ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ВІРТУАЛЬНОГО СЕРЕДОВИЩА ДЛЯ ТЕСТУВАННЯ ТА НАВЧАННЯ МЕТОДАМ КІБЕРЗАХИСТУ

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Виконавець роботи Левцун Тимур Володимирович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.п.н. Кошова О. П.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 84 с., 10 рис., 1 таблиця, 1 додаток, 9 джерел.

CYBERSECURITY, REACT, DOCKER, NODE.JS, SQL INJECTION, XSS, МІКРОСЕРВІСИ, НАВЧАЛЬНА ПЛАТФОРМА

Об’єктом дослідження є процес розробки системи для навчання кіберзахисту з використанням вразливих вебдодатків.

Мета роботи — створити інтерактивну вебплатформу, що дозволяє користувачам вивчати основні веб-вразливості шляхом теоретичного ознайомлення та практичного моделювання атак у захищеному середовищі. Платформа повинна поєднувати зручний інтерфейс, системну ізоляцію компонентів, а також бути масштабованою й придатною до використання у навчальних закладах.

У процесі виконання роботи реалізовано такі функціональні можливості:

- фронтенд-платформа, побудована на базі React.js, з підтримкою тем, навігації та інтеграції з Docker-середовищем;
- набір мікросервісів, кожен з яких імітує певну вебвразливість (SQLi, XSS, CSRF, File Upload, XXE);
- вбудований теоретичний модуль з описами вразливостей, прикладами та рекомендаціями щодо захисту;
- запуск ізольованих середовищ за допомогою Docker Compose;
- обробка запитів користувачів і логування їхніх дій для подальшого аналізу;
- реалізація мінімалістичного, адаптивного дизайну з акцентом на UX.

Для реалізації архітектури було використано стек технологій: Node.js + Express для мікросервісів, React для інтерфейсу, Docker для контейнеризації, MySQL для роботи з базами даних. Структура проєкту передбачає поділ на окремі логічні блоки, що дозволяє легко масштабувати систему та додавати нові вразливості.

Результатом дипломної роботи стала платформа з інтуїтивно зрозумілим інтерфейсом, що забезпечує ефективне вивчення кіберзагроз. Рішення протестовано

у локальному середовищі, підтверджено його стабільність та придатність для навчального використання.

Практичне значення роботи полягає у створенні інструменту, що може бути застосований у закладах освіти, на технічних факультетах, курсах з кібербезпеки, а також при самостійному навчанні. Платформа дозволяє студентам краще розуміти принципи вебатак та методи їх запобігання.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Application Programming Interface – інтерфейс прикладного програмування
URL	Uniform Resource Locator – уніфікований покажчик ресурсу
SQL Injection	Тип вразливості, при якій зловмисник вставляє SQL-код у запит до бази даних
XSS	Cross-Site Scripting – міжсайтове виконання скриптів
CSRF	Cross-Site Request Forgery – міжсайтове підроблення запиту
XXE	XML External Entity – зовнішні сутності XML
Docker	Платформа контейнеризації застосунків
Docker Compose	Інструмент для оркестрації багатоконтейнерних застосунків
React	JavaScript-бібліотека для побудови користувацького інтерфейсу
Node.js	Середовище виконання JavaScript на стороні сервера
Express.js	Легкий фреймворк для побудови вебзастосунків на Node.js
MySQL	Реляційна система керування базами даних
EJS	Embedded JavaScript – шаблонізатор HTML для Node.js

ВСТУП

У сучасну епоху цифровізації кібербезпека стала одним із ключових напрямів не лише в IT-індустрії, а й у державному управлінні, освіті, бізнесі та повсякденному житті. Зі зростанням кількості онлайн-сервісів, вебзастосунків і взаємодії користувачів із мережею Інтернет, підвищується і рівень ризиків, пов’язаних із несанкціонованим доступом, витоком даних, атаками на вебресурси та іншими кіберзагрозами. Тому важливою складовою підготовки фахівців у галузі інформаційних технологій є практичне навчання методам виявлення та запобігання кібератакам.

Одним із ефективних підходів до такого навчання є створення віртуальних навчальних середовищ, у межах яких студенти можуть безпечно взаємодіяти з імітацією типових вразливостей, застосовувати експлойти, досліджувати наслідки атак і вивчати методи їх усунення. Такі платформи дають змогу поєднувати теоретичні знання з практичними навичками, що є надзвичайно важливим для формування компетентного фахівця з кібербезпеки.

Мета роботи — створити інтерактивну вебплатформу, що дозволяє користувачам вивчати основні веб-вразливості шляхом теоретичного ознайомлення та практичного моделювання атак у захищеному середовищі. Платформа повинна поєднувати зручний інтерфейс, системну ізоляцію компонентів, а також бути масштабованою й придатною до використання у навчальних закладах.

Завдання, що були поставлені в процесі реалізації:

- Проаналізувати сучасні підходи до побудови віртуальних навчальних середовищ з кібербезпеки;
- Розробити архітектуру системи на основі мікросервісів та контейнеризації (Docker);
- Реалізувати окремі мікросервіси з популярними типами вразливостей (SQL Injection, XSS, Command Injection, CSRF, File Upload, XXE);
- Створити вебінтерфейс на базі React для навігації по модулях, вивчення теорії та доступу до практичних завдань;

- Забезпечити ізоляцію компонентів та безпечне середовище для тестування;
- Розробити механізм відстеження прогресу та перевірки знань;
- Налаштувати конфігурацію контейнерів за допомогою Docker Compose;
- Оформити супровідну документацію та інструкцію для користувачів.

Об’єктом дослідження є процес розробки системи для навчання кіберзахисту з використанням вразливих вебдодатків.

Предметом дослідження є вебплатформа, що поєднує інтерфейс користувача на React, мікросервіси з уразливостями на Node.js, базу даних MySQL та механізми контейнеризації Docker.

Практичне значення роботи полягає у створенні повноцінного навчального інструменту, який дозволяє студентам, викладачам та фахівцям з кібербезпеки ефективно вивчати механізми атак та захисту в безпечному віртуальному середовищі. Платформа може використовуватись у навчальних курсах, тренінгах та сертифікаційних програмах з інформаційної безпеки.

У процесі виконання дослідження було застосовано методи аналізу архітектурних шаблонів, емпіричне програмування, тестування у Docker-середовищі, а також огляд існуючих рішень та інструментів для навчання з кібербезпеки.

Дипломна робота складається з теоретичної та практичної частин. У теоретичній частині описано принципи реалізації вебвразливостей, механізми їхнього виявлення та методи захисту. Практична частина містить опис архітектури проєкту, реалізації окремих компонентів та інструкцію з використання системи. У завершальному розділі наведено висновки щодо досягнутих результатів і перспективи подальшого розвитку платформи.

1.ПОСТАНОВКА ЗАДАЧІ

У сучасному інформаційному просторі відео- та аудіоконтент відіграють ключову роль у передачі знань, інформації та розваг. Зважаючи на стрімке зростання обсягів мультимедійних матеріалів, зростає й потреба у зручних інструментах для їх швидкої обробки та завантаження. Особливо актуальними є рішення, які дозволяють виконувати ці дії безпосередньо у середовищі користувача, наприклад, у месенджерах.

Метою цієї дипломної роботи є розробка функціонального Telegram-бота, який забезпечує завантаження та обробку відео- й аудіофайлів із популярних платформ, зокрема YouTube, TikTok, Instagram і Facebook. Передбачено, що користувач зможе отримувати відео, конвертувати його у формат аудіо, обрізати потрібні фрагменти, завантажувати цілі плейлисти, а також здійснювати пошук музики за назвою — усе це без виходу з месенджера Telegram.

Для досягнення поставленої мети визначено наступні задачі:

- Забезпечити підтримку платформ YouTube, TikTok, Instagram та Facebook, включаючи розпізнавання коротких і повних форматів URL.
- Реалізувати функціональність завантаження відео у форматі .mp4 і витягування аудіо у форматі .mp3 з можливістю іменування файлів відповідно до назви відео.
- Впровадити модуль пошуку музики за назвою з використанням YouTube API або аналога, з автоматичним вибором першого релевантного результату.
- Додати можливість обрізання аудіофрагментів за вказаним таймінгом, з подальшим збереженням отриманого треку.
- Реалізувати функцію завантаження повних YouTube-плейлистів у вигляді структурованої директорії з нумерацією композицій.
- Інтегрувати розроблений функціонал із Telegram Bot API, включаючи обробку команд /start, /video, /audio, /music, /cut_audio, /playlist.
- Побудувати логічну архітектуру проєкту на основі модульного підходу з

виділенням окремих блоків для обробки відео, аудіо, запитів до API та взаємодії з Telegram.

- Застосувати систему конфігураційних змінних за допомогою .env файлу для гнучкості та безпеки налаштувань.
- Реалізувати механізми логування помилок та обробки виняткових ситуацій з наданням зворотного зв’язку користувачеві.
- Забезпечити автоматичне очищення тимчасових файлів після завершення обробки та надсилання результату користувачеві.
- Підготувати супровідну документацію, що містить опис команд та інструкцію з локального запуску або розгортання на сервері.

Ключові вимоги до програмного забезпечення включають:

- **Зручність використання** — усі функції повинні бути доступні в межах простих Telegram-команд без необхідності у додаткових діях з боку користувача.
- **Стабільність** — обробка файлів великого розміру повинна відбуватись коректно без збоїв.
- **Швидкодія** — мінімальний час очікування між надсиланням запиту та отриманням результату.
- **Масштабованість** — можливість легко розширювати функціонал (наприклад, додати підтримку субтитрів, конвертацію у GIF, чи інтеграцію з хмарними сховищами).

Таким чином, реалізація Telegram-бота з описаним функціоналом дозволить створити сучасний інструмент для роботи з мультимедійним контентом, який поєднує в собі простоту, ефективність і широкий спектр можливостей, що є цінним як для звичайних користувачів, так і для професіоналів у сфері медіа, освіти чи цифрової безпеки.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Кіберзахист: основні поняття і виклики

У цьому розділі інформаційного огляду можна розглянути основні поняття, які стосуються кіберзахисту, а також сучасні виклики, з якими стикаються організації та індивідуальні користувачі в контексті забезпечення інформаційної безпеки.

1. Основні поняття кіберзахисту:

Кіберзахист - це заходи та процеси, що використовуються для захисту комп'ютерних систем, мереж та даних від несанкціонованого доступу або атак, які мають на меті експлуатацію.

Інформаційна безпека - захист інформації від несанкціонованого доступу, використання, розкриття, спотворення, модифікації, перевірки, запису або знищення.

Кіберзагроза - потенційна шкідлива операція, яка виконується через цифрові системи проти одного або більше користувачів.

Кібератака - спроба несанкціонованого втручання в системи, мережі або пристрої, зазвичай з метою крадіжки, зміни або знищення інформації.

Кібербезпека - практика захисту комп'ютерів, серверів, мобільних пристроїв, електронних систем, мереж та даних від зловмисних атак.

2. Сучасні виклики кіберзахисту:

- Зростання кіберзагроз: Швидкий розвиток нових технологій, таких як Інтернет речей (IoT) та хмарні обчислення, створює нові можливості для кіберзлочинців.

- Софістикованість атак: Кібератаки стають все більш складними та важко виявити, з використанням передових методів, таких як поліморфні віруси та шифрування.

- Людський фактор: Помилки користувачів або ненавмисні дії співробітників залишаються однією з найбільших вразливостей у кіберзахисті.

- Інсайдерські загрози: Загрози зсередини організації можуть бути найбільш шкідливими, оскільки інсайдери мають доступ до важливих систем та даних.

- Швидкість розвитку технологій: Швидке впровадження нових технологій часто випереджає розробку відповідних заходів безпеки.

- Кібервійна: Держави використовують кіберпростір для здійснення шпигунства, саботажу та інших форм гібридної війни.

- Законодавчі та нормативні виклики: Необхідність відповідності різним національним та міжнародним законам та стандартам може бути складною.

- Захист приватності: Забезпечення конфіденційності особистих даних у відповідності до регуляцій, таких як GDPR.

Для кожного з цих викликів потрібно розробити стратегії та рішення, які дозволять зміцнити кіберзахист та зменшити ризики. Це включає технічні рішення, такі як вдосконалення програмного забезпечення та апаратних засобів, а також організаційні заходи, такі як поліпшення політик безпеки, процедур та навчання персоналу.

2.2. Віртуальне навчання та тестування у сфері кіберзахисту

У цьому розділі можна розглянути, як віртуальне навчання та тестування можуть бути використані для підготовки фахівців у сфері кіберзахисту, а також для перевірки та вдосконалення систем безпеки.

1. Віртуальне навчання:

- Онлайн курси та сертифікації: Представлення різноманітних онлайн-платформ, які пропонують курси та сертифікаційні програми з кіберзахисту.

- Симуляції та ігри: Використання ігрових технологій для створення симуляційних середовищ, де можна безпечно відпрацьовувати реакції на кібератаки.

- Віртуальні лабораторії: Створення віртуальних лабораторій, які дозволяють студентам та фахівцям експериментувати з мережевими конфігураціями та тестуванням безпеки.

- Модульне навчання: Розробка модульних програм, які дозволяють користувачам вивчати конкретні аспекти кіберзахисту за власним графіком.

2. Тестування у сфері кіберзахисту:

- Тестування на проникнення: Використання віртуальних середовищ для імітації атак на системи з метою виявлення вразливостей.

- Військові навчання (Cyber Ranges): Створення складних сценаріїв, які імітують масштабні кібератаки для тренування команд реагування на інциденти.

- Автоматизоване тестування безпеки: Використання інструментів для автоматичного сканування та тестування веб-додатків та інших систем на наявність вразливостей.

- Ігрові вправи: Проведення командних ігрових вправ, які допомагають розвивати навички командної роботи та стратегічного планування в контексті кіберзахисту.

3. Інтеграція навчання та тестування:

- Адаптивне навчання: Використання систем, які адаптують навчальний матеріал в залежності від прогресу та потреб учня.

- Неперервне оцінювання: Розробка методів для неперервного оцінювання навичок та знань учнів через регулярні тестування та вправи.

- Інтерактивність та залученість: Створення інтерактивних вправ, які залучають учнів та мотивують їх до практичного застосування знань.

- Зворотний зв'язок та аналітика: Використання аналітичних інструментів для збору даних про ефективність навчання та надання зворотного зв'язку учням.

Віртуальне навчання та тестування у сфері кіберзахисту дозволяють створити безпечне та контрольоване середовище для розвитку навичок, необхідних для захисту від реальних кіберзагроз. Ці методи також сприяють підвищенню гнучкості та доступності навчання, що є ключовим для швидкого реагування на постійно змінювані виклики у сфері кібербезпеки.

2.3. Аналіз існуючих підходів та інструментів для віртуального навчання

У цьому розділі можна провести аналіз існуючих підходів та інструментів, які застосовуються для віртуального навчання у сфері кіберзахисту. Це включає огляд платформ, програмного забезпечення, методологій та навчальних ресурсів, які використовуються для підготовки фахівців у цій галузі.

1. Платформи для віртуального навчання:

- MOOCs (Massive Open Online Courses): Платформи, такі як Coursera, edX, Udacity, які пропонують курси з кібербезпеки від відомих університетів та коледжів.

- Спеціалізовані платформи з кібербезпеки: Платформи, такі як Cybrary, Infosec Institute, які зосереджені виключно на кібербезпекі та надають спеціалізовані курси та тренінги.

2. Програмне забезпечення для віртуальних лабораторій:

- Симулятори та емулятори: Інструменти, які дозволяють створювати віртуальні мережі та системи для тренування, наприклад, GNS3, Cisco Packet Tracer.

- Віртуальні машини та контейнери: Використання VMware, VirtualBox, Docker для створення ізольованих середовищ, де можна безпечно відтворювати та аналізувати кібератаки.

3. Методології віртуального навчання:

- Blended Learning: Комбінація онлайн навчання з традиційними класними заняттями.

- Flipped Classroom: Метод, при якому теоретичний матеріал вивчається самостійно онлайн, а практичні заняття проводяться в класі.

- Gamification: Використання ігрових елементів для збільшення мотивації та залучення учнів.

4. Навчальні ресурси та інструменти:

- Інтерактивні підручники: Наприклад, "WebGoat" від OWASP для вивчення веб-безпеки через інтерактивні вправи.

- Відео та вебінари: Використання відео на платформах як YouTube або

спеціалізованих вебінарів для демонстрації та пояснення концепцій.

- Форуми та спільноти: Платформи для обговорення та обміну знаннями, такі як Stack Overflow, Reddit, спеціалізовані групи LinkedIn.

Оцінювання та сертифікація:

- Онлайн іспити та тестування: Використання платформ для проведення іспитів та тестів з автоматичною перевіркою відповідей.

- Бейджі та сертифікати: Надання цифрових бейджів та сертифікатів, які можна використовувати для демонстрації кваліфікації.

5. Аналіз ефективності:

- Аналітика навчання: Збір та аналіз даних про успішність учнів для вдосконалення курсів та методів навчання.

- Зворотний зв'язок від учнів: Регулярне збирання відгуків від учнів для покращення якості навчального матеріалу та підходів до навчання.

Аналізуючи існуючі підходи та інструменти, можна визначити найбільш ефективні стратегії для розробки та впровадження віртуальних навчальних програм у сфері кіберзахисту. Це також дозволить ідентифікувати потенційні прогалини та можливості для інновацій у цій області.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Огляд сучасних методів віртуального навчання

Віртуальне навчання стає все більш популярним у сучасному світі, завдяки технологічному прогресу та зростанню потреби в гнучкості освітніх процесів. Сучасні методи віртуального навчання можуть бути використані для навчання різних аспектів кіберзахисту. Давайте розглянемо деякі з них:

1. Віртуальні лабораторії: це спеціалізовані середовища, які дозволяють студентам взаємодіяти з реальними системами та програмами в безпечному середовищі. Вони ідеально підходять для вивчення практичних навичок у сфері кіберзахисту.
2. Імітаційне моделювання: цей метод використовує програмне забезпечення для імітації реальних кібератак та захисних механізмів, дозволяючи студентам вивчати реакції на різні загрози.
3. Геймифікація: використання ігрових елементів у навчальному процесі може зробити навчання більш захоплюючим та ефективним. Ігри на тему кіберзахисту можуть допомогти студентам краще зрозуміти складні концепції.
4. Мобільне навчання: завдяки розвитку мобільних технологій, студенти можуть вивчати кіберзахист будь-де і будь-коли, використовуючи свої смартфони або планшети.
5. Обернене навчання: цей метод передбачає, що основна частина навчального матеріалу вивчається студентами самостійно, а час у класі використовується для обговорення, практичних завдань та розв'язання проблем.
6. Співпраця в реальному часі: завдяки сучасним технологіям, студенти можуть спілкуватися та співпрацювати з іншими учасниками навчального процесу в реальному часі, незалежно від їх географічного розташування.

Сучасні методи віртуального навчання надають величезні можливості для підготовки фахівців у сфері кіберзахисту[1]. Вони дозволяють студентам

отримувати практичний досвід, адаптуватися до змінюваних умов та ефективно спілкуватися з іншими учасниками навчального процесу.

Сучасний ринок пропонує багато платформ для віртуального навчання з кіберзахисту. Ці платформи надають можливість отримати практичний досвід, виконуючи реальні завдання та виклики. Деякі з них:

1. HackTheBox (HTB): це онлайн-платформа, яка дозволяє користувачам тестувати та покращувати свої навички в області кіберзахисту. HTB пропонує ряд віртуальних машин, які користувачі можуть атакувати, щоб отримати доступ до них(див рис 2.1). Це відмінний спосіб навчитися реальним методам атаки та захисту в безпечному середовищі.

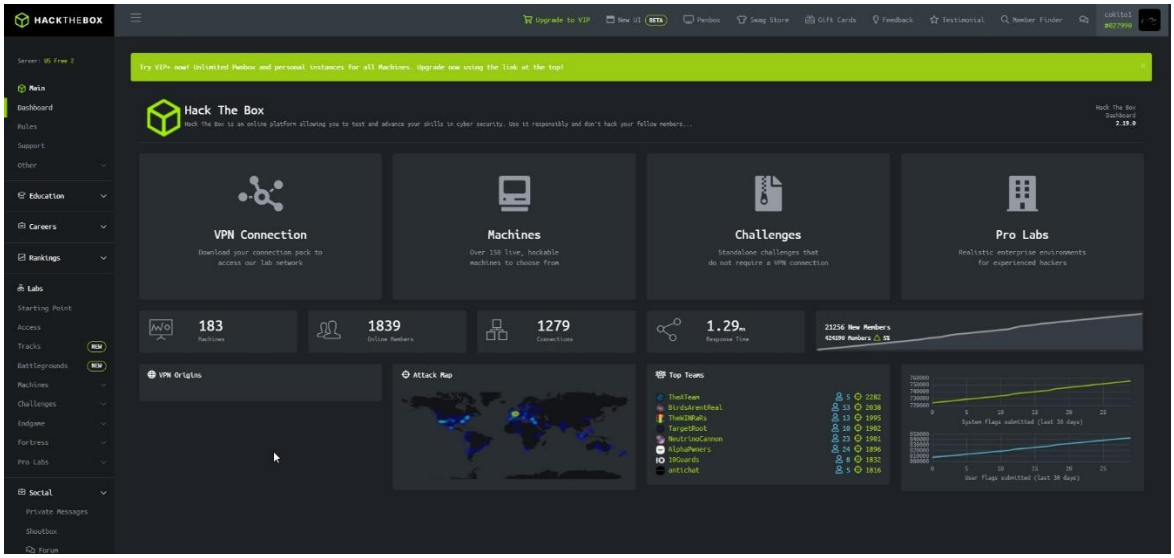

Рисунок 2.1 – HackTheBox

2. TryHackMe: ця платформа також пропонує віртуальне навчальне середовище для тестування та покращення навичок кіберзахисту(див рис. 2.2). Вона включає в себе ряд навчальних курсів та викликів, які допомагають користувачам розвивати свої навички в різних областях кіберзахисту.

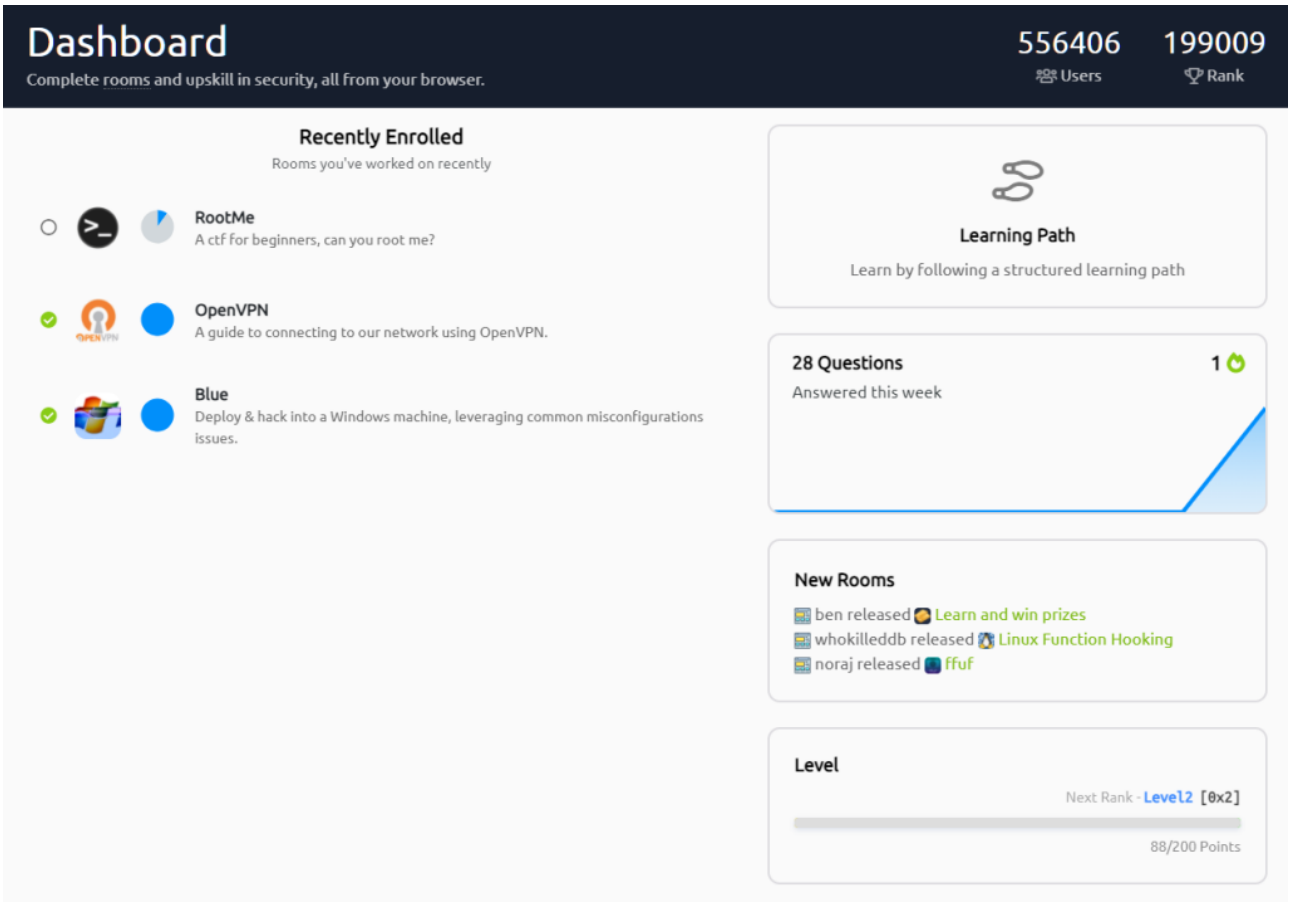

Рисунок 2.2 – TryHackMe

3. PortSwigger: відомий завдяки своєму інструменту для тестування веб-застосунків - Burp Suite(див рис. 2.3). PortSwigger також пропонує безкоштовні навчальні матеріали та ресурси для вивчення веб-безпеки. Їхній веб-сайт містить багато практичних завдань, які допомагають користувачам навчитися різним аспектам веб-безпеки(див рис. 2.4).

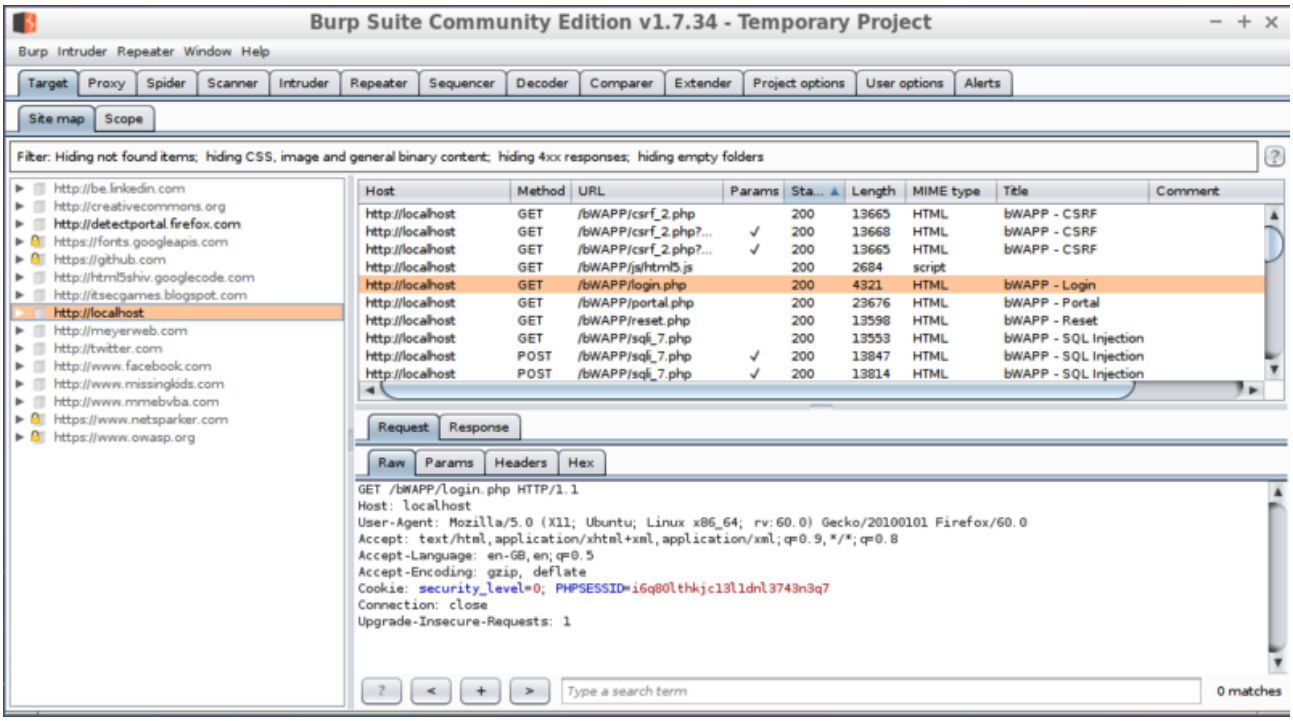

Рисунок 2.3 – інструмент Burp Suite

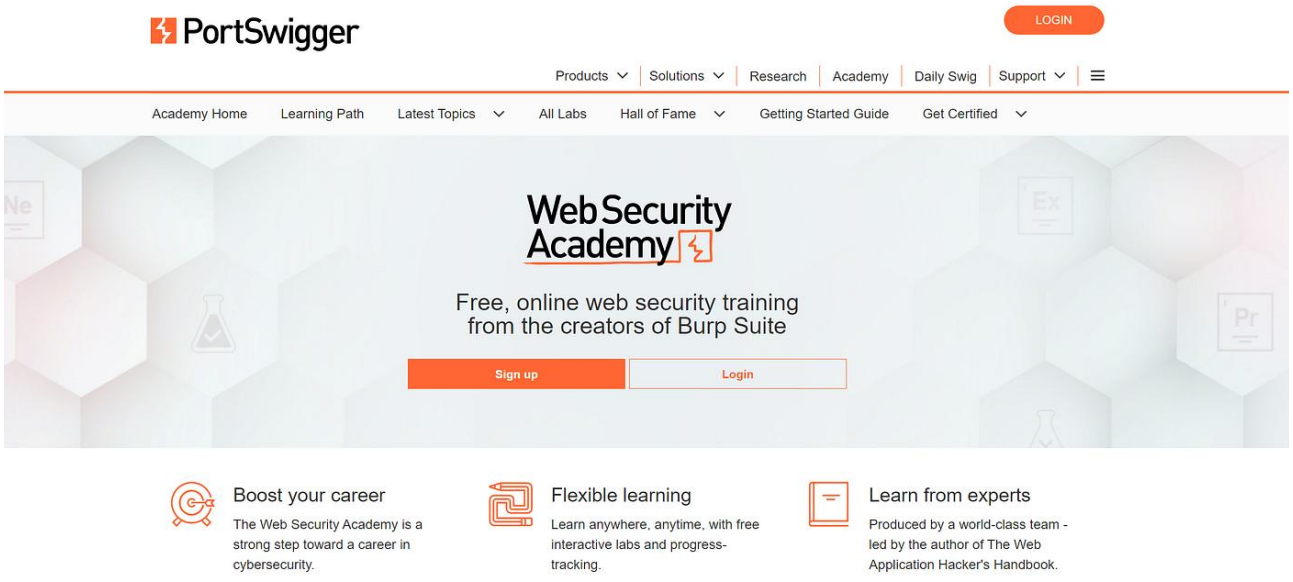

Рисунок 2.4 – навчальна платформа PortSwigger

Ці платформи надають величезні можливості для тих, хто хоче навчитися кіберзахисту. Вони пропонують реальні завдання та виклики, які допомагають користувачам отримати практичний досвід та покращити свої навички.

3.2. Технології та інструменти для створення віртуального середовища

Створення віртуального середовища для навчання та тестування методів кіберзахисту вимагає використання спеціалізованих технологій та інструментів. Давайте розглянемо деякі з них:

1. Віртуалізація:

- VMware: це один з найпопулярніших інструментів для створення та управління віртуальними машинами. VMware дозволяє створювати ізольовані середовища для тестування та навчання.
- VirtualBox: безкоштовний інструмент від Oracle, який також дозволяє створювати віртуальні машини на різних операційних системах.

2. Контейнеризація:

- Docker[2]: це платформа для створення, розгортання та управління контейнерами. Docker дозволяє пакувати програми з усіма їхніми залежностями в контейнери, які можна легко переносити та запускати.
- Kubernetes: система оркестрації контейнерів, яка дозволяє автоматизувати розгортання, масштабування та управління контейнерізованими додатками.

3. Середовища для моделювання мережі:

- GNS3: графічний інструмент для моделювання та тестування мережевих конфігурацій.
- Cisco Packet Tracer: віртуальне мережеве середовище, яке дозволяє створювати мережеві симуляції.

4. Платформи для тестування безпеки:

- Metasploit[3]: інструмент для розробки, тестування та виконання експлойтів проти віддалених машин.
- OWASP ZAP: відкритий інструмент для тестування веб-застосунків на наявність вразливостей.

5. Інструменти для автоматизації:

- Ansible: інструмент для автоматизації конфігурації, управління та

розгортання додатків.

- Terraform: інструмент для опису та створення інфраструктури як коду.

Використання цих технологій та інструментів дозволяє створювати віртуальні середовища, які відтворюють реальні мережеві сценарії та вразливості, надаючи можливість для практичного навчання та тестування методів кіберзахисту[4].

3.3. Класифікація кіберзагроз

1. Маліційне програмне забезпечення (Malware):

- Віруси: Програми, які прикріплюються до інших програм та виконують шкідливі дії, такі як знищення даних.

- Черв'яки: Самореплікуючі програми, які розповсюджуються через мережі, використовуючи вразливості в програмному забезпеченні.

- Троянські програми (Trojans): Маскуються під легітимне програмне забезпечення, але виконують шкідливі дії без відома користувача.

- Рекламне ПЗ (Adware) та Шпигунське ПЗ (Spyware): Програми, що відображають небажану рекламу та/або стежать за діями користувача без його згоди.

2. Атаки на мережеву інфраструктуру:

- Атаки "відмова у обслуговуванні" (DoS/DDoS): Завантажують мережеві ресурси або сервери до такої міри, що легітимні користувачі не можуть до них отримати доступ.

- Атаки "людина посередині" (MitM)[5]: Несанкціоноване перехоплення або модифікація даних між двома сторонами.

- Сеансове викрадення (Session Hijacking): Перехоплення та використання сеансових ключів для отримання несанкціонованого доступу до інформації або сервісів.

3. Цільові атаки:

- Фішинг: Використання підроблених електронних повідомлень або веб-

сайтів для введення в оману користувачів з метою отримання конфіденційної інформації.

- Спірфінг: Маскування атакуючого під іншого користувача або пристрій для обману системи або інших користувачів.

- Вішинг (Voice Phishing): Використання телефонного зв'язку для отримання конфіденційної інформації від жертви.

- Фармінг: Перенаправлення користувачів на фальшивий веб-сайт без їх відома, зазвичай через зміну DNS записів.

4. Розширені постійні загрози (APT):

- APT кампанії: Довготривалі та цілеспрямовані атаки, які використовують різноманітні методи для отримання несанкціонованого доступу до інформаційних ресурсів. Часто використовуються для шпіонажу або саботажу.

5. Інші загрози:

- Інсайдерські загрози: Шкідливі дії, вчинені співробітниками або колишніми працівниками, які мають доступ до мережі та ресурсів компанії.

- Фізичні загрози: Ушкодження або крадіжка фізичних компонентів мережі або інформаційних систем.

- Соціальна інженерія: Методи, які використовують психологічний вплив на людей для отримання конфіденційної інформації або доступу до систем.

Кожна з цих категорій має свої підтипи та специфічні методи атаки, які постійно еволюціонують. Розуміння цих загроз є ключовим для розробки стратегій захисту та створення ефективних віртуальних середовищ для навчання кіберзахисту.

3.4. Методи захисту

1. Антивірусне програмне забезпечення:

- Сканування на віруси: Використання антивірусних програм для перевірки файлів на наявність відомих вірусів за допомогою сигнатурних баз.

- Гевристичний аналіз: Використання алгоритмів для виявлення невідомих

вірусів на основі підозрілої поведінки.

- Захист в реальному часі: Моніторинг системи на предмет шкідливих дій відразу ж після їх виявлення.

2. Мережеві брандмауери та IDS/IPS системи:

- Брандмауери: Встановлення правил, які контролюють вхідний та вихідний мережевий трафік для блокування несанкціонованих спроб доступу.

- Системи виявлення вторгнень (IDS): Аналіз мережевого трафіку на предмет відомих атак або підозрілої поведінки.

- Системи запобігання вторгнень (IPS): Активне блокування атак на основі інформації отриманої від IDS.

3. Шифрування даних:

- Шифрування на рівні диска: Захист інформації шляхом шифрування всього диска, що запобігає доступу до даних без відповідного ключа.

- Шифрування передачі даних: Використання протоколів, таких як SSL/TLS, для захисту даних, які передаються через мережу.

- Кінцеве шифрування: Шифрування повідомлень або даних перед їх відправкою, з розшифровкою лише на стороні отримувача.

4. Менеджмент ідентифікації та доступу (IAM):

- Сильна аутентифікація: Використання багатофакторної аутентифікації для підвищення рівня безпеки при вході в систему.

- Управління привілеями: Надання користувачам мінімально необхідних прав для виконання їхніх завдань.

- Аудит та моніторинг: Відстеження та запис дій користувачів для виявлення несанкціонованої або підозрілої активності.

5. Фізичний захист:

- Контроль доступу: Встановлення бар'єрів та механізмів контролю доступу для запобігання несанкціонованому фізичному доступу до критичних систем.

- Захист від стихійних лих: Використання заходів для захисту від природних катастроф, таких як пожежі, повені тощо[6].

6. Резервне копіювання та відновлення:

- Регулярне резервне копіювання: Створення копій важливих даних для забезпечення можливості відновлення після інцидентів.

- План відновлення після аварій (DRP): Розробка стратегій для швидкого відновлення операцій після серйозних інцидентів.

7. Політики безпеки та освітні програми:

- Розробка політик безпеки: Встановлення чітких правил та процедур для забезпечення безпеки інформації.

- Тренінги з безпеки: Проведення регулярних навчальних сесій для співробітників з метою підвищення обізнаності про кіберзагрози та методи захисту.

8. Реагування на інциденти:

- План реагування на інциденти: Розробка та впровадження процедур для ефективного реагування на безпекові інциденти.

- Команди швидкого реагування (CERT/CSIRT): Створення спеціалізованих груп для реагування на інциденти та відновлення операцій.

9. Законодавче регулювання та стандарти:

- Дотримання законодавства: Забезпечення відповідності діяльності організації законодавчим вимогам у сфері кібербезпеки.

- Слідування стандартам: Впровадження міжнародних та національних стандартів безпеки, таких як ISO/IEC 27001, NIST Cybersecurity Framework тощо.

Ці методи захисту не існують ізольовано; вони найефективніші, коли використовуються у комплексі, створюючи багаторівневу систему захисту, яка може захистити організацію від різноманітних кіберзагроз.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Загальна структура платформи та архітектура проєкту

Розроблена платформа є віртуальним середовищем для тестування та навчання методам кіберзахисту. Вона побудована за принципами мікросервісної архітектури, де кожна вразливість реалізована як окремий серверний додаток, ізольований у власному Docker-контейнері. Такий підхід забезпечує високу масштабованість, гнучкість та безпечну експлуатацію всієї системи. Завдяки ізоляції кожен сервіс може працювати незалежно від інших, що дозволяє проводити експерименти, не порушуючи роботу інших компонентів.

Центральне місце у структурі займає веб-інтерфейс, розроблений за допомогою React.js. Через нього користувачі отримують доступ до навчального контенту, обирають потрібну тему, переглядають теоретичні матеріали, виконують практичні завдання та спостерігають за своїм прогресом. Інтерфейс реалізовано у вигляді єдиної сторінки з маршрутизацією, що забезпечує швидку навігацію та зручність у користуванні.

Кожен мікросервіс відповідає за демонстрацію конкретної уразливості, зокрема SQL Injection, Cross-Site Scripting (XSS), Command Injection, File Upload Vulnerability, Cross-Site Request Forgery (CSRF) та XML External Entity (XXE). Ці сервіси виконуються на окремих портах і мають власні обробники запитів. Уразливості реалізовано з навмисними помилками у коді для того, щоб користувач мав змогу практично випробувати техніки атак та захисту. Комунікація між мікросервісами та базою даних здійснюється через вбудовану віртуальну мережу Docker, яка забезпечує взаємодію між контейнерами в межах одного середовища.

Для зберігання даних використовується реляційна база даних MySQL, яка також розгортається у вигляді контейнера. Вона використовується для збереження облікових записів, коментарів, логів тощо, що необхідні для демонстрації атак. База даних автоматично ініціалізується при запуску системи за допомогою SQL-скриптів, які завантажуються в контейнер під час його старту.

Всі компоненти проєкту керуються за допомогою Docker Compose. У конфігураційному файлі docker-compose.yml описано всі сервіси, змінні середовища, порти, залежності між компонентами, параметри моніторингу доступності (health checks), а також іменовані томи для збереження стану бази даних. Завдяки цьому запуск усієї платформи зводиться до виконання однієї команди, що значно спрощує розгортання в будь-якому середовищі.

Для забезпечення стабільної роботи системи передбачено механізми моніторингу, зокрема використання healthcheck-ів для виявлення збоїв сервісів. У разі виявлення помилок відповідні контейнери перезапускаються автоматично, що підвищує надійність всієї інфраструктури. Крім того, у конфігурації Docker вказані обмеження на використання ресурсів, політики безпеки контейнерів, а також директиви, що забороняють ескалацію привілеїв.

Проєкт також передбачає можливість розширення: у платформу можна додавати нові типи вразливостей, додаткові теоретичні розділи, систему оцінювання, багатокористувацький режим тощо. Така модульна структура робить платформу зручною для адаптації до різних освітніх програм у сфері інформаційної безпеки.

Загалом, архітектура платформи об’єднує передові підходи до проєктування навчальних середовищ: вона поєднує зручність веб-інтерфейсу, гнучкість контейнеризації, безпечність ізоляції та реалізацію реальних прикладів атак у контрольованому середовищі. Це дозволяє ефективно навчати основам кіберзахисту як початківців, так і слухачів із досвідом.

4.2. Огляд функціональних можливостей системи

Розроблена платформа надає користувачеві набір інструментів для повноцінного вивчення методів атак на вебдодатки та відповідних заходів захисту. Система поєднує теоретичну базу знань із практичними вправами, що дозволяє опанувати як базові, так і складні сценарії кіберзагроз. Особливу увагу приділено інтерактивності, простоті використання та безпеці самого навчального середовища.

Навчальні функції

Однією з ключових особливостей платформи є модульність контенту: кожна вразливість представлена у вигляді окремої картки, яка містить короткий опис, посилання на теоретичні матеріали та кнопку для запуску практичного модуля. Для кожної теми передбачено:

- Теоретичний блок — пояснення принципів роботи вразливості, історичні приклади реальних атак, типові payload-и, приклади коду та заходи запобігання.
- Практичне завдання — симуляція вразливого вебдодатку, де користувач має можливість самостійно провести атаку.
- Система підказок — допомагає студенту поступово прийти до правильного рішення, даючи натяки та приклади без прямого розв’язку.
- Відстеження прогресу — уся взаємодія з платформою (початок вивчення, завершення вправи, остання активність) зберігається локально в браузері, щоб користувач міг бачити свій розвиток.

Сценарії взаємодії з системою

У межах використання платформи користувач проходить низку типових сценаріїв, що відповідають реальним навчальним цілям. Нижче наведено приклади таких сценаріїв:

Сценарій 1: Вивчення SQL Injection

1. Користувач відкриває головну сторінку платформи.
2. Вибирає картку з темою “SQL Injection”.
3. Переглядає теоретичні матеріали, дізнається про суть ін’єкцій та типові атаки.
4. Запускає практичний модуль через окрему вкладку браузера.
5. Проводить атаку, використовуючи запропоновані приклади.
6. Після успішної атаки прогрес оновлюється автоматично.

Сценарій 2: Демонстрація XSS

1. Користувач переходить до розділу “XSS”.
2. Вивчає різновиди XSS (Reflected, Stored, DOM-based).
3. Відкриває вразливий сервіс, у якому реалізована система коментарів.

4. Вводить XSS payload у форму.
5. Перевіряє виконання скрипта у браузері.
6. Аналізує, як це можна було запобігти — відповідні пояснення подано нижче у матеріалі.

Такі сценарії дозволяють структурувати навчання, сприяють глибшому розумінню механізмів атак і формують практичні навички, необхідні для майбутнього фахівця з кібербезпеки.

Система безпеки та ізоляції

Особливу увагу при проєктуванні було приділено захисту самої навчальної інфраструктури. Оскільки користувачі навчаються проводити атаки, важливо забезпечити повну ізоляцію між модулями, а також захист хост-машини.

З цією метою в платформі реалізовано наступні механізми:

- Контейнерна ізоляція — кожен мікросервіс виконується в окремому Docker-контейнері з мінімальними привілеями. Це унеможливлює поширення впливу між сервісами або на інші частини системи.
- Обмеження доступу до хосту — всі сервіси не мають доступу до файлової системи або мережі поза межами віртуальної Docker-мережі.
- Автоматичне відновлення сервісів — у разі збою або навмисного “виведення з ладу” контейнера з боку користувача, docker-compose автоматично перезапускає відповідний сервіс.
- Логування — у логах можна простежити, як користувач взаємодіє з сервісом, що дозволяє викладачеві або адміністратору проводити аудит активності.

Адміністративні можливості

Хоча поточна версія орієнтована переважно на кінцевого користувача, архітектура платформи дозволяє легко інтегрувати адміністративні функції:

- Додавання нових сценаріїв вразливостей.
- Оновлення теоретичних блоків без перезбірки контейнерів.
- Аудит дій користувачів через лог-файли або інтеграцію з SIEM-системами.
- Встановлення багатокористувацького режиму з автентифікацією.

Таким чином, розроблена система не лише виконує роль інтерактивного

тренажера з кібербезпеки, а й може слугувати основою для повноцінного освітнього комплексу з можливістю масштабування та кастомізації.

4.3. Реалізація модуля фронтенду на базі React

Фронтенд-платформа реалізована з використанням бібліотеки **React.js** версії 18.2.0. Вона забезпечує динамічну взаємодію користувача з інтерфейсом, зручну маршрутизацію між теоретичними та практичними розділами, а також надає можливість зберігати прогрес локально у браузері.

Основні цілі реалізації фронтенду:

- Зробити навігацію інтуїтивно зрозумілою;
- Надати структуру для відображення вразливостей;
- Забезпечити зв’язок між теорією та практикою;
- Реалізувати збереження прогресу для кожного користувача.

Структура React-додатку

Проєкт побудований на компонентній архітектурі. Основна структура виглядає так:

```
web/
├── src/
│   ├── App.js              # Головний компонент з маршрутизатором
│   ├── App.css             # Глобальні стилі
│   └── components/
│       ├── Home.js         # Головна сторінка з переліком вразливостей
│       ├── Theory.js       # Теоретичні матеріали по кожній темі
│       ├── Home.css        # Стилі для головної сторінки
│       └── Theory.css      # Стилі для сторінки теорії
```

Головний компонент App.js

Файл App.js відповідає за ініціалізацію прогресу, конфігурацію вразливостей та маршрутизацію між сторінками.

```
import React, { useState, useEffect } from 'react';
```

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './components/Home';
import Theory from './components/Theory';

function App() {
  const [progress, setProgress] = useState(() => {
    const saved = localStorage.getItem('progress');
    return saved ? JSON.parse(saved) : {};
  });

  useEffect(() => {
    localStorage.setItem('progress', JSON.stringify(progress));
  }, [progress]);

  const updateProgress = (vulnerabilityId, status) => {
    setProgress(prev => ({
      ...prev,
      [vulnerabilityId]: {
        ...prev[vulnerabilityId],
        ...status,
        lastUpdated: new Date().toISOString()
      }
    }));
  };

  const vulnerabilities = [
    {
      id: 'sql-injection',
      title: 'SQL Injection',
      description: 'Уразливість, пов’язана з виконанням несанкціонованих SQL-
```

```
запитів.',
        port: 3001
      },
      {
        id: 'xss',
        title: 'Cross-Site Scripting (XSS)',
        description: 'Вставка шкідливого JavaScript-коду у вебсторінки.',
        port: 3002
      },
      // ... інші вразливості
    ];

    return (
      <Router>
        <div className="app">
          <Routes>
            <Route path="/" element={<Home vulnerabilities={vulnerabilities} progress={progress} />} />
            <Route path="/theory/:id" element={<Theory vulnerabilities={vulnerabilities} updateProgress={updateProgress} />} />
          </Routes>
        </div>
      </Router>
    );
  }

  export default App;
```

Компонент Home.js

На головній сторінці відображається список карток із вразливостями, кожна з яких має статус (не розпочато, в процесі, завершено).

```
import React from 'react';
import { Link } from 'react-router-dom';
import './Home.css';

function Home({ vulnerabilities, progress }) {
 const getStatus = (id) => {
  const p = progress[id];
  if (!p) return 'not-started';
  if (p.completed) return 'completed';
  if (p.started) return 'in-progress';
  return 'not-started';
 };

 return (
  <div className="home-container">
   <h1>Платформа навчання кібербезпеці</h1>
   <div className="card-grid">
    {vulnerabilities.map(v => (
     <div key={v.id} className={`vulnerability-card ${getStatus(v.id)}`}>
      <h3>{v.title}</h3>
      <p>{v.description}</p>
      <div className="actions">
       <Link to={`/theory/${v.id}`} className="btn btn-theory">Теорія</Link>
       <a href={`http://localhost:${v.port}`} className="btn btn-practice"
target="_blank">Практика</a>
      </div>
      <div className="status">
       Статус: <strong>{getStatus(v.id)}</strong>
      </div>
     </div>
```

```
        ))}
      </div>
    </div>
  );
}

export default Home;
```

Компонент Theory.js

Сторінка з теоретичним поясненням вибраної вразливості.

```
import React, { useEffect } from 'react';
import { useParams } from 'react-router-dom';
import './Theory.css';

function Theory({ vulnerabilities, updateProgress }) {
  const { id } = useParams();
  const vuln = vulnerabilities.find(v => v.id === id);

  useEffect(() => {
    updateProgress(id, { started: true });
  }, [id]);

  if (!vuln) return <p>Вразливість не знайдена</p>;

  return (
    <div className="theory-container">
      <h2>{vuln.title}</h2>
      <p><strong>Опис:</strong> {vuln.description}</p>
      <p><strong>Ціль атаки:</strong> отримаємо несанкціонований доступ, виконаємо код тощо.</p>
      <p><strong>Приклад Payload:</strong></p>
```

```
            <pre>{vuln.id === 'sql-injection' && "admin' OR '1'='1"}{vuln.id === 'xss' &&
"<script>alert('XSS')</script>"}</pre>
          </div>
        );
      }

      export default Theory;
      Стилі
      /* App.css */
      body {
        margin: 0;
        font-family: Arial, sans-serif;
        background-color: #f9f9f9;
      }

      .home-container {
        padding: 2rem;
      }

      .card-grid {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
        gap: 2rem;
      }

      .vulnerability-card {
        background-color: #fff;
        border-radius: 8px;
        box-shadow: 0 2px 6px rgba(0,0,0,0.1);
        padding: 1rem;
```

```
  transition: transform 0.2s;
}

.vulnerability-card:hover {
  transform: translateY(-5px);
}

.actions {
  margin-top: 1rem;
  display: flex;
  justify-content: space-between;
}

.btn {
  padding: 0.5rem 1rem;
  text-decoration: none;
  color: white;
  border-radius: 4px;
}

.btn-theory {
  background-color: #3498db;
}

.btn-practice {
  background-color: #2ecc71;
}

.status {
  margin-top: 1rem;
```

```
  font-size: 0.9rem;
  color: #555;
}
```

4.4. Реалізація мікросервісів з уразливостями

Мікросервіси в системі створено для симуляції найпоширеніших веб-вразливостей, з якими стикаються розробники та фахівці з кібербезпеки. Кожна вразливість реалізована як окремий веб-додаток на основі Node.js + Express, ізольований у власному Docker-контейнері. Такий підхід дозволяє безпечно експериментувати з техніками атак, не ризикуючи пошкодити систему або вплинути на інші сервіси.

Архітектурна модель

Кожен мікросервіс має однакову базову структуру:

```
vulnerable-apps/
└── sql-injection/
    ├── server.js
    ├── Dockerfile
    ├── package.json
    └── views/
        ├── index.ejs
        └── success.ejs
```

Контейнери ізольовані один від одного, але мають доступ до спільної бази даних MySQL, де зберігаються демонстраційні таблиці для навчання.

SQL Injection (порт 3001)

Цей мікросервіс демонструє вразливість, що виникає при складанні SQL-запитів без параметризації.

Вразливий код:

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;
```

```
  const query = `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`;

  db.query(query, (err, results) => {
    if (err) return res.send('DB error');
    if (results.length > 0) {
      res.render('success', { username });
    } else {
      res.render('index', { error: 'Invalid credentials' });
    }
  });
});
```

Приклад експлойту:

admin' OR '1'='1

XSS (Cross-Site Scripting) (порт 3002)

Сервіс із вразливою системою коментарів. Дані не проходять фільтрацію, що дозволяє впровадити JavaScript-код.

Вразливий код:

```
app.post('/comment', async (req, res) => {
  const { content } = req.body;

  await db.promise().query('INSERT INTO comments (content) VALUES (?)', [content]);
  res.redirect('/');
});
```

В шаблоні index.ejs:

<%- comment.content %> <!-- ВРАЗЛИВО: неекранований HTML -->

Приклад payload:

<script>alert('XSS')</script>

Command Injection (порт 3003)

Сервіс виконує системну команду ping, підставляючи параметри з клієнта без перевірки.

Вразливий код:

```
app.post('/ping', (req, res) => {
  const { host } = req.body;

  const command = `ping -c 2 ${host}`;

  exec(command, (error, stdout) => {
    res.send(`<pre>${stdout}</pre>`);
  });
});
```

Приклад payload:

```
127.0.0.1 && ls /
```

File Upload Vulnerability (порт 3004)

Сервіс дозволяє завантажити файл, але не перевіряє MIME-тип або розширення.

Вразливий код:

```
const storage = multer.diskStorage({
  filename: (req, file, cb) => {
    cb(null, file.originalname); // без перевірки
  }
});
```

Можливість обходу перевірки:

- Завантаження .php.jpg файлів
- Потенційне виконання на сервері, якщо не налаштовані права

CSRF (Cross-Site Request Forgery) (порт 3005)

Веб-додаток змінює стан користувача (баланс), не перевіряючи походження запиту.

Вразливий код:

```
app.post('/transfer', (req, res) => {
  const { amount } = req.body;

  users[req.session.user].balance -= parseInt(amount);
  res.redirect('/');
});
```

Немає CSRF токена. Зовнішній сайт може підставити користувача на дію.

XXE (XML External Entity) (порт 3006)

Сервіс обробляє XML, дозволяючи вставку зовнішніх сутностей.

Вразливий код:

```
const xmlDoc = libxmljs.parseXml(req.body, {
  noent: true // ДОЗВОЛЯЄ завантаження зовнішніх сутностей
});
```

Приклад небезпечного XML:

```
<!DOCTYPE root [
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>&xxe;</root>
```

Dockerfile (для всіх мікросервісів)

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

Мережева взаємодія

Усі сервіси взаємодіють із базою даних db через мережу ctf-network, яку створює docker-compose. У кожному мікросервісі вказано:

```
environment:
  DB_HOST: db
  DB_USER: root
  DB_PASSWORD: root
  DB_NAME: ctf_db
```

4.5. Контейнеризація компонентів за допомогою Docker

Контейнеризація є фундаментальною основою архітектури навчальної платформи з кібербезпеки. Вона забезпечує ізольоване, відтворюване та контрольоване середовище для кожного сервісу, дозволяючи уникнути конфліктів залежностей та спростити розгортання. Усі компоненти — веб-інтерфейс, база даних та окремі мікросервіси з уразливостями — упаковані у Docker-контейнери. Оркестрація контейнерів здійснюється за допомогою утиліти Docker Compose, яка забезпечує одночасний запуск усіх сервісів відповідно до опису в конфігураційному файлі docker-compose.yml.

Завдяки використанню контейнерів кожен мікросервіс виконується ізольовано, має власний порт та чітко визначену зону відповідальності. Наприклад, сервіс SQL Injection доступний за портом 3001, XSS — на 3002, Command Injection — на 3003 і так далі. Усі сервіси об'єднані в одну віртуальну мережу ctf-network, що забезпечує внутрішню взаємодію між контейнерами та одночасно унеможливлює випадкове втручання ззовні.

Базу даних реалізовано як окремий сервіс db, що використовує офіційний образ MySQL 8.0. Під час старту автоматично виконується SQL-скрипт ініціалізації бази (init.sql), який монтується у контейнер за допомогою volume. Також до сервісу додано healthcheck, що перевіряє готовність бази до роботи, і лише після цього запускаються залежні від неї сервіси.

Фронтенд web зібрано з React-коду, який компілюється всередині контейнера. Порт 3000 відкрито для доступу через браузер. Для відстеження змін у розробницькому режимі застосовано змінні середовища

(CHOKIDAR_USEPOLLING=true), що дає змогу оновлювати інтерфейс без перезапуску контейнера.

Нижче наведено повний приклад конфігурації docker-compose.yml, який використовується у проєкті:

```
version: '3.8'

services:
  db:
    image: mysql:8.0
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ctf_db
    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
      - mysql_data:/var/lib/mysql
    networks:
      - ctf-network
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  web:
    build:
      context: ./web
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
```

```
    volumes:
      - ./web:/app
    environment:
      - NODE_ENV=development
      - CHOKIDAR_USEPOLLING=true
    depends_on:
      db:
        condition: service_healthy
    networks:
      - ctf-network

  sql-injection:
    build:
      context: ./vulnerable-apps/sql-injection
    ports:
      - "3001:3000"
    environment:
      DB_HOST: db
      DB_USER: root
      DB_PASSWORD: root
      DB_NAME: ctf_db
    depends_on:
      db:
        condition: service_healthy
    networks:
      - ctf-network

  xss:
    build:
      context: ./vulnerable-apps/xss
```

```
    ports:
      - "3002:3000"
    environment:
      DB_HOST: db
      DB_USER: root
      DB_PASSWORD: root
      DB_NAME: ctf_db
    depends_on:
      db:
        condition: service_healthy
    networks:
      - ctf-network

  command-injection:
    build:
      context: ./vulnerable-apps/command-injection
    ports:
      - "3003:3000"
    networks:
      - ctf-network

  file-upload:
    build:
      context: ./vulnerable-apps/file-upload
    ports:
      - "3004:3000"
    volumes:
      - ./vulnerable-apps/file-upload/uploads:/app/uploads
    networks:
      - ctf-network
```

```
  csrf:
    build:
      context: ./vulnerable-apps/csrf
    ports:
      - "3005:3000"
    networks:
      - ctf-network

  xxe:
    build:
      context: ./vulnerable-apps/xxe
    ports:
      - "3006:3000"
    networks:
      - ctf-network

volumes:
  mysql_data:

networks:
  ctf-network:
    driver: bridge
```

Контейнеризація гарантує, що кожен студент чи користувач отримає ідентичне середовище, яке не залежить від ОС чи локальної конфігурації. Сервіси не мають доступу до хост-системи або один до одного без необхідності, а також використовують лише ті ресурси, які їм дозволено через механізми Docker. У майбутньому ця система може бути легко масштабована, розгорнута в хмарі або інтегрована з навчальними платформами, що робить її універсальним інструментом для практичного вивчення кібербезпеки.

4.6. Інструкція для користувача

У цьому розділі подано покрокову інструкцію з користування веб-платформою «CTF Training Platform». Інтерфейс реалізовано таким чином, щоб забезпечити зручну навігацію, доступ до навчального матеріалу з кібербезпеки та виконання практичних завдань у захищеному середовищі.

1. Головна сторінка платформи (див. рис. 4.1)

   Після запуску платформи відкривається головна сторінка з вітальним повідомленням та коротким описом призначення системи. Нижче розміщено інтерактивні картки вразливостей, зокрема SQL Injection, XSS, Command Injection, CSRF, File Upload, XXE тощо. Кожна картка містить короткий опис, дві кнопки для переходу до теоретичних матеріалів та практичного завдання, а також індикатор статусу проходження: «Не розпочато», «У процесі» або «Завершено».

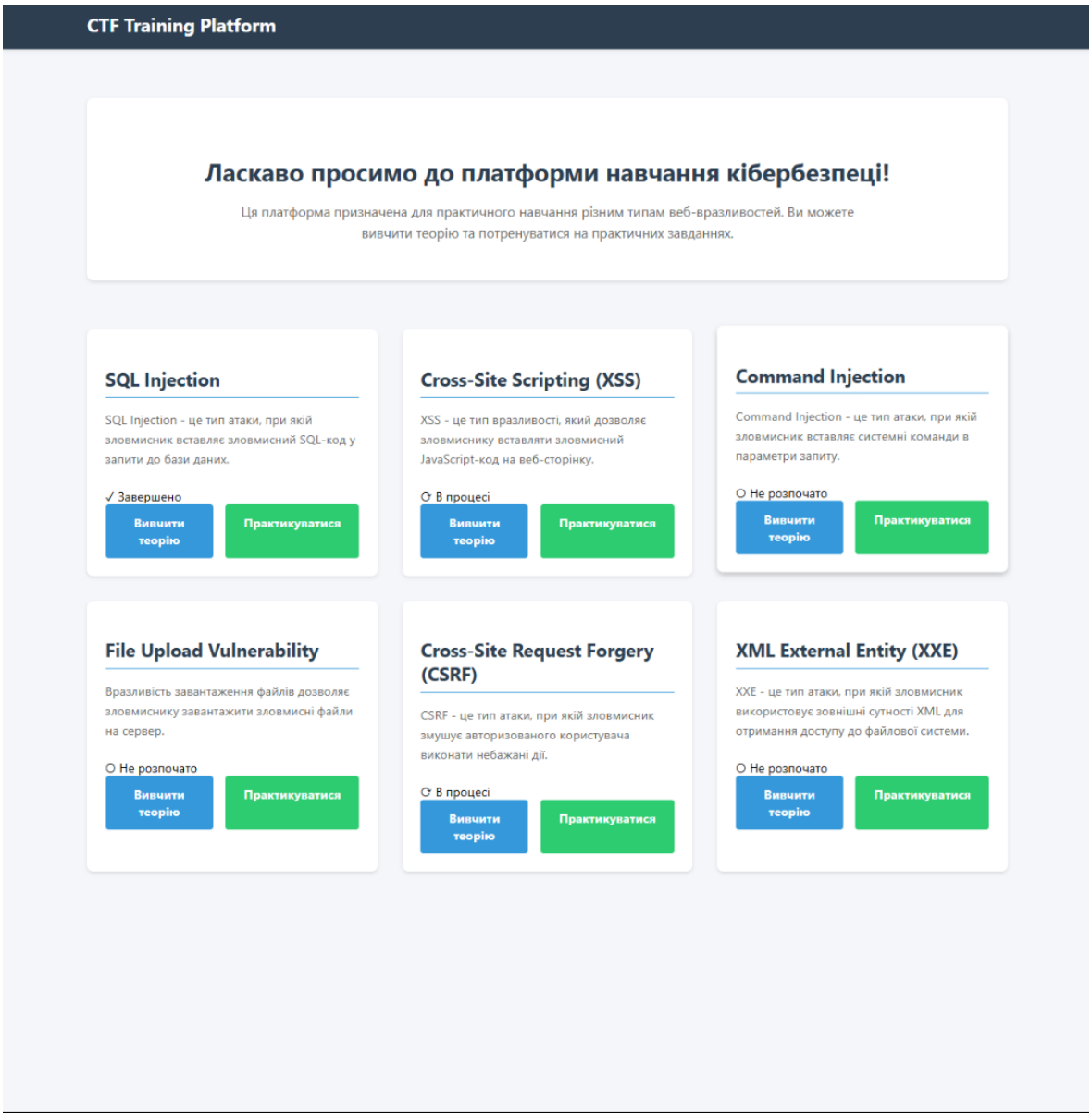

Рисунок 4.1 – Головна сторінка з переліком вразливостей

2. Теоретичні матеріали (див. рис. 4.2)

   Натиснувши кнопку «Вивчити теорію» на будь-якій картці, користувач переходить до сторінки з теоретичним викладом вибраної теми. Тут подано повний текстовий опис суті вразливості, історію її виникнення, типові приклади коду, схеми атак та пояснення методів захисту. Кожна сторінка оформлена в тематичному кольорі з виділеними блоками «Теорія», «Захист», «Приклади експлойтів» і «Перевір себе».

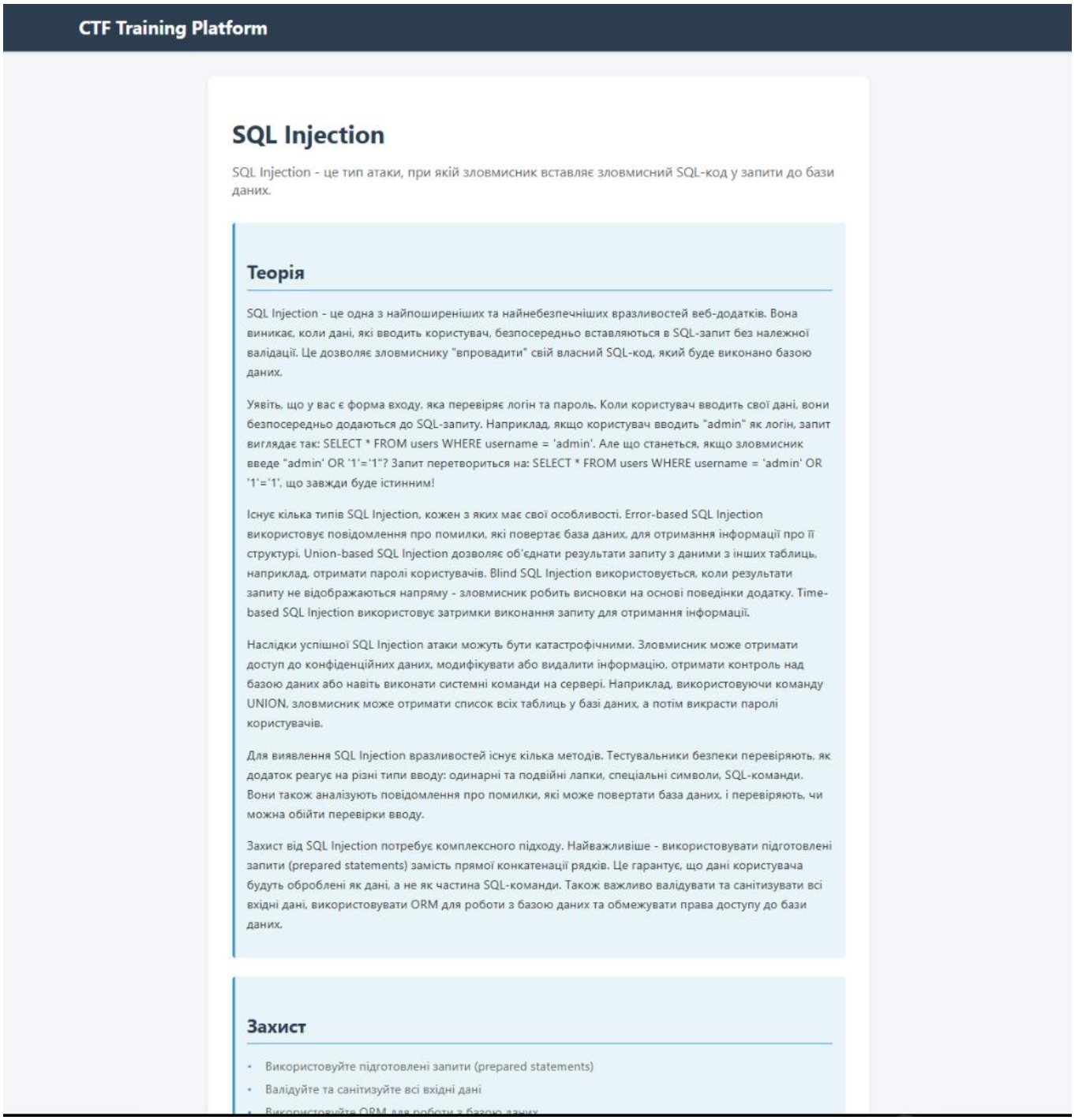

Рисунок 4.2 – Сторінка теорії: SQL Injection

3. Перевірка знань (див. рис. 4.3)

   У нижній частині теоретичного розділу міститься блок для самоперевірки. Користувачеві пропонується ознайомитись із фрагментом коду та визначити, у чому полягає його вразливість. Після введення відповіді можна перевірити правильність свого варіанту, що сприяє активному засвоєнню матеріалу.

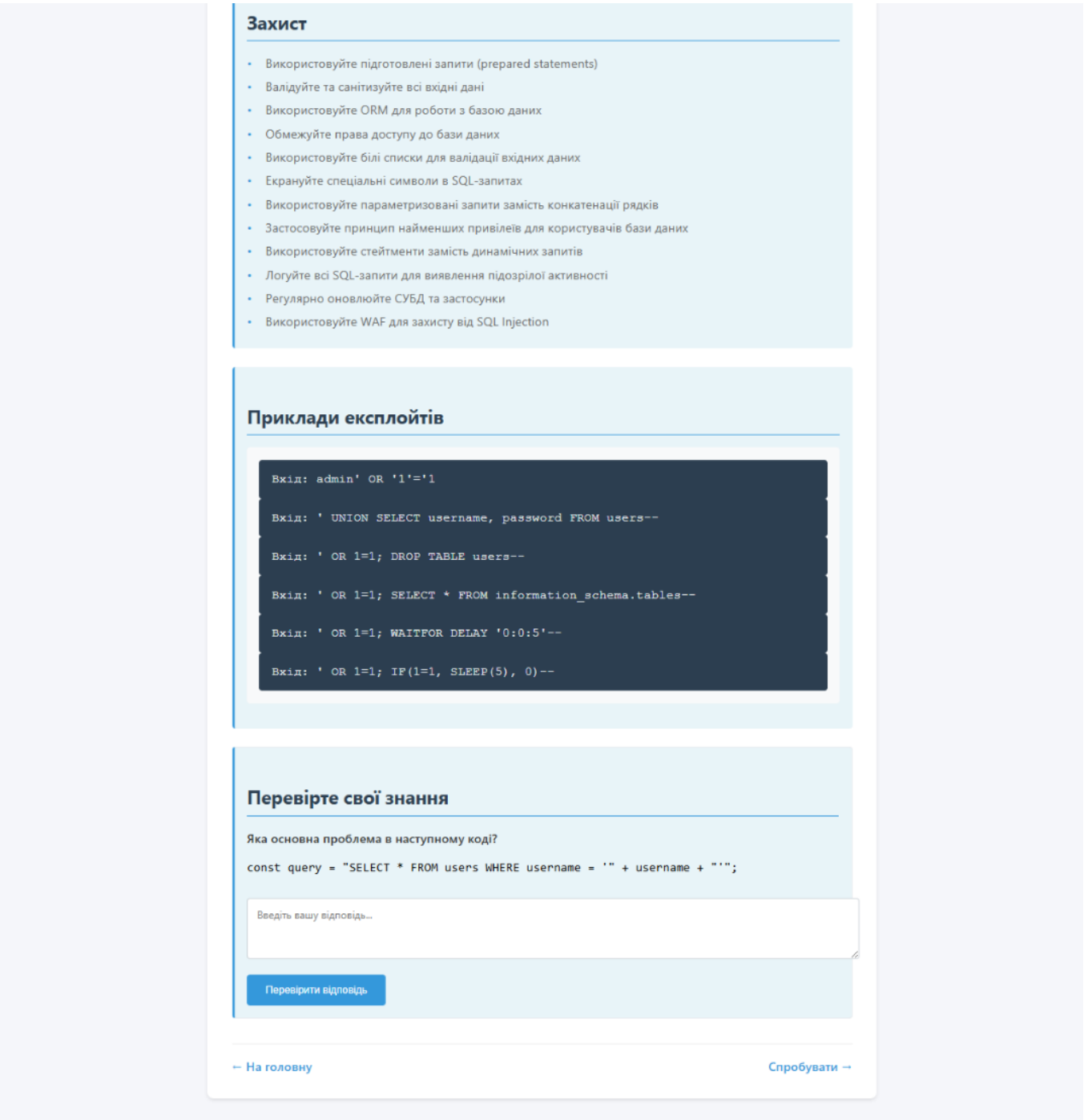

Рисунок 4.3 – Перевірка знань на сторінці теорії

4. Практичне завдання (див. рис. 4.4)

   Після натискання кнопки «Практикуватися» на головній сторінці відкривається окреме вікно з вразливим вебдодатком. Наприклад, у випадку з SQL Injection користувач бачить форму входу, яка є вразливою до SQL-ін’єкцій. Метою завдання є обхід аутентифікації шляхом введення спеціального payload (наприклад: admin' OR '1'='1).

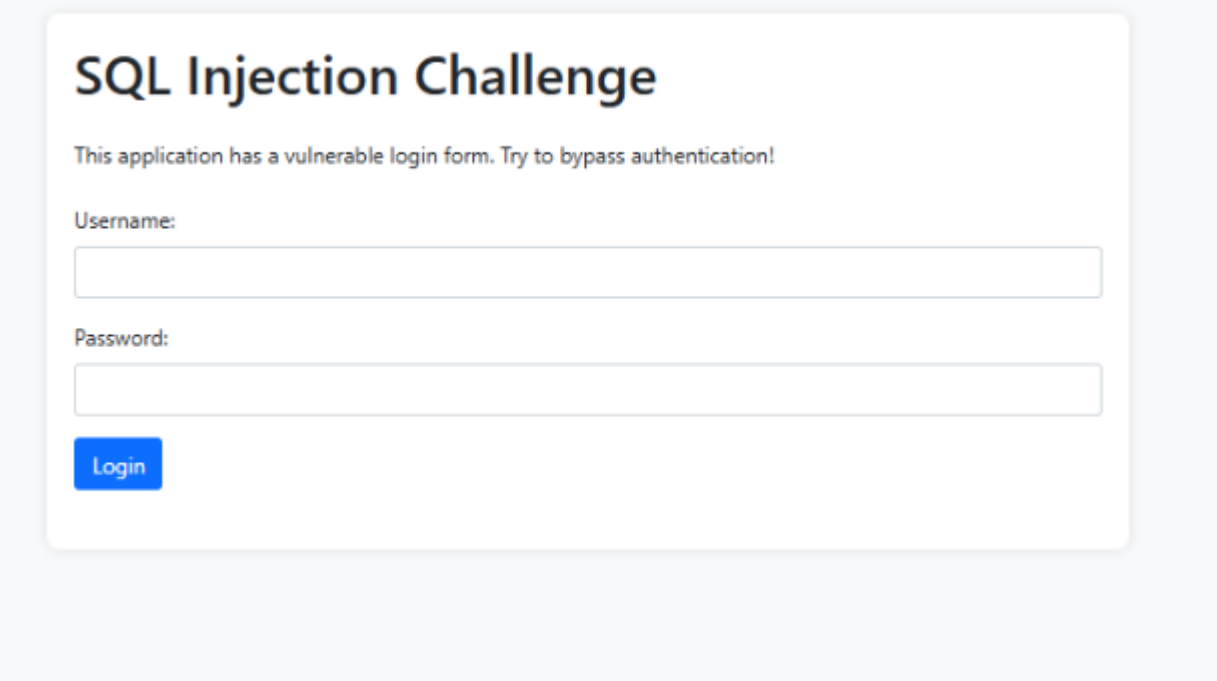

Рисунок 4.4 – Вразлива форма логіну: SQL Injection

5. Виконання експлойту (див. рис. 4.5)

   Користувач вводить шкідливе значення у поле «Username» та будь-яке значення у поле «Password», після чого натискає «Login». Якщо експлойт успішний, платформа повідомляє про успішний вхід і демонструє результат.

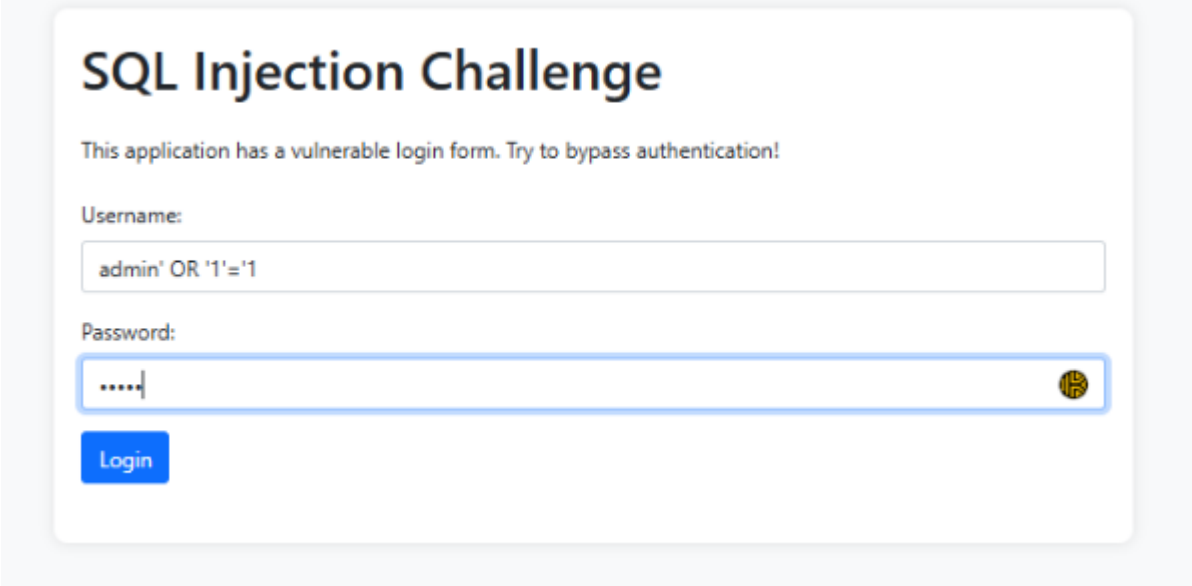

Рисунок 4.5 – Демонстрація результату успішного SQL-експлойту

6. Зворотний зв’язок (див. рис. 4.6)

   Після проходження теоретичного та практичного модулів статус вразливості на головній сторінці змінюється на «Завершено». Таким чином, користувач може відстежувати свій прогрес по кожному з блоків та повертатися до тих, які ще не пройдено або потребують повторного вивчення.

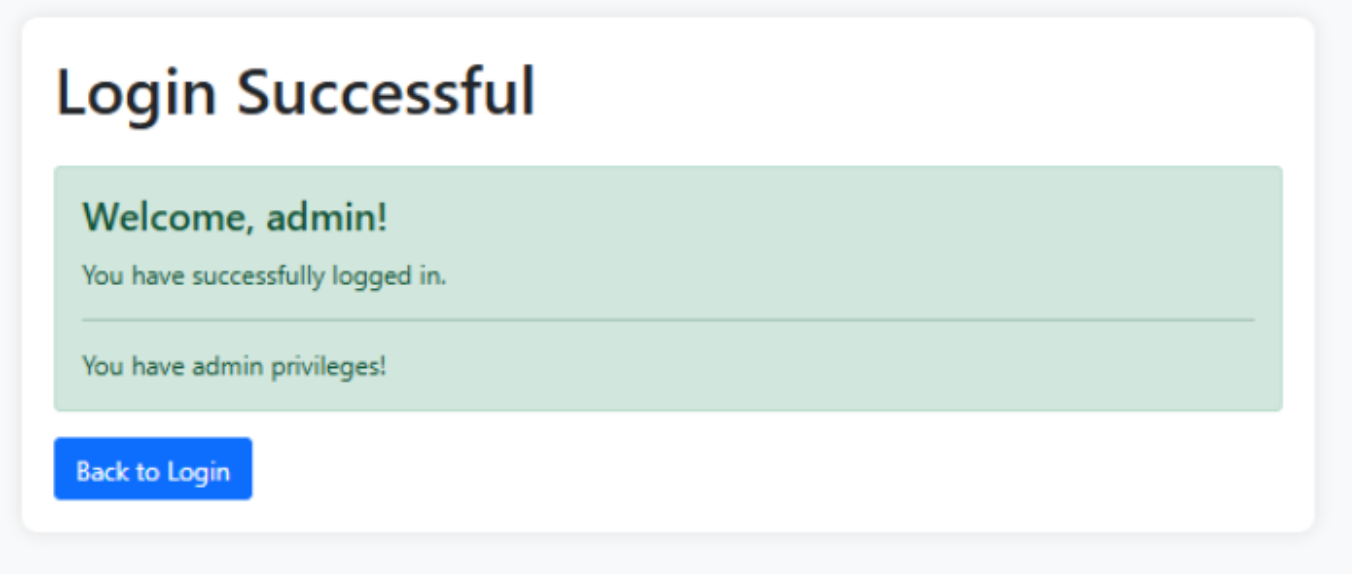

Рисунок 4.6 – Успішне виконання техніки

Інтерфейс платформи розроблений у мінімалістичному стилі, з акцентом на доступність контенту, адаптивність до різних розмірів екранів та простоту використання. Усі дії користувача зберігаються у локальному сховищі браузера, що дозволяє повертатися до навчання у зручний час і з того місця, де було завершено роботу.

ВИСНОВКИ

У дипломній роботі було розроблено веб-платформу для навчання методам кібербезпеки, яка забезпечує інтерактивне вивчення типових веб-вразливостей у захищеному середовищі. Реалізована система поєднує теоретичні матеріали з практичними завданнями, що дозволяє студентам та фахівцям з інформаційної безпеки набувати прикладних навичок в умовах, наближених до реальних.

У ході виконання роботи було досягнуто наступних результатів:

- Проаналізовано основні типи веб-вразливостей, їх класифікацію, способи реалізації атак і методи протидії.
- Запроектовано архітектуру системи на основі мікросервісного підходу, з урахуванням принципів ізоляції, масштабованості та безпеки.
- Реалізовано низку окремих мікросервісів, що імітують поширені типи вразливостей: SQL Injection, XSS, Command Injection, CSRF, XXE та інші.
- Створено сучасний веб-інтерфейс на основі бібліотеки React, який дозволяє користувачеві взаємодіяти з теоретичним і практичним контентом у зручній формі.
- Забезпечено повну контейнеризацію системи за допомогою Docker та автоматизовано розгортання всіх компонентів через Docker Compose.
- Реалізовано систему збереження прогресу навчання користувача, а також блоки перевірки знань після кожного теоретичного модуля.
- Створено детальну інструкцію користувача для забезпечення зручності при використанні системи навіть без попередньої технічної підготовки.

Розроблена платформа має значне практичне значення, оскільки дозволяє безпечно моделювати поведінку атакуючої сторони без ризику для продуктивних систем. Вона може бути ефективно використана у закладах вищої освіти, на спеціалізованих курсах з кібербезпеки або як інструмент самонавчання. Завдяки модульності та відкритій архітектурі, платформа легко розширюється новими вразливостями та сценаріями.

Результати виконаної роботи демонструють доцільність використання сучасних підходів до контейнеризації, веброзробки та моделювання кіберзагроз для побудови інтерактивного середовища навчання. Система є надійною, масштабованою, адаптованою до подальшого розвитку і відповідає актуальним вимогам у сфері освітніх інструментів з інформаційної безпеки.

СПИСОК ЛІТЕРАТУРИ

1. OWASP Foundation. OWASP Top Ten Web Application Security Risks – 2021. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/
2. Mozilla Developer Network. Web Security Guidelines. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Security
3. Acunetix. SQL Injection Overview and Prevention. [Електронний ресурс]. – Режим доступу: https://www.acunetix.com/websitesecurity/sql-injection/
4. PortSwigger. Cross-site Scripting (XSS). [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/cross-site-scripting
5. Docker Documentation. Docker Security Best Practices. [Електронний ресурс]. – Режим доступу: https://docs.docker.com/engine/security/security/
6. National Institute of Standards and Technology (NIST). Cybersecurity Framework. [Електронний ресурс]. – Режим доступу: https://www.nist.gov/cyberframework
7. OWASP Cheat Sheet Series. Security Logging and Monitoring. [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
8. Google Cloud. Security Best Practices for Web Applications. [Електронний ресурс]. – Режим доступу: https://cloud.google.com/architecture/security-best-practices-for-web-applications
9. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2024. – 67 с. – 1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

docker-compose.yml

```
version: '3'

services:
  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ctf_db
    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  web:
    build: ./web
    ports:
      - "3000:3000"
    volumes:
      - ./web:/app
      - /app/node_modules
    environment:
      - NODE_ENV=development
    depends_on:
      db:
        condition: service_healthy

  sql-injection:
    build: ./vulnerable-apps/sql-injection
    ports:
      - "3001:3000"
    environment:
```

```
    DB_HOST: db
    DB_USER: root
    DB_PASSWORD: root
    DB_NAME: ctf_db
  depends_on:
    db:
      condition: service_healthy

xss:
  build: ./vulnerable-apps/xss
  ports:
    - "3002:3000"
  environment:
    DB_HOST: db
    DB_USER: root
    DB_PASSWORD: root
    DB_NAME: ctf_db
  depends_on:
    db:
      condition: service_healthy

command-injection:
  build: ./vulnerable-apps/command-injection
  ports:
    - "3003:3000"
  environment:
    DB_HOST: db
    DB_USER: root
    DB_PASSWORD: root
    DB_NAME: ctf_db
  depends_on:
    db:
      condition: service_healthy

file-upload:
  build: ./vulnerable-apps/file-upload
  ports:
    - "3004:3000"
```

```
    volumes:
      - ./vulnerable-apps/file-upload/uploads:/app/uploads
    depends_on:
      db:
        condition: service_healthy

  csrf:
    build: ./vulnerable-apps/csrf
    ports:
      - "3005:3000"
    depends_on:
      db:
        condition: service_healthy

  xxe:
    build: ./vulnerable-apps/xxe
    ports:
      - "3006:3000"
    depends_on:
      db:
        condition: service_healthy

volumes:
  mysql_data:

db/init.sql
CREATE DATABASE IF NOT EXISTS ctf_db;
USE ctf_db;

-- Users table for SQL injection challenge
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    is_admin BOOLEAN DEFAULT FALSE
);

-- Insert default admin user
```

```
INSERT IGNORE INTO users (username, password, is_admin)
VALUES ('admin', 'admin123', TRUE);

-- Comments table for XSS challenge
CREATE TABLE IF NOT EXISTS comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

web/package.json

```
{
  "name": "ctf-training-platform",
  "version": "0.1.0",
  "private": true,
  "description": "Web interface for CTF training platform",
  "main": "src/index.js",
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.22.3",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
```

```
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

web/Dockerfile

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

web/src/App.js

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from './components/Home';
import Theory from './components/Theory';
import './App.css';

function App() {
  const [progress, setProgress] = useState(() => {
    const savedProgress = localStorage.getItem('progress');
    return savedProgress ? JSON.parse(savedProgress) : {};
  });

  useEffect(() => {
```

```
    localStorage.setItem('progress', JSON.stringify(progress));
  }, [progress]);

  const updateProgress = (vulnerabilityId, status) => {
    setProgress(prev => ({
      ...prev,
      [vulnerabilityId]: {
        ...prev[vulnerabilityId],
        ...status,
        lastUpdated: new Date().toISOString()
      }
    }));
  };

  const vulnerabilities = [
    {
      id: 'sql-injection',
      title: 'SQL Injection',
      description: 'SQL Injection - це тип атаки, при якій зловмисник вставляє зловмисний SQL-код у запити до бази даних.',
      port: 3001,
      hints: [
        'Спробуйте використати одинарні лапки в полі вводу',
        'Використайте оператор UNION для об\'єднання запитів',
        'Спробуйте коментувати частину запиту за допомогою --'
      ]
    },
    {
      id: 'xss',
      title: 'Cross-Site Scripting (XSS)',
      description: 'XSS - це тип вразливості, який дозволяє зловмиснику вставляти зловмисний JavaScript-код на веб-сторінку.',
      port: 3002,
      hints: [
        'Спробуйте вставити HTML-теги в коментар',
        'Використайте JavaScript-події (onclick, onload)',
        'Спробуйте використати тег <script>'
      ]
```

```
  },
  {
    id: 'command-injection',
    title: 'Command Injection',
    description: 'Command Injection - це тип атаки, при якій зловмисник вставляє системні команди в параметри запиту.',
    port: 3003,
    hints: [
      'Спробуйте використати символ ; для розділення команд',
      'Використайте | для перенаправлення виводу',
      'Спробуйте використати && для виконання послідовності команд'
    ]
  },
  {
    id: 'file-upload',
    title: 'File Upload Vulnerability',
    description: 'Вразливість завантаження файлів дозволяє зловмиснику завантажити зловмисні файли на сервер.',
    port: 3004,
    hints: [
      'Спробуйте завантажити PHP-файл',
      'Використайте обхід перевірки розширення файлу',
      'Спробуйте завантажити файл з подвійним розширенням'
    ]
  },
  {
    id: 'csrf',
    title: 'Cross-Site Request Forgery (CSRF)',
    description: 'CSRF - це атака, при якій зловмисник змушує користувача виконати небажані дії на веб-сайті.',
    port: 3005,
    hints: [
      'Створіть зловмисну HTML-сторінку з формою',
      'Використайте автоматичну відправку форми',
      'Спробуйте змінити баланс користувача'
    ]
  },
  {
```

```
        id: 'xxe',
        title: 'XML External Entity (XXE)',
        description: 'XXE - це вразливість, яка дозволяє зловмиснику отримати доступ до локальних файлів через XML.',
        port: 3006,
        hints: [
          'Використайте зовнішні XML сутності',
          'Спробуйте прочитати файл /etc/passwd',
          'Створіть XML з DOCTYPE декларацією'
        ]
      }
    ];

    return (
      <Router>
        <div className="app">
          <nav className="navbar">
            <div className="container">
              <h1>Платформа навчання кібербезпеки</h1>
            </div>
          </nav>
          <main className="main-content">
            <div className="container">
              <Routes>
                <Route path="/" element={<Home vulnerabilities={vulnerabilities} progress={progress} />} />
                <Route path="/theory/:id" element={<Theory vulnerabilities={vulnerabilities} updateProgress={updateProgress} />} />
              </Routes>
            </div>
          </main>
        </div>
      </Router>
    );
  }

  export default App;
```

```
web/src/components/Home.js
import React from 'react';
import { Link } from 'react-router-dom';
import './Home.css';

function Home({ vulnerabilities, progress }) {
 const getProgressStatus = (vulnerabilityId) => {
  const vulnProgress = progress[vulnerabilityId];
  if (!vulnProgress) return 'not-started';
  if (vulnProgress.completed) return 'completed';
  if (vulnProgress.started) return 'in-progress';
  return 'not-started';
 };

 return (
  <div className="home">
   <section className="intro-section">
    <h2>Ласкаво просимо до платформи навчання кібербезпеці!</h2>
    <p>
     Ця платформа призначена для практичного навчання різним типам веб-вразливостей.
     Ви можете вивчити теорію та потренуватися на практичних завданнях.
    </p>
   </section>

   <div className="vulnerabilities-grid">
    {vulnerabilities.map((vuln) => {
     const status = getProgressStatus(vuln.id);
     return (
      <div key={vuln.id} className={`vulnerability-card ${status}`}>
       <h3>{vuln.title}</h3>
       <p>{vuln.description}</p>
       <div className="progress-indicator">
        {status === 'completed' && <span className="status completed">✓ Завершено</span>}
        {status === 'in-progress' && <span className="status in-progress">⟳ В процесі</span>}
        {status === 'not-started' && <span className="status not-started">○ Не
розпочато</span>}
       </div>
       <div className="card-actions">
```

```
            <Link to={`/theory/${vuln.id}`} className="theory-button">
              Вивчити теорію
            </Link>
            <a
              href={`http://localhost:${vuln.port}`}
              target="_blank"
              rel="noopener noreferrer"
              className="practice-button"
            >
              Практикуватися
            </a>
          </div>
        </div>
      );
    })}
  </div>
 </div>
);
}

export default Home;
```

web/src/App.css

```
.app {
  min-height: 100vh;
  background-color: #f5f6fa;
}

.navbar {
  background-color: #2c3e50;
  color: white;
  padding: 1rem 0;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.navbar .container {
  display: flex;
  justify-content: space-between;
```

```
  align-items: center;
}

.navbar h1 {
  margin: 0;
  font-size: 1.5rem;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 1rem;
}

.main-content {
  padding: 2rem 0;
}

.intro-section {
  text-align: center;
  padding: 3rem 0;
  background-color: white;
  border-radius: 8px;
  margin: 2rem 0;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.intro-section h2 {
  color: #2c3e50;
  margin-bottom: 1rem;
  font-size: 2rem;
}

.intro-section p {
  color: #666;
  font-size: 1.1rem;
  line-height: 1.6;
  max-width: 800px;
```

```
  margin: 0 auto;
}

.vulnerabilities-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 2rem;
  padding: 2rem 0;
}

.vulnerability-card {
  background-color: white;
  border-radius: 8px;
  padding: 1.5rem;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  transition: transform 0.2s, box-shadow 0.2s;
}

.vulnerability-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

.vulnerability-card h3 {
  color: #2c3e50;
  margin-bottom: 1rem;
  font-size: 1.5rem;
  border-bottom: 2px solid #3498db;
  padding-bottom: 0.5rem;
}

.vulnerability-card p {
  color: #666;
  margin-bottom: 1.5rem;
  line-height: 1.6;
}

@media (max-width: 768px) {
```

```
  .vulnerabilities-grid {
    grid-template-columns: 1fr;
  }

  .intro-section {
    padding: 2rem 1rem;
  }

  .intro-section h2 {
    font-size: 1.5rem;
  }

  .intro-section p {
    font-size: 1rem;
  }

  .navbar h1 {
    font-size: 1.2rem;
  }
}
```

vulnerable-apps/sql-injection/package.json

```
{
  "name": "sql-injection",
  "version": "1.0.0",
  "description": "Vulnerable application demonstrating SQL injection attacks",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "mysql2": "^3.6.0",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/sql-injection/Dockerfile

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

vulnerable-apps/sql-injection/server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql2');
const path = require('path');

const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Database connection
const db = mysql.createConnection({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'root',
  database: process.env.DB_NAME || 'ctf_db'
});

// Initialize database
db.query(`
  CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
```

```
    is_admin BOOLEAN DEFAULT FALSE
  )
`);

// Insert default admin user
db.query(`
  INSERT IGNORE INTO users (username, password, is_admin)
  VALUES ('admin', 'admin123', TRUE)
`);

// Routes
app.get('/', (req, res) => {
  res.render('index', {
    title: 'SQL Injection Challenge',
    description: 'This application has a vulnerable login form. Try to bypass authentication!',
    error: null
  });
});

app.post('/login', (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) {
    return res.render('index', {
      title: 'SQL Injection Challenge',
      description: 'This application has a vulnerable login form. Try to bypass authentication!',
      error: 'Username and password are required'
    });
  }

  // Vulnerable to SQL injection
  const query = `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`;

  db.query(query, (error, results) => {
    if (error) {
      return res.render('index', {
        title: 'SQL Injection Challenge',
```

```
            description: 'This application has a vulnerable login form. Try to bypass authentication!',
            error: 'Database error occurred'
          });
        }

        if (results.length > 0) {
          const user = results[0];
          res.render('success', {
            title: 'Login Successful',
            username: user.username,
            isAdmin: user.is_admin
          });
        } else {
          res.render('index', {
            title: 'SQL Injection Challenge',
            description: 'This application has a vulnerable login form. Try to bypass authentication!',
            error: 'Invalid username or password'
          });
        }
      });
    });

    // Start server
    app.listen(port, () => {
      console.log(`SQL injection app listening at http://localhost:${port}`);
    });

    vulnerable-apps/sql-injection/views/index.ejs
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title><%= title %></title>
        <link                    href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
        <style>
            body {
```

```
            padding: 20px;
            background-color: #f8f9fa;
        }
        .container {
            max-width: 800px;
            margin: 0 auto;
            background-color: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0,0,0,0.1);
        }
    </style>
</head>
<body>
    <div class="container">
        <h1 class="mb-4"><%= title %></h1>
        <p class="mb-4"><%= description %></p>

        <% if (error) { %>
            <div class="alert alert-danger" role="alert">
                <%= error %>
            </div>
        <% } %>

        <form action="/login" method="POST" class="mb-4">
            <div class="mb-3">
                <label for="username" class="form-label">Username:</label>
                <input type="text" class="form-control" id="username" name="username" required>
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Password:</label>
                <input type="password" class="form-control" id="password" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary">Login</button>
        </form>
    </div>
</body>
</html>
```

vulnerable-apps/xss/package.json

```
{
  "name": "xss",
  "version": "1.0.0",
  "description": "Vulnerable application demonstrating XSS attacks",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "mysql2": "^3.6.0",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/xss/server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql2');
const path = require('path');

const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Database connection
const db = mysql.createConnection({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'root',
```

```
  database: process.env.DB_NAME || 'ctf_db'
});

// Initialize database
db.query(`
  CREATE TABLE IF NOT EXISTS comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  )
`);

// Routes
app.get('/', async (req, res) => {
  try {
    const [comments] = await db.promise().query('SELECT * FROM comments ORDER BY created_at DESC');
    res.render('index', {
      title: 'XSS Challenge',
      description: 'This application has a vulnerable comment system. Try to inject malicious JavaScript!',
      comments
    });
  } catch (error) {
    res.status(500).send('Database error');
  }
});

app.post('/comment', async (req, res) => {
  const { content } = req.body;

  if (!content) {
    return res.status(400).json({ error: 'Comment content is required' });
  }

  try {
    // Vulnerable to XSS - no sanitization
    await db.promise().query('INSERT INTO comments (content) VALUES (?)', [content]);
    res.redirect('/');
```

```
  } catch (error) {
    res.status(500).json({ error: 'Failed to save comment' });
  }
});

// Start server
app.listen(port, () => {
  console.log(`XSS app listening at http://localhost:${port}`);
});
```

vulnerable-apps/xss/views/index.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= title %></title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        body {
            padding: 20px;
            background-color: #f8f9fa;
        }
        .container {
            max-width: 800px;
            margin: 0 auto;
            background-color: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0,0,0,0.1);
        }
        .comment {
            margin-bottom: 15px;
            padding: 15px;
            background-color: #f8f9fa;
            border-radius: 5px;
        }
```

```
            .comment-time {
                font-size: 0.8rem;
                color: #666;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <h1 class="mb-4"><%= title %></h1>
            <p class="mb-4"><%= description %></p>

            <form action="/comment" method="POST" class="mb-4">
                <div class="mb-3">
                    <label for="content" class="form-label">Add a comment:</label>
                    <textarea class="form-control" id="content" name="content" rows="3" required></textarea>
                </div>
                <button type="submit" class="btn btn-primary">Post Comment</button>
            </form>

            <div class="comments">
                <h3>Comments</h3>
                <% comments.forEach(comment => { %>
                    <div class="comment">
                        <div class="comment-content"><%- comment.content %></div>
                        <div class="comment-time"><%= new Date(comment.created_at).toLocaleString() %></div>
                    </div>
                <% }); %>
            </div>
        </div>
    </body>
    </html>

    vulnerable-apps/command-injection/package.json
    {
      "name": "command-injection",
      "version": "1.0.0",
```

```
  "description": "Vulnerable application demonstrating command injection attacks",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "mysql2": "^3.6.0",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/command-injection/server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql2');
const { exec } = require('child_process');
const path = require('path');

const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Database connection
const db = mysql.createConnection({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'root',
  database: process.env.DB_NAME || 'ctf_db'
});

// Routes
```

```
app.get('/', (req, res) => {
  res.render('index', {
    title: 'Command Injection Challenge',
    description: 'This application allows you to ping hosts. Try to find a way to execute arbitrary commands!'
  });
});

app.post('/ping', (req, res) => {
  const { host } = req.body;

  if (!host) {
    return res.status(400).json({ error: 'Host is required' });
  }

  // Vulnerable to command injection
  const command = `ping -c 4 ${host}`;

  exec(command, (error, stdout, stderr) => {
    if (error) {
      return res.status(500).json({ error: error.message });
    }

    res.json({
      output: stdout,
      error: stderr
    });
  });
});

// Start server
app.listen(port, () => {
  console.log(`Command injection app listening at http://localhost:${port}`);
});
```

vulnerable-apps/file-upload/package.json

```
{
  "name": "file-upload",
```

```
  "version": "1.0.0",
  "description": "Vulnerable file upload application",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.0",
    "multer": "^1.4.5",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/file-upload/server.js

```
const express = require('express');
const multer = require('multer');
const path = require('path');
const fs = require('fs');

const app = express();
const port = 3000;

// Налаштування multer для завантаження файлів
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    const uploadDir = path.join(__dirname, 'uploads');
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir);
    }
    cb(null, uploadDir);
  },
  filename: function (req, file, cb) {
    // Вразливість: не перевіряємо розширення файлу
    cb(null, file.originalname);
  }
});

const upload = multer({ storage: storage });
```

```
// Налаштування EJS
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Статичні файли
app.use(express.static(path.join(__dirname, 'public')));
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

// Головна сторінка
app.get('/', (req, res) => {
  res.render('index', { error: null });
});

// Обробка завантаження файлу
app.post('/upload', upload.single('file'), (req, res) => {
  if (!req.file) {
    return res.render('index', { error: 'Будь ласка, виберіть файл для завантаження' });
  }

  // Вразливість: не перевіряємо тип файлу
  res.render('success', {
    filename: req.file.originalname,
    path: `/uploads/${req.file.originalname}`
  });
});

app.listen(port, () => {
  console.log(`File Upload Vulnerability Demo running at http://localhost:${port}`);
});

vulnerable-apps/csrf/package.json
{
  "name": "csrf",
  "version": "1.0.0",
  "description": "Vulnerable CSRF application",
  "main": "server.js",
  "scripts": {
```

```
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.0",
    "express-session": "^1.17.0",
    "body-parser": "^1.20.0",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/csrf/server.js

```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const path = require('path');

const app = express();
const port = 3000;

// Налаштування сесій
app.use(session({
  secret: 'your-secret-key',
  resave: false,
  saveUninitialized: true
}));

// Налаштування EJS
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));

// Імітація бази даних
const users = {
  'admin': {
    password: 'admin123',
```

```
    balance: 1000
  }
};

// Middleware для перевірки авторизації
const requireAuth = (req, res, next) => {
  if (!req.session.user) {
    return res.redirect('/login');
  }
  next();
};

// Головна сторінка
app.get('/', requireAuth, (req, res) => {
  res.render('dashboard', {
    user: req.session.user,
    balance: users[req.session.user].balance
  });
});

// Сторінка входу
app.get('/login', (req, res) => {
  res.render('login', { error: null });
});

// Обробка входу
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  if (users[username] && users[username].password === password) {
    req.session.user = username;
    res.redirect('/');
  } else {
    res.render('login', { error: 'Невірний логін або пароль' });
  }
});

// Вихід
```

```
app.get('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/login');
});

// Вразливий ендпоінт для зміни балансу
app.post('/transfer', requireAuth, (req, res) => {
  const { amount } = req.body;
  // Вразливість: немає перевірки CSRF токена
  users[req.session.user].balance -= parseInt(amount);
  res.redirect('/');
});

app.listen(port, () => {
  console.log(`CSRF Vulnerability Demo running at http://localhost:${port}`);
});
```

vulnerable-apps/xxe/package.json

```
{
  "name": "xxe",
  "version": "1.0.0",
  "description": "Vulnerable XXE application",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.0",
    "body-parser": "^1.20.0",
    "libxmljs2": "^0.33.0",
    "ejs": "^3.1.9"
  }
}
```

vulnerable-apps/xxe/server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const libxmljs = require('libxmljs2');
```

```
const path = require('path');

const app = express();
const port = 3000;

// Налаштування EJS
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Middleware
app.use(bodyParser.text({ type: 'application/xml' }));
app.use(express.static(path.join(__dirname, 'public')));

// Головна сторінка
app.get('/', (req, res) => {
  res.render('index', { result: null, error: null });
});

// Вразливий ендпоінт для обробки XML
app.post('/parse', (req, res) => {
  try {
    // Вразливість: не відключаємо зовнішні сутності
    const xmlDoc = libxmljs.parseXml(req.body, {
      noblanks: true,
      noent: true, // Вразливість: дозволяємо зовнішні сутності
      nocdata: true
    });

    // Отримуємо значення з XML
    const result = xmlDoc.get('//data').text();

    res.render('index', {
      result: result,
      error: null
    });
  } catch (error) {
    res.render('index', {
      result: null,
```

```
      error: 'Помилка обробки XML: ' + error.message
    });
  }
});

app.listen(port, () => {
  console.log(`XXE Vulnerability Demo running at http://localhost:${port}`);
});

package.json
{
  "name": "cybersecurity-training-platform",
  "version": "1.0.0",
  "description": "Platform for cybersecurity training with vulnerable applications",
  "main": "index.js",
  "scripts": {
    "start": "docker-compose up --build",
    "dev": "docker-compose up",
    "stop": "docker-compose down",
    "clean": "docker-compose down -v"
  },
  "keywords": [
    "cybersecurity",
    "training",
    "vulnerable",
    "education",
    "ctf"
  ],
  "author": "Cybersecurity Training Platform",
  "license": "MIT",
  "dependencies": {
    "docker-compose": "^0.24.2"
  }
}

README.md
# Cybersecurity Training Platform
```

This is a virtual environment for testing and learning cybersecurity methods. The platform allows students to practice various cybersecurity attacks and defense techniques in a safe, isolated environment.

Features

- Web-based interface for accessing training modules
- Isolated Docker containers for each training scenario
- Pre-configured vulnerable applications for practice
- Real-time feedback and scoring system
- Progress tracking

Prerequisites

- Docker
- Docker Compose
- Modern web browser

Setup Instructions

1. Clone this repository
2. Navigate to the project directory
3. Run the following command to start the platform:
  ```bash
  docker-compose up --build
  ```
4. Access the platform at http://localhost:3000

Available Training Modules

1. SQL Injection
2. Cross-Site Scripting (XSS)
3. Command Injection
4. File Upload Vulnerabilities
5. Authentication Bypass

Security Notice

This platform is designed for educational purposes only. All training environments are isolated and should not be used for malicious purposes.

License

MIT License