ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської роботи
на тему:

«МЕТОДИКА РОЗРОБКИ НАВЧАЛЬНОЇ ПЛАТФОРМИ ДЛЯ ПОШУКУ ТА ЕКСПЛУАТАЦІЇ ЗАГАЛЬНОВІДОМИХ ВРАЗЛИВОСТЕЙ»

Виконав: студент 6 курсу, групи БСДМ-61
Спеціальності 125 Кібербезпека
Освітньо-професійної програми
«Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)
Седлецький Д. В.
(прізвище та ініціали)
Керівник Гайдур Г.І.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.

КИЇВ – 2020

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Інститут ННІЗІ
Кафедра Інформаційної та кібенетичної безпеки
Ступінь вищої освіти Магістр
Спеціальність 125 Кібербезпека
Освітньо-професійна програма Інформаційна та кібернетична безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Г.І. Гайдур
"___" _____ 2019 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Седлецькому Денису Володимировичу
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи: «Методика розробки навчальної платформи для пошуку та експлуатації загальновідомих вразливостей»
керівник магістерської роботи Гайдур Галина Іванівна, д.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «14» листопада 2019 р. №518

2. Строк подання студентом магістерської роботи 15.12.2019р.

3. Вихідні дані до магістерської роботи
Тестування на проникнення.
Загальновідомі вразливості і слабкості безпеки.
Технології віртуалізації.
Науково-технічна література. Стандарти. Рекомендації.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Тестування на проникнення. Загальновідомі вразливості та слабкості безпеки. Віртуалізація.
2. Розробка програмно-апаратної платформи.
3. Практична реалізація проекту.

5. Перелік графічного матеріалу (презентації):

1. Тема.
2. Тестування на проникнення. Загальновідомі вразливості.
3. Технології віртуалізації. Docker.
4. Результати розробки навчальної платформи для пошуку та експлуатації загальновідомих вразливостей.
5. Результати реалізації платформи в рамках навчального проекту для студентів.
6. Висновки.

6. Дата видачі завдання 30.09.2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1.	Аналіз науково-технічної літератури	14.10.2019 р.	
2.	Тестування на проникнення. Загальновідомі вразливості та слабкості безпеки. Віртуалізація.	28.10.2019 р.	
3.	Розробка програмно-апаратної платформи.	15.11.2019 р.	
4.	Практична реалізація проекту.	29.11.2019 р.	
5.	Реферат, вступ, висновки.	07.12.2019 р.	
6.	Підготовка презентації до захисту.	14.12.2019 р.	

Студент _____ Седлецький Д. В.
(підпис) (прізвище та ініціали)

Керівник магістерської роботи _____ Гайдур Г.І.
(підпис) (прізвище та ініціали)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ МАГІСТЕРСЬКОЇ РОБОТИ

Направляється студент Седлецький Д. В. до захисту магістерської роботи
(прізвище та ініціали)
спеціальності 125 Кібербезпека
освітньо-професійної програми Інформаційна та кібернетична безпека
(шифр і назва спеціальності)
на тему: «Методика розробки навчальної платформи для пошуку та експлуатації загальновідомих вразливостей».
Магістерська робота і рецензія додаються.

Директор інституту _____ В. А. Савченко
(підпис)

Довідка про успішність

Седлецький Д. В. _____ за період навчання в інституті
(прізвище та ініціали студента)
ННІЗІ з 2018 року до 2020 року повністю виконав навчальний план за напрямом підготовки, спеціальністю з таким розподілом оцінок за:
національною шкалою: відмінно ____%, добре ____%, задовільно ____%;
шкалою ECTS: A ____%; B ____%; C ____%; D ____%; E ____%.
Секретар інституту, факультету (відділення) _____ Гребенніков А.Б.
(підпис) (прізвище та ініціали)

Висновок керівника магістерської роботи

Студент ***Седлецький Д. В.*** обрав тему роботи, метою якої було сформувати та описати методику розробки навчальної платформи для пошуку та експлуатації загальновідомих вразливостей. Перелік використаних джерел свідчить про вміння магістром розбиратись в наукових питаннях та застосовувати їх при дослідженнях. Під час виконання магістерської роботи Седлецький Д. В. показав відмінну теоретичну та практичну підготовку, вміння самостійно вирішувати питання і робити висновки. Роботу виконував сумлінно, акуратно та вчасно за планом.

Все це дозволяє оцінити виконану магістерську роботу студента Седлецького Дениса Володимировича на оцінку **«відмінно»** та присвоїти йому кваліфікацію 2149.2 професіонал з організації інформаційної безпеки, викладач вищих навчальних закладів.

Керівник магістерської роботи _____ Гайдур Г.І.
(підпис)
“____”_____2019 року

Висновок кафедри про магістерську роботу

Магістерська робота розглянута. Студент Седлецький Д. В.
(прізвище та ініціали)
допускається до захисту даної магістерської роботи в Державній екзаменаційній комісії.
Завідувач кафедри Інформаційної та кібернетичної безпеки
(назва)
_____ Гайдур Г.І.
(підпис) (прізвище та ініціали)
“____”_____2019 рік

РЕФЕРАТ

Текстова частина магістерської роботи: 84 сторінки, 34 рисунки, 27 джерел.

Об'єкт дослідження – процес створення навчальної платформи для пошуку та експлуатації загальновідомих вразливостей.

Предмет дослідження – навчальна платформа для пошуку та есплуатації загальновідомих вразливостей та методика її розробки.

Мета роботи – розробити навчальну платформу для пошуку та експлуатації загальновідомих вразливостей.

Методи дослідження – огляд методичних матеріалів, аналіз технік наступальної кібербезпеки, дослідження технологій віртуалізації.

В роботі наведено основні відомості про тестування на проникнення і його мету, класифікацію загальновідомих вразливостей і проблем безпеки. Детально досліджено технології віртуалізації, в тому числі контейнерну віртуалізацію на прикладі інструменту Docker. На основі результатів дослідження розроблено методику створення навчальної платформи для пошуку та експлуатації загальновідомих вразливостей. В рамках магістерської роботи проведено освітній захід для студентів Державного університету телекомунікацій з використанням розробленої платформи.

Галузь використання – матеріали роботи доцільно використовувати при розробці сучасних профільних програм занять і дисциплін із залученням інноваційних методів навчання, що будуть орієнтовані на практичне застосування отриманих навичок.

ТЕСТУВАННЯ БЕЗПЕКИ, ВРАЗЛИВІСТЬ, СЛАБКІСТЬ, БЕЗПЕКА, ВІРТУАЛІЗАЦІЯ, DOCKER, РОЗРОБКА, ПРОЕКТ, ОПЕРАЦІЙНА СИСТЕМА.

ЗМІСТ

Стор.

ВСТУП **7**

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **9**

1.1. Тестування на проникнення 9

1.2. Загальні ідентифікатори вразливостей CVE та слабкостей програмного забезпечення CWE 12

1.2.1. Ідентифікатор загальновідомих вразливостей і проблем кібербезпеки CVE 12

1.2.2. Перелік типів слабкостей програмного забезпечення CWE 15

1.3. Різновид змагань з кібербезпеки у форматі гри 19

1.4. Технологія віртуалізації 21

1.4.1. Історія віртуалізації 23

1.4.2. Принцип роботи віртуалізації 24

1.4.3. Типи віртуалізації 25

1.4.4. Docker 29

2 ПРОГРАМНО-АПАРАТНА ПЛАТФОРМА **34**

2.1. Постановка задачі 34

2.2. Вибір технологій та серверних елементів. Обґрунтування вибору 34

2.3. Розгортання хостової операційної системи 37

2.4. Проектування й розробка вразливих систем 39

2.4.1. Розгортання системи, вразливої до SQL-Injection 39

2.4.2. Розгортання системи, вразливої до SambaCry 50

2.4.3. Розгортання системи, вразливої до Sensitive Infotmation Insecure Storage 55

2.5. Розгортання контрольної панелі CTF 62

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ **65**

3.1. Організація змагань для студентів Державного університету телекомунікацій 65

3.2. Результати змагань і досягнення студентів 70

3.3. Підсумки реалізації проекту 76

ВИСНОВКИ **77**

ПЕРЕЛІК ПОСИЛАНЬ **78**

ДОДАТОК А. ВИХІДНИЙ КОД СТОРІНКИ INDEX.PHP ДОДАТКУ, ВРАЗЛИВОГО ДО SQL INJECTION **81**

ДОДАТОК Б. ВМІСТ ФАЙЛУ DOCKER-COMPOSE.YML ДОДАТКУ CTFD **84**

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ) **86**

ВСТУП

В сучасному світі найбільшу цінність становить інформація. Проте, з невпинним розвитком технологій та засобів комунікації інформація може втрачати свою актуальність дуже швидко. Разом із тим знання, які отримує людина, можуть втратити свою значущість у короткий проміжок часу. Світ, що нас оточує, спонукає до змін і сучасних рішень. Поміж іншого, система освіти також потребує запровадження інноваційних методик, які б відповідали найсучаснішим тенденціям. Щороку публікуються десятки тисяч нових вразливостей та слабкостей безпеки, знайдених в програмних продуктах, що широко використовуються по всьому світу. Для того, щоб підготувати професіоналів у сфері кібербезпеки, які після випуску з університету могли б застосувати свої знання на практиці, програма профільних дисциплін має швидко пристосовуватись до сучасних викликів. Запровадження методики освіти із залученням новітніх технологій та неформальних методів навчання зможе не тільки підвищити зацікавленість студентів у профільній дисципліні, а й значно покращити рівень володіння практичними навичками, що безперечно стане вкрай корисним для застосування на практиці.

Об'єкт дослідження – процес створення навчальної платформи для пошуку та експлуатації загальновідомих вразливостей.

Предметом дослідження є навчальна платформа для пошуку та есплуатації загальновідомих вразливостей та проблем безпеки, а об'єктом дослідження – процес її розробки та запровадження.

Метою магістерської роботи є розробка навчальної платформи для пошуку та експлуатації загальновідомих вразливостей, яку було б доцільно використовувати при розробці сучасних профільних програм занять і дисциплін із залученням інноваційних методів навчання, що будуть орієнтовані на практичне застосування отриманих навичок.

Для досягнення мети необхідно вирішити такі основні задачі:

1. Дослідити процеси та технології тестування на проникнення;
2. Проаналізувати існуючі класифікації та списки загальновідомих вразливостей та слабкостей безпеки;
3. Дослідити технології віртуалізації;
4. Використовуючи контейнерну віртуалізацію, розробити навчальну платформу для пошуку та експлуатації загальновідомих вразливостей;
5. Застосувати розроблену платформу у навчальному проекті для студентів університету.

Результати дослідження безпеки контейнерів Docker оприлюднено у публікації на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки» у 2019 році:

Седлецький Д. В. Безпека контейнерів Docker. Кращі практики / Д. В. Седлецький // Актуальні проблеми кібербезпеки: Всеукраїнська наукова конференція: тези доп. – К.: ДУТ, 2019. – С. 11-13.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В розділі розглянуто теоретичні відомості щодо тестування на проникнення: що це таке і чому це потрібно, класифікацію загальновідомих вразливостей, список ідентифікаторів для загальновідомих вразливостей і проблем безпеки, що таке CTF у кібербезпеці і історію його появи. Також описано технології віртуалізації, контейнеризації, проведено аналіз роботи інструменту Docker.

1.1. Тестування на проникнення

Тестування на проникнення (penetration testing, pentest – тести на подолання захисту) – це детальний аналіз мережі і систем з точки зору потенційного зловмисника. Суть тесту полягає в санкціонованій спробі обійти існуючий комплекс засобів захисту інформаційної системи. В ході тестування роль зловмисника відіграє спеціаліст, який повинен визначити рівень захищеності, виявити вразливості, ідентифікувати найбільш вірогідні шляхи злому і визначити наскільки добре працюють засоби виявлення і захисту інформаційної системи від атак на підприємстві.

Тест на проникнення дозволяє отримати об'єктивну оцінку того, наскільки легко отримати доступ до ресурсів корпоративної мережі і сайту компанії, яким способом і через які вразливості. Тест на проникнення є моделюванням дій зловмисника по проникненню в інформаційну систему і дозволяє виявити вразливості в захисті мережі. Тест на проникнення проводять для здобуття незалежної оцінки захищеності своєї корпоративної мережі.

Метою тесту на проникнення є виявлення слабких місць в захисті інформаційної системи і, якщо це можливо, і відповідає бажанню замовника, здійснити показовий злом.

Основна задача тесту на проникнення: повністю імітуючи дії зломщика, здійснити атаку на веб-сервер, сервер застосувань або баз даних, персонал, корпоративну мережу.

Тестування на проникнення може проводитися як у складі аудиту на відповідність стандартам, так і у вигляді самостійної роботи.

На відповідність стандартам тест на проникнення проводиться перед початком діяльності підприємства або після впровадження комплексу засобів захисту. [1]

У вигляді самостійної роботи тести можуть проводитися з двома основними цілями: необхідності проведення робіт по підвищенню захищеності або здобуття незалежної оцінки рівня безпеки інформаційної системи.

У першому випадку замовниками часто виступають керівники підрозділів ІБ, яким необхідно продемонструвати вищому керівництву недоліки існуючої системи управління інформаційною безпекою.

У другому випадку роботи проводяться перед переведенням системи в промислову експлуатацію. В цьому випадку результати тестування дозволяють реально оцінити залишкові ризики, а можливо і виявити приховані недоліки в системі. Типовими прикладами подібних робіт є тест з використанням методів соціальної інженерії, після проведення циклу робіт по підвищенню обізнаності користувачів або злом нового веб-інтерфейсу системи клієнт-банк перед введенням її в промислову експлуатацію.

Перед початком проведення тесту на проникнення між сторонами підписується угода про нерозголошення, в рамках якої гарантується збереження в таємниці всієї конфіденційної інформації, яка буде отримана в процесі виконання робіт. В рамках договору на виконання робіт також може окремо визначатися мета тесту на проникнення. Метою, наприклад, може виступати доступ до певного файлу або спотворення запису у вказаній таблиці бази даних.

При плануванні тестування на проникнення необхідно визначити кордони і режим проведення тесту. Роботи можуть проводитися з повідомленням персоналу (системних і мережевих адміністраторів) або без повідомлення.

Якщо користувачі і адміністратори не знають про підготовлюваний злом, замовник отримає можливість оцінити ефективність використаних механізмів виявлення і розслідування комп'ютерних інцидентів, і підвищення обізнаності в області ІБ. З іншого боку прихований тест підвищує вірогідність виникнення відмови в результаті помилки експерта або зовсім некоректного налаштування серверів і мережевого устаткування. Тому досить часто тести на подолання периметра мережевої безпеки розбиваються на дві фази: зовнішню і внутрішню. На першому етапі аудитори працюють з мінімальними знаннями про систему, і їх метою є «пробити» периметр, наприклад, встановивши шкідливе програмне забезпечення на робочу станцію внутрішнього користувача. Після успішного виконання цього завдання вони переходять до оцінки захищеності мережі з боку внутрішнього зловмисника, вже координуючи свої дії з адміністраторами системи.

Типовий сценарій проникнення виглядає таким чином: досліджується можливість проникнення в систему за допомогою інформаційної атаки на одну або декілька робочих станцій, а також серверів. В разі успішного проведення атаки, за допомогою захоплених робочих станцій або серверів, встановлюється прихований канал зв'язку, що дозволяє контролювати захоплені комп'ютери ззовні, і ці майданчики використовуються для подальшого проникнення по заздалегідь позначеним цілям вже усередині периметра інформаційної системи, що атакується.

В результаті успішного і якісного проведеного тестування на проникнення, замовник отримує підсумковий звіт, що відображає об'єктивний і реальний погляд на рівень інформаційної безпеки компанії. Звіт також міститиме детальний технічний опис всіх зроблених і вдало реалізованих сценаріїв проникнення, аналіз всього можливого спектру дій на інформаційну інфраструктуру замовника, а також рекомендації по підвищенню рівня захищеності підприємства. Рекомендації містять

покроковий опис тих дій, які необхідно зробити для підвищення рівня захищеності компанії від загроз інформаційної безпеки. [2]

1.2. Загальні ідентифікатори вразливостей CVE та слабкостей програмного забезпечення CWE

1.2.1. Ідентифікатор загальновідомих вразливостей і проблем кібербезпеки CVE

Common Vulnerabilities and Exposures (CVE) – це список загальних ідентифікаторів для загальновідомих вразливостей і проблем кібербезпеки.

Вразливість (vulnerability) – це слабкість обчислювальної логіки (наприклад, в коді), виявлена в програмному забезпеченні та деяких апаратних компонентах (наприклад, вбудованому програмному забезпеченні), яка при експлуатації призводить до негативного впливу на конфіденційність, цілісність, або доступність. Усунення вразливості в цьому контексті, як правило, включає зміни в коді додатку, але може також включати зміни специфікацій або компонентів (наприклад, видалення порушених протоколів або функціональності в повному обсязі).

Проблема (exposure) – це вада конфігурації системи або помилка в програмному забезпеченні, що дозволяє отримати доступ до інформації або можливостей, які хакер може використовувати як перехідний етап при спробі отримання доступу до системи чи мережі.

CVE вважає певну слабкість конфігурації або помилку проблемою в тому випадку, якщо це не дозволяє безпосередньо скомпрометувати систему, але може стати важливою складовою успішної атаки і є порушенням розумної політики безпеки.

"Проблема" описує конкретний стан в обчислювальній системі (або наборі систем), що не є вразливим, але в той же час:

- дозволяє зловмиснику проводити діяльність щодо збору інформації;
- дозволяє зловмиснику приховувати діяльність;
- включає можливість очікуваної поведінки, яка може бути легко порушена;
- є первинною точкою входу, яку зловмисник може використати для отримання доступу до системи або даних;
- вважається проблемою відповідно до певної розумної політики безпеки.

Використання записів CVE, призначених CNA (CVE Numbering Authorities – органи нумерації CVE) з усього світу, забезпечує впевненість сторін при використанні для обговорення або обміну інформацією про унікальне програмне забезпечення або вразливість програмного забезпечення, забезпечує базову лінію для оцінки інструментів та дозволяє автоматизований обмін даними.

Переваги CVE:

- Один ідентифікатор однієї вразливості;
- Один стандартизований опис для кожної вразливості;
- Є словником, а не базою даних;
- Є універсальною «мовою» для різноманітних баз даних та інструментів;
- Є шляхом до кращої сумісності та поширення безпеки;
- Основа для оцінки серед сервісів, інструментів та баз даних;
- Безкоштовний для публічного завантаження та використання;
- Схвалена галуззю: існують органи нумерації CVE, рада CVE, численні продукти та сервіси використовують CVE.

CVE був запущений у 1999 році, коли більшість інструментів кібербезпеки використовували власні бази даних із власними ідентифікаторами для вразливостей безпеки. У той час існувала значна різниця між продуктами і було важко визначити,

коли різні бази даних посилаються на одну й ту саму проблему. Як наслідок, існували прогалини в загальному поширенні безпеки, ефективна сумісність між різними базами даних та інструментами була відсутня. Крім того, кожен постачальник інструментів використовував різні показники, щоб вказати кількість виявлених вразливостей, що означало, що серед інструментів немає стандартизованої основи для оцінки.

Загальні, стандартизовані ідентифікатори CVE забезпечили вирішення цих проблем.

CVE тепер є галузевим стандартом для ідентифікаторів вразливості та впливу. Записи CVE – спільнотою також звуться "CVEs," "CVE IDs," і "CVE numbers" – надають орієнтири для обміну даними, щоб продукти та послуги з кібербезпеки могли комунікувати між собою. Записи CVE також надають базис для оцінки охоплення інструментів та сервісів, щоб користувачі могли визначити, які інструменти є найбільш ефективними та відповідними потребам організації. Таким чином, продукти та послуги, сумісні з CVE, забезпечують краще покриття, простішу сумісність та підвищену безпеку.

Кожний запис CVE містить:

– Ідентифікаційний номер CVE (наприклад, "CVE-1999-0067", "CVE-2014-10001", "CVE-2014-100001");
– Короткий опис вразливості або проблеми безпеки;
– Будь-які відповідні посилання (тобто звіти про вразливість та рекомендації).

Процес створення запису CVE починається з виявлення потенційної вразливості безпеки. Потім інформації присвоюється ідентифікатор CVE органом нумерації CVE (CNA). CNA створює опис і додає посилання, а потім готовий запис додається до списку CVE та розміщується на веб-сайті командою CVE.

Корпорація MITRE в даний час підтримує CVE та публічний веб-сайт cve.mitre.org, здійснює нагляд за органами CNA та радою CVE і надає неупереджене технічне керівництво протягом усього процесу, щоб забезпечити відповідність CVE суспільним інтересам.

Крім того, команда MITRE CVE в даний час функціонує як кореневий CNA програми CVE. Органи нумерації CVE (CNA) – це вендори, проекти, дослідники вразливостей, національні та галузеві CERT, а також програми bug bounty, що присвоюють записи CVE нововиявленим проблемам без безпосереднього залучення команди CVE до подробиць конкретних вразливостей, і включають ідентифікатори CVE під час першого публічного розкриття вразливостей.

Рада CVE – до складу Ради входять численні організації, пов'язані з кібербезпекою, включаючи постачальників інструментів комерційної безпеки, наукові установи, науково-дослідні установи, урядові відомства та інших відомих експертів з питань безпеки, а також кінцевих користувачів. Шляхом відкритих та спільних обговорень Рада надає критичну інформацію щодо джерел даних, покриття продукції, її мети, операційної структури та стратегічного напряму програми CVE.

Спонсор CVE – CVE спонсорується Агентством кібербезпеки та безпеки інфраструктури США (CISA) Міністерства внутрішньої безпеки (DHS).[3]

1.2.2. Перелік типів слабкостей програмного забезпечення CWE

Загальний перелік слабкостей (CWE – Common Weakness Enumeration) – офіційний перелік типів слабкостей програмного забезпечення, який:

– Служить загальною мовою для опису слабкостей безпеки програмного забезпечення в архітектурі, дизайні чи коді;

– Служить стандартною міркою для інструментів безпеки програмного забезпечення, спрямованих на ці слабкі сторони;

– Забезпечує загальний базовий стандарт для виявлення слабкостей, їх пом'якшення та запобігання.

Організації хочуть впевнитися, що програмні продукти, які вони розробляють, не містять відомих типів вад безпеки. Сьогодні високоякісні інструменти та сервіси для пошуку недоліків у безпеці та коді є новими, і на питання про те, який інструмент чи послуга підходить краще для певної роботи, важко відповісти, враховуючи відсутність структури та конкретних визначень в галузі оцінки коду. CWE створений спеціально для вирішення цих проблем.

Компанія MITRE почала працювати над питанням категоризації слабких місць програмного забезпечення ще на початку 1999 року, коли запустила список CVE. У рамках розробки командою CVE MITRE було створено попередню класифікацію та категоризацію вразливих місць, атак, несправностей та інших концепцій, які допоможуть визначити загальні слабкості програмного забезпечення. Однак, хоча вони є достатніми для CVE, ця категоризація є занадто грубою, щоб її можна було використовувати для ідентифікації та категоризації функціональних можливостей, для викладення у пропозиціях галузі щодо оцінки безпеки коду. Для цього потрібна додаткова вірність та стислість, а також додаткові деталі та опис для кожної групи, такі як ефекти, поведінка, деталі реалізації тощо.

Для цього MITRE випустив першу редакцію при перегляді роботи внутрішніх категорій CVE для використання в галузі оцінки коду у 2005 році в рамках участі MITRE в Проекті Метрики та оцінювання інструментів (SAMATE) Національного інституту стандартів та технологій (NIST), що спонсорується Міністерством національної безпеки США (DHS). Отриманий документ під назвою Попередній перелік прикладів вразливостей для дослідників (PLOVER) був робочим документом, в якому перераховано понад 1500 різноманітних прикладів вразливих ситуацій у реальному світі, ідентифікованих за їх назвою CVE. Вразливості в PLOVER організовані в детальній концептуальній базі, яка наразі налічує 290

окремих типів слабких місць програмного забезпечення з великою кількістю прикладів вразливостей на реальних випадках для кожної з них. PLOVER представляв собою першу спробу по-справжньому перевернути спосіб використання відомих в реальному світі недоліків, які існують у коді, абстрагувати їх та об'єднати у загальні класи, що представляють більш загальні потенційні вразливості, які можуть існувати в коді, а потім, організувати їх у відповідній відносній структурі, щоб зробити їх доступними та корисними для широкої аудиторії для різноманітного набору цілей.

Наступним кроком після PLOVER було встановлення прийнятних визначень та описів цих загальних слабкостей громадою в рамках проекту NIST SAMATE, що призвело до створення списку «Загальне перерахування слабкостей (CWE)» та відповідної класифікаційної систематики. Список CWE тепер слугує механізмом опису можливостей оцінки вразливості коду з точки зору їх охоплення різними CWE.

Покупці програмного забезпечення хочуть бути впевненими у тому, що програмні продукти, які вони отримують, перевіряються на відомі типи недоліків у безпеці. Однак інструменти та послуги, які можна використовувати для таких перевірок, є в кращому випадку новими, і не мають систематики чи стандартів, які б визначали можливості та охоплення цих інструментів та послуг. Це ускладнює порівняльне визначення того, який інструмент чи послуга найкраще підходить для певної роботи. Тому постає необхідність в стандартному переліку та класифікації слабких місць безпеки програмного забезпечення, які слугуватимуть об'єднавчою мовою дискурсу та виміром для інструментів та послуг.

CWE – це розроблений спільнотою офіційний перелік поширених слабкостей програмного забезпечення. Він слугує загальною мовою для опису слабких місць безпеки програмного забезпечення, стандартним виміром для засобів захисту програмного забезпечення, спрямованих на ці вразливості, і є базовим стандартом для виявлення вразливостей, їх пом'якшення та запобігання. Враховуючи

різноманітні бачення на цю тему зі сфер науки, комерційного сектору та уряду, CWE об'єднує найціннішу ширину та глибину змісту та структури, щоб слугувати єдиним стандартом. Мета CWE – допомогти сформувати та поліпшити галузь оцінки безпеки коду, а також поширити досвід проведення оцінки безпеки програмного забезпечення для організацій при перегляді програмних систем, які вони купують чи розробляють.

CWE не тільки охоплює значну частину (15 000 записів) зі списку CVE, але також включає деталізацію, широту та класифікаційну структуру з різноманітного набору інших галузевих та наукових джерел та прикладів, включаючи серед інших таксономію McGraw/Fortify "Kingdoms"; "19 смертних гріхів" Говарда, ЛеБланка та Віги; і проект CLASP Secure Software.

Тож і користувачі, і розробники інструментів оцінки безпеки програмного забезпечення тепер мають механізм опису кожної з можливих вразливостей безпеки програмного забезпечення з точки зору їх охоплення різними CWE. Якщо необхідно, CWE також може бути визначений для конкретних мов, рамок, платформ та машинних архітектур.

Характерні ознаки CWE:

- Забезпечує загальну мову дискурсу для обговорення, пошуку та вирішення причин слабкостей безпеки програмного забезпечення, що виявляються в коді, дизайні чи архітектурі.
- Дозволяє постачальникам інструментів програмного забезпечення та постачальникам послуг чітко і послідовно заявляти про недоліки в безпеці перед своїми потенційними користувачами, використовуючи коректну термінологію, запроваджену CWE. Крім того, нова «Сумісність із CWE» була розроблена для того, щоб дозволити постачальникам інструментів безпеки та послуг публічно заявляти та описувати свої продукти у відповідності до CWE.

- Дозволяє покупцям порівнювати, оцінювати та вибирати інструменти та послуги із забезпечення безпеки програмного забезпечення, які є найбільш відповідними їх потребам, включаючи наявність певного рівня покриття CWE, які може знайти даний інструмент. Завдяки цьому, покупці програмного забезпечення мають змогу порівняти пропозиції інструментів та послуг із переліком CWE та мовами програмування, які використовуються в програмному забезпеченні.
- CWE надають для пояснення приклади коду кожного запису CWE.
- Забезпечує перевірку заяв щодо покриття, зроблених постачальниками інструментів безпеки програмного забезпечення та постачальниками послуг (це підтримується за допомогою метаданих CWE та порівняння з базовим набором даних SAMATE).
- Дозволяє уряду та промисловості використовувати цю стандартизацію в договірних умовах. [4]

1.3. Різновид змагань з кібербезпеки у форматі гри

Capture the flag (CTF, укр. Захоплення прапора) – різновид змагань з кібербезпеки, що відбуваються у форматі гри. CTF вимагає від учасників вирішення завдань різного рівня складності, що охоплюють різні напрями інформаційної та кібернетичної безпеки. Для вирішення завдань учасникам знадобляться різноманітні навички та набори програмних засобів. В результаті виконання завдання учасник отримує «прапор», який має відправити на сервер CTF, щоб заробити бали. Зазвичай прапор є довільним текстовим рядком подібного формату: “ctf{you_found_first_flag!}” або “flag{327a6c4304ad5938eaf0efb6cc3e53dc}”. Такі події зазвичай проводяться на конференціях з кібербезпеки, наприклад DEFCON, BSides, BlackHat тощо.

Перший CTF з кібернетичної безпеки було проведено на конференції DEFCON у 1996 році у Лас Вегасі, штат Невада. DEFCON – це найбільша конференція з кібербезпеки у Сполучених Штатах Америки, що була офіційно започаткована у 1993 році Джефом Моссом. DEFCON став платформою для обміну досвідом і навичками з кібернетичної безпеки, і в міру розростання Інтернету популярними стали і DEFCON, і CTF. Змагання CTF стали глобальними, оскільки вони не мали жодних кордонів і їх можна проводити через Інтернет. Міжнародні команди змагалися за різні типи призів та місця в загальному рейтингу.

Існує два основні формати проведення CTF з кібербезпеки:

- Attack-Defence;
- Jeopardy.

Attack-Defence (атака і захист) – формат проведення CTF, коли кожна команда атакує систему іншої команди, а також захищає власну систему. У інформаційних системах (наприклад, серверах), які є об’єктами захисту, є прапори (текстові файли, папки, зображення тощо), які атакуюча команда намагається знайти за допомогою атак на вразливі системи та сервіси. Команда, що атакує, може використовувати різні інструменти для злому та компрометації машин, що захищаються, але існують певні правила, які гарантують, що команди мають рівні умови. Команда, яка захищається, може здійснювати будь-які дії в межах правил, щоб захистити свої машини від атакуючої команди. Їм заборонено відключати мережеві з'єднання або вимикати машини. Якщо порушення правил матиме місце, команда понесе штраф або буде дискваліфікована.

Jeopardy – CTF у стилі вікторини. В цій грі наведені завдання різних категорій, що стосуються інформаційної та кібернетичної безпеки, програмування, комп’ютерних технологій тощо. При цьому команд може бути більше, ніж дві, оскільки команди не атакують одна одну. До основних категорій можна віднести криптографію, стеганографію, експлуатацію вразливостей, аналіз інцидентів

кібербезпеки, сканування тощо. Виконання завдань різної складності дають різну кількість переможних балів. Гра є обмеженою в часі, для запуску і зупинки CTF використовується таймер, і як тільки час буде вичерпано, гра закінчена. Перемагає команда з найбільшою кількістю балів в кінці.

Метою проведення таких змагань є поширення зацікавленості інформаційною та кібернетичною безпекою, здобуття учасниками корисних навичок з практичної кібербезпеки, як з боку атакуючого (що може стати у нагоді, наприклад, при оцінці рівня захищеності інформаційної системи), так і з боку захисника систем.

Змагання Capture the flag є популярними і тому, що для участі в них не обов’язково володіти особливими навичками з кібербезпеки, спробувати себе може будь-який охочий навіть без попередньої підготовки. Безперечно, для перемоги необхідно вміти зорієнтуватися в умовах обмеженого часу, швидко знаходити інформацію, розумітись в інформаційних технологіях, інформаційній та кібернетичній безпеці, бути в курсі новин зі світу кібербезпеки.[11, 12]

1.4. Технологія віртуалізації

Віртуалізація - це технологія, яка дозволяє створювати кілька модельованих середовищ або виділених ресурсів з єдиної фізичної апаратної системи. Програмне забезпечення під назвою гіпервізор підключається безпосередньо до цього обладнання та дозволяє розділити одну систему на відокремлені та безпечні середовища, відомі як віртуальні машини (VM). Ці віртуальні машини покладаються на здатність гіпервізора відокремлювати ресурси машини від обладнання та розподіляти їх належним чином. Віртуалізація допомагає отримати найбільшу користь з найменших ресурсів.

Фізичне обладнання, оснащене гіпервізором, називається хостом (host), в той час як віртуальні машини, які використовують його ресурси, - гості (guests). Ці гості розглядають обчислювальні ресурси (наприклад, процесор, пам'ять та сховище) як

пул ресурсів, який легко перемістити. Оператори можуть контролювати віртуальні екземпляри процесора, пам'яті, сховища та інших ресурсів, тому гості отримують необхідні їм ресурси, коли вони їм потрібні.

Віртуалізація дозволяє використовувати повну потужність фізичної машини, розподіляючи її можливості серед багатьох користувачів або середовищ.

У практичному розумінні, уявімо, що у є 3 фізичні сервери з індивідуальними спеціальними функціями. Один - поштовий сервер, другий - веб-сервер, а останній запускає внутрішні застарілі програми. Кожен сервер використовується з потужністю близько 30% - це лише частка їх потенціалу. Але оскільки застарілі програми залишаються важливими для ваших внутрішніх операцій, ви повинні зберегти їх та третій сервер, який їх розміщує (рис. 1.1).

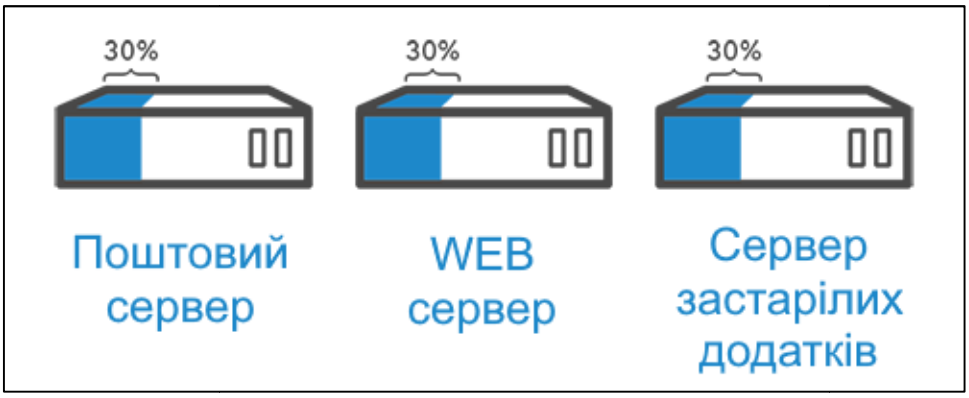

Рис. 1.1. Розподіл задач між фізичними серверами

Зазвичай, було б легше і надійніше виконувати окремі завдання на окремих серверах: один сервер - одна операційна система - одне завдання. Але за допомогою віртуалізації ви можете розділити поштовий сервер на 2 унікальних відокремлених сервери, які можуть працювати на різних операційних системах та з незалежними завданнями, щоб перенести застарілі програми. Це одне і те ж обладнання, але застосовується більш ефективне його використання (рис 1.2).

Рис. 1.2. Об’єднання функцій двох серверів за допомогою віртуалізації

Знову розділивши перший сервер, щоб він міг вирішувати ще одне завдання – збільшимо його використання з 30% до 90%. Завдяки цьому отримуємо ефективне використання ресурсів та два вільні сервери, які тепер можна буде повторно використовувати для виконання інших завдань або взагалі звільнити для зменшення витрат на охолодження та обслуговування.

1.4.1. Історія віртуалізації

Незважаючи на те, що технологія віртуалізації була розроблена ще до 1960-х років, вона не набула широкого сприйняття до початку 2000-х. Технології, що надавали можливість віртуалізації - як гіпервізори - були розроблені десятки років тому, щоб мати можливість надати декільком користувачам одночасний доступ до комп'ютерів, які виконували пакетну обробку. Пакетна обробка була популярним стилем обчислень у бізнес-секторі, який виконував рутинні завдання тисячі разів дуже швидко.

Але протягом кількох наступних десятиліть популярність проблеми «кілька користувачів – один комп’ютер» зростала, в той час як популярність віртуалізації поступово згасала. Одним з рішень був розподіл часу, який ізолював окремих користувачів в операційній системі. Саме це ненавмисно призвело до появи інших

операційних систем, таких як UNIX, які врешті-решт поступилися місцем Linux. В цей час віртуалізація залишалася в основному непопулярною, нішевою технологією.

В1990-х роках більшість підприємств мали фізичні сервери та IT-стеки від одного постачальника, що не дозволяло додаткам запускатись на обладнанні іншого постачальника. Оскільки компанії оновлювали своє IT-середовище дешевшими серверами, операційними системами та додатками від різних постачальників, це призводило до неефективного використання фізичного обладнання - кожен сервер міг виконувати лише одне завдання, яке стосується одного постачальника.

Саме в цей час почала розвиватись віртуалізація. Це було вигідним рішенням двох проблем: компанії могли розділити свої сервери та запускати додатки різних версій на різних операційних системах. Сервери почали використовуватись більш ефективно, тим самим зменшивши витрати, пов'язані з придбанням, налаштуванням, охолодженням та технічним обслуговуванням.

Широка застосованість віртуалізації допомогла зменшити залежність від постачальників і стала основою хмарних обчислень. Сьогодні вона настільки поширена на підприємствах, що часто потрібно спеціалізоване програмне забезпечення для управління віртуалізацією, яке допомагає все відстежувати.

1.4.2. Принцип роботи віртуалізації

Програмне забезпечення під назвою гіпервізори відокремлює фізичні ресурси від віртуальних середовищ, які потребують цих ресурсів. Гіпервізори можуть розташовуватись поверх операційної системи (наприклад, на ноутбуці чи персональному комп’ютері) або встановлюватися безпосередньо на апаратне забезпечення (наприклад, на сервер), саме так і відбувається на більшості підприємств. Гіпервізори беруть ваші фізичні ресурси та розділяють їх таким чином, щоб віртуальне середовище могло ними користуватися (рис 1.3).

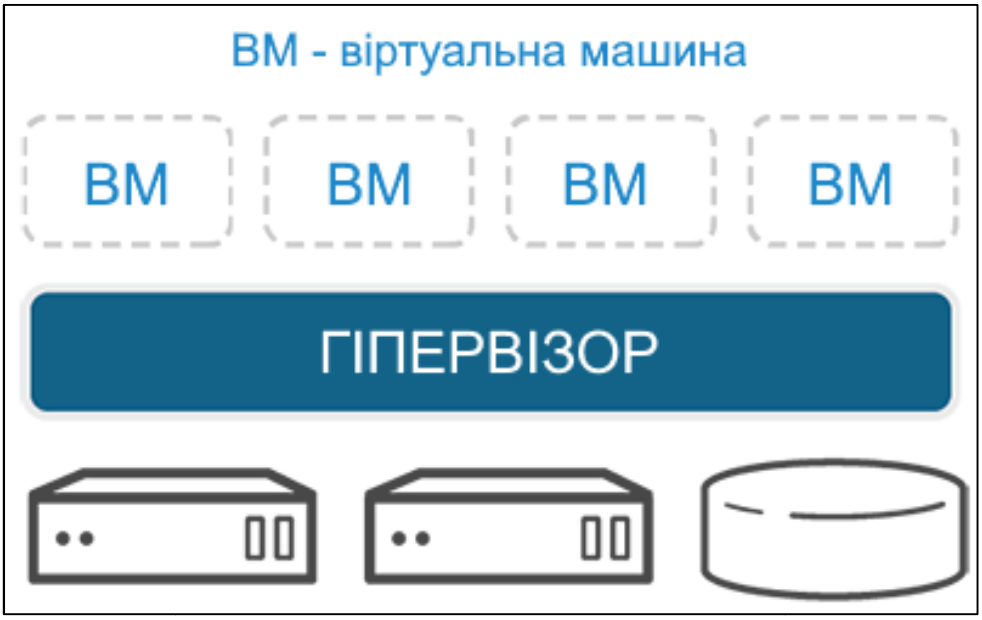

Рис. 1.3. Приклад застосування гіпервізора

Ресурси розподіляються за необхідністю від фізичного середовища до багатьох віртуальних середовищ. Користувачі взаємодіють з обчисленнями та проводять обчислення у віртуальному середовищі (як правило, це називається гостьова машина або віртуальна машина). Віртуальна машина функціонує як єдиний файл даних. І як і будь-який цифровий файл, його можна переміщувати з одного комп'ютера на інший, відкривати на будь-якому комп’ютері, і, як очікується, він буде працювати однаково.

Коли віртуальне середовище працює і користувач або програма видає інструкцію, яка вимагає додаткових ресурсів у фізичному середовищі, гіпервізор переносить запит у фізичну систему та кешує зміни - це все відбувається з наближеною до фізичного середовища швидкістю (особливо, якщо запит надсилається через гіпервізор з відкритим кодом на базі KVM, тобто віртуальної машини на основі ядра).

1.4.3. Типи віртуалізації

1.4.3.1. Віртуалізація даних (рис. 1.4)

Рис. 1.4. Віртуалізація даних

Дані, які поширюються повсюди, можна об'єднати в єдине джерело. Віртуалізація даних дозволяє компаніям ставитися до даних як до динамічного джерела - надаючи можливості обробки, які можуть консолідувати дані з різних джерел, легко розміщувати нові джерела даних та трансформувати дані відповідно до потреб користувачів. Інструменти віртуалізації даних розташовуються перед кількома джерелами даних і дозволяють їм розглядатися як єдине джерело, доставляючи потрібні дані (у необхідній формі і в потрібний час) будь-якій програмі чи користувачеві.

1.4.3.2. Віртуалізація робочих столів (рис. 1.5)

На відміну від віртуалізації операційних систем, що дозволяє розгортати кілька операційних систем на одній машині, віртуалізація робочих столів дозволяє центральному адміністратору (або автоматизованому інструменту адміністрування) розгортати імітовані середовища робочого столу на сотні фізичних машин одночасно. На відміну від традиційних середовищ робочого столу, які фізично встановлюються, налаштовуються та оновлюються на кожній машині, віртуалізація

робочих столів дозволяє адміністраторам проводити масові конфігурації, оновлення та перевірку безпеки на всіх віртуальних робочих столах.

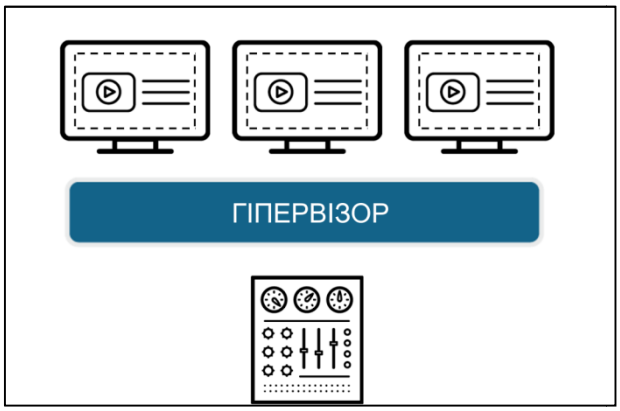

Рис. 1.5. Віртуалізація робочих столів

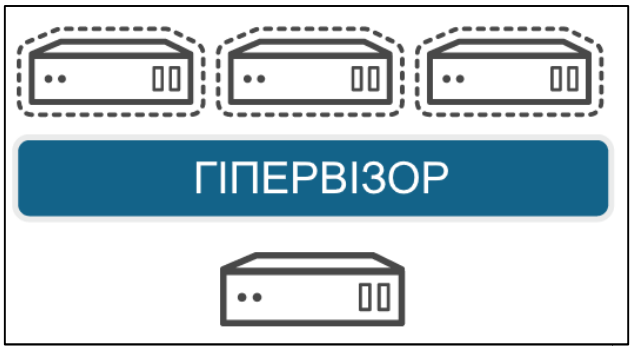

Рис. 1.6. Віртуалізація серверів

Сервери - це комп’ютери, призначені для дуже високої кількості конкретних завдань, в той час як інші комп’ютери, як-от ноутбуки чи персональні настільні комп’ютери, можуть виконувати різні завдання. Віртуалізація сервера дозволяє йому виконувати більше цих специфічних функцій і включає розділення, щоб його компоненти могли використовуватися для обслуговування декількох функцій.

1.4.3.3. Віртуалізація операційних систем (рис. 1.7)

Віртуалізація операційної системи відбувається в ядрі - центральному менеджері завдань операційних систем. Це зручний спосіб запускати середовища Linux та Windows поряд.

Рис. 1.7. Віртуалізація операційних систем

Підприємства можуть також встановлювати віртуальні операційні системи на комп'ютери, оскільки це:

- Знижує об'ємні витрати на обладнання, оскільки комп’ютери не потребують таких високих можливостей;
- Підвищує безпеку, оскільки всі віртуальні екземпляри можна відстежувати та ізолювати;
- Обмежує час, витрачений на обслуговування IT, наприклад, оновлення програмного забезпечення.

1.4.3.4. Віртуалізація мережевих функцій (рис. 1.8)

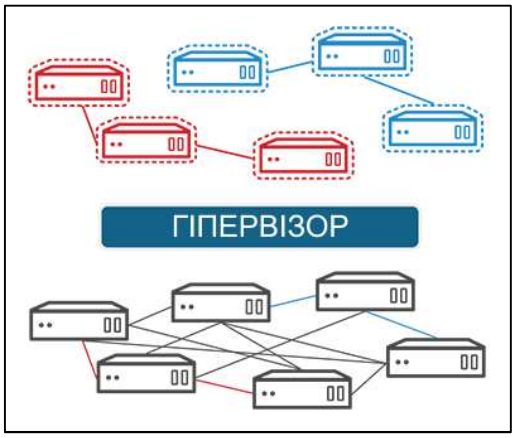

Рис. 1.8. Віртуалізація мережевих функцій

Віртуалізація мережевих функцій (NFV) розділяє основні функції мережі (наприклад, служби каталогів, обмін файлами та конфігурація IP), щоб вони могли бути розподілені між середовищами. Якщо функції програмного забезпечення не залежать від фізичних машин, на яких колись вони існували, конкретні функції можуть бути упаковані разом у нову мережу та призначені для оточення. Віртуалізація мереж зменшує кількість фізичних компонентів, таких як комутатори, маршрутизатори, сервери, кабелі та концентратори, що необхідні для створення декількох незалежних мереж, і це особливо популярно в телекомунікаційній галузі.[5]

1.4.4. Docker

Docker - це програмна платформа для побудови додатків на основі контейнерів - невеликих та легких середовищ виконання, які спільно використовують ядро операційної системи, але в іншому працюють окремо одне від одного. Незважаючи на те, що контейнери як концепція існують вже деякий час, Docker, проект з відкритим кодом, що був запущений у 2013 році, допоміг популяризувати технології та допоміг просунути тенденцію до контейнеризації та використання мікросервісів у

розробці програмного забезпечення, що стало великим поштовхом до розвитку хмарного середовища.

Однією з цілей сучасної розробки програмного забезпечення є розташування додатків на одному хості або кластері ізольовано, щоб вони не надто перешкоджали роботі або обслуговуванню один одного. Це може бути складно, через пакети, бібліотеки та інші програмні компоненти, що необхідні для їх роботи. Одним з варіантів вирішення цієї проблеми були віртуальні машини, які підтримують додатки на одному і тому ж апаратному забезпеченні повністю відокремленими і зводять до мінімуму конфлікти між компонентами програмного забезпечення та конкуренцію за апаратні ресурси. Але віртуальні машини є громіздкими - кожна потребує власну операційну систему, тому типово є розміром в кілька гігабайт, складна в обслуговуванні та модернізації. На противагу віртуальним машинам, контейнери ізолюють середовища виконання програм одне від одного, але поділяють базове ядро ОС. Зазвичай вони вимірюються в мегабайтах, використовують набагато менше ресурсів, ніж віртуальні машини, і запускаються практично миттєво. Контейнери забезпечують високоефективний та деталізований механізм для комбінування програмних компонентів у стеки додатків та сервісів, необхідних на сучасному підприємстві, та для постійного оновлення та обслуговування цих компонентів програмного забезпечення.[7]

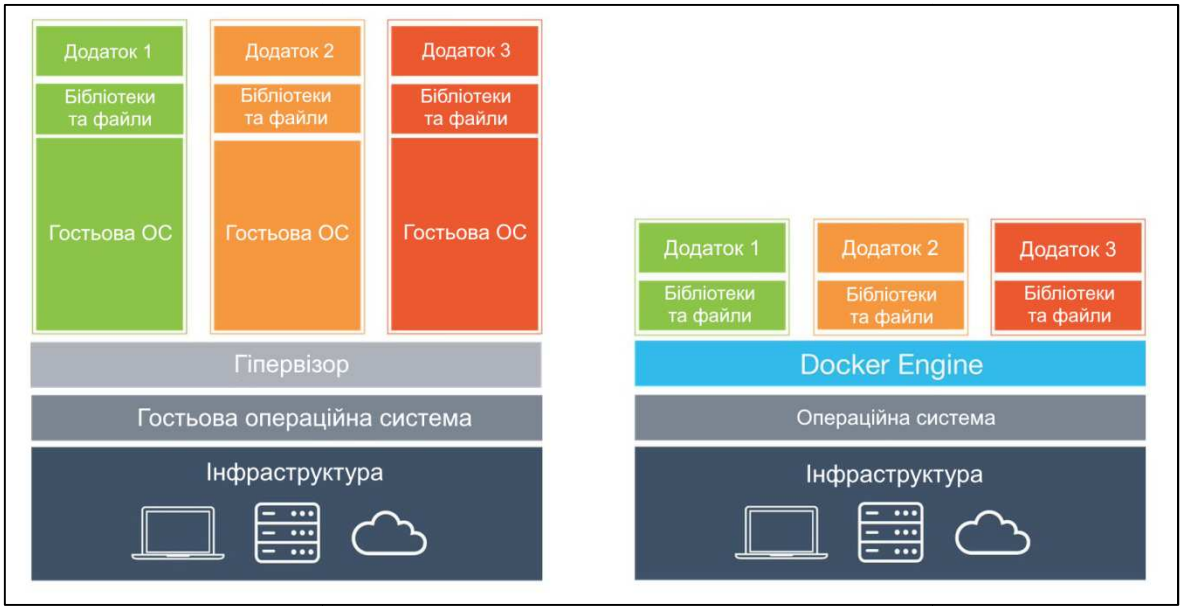

Рис. 1.9. Порівняння структури віртуалізації та контейнерної структури

1.4.4.1. Компоненти Docker

Docker - це проект з відкритим кодом, який дозволяє легко створювати контейнери та додатки на основі контейнерів. Спочатку створений для Linux, тепер Docker також працює на Windows та MacOS. Щоб зрозуміти, як працює Docker, давайте розглянемо деякі компоненти, які ви використовуються для створення додатків, що містять контейнери Docker.

1.4.4.1.1. Dockerfile

Кожен контейнер Docker починається з Dockerfile. Dockerfile - це текстовий файл, написаний у простому для розуміння синтаксисі, що включає в себе інструкції зі створення образу Docker (Docker image). Dockerfile визначає операційну систему, яка буде лежати в основі контейнера, а також мови, змінні середовища, місця розташування файлів, порти мережі та інші необхідні йому компоненти, і, звичайно, те, що контейнер насправді буде робити, коли ми запустимо його.

1.4.4.1.2. Docker Image

Після створення Dockerfile, викликається утиліта Docker build, щоб створити образ на основі цього Dockerfile. Тоді як Dockerfile - це набір інструкцій, який розповідає про побудову, як створити образ, Docker image - це портативний файл, що містить специфікації, для яких програмних компонентів буде запускатися контейнер і як. Оскільки Dockerfile, ймовірно, містить інструкції щодо захоплення деяких програмних пакетів із онлайн-сховищ, вам слід подбати про те, щоб чітко вказати належні версії, інакше ваш Dockerfile може створити непослідовні образи залежно від того, коли він викликаний. Але щойно образ створюється, він стає статичним.

Образ складається із "шарів" інших образів, і будь-які зміни, які вносяться до образу, здійснюються додаванням нового шару.

Залежності шарів Docker визначають впорядкування, і будь-який шар, який не залежить від будь-якого іншого шару, називається базовим образом. Зазвичай цей шар описує операційну систему для середовища.

1.4.4.1.3. Docker run

Утиліта Docker run - це команда, яка фактично запускає контейнер. Кожен контейнер - це екземпляр образу. Контейнери мають тимчасовий характер, але їх можна зупинити і перезапустити, після чого контейнер повертається у той самий стан, що і під час його зупинки. Крім того, кілька екземплярів контейнера одного і того ж образу можуть бути запущені одночасно (до тих пір, поки кожен контейнер має унікальну назву).

1.4.4.1.4. Docker контейнер

Контейнер Docker - це віртуальний приватний сервер. Він може розміщувати програмні програми та їх залежності, які можна запускати з різних середовищ. Контейнери складаються з операційної системи, файлів, доданих користувачем файлів та метаданих.

Контейнери побудовані з образів, але мають верхній шар читання-запису, тому, на відміну від образів, є інтерактивним і можуть зберігати стан. Це означає, що ви можете запускати програми всередині них. Після будь-яких змін у вашому контейнері Docker ви можете зберегти знімок отриманого стану у вигляді іншого образу.

Використання Docker виключає можливість конфліктів між вашим місцевим середовищем розробки та середовищем виконання і гарантує, що інструменти будуть працювати на платформі незалежно від того, в якому форматі вони перебувають.

1.4.4.1.5. Docker Hub

Хоча створювати контейнери легко, це не означає, що необхідно створювати кожний образ з нуля. Docker Hub - сховище для обміну та керування контейнерами, де знаходяться офіційні образи Docker від проектів з відкритим кодом та постачальників програмного забезпечення та неофіційні образи від широкої громадськості. Ви можете отримати образи контейнерів, що містять корисний код, або завантажити власні, відкрито поділитися ними або зробити їх приватними. Також є можливість створення локального реєстру Docker.

1.4.4.1.6. Docker Compose

Docker-compose - це інструмент, який надається Docker. Щоб зробити використання Docker ще простішим, цей інструмент реалізований для вирішення архітектурних проблем у проектах.

Коли ви розробник, ви рідко створюєте окрему програму (додаток, який не потребує запуску інших служб, наприклад, бази даних). Тож все дуже просто - якщо для роботи програми потрібно запустити кілька служб, знадобиться інструмент Docker-compose. Наприклад, при створенні веб-сайту, що потребує підключення до бази даних для автентифікації користувачів (2 сервіси: веб-сайт та база даних). Docker-compose надає можливість запустити всі ці сервіси однією командою.[8,9]

2 ПРОГРАМНО-АПАРАТНА ПЛАТФОРМА

2.1. Постановка задачі

З розвитком технологій розвиваються й методи навчання. Тому й програма профільних дисциплін має швидко пристосовуватись до новітніх тенденцій та технологій. Щороку публікуються десятки тисяч нових вразливостей, знайдених в програмних продуктах, що широко використовуються по всьому світу. Для того, щоб захисти свої системи та додатки від подібних вразливостей, необхідно розглянути їх детальніше та зрозуміти принципи їх роботи.

Мета роботи: розробити платформу, на основі якої студенти матимуть змогу вивчати загальновідомі вразливості та експлуатувати їх в умовах, максимально наближених до умов роботи реальних продуктивних систем.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- Розгортання відмовостійкого серверу, що здатний витримувати вхідні атаки;
- Розробка інформаційних систем, які будуть містити загальновідомі вразливості;
- Моделювання умов роботи реальних продуктивних систем;
- Забезпечення цілісної інфраструктури для вразливих систем.

2.2. Вибір технологій та серверних елементів. Обґрунтування вибору

Для того, щоб забезпечити цілісну і достатньо відмовостійку інфраструктуру в умовах обмежених ресурсів, буде використано технології віртуалізації, а саме контейнерну віртуалізацію за допомогою Docker.

Переваги використання Docker:

– Швидке розгортання – час розгортання контейнеру може займати лічені секунди, оскільки Docker створює контейнери для кожного процесу окремо і навіть не завантажує операційну систему;

– Простота конфігурації – Docker надає гнучкість користувачам імплементувати власну конфігурацію безпосередньо в код і надалі розгортати в будь-яких системах, оскільки Docker може використовуватись в найрізноманітніших середовищах;

– Доступність – Docker Engine має версію Community Edition, яка розповсюджується абсолютно безкоштовно;

– Безпека – Docker гарантує, що програми, що працюють в контейнерах, повністю відокремлені та ізольовані один від одного.

Використовуючи Docker, буде реалізовано одночасну роботу кількох окремих операційних систем, розташованих на одному фізичному сервері. При цьому, завдяки Docker, є можливість забезпечити ефективний розподіл ресурсів серверу та забезпечити ізольовані середовища для кожної системи. Тобто після експлуатації однієї системи, атакуючий не матиме змоги вийти за межі гостьової віртуальної системи і отримати доступ до хостового серверу та інших гостьових віртуальних систем.

В якості хостової серверної операційної системи буде використовуватись Ubuntu Server 18.04.3 LTS. Ubuntu Server – це серверна операційна система, розроблена компанією Canonical, яка працює на всіх основних архітектурах: x86, x86-64, ARM v7, ARM64, POWER8 та IBM System z mainframes через LinuxONE. Ubuntu - це серверна платформа, яку кожен може використовувати для багатьох потреб, зокрема для:

– Веб-сайтів;
– FTP-серверів;

– Серверів електронної пошти;
– Серверів друку;
– Платформ розробки;
– Розгортання контейнерів;
– Хмарних послуги;
– Серверів баз даних та ін.

Переваги використання операційної системи Ubuntu Server:

– Ubuntu є безкоштовною – це проект з відкритим вихідним кодом (Open Source);
– Ubuntu є невибагливою до системних ресурсів – мінімальні вимоги системи: 2 Гб оперативної пам’яті, 20 Гб сховища та 2 ГГц процесор;
– Ubuntu Server за замовчуванням не має графічного інтерфейсу, а тому є дуже ефективною у споживанні системних ресурсів;
– Ubuntu є досить стабільною і відмовостійкою системою;
– Ubuntu має величезну спільноту користувачів і розробників – в разі виникнення складностей можна звернутись до інтернет-джерел і легко знайти потрібне рішення.

Для розгортання системи буде використано таку апаратну конфігурацію:

– Чотири-ядерний процесор Intel Core i5-3570 (3,8 GHz);
– 16 Гб оперативної пам’яті;
– Твердотілий накопичувач SSD на 250 Гб.

2.3. Розгортання хостової операційної системи

Перш за все необхідно встановити хостову операційну систему на сервер. Операційну систему для встановлення було обрано у п. 2.2 – Ubuntu Server 18.04.3 LTS. На момент написання роботи, цей дистрибутив є найновішою стабільною версії операційної системи Ubuntu Server з подовженим терміном підтримки.

Завантажимо образ для встановлення системи з офіційного веб-сайту https://ubuntu.com/download/server/. Операційну систему буде встановлено з USB-накопичувача, тому для запису образу на USB-носій нам також знадобиться спеціалізоване програмне забезпечення і USB-носій об’ємом не менше 1 Гб. Для запису операційної використаймо безкоштовне програмне забезпечення під назвою Win32 Disk Imager. Для завантаження переходимо за посиланням https://sourceforge.net/projects/win32diskimager/. Після завантаження встановлюємо і запускаємо програму.

У відкритому вікні програми біля напису «Image File» обираємо шлях до завантаженого образу операційної системи, під’єднуємо USB-носій і біля напису «Device» обираємо його із випадаючого списку і натискаємо кнопку «Write» (рис. 2.1).

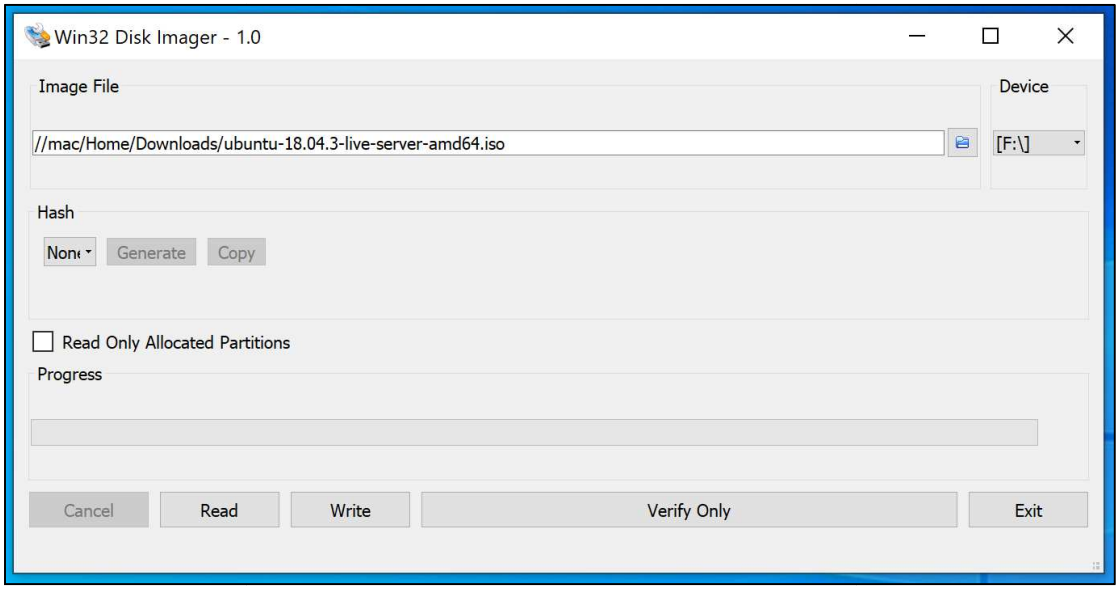

Рис. 2.1. Вікно програми Win32 Disk Imager

Після цього почнеться запис образу операційної системи на USB-носій в якості Live-USB. Live-USB – це USB-флеш накопичувач, або зовнішній твердий диск, що містить повну операційну систему, з якої можна завантажити систему або встановити.

Вставляємо флешку в комп’ютер, в меню BIOS обираємо завантаження системи з USB-носія. Слідуючи інструкціям майстра встановлення виконуємо встановлення операційної системи.

Коли операційну систему встановлено, переходимо до встановлення Docker Engine. Для цього скористаємось інструкцією з офіційної документації Docker (https://docs.docker.com/install/linux/docker-ce/ubuntu/).

1. Оновлюємо індекс пакетів apt:

```
$ sudo apt-get update
```

2. Встановлюємо необхідні пакети:

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
```

```
software-properties-common
```

3. Додаємо в систему офіційний GPG-ключ Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add –
```

4. Додаємо в систему репозиторій Docker гілки stable:

```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

5. Повторно оновлюємо індекст пакетів (п.1);
6. Встановлюємо останню версію Docker Engine Community та conainerd:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

7. Для того, щоб не запускати команди Docker від імені суперкористувача (root), оскільки запуск контейнерів від імені користувача root є потенційно небезпечним, додаємо поточного користувача до групи docker:

```
$ sudo usermod –aG docker $USER
```

На цьому маємо розгорнуту хостову операційну систему, встановлений Docker і готове до розгортання нових систем середовище.

2.4. Проектування й розробка вразливих систем

Найбільш важливою частиною платформи є власне вразливі системи, які б містили загальновідомі вразливості чи слабкості безпеки, при цьому повторюючи конфігурації та сценарії роботи реально існуючих систем.

Буде розгорнуто три вразливі системи – дві з них для демонстрації та експлуатації вразливостей веб-додатків і одну для демонстрації та експлуатації вразливостей серверної інфраструктури.

2.4.1. Розгортання системи, вразливої до SQL-Injection

SQL ін'єкція — один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного коду SQL.

Впровадження SQL, залежно від типу СУБД та умов ін’єкції, може дати можливість атакуючому виконати довільний запит до бази даних (наприклад, прочитати вміст будь-яких таблиць, видалити, змінити або додати дані), отримати можливість читання та/або запису локальних файлів та виконання довільних команд на сервері. Атака такого типу може бути можлива за некоректної обробки вхідних даних, що використовуються в SQL-запитах.

Вразливість SQL Injection у списку CWE має ідентифікатор CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

Згідно з CWE вразливість може порушувати конфіденційність, цілісність, доступність та контролі доступу. Вразливість типу SQL Injection має високий рівень імовірності експлуатації.

Для розгортання системи, вразливої до атаки типу SQL Injection, нам знадобиться створити веб-додаток та розгорнути базу даних, до якої зможе підключатись веб-додаток.

Для написання веб-додатку буде використано мови HTML та PHP, а для створення бази даних – MySQL.

Переваги використання мови PHP:

- Це швидка та проста серверна мова скриптів, що використовується у веб-розробці;
- Має велику спільноту користувачів, отже більшість проблем, з якими стикається веб-розробник, мають існуючі рішення;
- Добре працює з базами даних, файловими системами, зображеннями і т.п.;
- Має підтримку регулярних виразів (RegExp – Regular Expression);
- Має широку підтримку баз даних, в стандартній бібліотеці має вбудовану підтримку SQLite;

- Має вбудований CLI (Command Line Interface) інтерпретатор, що дозволяє з легкістю писати Shell-скрипти на PHP;
- Простий синтаксис, подібний до мов програмування на основі C;
- Є інтерпретованою мовою, тому цикл побудови дуже короткий;
- Добре працює на операційних системах *nix, Windows і на основі майже будь-якого веб-серверу (Apache, nginx, IIS і т.п.);
- Випускається під ліцензією PHP, яка дуже подібна до BSD і є безкоштовною.

Переваги використання MySQL:

- Безпека даних. MySQL всесвітньо відомий тим, що є найбільш захищеною та надійною системою управління базами даних, яка використовується у популярних веб-додатках, таких як WordPress, Drupal, Joomla, Facebook та Twitter.
- Масштабованість. MySQL пропонує неперевершену масштабованість, щоб полегшити управління глибоко вбудованими додатками. Гнучкість на вимогу - головна особливість MySQL. Це рішення з відкритим кодом дозволяє повне налаштування для електронної комерції з мінімальними вимогами до сервера баз даних.
- Висока продуктивність. MySQL оснащений чіткою основою системи зберігання даних, яка полегшує системним адміністраторам налаштування сервера баз даних MySQL для бездоганної продуктивності. Будь то веб-сайт електронної комерції, який щодня отримує мільйон запитів, або високошвидкісна система обробки транзакцій, MySQL призначений для задоволення навіть найвибагливіших програм, забезпечуючи оптимальну швидкість, повнотекстові індекси та унікальні кеші пам'яті для підвищення продуктивності.
- Цілодобовий час роботи. MySQL має гарантію тривалості роботи 24/7 і пропонує широкий спектр рішень з високою доступністю, таких як

спеціалізовані кластерні сервери та конфігурації головного/підлеглого елементів.

– Повний контроль робочого процесу. З середнім часом завантаження та встановлення менше 30 хвилин, MySQL означає зручність використання з першого дня. Незалежно від того, чи є ваша платформа Linux, Microsoft, Macintosh чи UNIX, MySQL - це комплексне рішення з функціями самоуправління, яке автоматизує все, починаючи з розширення та конфігурації простору до проектування даних та адміністрування баз даних.
– Гнучкість відкритого коду. Безпечна обробка та надійне програмне забезпечення MySQL поєднуються для забезпечення ефективних транзакцій для проектів великого об'єму. Це робить технічне обслуговування, налагодження та оновлення швидким та легким, покращуючи при цьому досвід кінцевого користувача.

Отже, мова PHP чудово підходить для створення веб-додатку, який підключатиметься до бази даних MySQL.

Оскільки система матиме два елементи – веб-сервер і базу даних, будемо використовувати Docker Compose для створення зв’язки з двох контейнерів.

Для основи контейнеру бази даних використаємо образ найновішої версії MySQL.

Спочатку, створимо скрипт, який буде створювати базу даних, таблицю, заносити в неї необхідні дані, створювати користувачів та надавати їм необхідні права.

Створюємо базу даних, яку називаємо emailbook (веб-додаток буде являти собою довідник адрес електронної пошти, де за іменем користувача можна отримати адресу електронної пошти). Після створення переходимо у базу даних:

```
CREATE DATABASE emailbook;
USE emailbook;
```

Створюємо основну таблицю, яка буде містити такі поля, як ідентифікатор, ім’я користувача, електронна адреса користувача, та прапорець:

```
CREATE TABLE IF NOT EXISTS core
(id int,
username varchar(255),
email varchar(255),
flag varchar(255));
```

У новостворену таблицю заносимо записи, які і слугуватимуть наповненням бази даних і зможуть надавати адресу електронної пошти за запитом імені користувача. Оскільки система буде використовуватись як елемент змагання CTF, у полі FLAG сховаємо прапорець, який буде свідчити про виконане учасником завдання:

```
INSERT INTO core(id,username,email,flag) VALUES
(0,'notadmin','notadmin@hid.ctf','0'),
(774,'emma','emma@hid.ctf','0'),
(381,'mason','mason@hid.ctf','0'),
(220,'mia','mia@hid.ctf','0'),
(691,'zzadmin','zzadmin@hid.ctf','FLAG{SQLInjecti0n}'),
(404,'william','william@hid.ctf','0'),
(619,'anna','anna@hid.ctf','0'),
(841,'alex','alex@hid.ctf','0');
```

Створюємо користувача, за допомогою якого веб-додаток буде підключатись до бази даних та задаємо пароль. Для того, щоб веб-додаток міг використовувати цього користувача для підключення до бази даних віддалено, а не тільки з локального серверу, створюємо користувача двічі – для localhost та для будь-яких адрес («%»):

```
CREATE USER 'ctf'@'localhost' IDENTIFIED WITH
mysql_native_password BY '73292d1fe6c43a39a59b4e9fb99163ca';
CREATE USER 'ctf'@'%' IDENTIFIED WITH mysql_native_password BY
'73292d1fe6c43a39a59b4e9fb99163ca';
```

Після створення користувача надаємо йому права на читання з бази «emailbook». Оскільки користувачів два (для локального серверу та віддалених), то повторюємо цю процедуру також двічі:

```
GRANT SELECT ON emailbook.* TO 'ctf'@'localhost';
GRANT SELECT ON emailbook.* TO 'ctf'@'%';
```

Оновлюємо права користувачів командою:

```
FLUSH PRIVILEGES;
```

Збережемо скрипт під назвою init.sql.

Для того, щоб PHP веб-сервер мав змогу працювати з базою даних MySQL у конфігураційному файлі php.ini розкоментуємо наступний рядок, щоб він мав вигляд:

```
extension=mysqli
```

Створимо файл index.php, який і буде основною веб-сторінкою нашого веб-додатку. Для початку задаємо значення основних змінних. Оскільки для запуску зв’язки контейнерів ми будемо використовувати docker-compose, немає необхідності задавати IP-адресу сервера бази даних, ми задаємо лише ім’я контейнера, який згодом вкажемо при створенні конфігураційного файлу для Docker Compose. Ім’я користувача, пароль та назву бази даних вкажемо такі, які були використані в скрипті init.sql:

```
$servername = "db";
$username = "ctf";
$password = "73292d1fe6c43a39a59b4e9fb99163ca";
$dbname = "emailbook";
```

Створимо підключення до бази даних, використовуючи значення імені серверу, користувача, паролю та бази даних, вказаних на попередньому кроці. Якщо підключення буде неуспішним, виводимо текст помилки на сторінку:

```
$link = mysqli_connect($servername,$username,$password,$dbname);
if (!$link) {
   echo "Connection error: " . mysqli_connect_error() . PHP_EOL;
   exit;
   }
```

Створюємо запит до бази даних, в якому використовуємо значення параметру user, яке отримуємо від користувача веб-додатку (форма відправлення параметру буде описана згодом). В цьому запиті вказуємо базі даних вибрати поля імені користувача та електронну адресу з таблиці core бази даних emailbook, де значення імені користувача еквівалентне введеному і обмежуємо кількість записів у відповіді до одного (першого):

```
$query = "SELECT username,email FROM emailbook.core WHERE
username='" . $_POST["usr"] . "' LIMIT 0,1;";
```

Результат запиту записуємо у масив і якщо масив містить кількість елементів більшу за нуль, виводимо на сторінку результати виконання запиту базою даних:

```
$result = $link->query($query);
if ($result->num_rows > 0) {
   $row = $result->fetch_assoc();
   echo '<P class="mb-5">' . $row["username"] . " has email: " .
$row["email"] . "</P>";
}
```

Тепер, коли механізм обробки запитів до бази даних створено, переходимо до створення форми, за допомогою якої користувачі веб-додатку зможуть виконувати запити та отримувати у відповідь результат.

Створюємо просту форму, яка відправлятиме запит POST з параметром ‘usr’ на сторінку index.php (до самої себе) за допомогою кнопки:

```
<form action="#" method="post">
<div>
<input name="usr" type="text" placeholder="Enter username here..."
required/>
</div>
<button type="submit">Send</button>
```

На цьому веб-додаток підключений до бази даних, виконує запити і виводить результат на сторінку. При цьому він має такий вигляд (рис 2.2):

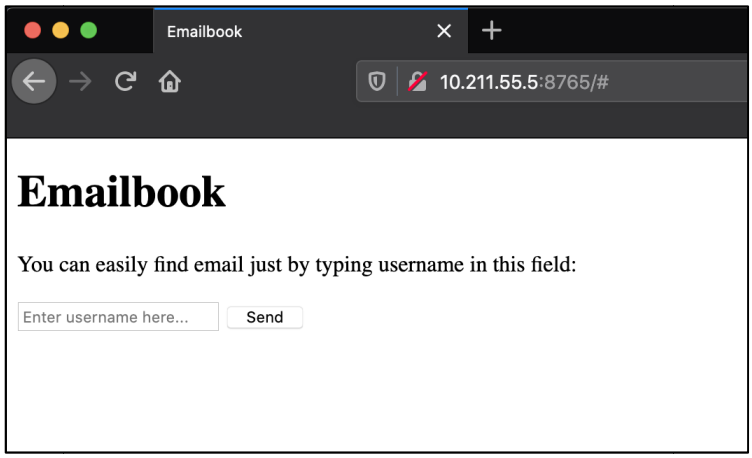

Рис. 2.2. Інтерфейс веб-додатку

Для того, щоб наблизити вигляд веб-додатку до реальних продуктивних систем, треба додати CSS стилі до веб-сторінки.

CSS (абревіатура від Cascading Style Sheets, що в перекладі означає каскадні таблиці стилів) - це спеціальна мова (мова стилів), за допомогою якої описують вигляду документів (як і де відображати елементи веб-сторінки), написаних мовами розмітки даних. Найчастіше CSS використовується для документів, котрі розмічені мовою HTML, XHTML та XML.

Звернемось до мережі Інтернет і завантажимо готовий шаблон. Додамо CSS та JavaScript елементи шаблону до створеної веб-сторінки. Повний вихідний код файлу index.php у Додатку А. Після додавання стилів сторінка має вигляд реального продуктивного веб-сайту в мережі Інтернет (рис. 2.3):

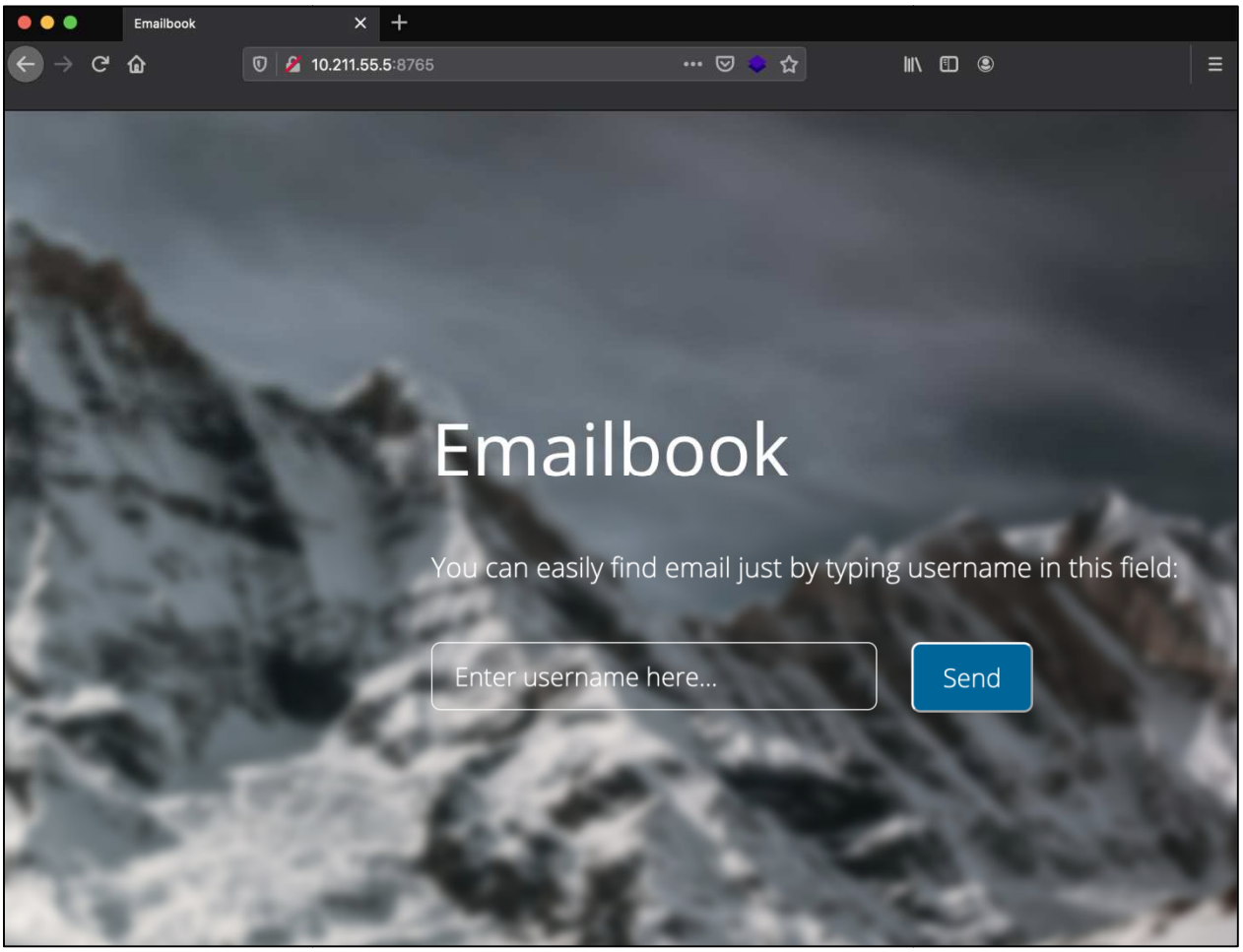

Рис. 2.3. Інтерфейс веб-додатку після застосування стилів

Коли всі компоненти системи готові, переходимо до створення docker-compose.yml файлу.

Першим рядком вказуємо версію docker-compose файлу:

```
version: '3'
```

Описуємо сервіси, які будуть створені і запущені за допомогою цього docker-compose файлу. Це будуть сервіси бази даних і веб-серверу:

```
services:
  db:

  php:
```

Для кожного сервісу вкажемо назву образу, який буде взято за основу:

```
services:
  db:
   image: mysql:latest

  php:
   image: php:7-apache
```

За допомогою container_name встановимо ім’я, яке контейнер матиме після створення:

```
services:
  db:
   image: mysql:latest
   container_name: sqli_database
  php:
   image: php:7-apache
   container_name: sqli_webserver
```

Вказуємо імена файлів та директорій локального серверу, які контейнер буде використовувати. Тобто їх можна буде змінювати динамічно, без потреби входу в контейнер:

```
services:
  db:
   image: mysql:latest
   container_name: sqli_database
   volumes:
    - ./mysql/my.cnf:/etc/mysql/my.cnf
  php:
   image: php:7-apache
   container_name: sqli_webserver
   volumes:
    - ./php/src/:/var/www/html/
    - ./php/php.ini:/usr/local/etc/php/php.ini
    - ./php/logs/:/var/log/apache2/
```

Для контейнеру бази даних задамо змінну середовища – пароль користувача root бази даних MySQL:

```
services:
  db:
   image: mysql:latest
   container_name: sqli_database
   volumes:
    - ./mysql/my.cnf:/etc/mysql/my.cnf
   environment:
    MYSQL_ROOT_PASSWORD: 863bb2b730b7fb2d2a4a6214906d6dad
  php:
   image: php:7-apache
   container_name: sqli_webserver
   volumes:
    - ./php/src/:/var/www/html/
    - ./php/php.ini:/usr/local/etc/php/php.ini
    - ./php/logs/:/var/log/apache2/
```

Для контейнера веб-серверу вкажемо порт, який буде «прокинуто» на зовнішній сервер і встановимо значення links, завдяки якому веб-сервер буде запускатись лише після готовності бази даних:

```
services:
  db:
   image: mysql:latest
   container_name: sqli_database
   volumes:
    - ./mysql/my.cnf:/etc/mysql/my.cnf
   environment:
    MYSQL_ROOT_PASSWORD: 863bb2b730b7fb2d2a4a6214906d6dad
  php:
   image: php:7-apache
   container_name: sqli_webserver
   volumes:
    - ./php/src/:/var/www/html/
    - ./php/php.ini:/usr/local/etc/php/php.ini
    - ./php/logs/:/var/log/apache2/
   ports:
    - 8765:80
   links:
    - db
```

Для обох контейнерів встановимо директиву restart:always, завдяки якій контейнери будуть самостійно перезапускатись у випадку раптової зупинки внаслідок помилки. Врешті-решт, файл docker-compose.yml матиме наступний вигляд:

```
version: '3'
services:
  db:
     image: mysql:latest
     container_name: sqli_database
     volumes:
       - ./mysql/my.cnf:/etc/mysql/my.cnf
     environment:
       MYSQL_ROOT_PASSWORD: 863bb2b730b7fb2d2a4a6214906d6dad
     restart: always
  php:
     image: php:7-apache
     container_name: sqli_webserver
     ports:
       - 8765:80
     restart: always
     links:
       - db
     volumes:
       - ./php/src/:/var/www/html/
       - ./php/php.ini:/usr/local/etc/php/php.ini
       - ./php/logs/:/var/log/apache2/
```

За допомогою команди docker-compose up запускаємо побудову контейнерів. Docker автоматично завантажить необхідні образи з репозиторію, виконає всі налаштування, вказані у конфігураційному файлі docker-compose.yml. Після успішного розгортання контейнерів виконаємо команду docker ps, щоб побачити запущені контейнери (рис. 2.4.).

```
ds@ubuntu_ctf:~/sqli$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
5d57366ba0bb        php:7-apache        "docker-php-entrypoi…"   57 minutes ago      Up 2 seconds        0.0.0.0:8765->80/tcp     sqli_webserver
936308098d4e        mysql:latest        "docker-entrypoint.s…"   57 minutes ago      Up 3 seconds        3306/tcp, 33060/tcp      sqli_database
```

Рис. 2.4. Список запущених контейнерів

Вразливість даної системи полягає в тому, що веб-сторінка приймає ввід користувача, як абсолютно нешкідливий і передає його в запит до бази даних. Наприклад, якщо в поле вводу ввести emma, то веб-додаток виконає запит:

```
SELECT username,email FROM emailbook.core WHERE username='emma' LIMIT 0,1;
```

І у відповідь від бази даних ми отримаємо:

```
emma, emma@hid.ctf
```

Але якщо користувач у поле вводу введе команди SQL, а не ім’я користувача, то веб-додаток все одно ідентифікує ці дані як легітимні і передасть до бази даних. Наприклад, якщо ввести ‘ or 1=1 -- , запит буде виконано, оскільки оператор OR повертає true, коли хоча б один з елементів true, а один завжди дорівнює одному, тому результат операції буде true. Символ «-» написаний двічі означає, що все, що йде після нього має розцінюватись як коментар. Тож запит матиме наступний вигляд:

```
SELECT username,email FROM emailbook.core WHERE username='' OR 1=1
-- LIMIT 0,1;
```

База даних у відповідь на такий запит поверне значення полів username і email з таблиці core, але лише в тих випадках, де 1=1. А оскільки один завжди дорівнює одному, база даних поверне у відповідь всі записи з таблиці. При цьому обмеження на повернення лише першого запису (LIMIT 0,1) не береться до уваги, оскільки база даних вважає це коментарем.

Для уникнення такого типу ін’єкції варто перевіряти ввід користувача на наявність в ньому заборонених символів (blacklist) та використовувати підготовлені запити (Preparement Statements).

2.4.2. Розгортання системи, вразливої до SambaCry

Вразливість SambaCry (CVE-2017-7494) – критична вразливість, що була знайдена в сервісі Samba (Linux). Ця вразливість призводить до віддаленого виконання коду (RCE) з правами суперкористувача. Зловмисний клієнт, маючи доступ до спільної директорії Samba з правами на запис, в результаті експлуатації вразливості може виконати довільний код як користувач root. На момент виявлення вразливості (2017 рік), вразливими було визначено всі версії Samba, починаючи з версії 3.5.0, що була випущена ще в 2010 році, до 4.6.4.

У списку CWE вразливість SambaCry має ідентифікатор CWE-94: Improper Control of Generation of Code ('Code Injection'), тобто ін’єкція довільного коду.

Щоб розгорнути систему, вразливу до SambaCry, необхідно створити контейнер Docker, в якому за основу взяти Linux дистрибутив і встановити на нього вразливу версію Samba.

Для цього створимо Dockerfile. Dockerfile – це текстовий файл, написаний у простому для розуміння синтаксисі, що включає в себе інструкції зі створення образу Docker (Docker image). Dockerfile визначає операційну систему, яка буде лежати в основі контейнера, а також мови, змінні середовища, місця розташування файлів, порти мережі та інші необхідні йому компоненти, і, звичайно, те, що контейнер насправді буде робити, коли ми запустимо його.

За основу буде взято дистрибутив Debian версії Jessie. Переваги використання Debian:

- Легке встановлення;
- Високий рівень інтеграції пакетів;
- Відкритий вихідний код;
- Безкоштовний дистрибутив;
- Просте оновлення і встановлення пакетів;
- Стабільність;
- Безпека системи;
- Підтримка багатьох архітектур.

Створюємо текстовий файл без розширення з назвою Dockerfile. Записуємо в нього основу майбутнього контейнера – операційну систему, яку буде взято за основу:

```
FROM debian:jessie
```

Оновлюємо репозиторії пакетів apt і встановлюємо необхідні нам утиліти:

```
RUN apt-get install wget build-essential libwbclient-dev python python-dev python-pip -y
RUN pip install python-config
```

З файлового серверу Samba завантажуємо необхідну нам версію Samba, розархівовуємо її, компілюємо і встановлюємо:

```
RUN wget https://ftp.samba.org/pub/samba/stable/samba-3.5.0.tar.gz -O /tmp/samba.tar.gz
RUN tar -zxvf /tmp/samba.tar.gz --directory /tmp/
RUN cd /tmp/samba-3.5.0/source3/ && ./configure --without-ads --without-ldap
RUN cd /tmp/samba-3.5.0/source3/ && make
RUN cd /tmp/samba-3.5.0/source3/ && make install
```

Створюємо користувача і встановлюємо для нього пароль:

```
RUN adduser --quiet --home /home/ctf --disabled-password --gecos "ctf" ctf
RUN echo "ctf:60b9afe48cd6a2a3ea8a043fd718cd4f" | chpasswd
```

Створюємо директорію для Samba, для якої буде відкрито спільний доступ і встановлюємо відповідні права для цієї директорії:

```
RUN mkdir -p /home/ctf/shared
RUN chmod -R a+rwx /home/ctf/shared
```

Копіюємо файл конфігурації з хостового серверу в контейнер:

```
COPY ./smb.conf /etc/samba/smb.conf
COPY ./smb.conf /usr/local/samba/lib/smb.conf
```

Створюємо користувача Samba з іменем ctf, встановлюємо для нього пароль і активуємо користувача:

```
RUN (echo a7a3b43f56d9caeb0c959b24f71153d8; echo a7a3b43f56d9caeb0c959b24f71153d8) | /usr/local/samba/bin/smbpasswd -a -s ctf
RUN /usr/local/samba/bin/smbpasswd -e ctf
```

Оскільки система буде використовуватись як елемент змагання CTF, створюємо директорію FLAG, в якій сховаємо прапорець, який буде свідчити про виконане учасником завдання:

```
RUN mkdir -p /tmp/FLAG
RUN echo 'FLAG{SambaCry}' > /tmp/FLAG/.flag
```

Для запуску серверу необхідно буде запускати три процеси, тому запишемо їх на хостовому сервері в файл docker-entrypoint.sh і скопіюємо на сервер:

```
COPY docker-entrypoint.sh /usr/local/bin/
RUN chmod +x /usr/local/bin/docker-entrypoint.sh
```

Робимо «проброс» портів з контейнеру на хостовий сервер:

```
EXPOSE 139
EXPOSE 445
```

І в якості процесу, з якого запускатиметься контейнер вказуємо скопійований раніше файл docker-entrypoint.sh:

```
CMD ["/usr/local/bin/docker-entrypoint.sh"]
```

На цьому зберігаємо Dockerfile. Вміст скрипту docker-entrypoint.sh:

```
#!/bin/sh
/usr/local/samba/sbin/smbd -D
/usr/local/samba/sbin/nmbd -D
/usr/local/samba/sbin/winbindd -D
tail -f /dev/null
```

В цей файл ми записали запуск процесів smb, nmb та winbind у фоновому режимі. Оскільки контейнер Docker зупиняється тоді, коли ініціюючий процес успішно виконано, додаємо рядок tail –f /dev/null, для того, щоб процес виконувався до тих пір, поки не буде зупинений користувачем.

Внесемо необхідні нам конфігурації у файл налаштувань Samba – smb.conf:

```
[global]
     netbios name = CRY
     workgroup = CTF
     map to guest = Bad User
```

```
        log file = /var/log/samba/%m.log
           log level = 1

[Share1]
      path = /home/ctf/shared
      browsable = yes
      writable = yes
      guest ok = yes
      read only = no
```

Реєструємо спільну директорію Share1, для якої дозволяємо права на запис для неавтентифікованих користувачів.

Побудуємо образ, виконавши команду “docker build –t samba_cry .”, де опція -t означає ім’я, яке отримає образ після побудови, а крапка означає шлях до Dockerfile, тобто він знаходиться у поточній директорії.

Після успішної побудови образу запускаємо контейнер з цього образу, використовуючи команду:

```
docker run -p 139:139 -p 445:445 -d samba_cry
```

де опція -p – це проброс портів з контейнеру на хостову операційну систему, -d – запуск контейнеру в фоновому режимі, а samba_cry – ім’я образу.

Для перевірки доступних спільних директорій Samba запустимо сканування портів nmap зі скриптом smb-enum-shares, який збере інформацію про доступні спільні директорії та права поточного користувача в цих директоріях (рис 2.5):

```
ds@mbp-ds.local:~ % nmap 10.211.55.5 -p 139,445 -sV --script smb-enum-shares
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-15 19:53 EET
Nmap scan report for ubuntu-ctf.shared (10.211.55.5)
Host is up (0.35s latency).

PORT    STATE SERVICE     VERSION
139/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: CTF)
445/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: CTF)

Host script results:
| smb-enum-shares:
|   account_used: guest
|   \\10.211.55.5\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: IPC Service (Samba 3.5.0)
|     Users: 1
|     Max Users: <unlimited>
|     Path: C:\tmp
|     Anonymous access: READ/WRITE
|     Current user access: READ/WRITE
|   \\10.211.55.5\Share1:
|     Type: STYPE_DISKTREE
|     Comment:
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\home\ctf\shared
|     Anonymous access: READ/WRITE
|_    Current user access: READ/WRITE
```

Рис. 2.5. Результат сканування утилітою nmap

Як бачимо, на спільній директорії Share1 активовано права на читання та запис для анонімного користувача. Це означає, що її можливо проексплуатувати для отримання віддаленого доступу до системи.

24 травня 2017 року було випущено патч до всіх вразливих версій Samba, тому для закриття цієї вразливості необхідно оновити пакет Samba до найновішої версії. Зробити це можна за допомогою вбудованого менеджера пакетів apt:

```
sudo apt update && sudo apt upgrade –y
```

2.4.3. Розгортання системи, вразливої до Sensitive Infotmation Insecure Storage

Часто трапляється, що розробники використовують кодування інформації замість шифрування, вважаючи, що це безпечно. Насправді, кодування інформації –

це лише представлення символів одного алфавіту або системи символів у вигляді іншого, тобто без використання секретного елементу. Таким чином кодування є зворотнім і зловмисник може без проблем отримати вихідне повідомлення лише знаючи, яке кодування використовувалось.

У списку CWE вразливість небезпечного зберігання чутливої інформації має ідентифікатор CWE-922: Insecure Storage of Sensitive Information, а проблеми криптографії охоплюють цілу категорію CWE-310: Cryptographic Issues.

Для демонстрації вразливості буде створено простий веб-додаток, в якому за допомогою коду JavaScript буде створено таймер, який виконуватиме відлік до виконання перенаправлення на сторінку з прапорцем. Суть CTF-завдання полягає в тому, що таймер спрацьовує через проміжок часу, що більший за проміжок часу проведення змагань, тому отримати прапорець таким чином не вдасться. Оскільки JavaScript-код зберігається на клієнтській частині веб-сайту, його буде двічі обфусковано, використовуючи різні техніки. Саме таким чином веб-розробники зазвичай «захищаються» від зловмисників.

Для початку, створимо скрипт, який кожні півтори секунди збільшуватиме відсоток завантаження на 0.01.

Створюємо функцію tak(), яка кожні півтори секунди записуватиме значення відсотку у елемент HTML-сторінки з іменем tik і збільшуватиме інкремент на одиницю. При досягненні інкрементом значення 10001 (через 250 хвилин), в браузері клієнта буде встановлено cookie з ідентифікатором secret і значенням 14128ce1cb611ea72fa28795c4d95746 і перенаправлено на сторінку з прапорцем:

```
var i = 1;
function tak () {
 setTimeout(function () {
  document.getElementById('tik').innerHTML=i/100;
  i++;
  if (i < 10001) {
   tak();
```

```
    }
   else if (i==10001) {
    document.cookie = "secret=14128ce1cb611ea72fa28795c4d95746";
window.location.href="/709f88a6ba23afd54c777cf2ef389c64/flag.php";
    }
 }, 1500)
}
tak();
```

Створимо директорію 709f88a6ba23afd54c777cf2ef389c64, в якій буде розміщено серверну частину додатку у файлі flag.php. Ім’я директорії обрано таким чином для того, щоб було неможливо її виявити методом перебору імен директорій. Файл flag.php буде написаний мовою PHP і міститиме код, який у разі отримання від користувача запита зі встановленою cookie повертатиме у відповідь прапорець. В разі отримання запиту без встановленої cookie, на сторінку виводиться текст про те, що потрібно повернутись на головну сторінку сайту і зачекати. Вміст файлу flag.php:

```
<!DOCTYPE html>
<html>
<head>
<title>FLAG</title>
</head>
<body>
<?php
if ($_COOKIE['secret']=='14128ce1cb611ea72fa28795c4d95746') {
echo 'HID{D0_n0t_st0r3_data_1n_JS}';
} else {
echo 'Please go <a href="/">back</a> and wait!';
}
?>
</body>
</html>
```

До основної сторінки index.html додамо CSS стилів, встановимо фонове зображення для надання більш реалістичного вигляду вразливій системі:

```
body {
  background-image: url("/img/back.jpg");
  background-position: center top;
}
.center {
  margin-left: auto;
  margin-right: auto;
  width: 800px;
  background: rgba(128, 128, 128, 0.4);
  margin-top: 5%;
  font-family: 'Montserrat', sans-serif;
  font-size: 36px;
  text-transform: uppercase;
  text-align: center;
}

p {
display: inline;
}
```

Коли контент веб-додатку повністю готовий, необхідно обфускувати JavaScript код, для того, щоб зробити його нереальним для читання людиною, при цьому браузер все одно зможе його виконувати.

Обфускація або заплутування коду – приведення початкового коду або виконуваного програмного коду до вигляду, який зберігає його функціональність, але ускладнює аналіз, розуміння алгоритму роботи і модифікації при декомпіляції. [14]

Скористаймось онлайн-обфускатором JavaScript коду за методом Array. В результаті першого рівня обфускації, код виглядатиме наступним чином:

```
var
_0xf91f=["\x69\x6E\x6E\x65\x72\x48\x54\x4D\x4C","\x74\x69\x6B","\x67\x6
5\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64","\x63\x6F\x6F\x6B\x6
9\x65","\x73\x65\x63\x72\x65\x74\x3D\x31\x34\x31\x32\x38\x63\x65\x31\x6
3\x62\x36\x31\x31\x65\x61\x37\x32\x66\x61\x32\x38\x37\x39\x35\x63\x34\x
64\x39\x35\x37\x34\x36","\x68\x72\x65\x66","\x6C\x6F\x63\x61\x74\x69\x6
F\x6E","\x2F\x37\x30\x39\x66\x38\x38\x61\x36\x62\x61\x32\x33\x61\x66\x6
4\x35\x34\x63\x37\x37\x37\x63\x66\x32\x65\x66\x33\x38\x39\x63\x36\x34\x
2F\x66\x6C\x61\x67\x2E\x70\x68\x70"];var i=1;function
tak(){setTimeout(function(){document[_0xf91f[2]](_0xf91f[1])[_0xf91f[0]
]= i/ 100;i++;if(i< 10001){tak()}else {if(i==
10001){document[_0xf91f[3]]= _0xf91f[4];window[_0xf91f[6]][_0xf91f[5]]=
_0xf91f[7]}}},1500)}tak()
```

Наступним кроком скористаймось обфускацією, що має назву JSFuck. Назва походить від Brainfuck, езотеричної мови програмування, яка також використовує мінімалістичний алфавіт пунктуації. На відміну від Brainfuck, який вимагає власного компілятора чи інтерпретатора, JSFuck є дійсним кодом JavaScript, тобто програми JSFuck можна запускати в будь-якому веб-браузері або двигуні, що інтерпретує JavaScript. JSFuck здатний відтворити всю функціональність JavaScript, використовуючи такий обмежений набір символів, оскільки JavaScript є слабо типізованою мовою програмування, і це дозволяє відтворити будь-який вираз в якості будь-якого типу. В результаті такої обфускації весь код буде інтерпритовано за допомогою шести символів: “[ ] ( ) ! +”.

В результаті обфускації з використанням JSFuck, було отримано код довжиною 1 443 545 символів, а це складає 287 сторінок, тому обфускований код не буде наведено в цій роботі.

Переходимо до створення контейнеру.

Використовувати Docker Compose не обов’язково лише тоді, коли обхідно запустити два або більше контейнерів. Docker Compose можна використовувати ще й для того, щоб контейнер використовував спільну директорію з хостовим сервером.

Таким чином, ми можемо змінювати вміст веб-серверу без необхідності перебудови і перезавантаження контейнеру. Тож, створюємо docker-compose.yml файл:

```
version: '3'
services:
  php:
     image: php:7-apache
     container_name: jstimer_webserver
     ports:
       - 8123:80
     restart: always
     volumes:
       - ./src/:/var/www/html/
```

Контейнер буде створено на основі PHP з вбудованим веб-сервером Apache, з заданим ім’ям jstimer_webserver, внутрішній 80 порт буде прокинуто назовні на 8123 порт, перезапуск контейнеру налаштований автоматично у випадку непередбачених збоїв, а вміст директорії src, яка містить всі вихідні коди веб-сторінок, буде віртуалізовано у домашню директорію веб-серверу.

Командою docker-compose up побудуємо і запустимо контейнер. При переході до веб-додатку, він матиме наступний вигляд (рис. 2.6):

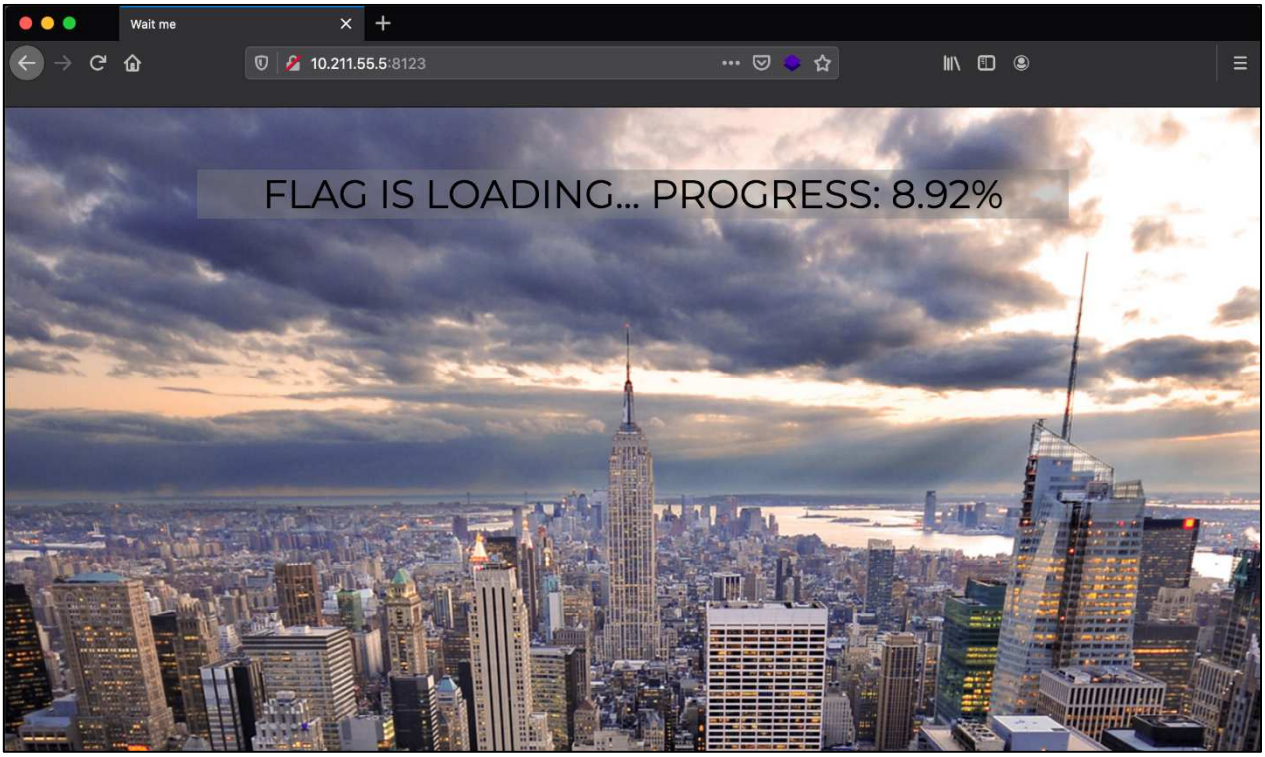

Рис. 2.6. Інтерфейс веб-додатку

Будь-який користувач веб-додатку може переглянути вихідний код, оскільки він виконується браузером на клієнтській частині. В цьому і полягає вразливість – небезпечне зберігання чутливих даних (в нашому випадку cookie, яка є ключем до отримання прапорця). До того ж обфускація не є достатнім захистом, оскільки це зворотній процес (на відміну від хешування), і будь-хто може відновити оригінальний вихідний код.

Для того, щоб уникнути вразливості такого типу, необхідно зберігати чутливі дані захищено в базах даних, використовуючи контроль доступу, хешування (якщо це паролі користувачів системи) та інші механізми захисту. І ні в якому разі не зберігати будь-яку чутливу інформацію на стороні клієнта – в файлах JavaScript, HTML або PHP.

2.5. Розгортання контрольної панелі CTF

Контрольна палень (або дашборд – від англ. dashboard) – елемент CTF платформи, який виконує такі функції:

- Реєстрація учасників;
- Розміщення завдань;
- Фіксація виконаних учасником завдань;
- Підрахунок балів учасників;
- Побудова турнірної таблиці змагань;
- Логування та контроль дій учасників та ін.

Дашборд потрібен в першу чергу для зручності учасників та адміністраторів змагань, для об’єднання всіх завдань в єдину систему, де учаснику автоматично будуть нараховуватись переможні бали за надсилання отриманого в результаті успішного виконання завдання прапорцю, та для візуалізації результатів учасників. Ця контрольна панель поєднує в собі функції особистого кабінету учасників, палені адміністрування змагань та турнірної таблиці.

Створення такої системи з нуля потребує дуже багато часу, професійних навичок в програмуванні та створення власної логіки роботи такої системи, тож нераціонально розробляти таку систему самостійно. На щастя, існує чимала кількість фреймворків з відкритим вихідним кодом, які здатні виконувати функції контрольної панелі CTF. Саме один з таких фреймворків, CTFd, і буде взято за основу системи.

CTFd - це фреймворк Capture The Flag, орієнтований на простоту використання та налаштування. CTFd має більшість функцій, необхідних організатору події для проведення змагань або семінару. Крім того, якщо набір функцій CTFd недостатній, CTFd дозволяє використовувати плагіни та теми для контролю майже кожного аспекту того, як він виглядає та поводиться.

CTFd використовується багатьма різними клубами, університетами та компаніями для влаштування власних Capture The Flags. CTFd написаний на Python і використовує веб-фреймворк Flask. Використовуючи Docker Compose, розгорнути систему CTFd можна без складностей. Для цього встановимо на хостовому сервері утиліту git:

```
sudo apt install git
```

Після встановлення клонуємо репозиторій CTFd з GitHub. GitHub – один з найбільших веб-сервісів для спільної розробки програмного забезпечення, що широко використовується проектами з відкритим вихідним кодом.

```
git clone https://github.com/CTFd/CTFd.git
```

Тепер ми можемо модифікувати файл docker-compose відповідно до наших потреб. Наприклад, збільшимо кількість одночасних потоків до 4. Для цього в рядку `Workers=1` замінимо значення на 4. Весь вміст файлу docker-compose.yml сервісу CTFd наведено у Додатку Б. Виконавши команду `docker-compose up` запускаємо контейнери сервісу CTFd. При налаштуваннях за замовчуванням сервіс буде запущено на порті 8000 і при першому переході до цього додатку нам буде запропоновано створити обліковий запис адміністратора на налаштувати систему (рис. 2.7):

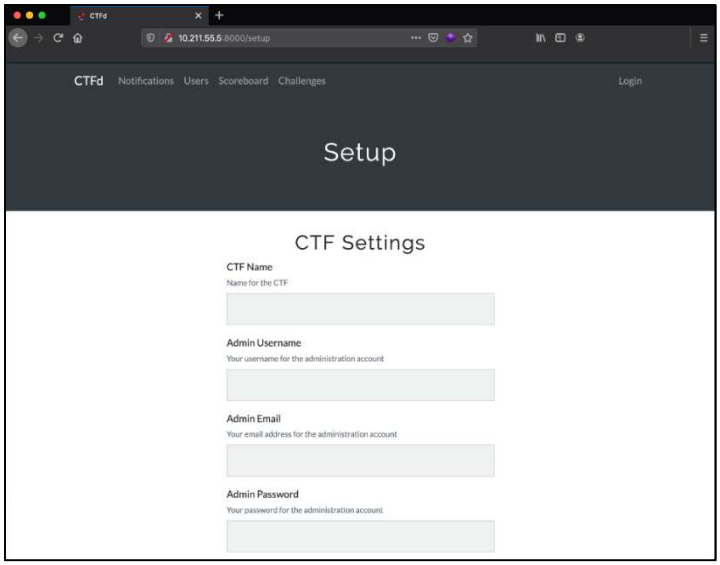

Рис. 2.7. Створення облікового запису адміністратора в системі

Після успішного створення акаунту адміністратора і налаштування основних елементів системи, отримуємо доступ до панелі керування CTF (рис. 2.8):

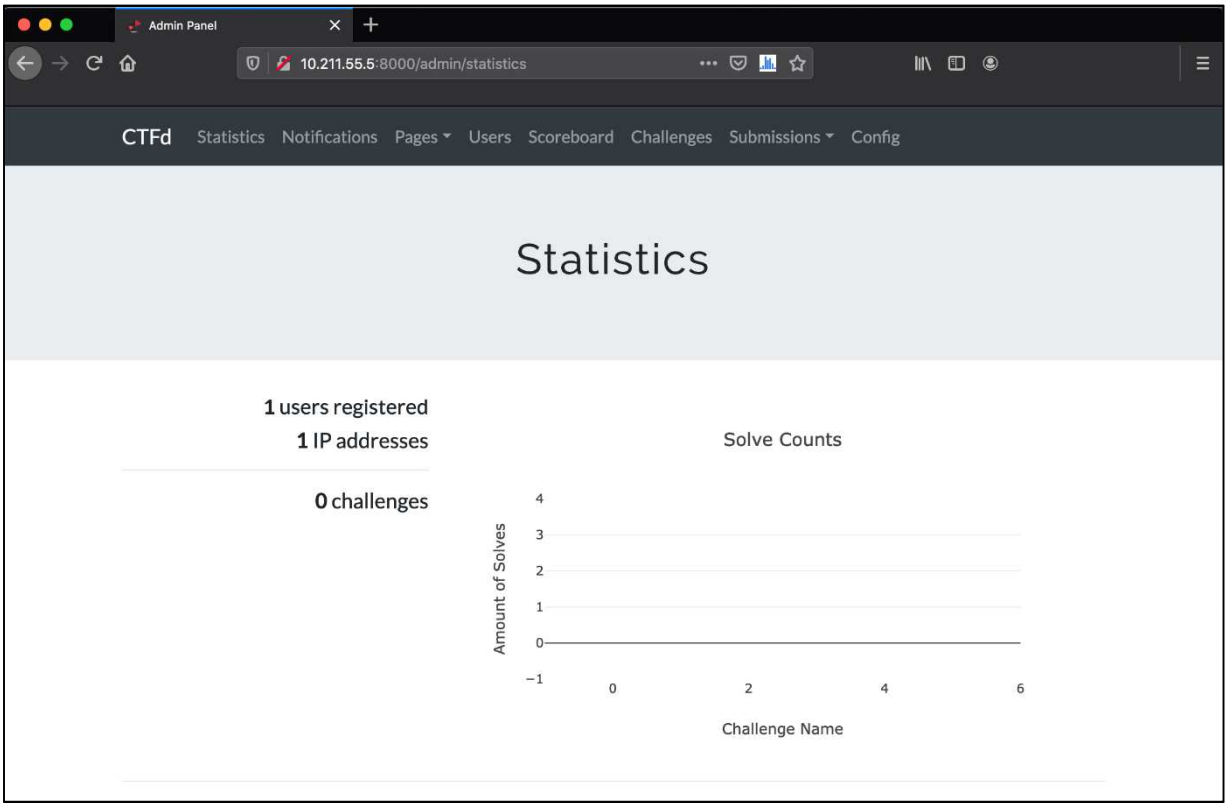

Рис. 2.8. Панель керування платформою CTFd

На цьому проектування та розгортання елементів майбутнього CTF, а саме вразливих систем та панелі керування CTF завершено.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1. Організація змагань для студентів Державного університету телекомунікацій

В рамках магістерської роботи 12 грудня 2019 року було проведено змагання в форматі CTF для студентів Державного університету телекомунікацій. Мета змагань: поширення зацікавленості темою інформаційної безпеки серед студентів університету, ознайомлення студентів з деякими типами загальновідомих вразливостей та навчання методам захисту від типових вразливостей.

Змагання було проведено за підтримки кафедри інформаційно-комунікаційної безпеки навчально-наукового інституту Державного університету телекомунікацій. В якості спонсора та співорганізатора було залучено компанію «IT Спеціаліст». Компанія «IT Спеціаліст» – це українська компанія-інтегратор, що має великий досвід проектів у сфері кібербезпеки.

Змагання отримали назву Hack’in’DUT. До складу CTF увійшло 12 завдань різних категорій, завдяки чому було охоплено основні напрями інформаційної та кібернетичної безпеки в цілому та наступальної безпеки (offensive security) зокрема:

- OSINT (Open Source Intelligence – тобто пошук і збір інформації з відкритих джерел);
- Криптографія;
- Зворотній інжиніринг (Reverse engineering);
- Мережева безпека;
- Безпека веб-додатків та ін.

Було розроблено унікальний бренд Hack’in’DUT, який має власну стилістику, логотип, гасло, рекламні постери та стікери для учасників змагань. Нижче наведено логотип змагань Hack’in’DUT CTF (рис. 3.1).

Рис. 3.1. Логотип заходу

Для реклами було у соціальних мережах було використано зображення (рис. 3.2), яке містить назву, час і місце проведення події, логотип організатора та гасло заходу:

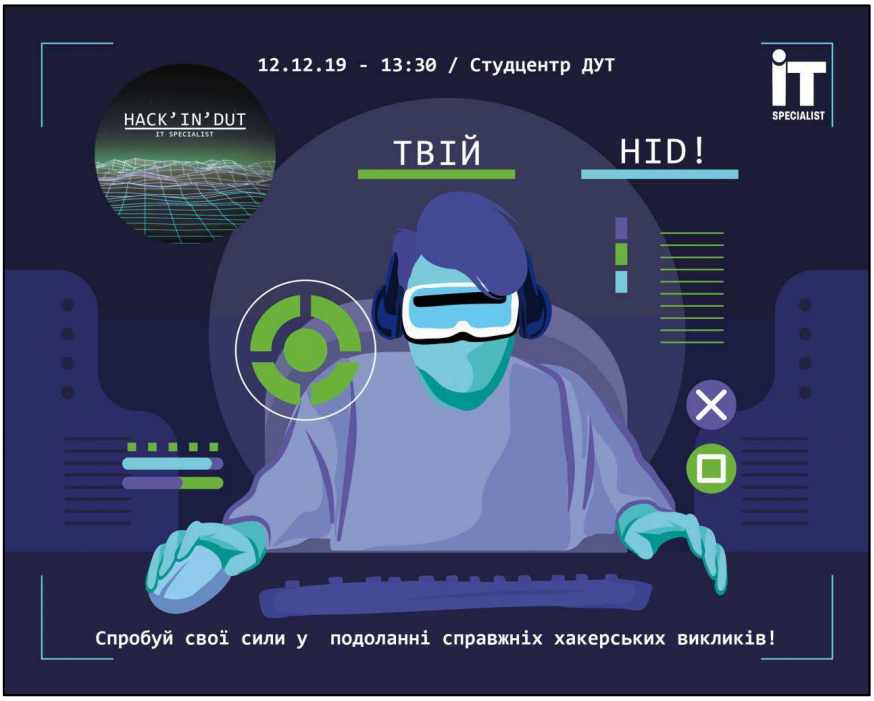

Рис. 3.2. Рекламне зображення заходу

Для рекламних постерів (рис. 3.3) було використано більше зображення, на який було додано короткий опис події, категорії завдань, які очікували на учасників, перелік нагород за призові місця та посилання на форму реєстрації у вигляді QR-коду:

Рис. 3.3. Рекламний плакат заходу

Спонсором нагород за призові місця змагань виступила компанія «IT Спеціаліст». Призовий фонд змагань складав:

– За перше місце – USB Wi-Fi адаптер Alfa AWUS1900;

- За друге місце – зовнішній жорсткий диск WD Elements об’ємом 1 ТВ;
- За третє місце – медіаплеєр Full HD Google Chromecast.

Реєстрацію було відкрито за 7 днів до заходу і за цей час було зареєстровано 41 учасник, серед яких були студенти таких технічних спеціальностей, як кібербезпека, комп’ютерна інженерія, інформаційні технології, телекомунікації.

Платформа для змагань була розгорнута, використовуючи описані у розділі 2 серверні ресурси та технології розгортання вразливих систем і панелі керування змаганнями. Для організації інфраструктури змагань було використано один сервер, який було з’єднано з Wi-Fi маршрутизатором. В панелі керування маршрутизатором було створено локальну мережу (без доступу в Інтернет), до якої учасники приєднувались за допомогою WiFi. Всі завдання змагань були доступні на одному сервері.

Система, до якої було внесено всі завдання мала наступний вигляд (рис. 3.4):

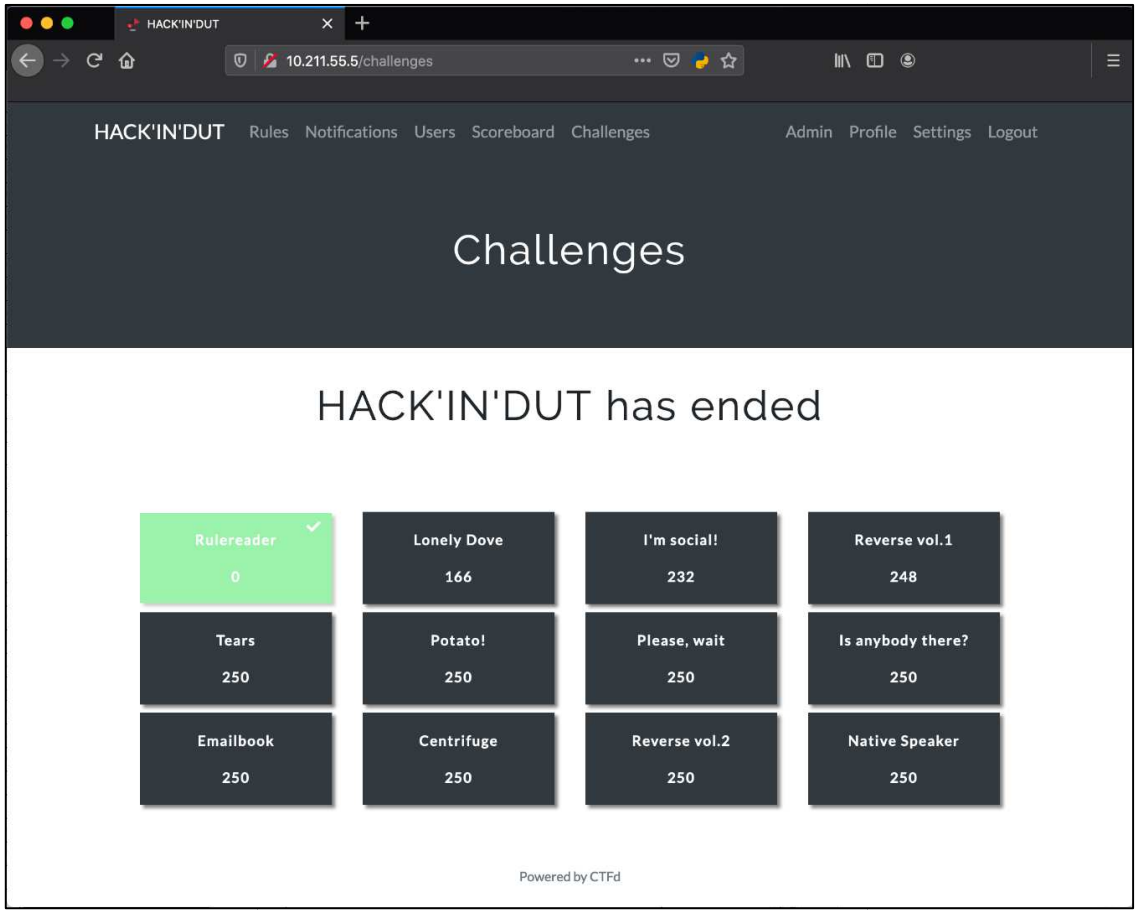

Рис. 3.4. Система CTF, до якої внесено всі завдання

Кожне завдання містило назву, короткий опис, посилання на вразливу систему та поле для введення прапорцю, який учасник отримує внаслідок успішного виконання завдання (рис. 3.5):

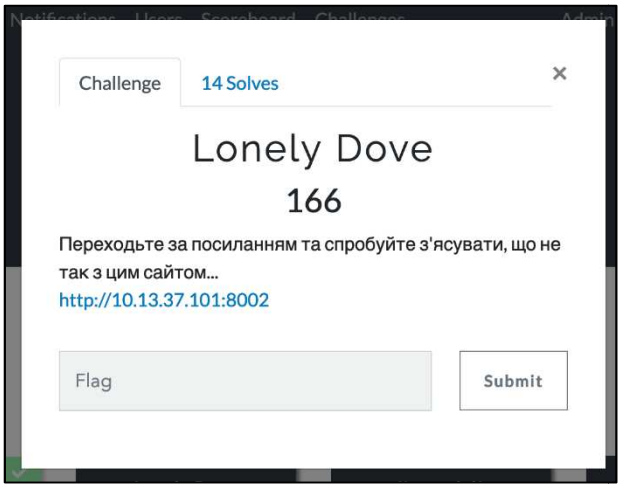

Рис. 3.5. Зовнішній вигляд завдання

3.2. Результати змагань і досягнення студентів

В день проведення змагань, в системі зареєструвався 21 учасник. Серед них 17 учасників виконали хоча б одне з завдань (рис. 3.6).

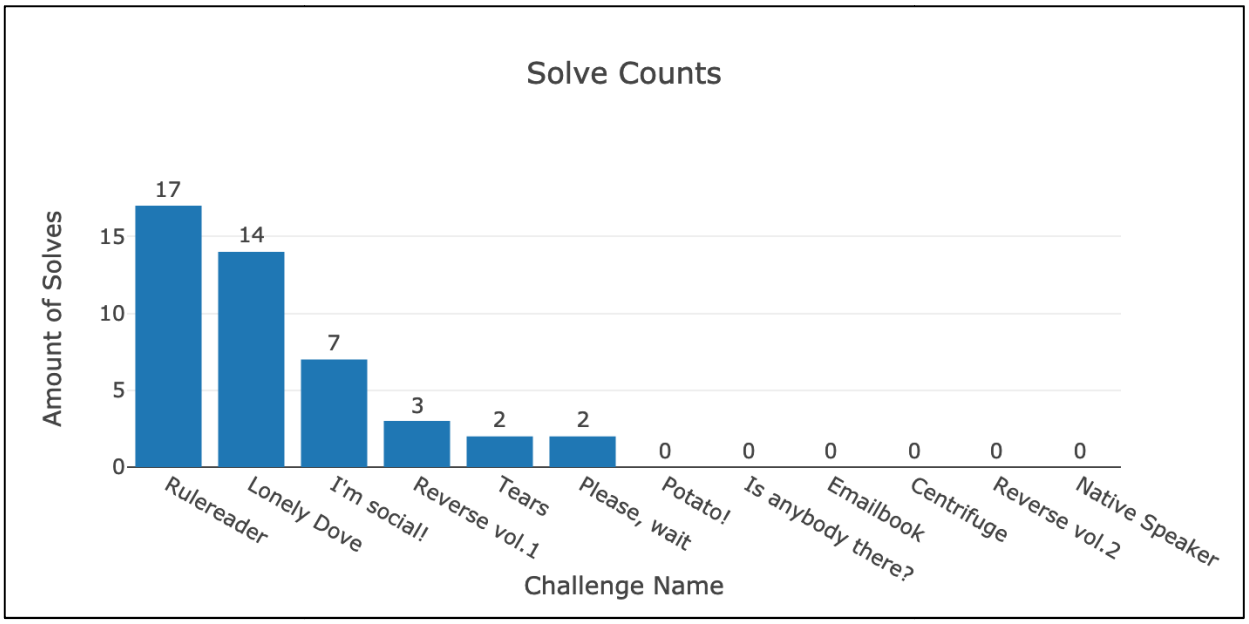

Рис. 3.6. Статистика виконаних учасниками завдань

Всього учасники зробили 73 спроби відправки прапорців на сервер, серед яких правильні вирішення завдань становили лише 63.2%, тобто 45 спроб (рис. 3.7).

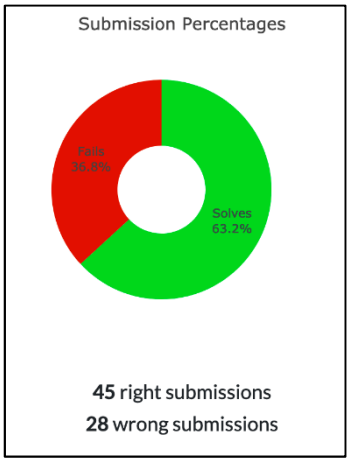

Рис. 3.7. Статистика відправлених прапорців

Слід звернути увагу на те, що в змаганнях використовувалась динамічна система оцінки. Тобто на початку, переможні бали за виконання кожного з завдань становили 250 очок. Проте, чим більша кількість учасників вирішувала одне й те саме завдання, тим меншу кількість очок отримували учасники, які вирішили це завдання.

Змагання тривали чотири години. По закінченню змагань, турнірна таблиця мала наступний вигляд (рис. 3.8):

Place	Team	Score
1	locki_g	896
2	Morronel	730
3	Morriell	648
4	uniff	648
5	MatviiYu	646
6	JB	398
7	Vitalii M.	398
8	gaposha	166
9	Xsayr	166
10	Weedx19	166

Рис. 3.8. Турнірна таблиця учасників по завершенню змагань

Переможцю вдалось вирішити чотири завдання, а учасникам, що посіли друге та третє місця – піддалось по три завдання. Учасники, що посіли перше та друге місця є студентами п’ятого курсу, а учасник, що посів третє місце є студентом другого курсу.

По завершенню змагань учасникам було запропоновано заповнити форму зворотного зв’язку, яка містила питання пов’язані з участю в CTF та навчальним процесом в університеті. Форму зворотного зв’язку заповнило 12 студентів. Всі студенти навчаються за спеціальністю кібербезпека.

Серед них 66.7% ніколи раніше не брали участі у заходах подібного формату, в тому числі й онлайн CTF (рис. 3.9):

Рис. 3.9. Відповіді учасників опитування

83.3% опитуваних вважають цей CTF цікавим і таким, що прийшовся їм до вподоби. 8.3% отримала відповідь «скоріше, так» і 8.3%, тобто один учасник відповів, що CTF йому не сподобався і не був цікавим (через складність завдань) (рис. 3.10, 3.11):

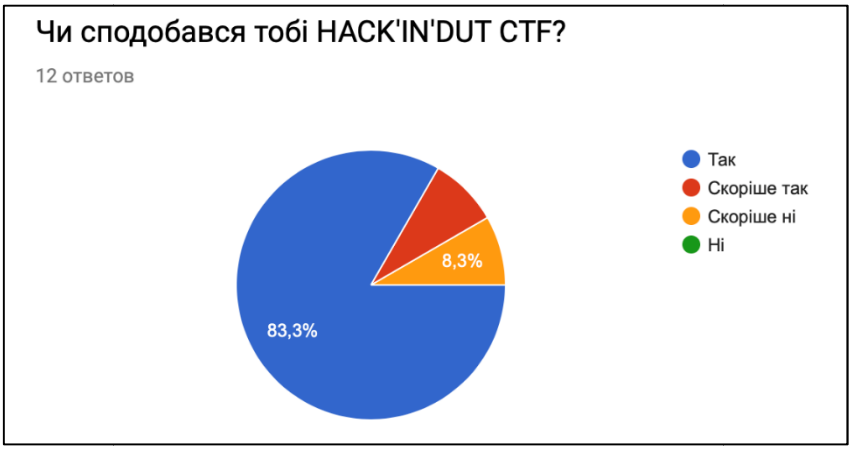

Рис. 3.10. Відповіді учасників опитування

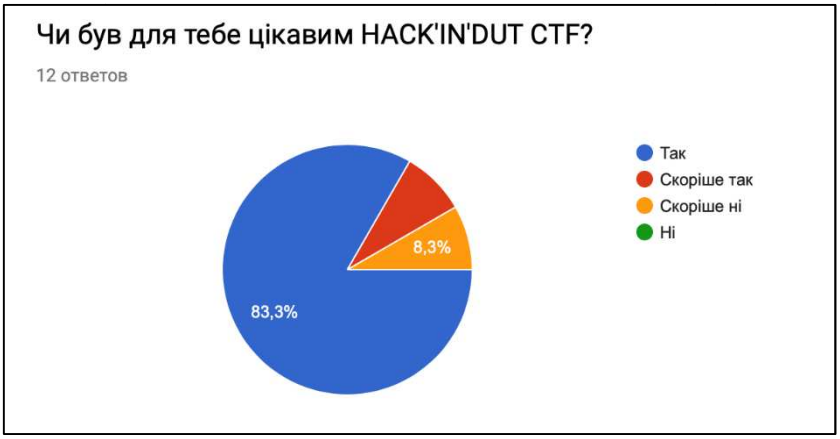

Рис. 3.11. Відповіді учасників опитування

В цілому, всім учасникам, які пройшли опитування сподобався призовий фонд змагань Hack’in’DUT CTF (рис. 3.12):

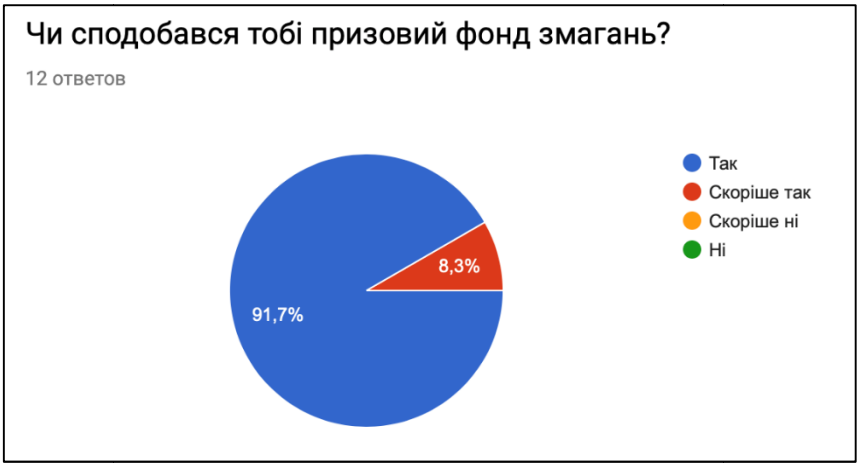

Рис. 3.12. Відповіді учасників опитування

Думки щодо оцінки учасниками складності завдань розділились рівно на три частини. З цього можемо зробити висновок, що складність завдань – вище середньої (рис. 3.13):

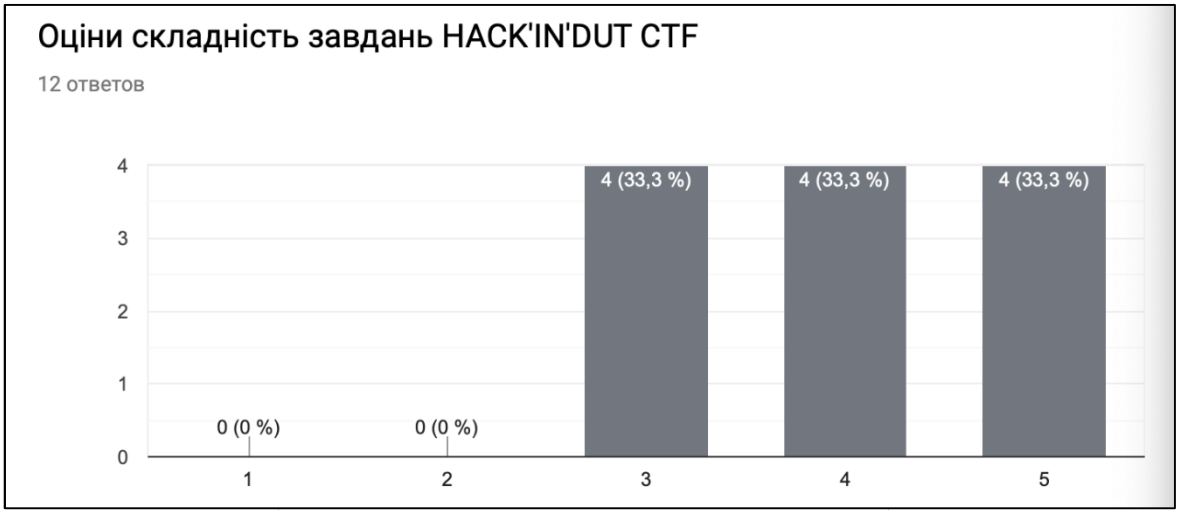

Рис. 3.13. Відповіді учасників опитування щодо складності завдань

11 з 12 учасників, що пройшли опитування відповіли, що завдяки цим змаганням змогли дізнатись чомусь новому (рис. 3.14):

Рис. 3.14. Відповіді учасників опитування

11 з 12 учасників опитування вважають, що тих практичних навичок з профільної дисципліни, які вони отримують на заняттях в університеті недостатньо (рис. 3.15):

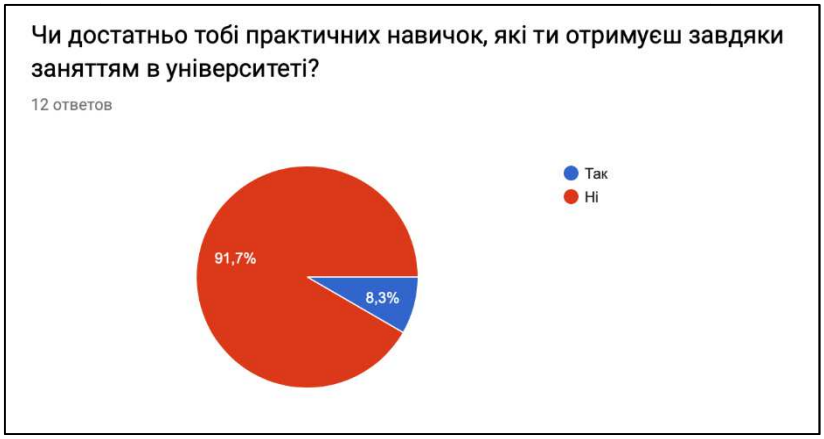

Рис. 3.15. Відповіді учасників опитування

Всі учасники, які взяли участь в опитування вважають, що такі заходи обов'язково мають відбуватись в стінах університету та вважають гарною думкою включення елементів CTF до навчальної програми профільних дисциплін з кібербезпеки (рис. 3.16, 3.17):

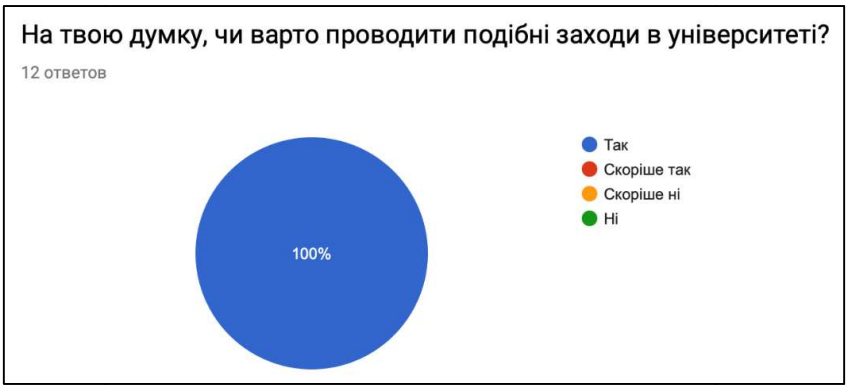

Рис. 3.16. Відповіді учасників опитування

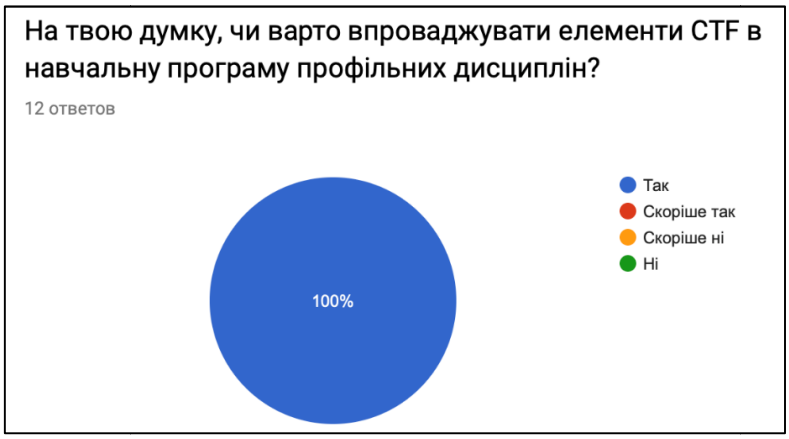

Рис. 3.17. Відповіді учасників опитування

3.3. Підсумки реалізації проекту

В результаті реалізації такого проекту студенти-учасники отримали нові знання і навички зі спеціальності кібербезпека, а зокрема дізнались про нові техніки наступальної кібербезпеки, отримали змогу застосувати свої знання на практиці, перевірити свої можливості в якості знавця інформаційної та кібернетичної безпеки.

Завдяки цьому проекту до участі було залучено студентів не тільки спеціальності кібербезпека, а й студентів інших технічних спеціальностей.

Завдяки опитуванню студентів було встановлено, що практичних навичок із профільної спеціальності, які вони отримують на заняттях в університеті, недостатньо для ретельного вивчення вразливостей та проблем безпеки сучасних систем. Саме тому, описана у другому розділі методика розробки навчальної платформи для пошуку і експлуатації загальновідомих вразливостей стане у нагоді при розробці програм занять і навіть дисциплін із залученням інноваційних методів навчання.

ВИСНОВКИ

В результаті аналізу предметної області було детально розглянуто теоретичні відомості, що стосуються тестування на проникнення, списків та класифікацій загальновідомих вразливостей та проблем безпеки, технологій віртуалізації в цілому, та контейнерної віртуалізації на прикладі Docker зокрема.

В практичній частині роботи було успішно реалізовано навчальну платформу для пошуку та експлуатації загальновідомих вразливостей та проблем безпеки за допомогою технологій контейнерної віртуалізації Docker з використанням програмного забезпечення з відкритим кодом. Після завершення розробки, платформу було використано у навчальному проекті для студентів Державного університету телекомунікацій, в результаті якого його учасники отримали практичні навички з профільних дисциплін та навчились застосовувати їх на практиці.

Було вирішено поставлені завдання:

1. Досліджено процеси та технології тестування на проникнення;
2. Проаналізовано існуючі класифікації та списки загальновідомих вразливостей та слабкостей безпеки;
3. Досліджено технології віртуалізації;
4. Використовуючи контейнерну віртуалізацію, розроблено навчальну платформу для пошуку та експлуатації загальновідомих вразливостей;
5. Застосовано розроблену платформу у навчальному проекті для студентів університету.

В результаті виконання поставлених задач, навчальну платформу для пошуку та експлуатації загальновідомих вразливостей розроблено. Платформа готова до використання при розробці сучасних профільних програм занять і дисциплін із залученням інноваційних методів навчання, що будуть орієнтовані на практичне застосування отриманих навичок. А отже, поставлену мету роботи досягнуто.

ПЕРЕЛІК ПОСИЛАНЬ

1. Тест на проникнення – Вікіпедія [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://uk.wikipedia.org/wiki/Тест_на_проникнення
2. Тест на проникнення – Захожий А.Г. [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: http://www.rusnauka.com/16_NPRT_2009/Informatica/47627.doc.htm
3. CVE – Common Vulnerabilities and Exposures [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://cve.mitre.org
4. CWE – Common Weakness Enumeration [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://cwe.mitre.org
5. What is virtualization? [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.redhat.com/en/topics/virtualization/what-is-virtualization
6. About Docker [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://docs.sevenbridges.com/docs/docker-basics
7. What is Docker? The spark for the container revolution [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html
8. Docker Documentation [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://docs.docker.com/
9. A begginer’s guide to Docker – how to create a client/server side with docker-compose [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.freecodecamp.org/news/a-beginners-guide-to-docker-how-to-create-a-client-server-side-with-docker-compose-12c8cf0ae0aa/
10. Capture the flag (CTF) – Вікіпедія [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://uk.wikipedia.org/wiki/Capture_the_flag_(CTF)

11. Cyber Security Capture The Flag (CTF): What Is It? [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://blogs.cisco.com/perspectives/cyber-security-capture-the-flag-ctf-what-is-it
12. Behind the Scenes at a Capture the Flag (CTF) Competition [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://securityintelligence.com/behind-the-scenes-at-a-capture-the-flag-ctf-competition/
13. Ubuntu Server: A cheat sheet – TechRepublic [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.techrepublic.com/article/ubuntu-server-the-smart-persons-guide/
14. Обфускація – Вікіпедія [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://uk.wikipedia.org/wiki/Обфускація
15. Home – Ubuntu Wiki [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://wiki.ubuntu.com
16. Advantages and Disadvantages of Docker – Learn Docker – DataFlair [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://data-flair.training/blogs/advantages-and-disadvantages-of-docker/
17. 8 Major Advantages of Using MySQL [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.datamation.com/storage/8-major-advantages-of-using-mysql.html
18. CWE – CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (3.4.1) [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://cwe.mitre.org/data/definitions/89.html
19. CWE – CWE-922: Insecure Storage of Sensitive Information (3.4.1) [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://cwe.mitre.org/data/definitions/922.html

20. OWASP Top Ten Project – OWASP [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
21. Що таке CSS [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://css.in.ua/article/shcho-take-html_10
22. Red Hat Customer Portal [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://access.redhat.com/security/cve/cve-2017-7494
23. CVE – CVE-2017-7494 [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7494
24. Samba CVE-2017-7494 Remote Code Execution Vulnerability [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://www.securityfocus.com/bid/98636/info
25. CTFd – CTFd 2.1.5 documentation [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://docs.ctfd.io/en/latest/
26. Deployment – CTFd 2.1.5 documentation [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://docs.ctfd.io/en/latest/deployment.html
27. Седлецький Д. В. Безпека контейнерів Docker. Кращі практики / Д. В. Седлецький // Актуальні проблеми кібербезпеки: Всеукраїнська наукова конференція: тези доп. – К.: ДУТ, 2019. – С. 11-13.

ДОДАТОК А. ВИХІДНИЙ КОД СТОРІНКИ INDEX.PHP ДОДАТКУ, ВРАЗЛИВОГО ДО SQL INJECTION

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta   name="viewport"   content="width=device-width,   initial-
scale=1">
  <title>Emailbook</title>
  <link rel="stylesheet" href="css/googlefonts.css">
  <link rel="stylesheet" href="css/font-awesome.min.css">
  <link rel="stylesheet" href="css/bootstrap.min.css">

  <link rel="stylesheet" href="css/demo.css" />
  <link rel="stylesheet" href="css/templatemo-style.css">

  <script                                    type="text/javascript"
src="js/modernizr.custom.86080.js"></script>
     </head>
     <body>
               <ul class="cb-slideshow">
                 <li></li>
                 <li></li>
                 <li></li>
                 <li></li>
                 <li></li>
                 <li></li>
             </ul>
               <div class="container-fluid">
                    <div class="row cb-slideshow-text-container ">
```

```
                                <div class= "tm-content col-xl-6 col-sm-8
col-xs-8 ml-auto section">
     <header class="mb-5"><h1>Emailbook</h1></header>
     <P class="mb-5">You can easily find email just by typing username
in this field:</P>
             <form action="#" method="post" class="subscribe-form">
                    <div class="row form-section">
                    <div class="col-md-7 col-sm-7 col-xs-7">
             <input    name="usr"    type="text"    class="form-control"
id="contact_email" placeholder="Enter username here..." required/>
     </div>
     <div class="col-md-5 col-sm-5 col-xs-5">
     <button type="submit" class="tm-btn-subscribe">Send</button>
     </div>
     </div>
     </form>
               <?php
               $servername = "db";
               $username = "ctf";
               $password = "73292d1fe6c43a39a59b4e9fb99163ca";
               $dbname = "emailbook";

               if         (((strpos($_POST["usr"],'UNION'))         ||
(strpos($_POST["usr"],'AND'))  ||  (strpos($_POST["usr"],'RLIKE'))  ||
(strpos($_POST["usr"],'OR')))  == false) {
                    $link                                          =
mysqli_connect($servername,$username,$password,$dbname);
                    if (!$link) {
          echo "Connection error: " . mysqli_connect_error() . PHP_EOL;
                         exit;
                    }
     $query  =  "SELECT  username,email  FROM  emailbook.core  WHERE
username='" . $_POST["usr"] . "' LIMIT 0,1;";
```

```
                    $result = $link->query($query);
                    if ($result->num_rows > 0) {
                         $row = $result->fetch_assoc();
     echo '<P class="mb-5">' . $row["username"] . " has email: " .
$row["email"] . "</P>";
                    } else {
                         echo '<P class="mb-5">' . "Lets find
something... </P>";
                    }
                    mysqli_close($link);
                    } else {
                         echo "Hacking detected!";
                    }
               ?>
               <div class="tm-social-icons-container text-xs-center">
                              <a href="#" class="tm-social-link"><i
class="fa fa-facebook"></i></a>
                              <a href="#" class="tm-social-link"><i
class="fa fa-google-plus"></i></a>
                              <a href="#" class="tm-social-link"><i
class="fa fa-twitter"></i></a>
                              <a href="#" class="tm-social-link"><i
class="fa fa-linkedin"></i></a>
                               </div>
                              </div>
                         </div>
                         <div class="footer-link">
                              <p>Copyright © 2019 HID</p>
                         </div>
                    </div>
          </body>
          <script type="text/javascript" src="js/app.js"></script>
     </html>
```

ДОДАТОК Б. ВМІСТ ФАЙЛУ DOCKER-COMPOSE.YML ДОДАТКУ CTFD

```
version: '2'
services:
  ctfd:
    build: .
    user: root
    restart: always
    ports:
      - "8000:8000"
    environment:
      - UPLOAD_FOLDER=/var/uploads
      - DATABASE_URL=mysql+pymysql://root:ctfd@db/ctfd
      - REDIS_URL=redis://cache:6379
      - WORKERS=4
      - LOG_FOLDER=/var/log/CTFd
      - ACCESS_LOG=-
      - ERROR_LOG=-
    volumes:
      - .data/CTFd/logs:/var/log/CTFd
      - .data/CTFd/uploads:/var/uploads
      - .:/opt/CTFd:ro
    depends_on:
      - db
    networks:
        default:
        internal:
  db:
    image: mariadb:10.4
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=ctfd
      - MYSQL_USER=ctfd
```

```
            - MYSQL_PASSWORD=ctfd
            - MYSQL_DATABASE=ctfd
          volumes:
            - .data/mysql:/var/lib/mysql
          networks:
              internal:
          # This command is required to set important mariadb defaults
          command: [mysqld, --character-set-server=utf8mb4, --collation-
server=utf8mb4_unicode_ci, --wait_timeout=28800, --log-warnings=0]

        cache:
          image: redis:4
          restart: always
          volumes:
          - .data/redis:/data
          networks:
              internal:
      networks:
          default:
          internal:
              internal: true
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)