якісь вразливості й чи можуть вони бути корисними. Також на даному кроці відбувається пошук PoC, тобто випробуються припущення спеціаліста до можливих вразливостей.

Друга фаза – проведення атаки, включає в себе наступні три кроки:

– стадія 4: експлуатація вразливостей. На даному етапі проводяться заплановані на попередньому кроці активності по проникненню у систему. Атаки можуть включати: Reverse-Shells, Ін’єкції, Brute-forcing паролів, тощо;

– стадія 5: локальна енумерація. Даний крок схожий на Кроки 1-2, лише відмінністю є те, що на даному кроці спеціаліст потрапив в систему й досліджує її зсередини. Основною метою є – знайти якомога більше шляхів для подальшої експлуатації та отримати якомога більше інформації, яка не є публічною;

– стадія 6: постексплуатація. На даному кроці спеціаліст за допомогою інформації, отриманої на минулому кроці намагається закріпитись у системі шляхом створення backdoor (функціоналу, який допоможе спеціалісту отримати доступ до системи в будь-який момент), а також з можливих активностей на даному кроці можуть бути – ескалація привілеїв, завантаження шкідливого коду, копіювання ключів та паролів, тощо.

Третя фаза – підготовка звіту, містить у собі останні три кроки:

– стадія 7: документування. На даній стадії зазначаються усі дії, пророблені спеціалістом (пентестером), які відбувались під час тестування та призвели до експлуатації;

– стадія 8: представлення даних. На даному кроці оформлюється доказова база, це потрібно для того, щоб знайдені вразливості дійсно є вразливостями. Доказами можуть бути записи екрану, скріншоти, посилання, тощо;

– стадія 9: формування висновків. Даний крок є важливим для замовника та його компанії в першу чергу, адже на даному кроці спеціаліст з безпеки презентує сформований звіт разом з переліком рекомендацій та висновків.

Саме перші дві фази спрямовані на тестування безпеки й приводять до зламу системи, в той час коли, коли третя фаза виключно для донесення результатів тестування до зацікавлених осіб (рис. 1.1). Остання фаза безумовно є також важливою, але в даній роботі фокус буде на перших двох фазах, а саме на вразливостях.

Рисунок 1.1 – Cyber Kill Chain (перші дві фази життєвого циклу на проникнення)

1.2 Хакери. Типи хакерів

Хакер – це особа, що намагається отримати несанкціонований доступ до комп'ютерних систем з метою отримання приватної інформації або виявлення вразливостей системи. Серед людей, що близькі до сфери діяльності у ІТ існує ще одне визначення терміну «хакер», згідно з ним, хакер – це досвідчений користувач комп'ютером. Але друге визначення більше відноситься до сленгу та в рамках даного дослідження поняття хакер буде трактуватись відповідно до першого визначення.

Тобто звичайного спеціаліста з проникнення від хакера відрізняє наявність (чи відсутність) дозволу на проведення тестових активностей у системі.

Хакери бувають різні, вони можуть переслідувати різні цілі, в залежності від цих цілей хакерів поділяють на доволі багату кількість типів (згідно деяких джерел виокремлюються до 15 типів хакерів), проте основними типами є (рис. 1.2):

– чорні капелюхи. Дані хакери є вмотивованими особистою та фінансовою вигодою, авжеж вони не мають потрібних дозволів й тестують системи зі зловмисними намірами через що даний тип хакерів є небезпечним;

– білі капелюхи. Хакери, що входять до цієї групи також відомі як «етичні хакери». Вони роблять все те саме, що й чорні капелюхи, але в них є юридичні дозволи від організацій на тестування їх систем;

– сірі капелюхи. Знаходяться на межі між чорними та білими капелюхами, виконують те саме тестування, проте не мають юридичних дозволів. Відмінністю між сірими та чорними капелюхами є те, що знайдені вразливості доносяться до відома компанії, у межах якої були знайдені дані вразливості. Авжеж за це сірі капелюхи очікують на фінансову винагороду.

Як вже зазначалось, існують більше різновидів хакерів, серед яких є зелені капелюхи (хакери-новачки), червоні капелюхи (переслідують чорних

капелюхів та зламують їх), блакитні капелюхи (спеціалісти на проникнення у компаніях, що тестують системи перед їх запуском).

Рисунок 1.2 – Типи хакерів

Існують також й хакери, що спонсуються державою (популярна практика наприклад у Північній Кореї, як це було з вірусом WannaCry у 2017 році). Ще окремим типом є хактивісти – активісти, які займаються хакінгом. Цей процес з їх точки зору є своєрідним протестом. Напевно найбільш популярною та відомою групою хактивістів є угрупування Anonymous [6].

Важливо зауважити, що попри згадане різноманіття існуючих видів хакерів, будь-яке несанкційоване тестування на безпеку, навіть на ранніх стадіях життєвого циклу тестування на безпеку (збір інформація, моніторинг чи аналіз мережі, тощо), несе за собою кримінальну відповідальність. Увесь матеріал, що наведено у даному дослідженні, викладено виключно у навчальних цілях.

1.3 OSINT мистецтво

Не для усього у процесі пошуку вразливостей та експлуатацію ресурсів потрібні знання коду, іноді уся потрібна інформація знаходиться прямо перед нашими очима й лише очікує поки ми її використаємо. Також потрібно

зазначити, що кажучи про кібербезпеку не завжди мова йде лише про Інтернет та хакерів у чорних худі з капюшонами, які пишуть багато коду, адже кібербезпека також відноситься й до фізичних пристроїв, таких як сервери. OSINT (Open-source intelligence) – це процес збору даних у публічно доступних ресурсах [7]. Цей підхід до отримання інформації є найбільш безпечним для потенційного хакера, адже таким чином не залишається слідів, бо:

– хакер напряму не взаємодіє з системою, яку хоче взламати;

– хакер може напряму взаємодіяти з системою, яку хоче взламати, але в межах звичайних дій, які роблять усі інші користувачі.

Під звичайними діями мається на увазі, що не використовуються ніякі сканери та інші автоматичні програми пошуку портів, вразливостей, тощо (що з дуже великою ймовірністю приверне увагу відповідальних спеціалістів з кібербезпеки), а використовуються лише доступні за замовченням дані, наприклад source code сторінки або електронні пошти працівників компанії.

Доволі об’ємним ресурсом для OSINT є соціальні мережі, чим наразі користуються не лише потенційні хакери, а й моделі штучного інтелекту. У соціальних мережах знаходиться доволі значна частина життя людини, чи то особистого, чи то професійного. Наприклад у LinkedIn (сайт для пошуку роботи та розвитку кар’єри) можна побачити контактні дані користувачів, стек технологій з якими вони працюють на роботі (що звужує вектор атаки для хакера), можна знайти колег по компанії, тощо.

Іноді навіть якась сама незначна та маленька деталь може стати дуже цінною інформацією. Користувачі зазвичай ледащі й не приділяють достатньо уваги власній кібер-гігієні. Таким чином створюються ненадійні паролі, які легко запам’ятати людині, тобто якісь асоціативні речі, як наприклад дата народження, ім’я дитини або домашньої тварини.

Саме такий випадок стався з доволі відомою людиною, колишнім президентом Сполучених Штатів Америки, Дональдом Трампом, здавалося б, що міри безпеки діючого на той момент президента повинні були бути

максимальними, але одній людині вдалося заволодіти акаунтом Трампа шляхом вгадування пароля. Пароль був авжеж ненадійний та легко асоціювався з людиною, пароль до речі був “maga2020!” [8].

Ще одним з поширених варіантів OSINT є Google Dorks – вбудовані фільтри у браузері, які можна використовувати у пошуковому рядку під час введення запиту. Уявімо ситуацію, що ви, як користувач Інтернета, маєте бажання отримати сторінки сайту www.bbc.com, які у своєму заголовку мають слово «Ukraine». Скоріше за все запит буде сформовано якось на кшталт «BBC Ukraine», що поверне дуже велику кількість результатів (більше 300 мільйонів). Авжеж дуже велика кількість зі знайдених результатів не є релевантною, наприклад у переліку можуть бути сторінки, де просто згадуються й BBC й Україна, але це не буде мати відношення до сайту BBC. Саме використання Google Dorks в даному разі допоможе отримати найбільш точні результати [9].

Серед найбільш поширених операторів існують:

– site: – обмежує пошук конкретним доменом або сайтом;

– filetype: – дозволяє шукати файли певного типу (наприклад, PDF, DOCX);

– intitle: – шукає сторінки з певним словом у заголовку;

– inurl: – шукає сторінки з певним словом у URL-адресі;

– ext: – аналогічний filetype, шукає файли з певним розширенням;

– cache: – показує кешовану версію сторінки;

– intext: – шукає певні слова в тексті сторінки.

Тому, повертаючись до ситуації, описаної вище, запит з використанням Google Dorks може виглядати наступним чином: «site: www.bbc.com intitle: Ukraine». Таким чином буде повернуто лише близько 300 тисяч сторінок, тобто у 1000 разів менше результатів. Наглядно результати з використанням Google Dorks й без наведено нижче (рис. 1.3).

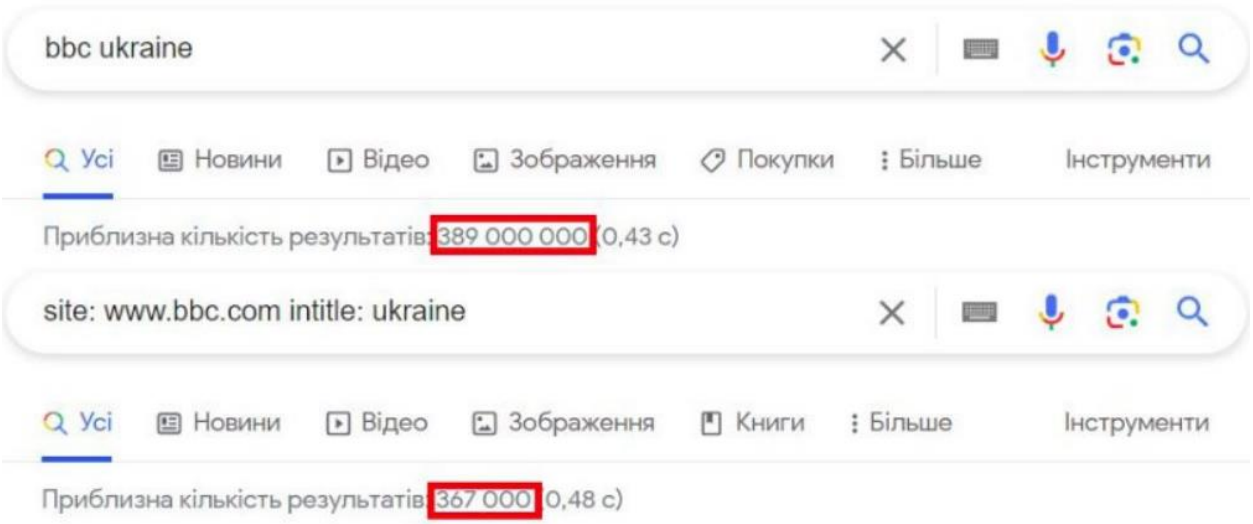

Рисунок 1.3 – Використання Google Dorks

Можливо по інформації, що була надана до сих пір не дуже зрозуміло чому саме розглядаються Google Dorks у розрізі кібербезпеки та OSINT, але за допомогою даних фільтрів потенційний хакер може отримати публічно-доступну інформацію, що відноситься до компанії або продукту, що є ціллю потенційного хакера. Таким чином можна отримати доступ до файлів, які недостатньо захищені чи вже є заархівованими, проте вони всеодно можуть містити конфіденційну інформацію, яка не передбачала публічного розголосу.

Ще одним схожим варіантом отримати публічно-доступну інформацію, що є потенційно конфіденційною можна завдяки /robots.txt та /sitemap.xml. Ці файли можуть мати трішки інші назви та розуміється наповнення, але вони використовуються для кращої індексації та оптимізації SEO (Search Engine Optimization) [10]. Якщо повернутись до попереднього прикладу, то майже 400 мільйонів результатів було повернено за 0.43 секунди, така гарна результативність відбувається саме завдяки індексації. Тож /robots.txt має у своєму описі інструкції для індексації, а саме (рис. 1.4):

- які User-Agent дозволені;
- які ендпоінти є доступними до індексації;

- які ендпоінти є недоступними до індексації.

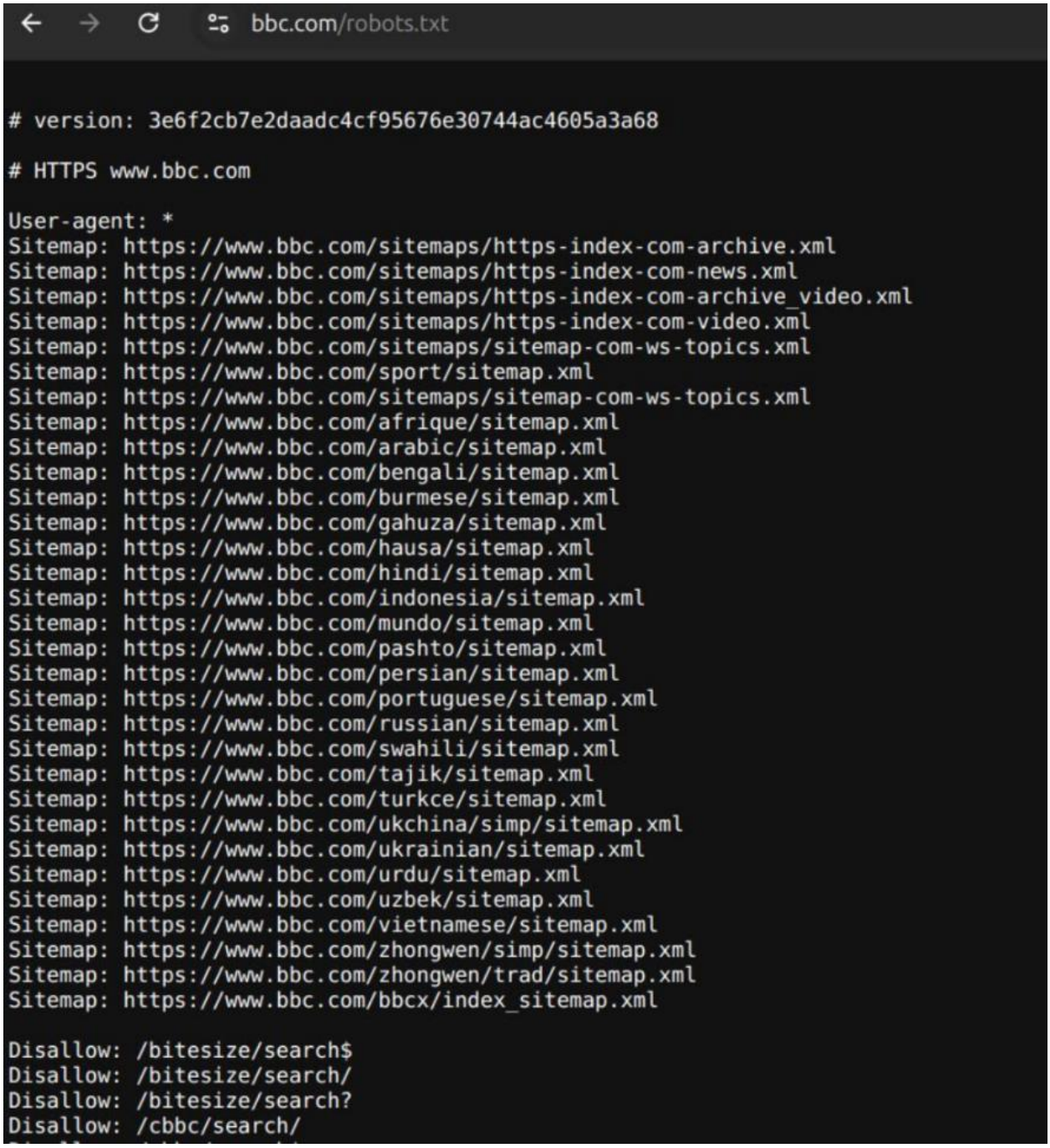

Рисунок 1.4 – robots.txt файл для сайту bbc.com

Так само й /sitemap.xml має певні інструкції для індексації (рис. 1.5):

– <urlset>: кореневий елемент для списку URL-адрес;

– <url>: обгортка для кожної URL-адреси;

– <loc>: адреса сторінки;

– <lastmod>: дата останнього оновлення;

– <changefreq>: частота зміни (наприклад, "daily", "weekly");
– <priority>: важливість сторінки в діапазоні від 0.0 до 1.0.

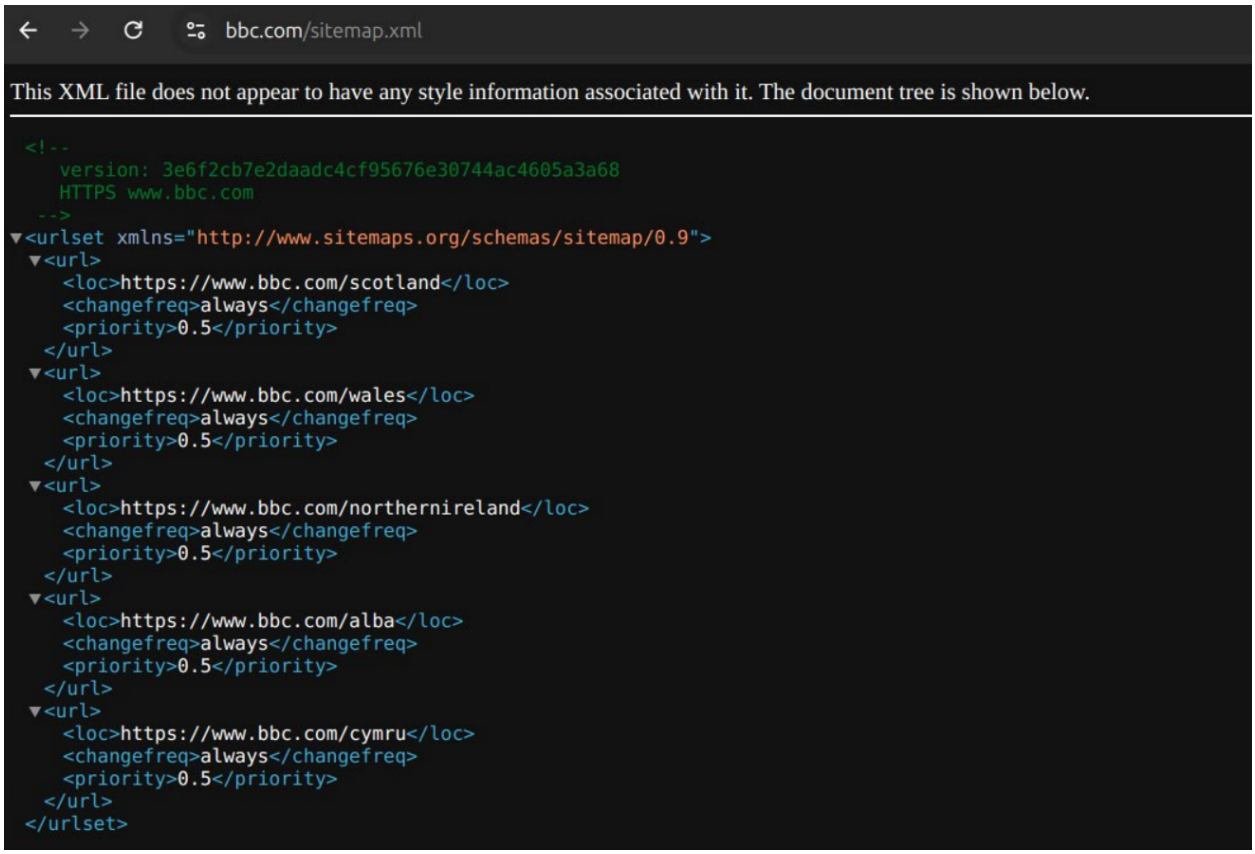

Рисунок 1.5 – sitemap.xml файл для сайту bbc.com

1.4 OWASP. OWASP TOP 10

OWASP (Open Web Application Security Project) – Це онлайн-спільнота, що створює та розповсюджує різноманітні ресурси та матеріали (статті, інструменти, методології, документації, тощо) в галузі безпеки вебзастосунків. Це угрупування є доволі популярним у сфері кібербезпеки, а їх рейтинг OWASP TOP 10 є головним показником популярності й критичності певних вразливостей [11].

OWASP TOP 10 – Рейтинг, що складається з 10 вразливостей, що є найбільш розповсюдженими та несуть найбільшу шкоду на даний момент. Рейтинг зазвичай оновлюється кожні 4 роки, остання опублікована версія

рейтингу датується 2021 роком, тож формування наступної планується вже у 2025 році [12] (рис. 1.6).

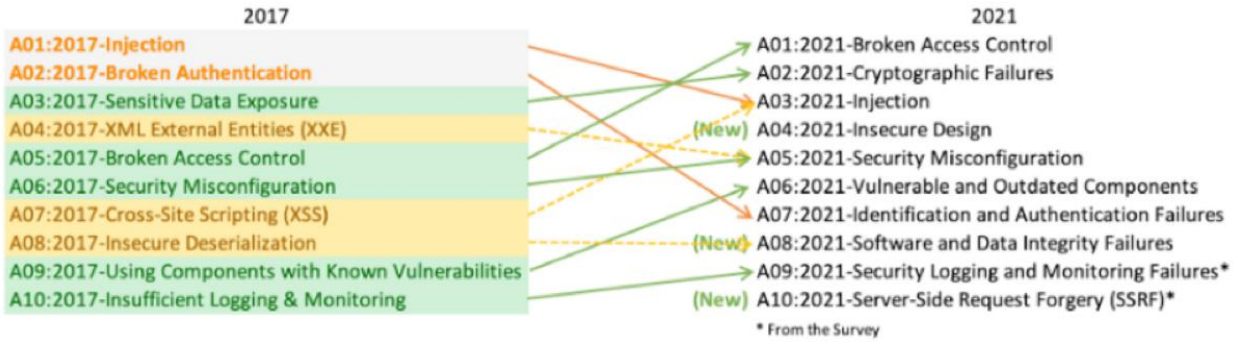

Рисунок 1.6 – Порівняння рейтингів OWASP TOP 10 за 2017 та 2021 роки

Згідно до рисунку вище багато вразливостей (7) залишились незмінними й присутні в обох рейтингах, змінились лише їх позиції у рейтингу. Також можна помітити, що деякі назви вразливостей було змінено, проте це не міняє суті самої вразливості.

Сфокусуємо увагу на найбільш актуальному рейтингу на даний момент та на вразливостях, занотованих у ньому. Broken Access Control – проблеми з правами у користувачів [13], які дозволяють отримати доступи до функціонала чи модулів, які повинні бути недоступні для даного користувача. Згідно рейтингу це найпоширеніша вразливість, яка полягає у:

– порушення принципу найменших привілеїв, що полягає в наданні користувачу мінімальної кількості потрібних привілеїв (наприклад користувач може створити акаунт та залишити коментар, але не може видалити акаунт іншого користувача);

– недостатні перевірки на наявні права у користувача, в такому випадку користувач може змінюючи URL або надсилаючи запити через Postman або Curl [14] отримувати доступ до сторінок та виконувати такі дії, які не передбачались для даного користувача.

Cryptographic failures – проблеми з недостатнім або відсутнім захистом даних, що передаються та зберігаються у системі. Дана вразливість раніше мала назву Sensitive Data Exposure та проявляється у випадках:

– коли дані передаються у вигляді простого тексту, без кодування, хешів, солі, іноді в URL за допомогою GET методу;

– коли використовуються слабкі, застарілі або самописні алгоритми для хешування даних (наприклад MD5).

Injection – включає в себе різноманітні ін’єкції (SQL, XSS, тощо), дані вразливості відтворювались у випадках:

– коли введенні користувачем дані не проходять валідацію;

– коли використовуються не параметризовані запити.

Insecure Design – нова категорія вразливостей, яка охоплює наступні проблеми:

– відображення повідомлень з помилками, які містять в собі чутливу інформацію [15] (наприклад, дають більше інформації, яка непотрібна звичайному користувачу й є корисною для потенційного хакера);

– неправильне чи недостатньо захищене збереження чутливої інформації (наприклад, паролів користувачів).

Security Misconfiguration – ряд проблем, пов’язаних з недостатніми або відсутніми налаштування та конфігураціями, серед яких:

– певні функції, що не несуть ніякої користі є увімкненими (наприклад, порти, що не використовуються мають відкритий статус);

– повідомлення з помилками відображають невідформатований текст помилки, а такий, який генерує IDE, що розкриває stack traces та іншу важливу інформацію.

Vulnerable and Outdated Components – ряд проблем, пов’язаних з використанням компонентів (програм, браузерів, операційних систем тощо), які містять в собі вразливості та не оновлені до останньої версії.

Identification and Authentication Failures – сукупність недоробок, яка дозволяє потенційному хакеру користуватись спеціалізованими інструментами для взлому:

– відсутність ліміту запитів, таким чином хакер може скористатись автоматизованим підбором даних користувача;

– перевикористання ідентифікатор сесії.

Software and Data Integrity Failures – вразливості, що можуть бути викликані через недбале ставлення до перевірки валідності таких даних, як сертифікати або використання бібліотек та модулів від сумнівних розробників [16].

Security Logging and Monitoring Failures – труднощі, що виникають через недостатнє, відсутнє або незрозуміле логування, це може виражатись у вигляді:

– невдалі спроби логіну не відображаються під час логування, що може бути проблемою у моніторингу brute-forcing атак, коли потенційний хакер підбирає комбінацію логіну та пароля користувача;

– логи з помилками, що генеруються, відображають незрозумілі інформацію, яка не допомагає у розумінні проблеми та потенційної причини такої поведінки.

Server-Side Request Forgery – вразливість, яка полягає у відсутній або недостатній валідації вводу користувача. Дана вразливість відтворюється, якщо користувач якимось чином заставить систему зробити запит на вказаний користувачем ресурс [17]. Без належної перевірки це може призвести до випадку, коли користувач вказує URL якогось шкідливого сайту, й сервер робить туди запит, отримує шкідливий файл у відповідь, який потім в залежності від контенту може спричинити усілякі негоди (reverse-shell наприклад).

1.5 Постановка задачі дослідження

Безпека, хоч в віртуальному світі, хоч в реальному завжди була пріорітетом й досі залишається актуальною потребою. Кількість відкритих вакансій, пов’язаних з кібербезпекою лише зростає, що в черговий раз підкреслює важливість даного ремесла. Саме тому була поставлена задача розробки комплексного методу тестування на проникнення у різноманітні системи. В рамках дослідження буде розібрана певна кількість відомих та найпоширеніших вразливостей (більшість з яких входять до рейтингу OWASP TOP-10).

Об’єктом дослідження є процес тестування тестових сайтів на різноманітні вразливості (міжсайтовий скриптинг, ін’єкції, атаки на стороні API, проблеми з аутентифікацією та авторизацією, знаходження непублічного контенту) з наданням усієї необхідної теоретичної інформації та демонстрацією вразливостей на практичному досвіді за допомогою віртуальних машин, які надаються платформою HackTheBox.

Метою дослідження є розробка комплексного підходу до тестування безпеки веб-ресурсу, а саме створення інструкції для тестувальників на проникнення, що стане у нагоді перед введенням веб-ресурсу у експлуатацію

Для досягнення мети необхідно сфокусуватись на вирішенні наступних завдань:

– проведення мережевої та технічної розвідки;
– проведення комплексної розвідки;
– аналізування та оброблення отриманої інформації;
– експлуатування вразливостей;
– проведення локальної енумерації.

Наведені вище завдання відповідають першим шести крокам, наведеним у життєвому циклі тестування на проникнення та складають The Cyber Kill Chain фреймворк. Щодо задачі по експлуатації вразливостей, то в рамках неї буде проведена саме атака по різних векторах, серед яких будуть:

– міжсайтовий скриптинг;

– XSS ін’єкція;

– SQL ін’єкція;

– поламана авторизація;

– API атаки;

– brute-forcing даних;

– розкриття контенту, який не є задокументованим та публічним (є прихованим);

– reverse shell.

2 ЯК ПРАЦЮЮТЬ ВЕБЗАСТОСУНКИ. РОЗУМІННЯ ВРАЗЛИВОСТЕЙ

2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура – це модель обробки даних, яка розділяє клієнтські та серверні функції в комп’ютерних мережах. Основні компоненти цієї архітектури – це клієнти, сервери та мережа, яка їх з’єднує [18]. Розглянемо детальніше принцип роботи цієї архітектури.

Основні компоненти:

– клієнт: це програма або пристрій, який надсилає запити на сервер для отримання ресурсів або послуг. Клієнти можуть бути різного роду: веббраузери, мобільні додатки, настільні програми тощо;

– сервер: це потужний комп’ютер або система, яка обробляє запити від клієнтів, надає ресурси та виконує обчислення. Сервери можуть бути спеціалізованими (наприклад, вебсервери, бази даних, файлові сервери) або універсальними;

– мережа: це система комунікаційних каналів, через які клієнти та сервери взаємодіють один з одним. Це може бути локальна мережа (LAN) або глобальна мережа (Internet).

Принцип роботи (рис. 2.1):

– запит від клієнта: клієнт ініціює взаємодію, формуючи запит. Це може бути, наприклад, запит на отримання вебсторінки, на збереження даних або на виконання певної обробки;

– передача запиту через мережу: запит відправляється через мережу до сервера. Це може здійснюватися за допомогою різних протоколів, таких як HTTP, FTP, SMTP тощо;

– обробка запиту на сервері: сервер отримує запит, обробляє його, виконуючи необхідні дії (наприклад, звертаючись до бази даних, виконуючи обчислення) і готує відповідь;

– відправка відповіді назад до клієнта: після обробки запиту сервер формує відповідь і надсилає її назад клієнту через мережу;

– отримання відповіді клієнтом: клієнт отримує відповідь від сервера та виконує подальші дії.

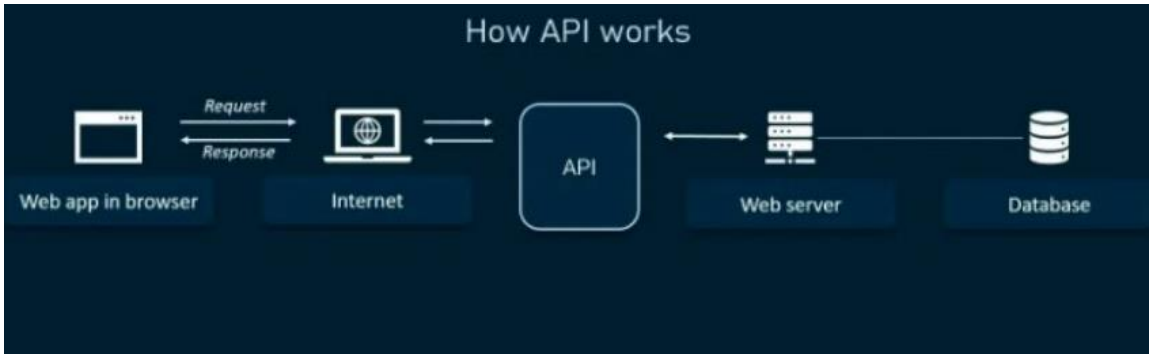

Рисунок 2.1 – Взаємодія клієнта з сервером

Як можна помітити з рисунка 2.1, у взаємодії приймає участь API (application programming interface, інтерфейс прикладного програмування) – це набір правил, протоколів та інструментів для створення програм [19]. За цими правилами й відбувається комунікація між клієнтом та сервером. Часто згадуючи API приводять у приклад роботу ресторану:

– backend (кухня) – місце, де відбувається уся логіка та оброблюються запити від клієнта (відвідувача ресторану);

– frontend (зал ресторану) – місце, з яким безпосередньо взаємодіють користувачі та де відвідувачі роблять свої замовлення (надсилають запити);

– API (офіціанти) – зв’язуюча ланка у цій схемі, які передає запит від відвідувача (замовлення) до поварів та повертає назад до відвідувача відповідь (замовлене блюдо).

Тобто API отримує на вхід певні дані від клієнта, далі передає їх до сервера й потім повертає до клієнта вихідні дані. Саме тому важливо приділяти увагу тестуванню усієї системи, а не лише якимось окремим її компонентам.

API підтримує усі CRUD (Create, Read, Update, Delete) операції, таким чином усі маніпуляції з даними є доступними [20]. Задля того, щоб сервер зрозумів яку саме операцію потрібно застосувати до даних існують HTTP методи.

Таблиця 2.1 – Приклади ендпоінтів (REST)

Ендпоінт	Метод HTTP	Дія
api/users	GET	Отримати усіх користувачів
api/users	POST	Створити нового користувача
api/users/1	GET	Отримати користувача з id=1
api/users/1	PUT	Оновити дані користувача з id=1
api/users/1	DELETE	Видалити користувача з id=1

Варто зауважити, що є різні архітектурні стилі імплементації API, серед найбільш поширених є REST (Representational State Transfer) та GraphQL [21]. Кожен архітектурний стиль має свої правила (або навіть правильніше рекомендації), так само як й свої переваги й недоліки. Порівняння цих двох архітектурних стилів не входить у рамки цієї роботи, уся наведена інформація відноситься до REST API (рис. 2.2).

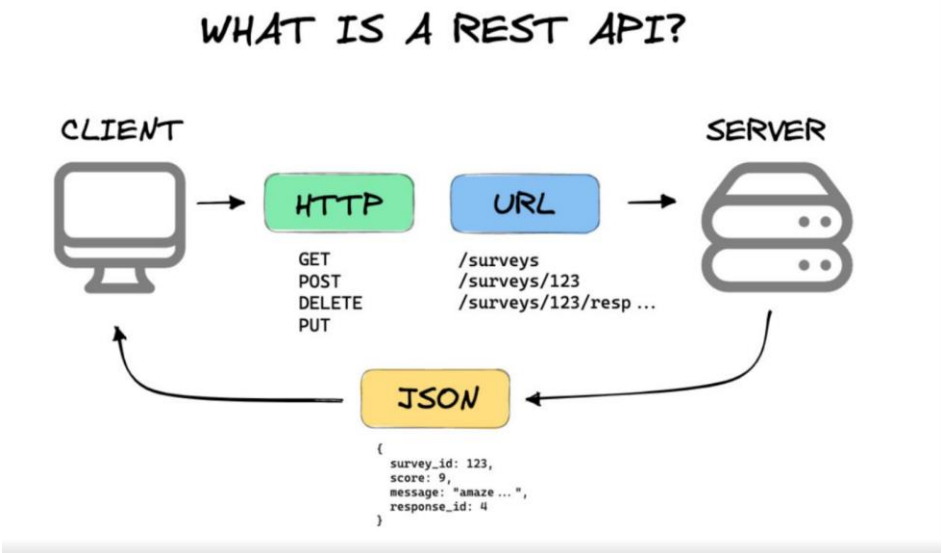

Рисунок 2.2 – Як працює REST API

2.2 Токен. Аутентифікація та авторизація

Токен – це зазвичай строкове значення, виглядає як якась нісенітниця, яке використовується для ідентифікації користувача у системі. Справа в тому, що HTTP є stateless протоколом, тобто він не запам’ятовує поточний стан [22] (рис. 2.3). Наприклад користувач ввів свої дані, потрапив у свій кабінет на платформі, а потім хоче скористатися функцією пошуку, наприклад. Після того як користувач введе якесь значення у пошуковий рядок та натисне на відповідну кнопку, щоб розпочати пошук, то сервер не зможе вас ідентифікувати, одночасно відбуваються сотні, а то й тисячі запитів від клієнта до сервера певної системи, й система не буде розуміти сама по собі який користувач які запити надсилає.

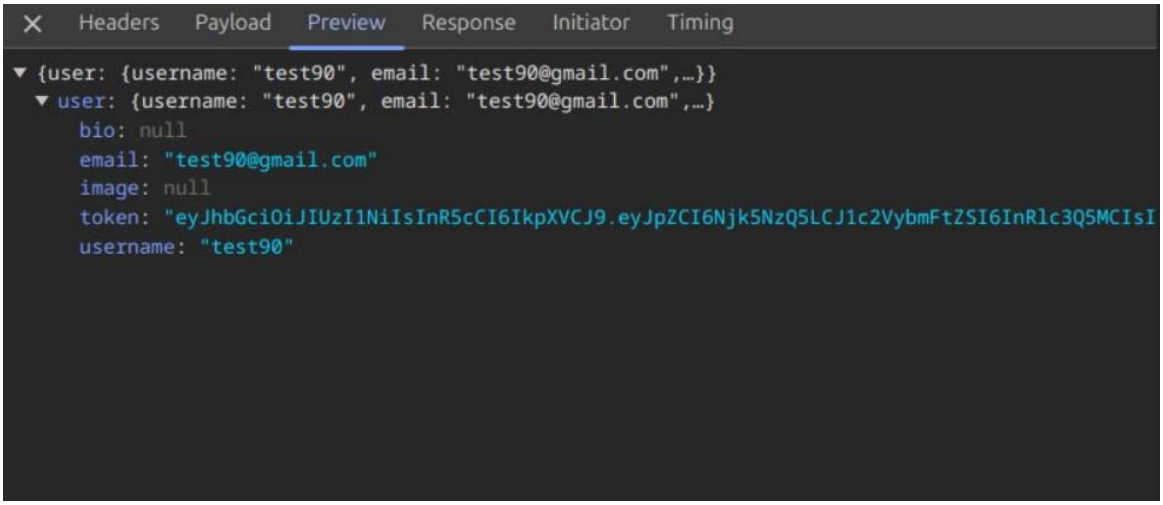

Рисунок 2.3 – Отримання токену

Цю проблему вирішують токени, які надаються користувачу після вдалої аутентифікації (процес ідентифікації користувача, перевірка, що користувач з наданими даними існує) й використовуються потім при авторизації (процес, що відбувається після аутентифікації й полягає в тому, що в залежності від результату аутентифікації користувач отримує права доступу) [23]. Наприклад на рисунку вище, даний токен надсилається після

вдалого логіну й надалі, даний токен надсилається з кожним запитом від клієнта й передається до сервера (рис. 2.4).

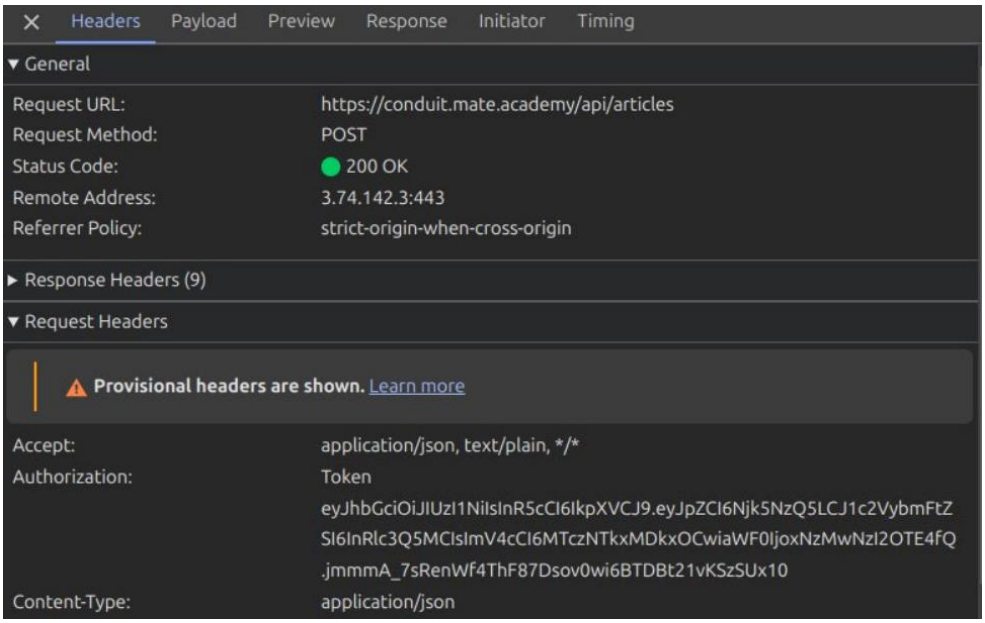

Рисунок 2.4 – Надсилання токену

У даному випадку токен надсилається за допомогою Authorization header, насправді, це найбільш поширений спосіб, проте існують й інші. У розрізі вразливостей варто приділити увагу тестування токенів та доступів, які вони надають, щоб запобігти випадкам несанкціонованого втручання у систему, адже іноді бувають випадки, коли користувачі можуть отримувати доступ до інформації та ендпоінтів, до яких би не мали мати доступу [24]. Ця проблема є настільки розповсюдженою та актуальною, що згідно рейтингу OWASP TOP 10 вона має назву Broken Access Control та посідає перше місце.

2.3 Міжсайтовий скриптинг

Міжсайтовий скриптинг – це тип вразливості в вебзастосунках, при якому зловмисник вставляє шкідливий JavaScript код у вебсторінки, що потім виконуються в браузерах інших користувачів [25]. Цей код може призвести

до різноманітних атак, таких як крадіжка cookies файлів, фішинг, зміна вмісту вебсторінок або виконання небажаних дій від імені користувача.

Процес експлуатації цієї вразливості можна описати у трьох кроках:

– ін’єкція коду: зловмисник вставляє шкідливий JavaScript код у форму, URL-адресу або інше поле введення на вебсторінці;

– виконання на стороні клієнта: коли інший користувач відкриває таку сторінку, браузер виконує код, який вставлений зловмисником. Це може статися, навіть якщо вебсайт на вигляд не містить нічого підозрілого;

– наслідки: за допомогою цього шкідливого коду зловмисник може красти інформацію користувача (наприклад, cookies та дані сесії), викликати фішингові атаки, перенаправити на шкідливі сайти або навіть здійснити дії від імені користувача, наприклад, відправлення повідомлень чи здійснення транзакцій (рис. 2.5).

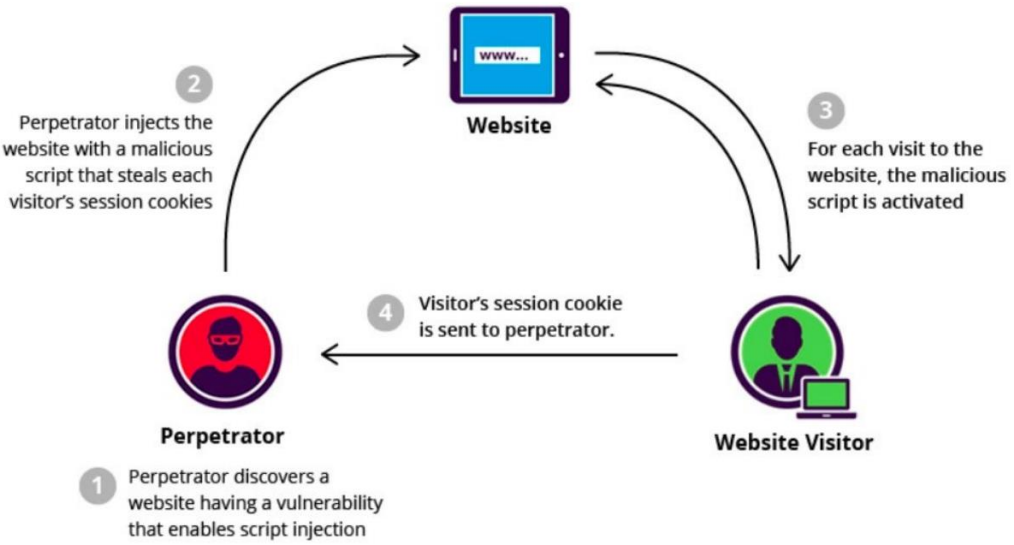

Рисунок 2.5 – Алгоритм експлуатації XSS вразливості

Міжсайтовий скриптинг поділяється на 3 типи:

– reflected XSS (віддзеркалений XSS): цей тип атаки виникає, коли шкідливий код передається через URL або параметри запиту. Наприклад, зловмисник може надіслати посилання на сайт, де параметри URL містять шкідливий JavaScript. Коли користувач переходить за посиланням, код

виконується на стороні клієнта. Віддзеркалений XSS зазвичай використовує безпосередньо параметри запиту, і шкідливий код не зберігається на сервері;

– stored XSS (збережений XSS): цей тип є більш небезпечним, оскільки шкідливий код зберігається на сервері. Наприклад, зловмисник може вставити JavaScript-код у форму на сайті, і цей код зберігається в базі даних або на сервері. Коли інші користувачі заходять на сторінку, де цей код збережений, він автоматично виконується у їхніх браузерах. Stored XSS може тривати набагато довше і зазвичай важче виявити та виправити;

– DOM-based XSS (DOM-орієнтований XSS): у цьому випадку атака використовує уразливості в клієнтському JavaScript-коді вебсторінки, а не серверну частину. Зловмисник маніпулює документом HTML або JavaScript через DOM (Document Object Model) [26] таким чином, що відбувається виконання шкідливого коду в контексті поточної сторінки. Цей тип атаки може бути важким для виявлення, оскільки шкідливий код не обов’язково передається на сервер.

2.4 SQL ін’єкції

SQL ін’єкція – це один із найбільш розповсюджених і небезпечних типів атак на вебзастосунки, що дозволяє зловмисникам вставляти шкідливі SQL-запити в поля вводу, що потім виконуються сервером бази даних. Це може призвести до компрометації конфіденційних даних, змін вмісту бази даних або навіть до повного контролю над сервером [27] (рис. 2.6).

Процес експлуатації цієї вразливості можна описати у трьох кроках:

– ін’єкція шкідливого коду: зловмисник вводить шкідливий SQL-код у форму вводу, URL або інше місце, яке використовує дані користувача для створення SQL-запиту. Наприклад, на сайті є форма для логіну, де користувач вводить ім’я та пароль. Якщо ці дані не обробляються належним

чином (наприклад, через валідацію або екранізацію), зловмисник може спробувати вставити шкідливий SQL-запит;

– виконання запиту: зловмисник може передати SQL-запит, який буде виконано на сервері бази даних. Це дозволяє йому маніпулювати даними, витягувати конфіденційну інформацію або навіть змінювати структуру бази даних;

– наслідки, серед яких: крадіжка даних (зловмисник може витягнути користувацькі особисті дані, такі як логіни, паролі, тощо), модифікація даних (атака може змінити або видалити дані в базі даних), отримання доступу до адміністративних функцій (через SQL-ін’єкцію можна отримати доступ до адміністраторських прав або навіть виконати довільні команди на сервері) та виконання довільного коду (якщо база даних підтримує спеціальні функції на кшталт через xp_cmdshell в SQL Server, то зловмисник може виконувати системні команди на сервері).

SQL ін’єкції поділяються на декілька видів:

– класична SQL-ін’єкція: зловмисник вставляє шкідливий SQL-код в поля вводу, такі як форма логіну або пошуковий запит;

– blind SQL Injection (сліпа SQL-ін’єкція): у цьому випадку, зловмисник не отримує прямого виведення результату запиту, але може оцінити істинність умов за допомогою додаткових запитів;

– union-based SQL Injection: у цьому випадку зловмисник використовує оператор UNION, щоб об’єднати результати оригінального запиту з результатами свого власного, які містять дані з інших таблиць. Це дозволяє зловмиснику отримати доступ до даних з інших таблиць в базі даних;

– time-based blind SQL Injection: зловмисник змушує сервер затримати відповідь на запит, якщо певна умова істинна. Це дає змогу отримати інформацію про структуру бази даних, не отримуючи прямого доступу до даних.

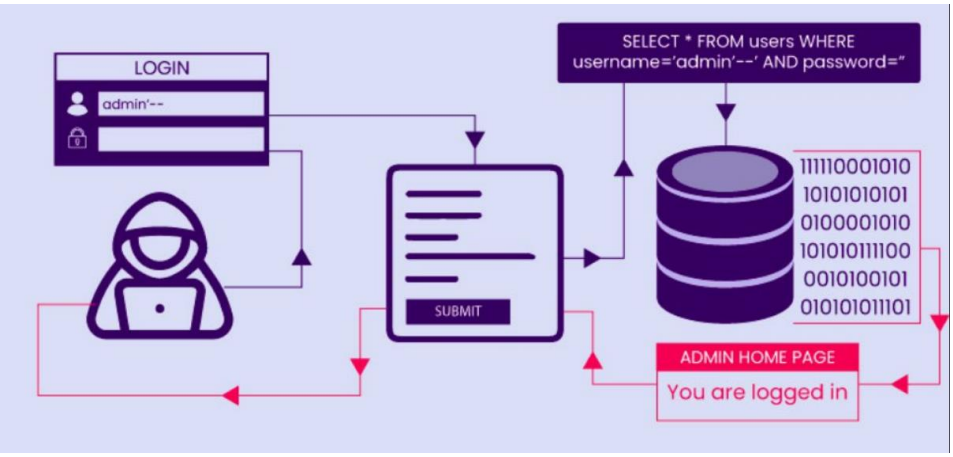

Рисунок 2.6 – Алгоритм експлуатації SQL-ін’єкції

2.5 Знаходження скритого контенту

Знаходження скритого або прихованого контенту за допомогою інструментів типу ffuf – це процес виявлення ресурсів, файлів або директорій, які не є очевидними або публічно доступними через стандартні HTTP-запити, але можуть бути доступні в рамках вебсайту чи вебзастосунку (рис. 2.7). Цей процес зазвичай використовується в етапах пентестів або для збору інформації (footprinting) при виявленні вразливостей, коли потрібно знайти «приховані» частини вебзастосунка, що можуть містити конфіденційну або важливу інформацію [28].

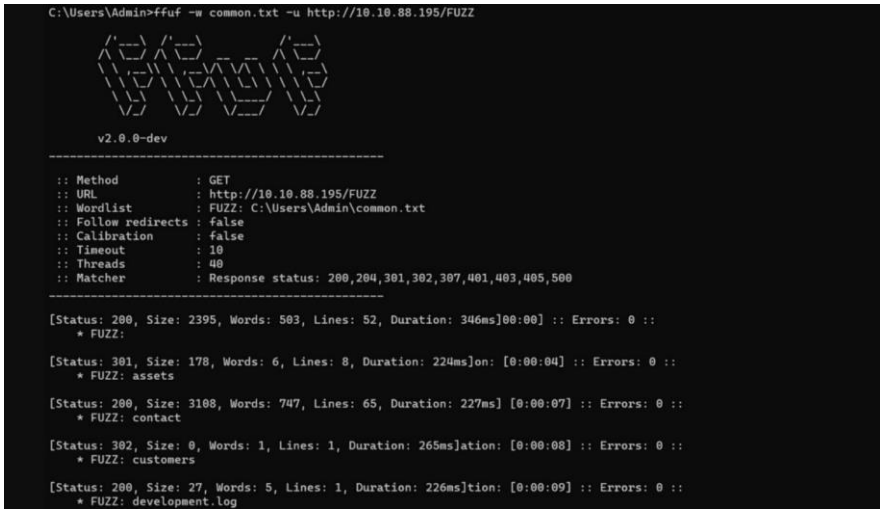

Рисунок 2.7 – Використання ffuf

Процес пошуку складається з наступних стадій:

– аналіз ідентифікації ресурсів: вебзастосунки можуть містити невидимі для користувачів сторінки, файли або директорії, які можуть бути доступні за прямими URL-адресами, але не відображаються у головному інтерфейсі сайту. Прикладами таких ресурсів є адміністративні панелі, старі або тимчасові файли, резервні копії, або навіть вихідний код застосунків, які не повинні бути відкритими для загального доступу;

– пошук за допомогою словників [29]: інструменти типу ffuf (Fast Fuzzer) або схожі (як gobuster, dirb, dirbuster) використовують техніку brute-force або словникового пошуку для перебору можливих URL шляхів на основі списку слів (словника). Наприклад, інструмент може автоматично відправляти запити на URL-адреси, такі як example.com/admin, example.com/upload, example.com/backup, і перевіряти відповіді сервера. Слова для пошуку зазвичай йдуть у вигляді списку, який включає типові директорії (наприклад, "admin", "test", "files"), типові файли (наприклад, "index.php", "config.js") або конкретні розширення (наприклад, .bak, .tar, .zip);

– перевірка статусу відповіді: кожен запит до певного URL може повернути різні статуси HTTP (наприклад, 200 OK, 301 Moved Permanently, 403 Forbidden, 404 Not Found). Інструмент записує ці відповіді і, залежно від коду статусу, визначає, чи є ресурс доступним. Якщо отримано код 200 (OK) або інші позитивні статуси (наприклад, 301), це свідчить про те, що ресурс існує і доступний, і його можна вивчити більш детально;

– оптимізація пошуку: щоб пошук був ефективнішим, можна використовувати різні техніки фільтрації, наприклад, задаючи конкретні розширення файлів (.php, .html, .js), або навіть фільтрувати за розміром відповіді сервера. Зазвичай атаки по пошуку скритих ресурсів включають варіанти з низьким рівнем помилок (найбільш вірогідні результати на основі історії запитів, структур директорій).

2.6 Network Mapper

Nmap (Network Mapper) – це потужний інструмент з відкритим вихідним кодом для сканування мереж, пошуку вразливостей та аудиту безпеки [30]. Він використовується для виявлення активних хостів у мережі, визначення їхніх відкритих портів, служб та операційних систем, а також для виконання інших типів тестів на безпеку, таких як виявлення вразливих сервісів чи виконання сканувань на наявність відомих вразливостей.

Nmap серед фахівців у сфері кібербезпеки зазвичай використовується для:

– виявлення активних хостів та пристроїв у мережі: Nmap дозволяє швидко отримати список всіх активних пристроїв, підключених до мережі. Це корисно як для системних адміністраторів (для картографування мережі), так і для пентестерів (для визначення потенційних цілей для атаки);

– аналіз відкритих портів: один із найбільш поширених способів використання Nmap – це виявлення відкритих портів на хості. Це дозволяє визначити, які служби (наприклад, HTTP, FTP, SSH) доступні на певному пристрої або сервері, що є важливим для подальшого тестування на вразливості [31];

– визначення версій програмного забезпечення: Nmap може виявляти не лише відкриті порти, але й визначати версії служб, які «слухають» ці порти. Це важливо для пентестерів, оскільки допомагає виявити застарілі або уразливі версії програм, що можуть бути ненадійними та містити вразливості;

– сканування на наявність уразливостей: Nmap може використовувати спеціальні скрипти для виявлення відомих вразливостей в програмному забезпеченні, що працює на відкритих портах. Це робить його корисним для виявлення небезпечних конфігурацій або експлойтів, доступних для використання зловмисниками;

– ідентифікація операційних систем та пристроїв: Nmap може намагатися визначити операційну систему хоста (OS fingerprinting), що дозволяє оцінити тип пристрою або його конфігурацію. Це може бути корисним для пентестерів для виявлення уразливостей, специфічних для певних операційних систем;

– мапування мережі та її топології: для великих мереж Nmap дозволяє створювати топологічні карти, що допомагає візуалізувати взаємозв'язки між хостами, маршрутизаторами та іншими пристроями;

– безпека та аудит мережі: Nmap часто використовується для аудиту безпеки та виявлення потенційних вразливостей. Адміністратори мереж можуть використовувати Nmap для перевірки, чи є у них відкриті порти або неналежно налаштовані служби, які можуть стати мішенями для атак.

Процес використання Nmap може виглядати наступним чином:

– сканування портів: основною функцією Nmap є сканування портів, що дозволяє визначити, які порти на цільовому хості відкриті і на яких працюють служби. Сканування може бути виконано за допомогою різних методів: TCP Connect scan (найпростішій метод, де встановлюється стандартне TCP-з'єднання з портами), SYN Scan (більш швидкий метод, при якому Nmap відправляє SYN-пакет – початковий пакет для TCP-з'єднання та аналізує відповідь) та UDP Scan (використовується для сканування відкритих UDP-портів);

– визначення версії служб: Nmap може здійснювати додаткові запити до відкритих портів для визначення, яка версія служби працює. Наприклад, якщо на порту 80 працює вебсервер, Nmap може отримати відповідь, яка дозволить визначити його версію (наприклад, Apache 2.4.1);

– операційна система та пристрій: за допомогою техніки OS fingerprinting, Nmap намагається визначити тип операційної системи, що працює на цільовому хості. Це відбувається через аналіз низки мережевих характеристик, таких як типи пакетів, TTL (Time To Live), значення поля Window Size тощо;

– Nmap Scripting Engine (NSE): Nmap має вбудовану систему скриптів, що дозволяє розширити функціональність інструмента. Ці скрипти можуть використовуватися для виявлення вразливостей, експлояцій, перевірки конфігурацій і багатьох інших задач. Наприклад, Nmap має скрипти для перевірки вразливостей Heartbleed, Shellshock, SMB і багатьох інших відомих вразливостей;

– аргументи та налаштування: Nmap надає величезну кількість параметрів, що дозволяють точно налаштувати сканування. Наприклад, можна вибрати, які порти сканувати, які методи сканування використовувати, чи хочете ви отримати докладний вивід або просто основну інформацію. Існують й інші аргументи та налаштування, наприклад такі, які дозволяють бути більш «скритним» (тобто системі складніше розпізнати спробу проникнення або ідентифікувати ту людину, яка за цим стоїть).

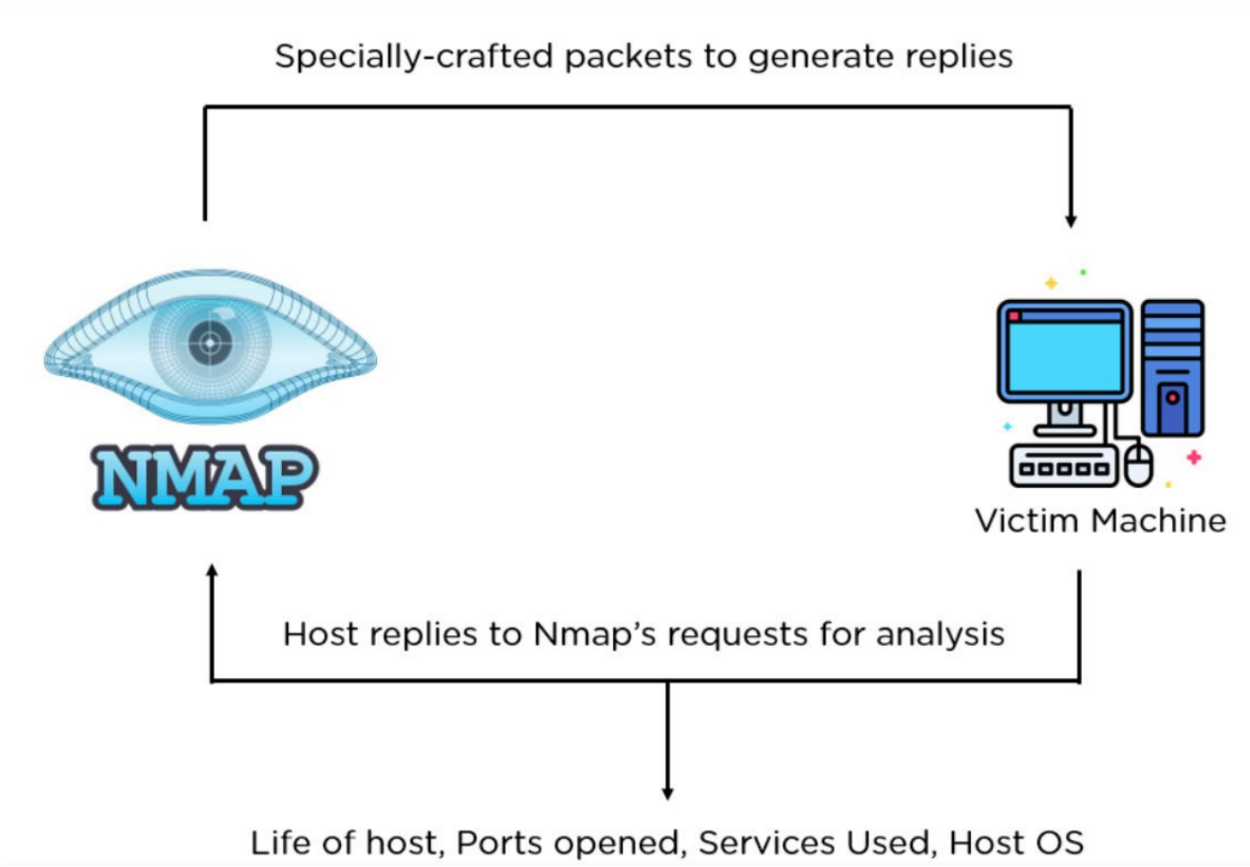

Рисунок 2.8 – Алгоритм роботи Nmap

2.7 Reverse shell

Reverse shell – це тип взаємодії між двома комп’ютерами, при якому зловмисник (атакуюча сторона) встановлює з’єднання з жертвою таким чином, що сам комп’ютер жертви ініціює підключення до атакуючої машини, замість того, щоб атака починалася від атакуючої машини [32]. Reverse shell є важливою частиною багатьох атак, зокрема при експлуатації вразливостей у програмному забезпеченні чи неправильно налаштованих серверах. За своєю природою дана вразливість є дуже критичною, адже надає потенційному хакеру в деяких випадках майже повний контроль над сервером.

Reverse shell використовується для наступних цілей:

– обхід брандмауерів та NAT: однією з основних причин використання reverse shell є обходження мережевих обмежень, таких як фаєрволи, NAT (Network Address Translation) або маршрутизатори. У традиційних підходах атакуючий комп’ютер має встановити з’єднання з жертвою, однак у багатьох випадках сервери чи мережі можуть блокувати вхідні з’єднання, особливо з неавторизованих джерел. З використанням reverse shell, підключення ініціюється зі сторони жертви, що робить з’єднання легшим для встановлення, оскільки фаєрволи та NAT частіше дозволяють вихідний трафік;

– отримання доступу до віддаленої машини: використовуючи reverse shell, зловмисник може отримати командний доступ до віддаленого комп’ютера (як правило, за допомогою командного рядка або інтерфейсу shell). Це дозволяє атакуючій стороні виконувати команди, маніпулювати файлами, читати конфіденційну інформацію, змінювати налаштування системи, а іноді навіть встановлювати додаткове шкідливе програмне забезпечення;

– анонімність та скритність: використання reverse shell часто є частиною тактики, спрямованої на мінімізацію виявлення під час атаки. Це дає атакуючій стороні можливість отримувати доступ до системи жертви, не

привертаючи увагу систем безпеки або відомих методів моніторингу трафіку (оскільки з’єднання виглядає як звичайний вихідний трафік);

– виконання команд або скриптів: reverse shell дозволяє зловмиснику не тільки отримати доступ до віддаленого хоста, а й виконувати команди в реальному часі. Це може включати запуск шкідливих програм, маніпуляції з даними або навіть перевстановлення цілого програмного забезпечення для подальших атак.

Reverse shell працює наступним чином (рис. 2.9):

– ініціація з’єднання: відмінність від стандартного shell або сполучення по протоколу SSH чи Telnet полягає в тому, що в reverse shell ініціатором з’єднання є не атакуюча машина, а комп’ютер жертви. Жертва виконує команду або використовує вразливість, що дозволяє їй підключитися до атакуючого комп’ютера;

– технічна реалізація: зловмисник встановлює «слухаючий» порт на своєму комп’ютері (наприклад, за допомогою утиліт на зразок Netcat або Metasploit). Потім зловмисник шукає спосіб націлити жертву (наприклад, через вразливість в програмному забезпеченні, фішинг або інші методи) і виконує команду, яка змусить жертву ініціювати з’єднання з атакуючим сервером. Як тільки жертва підключається, вона надає зловмиснику повний доступ до командного рядка або shell на своїй системі;

– комунікація: після того як жертва підключається до атакуючої машини, обидві сторони можуть взаємодіяти через командний рядок. Зловмисник може вводити команди в реальному часі, а результат виконання буде відправлений назад на його машину;

– скритність: зловмисники можуть використовувати шифрування трафіку або встановлювати з’єднання через нестандартні порти, що ускладнює виявлення під час моніторингу трафіку. Також для зменшення шансів на виявлення можуть використовуватися додаткові засоби маскування (наприклад, обробка трафіку через HTTP або HTTPS).

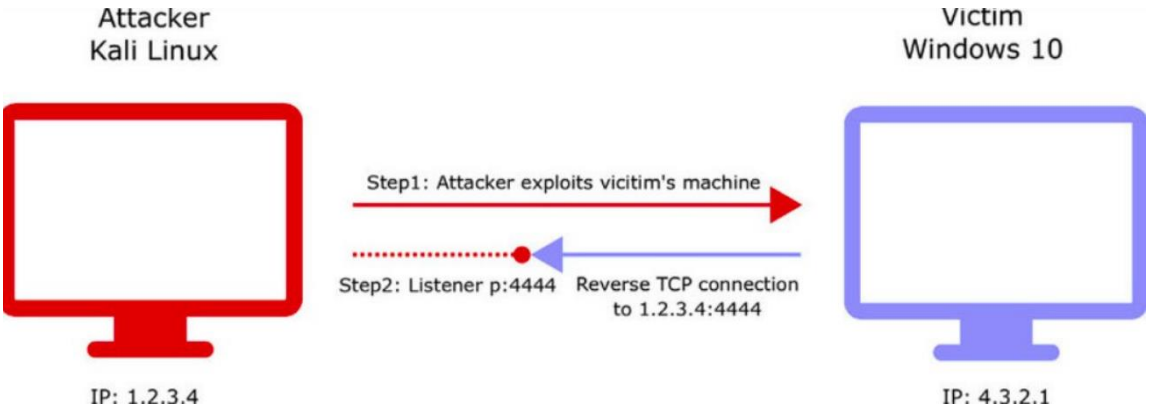

Рисунок 2.9 – Алгоритм експлуатації reverse shell

2.8 Платформа HackTheBox. Віртуальні машини

Дослідження проводилось за допомогою платформи HackTheBox, адже вона спеціалізується на кібербезпеці та має дуже велику кількість віртуальних машин та розроблених сайтів, які є вразливими до тієї чи іншої вразливості [33] (рис. 2.10). Використання допоміжної платформи, як HackTheBox, було обрано з міркувань приділення фокусу роботи саме тестуванню безпеки, інакше потрібно було б розробляти вразливі сервіси самотужки, щоб зайняло багато часу, не увійшло б в кваліфікаційну роботу та було б неможливо опрацювати весь той запланований об’єм роботи.

Рисунок 2.10 – Платформа HackTheBox (модулі)

Ідея використання віртуальних машин при тестуванні безпеки була актуальна ще давно, й ось чому [34] (рис. 2.11):

– ізоляція та безпека. Віртуальні машини дозволяють фахівцям створювати ізольовані середовища для проведення атак або інших досліджень без ризику для основної системи. Це важливо при тестуванні на проникнення або вивченні вразливостей, оскільки будь-які шкідливі дії, які можуть спричинити пошкодження або компрометацію системи, обмежуються лише віртуальним середовищем, не впливаючи на реальні операційні системи чи мережі;

– підтримка різних операційних систем. Віртуалізація дозволяє легко запускати різні операційні системи на одному апаратному забезпеченні. Це є особливо корисним при тестуванні безпеки в умовах, коли потрібно перевірити вразливості на різних платформах (наприклад, Linux, Windows, macOS) або на різних версіях програмного забезпечення. Це дозволяє фахівцям досліджувати та виявляти вразливості на конкретних системах без необхідності мати кілька фізичних машин;

– масштабованість та економія ресурсів. Віртуальні машини дозволяють запускати кілька середовищ на одному фізичному сервері або комп’ютері, що значно скорочує витрати на апаратне забезпечення. Це дозволяє створювати численні середовища для тестування без потреби в великій кількості фізичних машин. Для тестування різних атак на різних ОС або версіях програмного забезпечення досить просто створити нові віртуальні машини.

Це лише «вершина айсбергу» серед причин, по яким обирають віртуальні машини, коли справа доходить до тестування вразливостей. Ще однією не менш важливою перевагою використання віртуальних машин – це, як вже зазначалось, є можливість встановити будь-яку операційну систему, але так склалося, що найбільш user-friendly операційна система для фахівців з кібербезпеки – це Linux, а саме його дистрибутиви, такі як Kali Linux наприклад. В даній віртуальній машині за замовченням встановлені сотні

найбільш популярних програм, які використовуються для автоматизації процесу тестування на вразливості, а ще в даному дистрибутиві наявні усі популярні «словники» (набори найбільш поширених значень, таких як паролі, приховані сторінки та інше, що формувались протягом тривалого часу під час масштабних витоків інформації про користувачів через вдалі кібератаки, на кшталт RockYou). Проте іноді має сенс створювати власні «словники», наприклад коли потрібно вгадати пароль певної людини, то можна сформувати лист паролів, які може мати дана людина (наприклад якісь варіації з використанням дати народження, імені улюбленого домашнього любимця, тощо).

Авжеж ці програми можна встановити й на якусь іншу операційну систему, або можна написати свої аналоги існуючих програм, але для того, щоб не винаходити велосипед за замовченням використовують віртуальні машини, які вже містять усі необхідні інструменти для того, щоб провести процес тестування вразливостей.

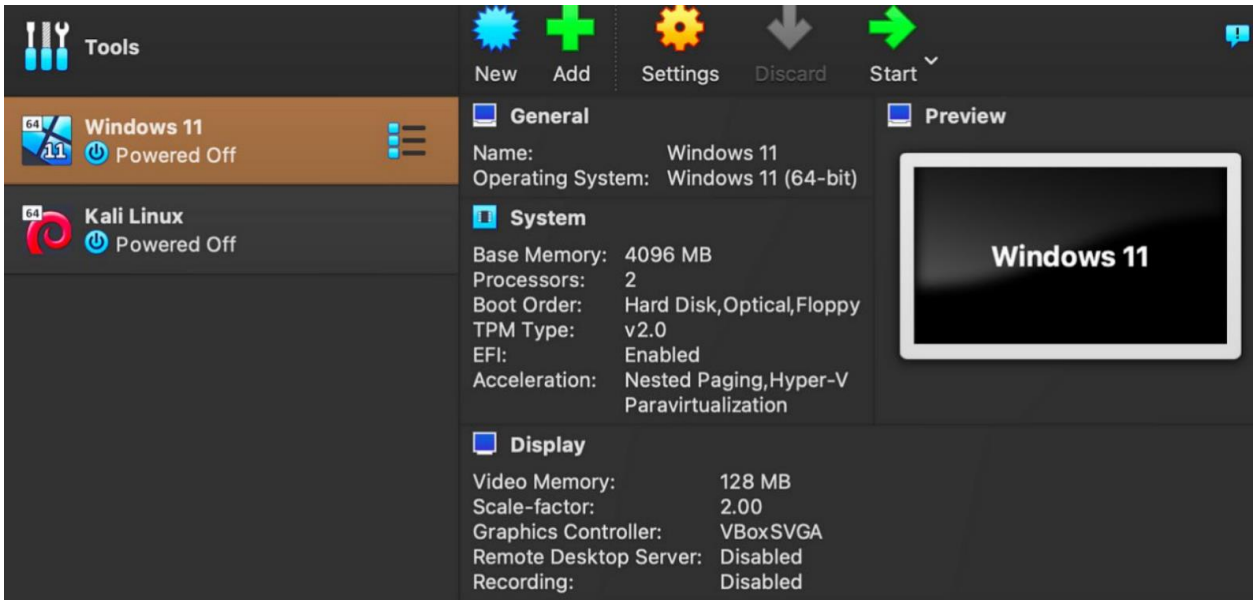

Рисунок 2.11 – Приклади віртуальних машин у Oracle VM VirtualBox Manager

3 ЕКСПЛУАТАЦІЯ ВРАЗЛИВОСТЕЙ

3.1 Експлуатація API Attacks

На рівні API можуть знаходитись багато помилок, пов’язаних з вразливостями. Першою поширеною помилкою може бути Broken Object Level Authorization (BOLA) або також дана вразливість відома як Insecure Direct Object Reference (IDOR) – це ситуація, коли система не перевіряє достатній рівень доступу у користувача, через що користувач може отримати доступ до конфіденційної інформації інших користувачів [35].

Задля демонстрації ряду вразливостей з розділу API Attacks буде використано сайт компанії «Inlanefreight», а точніше їх Swagger (документація по існуючим ендпоінтам, очікуваним відповідям, необхідним параметрам) [36]. За допомогою даного ресурсу можна також надсилати запити в реальному часі (рис. 3.1).

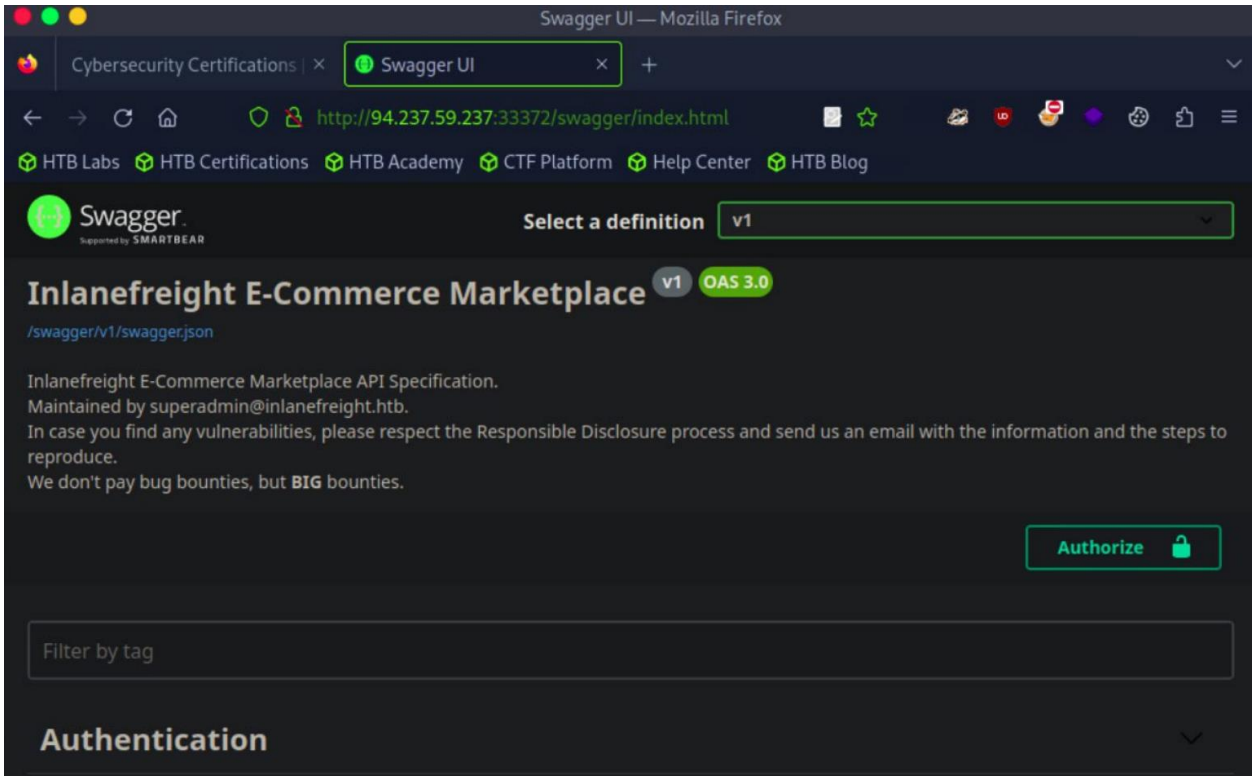

Рисунок 3.1 – Swagger продукту Inlanefreight

Дана платформа присвячена E-Commerce та має продавців й постачальників, у кожного користувача є своя роль та права, які надаються після надання пошти та паролю.

Далі потрібно авторизуватись як постачальник й ввести отриманий токен задля авторизації (рис. 3.2).

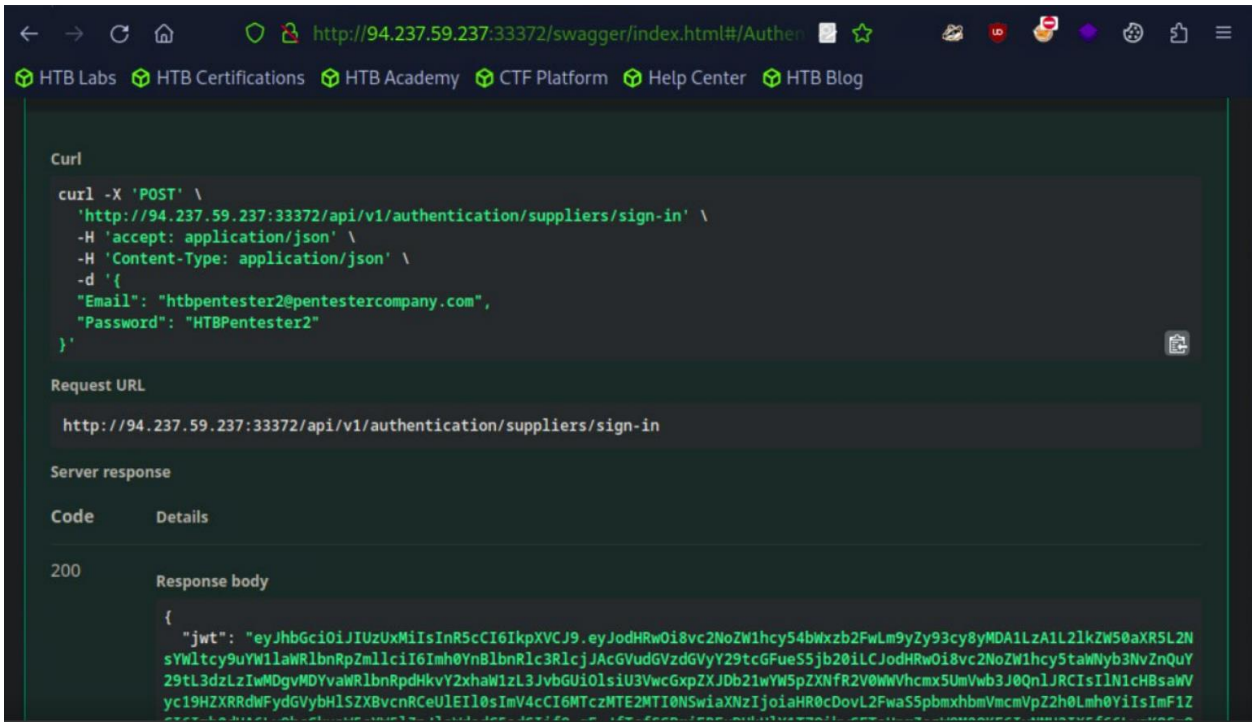

Рисунок 3.2 – Авторизація та отримання токену

За допомогою спеціального ендпоінта можна подивитись ролі, яку було надано користувача з такими правами доступу, які надав отриманий токен.

Отже, користувач має мати доступ до двох методів:

– отримання річного звіту компанії постачальника по унікальному номеру компанії;

– отримання квартального звіту постачальника по унікальному номеру постачальника.

За допомогою спеціального методу перевірено інформацію, яка відноситься до поточного користувача.

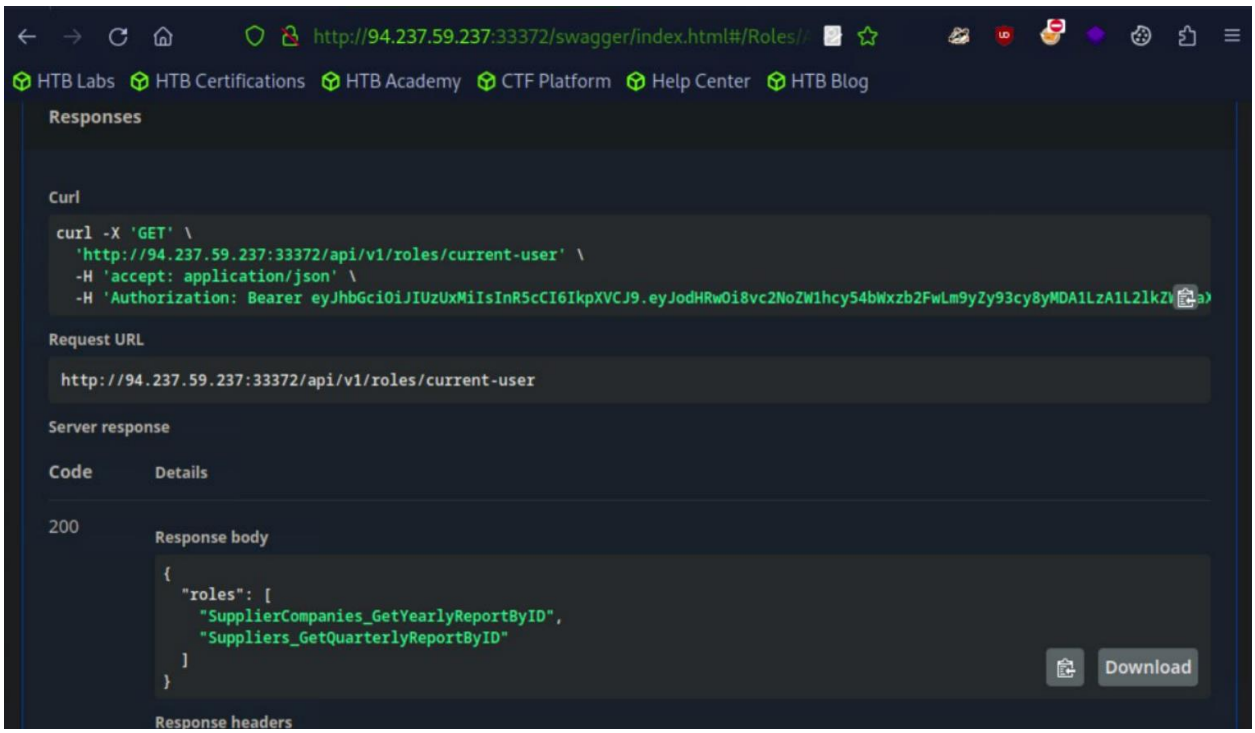

Рисунок 3.3 – Перегляд доступних ролей

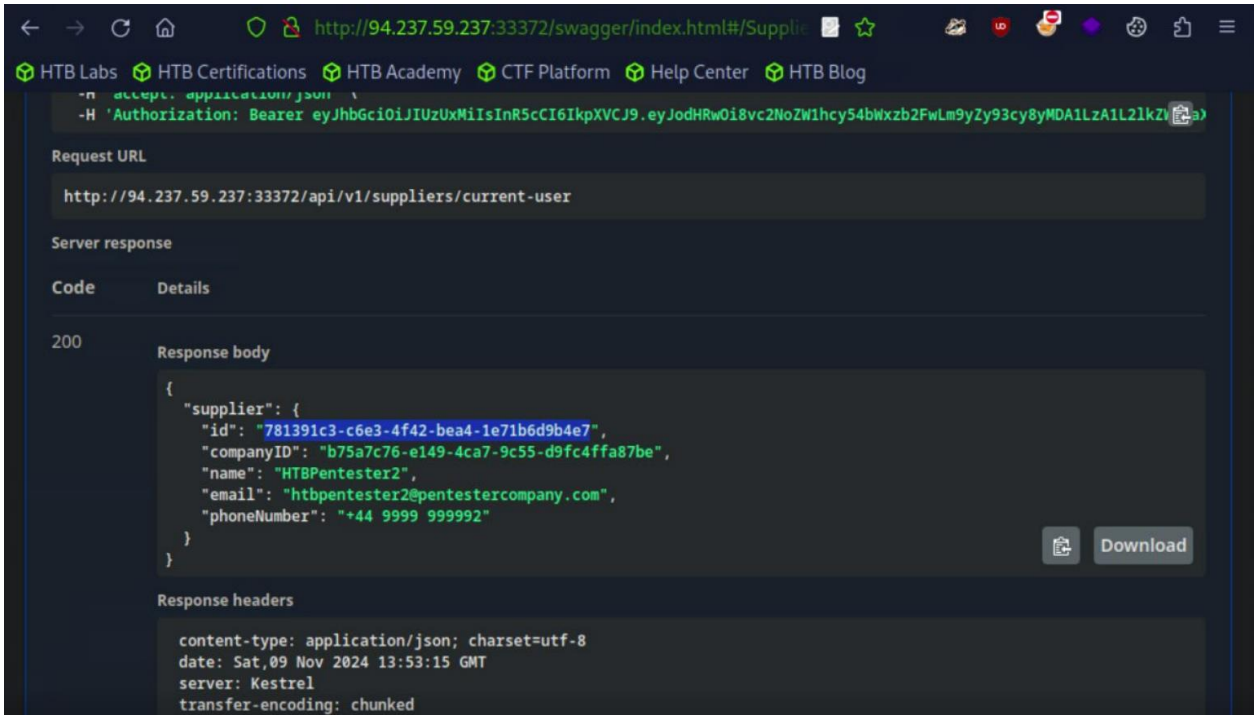

Рисунок 3.4 – Інформація по поточному користувачу

Серед важливого варто звернути увагу на значення атрибуту id. Тепер потрібно спробувати отримати квартальний звіт по id, який не відповідає даному користувачу.

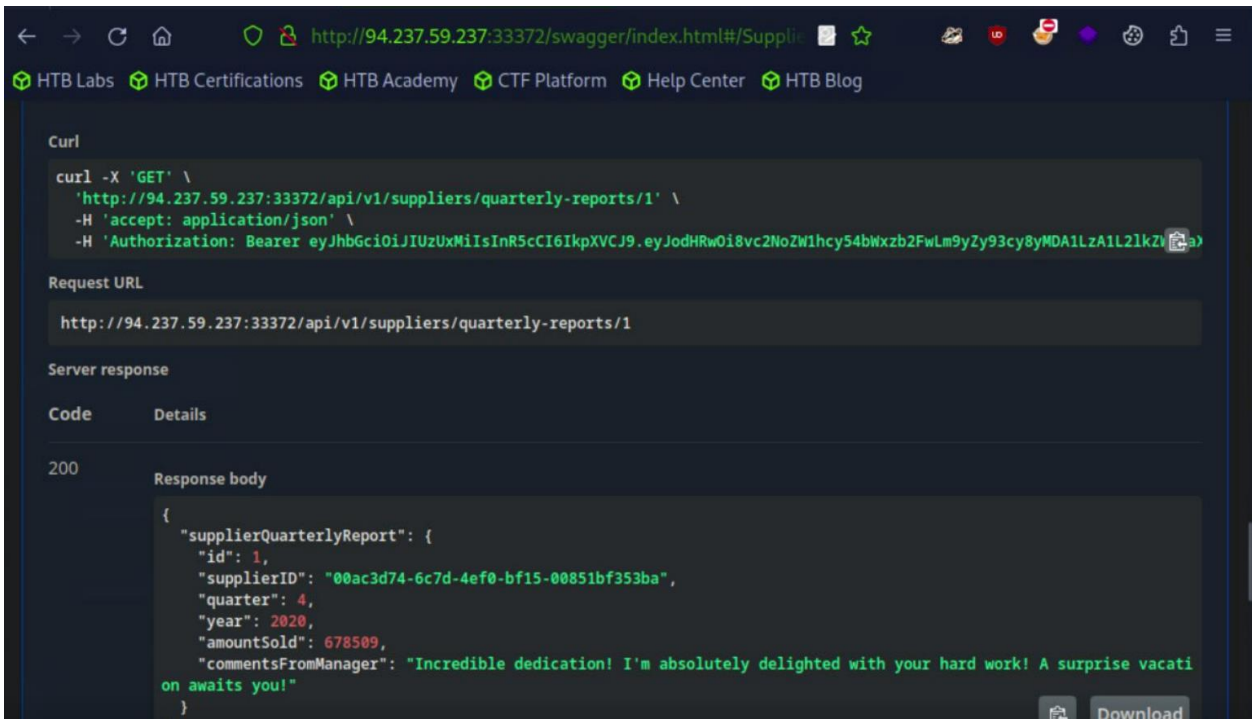

Рисунок 3.5 – Отримання квартального звіту іншого користувача

Таким чином користувач отримав доступ до інформації, в якої у нього не повинно було бути доступу. Вразливість заключається в тому, що потрібно не лише надавати права на користування методом, а ще й обмежувати користування лише типу сутностями, які належать певному користувачу.

Для того, щоб отримати флаг, що є симуляцією дуже важливої інформації, то можна спробувати використати цикл, щоб отримати перелік усіх квартальних звітів (що ймовірно потенційний злочинець й зробить), для цього потрібно скористатись наступним кодом (рис. 3.6).

Лістинг 3.1 Реалізація надсилання запитів за допомогою cURL:

```
$ for ((i=1; i<= 20; i++)); do
curl -s -w "\n" -X ‘GET’ \
  ‘http://94.237.59.237:33372/api/v1/suppliers/quarterly-reports/’$i’’ \
  -H ‘accept: application/json’ \
  -H ‘Authorization: Bearer <removed>‘ | jq
done
```

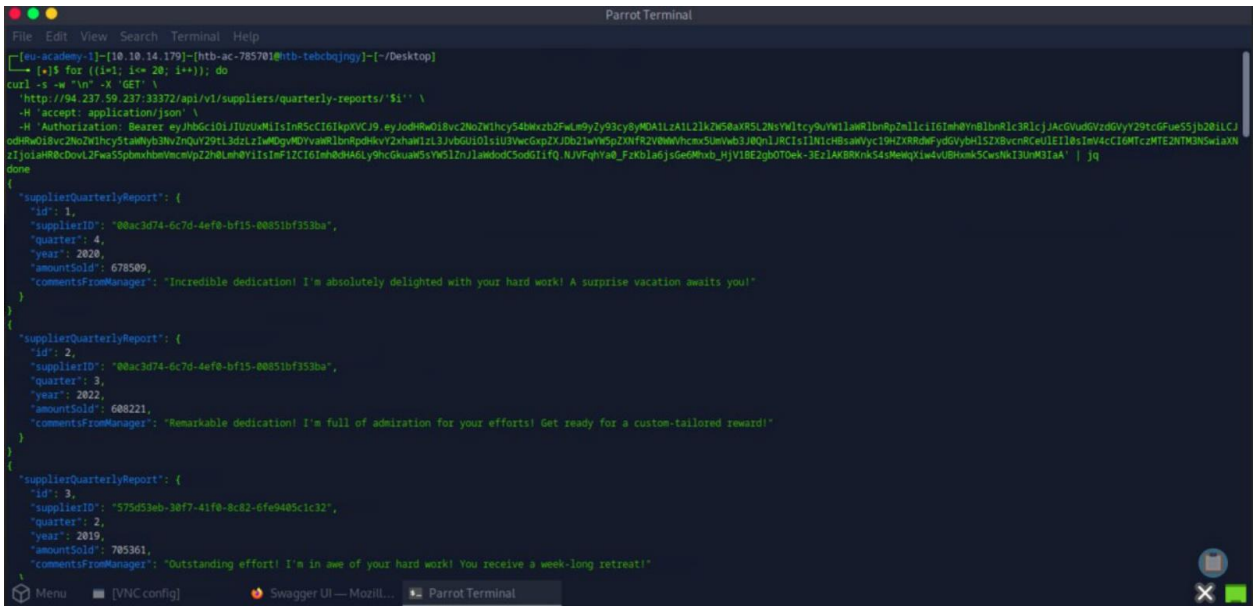

Рисунок 3.6 – Результат роботи команди

Таким чином можна надсилати довільну кількість запитів, в даному разі флаг знаходився у записі з id, що дорівнює 8 (рис. 3.7).

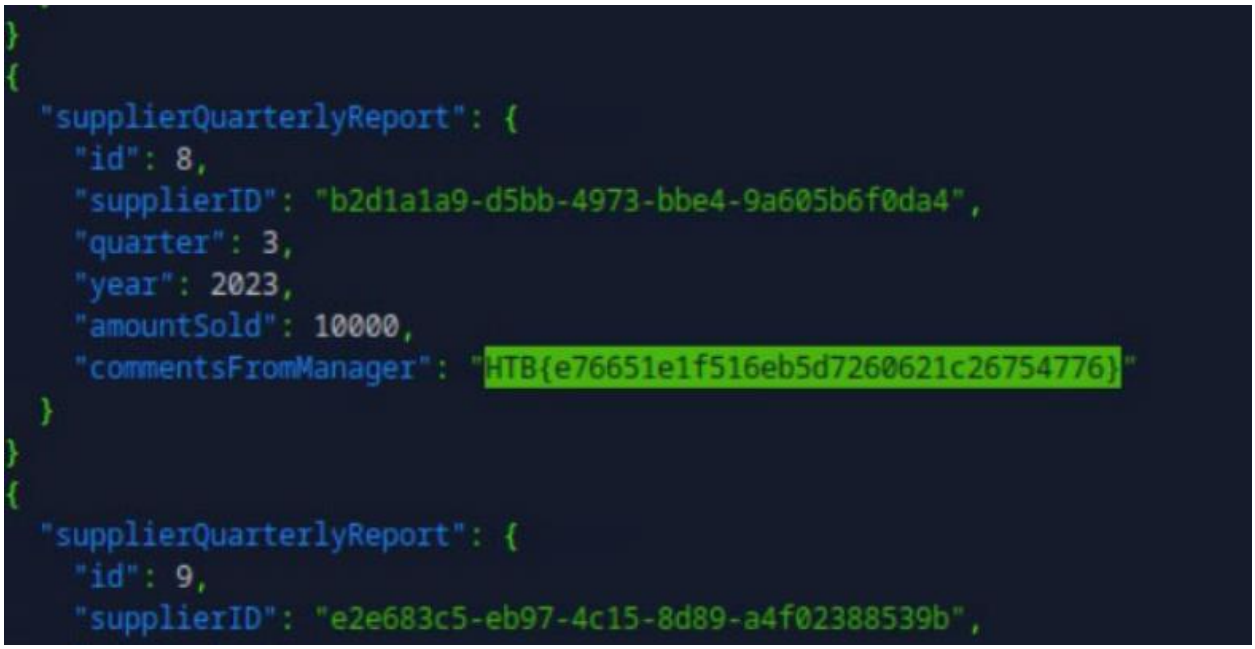

Рисунок 3.7 – Отримання флагу

Наступною вразливістю буде Broken Function Level Authorization, яка полягає у тому, що неавторизовані користувачі або користувачі з недостатніми правами доступу можуть користуватися недоступним

функціоналом. Після авторизації та отримання токену потрібно переглянути доступні ролі (рис.3.8).

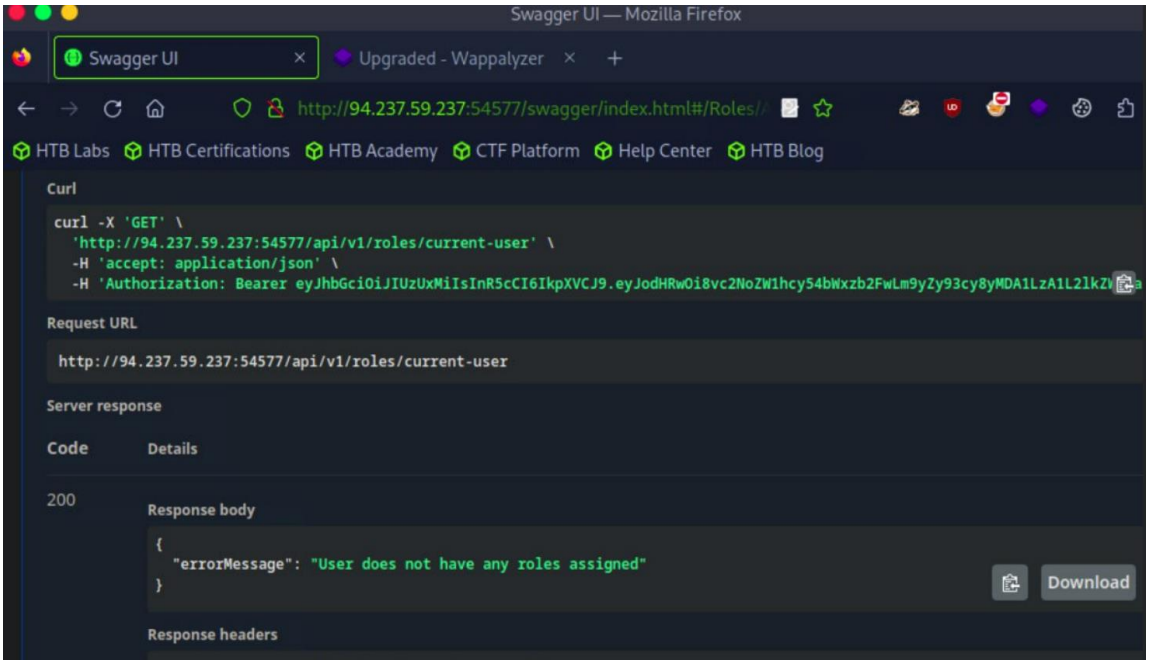

Рисунок 3.8 – Перегляд доступних ролей

При перегляді задокументованих ендпоінтів можна звернути увагу на /billing-addresses з HTTP методом GET. Зазначається, що даним функціоналом може скористуватись лише користувач, в якого CustomerBillingAddresses_GetAll роль наявна (рис. 3.9).

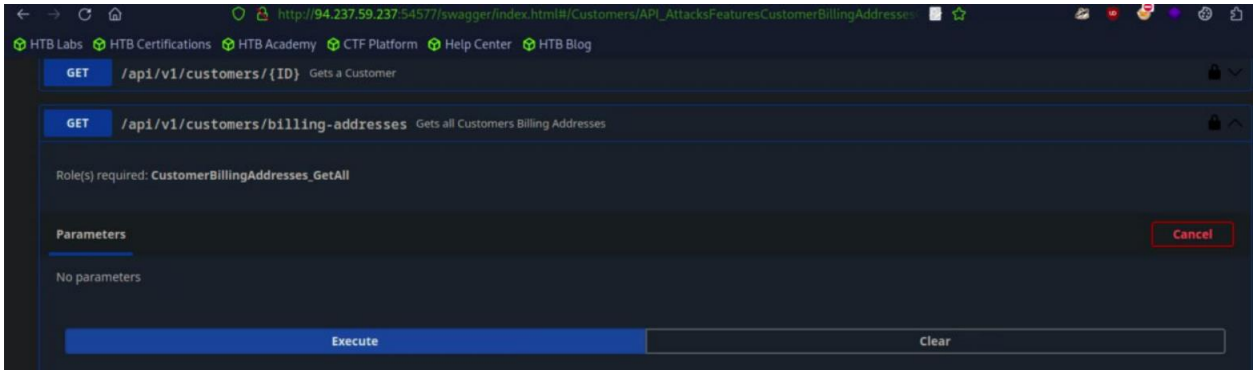

Рисунок 3.9 – Ендпоінт GET /billing-addresses

Даної ролі нема у поточного користувача, проте система дозволяє викликати даний метод та отримати масив (перелік) адрес доставки усіх

користувачів, що по суті є іншою вразливістю – Broken Object Property Level Authorization, адже це розкриття персональної інформації. Будь-то може отримати доступ до адрес усіх користувачів (рис. 3.10).

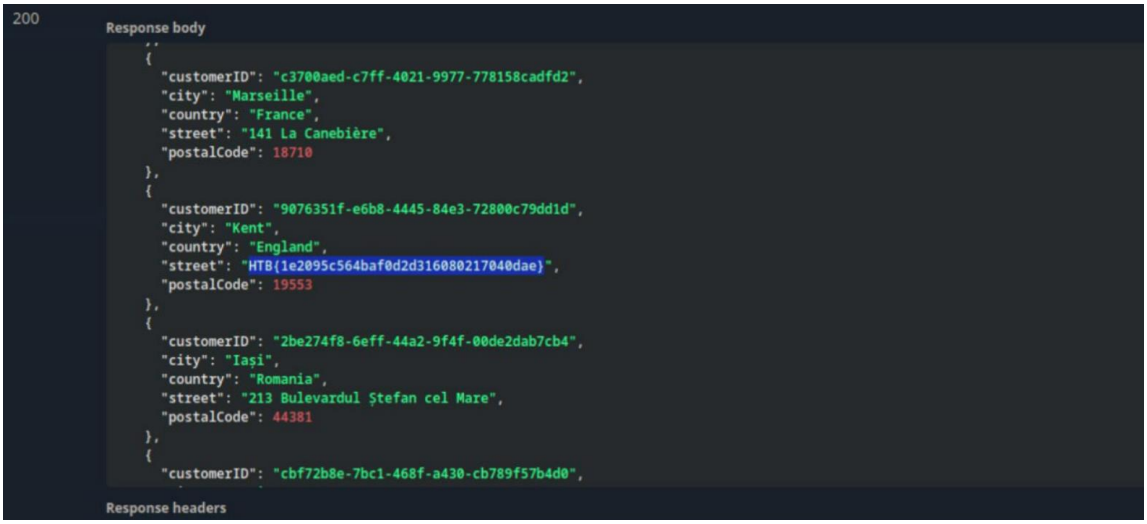

Рисунок 3.10 – Флаг

3.2 Експлуатація міжсайтового скриптингу

Задля демонстрації тестування на дану вразливість було обрано сайт, який є простеньким To Do списком. На сторінці наявне текстове поле, куди користувач вводить потрібну задачу (рис. 3.11). При натисканні на клавішу «Enter», значення з поля відображається на сторінці, а саме поле очищується (рис. 3.12). Також на сторінці присутня кнопка «Reset», яка очищує поточне значення у полі та на сторінці.

Далі було прийнято рішення подивитись як відреагує сайт, якщо у текстовому полі буде передано HTML код, тобто провести PoC (Proof of Concept), іншими словами демонстрацію потенційної наявності та загрози вразливості (рис. 3.13). Ця демонстрація не є шкідливою, бо ціллю є донести інформацію про можливі ризики, але не експлуатувати їх, адже таким чином може нанестись вагома шкода системі (рис. 3.14).

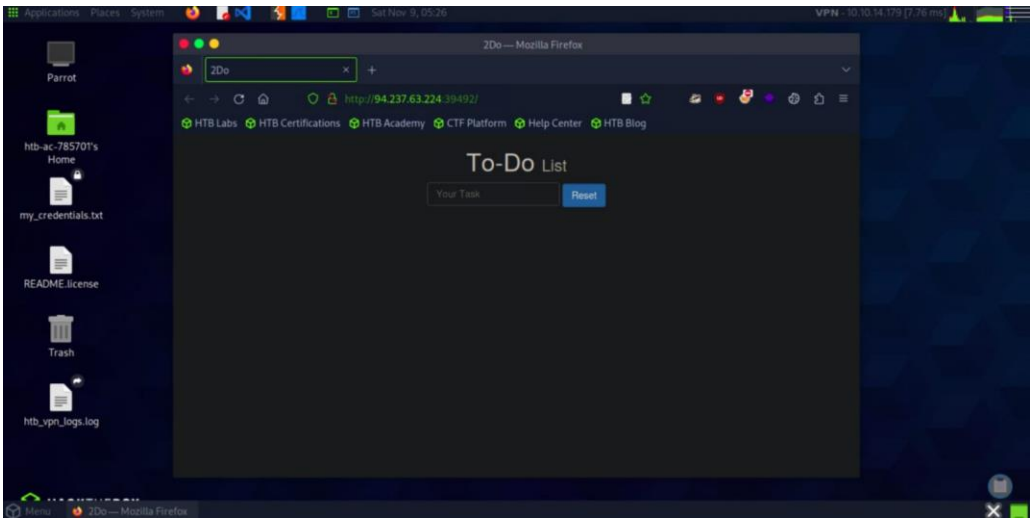

Рисунок 3.11 – Вразливий до XSS сайт

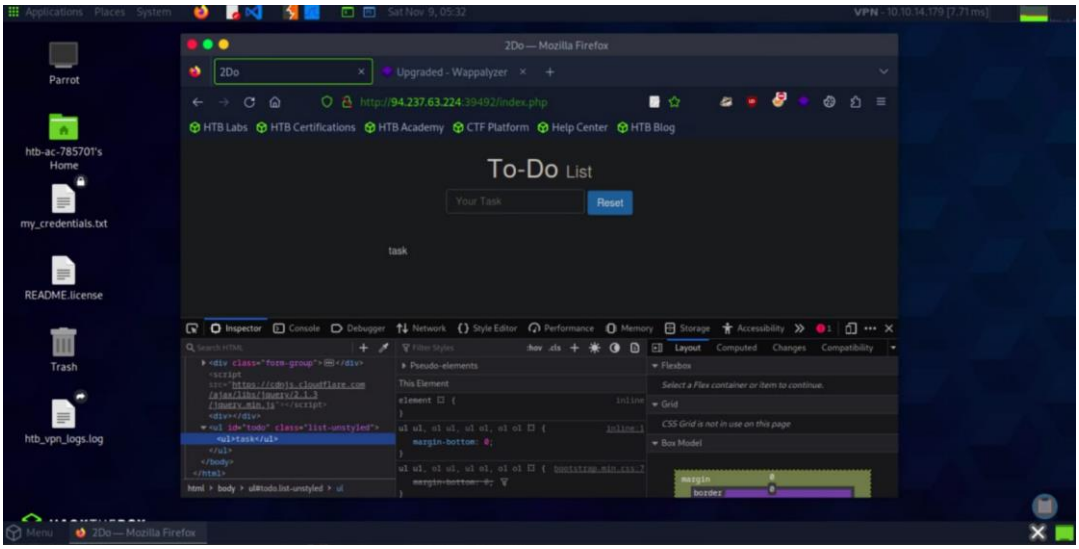

Рисунок 3.12 – DOM структура To Do списку

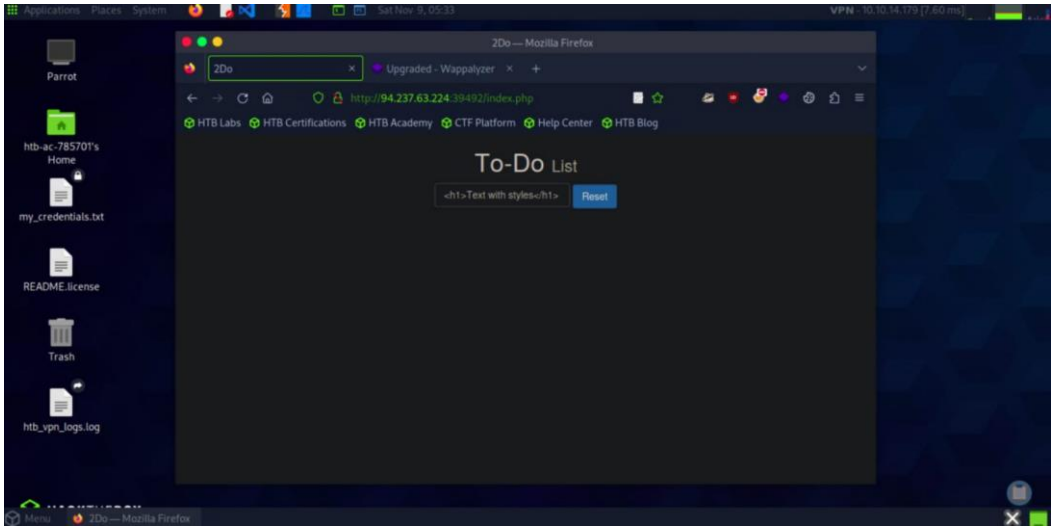

Рисунок 3.13 – Тестовий запит з використанням HTML коду

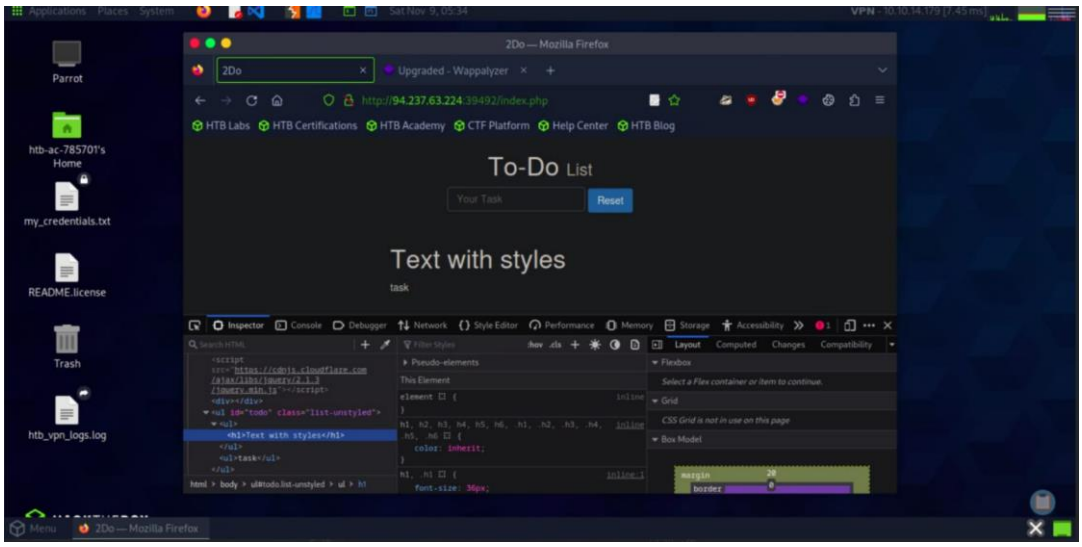

Рисунок 3.14 – DOM структура To Do списку

Таким чином можна побачити, що передані стилі застосувалися до тексту, й текст «Text with styles» візуально має більший розмір, до того ж у DOM Document Object Model), що містить у собі структуру усієї сторінки вказано, що тепер значення знаходиться не просто всередині тега <ul>, а ще й всередині тега <h1>. З цього можна зробити, що сайт є вразливим, але на даному етапі даний висновок буде помилковим, адже даний тег є нешкідливим й можливо на сайті реалізована валідація на більш шкідливі теги (рис. 3.15).

Як можна побачити з рисунку вище, то завдяки цьому запиту були відображені значення атрибуту cookie, що зазвичай зберігають особисту та важливу інформацію для користувача, таку як токен. Також було отримано флаг, що вказує на успішну експлуатацію вразливості міжсайтового скриптингу. Дана вразливість відноситься до типу Stored XSS, оскільки дані зберігаються на сервері та надсилаються з кожним запитом до клієнта. Проте є спосіб ще більш точно запевнитись у наявності XSS вразливості.

Справа в тому, що деякі сайти мають в собі iframe елементи, тобто форми з якихось інших сайтів. Таким чином, якщо форма є вразлива, то це не значить, що й увесь сайт є вразливим, бо ця форма може лише візуально відображатися у якості iframe, проте ніякого відношення не мати з сервером.

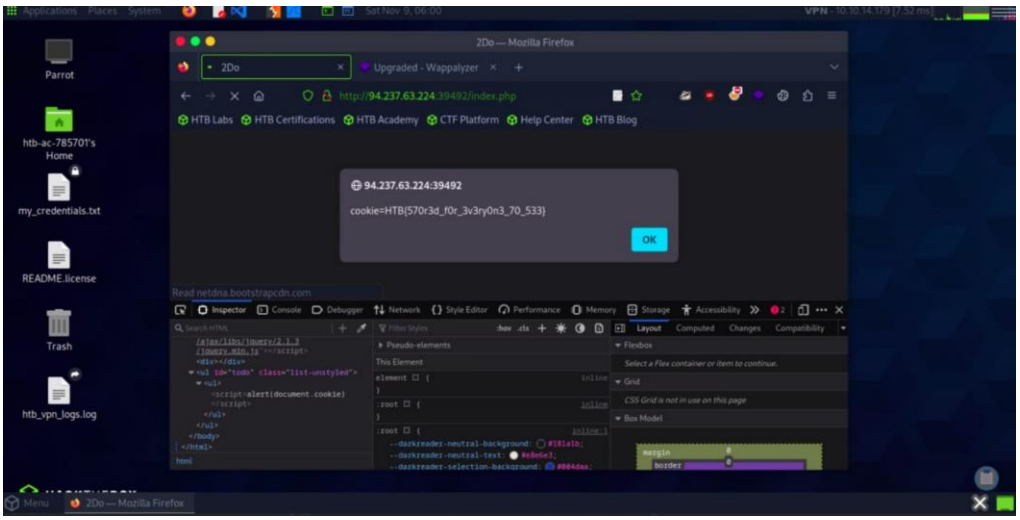

Рисунок 3.15 – Отримання даних сесії користувача (cookie)

Таким чином відображається посилання сайту, до якого відноситься форма й за допомогою цього можна точно ідентифікувати чи вразлива наша цільова система чи лише ось цей iframe компонент [37] (рис. 3.16).

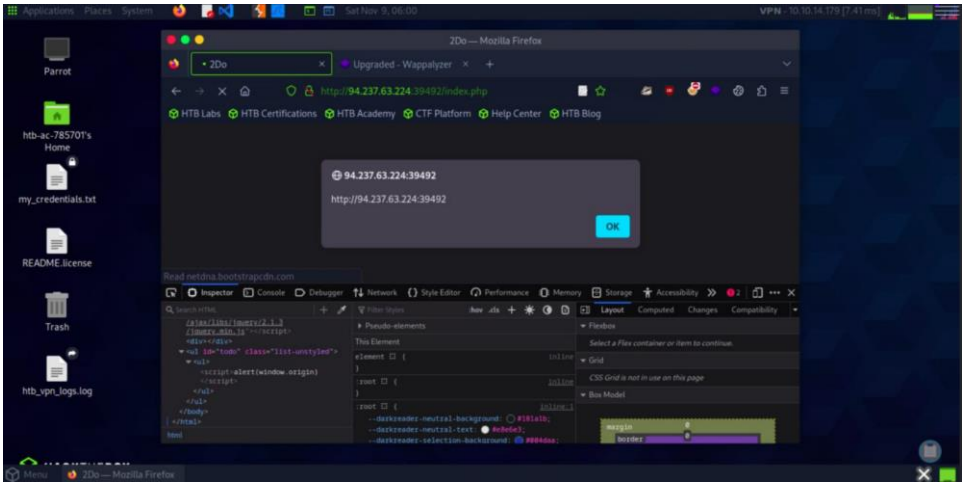

Рисунок 3.16 – Отримання посилання сторінки, з якою пов’язана форма

Наступні приклади є менш масштабними, адже вони можуть вразити лише обмежену кількість користувачів, тож reflected XSS: задля демонстрації використовується майже той самий сайт, лише тепер іноді він повертає помилки (рис. 3.17). Так як помилки відображаються в залежності від користувацького вводу та при оновленні сторінки зникають, то загроза від даної вразливості не така велика.

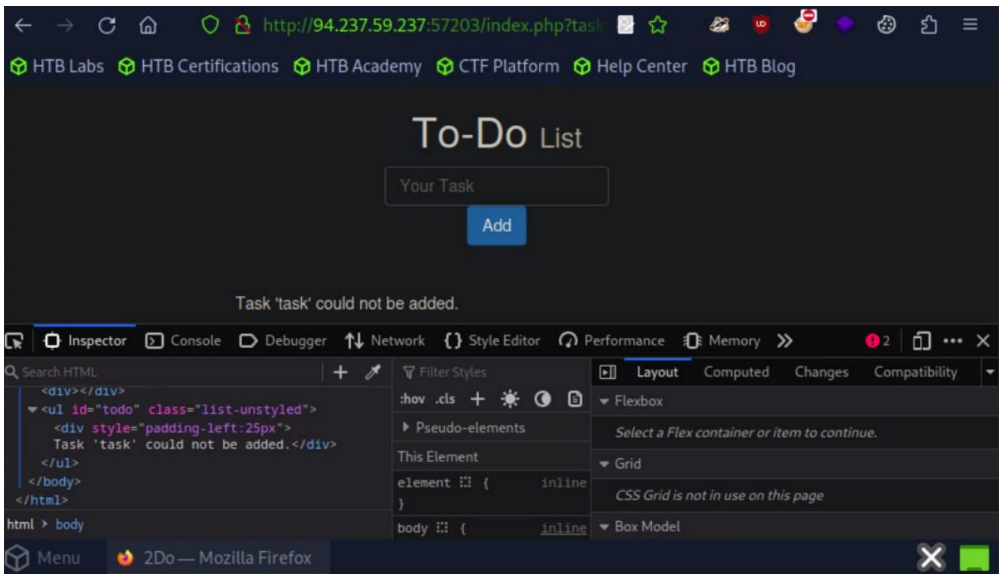

Рисунок 3.17 – DOM структура

Як можна помітити з рисунку вище, то користувач при вводі «task», отримав повідомлення з помилкою о неможливості додання такого елементу у список. Використовуючи той самий запит на отримання cookies користувача отримуємо той самий результат та флаг.

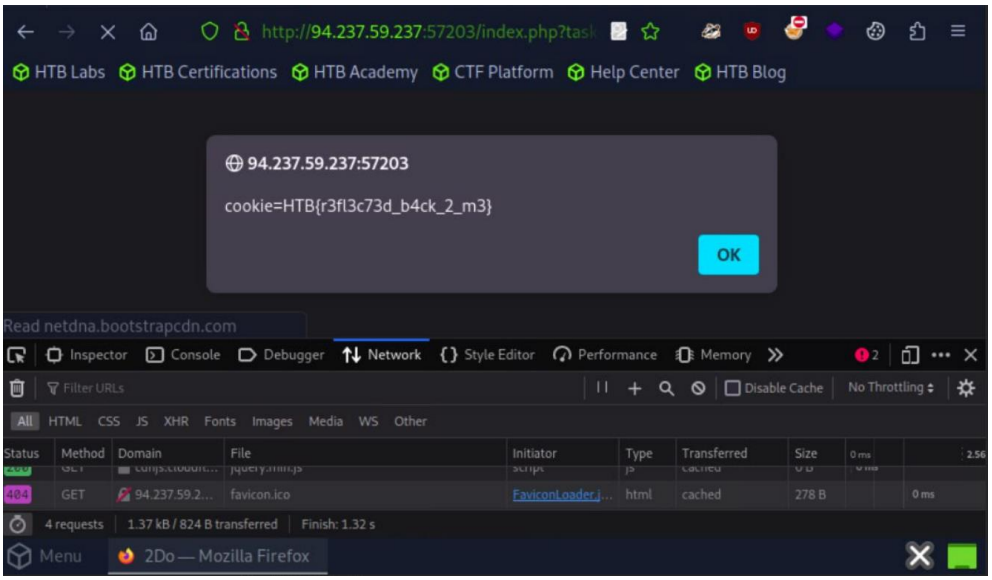

Рисунок 3.18 – Отримання даних сесії користувача (cookies)

Й при подальшій інспекції DOM структури сторінки можна помітити, що ввод користувача міститься у коді сторінки (рис. 3.19).

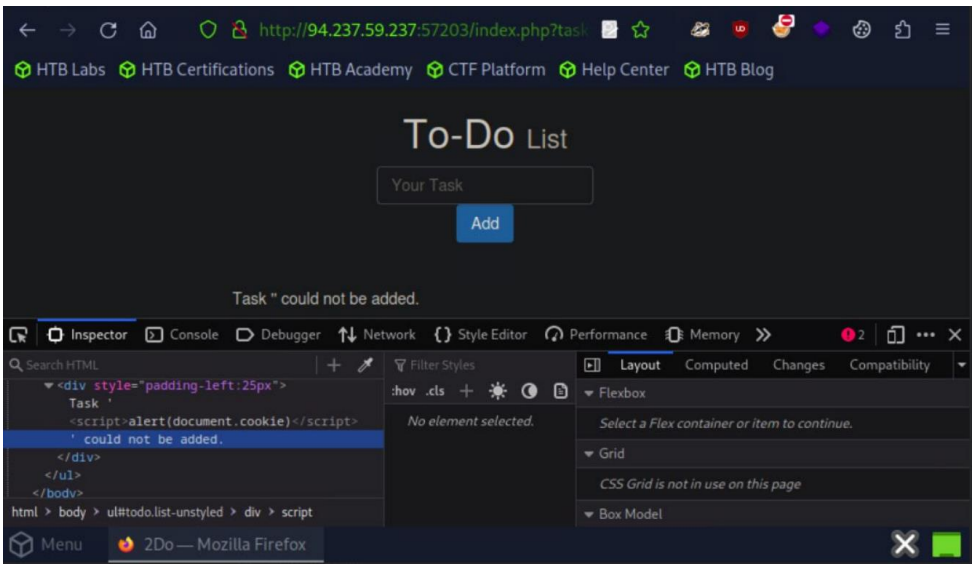

Рисунок 3.19 – DOM структура

Проте з описаного вище не зрозуміло як це може бути використано з ціллю експлуатації та й що це за така вразливість, при якій користувач вражає сам себе. В залежності від HTTP методу можуть бути різні варіації, в даному разі запит, який передає ввод користувача це GET, тобто усі параметри передаються у посиланні (рис. 3.20). Таким чином можна скопіювати це посилання та надіслати якомусь користувачу. Якщо прописати більш вишуканий код, то можна зробити таким чином, що дані сесії користувача надішлються на якийсь підконтрольний злочинцю сервер.

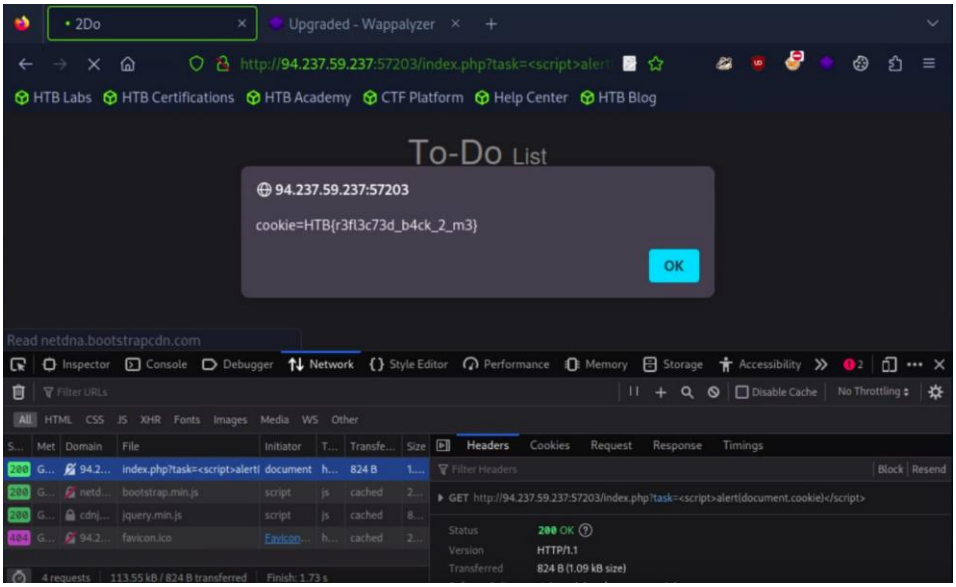

Рисунок 3.20 – GET запит у Network (DevTools)

3.3 Експлуатація SQL ін’єкції

Щоб продемонструвати дану вразливість, буде використано тестовий сайт, а саме форму авторизації для панелі адміністратора, яка складається з двох полів: користувацьке ім’я та пароль. Тобто звичайна форма авторизації, проте додатково для кращого розуміння над формою буде виводитись поточний стан SQL запиту після вводу користувача (рис. 3.21).

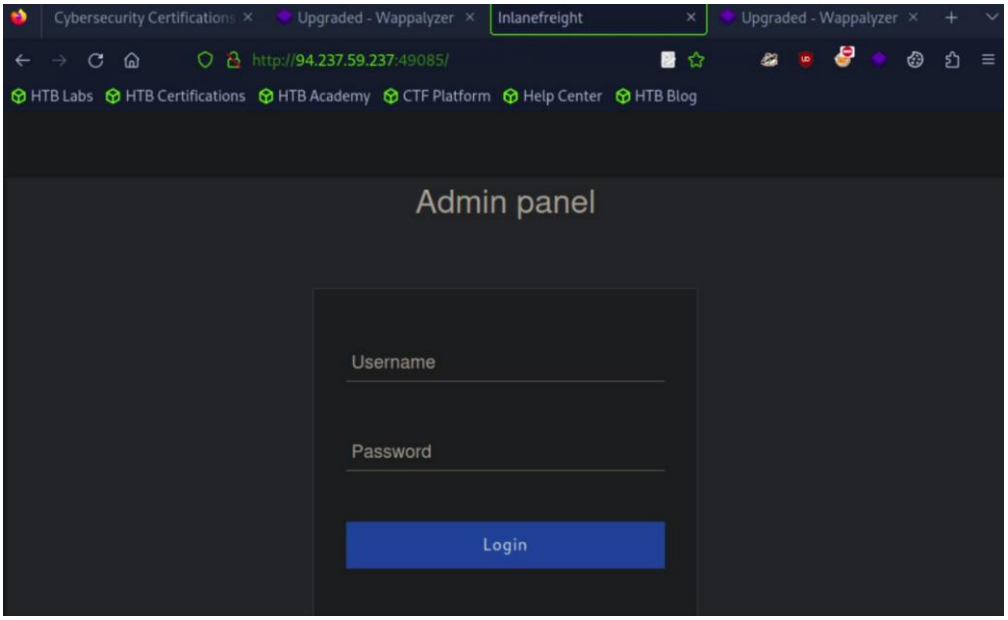

Рисунок 3.21 – Форма авторизації панелі адміністратора

Ще однієї вразливістю є погане налаштування та конфігурація середовищ, доступів та пари логіна та пароля, що не змінюється з заводських налаштувань. Тому варто спробувати вказати значення «admin» в обох полях (рис. 3.22).

Спроба авторизації виявилася невдалою, таким чином буде складно підбирати пару логіну та паролю, адже можливих варіантів може існувати дуже велика кількість, навіть для можливостей сучасних комп’ютерів. В такому разі завдяки SQL ін’єкції можна модифікувати запит й «обдурити» систему (рис. 3.23).

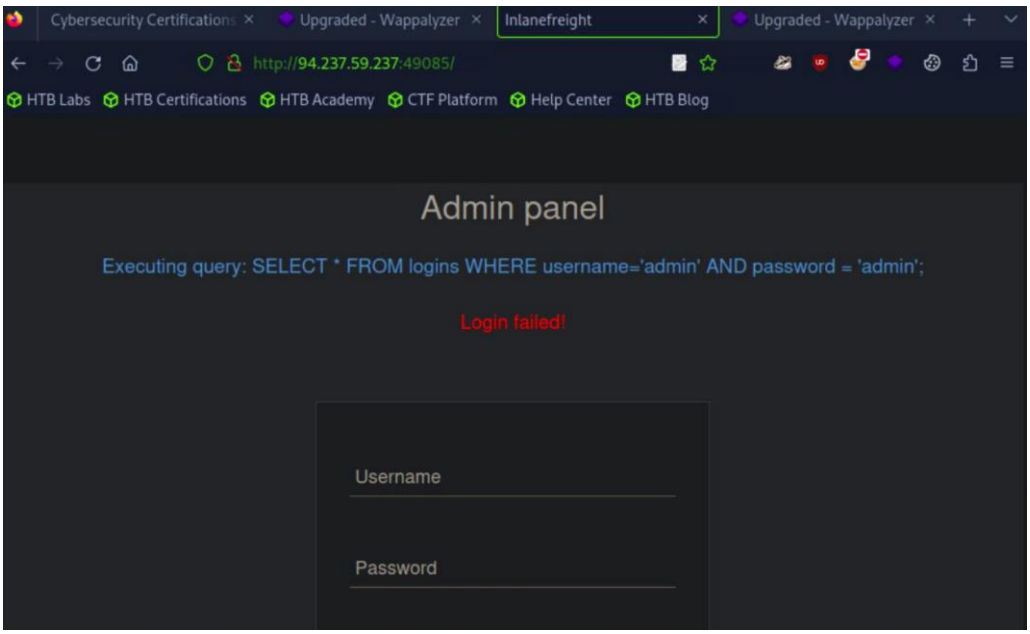

Рисунок 3.22 – Невдала спроба авторизації

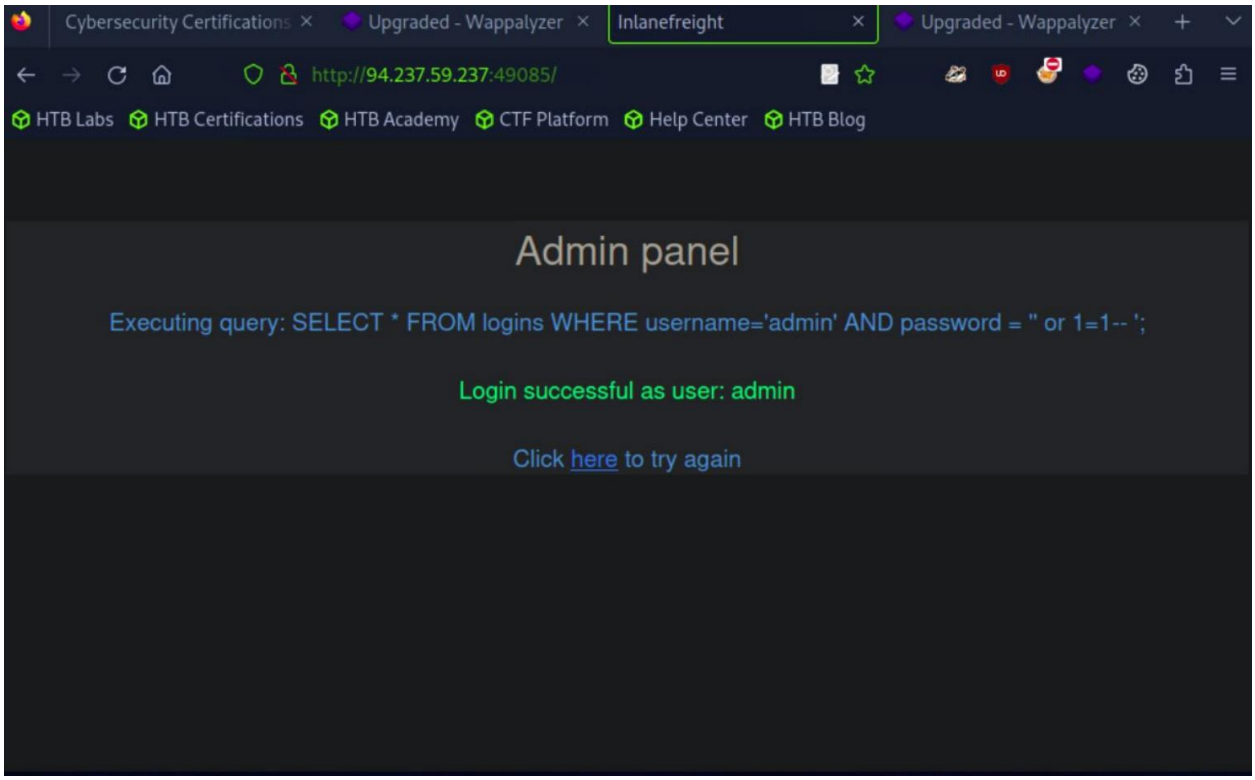

Рисунок 3.23 – Логін адміністратором завдяки SQL ін’єкції

Таким чином з рисунку вище можна побачити фінальний SQL запит, який було сформовано після користувацького вводу. У полі логіна було використано значення «admin», а у полі пароля «‘ OR 1=1-- ». Уся ін’єкція знаходиться у полі паролю:

– символ ‘ – закриває строку, таким чином утворюється пусте значення (password = ‘‘);

– частина ‘ OR 1=1’ – додає додаткову умову, яка завжди буде повертати true, адже одиниця дорівнює одиниці;

– частина ‘-- ‘ – додає коментар, який ігнорує усі SQL інструкції, які йдуть далі, пробіл після потрібен для того, що в деяких СУБД потрібно залишати пробіл для правильного синтаксису коментаря.

Таким чином першим виконується логічне «Та» (AND), що повертає false, адже нема такого запису у базі даних, де логін дорівнює «admin», а пароль – пусте значення. А потім виконується логічне «Або» (OR), що поверне true адже для цього достатньо лише того, щоб одна з умов повертала true. Тому достатньо лише вказати існуючий логін для того, щоб увійти в систему.

Але, варто зауважити, що використання «OR 1=1» або схожі конструкції не рекомендується та вживати їх потрібно із обережністю, адже якщо ввод користувача буде підставлятись у запит, відмінний від SELECT (наприклад DELETE або UPDATE), то дані будуть модифіковані, що може негативно сказитись на реальних та важливих даних [38].

В даному випадку можна використовувати інші SQL ін’єкції. В такому разі значення пароля буде проігноровано через коментар, що буде передаватись у полі логіну, й буде достатньо лише вказати існуючий логін для того, щоб авторизуватись у системі (рис. 3.24). Тобто по суті результат буде такий самий, як й з минулим прикладом, але дана реалізація буде більш вірною та безпечною.

Інший тип SQL ін’єкції який було розглянуто – Union ін’єкція. Даний тип ін’єкції гарно підходить у випадку, коли результат виконання запиту повертається до користувача й відображається на сторінці. Це можуть бути якісь повідомлення з помилкою або наприклад будь-який результат пошуку.

Для демонстрації було використано сайт, на якому міститься перелік портів (морських), а також мається пошукове поле, де за вказаним запитом користувача повернуться відповідні порти або помилка (рис. 3.25, 3.26).

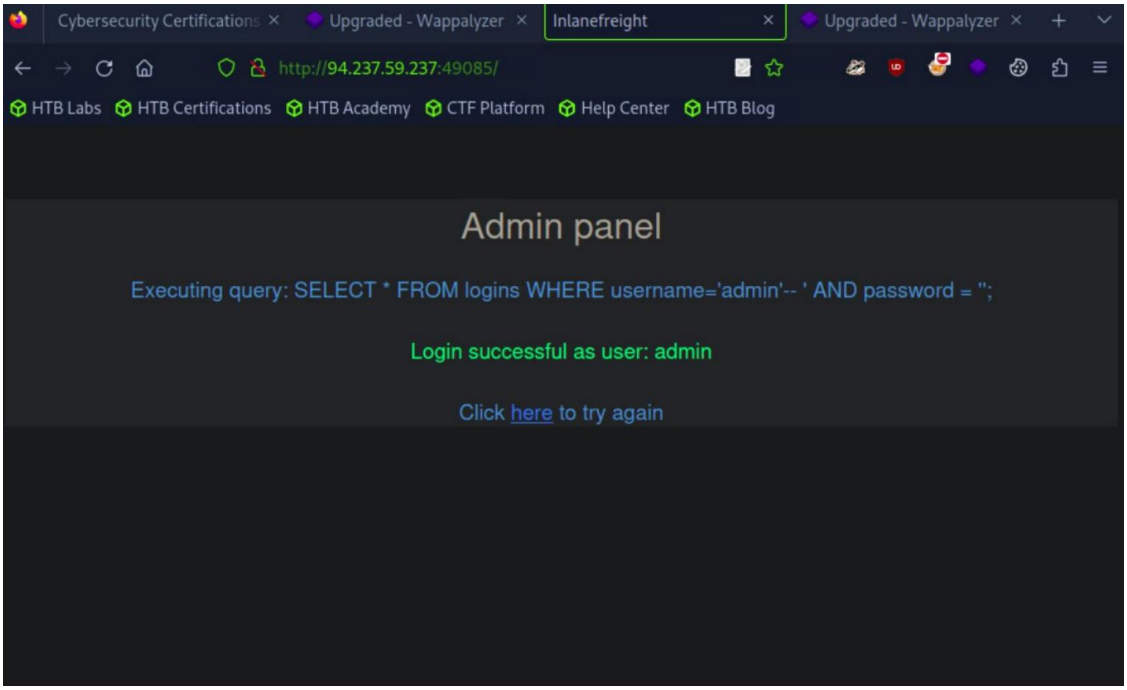

Рисунок 3.24 – Логін адміністратором завдяки SQL ін’єкції

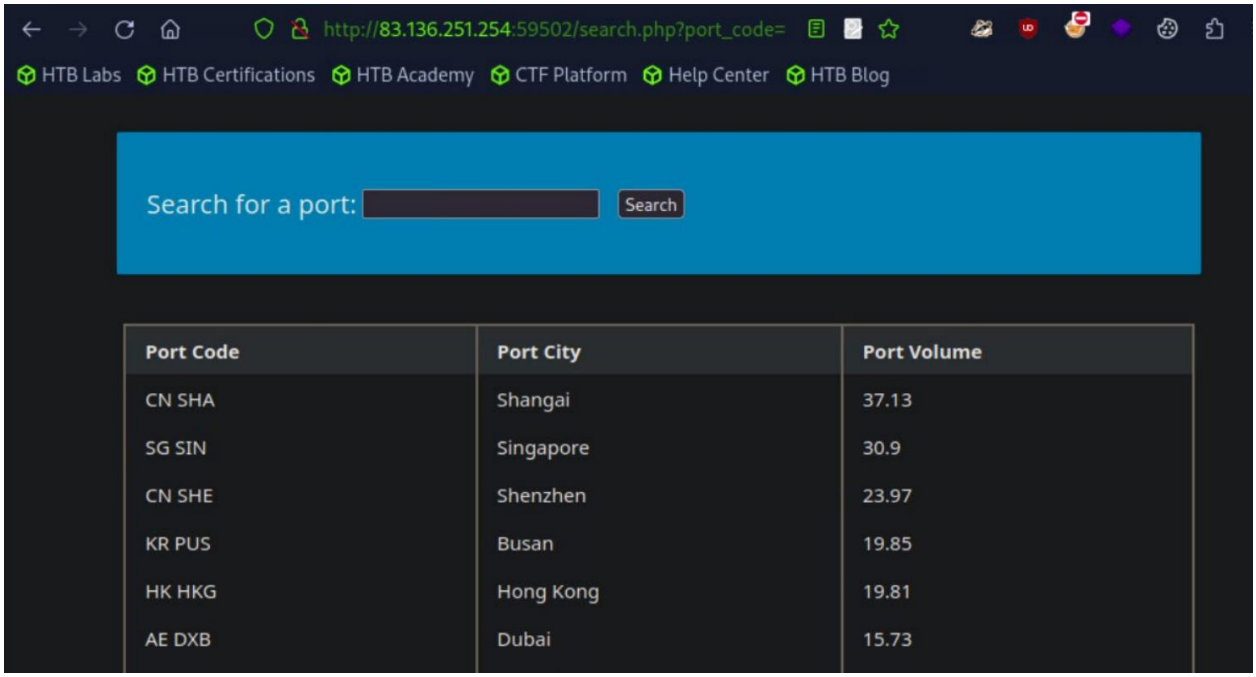

Рисунок 3.25 – Тестовий сайт

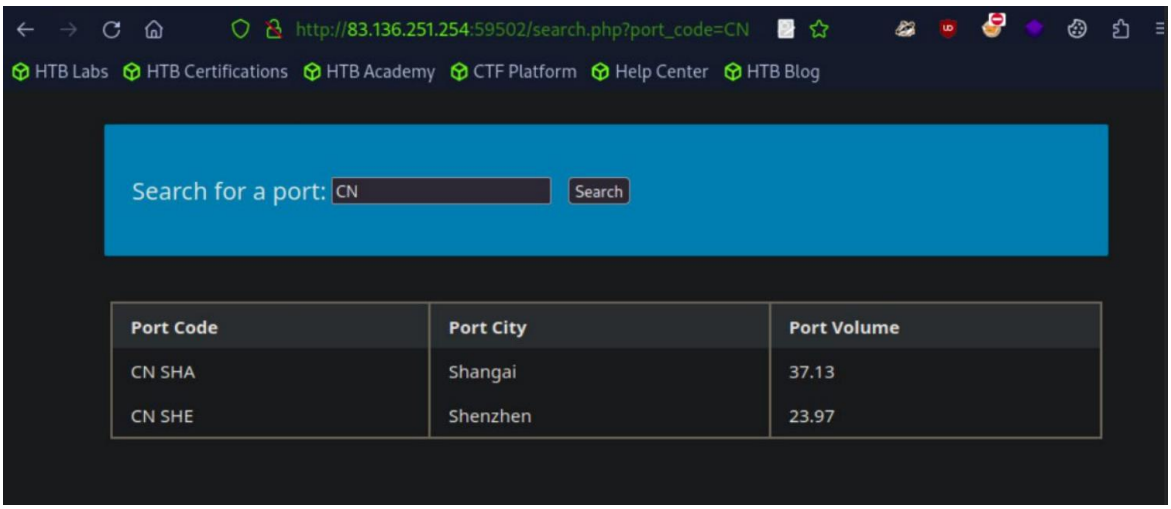

Рисунок 3.26 – Очікувана робота тестового сайту

За допомогою вже існуючих знань про коментарі та їх користь у ін’єкціях спробуємо модифікувати початковий запит, щоб завдяки оператору UNION отримати додаткову інформацію. Але спершу потрібно дізнатись кількість колонок, які приймають участь у оригінальному SELECT операторі, адже задля використання оператору UNION обов’язковою умовою є однакова кількість в усіх запитах, що приймають участь. Так як на сторінці можна побачити таблицю, що складається з трьох полів, то варто спробувати для початку зробити UNION для трьох полів (рис. 3.27).

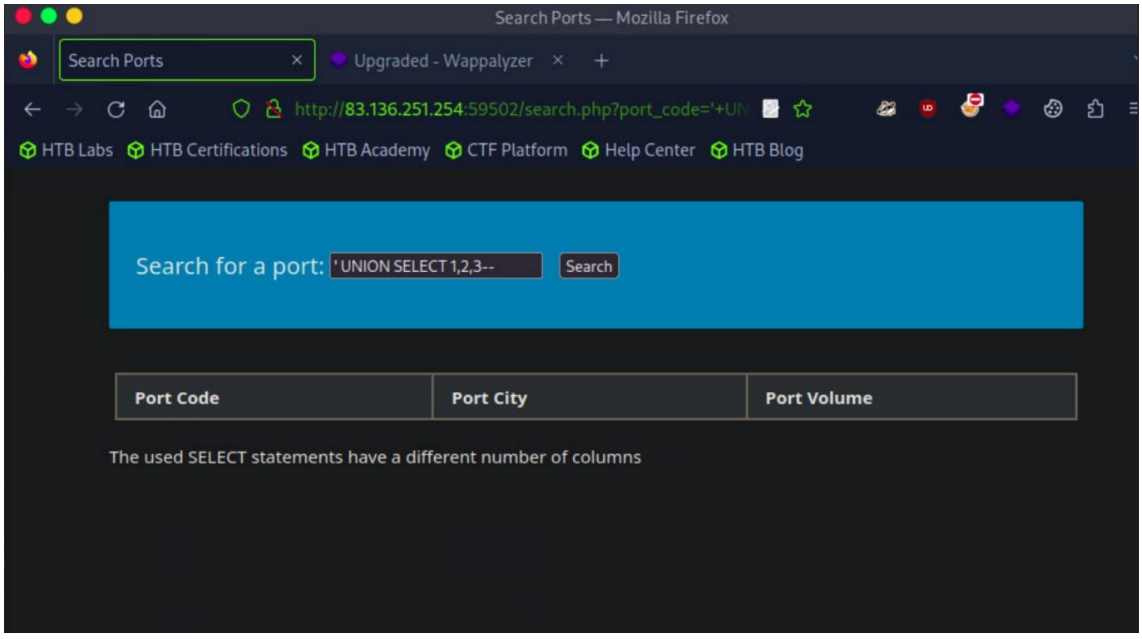

Рисунок 3.27 – Невдала UNION ін’єкція

Користувачу відображається повідомлення з помилкою. Схоже, що сам оригінальний запит використовує іншу кількість колонок, скоріше за все більшу, а вже на стороні фронтенду користувача відображаються лише деякі з цих колонок. Наступним кроком потрібно перевірити UNION з 4 значеннями (рис. 3.28).

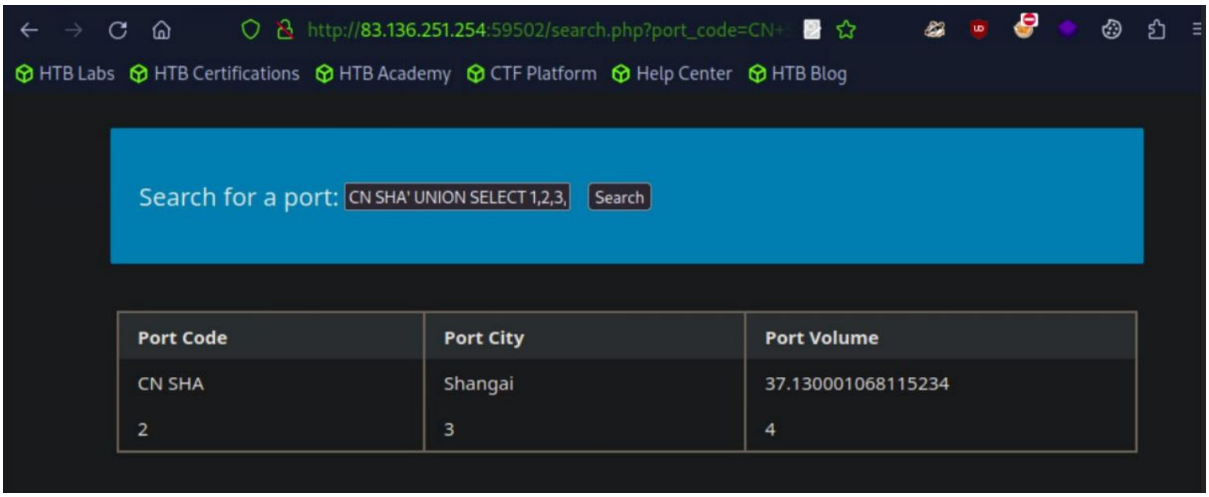

Рисунок 3.28 – Вдала UNION ін’єкція

Як можна побачити з рисунку вище, то користувачу окрім порту «CN SHA» повернулась строчка з вказаними під час ін’єкції даними. Також можна помітити, що перше значення не відображається, що підтверджує нашу догадку про те, що на сторінці відображаються не усі поля, які використовуються при запиті.

Задля того, щоб перетворити це на вразливість потрібно спробувати отримати якусь інформацію, наприклад як версію бази даних, що може допомогти у пошуку вразливостей, адже існують такі сайти, на яких зібрані усі CVE (Common Vulnerabilities and Exposures) – відомі вразливості. На даних сайтах можна знайти усю потрібну інформацію від вказаного рівня критичності знайденої вразливості до інструкції з експлуатації тієї чи іншої вразливості (рис. 3.29).

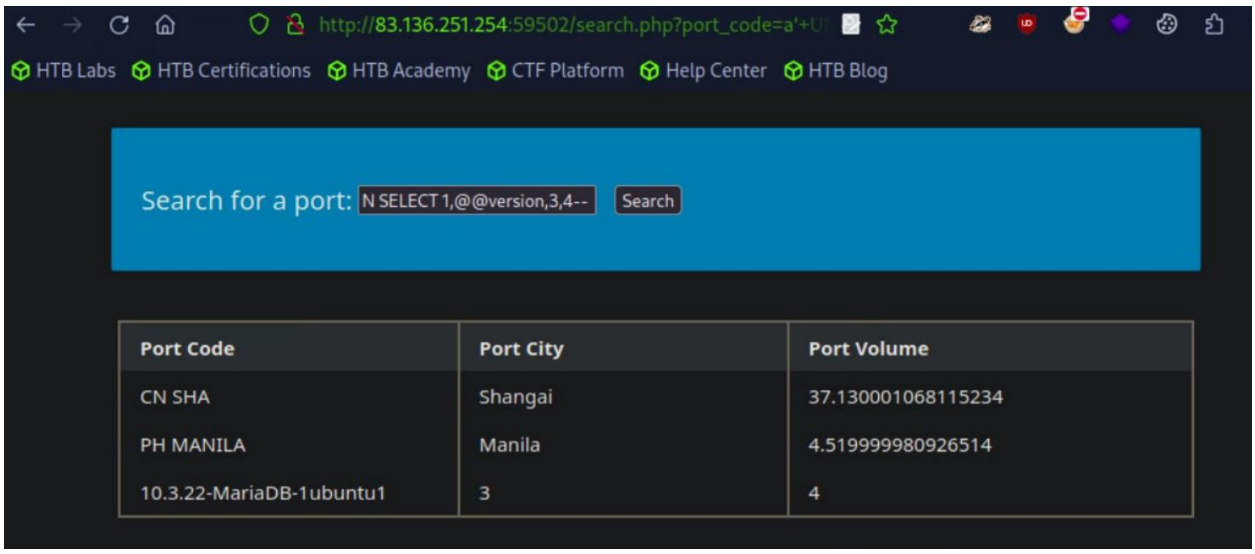

Рисунок 3.29 – Отримання версії бази даних

3.4 Експлуатація знаходження скритого контенту

Для демонстрації цієї вразливості було обрано такий інструмент як ffuf. Існує достатньо аналогів таких як wfuzz або gobuster та dirb проте на мою думку ffuf є найбільш user-friendly й містить усі необхідні методи для виконання задач з пошуку прихованого контенту. Для повноцінної роботи ffuf приймає наступні параметри [39]:

– -w – словник, в якому міститься перелік значень, які будуть одне за одним підставлятися та надсилатися з запитом;

– -u – URL, що веде до сайту, на якому потрібно знайти прихований контент.

Варто зауважити, що наприкінці словника потрібно прописувати «:FUZZ», й також вказувати це саме «FUZZ» у тому місці, де потрібно замість цього значення підставляти дані з обраного словника. Необов’язково давати назву цьому параметру як «FUZZ», проте це вважається як правило гарного тону (рис. 3.30).

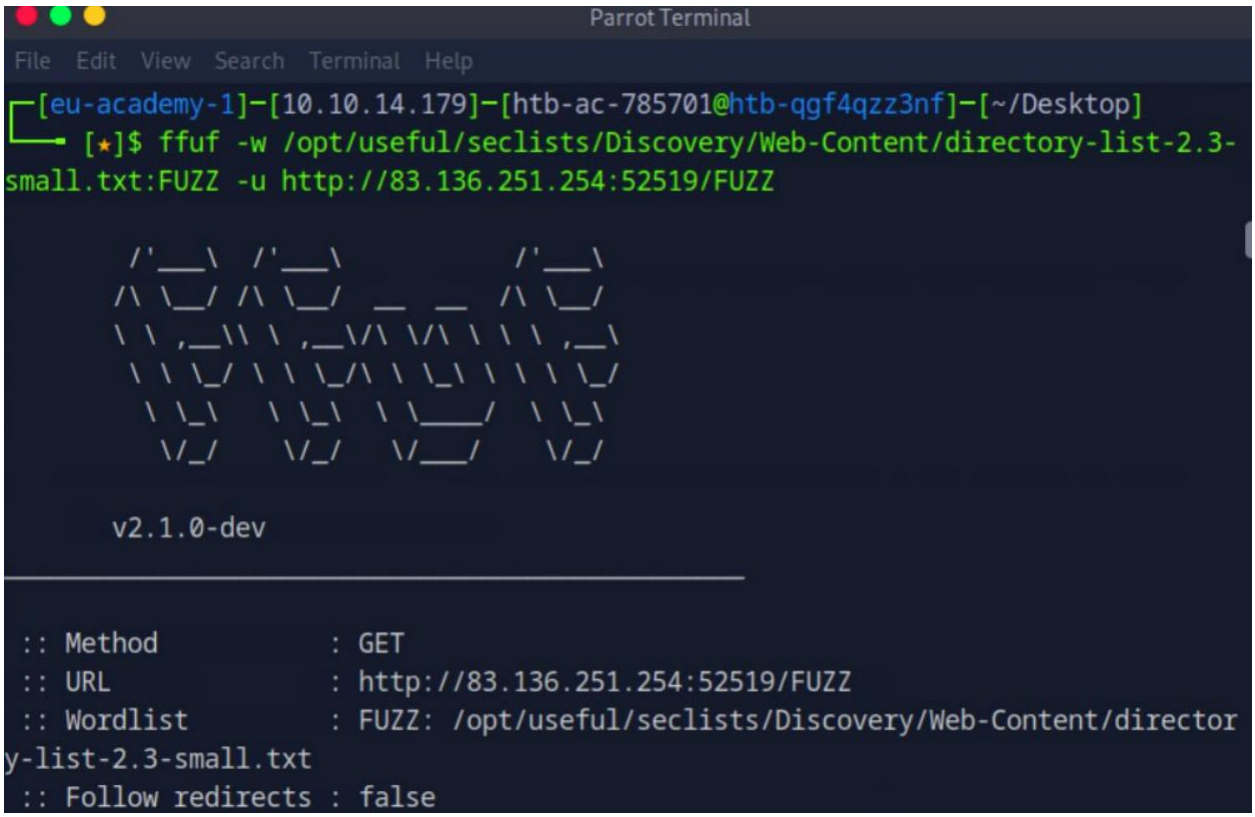

Рисунок 3.30 – Запуск команди ffuf (пошук директорій)

У консолі можна побачити багато «шуму», тобто інформації, яка не є потрібною, адже фахівцю неважливо бачити яких директорій не існує на сайті, тому помилки лише заважають. Кожні декілька запитів у консолі буде виводитись статус по запиту та його результату, потрібно звертати увагу на запити без помилки, як в даному випадку це запит з директорією «forum». Також можна помітити, що успішні запити від помилок відрізняються статус-кодом, розміром відповіді та кількістю слів та рядків, що надходять у відповіді (рис. 3.31).

Задля того щоб спробувати покращити вивід у консолі, то можна спробувати скористуватися фільтрами, які надає ffuf, існують два види фільтрів: ті, що залишають результати за вказаними фільтрам (починаються з «m») та ті, що прибирають результати за вказаними фільтрами (починаються з «f»). До таких фільтрів відносяться [40]:

– mc або fc (фільтр по статус коду);
– ml або fl (фільтр по кількості рядків);

– ms або fs (фільтр по розміру відповіді);
– mw або fw (фільтр по кількості слів).

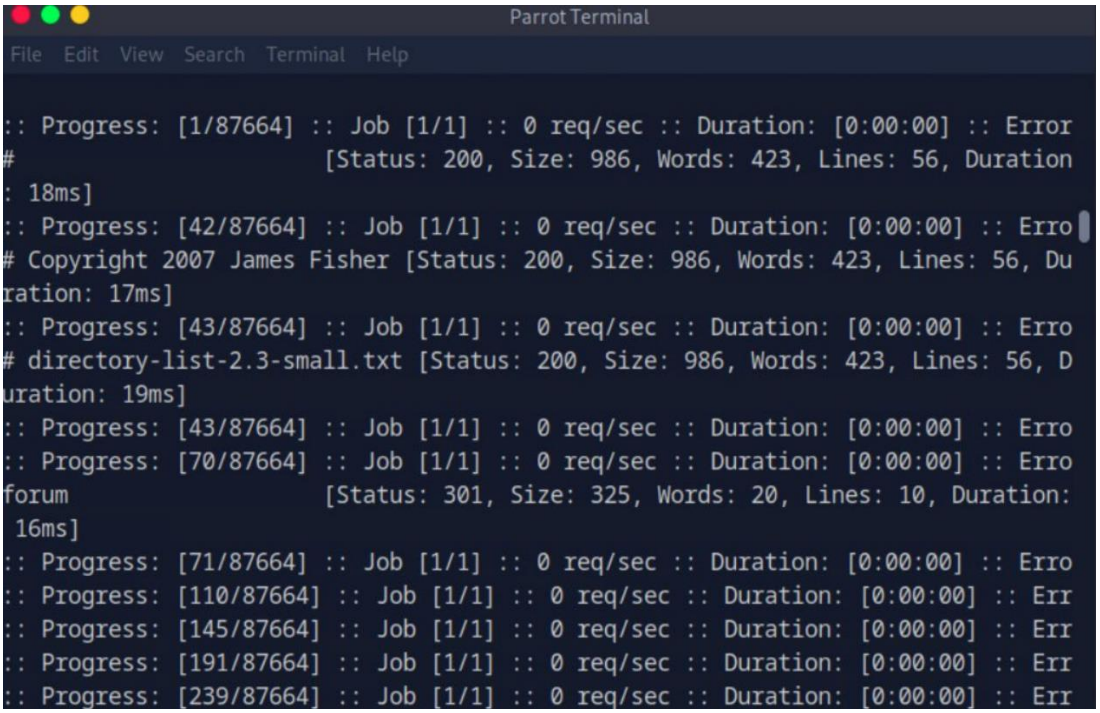

Рисунок 3.31 – Відпрацювання команди ffuf (пошук директорій)

Враховуючи те, що при помилці розмір відповіді, як й кількість слів та рядків буде однаковим, то можна спробувати відфільтрувати відповіді у консолі за будь-яким з зазначених критеріїв. Наприклад спробуємо відфільтрувати за розміром відповіді, це значення можна подивитись у консолі серед результатів виконання попередньої команди у будь-якого невдалого запиту, який повернув помилку (рис. 3.32).

При поточній команді з фільтром вивід результатів у консолі є більш «чистим» (рис. 3.33). Поточний результат виконання виводиться не так часто, також вивід помилок не займає так багато місця, а ще багато словників починаються з закоментованої інформації про словник та його творця, відповідно береться значення з кожного рядка, навіть з цих закоментованих й надсилається. При даному запиті з фільтром дані результати не повертають помилки, які займають багато місця.

Рисунок 3.32 – Запуск команди ffuf із застосуванням фільтру (пошук директорій)

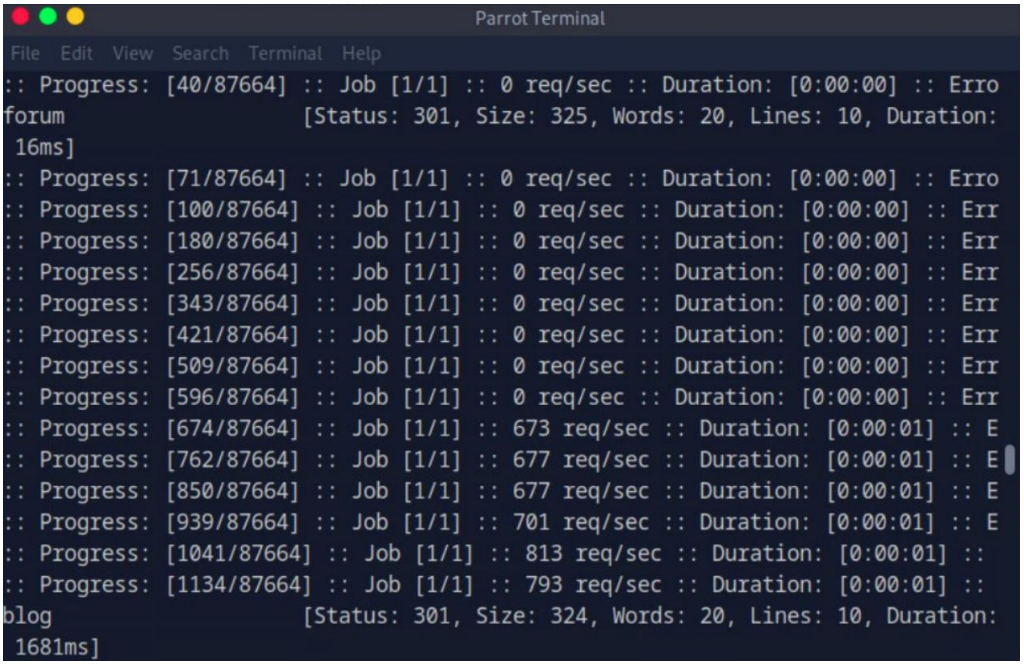

Рисунок 3.33 - Відпрацювання команди ffuf з застосуванням фільтру (пошук директорій)

За допомогою ffuf та схожих інструментів можна знаходити не лише директорії, а ще й:

– файли;

– параметри, що передаються з запитами;

– віртуальні хости;

– домени та субдомени.

Для прикладу, щоб здійснити пошук по файлам потрібно додати ще один флаг до команди «-e», в якому можна перерахувати усі потрібні розширення файлів, на які потрібно звернути увагу (рис. 3.34, 3.35).

Рисунок 3.34 – Запуск команди ffuf (пошук файлів)

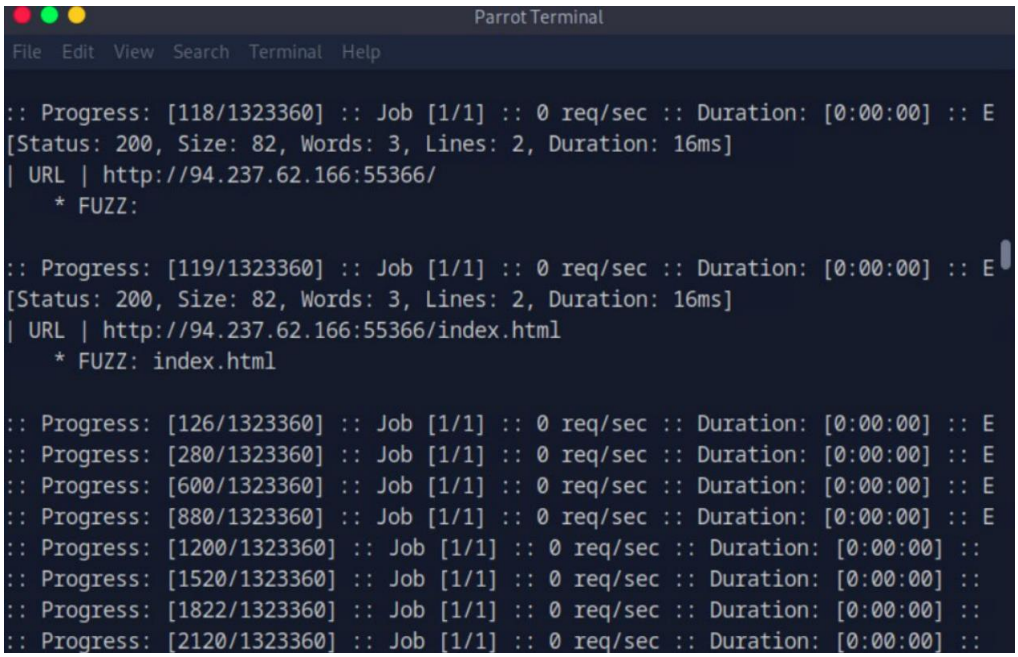

Рисунок 3.35 – Відпрацювання команди ffuf (пошук файлів)

Також у команді було застосовано додатковий флаг «-v», який відповідає за більш детальний вивід інформації у консолі.

3.5 Експлуатація reverse shell

Задля демонстрації даної вразливості було використано найлегший у розумінні випадок, коли в потенційного хакера вже є дані користувача, а саме:

- IP адреса;
- логін;
- пароль.

Це було зроблено для того, щоб саме сфокусуватись на експлуатації вразливості, бо процес отримання даних користувача – справа не з легких, для цього потрібно знати багато інших аспектів кібербезпеки, на кшталт file inclusion, що є поза межами цієї роботи.

Спочатку потрібно запустити процес «прослуховування» вхідного трафіку на зазначеному порті. Це можна зробити за допомогою декількох інструментів, в даній роботі було використано саме Netcat. Порт можна вказати будь-який, але краще користуватись здоровим глуздом та вказувати ті порти, які з великою ймовірністю не будуть заблокованими, адже адміністраторами системи можуть бути налаштовані спеціальні правила, за якими комунікація по певним портам буде неможлива.

Рисунок 3.36 – Запуск команди Netcat

Саме тому було використано порт 443, який зазвичай використовується для протоколу HTTPS, тобто основний порт для отримання веб-сторінок й не тільки.

Було використано протокол RDP (Remote Desktop Protocol) задля віддаленого підключення до девайсу. Варто зауважити, що даний протокол та шляхи до отримання доступу віддалено відрізняються в залежності від операційної системи «жертви».

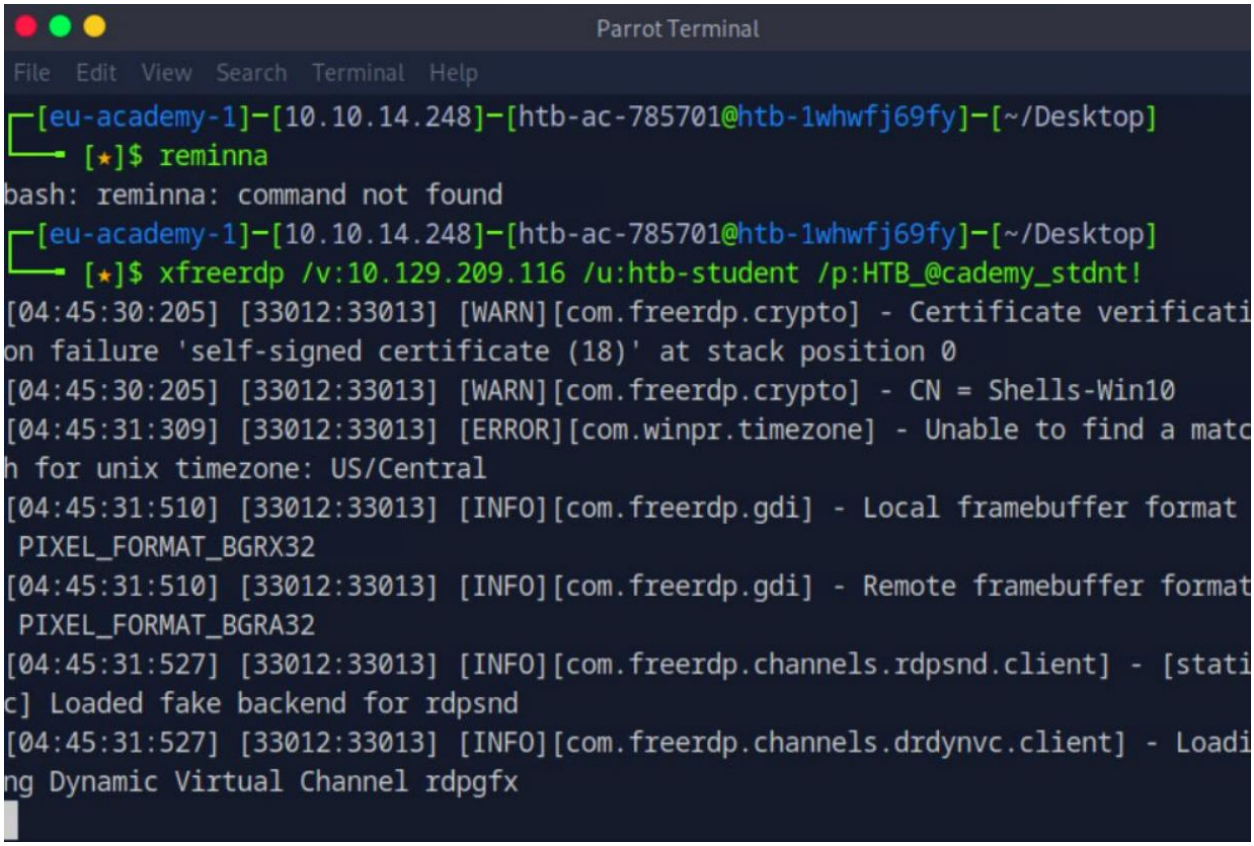

Рисунок 3.37 – Запуск команди xfreerdp

У даній команді використовуються IP віддаленного девайсу (/v), користувач, від якого потрібно отримати доступ до віддаленного девайсу (/u) та пароль обраного користувача (/p).

Після виконання даної команди, якщо усі дані були надані вірно, то відкриється екран віддаленного девайсу, тобто по суті користувач отримає доступ до свого комп’ютера з якогось іншого комп’ютера.

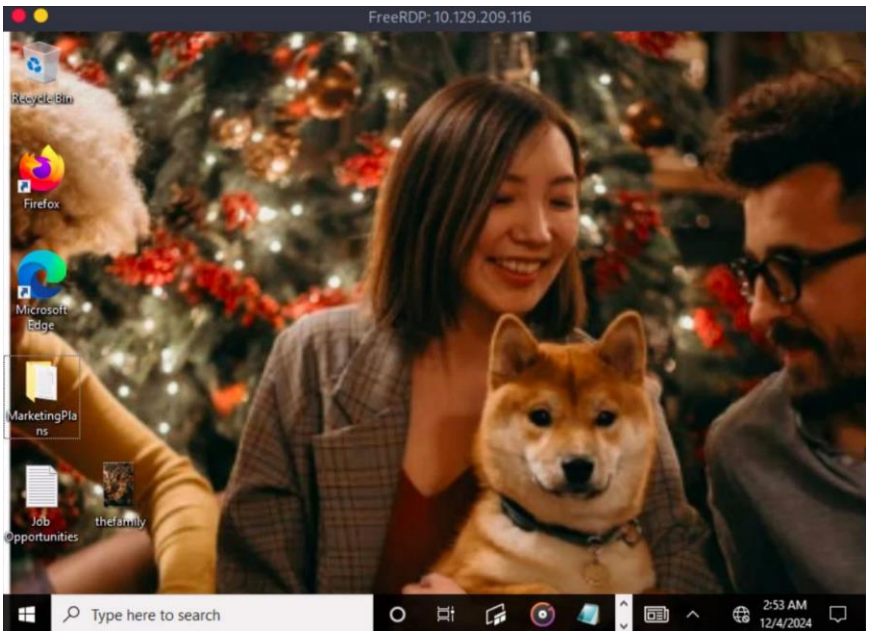

Рисунок 3.38 – Отримання доступу до віддаленого комп’ютеру

Далі, за допомогою reverse shell скрипта написаного на powershell (скриптів на різні випадки існує безліч, є багато спеціалізованих сайтів та репозиторіїв з прикладами rebershe shell написаних на powershell, bash, ruby, python, тощо) надсилаємо запит-встановлення комунікцаії з девайсом, який належить потенційному хакеру.

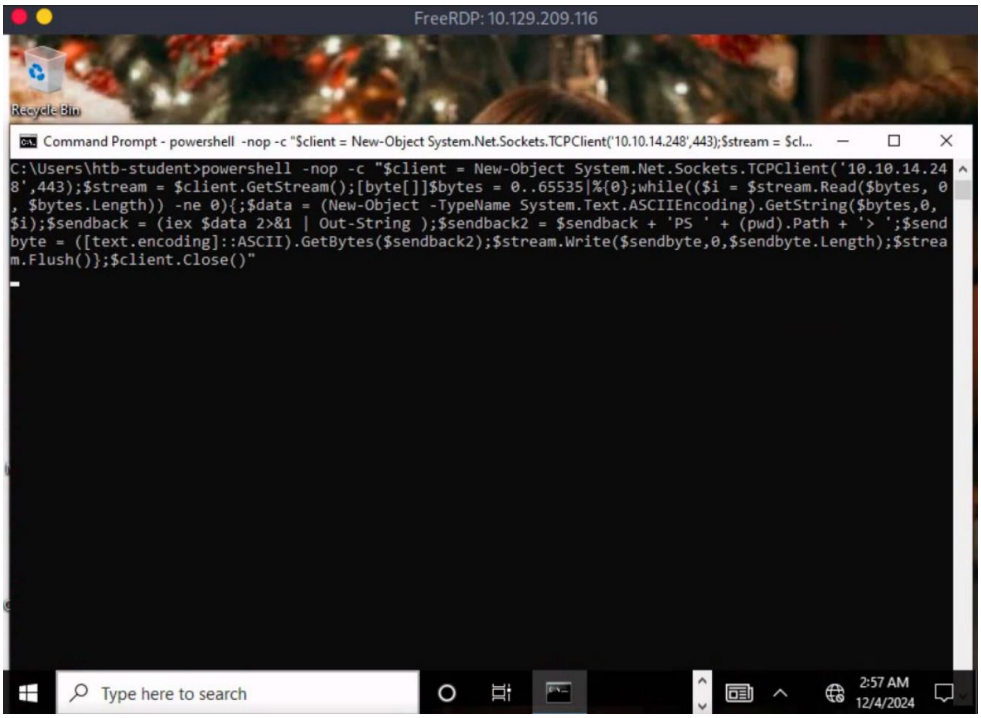

Рисунок 3.39 – Запуск команди reverse shell

Після цього у іншому терміналі, де Netcat «прослуховує» мережу на зазначеному порту (зверніть увагу, що у скрипті reverse shell потрібно вказати той самий порт, на якому «прослуховує» Netcat) відображається повідомлення про запит, що надійшов. Таким чином встановилась комунікація між двома комп’ютерами.

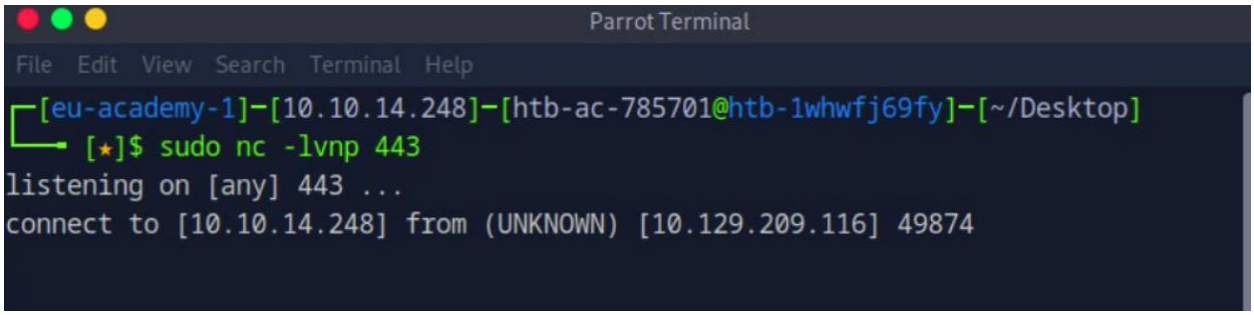

Рисунок 3.40 – Відпрацювання reverse shell

Встановлена комінкація означає те, що тепер потенційний хакер може отримати доступ до комп’ютера, з яким встановлена комунікація, а саме виконувати команди через термінал, які будуть виконані на іншому ком’ютері. Для приклада та порівняння було використано команду whoami, яка виводить у консоль поточного користувача.

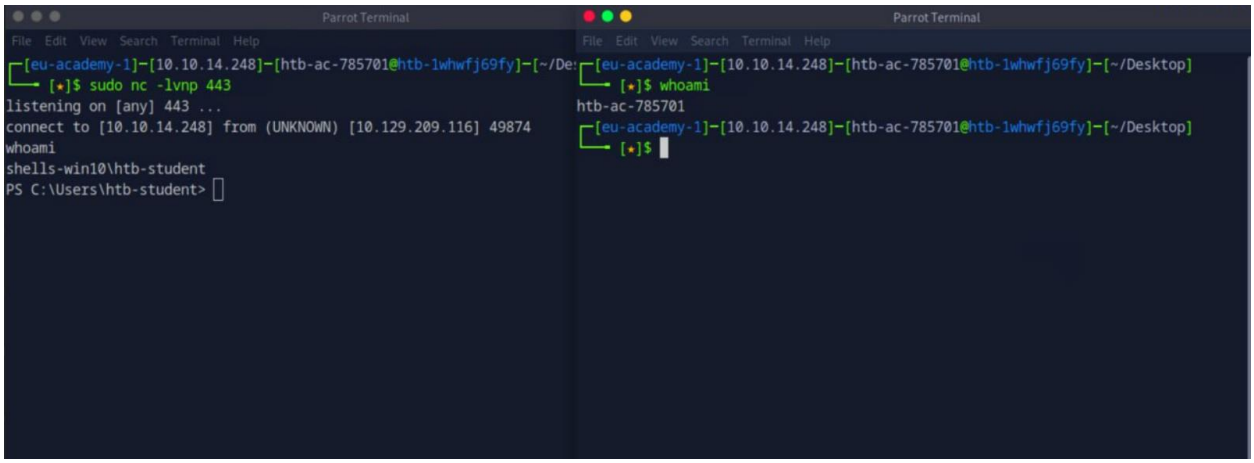

Рисунок 3.41 – Доказ вдалого відпрацювання reverse shell

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено дослідження методів та інструментів тестування вразливостей на прикладі віртуальних машин. Віртуальні машини були надані платформою HackTheBox, що спеціалізується на кібербезпеці. У ході кваліфікаційної роботи були досліджені різноманітні вразливості та їх види, серед яких:

– API Attacks (а саме Insecure Direct Object Reference та Broken Function Level Authorization);

– міжсайтовий скріптинг (а саме Stored та Reflected);

– SQL ін’єкція (а саме класична та UNION-based);

– знаходження скритого контенту за допомогою такого інструменту як ffuf.

Також була надана уся необхідна теоретична інформація, яка формує підґрунтя задля розуміння вразливостей та чому вони стаються й більше того, чому вони й на сьогодення є актуальними проблемами, які потрібно брати до уваги при розробці застосунків.

Усі експлуатації було супроводжено детальним та покроковим описом, а також задля кращої демонстрації були надані рисунки, які відповідали текстовому наповненню дослідження.

Результатом кваліфікаційної роботи є розроблений комплексний підхід до тестування безпеки веб-ресурсу, який містить найбільш поширені вразливості на сьогодення та покрокові інструкції для їх тестування.

Наукова новизна полягає у створенні «посібника» для спеціалістів-початківців з кібербезпеки, а також демонстрація широкого спектру розповсюджених вразливостей.

Результати дослідження апробовано у вигляді 1 тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [41].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Understanding RockYou.txt: A Tool for Security and a Weapon for Hackers. URL: https://www.keepersecurity.com/blog/2023/08/04/understanding-rockyou-txt-a-tool-for-security-and-a-weapon-for-hackers/ (дата звернення 10.10.2024).

2. Petya. URL: https://uk.wikipedia.org/wiki/Petya (дата звернення 10.10.2024).

3. Життєвий цикл розробки програмного забезпечення (Software Development Life Cycle – SDLC). URL: https://www.maxzosim.com/software-development-life-cycle-sdlc/ (дата звернення 11.10.2024).

4. Cyber Kill Chain. URL: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html (дата звернення 13.10.2024).

5. Kuzminska, O., Mazorchuk, M., Morze, N., & Kobylin, O. (2020). Digital learning environment of ukrainian universities: The main components to influence the competence of students and teachers. In Information and Communication Technologies in Education, Research, and Industrial Applications: 15th International Conference, ICTERI 2019, Kherson, Ukraine, June 12–15, 2019, Revised Selected Papers 15 (pp. 210-230). Springer International Publishing.

6. Anonymous (hacker group). URL: https://en.wikipedia.org/wiki/Anonymous_(hacker_group) (дата звернення 13.10.2024).

7. Ревак, І. О. OSINT В СИСТЕМІ ІНФОРМАЦІЙНИХ РЕСУРСІВ УПРАВЛІННЯ БЕЗПЕКОЮ. Аніловська Г., 156.

8. Trump Twitter ‘hack’: Police accept attacker’s claim. URL: https://www.bbc.com/news/technology-55337192 (дата звернення 14.10.2024).

9. What is Google Dorking? – A glossary for web security. URL: https://wolf-of-seo.de/en/what-is/google-dorking/ (дата звернення 16.10.2024).

10. Tvoroshenko, I. S., & Maksimenko, H. (2021). To the question of analysis of existing mechanisms of web application testing.

11. OWASP. URL: https://uk.wikipedia.org/wiki/OWASP (дата звернення 18.10.2024).

12. OWASP Top Ten | OWASP Foundation. URL: https://owasp.org/www-project-top-ten/ (дата звернення 18.10.2024).

13. Супрун, А. Є. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ «СОЦІАЛЬНА МЕРЕЖА ДЛЯ ПУБЛІКАЦІЇ ТВОРЧИХ РОБІТ». 28-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті». Зб. матеріалів форуму. Т. 7. Харків: ХНУРЕ. 2024. 320 с., 114.

14. Curl – Tutorial. URL: https://curl.se/docs/tutorial.html (дата звернення 19.10.2024).

15. Tvoroshenko, I., & Maksimenko, H. (2021). Research of regression and modular testing of web applications.

16. Iryna, T., & Maksym, K. (2021). Research results of functional, white box and smoke testing methods for mobile applications. Trends in science and practice of today, 5, 418.

17. Server-side request forgery (SSRF). URL: https://portswigger.net/web-security/ssrf (дата звернення 23.10.2024).

18. Tvoroshenko, I. S., & Kuznetsov, M. (2021). About the role of testing in process of mobile application development.

19. Що таке API: простими словами про складне. URL: https://hostiq.ua/blog/ukr/what-is-api/ (дата звернення 24.10.2024).

20. Руденко, Д. О., & Бондар, В. О. (2020, November). ОГЛЯД МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ СТРАТЕГІЙ ОБ'ЄКТНО-ОРІЄНТОВАНОГО МАППІНГУ ДЛЯ ЗІСТАВЛЕННЯ СУТНОСТЕЙ ПРИ РОЗРОБЦІ WEB ДОДАТКІВ. In The 3 rd International scientific and practical conference—Priority directions of science and technology development‖(November 22-24, 2020) SPC—Sci-conf. com. ua‖, Kyiv, Ukraine. 2020. 1488 p. (p. 377).

21. GraphQL vs. REST. URL: https://blog.postman.com/graphql-vs-rest/ (дата звернення 25.10.2024).

22. Why is HTTP stateless?. URL: https://dev.to/codexam/why-is-http-stateless-2m3p (дата звернення 27.10.2024).

23. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ.

24. Ahmad, M. A., Tvoroshenko, I., Baker, J. H., & Lyashenko, V. (2019). Modeling the structure of intellectual means of decision-making using a system-oriented nfo approach.

25. Cross Site Scripting (XSS) | OWASP Foundation. URL: https://owasp.org/www-community/attacks/xss/ (дата звернення 28.10.2024).

26. Titova, O., & Vysotskyi, D. (2021). The development of a subsystem for palliative patients information support.

27. SQL Injection. URL: https://www.w3schools.com/sql/sql_injection.asp (дата звернення 30.10.2024).

28. Automated content discovery with Burp Suite. URL: https://portswigger.net/burp/documentation/desktop/testing-workflow/mapping/hidden-content/automated-discovery (дата звернення 31.10.2024).

29. Using SecLists for Penetration Testing. URL: https://www.varutra.com/using-seclists-for-penetration-testing/ (дата звернення 01.11.2024).

30. Nmap. URL: https://uk.wikipedia.org/wiki/Nmap (дата звернення 02.11.2024).

31. Gorokhovatskyi, V. A., Rusakova, N., & Tvoroshenko, I. S. (2020). The application of image analysis methods and predicate logic in applied problems of magnetic monitoring. Telecommunications and Radio Engineering, 79(20).

32. What is a Reverse Shell?. URL: https://sysdig.com/learn-cloud-native/what-is-a-reverse-shell/ (дата звернення 02.11.2024).

33. HackTheBox. URL: https://app.hackthebox.com/home (дата звернення 03.11.2024).

34. What Is A Virtual Machine? VM Uses and Benefits. URL: https://cloud.google.com/learn/what-is-a-virtual-machine (дата звернення 03.11.2024).

35. Insecure direct object references (IDOR) – Access control. URL: https://portswigger.net/web-security/access-control/idor (дата звернення 04.11.2024).

36. What is Swagger. URL: https://swagger.io/docs/specification/v2_0/what-is-swagger/ (дата звернення 04.11.2024).

37. What is an iframe, and how do you use it on your website?. URL: https://www.siteground.com/kb/what-is-iframe/ (дата звернення 05.11.2024).

38. Avoid “OR 1=1” in SQL Injections. URL: https://www.youtube.com/watch?v=8iSGWP7lk-M (дата звернення 07.11.2024).

39. Ffuf man. URL: https://linuxcommandlibrary.com/man/ffuf (дата звернення 07.11.2024).

40. Ситніков, Д. Е., Ситнікова, П. Е., Тітов, С. В., & Тітова, О. В. (2021). Фільтрація результуючого набору асоціативних правил з точки зору оцінки цікавості. Системи обробки інформації, (1 (164)), 83-88.

41. Прокоп’єв С. (2024). Порівняння backend та frontend тестування у веб-застосунках.