КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

кібербезпеки та комп'ютерної інженерії

(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ
БАКАЛАВР

на тему:

«**Сценарії мережевого "Penetration Testing" у віртуалізованому середовищі кіберполігону**»

Заремба Богдан Вадимович

(прізвище, ім'я та по батькові здобувача повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Делембовський М.М.

„___" _____20___року

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

«Сценарії мережевого "Penetration Testing" у віртуалізованому середовищі кіберполігону»

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Заремба Богдан Вадимович

(прізвище, ім'я та по батькові повністю)

125 «Кібербезпека»

(спеціальність)

Безпека інформаційних та комунікаційних систем

(освітня програма)

Група БІКС-22

Керівник Делембовський М.М.

(прізвище та ініціали)

доцент кафедри кібербезпеки та комп'ютерної інженерії, кандидат технічних наук

(вчене звання, науковий ступінь)

Рецензент_____

(прізвище та ініціали)

Ідентичність підтверджую

Київ 2026р

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Випускова кафедра: кібербезпеки та комп'ютерної інженерії
Ступінь вищої освіти: бакалавр
Спеціальність: 125 «Кібербезпека»
Освітня програма: Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри

Делембовський М.М.
„___" _____20___року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Заремба Богдан Вадимович
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи «Сценарії мережевого "Penetration Testing" у віртуалізованому середовищі кіберполігону»

затверджена наказом ректора КНУБА № 576/52-15/13.2/26 від «14» травня 2026 року

2. Керівник роботи

доцент кафедри кібербезпеки та комп'ютерної інженерії, кандидат технічних наук Делембовський Максим Михайлович
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту _____

4. Зміст пояснювальної записки за розділами:

Р. 1. Аналіз предметної галузі та постановка задачі

Р. 2. Обґрунтування та розроблення рішення

Р. 3. Апробація та аналіз результатів тренажера

5. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	06.04.2026
Розділ 2	27.04.2026
Розділ 3	25.05.2026
Остаточне оформлення роботи	07.06.2026
Направлення роботи для перевірки на плагіат	10.06.2026
Попередній захист роботи	12.06.2026
Направлення роботи на рецензування	12.06.2026

6. Дата видачі завдання 10.02.2026

Керівник _____ _____

Здобувач _____ _____

РЕЗЮМЕ

до кваліфікаційної випускної роботи

здобувача: Заремба Богдан Вадимович

ЗВО	Київський національний університет будівництва і архітектури
Тема	Сценарії мережевого "Penetration Testing" у віртуалізованому середовищі кіберполігону
Освітній ступінь	Бакалавр
Факультет	Факультет автоматизації та інформаційних технологій
Випускова кафедра	Кафедра кібербезпеки та комп'ютерної інженерії
Спеціальність	125 «Кібербезпека»
Освітня програма	Безпека інформаційних і комунікаційних систем
Керівник	Делембовський М. М., к.т.н., доцент
Обсяг роботи	пояснювальна записка – 106 стор., 3 розділи, презентація – 8–10 слайдів
Зміст розділів роботи:	
Розділ 1.	Проаналізовано теоретичні засади penetration testing та методології PTES, NIST SP 800-115, OSSTMM, фреймворк MITRE ATT&CK і концепцію кіберполігонів. Виконано критичний огляд існуючих рішень (HackTheBox, TryHackMe, RangeForce, Metasploitable, DVWA) та обґрунтовано потребу у власному україномовному модульному тренажері.

Розділ 2.	Спроєктовано трирівневу архітектуру тренажера (Flask + SQLite + VirtualBox). Реалізовано модулі управління VM, сценаріїв з 5 типами валідаторів, генерації PDF-звітів. Розроблено модульну систему JSON-сценаріїв, CLI-інструменти та веб-редактор. Створено 5 навчальних сценаріїв (55 кроків) з покриттям PTES, ATT&CK, OWASP, NIST 800-115.
Розділ 3.	Проведено апробацію за 4 напрямами: інструментальне вимірювання (старт 43–87 с, RAM 1.2 ГБ, диск 30 ГБ); самотестування 5 сценаріїв (160 хв vs 290 планових, – 44,8%); тестування модульності (новий сценарій – 8 хв через веб); порівняння з аналогами. Підтверджено покриття 36 технік ATT&CK у 9 тактиках, 7/7 фаз PTES, 7/10 OWASP Top 10, 7/8 підфаз NIST 800-115.
Висновки по роботі:	Розроблено функціонально завершений програмний тренажер мережевого пентесту, готовий до інтеграції у навчальний процес кафедри. Тренажер має модульну архітектуру з розширенням без програмування, україномовний інтерфейс та повну автономність. Сформовано 8 напрямів подальшого розвитку.
Ключові слова:	*penetration testing, мережевий пентест, кіберполігон, віртуалізоване середовище, кібербезпека, навчальний тренажер, мережева безпека, MITRE ATT&CK, PTES, VirtualBox, модульна архітектура, JSON-сценарії, освітні технології.*

Здобувач: _____ / _____ /

Керівник: _____ / _____ /

«___» _____ 20__ р.

АНОТАЦІЯ

Кваліфікаційна робота бакалавра присвячена розробленню комплексу сценаріїв мережевого penetration testing у віртуалізованому середовищі кіберполігону та програмного тренажера для їх виконання, що призначений для підвищення рівня практичної підготовки здобувачів вищої освіти зі спеціальності 125 «Кібербезпека». Актуальність роботи обумовлена глобальним дефіцитом фахівців з кібербезпеки, критичним розривом у практичних навичках та відсутністю доступних навчальних кіберполігонів, адаптованих до освітніх програм українських ЗВО.

У роботі проаналізовано методології penetration testing (PTES, NIST SP 800-115, OSSTMM, MITRE ATT&CK, Cyber Kill Chain), досліджено сучасні платформи кіберполігонів та технології віртуалізації. Спроектовано та розроблено програмний тренажер з трирівневою архітектурою на базі Python і мікрофреймворку Flask та Oracle VirtualBox, що автоматизує розгортання сегментованого віртуального середовища (Kali Linux, pfSense, Metasploitable 2, DC-1). Реалізовано п’ять навчальних сценаріїв мережевого пентесту різного рівня складності (55 кроків) із покроковими інструкціями та маппінгом на техніки MITRE ATT&CK, а також CLI-інструменти й веб-редактор сценаріїв. Проведено апробацію тренажера та оцінено покриття стандартів PTES, MITRE ATT&CK, OWASP Top 10 і NIST SP 800-115.

Пояснювальна записка містить 3 розділи, 19 рисунків, 35 таблиць, список використаних джерел із 36 найменувань та 4 додатки.

Ключові слова: penetration testing, кіберполігон, віртуалізація, тренажер, мережева безпека, PTES, MITRE ATT&CK, Kali Linux, Metasploit, кібербезпека.

ABSTRACT

The bachelor's qualification work is devoted to the development of a set of network penetration testing scenarios in a virtualized cyber range environment and a software trainer for their execution, designed to improve the practical training of students in specialty 125 «Cybersecurity». The relevance of the work is determined by the global shortage of cybersecurity professionals, the critical practical skills gap, and the lack of accessible educational cyber ranges adapted to Ukrainian higher education programs.

The work analyzes penetration testing methodologies (PTES, NIST SP 800-115, OSSTMM, MITRE ATT&CK, Cyber Kill Chain) and examines modern cyber range platforms and virtualization technologies. A three-tier software trainer based on Python/Flask and Oracle VirtualBox is designed and implemented, automating the deployment of a segmented virtual environment (Kali Linux, pfSense, Metasploitable 2, DC-1). Five network penetration testing scenarios of varying difficulty (55 steps) with step-by-step instructions and MITRE ATT&CK mapping, together with CLI tools and a web-based scenario editor, are developed. The trainer is tested, and its coverage of the PTES, MITRE ATT&CK, OWASP Top 10 and NIST SP 800-115 standards is evaluated.

The explanatory note contains 3 chapters, 19 figures, 35 tables, a list of 36 references and 4 appendices.

Keywords: penetration testing, cyber range, virtualization, trainer, network security, PTES, MITRE ATT&CK, Kali Linux, Metasploit, cybersecurity.

ЗМІСТ

ВСТУП

Кібербезпека є однією з найважливіших галузей сучасних інформаційних технологій. За оцінками (ISC)², глобальний дефіцит фахівців з кібербезпеки у 2024 році перевищив 4 млн осіб, а потреба зростає швидше, ніж її здатні задовольнити ЗВО [19]. Значущість кіберпростору як складової національної безпеки України закріплена на державному рівні [26], а після початку повномасштабного вторгнення у 2022 році він став активним театром бойових дій, що формує високі вимоги до практичної підготовки фахівців спеціальності 125 «Кібербезпека» – здатних не лише описувати загрози, а й виявляти, експлуатувати та усувати реальні вразливості. Ключовим інструментом такої підготовки є мережевий penetration testing, опанування якого потребує спеціального ізольованого середовища – кіберполігону з навмисно вразливими цілями та методичним супроводом.

Наявні рішення не задовольняють потреб української вищої освіти: комерційні платформи (HackTheBox, TryHackMe, RangeForce) працюють онлайн з англомовним інтерфейсом, коштують 10–14 $/міс на студента та закриті для модифікації викладачем, а безкоштовні вразливі віртуальні машини (Metasploitable, DVWA, DC-1) не мають дидактичної надбудови – покрокових інструкцій, перевірки результатів і автоматичного оцінювання. Цей розрив доцільно закрити розробкою спеціалізованого програмного тренажера, що поєднує віртуалізований кіберполігон із дидактичним супроводом, україномовним інтерфейсом і безкоштовним розповсюдженням, чому й присвячено цю роботу.

Мета кваліфікаційної роботи. Розробити навчальні сценарії мережевого penetration testing та програмний тренажер для їх виконання у віртуалізованому середовищі кіберполігону, придатний для інтеграції у навчальний процес кафедри з можливістю самостійного розширення набору сценаріїв викладачем без потреби у програмуванні.

Завдання дослідження. Для досягнення зазначеної мети у кваліфікаційній роботі поставлено та послідовно вирішено такі завдання:

1) проаналізувати теоретичні засади penetration testing (методології PTES, NIST SP 800-115, фреймворк MITRE ATT&CK) та існуючі засоби навчання пентесту з оцінкою їх придатності для українських ЗВО;

2) обґрунтувати концепцію тренажера, обрати технологічний стек і спроєктувати трирівневу архітектуру програмного засобу;

3) розробити модуль управління віртуальними машинами через VirtualBox (керування життєвим циклом машин: запуск, зупинка, відновлення знімків стану (snapshots) для повторюваності експериментів);

4) розробити модуль сценаріїв із розширеним набором валідаторів відповідей, що забезпечують коректну перевірку результатів у середовищах зі змінними параметрами;

5) реалізувати модульну архітектуру (профілі віртуальних машин (ВМ), мережеві топології, сценарії), що дозволяє додавати контент без зміни коду;

6) розробити CLI-інструменти для роботи зі сценаріями: генератор шаблонів та валідатор JSON-файлів;

7) реалізувати веб-редактор сценаріїв для викладача без прямого редагування JSON;

8) розробити методику побудови навчальних сценаріїв і за нею реалізувати п’ять сценаріїв різного рівня складності (початкового, середнього та просунутого), що покривають фази PTES (Penetration Testing Execution Standard) та основні тактики MITRE ATT&CK;

9) провести апробацію розроблених сценаріїв та оцінити можливості тренажера за п’ятьма напрямами (вимірювання характеристик середовища, самотестування сценаріїв, перевірка можливості додавання нових сценаріїв, порівняльний аналіз з аналогами, верифікація покриття стандартів);

Об’єкт дослідження – процес практичної підготовки фахівців з інформаційної безпеки за компонентом мережевого penetration testing.

Предмет дослідження – сценарії та програмні засоби автоматизованого навчання мережевому penetration testing у віртуалізованому середовищі кіберполігону.

Застосовано теоретичні методи (аналіз науково-технічної літератури, порівняльний аналіз платформ навчання пентесту, системний аналіз стандартів PTES, NIST SP 800-115, MITRE ATT&CK, OWASP Top 10) та емпіричні (об'єктно-орієнтоване проєктування з архітектурними шаблонами, програмування на Python 3.10+/Flask/SQLite/JavaScript, інструментальне вимірювання продуктивності, ситуаційне моделювання створення сценарію трьома способами

Наукова новизна одержаних результатів. Запропоновано метод формування навчальних сценаріїв мережевого penetration testing, що поєднує покрокову дидактичну структуру завдань з автоматизованою перевіркою підказок на ненавмисне розкриття правильної відповіді; на відміну від наявних навчальних засобів, де коректність підказок перевіряється переважно вручну, це підвищує методичну якість сценаріїв. Удосконалено підхід до автоматизованої перевірки відповідей за рахунок використання розширеного набору валідаторів (точний збіг рядка, регулярний вираз, числовий діапазон, входження підрядка без урахування регістру, множинний вибір), що забезпечує коректну перевірку результатів у середовищах зі змінними параметрами та зменшує залежність оцінювання від конкретного стану цільової машини.

Практичне значення одержаних результатів: 1) розроблено україномовний тренажер мережевого penetration testing з модульною архітектурою орієнтований на локальне розгортання в навчальному процесі ЗВО без комерційних ліцензій та без залежності від доступу до Інтернету; 2) запропоновано алгоритм автоматичного виявлення семантичних витоків правильних відповідей у підказках сценаріїв; 3) реалізовано розширений набір валідаторів відповідей (text_flag, regex, numeric_range, case_insensitive_substring, multiple_choice); 4) розроблено дворівневу систему розширення (CLI-інструменти та веб-редактор з

довідником з автозаповненням ATT&CK), що суттєво знижує час створення сценарію.

Результатом роботи є функціонально завершений тренажер, готовий до інтеграції у навчальний процес кафедри (спеціальність 125): п’ять сценаріїв із 55 кроками (базовий і веб-пентест, багатоетапна атака з транзитом через компрометований хост, мережева розвідка, обхід базової детекції); модульне додавання сценаріїв через JSON або веб-редактор; CLI-валідатор сценаріїв; документація для викладачів; готовий комплекс ВМ (Kali Linux, Metasploitable 2, pfSense, DC-1) з налаштованою топологією і snapshots.

Тренажер пройшов п’ятивекторну апробацію: інструментальне вимірювання продуктивності, самотестування всіх п’яти сценаріїв з фіксацією часу, перевірку ефективності модульної архітектури та порівняльний аналіз з комерційними аналогами (HackTheBox, TryHackMe, RangeForce) і верифікацію покриття стандартів PTES, MITRE ATT&CK, OWASP Top 10 і NIST SP 800-115.

Структура та обсяг роботи. Робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел і додатків; обсяг основного тексту – 106 сторінок, 35 таблиць, 19 рисунків, 36 джерел. У першому розділі проаналізовано предметну галузь і поставлено задачу дослідження; у другому – обґрунтовано та розроблено тренажер (архітектура, віртуалізоване середовище, модулі, CLI та веб-редактор, розгортання стенду); у третьому – наведено результати апробації, порівняльний аналіз, покриття стандартів і напрями розвитку. Додатки містять лістинги коду, приклад JSON-сценарію та конфігураційні файли ВМ(віртуальних машин) і мережевої топології.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Penetration testing: поняття, цілі та класифікація

Забезпечення кібербезпеки інформаційних систем – ключове завдання сучасних організацій. Зі зростанням кіберзагроз зростає потреба у систематичній оцінці захищеності систем, мереж і додатків; одним із найефективніших методів є тестування на проникнення (penetration testing, пентест) – санкціонована імітація дій зловмисника для виявлення вразливостей до їх використання зловмисниками [1].

Згідно з NIST SP 800-115, пентест – метод оцінки безпеки, за якого оцінювачі в межах визначених обмежень намагаються обійти механізми захисту; цей підхід є складовою інструментальних методів перевірки, що застосовуються поряд із аналітичним дослідженням (інспектуванням) систем [1].

На відміну від автоматизованого сканування вразливостей (vulnerability assessment), яке лише ідентифікує потенційні слабкі місця, пентест передбачає їх реальну експлуатацію для оцінки фактичного впливу [1, 6]: сканування виявляє тисячі потенційних вразливостей, але саме пентест показує, які з них критичні для безпеки та дозволяють ескалювати атаку всередині системи.

Відмінним від пентесту є підхід red teaming – тривала (2–6 міс.) імітація дій реального противника з технічними, соціотехнічними та фізичними векторами для перевірки здатності організації виявляти й реагувати на загрози [4].

Bug Bounty – постійна програма залучення зовнішніх дослідників за винагороду (HackerOne, Bugcrowd), що, на відміну від пентесту, не має визначеного терміну й не гарантує систематичного покриття [6].

Порівняльну характеристику цих методів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика методів оцінки безпеки

Критерій	Penetration testing	Vulnerability assessment	Red teaming	Bug Bounty
Експлуатація вразливостей	Так	Ні	Так	На вибір замовника
Тривалість	1–4 тижні	Години–дні	Тижні–місяці	На вибір замовника
Межі тестування	Визначені	Визначені	Необмежені	Відкриті
Соціальна інженерія	Рідко	Ні	Так	Ні
Виконавець	Команда фахівців	Автомат. сканер	Команда фахівців	Зовнішні дослідники
Звітність	Детальний звіт	Список CVE	Детальний звіт	Окремі неструктуровані звіти
Вартість	Середня	Низька	Висока	На вибір замовника
Повторюваність	За запитом	Регулярна	За запитом	Постійна

Основні цілі пентесту [1, 2, 3]: виявлення вразливостей (помилки конфігурації, застаріле ПЗ з відомими CVE, слабкі паролі, архітектурні недоліки) до їх використання зловмисниками; оцінка реального ризику через демонстрацію наслідків експлуатації; перевірка ефективності засобів захисту (IDS/IPS, firewall, SIEM). Водночас пентест сприяє виконанню вимог стандартів (PCI DSS, ISO/IEC 27001 [33], Закону України «Про основні засади забезпечення кібербезпеки України» [5]) та підвищенню кваліфікації фахівців через набуття практичного досвіду, саме формування практичних навичок є ключовим орієнтиром цієї роботи.

Залежно від обсягу наданої інформації розрізняють три типи пентесту [1, 2]: black box (без даних про систему – максимально реалістично, але потребує найбільше часу), white box (повний доступ до документації, коду й конфігурацій – найглибший аналіз) і grey box (часткові знання – компроміс, найпоширеніший на практиці).

За об'єктами дослідження та векторами атак пентест поділяють на мережевий, соціотехнічний, фізичний, а також тестування безпеки веб-застосунків та бездротових мереж [1, 2, 6]. Мережевий пентест оцінює захищеність мережевої інфраструктури (маршрутизатори, сервери, сервіси SSH/FTP/SMB/HTTP/DNS) і охоплює сканування, експлуатацію мережевих протоколів, брутфорс, латеральний рух і підвищення привілеїв; він може бути зовнішнім або внутрішнім і є технічною основою навчальних сценаріїв, що розробляються в цій роботі [1].

Важливий аспект – правове регулювання: несанкціоноване тестування кваліфікується як кіберзлочин (ст. 361–363 Кримінального кодексу України; Будапештська конвенція) [5], тому всі дії мають бути санкціоновані письмовою угодою про межі тестування (дозволені дії, часові рамки, IP-адреси цілей) [2].

Ця вимога додатково обґрунтовує потребу в ізольованих віртуалізованих середовищах – кіберполігонах – де студенти можуть вільно застосовувати техніки атаки проти спеціально створених цілей без правових наслідків.

Отже, пентест – комплексний метод оцінки безпеки з активною експлуатацією вразливостей у контрольованих умовах; класифікація за рівнем знань і вектором атаки дозволяє обирати оптимальний підхід. Мережевий пентест потребує широкого інструментарію й структурованих методологій, аналіз яких – у наступному підрозділі.

1.2 Методології та фреймворки penetration testing

Ефективний пентест потребує структурованого підходу через методології та фреймворки, що визначають послідовність етапів, набір технік, формати звітності й критерії оцінки, забезпечуючи повторюваність і формалізацію навчання. Нижче проаналізовано основні методології для обґрунтування вибору підходу.

Penetration Testing Execution Standard (PTES) – одна з найбільш структурованих методологій, розроблена спільнотою фахівців з інформаційної безпеки (ІБ) [2]; вона визначає сім послідовних фаз:

1) Pre-engagement – узгодження обсягу й меж тестування, правил взаємодії, NDA, термінів і формату звітності [2];

2) Intelligence Gathering – збір інформації пасивними (OSINT) та активними (сканування, визначення ОС і версій) методами [2];

3) Threat Modeling – аналіз зібраного для визначення найвірогідніших векторів атаки та цінних активів [2];

4) Vulnerability Analysis – пошук вразливостей сканерами (OpenVAS, Nessus) і вручну, класифікація за Common Vulnerability Scoring System (CVSS), виключення хибних спрацювань [2];

5) Exploitation – використання підтверджених вразливостей для отримання доступу й обходу захисту [2, 3];

6) Post-Exploitation – підвищення привілеїв, латеральний рух, закріплення, збір даних, оцінка впливу компрометації [2];

7) Reporting – короткий звіт для керівництва й детальний технічний звіт з практичною демонстрацією вразливостей та оцінкою за CVSS [2].

Послідовність фаз PTES та їх взаємозв'язок представлено на рисунку 1.1.

Рисунок 1.1 – Послідовність фаз методології PTES

Перевага PTES – детальна регламентація фаз і конкретні технічні рекомендації, що робить її придатною для навчання: студенти отримують чіткий алгоритм дій, що знижує поріг входження [2].

Open-Source Security Testing Methodology Manual (OSSTMM) v3 фокусується на кількісному вимірюванні безпеки через метрику Risk Assessment Value (RAV) і визначає шість видів тестування людський фактор, фізична

безпека, бездротові технології, телекомунікації, мережі даних та відповідність стандартам)[7]; його методологічний апарат об'єктивний, але надто складний і абстрактний для засвоєння здобувачами бакалаврського рівня.

Cyber Kill Chain (Lockheed Martin, 2011) описує сім етапів цілеспрямованої атаки: розвідка, озброєння(створення шкідливого коду чи експлойту під конкретну вразливість), доставка(передача шкідливого контенту жертві через фішинг, вебресурси тощо), експлуатація(запуск шкідливого коду після спрацювання вразливості), встановлення(закріплення зловмисника у системі, створення бекдорів), управління(побудова каналу зв'язку з сервером зловмисника),дії на цільових об'єктах(безпосереднє виконання мети атаки: крадіжка даних, шифрування тощо) [8]. Модель представлено на рисунку 1.2.

Рисунок 1.2 – Етапи моделі Cyber Kill Chain (Lockheed Martin, 2011)

Значення моделі полягає в описі атаки як цілісного процесу, де кожен етап залежить від попереднього, що дозволяє структурувати сценарії як логічно пов'язані кроки. Обмеження: вона орієнтована переважно на застосування шкідливого програмного забезпечення і не охоплює внутрішніх загроз, використання легітимних облікових даних, атак на ланцюги постачання та компрометації хмарних сервісів [4, 8].

MITRE ATT&CK – глобальна база знань про тактики й техніки кібератак на основі реальних інцидентів та APT-угруповань [9];корпоративна матриця містить 14 тактик і понад 200 технік та 400 підтехнік [10].

Релевантні для мережевого пентесту тактики охоплюють: збір інформації (Reconnaissance, TA0043), первинний доступ (Initial Access, TA0001), виконання (Execution, TA0002), закріплення в системі (Persistence, TA0003), підвищення привілеїв (Privilege Escalation, TA0004), обхід засобів захисту (Defense Evasion, TA0005), доступ до облікових даних (Credential Access, TA0006), розвідку всередині мережі (Discovery, TA0007) та горизонтальне переміщення (Lateral Movement, TA0008) [9, 10]. Кожна техніка має унікальний ідентифікатор (наприклад, T1190, T1078), технічний опис, приклади реальних атак APT-угруповань і рекомендації щодо їх виявлення та протидії [10]. На відміну від моделі Cyber Kill Chain, база знань ATT&CK є значно деталізованішою та постійно оновлюється [9].

У роботі MITRE ATT&CK використовується для співвіднесення елементів сценаріїв: кожна дія сценарію прив'язується до конкретної техніки, що формує у студентів зв'язок навчальних вправ із реальними атаками [9, 10].

OWASP Testing Guide v4.2 – методологія тестування безпеки веб-застосунків; окремі її розділи (тестування автентифікації, авторизації, управління сесіями, криптографічного захисту) є застосовними й на мережевому рівні. Методологія визначає такі базові фази, як пасивний збір інформації (passive information gathering), активне тестування (active testing), аналіз (analysis) та звітування (reporting).

Порівняльну характеристику розглянутих методологій та фреймворків наведено у таблиці 1.2

Таблиця 1.2 – Порівняльна характеристика методологій та фреймворків пентесту

Критерій порівняння	**PTES**	**OSSTMM v3**	**Cyber Kill Chain**	**MITRE ATT&CK**	**OWASP TG**
Фокус	Процес пентесту	Метрики безпеки	Етапи атаки	Тактики і техніки	Веб-застосунки
Деталізація	Висока	Середня	Низька	Дуже висока	Висока
Кількісні метрики	ні	так	ні	ні	ні
Прив'язка до реальних атак	Часткова	Ні	Так	Так (CVE, APT)	Часткова
Динаміка оновлення	Рідко	Рідко	Ніколи	Постійна	Постійна
Придатність для навчання	так	ні	так	так	так

Результати аналізу методологій підтверджують доцільність застосування комбінованого підходу: фази PTES визначено як базу для організації процесу; техніки MITRE ATT&CK використано для мапування дій у сценаріях відповідно до реальних загроз; модель Cyber Kill Chain обрано як теоретичну рамку для моделювання логіки зловмисника, а специфічні елементи OWASP Testing Guide — для тестування веб-залежних модулів [2, 6, 8, 9].

1.3 Кіберполігони: концепція, архітектура, існуючі рішення

Кіберполігон (cyber range) – спеціалізоване середовище для проведення практичних занять, тренувань та досліджень у сфері кібербезпеки [36], що моделює реалістичну мережеву інфраструктуру із взаємопов'язаних систем, сервісів і вразливостей [11]. Термін запозичено з військової термінології (навчальний полігон) [13].

На відміну від CTF-платформ і одиночних вразливих VM, кіберполігон відтворює реальну корпоративну мережу з серверами, робочими станціями, мережевим обладнанням і засобами захисту [11, 12], що дозволяє моделювати складні багатоетапні атаки.

Згідно з аналітичним оглядом М. М. Яміна (M. M. Yamin) та колег [11], до основних завдань кіберполігонів належать: формування навичок оцінювання захищеності (offensive) та організації оборони (defensive); функціональне тестування засобів захисту інформації (IDS/IPS, SIEM, міжмережевих екранів); проведення командних кібертренувань (для команд атаки — Red Team та захисту — Blue Team); дослідження актуальних кіберзагроз, а також комплексне оцінювання компетентності здобувачів через виконання практичних завдань.

За способом розгортання кіберполігони поділяють на локальні, хмарні та гібридні [11, 12]. Локальні рішення (on-premises) забезпечують повний контроль над даними, незалежність від доступу до мережі інтернет та максимальну ізоляцію середовища, проте потребують значних фінансових інвестицій в апаратне забезпечення та його подальше технічне адміністрування [12]. Хмарні платформи (на базі AWS, Azure, GCP) є високомасштабованими й не потребують власних фізичних серверів, але мають пряму залежність від інтернет-провайдера, містять юридичні обмеження на типи дозволених кібератак і зумовлюють високі операційні витрати [11, 12]. Гібридні рішення поєднують переваги та архітектурні особливості обох підходів [11, 12].

За цільовим призначенням кіберполігони класифікують на навчальні, дослідницькі, військові та комерційні [12, 13]. До військових рішень належать

масштабні платформи для міжнародних навчань (як-от Locked Shields під егідою NATO CCDCOE або Michigan Cyber Range). Прикладами комерційних платформ є Immersive Labs, RangeForce та Cyberbit; проте вартість їхнього щорічного ліцензування є критично високою для більшості вітчизняних закладів вищої освіти й становить від 10 000 до 500 000 доларів США [12].

Архітектура типового кіберполігону складається з шести основних компонентів, що забезпечують його функціонування [11, 14]:

1. Платформа віртуалізації — базовий гіпервізор із підтримкою ізоляції, механізмів клонування та створення знімків стану (snapshots) для швидкого відновлення середовища (наприклад, VMware ESXi, Proxmox VE, OpenStack, VirtualBox) [11, 17].
2. Мережева інфраструктура — віртуальні мережі із забезпеченням суворої ізоляції та сегментації (за допомогою технологій VLAN, Internal Network, SDN), а також віртуальні маршрутизатори, комутатори, сервери DNS і DHCP для реалістичного моделювання корпоративних топологій [14].
3. Вразливі цілі — віртуальні машини з навмисно інтегрованими дефектами безпеки: застарілими операційними системами, слабкими автентифікаційними даними, помилками конфігурування та спеціалізованими вразливими вебзастосунками (як-от DVWA або WebGoat) [11, 15].
4. Інструментальні станції тестування (атакувальні машини) — робочі місця користувачів, оснащені спеціалізованими операційними системами та утилітами для проведення пентесту (наприклад, Kali Linux, Parrot OS) [21].
5. Система моніторингу та логування — комплекс засобів для централізованого збору й аналізу системних логів (платформи ELK, Splunk, Graylog), виявлення вторгнень (IDS/IPS Snort чи Suricata) та збереження дампів мережевого трафіку у форматі pcap [11].
6. Система управління та оцінювання — підсистема автоматизованої перевірки виконання завдань (зокрема за концепцією CTF через пошук файлів

зафіксованих прапорів — *flag.txt*), відстеження загального прогресу користувачів, генерації звітів та надання зворотного зв'язку [11, 14].

Архітектуру кіберполігону та взаємозв'язки між його компонентами представлено на рисунку 1.3

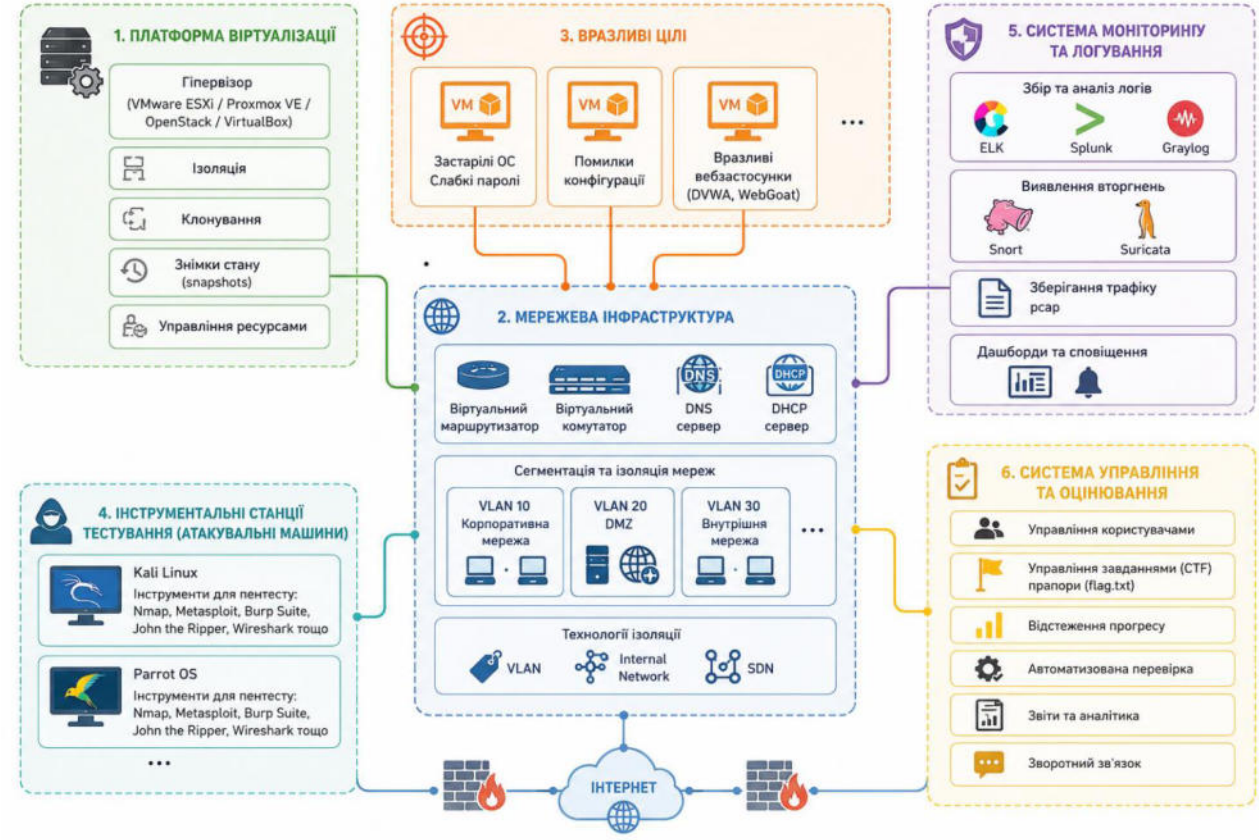


Рисунок 1.3 – Архітектура кіберполігону

Для обґрунтування доцільності розробки власного рішення проведено детальний аналіз наявних платформ, доступних для навчальних цілей.

Metasploitable 2 (Rapid7, на базі ОС Ubuntu 8.04) – навмисно вразлива віртуальна машина із заздалегідь інтегрованими сервісами vsftpd 2.3.4 (із наявним бекдором), Samba 3.x (уразливість CVE-2007-2447), Apache Tomcat, СКБД MySQL/PostgreSQL зі слабкими автентифікаційними даними, UnrealIRCd та платформою DVWA [15]. Переваги: безкоштовність, простота використання, високий рівень документованості. Обмеження: являє собою автономну машину

без комплексної мережевої інфраструктури, містить застарілі уразливості, не має чітких навчальних сценаріїв та автоматизованого оцінювання [15].

Metasploitable 3 – комплекс із двох віртуальних машин (Windows Server 2008 R2 та Ubuntu 14.04) із розширеним набором дефектів безпеки (зокрема EternalBlue, MS08-067, Active Directory, Jenkins, Apache Struts) [15]. Проте платформа є складнішою у розгортанні (потребує інструментів Vagrant/Packer), вибагливою до апаратних ресурсів (від 8 ГБ оперативної пам'яті) і не містить єдиних наскрізних сценаріїв [15].

DVWA – навмисно вразливий вебдодаток (PHP/MySQL) з десятьма типами веб-вразливостей (SQL Injection, XSS, Command Injection, File Upload, CSRF, File Inclusion тощо) і чотирма рівнями складності [30]. Відмінний для вивчення веб-вразливостей, але не охоплює мережевий рівень пентесту [6, 11].

VulnHub – відкрита колекція, що налічує понад 500 навмисно вразливих образів віртуальних машин для самостійного навчання у форматі здобуття привілеїв суперкористувача (root) [11]. Переваги: архітектурна різноманітність, безкоштовність, активна підтримка спільноти. Обмеження: машини є повністю ізольованими й не об'єднані в єдину топологію (що унеможливлює відпрацювання латерального руху), відсутня загальна навчальна структура та методичний супровід, а якість образів є неоднорідною [11].

Hack The Box (HTB, 2017) – комерційна хмарна платформа з динамічно оновлюваними образами хостів, сегментованими лабораторними мережами Pro Labs (для вивчення Active Directory, DMZ) та власною освітньою екосистемою HTB Academy [12]. Переваги: висока реалістичність (використання актуальних уразливостей CVE), масштабна спільнота (понад 1,5 млн користувачів). Обмеження: потребує постійного доступу до інтернету/VPN, є комерційною (вартість підписки на HTB Academy становить від 18 доларів США на місяць), не передбачає інструментів прямого контролю з боку викладача, є повністю англомовною, а її контент не адаптований до вітчизняних освітньо-професійних програм (ОПП) та не структурований за фазами PTES [12].

TryHackMe (THM, 2018) – платформа керованого навчання, побудована на концепції навчальних «кімнат» (поєднання теоретичного матеріалу та покрокових завдань з автоперевіркою за прапорами) і використання інтегрованого браузерного середовища AttackBox [12]. Переваги: чітка покроковість, баланс між теорією і практикою, градація складності. Обмеження: комерційний характер платформи (індивідуальна підписка — 14 доларів США на місяць, академічна ліцензія — від 180 доларів США на рік), відсутність адаптації під українські стандарти, англомовний інтерфейс, відсутність жорсткої прив'язки до моделей PTES/ATT&CK та відсутність панелі керування для викладача ЗВО [12].

KYPO Cyber Range (Університет Масарика, на базі OpenStack/KVM) – академічний кіберполігон, що підтримує створення складних багаторівневих сценаріїв, реалізує концепцію «інфраструктура як код» (IaC через Terraform, Ansible) та забезпечує детальне оцінювання прогресу [14]. Переваги: найбільш комплексна та відкрита архітектура серед університетських рішень. Обмеження: критично складне розгортання (потребує глибокого досвіду роботи з хмарною платформою OpenStack), надзвичайно високі вимоги до обладнання (від 64 ГБ оперативної пам'яті та 16 ядер процесора), обов'язкова потреба в окремому системному адміністраторі для супроводження системи та повна неадаптованість до стандартів вищої освіти України [14, 25].

Порівняльну характеристику усіх розглянутих платформ наведено у таблиці 1.3

Таблиця 1.3 – Порівняльний аналіз існуючих платформ кіберполігонів

Платформа	**Тип розгортання**	**Ліцензія**	**Навчальні сценарії**	**Методичний супровід**	**Панель контролю викладача**
Metasploitable 2/3	Локальна	Безкоштовна	ні	ні	ні
DVWA	Локальна	Безкоштовна	ні	ні	ні
VulnHub	Локальна	Безкоштовна	ні	ні	ніі
Hack The Box	Хмарна	Комерційна	так	так	ні
TryHackMe	Хмарна	Комерційна	так	так	ні
KYPO	Локальна	Відкрита	так	так	так

Результати порівняльного аналізу свідчать про відсутність рішень, що поєднують: безкоштовність та запуск на локальних комп'ютерах студентів; структурування сценаріїв за моделями PTES та MITRE ATT&CK; україномовне методичне забезпечення; інструменти викладацького контролю та градацію складності. Це повністю обґрунтовує необхідність проєктування та програмної реалізації авторських сценаріїв пентесту в ізольованому віртуальному середовищі (матеріали викладено в розділах 2 та 3).1.4 Технології віртуалізації для побудови кіберполігону

1.4 Технології віртуалізації для побудови кіберполігону

Віртуалізація є фундаментальною технологією побудови кіберполігонів, оскільки вона забезпечує ізоляцію процесів навчання від промислових (штатних) систем, можливість швидкого відновлення вихідного стану через механізм знімків стану (snapshots), а також високу масштабованість і ефективне використання обчислювальних ресурсів [17].

За типом архітектури гіпервізора розрізняють два базові підходи [17]. Гіпервізори 1-го типу (bare-metal) встановлюються безпосередньо на апаратне

забезпечення (наприклад, VMware ESXi, Proxmox VE, Hyper-V); вони характеризуються найвищою продуктивністю та рівнем ізоляції[17].

Гіпервізори 2-го типу (hosted) функціонують як спеціалізовані застосунки у складі хостової операційної системи (Oracle VirtualBox, VMware Workstation); вони є значно простішими в налаштуванні, не потребують окремого серверного обладнання й підходять для запуску на персональних комп'ютерах (ноутбуках) здобувачів вищої освіти, що є критично важливим чинником для забезпечення доступності навчання [17].

Порівняння архітектур гіпервізорів Type 1 та Type 2 представлено на рисунку 1.4

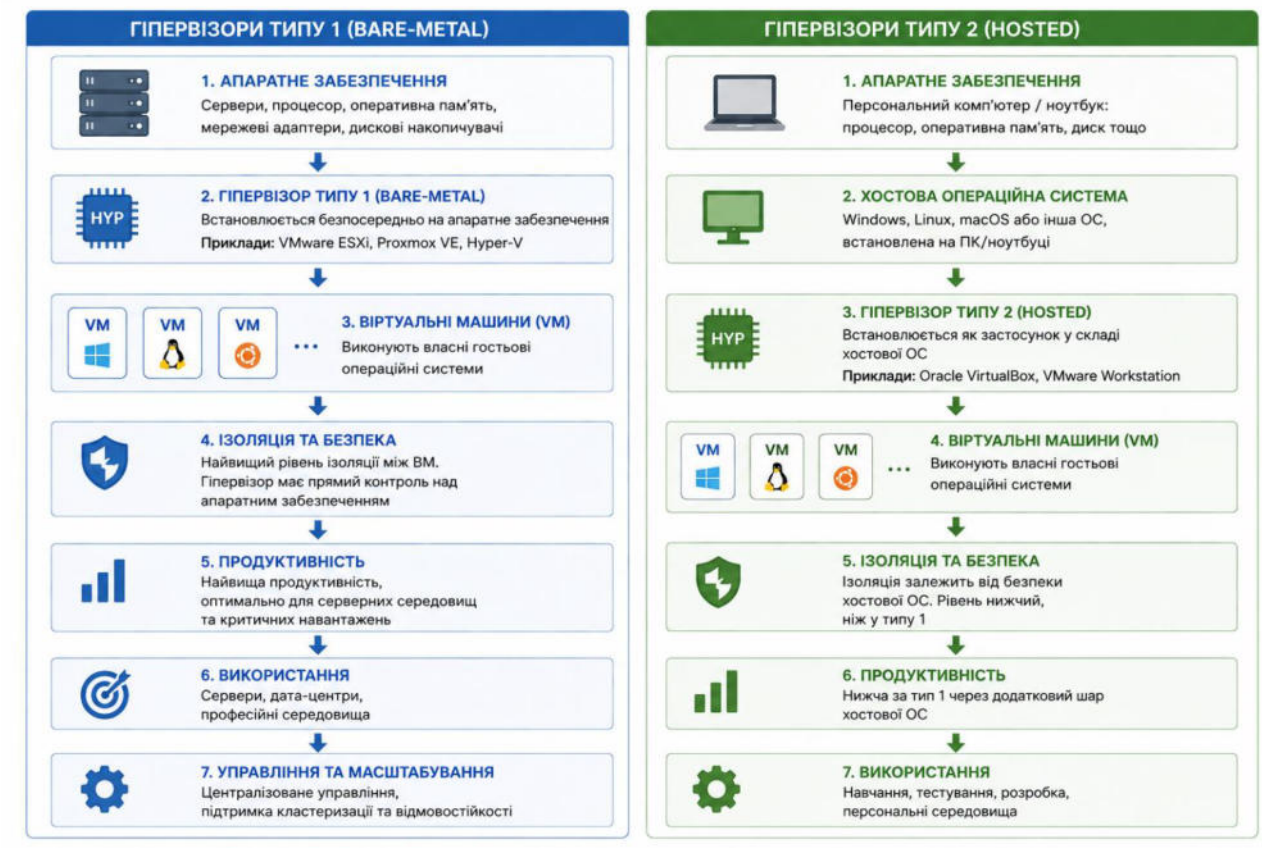


Рисунок 1.4 – Порівняння архітектур гіпервізорів Type 1 та Type 2

Контейнерна віртуалізація (Docker, LXC) забезпечує високу швидкість розгортання та мінімальні накладні витрати обчислювальних ресурсів завдяки використанню спільного ядра операційної системи [12, 17]. Проте така

архітектура обмежує можливості повноцінної емуляції повнофункціональних мережевих пристроїв (як-от маршрутизатор pfSense) та багатокомпонентних інфраструктур (як-от інфраструктура на базі хоста DC-1), що є критичним для відпрацювання багатоетапних атак; саме тому для завдань мережевого тестування на проникнення рекомендовано застосовувати повну апаратну віртуалізацію (віртуальні машини), тоді як контейнеризацію доцільно використовувати лише для розгортання допоміжних компонентів [17].

Для віртуалізації мережі Oracle VirtualBox надає чотири типи адаптерів [16]: NAT (доступ до інтернету, ізоляція від інших VM), Bridged (підключення до фізичної мережі), Internal Network (ізольована мережа між VM – ідеально для кіберполігону) і Host-Only (мережа між VM і хостом без зовнішнього доступу) [16].

Для складніших топологій із використання маршрутизаторів й комутаторів використовують емулятори мережевого обладнання GNS3 та EVE-NG (Cisco, Juniper, MikroTik), які дозволяють повноцінно відтворювати VLAN, маршрутизацію та списки контролю доступу (ACL) [14].

Порівняння платформ віртуалізації для кіберполігону наведено у таблиці Г.1 (дод. Г).

Для практичної реалізації тренажера було обрано платформу Oracle VirtualBox, що обумовлено такими її характеристиками: приналежність до гіпервізорів 2-го типу, що дозволяє запускати систему безпосередньо на персональних комп'ютерах здобувачів вищої освіти; безкоштовність та відкритість ліцензії (GPLv2); кросплатформність (підтримка операційних систем Windows, Linux та macOS); наявність механізму знімків стану (snapshots); підтримка режиму внутрішньої мережі (Internal Network) для забезпечення повної ізоляції навчального середовища; підтримка вкладеної віртуалізації (nested virtualization), а також наявність активної світової спільноти[16]. Водночас для серверного варіанту розгортання на базі кафедри передбачено можливість подальшого масштабування інфраструктури до платформи Proxmox VE з централізованим управлінням обчислювальними ресурсами.

1.5 Інструменти мережевого penetration testing

Для реалізації навчальних сценаріїв мережевого тестування на проникнення необхідний комплекс інструментів, структурований за фазами стандарту PTES [2]. Принциповою вимогою при цьому є використання виключно рішень із відкритим вихідним кодом (open-source) або безкоштовних продуктів зі складу дистрибутива Kali Linux [21].

Фаза збору інформації та розвідки (Intelligence Gathering) передбачає накопичення максимального обсягу даних про цільову інфраструктуру [2, 18]. Універсальним інструментом для виконання цих завдань є Nmap (Network Mapper), який забезпечує виявлення активних хостів (host discovery), сканування відкритих портів (port scanning), визначення версій запущених сервісів (прапор -sV), ідентифікацію операційних систем (прапор -O) та використання вбудованого рушія скриптів NSE для автоматизації розвідки й виявлення вразливостей [18].

Сканер Nmap підтримує режими SYN-сканування (прапор -sS, швидкий напіввідкритий метод, що потребує привілеїв суперкористувача), TCP-підключення (-sT), UDP-сканування (-sU) та агресивний режим (-A) [18]. У контексті моделювання загроз ці дії реалізують техніки T1046 (Network Service Discovery) та T1018 (Remote System Discovery) матриці MITRE ATT&CK [10].

Крім того, спеціалізовані утиліти Netdiscover та ARP-scan дозволяють оперативно виявляти хости в локальній мережі на канальному рівні за допомогою протоколу ARP, що є високоефективним у випадках, коли ICMP-трафік блокується міжмережевим екраном, — особливо під час проведення внутрішнього тестування на проникнення [3].

Фаза аналізу вразливостей (Vulnerability Analysis) передбачає пошук та ідентифікацію дефектів безпеки у виявлених мережевих службах.[1, 2]

Спеціалізований інструмент OpenVAS (Greenbone) виступає безкоштовним сканером уразливостей із базою NVT (Network Vulnerability Tests), яка налічує понад 100 000 перевірок. Платформа дозволяє виявляти відомі уразливості з реєстру CVE, класифікувати їх за шкалою CVSS та генерувати

детальні технічні звіти, що в контексті моделювання загроз реалізує техніку T1595 (Active Scanning) матриці MITRE ATT&CK[1, 10].

Для аудиту вебкомпонентів використовується Nikto — спеціалізований сканер вебсерверів, який здійснює перевірку на наявність понад 6700 небезпечних або потенційно шкідливих файлів, аналізує застарілі версії програмного забезпечення та виявляє типові помилки конфігурування, такі як можливість виведення вмісту каталогів (directory listing) або використання стандартних автентифікаційних даних за замовчуванням (default credentials).[6]

Для дослідження специфічних мережевих служб застосовується утиліта Enum4linux, яка забезпечує збір та перелічення інформації через протокол SMB, дозволяючи отримати дані про користувачів, групи, спільні мережеві ресурси (shares), чинні політики паролів та версію операційної системи. У матриці MITRE ATT&CK цей процес відповідає технікам T1087 (Account Discovery) та T1135 (Network Share Discovery)[10].

Фаза експлуатації (Exploitation) передбачає практичне використання підтверджених уразливостей для здобуття несанкціонованого доступу до цільових систем [2, 3].

Ключовим інструментом на цьому етапі є Metasploit Framework — спеціалізоване модульне середовище для проведення тестування на проникнення, яке містить понад 2000 модулів експлуатації [20]. У межах побудови сценаріїв тренажера використовуються такі типи модулів цієї платформи [3, 20]:

exploits — програмний код для експлуатації конкретних дефектів безпеки (наприклад, *exploit/unix/ftp/vsftpd_234_backdoor* [34] чи *ms17_010_eternalblue*);

payloads — корисне навантаження, яке виконується на цільовій машині після успішної атаки для створення зворотного (*reverse_tcp*) чи прямого (*bind_tcp*) каналу зв'язку, зокрема інтерактивна оболонка Meterpreter [20];

auxiliary — допоміжні модулі для сканування, перевірки автентифікації та розвідки [20];

post — модулі для виконання дій після компрометації (збір інформації, закріплення в системі) [20];

encoders — засоби кодування для обходу базових систем детекції та антивірусного захисту [20].

Для моделювання атак на автентифікаційні дані застосовується утиліта Hydra — інструмент для швидкого автоматизованого перебору паролів (brute-force) із підтримкою понад 50 мережевих протоколів (як-от SSH, FTP, HTTP, SMB, RDP, MySQL тощо). Програма реалізує словникові атаки із використанням класичної бази паролів *rockyou.txt* (понад 14 млн записів), що у матриці MITRE ATT&CK безпосередньо відображає техніку T1110 (Brute Force) [3, 10].

Для автоматизації аналізу вебскладової в сценаріях тренажера застосовується SQLmap — спеціалізований інструмент для автоматизованого виявлення та експлуатації SQL-ін'єкцій. Програма підтримує різні методи проведення атак, серед яких: логічні сліпі (boolean-based blind), сліпі на основі часових затримок (time-based blind), на основі виведення помилок (error-based), ін'єкції з використанням оператора UNION (UNION query-based), а також пакетовані запити (stacked queries).

Утиліта є сумісною з більшістю популярних систем керування базами даних (зокрема MySQL, PostgreSQL, MSSQL, Oracle) та має розширений функціонал, що дозволяє виконувати ексфільтрацію (вивантаження) вмісту баз даних, здійснювати читання/запис файлів у системі й отримувати інтерактивну командну оболонку (shell) на цільовому хості [6].

Фаза постексплуатації (Post-Exploitation) передбачає оцінювання повного обсягу компрометації цільової інфраструктури та аналіз потенційних деструктивних наслідків для функціонування організації [2].

Для забезпечення інтерактивного керування скомпрометованим хостом використовується розширене корисне навантаження Meterpreter зі складу платформи Metasploit [20]. Воно надає широкий спектр інструментів автоматизації дій зловмисника, серед яких: збір інформації про систему (sysinfo), вивантаження хешів паролів із пам'яті (hashdump), перехоплення введення з

клавіатури (keylogging), створення знімків екрана (screenshot) та файловий обмін. Крім того, модуль підтримує функції перенаправлення портів і динамічного тунелювання трафіку (portfwd/autoroute) для реалізації латерального руху (pivoting), а також утиліти автоматичного підвищення привілеїв, як-от getsystem [20].

Для автоматизованого пошуку векторів локального підвищення привілеїв у сценаріях тренажера застосовуються скрипти LinPEAS (для систем на базі Linux) та WinPEAS (для ОС Windows) [10]. Вони здійснюють комплексний аудит конфігурацій, виявляючи файли із встановленими прапорами SUID/SGID, незахищені завдання планувальника cron, застарілі версії ядер для експлуатації відомих уразливостей, помилки налаштування прав доступу команди sudo (misconfigurations) у середовищі Linux, а також незакавичковані шляхи до служб (unquoted service paths) і збережені автентифікаційні дані у середовищі Windows. У матриці MITRE ATT&CK ці дії безпосередньо реалізують техніки тактики підвищення привілеїв (Privilege Escalation, TA0004) [10].

Для моделювання складних атак на доменні інфраструктури використовується інструмент Mimikatz, призначений для зчитування автентифікаційних даних із пам'яті операційних систем Windows (зокрема паролів у відкритому вигляді, NTLM-хешів та Kerberos-квитків) [10]. Програма реалізує техніку T1003 (OS Credential Dumping) та її підтехніки, орієнтовані на компрометацію процесу Local Security Authority Subsystem Service (LSASS), бази даних Менеджер безпеки облікових записів(SAM) та проведення атак типу DCSync. Застосування цього інструмента є критично важливим для демонстрації вразливостей архітектури Active Directory та відпрацювання атак класу Pass-the-Hash [10].

У межах завдань з аналізу та перехоплення мережевого трафіку застосовуються такі інструменти: Wireshark — глибокий аналізатор мережевих протоколів (із підтримкою понад 2000 протоколів), що забезпечує гнучку фільтрацію й візуалізацію пакетів [1]; Tcpdump — консольний інструмент фіксації трафіку; а також утиліта Responder. Остання призначена для імітації

відповідей на запити протоколів LLMNR, NBT-NS та mDNS з метою перехоплення NTLMv2-хешів користувачів, що в класифікації MITRE ATT&CK відповідає техніці T1557 (Adversary-in-the-Middle) [10].

Систематизацію інструментів за фазами PTES та їх відповідність технікам MITRE ATT&CK наведено у таблиці Г.2 (додаток Г).

Усі зазначені інструментальні засоби інтегровані до складу операційної системи Kali Linux — спеціалізованого дистрибутива на базі Debian, розробленого компанією Offensive Security, який містить понад 600 утиліт безпеки, структурованих за категоріями (зокрема Information Gathering, Exploitation Tools, Post Exploitation тощо) [21]. У межах розробки тренажера платформу Kali Linux використано як головну станцію атакувальника, а взаємне зіставлення її утиліт із техніками бази знань MITRE ATT&CK забезпечує високу практичну релевантність навчальних сценаріїв.

1.6 Проблеми практичної підготовки фахівців з кібербезпеки

Глобальний дефіцит кваліфікованих кадрів у сфері кібербезпеки наразі є однією з найгостріших проблем галузі. За даними дослідження ISC2 Cybersecurity Workforce Study 2024, світова нестача фахівців становить 4,8 млн осіб, при цьому темпи зростання робочої сили майже зупинилися (0,1% порівняно з 8,7% у попередньому періоді), а основним чинником такої тенденції визначено бюджетні обмеження всередині організацій (39%) [19]. Безпосередньо в європейському регіоні дефіцит фахівців оцінюється у понад 1,3 млн осіб [19].

Особливо критичним є розрив у практичних навичках персоналу: 90% організацій засвідчують його наявність, 58% респондентів вважають це суттєвим ризиком для операційної діяльності, а 64% — оцінюють цей фактор як більшу загрозу, ніж безпосередній брак фізичної кількості працівників [19]. Серед найбільш дефіцитних компетенцій виділяють: тестування на проникнення та оцінювання вразливостей, аналіз кіберзагроз, реагування на інциденти комп'ютерної безпеки, а також забезпечення безпеки хмарних технологій [19].

На міжнародному рівні зазначені вимоги формалізовано у структурі Національної ініціативи з кібербезпечної освіти — NICE Cybersecurity Workforce Framework (NIST SP 800-181 Rev. 1) [22]. Згідно з цією методологією, для напряму тестування на проникнення найбільш релевантними є ролі аналітика з експлуатації вразливостей (Exploitation Analyst) та аналітика з оцінювання вразливостей (Vulnerability Assessment Analyst), виконання обов'язків на яких вимагає наявності значного практичного досвіду та прикладних навичок.

Програма професійної сертифікації CompTIA PenTest+ (PT0-002) структурована за п'ятьма основними доменами знань, серед яких найбільшу питому вагу має розділ «Атаки та експлуатація вразливостей» (Attacks and Exploits), на який припадає 30% від загального обсягу матеріалу [23]. Успішне засвоєння цього домену передбачає обов'язкове виконання практичних завдань у спеціалізованому віртуальному середовищі, оскільки виключно теоретичної підготовки для складання кваліфікаційного іспиту недостатньо [23].

Міжнародний стандарт CyBOK (Cyber Security Body of Knowledge), створений науковцями 16 університетів Великої Британії, виділяє 21 область знань [24]. Серед них такі напрями, як «Поведінка зловмисників» (Adversarial Behaviours), «Операційна безпека» (Security Operations) та «Мережева безпека» (Network Security), безпосередньо пов'язані з навичками тестування на проникнення. Оскільки ці напрями потребують обов'язкового практичного відпрацювання, автори стандарту рекомендують використовувати для навчання спеціалізовані кіберполігони та лабораторні роботи [24].

Аналіз ОПП «Безпека інформаційних і комунікаційних систем» спеціальності 125 у КНУБА показує, що ПРН включають РН10 (сучасні ІТ і методи кібербезпеки), РН12 (методи й засоби захисту інформації) та РН21 (системи виявлення несанкціонованого доступу, вразливостей і загроз). Водночас компетентності СК2, СК4 та СК10 прямо вимагають практичних навичок, які можна сформувати лише в контрольованому середовищі.

Досвід провідних університетів підтверджує ефективність кіберполігонів. Наприклад, Masaryk University інтегрував у навчання платформу KYPO, що

дозволило покращити практичні навички студентів та проводити кібернавчання у форматі Blue/Red Team з об'єктивним оцінюванням [14, 25]. Дослідники Vykopal та колеги виділяють кілька ключових факторів успіху таких занять: структуровані сценарії під конкретні цілі, градація складності, адаптація під рівень групи, а також автоматизоване розгортання середовища та оцінювання із зворотним зв'язком [25].

TalTech (Естонія) у співпраці з NATO CCDCOE(Cooperative Cyber Defence Centre of Excellence) і West Point (США) також використовують кіберполігони; інтеграція практичних вправ підвищує засвоєння матеріалу на 40–60% порівняно з лекційним форматом [12, 13].

Водночас використання наявних систем у вітчизняних ЗВО супроводжується низкою архітектурних та фінансових труднощів, які часто виникають при розгортанні подібних платформ [11, 12]. Зокрема, розгортання відомих академічних альтернатив, таких як платформа KYPO, висуває високі вимоги до локальної інфраструктури й потребує від 64 ГБ оперативної пам'яті [14], що підтверджує загальну проблему ресурсомісткості таких рішень [11]. Використання ж комерційних хмарних платформ (HTB Academy, TryHackMe) створює постійне фінансове навантаження на заклади освіти (від $14–18 на місяць за одного студента). Крім того, базові іноземні рішення, розглянуті у світовій практиці [11, 12], не інтегровані з українськими ОПП, не мають україномовного супроводу та не забезпечують наскрізного поєднання методології PTES із матрицею MITRE ATT&CK.

Отже, існує об'єктивна потреба у створенні доступного, легко відтворюваного та методично забезпеченого кіберполігону для підготовки бакалаврів за спеціальністю 125 «Кібербезпека», що й визначає головну мету цієї роботи.

1.7 Висновки до розділу та постановка задачі

У першому розділі проведено комплексний аналітичний огляд предметної області – мережевого penetration testing та кіберполігонів для навчання фахівців з кібербезпеки. За результатами аналізу встановлено наступне.

1. Пентест – комплексний метод оцінки безпеки з активною експлуатацією вразливостей; його класифікація за рівнем знань (black/white/grey box) і вектором атаки (мережевий, веб, бездротовий, соціотехнічний, фізичний) дозволяє обирати підхід. Мережевий пентест – один із найзатребуваніших і потребує систематичної практичної підготовки.

2. Найпридатнішою для навчальних сценаріїв є комбінація PTES (сім фаз процесу), MITRE ATT&CK (каталог 200+ технік з прив'язкою до APT і CVE) та Cyber Kill Chain (концептуальна модель), що поєднує засвоєння процесу з розумінням релевантності дій.

3. Аналіз існуючих кіберполігонів (Metasploitable, VulnHub, DVWA, HTB, TryHackMe, KYPO) показав, що жодне рішення не задовольняє потреб українських ЗВО одночасно: комерційні дорогі ($14–18/міс), академічні ресурсомісткі (64+ ГБ RAM), безкоштовні без сценаріїв і супроводу; жодне не адаптоване до ОПП спеціальності 125.

4. Oracle VirtualBox (Type 2, безкоштовний, кросплатформний) забезпечує оптимальне співвідношення доступності, ізоляції (Internal Network) і функціональності (snapshots, клонування) для навчального кіберполігону.

5. Open-source інструментарій Kali Linux (Nmap, Metasploit, OpenVAS, Hydra, Wireshark, Mimikatz та ін.) покриває всі сім фаз PTES і забезпечує маппінг кожної дії на техніки MITRE ATT&CK.

6. Дефіцит фахівців (4,8 млн за ISC2 2024) і розрив у практичних навичках (90% організацій) обґрунтовують актуальність розробки; вимоги NICE Framework, CompTIA PenTest+ (30% іспиту – Attacks and Exploits) і CyBOK підтверджують потребу формувати практичний досвід на бакалавраті.

На підставі аналізу сформульовано задачу: розробити комплекс навчальних сценаріїв мережевого penetration testing у віртуалізованому

середовищі на базі Oracle VirtualBox і Kali Linux, що: побудований на фазах PTES; покриває техніки MITRE ATT&CK мережевого рівня; має градацію складності (початковий → середній → просунутий); містить покрокові інструкції з україномовним супроводом; адаптований для бакалаврів спеціальності 125 КНУБА відповідно до РН10, РН12, РН21. Опис середовища й розробку сценаріїв виконано у розділах 2 і 3.

РОЗДІЛ 2. ОБҐРУНТУВАННЯ ТА РОЗРОБЛЕННЯ РІШЕННЯ

2.1 Концепція тренажера та обґрунтування підходу

На підставі аналізу в Розділі 1 встановлено, що існуючі кіберполігони не забезпечують одночасно доступності, україномовного методичного супроводу та адаптації до ОПП спеціальності 125, а просте розгортання вразливих VM (Metasploitable, DVWA) не дає систематичного оцінювання прогресу й вимагає значних зусиль на технічну підготовку. Тому запропоновано програмний тренажер, що автоматизує розгортання середовища, надає покрокові інструкції, перевіряє виконання завдань і генерує звіти про результати.

Концепція тренажера. Тренажер – це локальний вебзастосунок, що запускається на робочій станції студента й керує віртуальними машинами VirtualBox через команди VBoxManage. Взаємодія відбувається через браузер за адресою localhost на робочій станції студента й не потребує зовнішнього сервера. Підхід поєднує зручність вебтехнологій з повною автономністю: без інтернет-з’єднання, зовнішніх серверів і хмарних сервісів.

Типовий сценарій використання тренажера включає такі послідовні кроки:

1. Запуск застосунку: Студент ініціює роботу тренажера за допомогою ярлика на робочому столі, після чого автоматично відкривається веббраузер із інтерфейсом системи.
2. Вибір завдання: У головному меню користувач обирає необхідний практичний сценарій.
3. Автоматичне розгортання: Тренажер самостійно запускає відповідні VM у VirtualBox, автоматично відновлюючи їх до початкового еталонного стану (snapshot).
4. Виконання сценарію: Студент отримує покрокову інструкцію та починає проведення атак із атакуючої машини Kali Linux на цільові хости в ізольованій мережі.

5. Валідація результатів: Для підтвердження успішного злому користувач вводить знайдені прапорці (flags) у спеціальне поле вебсервісу для автоматичної перевірки.
6. Оцінювання та звітність: Система нараховує відповідні бали та автоматично генерує підсумковий PDF-звіт, який студент надсилає викладачеві для перевірки.

Обґрунтування вибору вебінтерфейсу. Розглянуто чотири підходи до взаємодії з тренажером – консольний (CLI), десктопний на Tkinter, вебінтерфейс на Flask та PowerShell-скрипти; їх порівняння наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння варіантів реалізації інтерфейсу тренажера

Критерій	CLI	Tkinter	Flask	PowerShell	Обрано
Зручність для студента	Низька	Середня	Висока	Низька	Flask
Крос-платформність	Так	Так	Так	Ні	Flask
Можливість візуалізації	Обмежена	Середня	Повна	Обмежена	Flask
Складність реалізації	Низька	Середня	Середня	Низька	–
Відображення документації	Ні	Обмежене	HTML+CSS	Ні	Flask
Можливість розширення	Низька	Середня	Висока	Низька	Flask

Таким чином, для реалізації тренажера було обрано вебінтерфейс на базі мікрофреймворку Flask. Він забезпечує найкращу взаємодію з користувачем (UX) завдяки можливості повноцінної візуалізації за допомогою сучасних вебтехнологій (HTML5, CSS3, JavaScript), а також дозволяє гнучко відображати форматовані інструкції, таблиці та графічні схеми завдань. Платформа легко розширюється новими навчальними сценаріями без необхідності модифікації коду самої графічної оболонки та стабільно працює у будь-якій операційній системі, де встановлено інтерпретатор Python і веббраузер. Локальний запуск

застосунку на localhost дозволяє повністю зберегти автономність, ізольованість та безпеку, які притаманні класичним десктопним програмам.

Принципи керування ВМ. Тренажер автоматизовано керує VM через утиліту VBoxManage [16], виконуючи її команди механізмом *subprocess* мови Python й абстрагуючи управління у графічний інтерфейс. Основні команди: *VBoxManage startvm <name> --type gui* (запуск), *VBoxManage controlvm <name> poweroff* (зупинка), *VBoxManage snapshot <name> restore <snapshot>* (відновлення стану зі snapshot) [16].

Мінімальні вимоги до системи студента. Тренажер проєктується з розрахунку на типовий ноутбук студента з Windows 10/11 або Linux. Апаратні та програмні вимоги наведено у таблиці 2.2.

Таблиця 2.2 – Системні вимоги для роботи тренажера

Параметр	Мінімум	Рекомендовано
Процесор (CPU)	4-ядерний x64 з обов'язковою підтримкою технологій віртуалізації Intel VT-x або AMD-V	6+ ядерний Intel Core i5 / AMD Ryzen 5 або новіший
Оперативна пам'ять (RAM)	8 ГБ (3.75 ГБ виділено під чотири ВМ, 4.25 ГБ під хостову ОС та локальний вебсервер	16 ГБ (Дозволяє виділити 4 ГБ під Kali Linux та прискорити виконання сканувань)
Вільний дисковий простір	30 ГБ на накопичувачі типу SSD (з урахуванням стиснення образів ВМ та динамічних дисків гіпервізора)	60 ГБ
Операційна система (Host)	Windows 10 / 11 (64-bit) або Ubuntu Linux версії 22.04+	Windows 11 (64-bit) або актуальна версія Ubuntu Linux LTS
Інтерпретатор Python	Python версії 3.10+	Python версії 3.12+
Гіпервізор VirtualBox	VirtualBox 7.0+	VirtualBox 7.1+
Браузер	Google Chrome / Mozilla Firefox / Microsoft Edge (актуальна версія)	Google Chrome / Mozilla Firefox / Microsoft Edge (актуальна версія)

Таким чином, концепція поєднує зручність вебінтерфейсу та розширюваність з автономністю локального розгортання й контролем над середовищем через VBoxManage: студент отримує готовий інструмент, а викладач – засіб систематичного оцінювання.

2.2 Архітектура тренажера

Архітектура тренажера побудована за трирівневою моделлю [17], що забезпечує чітке розділення обов’язків, спрощує підтримку коду й дозволяє змінювати окремі рівні незалежно.

Тренажер складається з трьох рівнів: представлення (вебінтерфейс у браузері), логіки (Flask-сервер, що обробляє HTTP-запити та реалізує основну логіку) і керування ресурсами (взаємодія з VirtualBox через VBoxManage та сховище даних – SQLite і JSON-файли сценаріїв).

Загальну архітектуру тренажера представлено на рисунку 2.1.

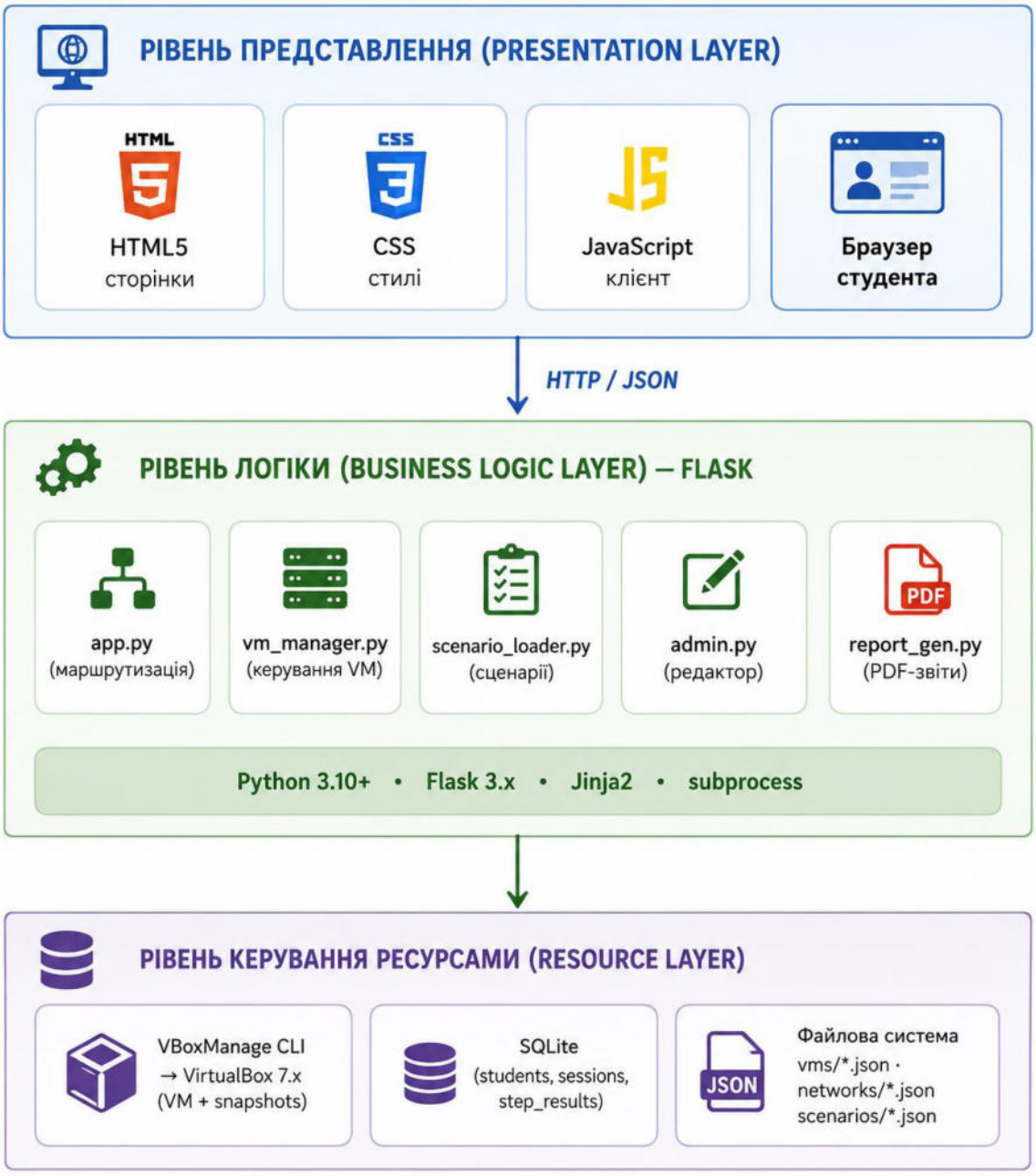


Рисунок 2.1 – Трирівнева архітектура тренажера

Рівень представлення – набір HTML-сторінок зі стилізацією CSS та інтерактивністю JavaScript: авторизація студента; головна сторінка зі списком сценаріїв; сторінка сценарію (інструкції, форма введення флагів, індикатор прогресу); сторінка статусу VM (запуск, зупинка, відкат до snapshot); історія виконаних сценаріїв зі звітами; розділ адміністрування для викладача (підрозділ 2.11). Відображення – у браузері за адресою *http://localhost:5000*.

Рівень логіки реалізований на Python [28] з мікрофреймворком Flask [27], обраним за простоту, мінімум залежностей і поширеність в освітніх проєктах [17]. Основні модулі: маршрутизація (app.py); управління VM (vm_manager.py) – обгортка над VBoxManage; завантаження сценаріїв (scenario_loader.py); адміністрування (admin.py) – ендпоінти редактора для викладача; звітність (report_generator.py) – генерація PDF через reportlab.

Рівень керування ресурсами включає інтерфейс до VirtualBox через VBoxManage (запуск, зупинка, snapshots) та локальне сховище – SQLite (прогрес студента, результати, часові мітки) і файлову систему JSON-сценаріїв. SQLite обрано як вбудовану БД без окремого сервера, що відповідає принципу автономності.

Схему взаємодії компонентів при виконанні типової операції «запуск сценарію студентом» представлено на рисунку 2.2.

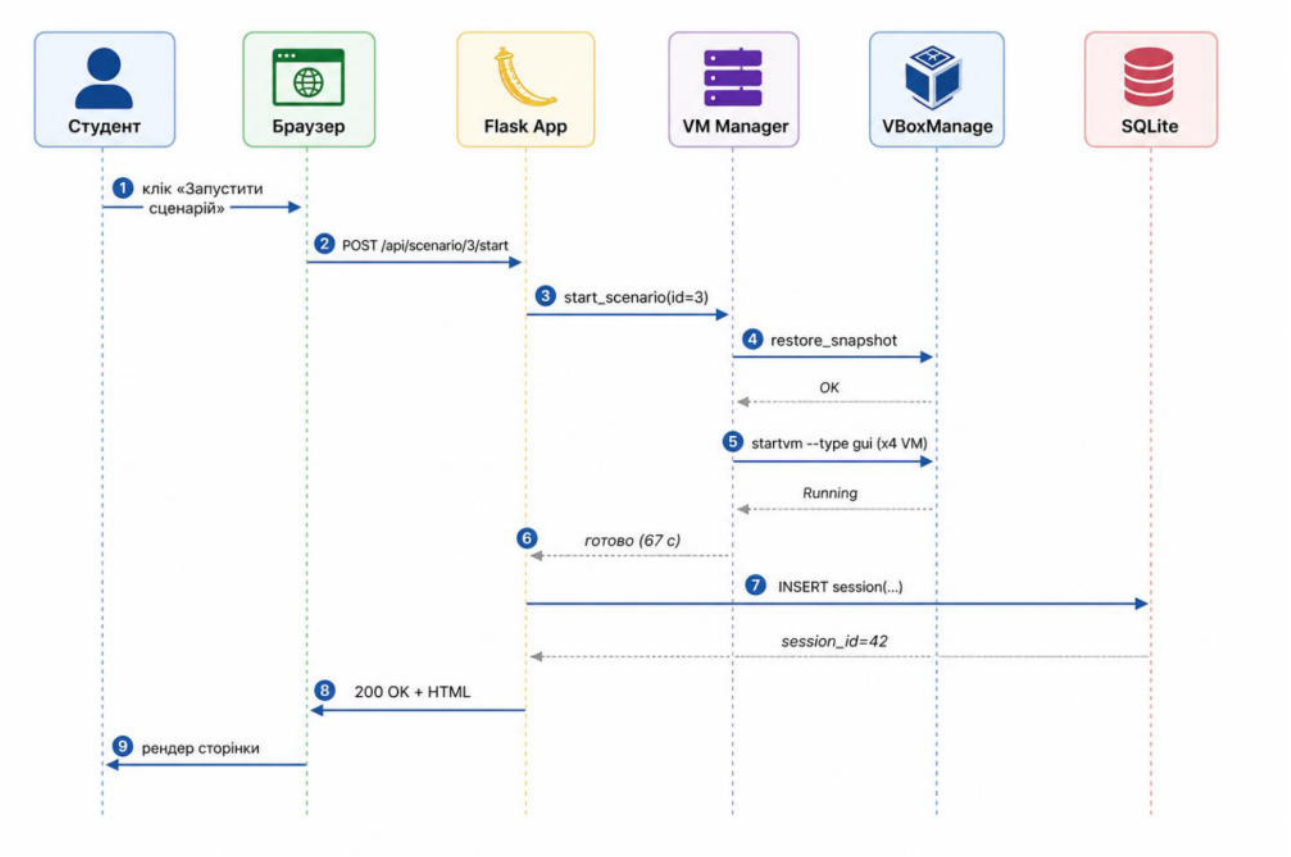


Рисунок 2.2 – Діаграма послідовності: запуск сценарію студентом

За діаграмою послідовності процес такий: студент натискає «Запустити сценарій» → браузер надсилає HTTP POST (/api/scenario/start) → Flask викликає vm_manager → той виконує команди VBoxManage через subprocess → VirtualBox запускає VM і повертає статус → Flask оновлює стан у SQLite і повертає результат → браузер показує сторінку сценарію. Процес займає 30 с – 2 хв залежно від кількості VM.

Повний перелік технологій та бібліотек, що використовуються у тренажері, наведено у таблиці 2.3.

Таблиця 2.3 – Технологічний стек тренажера

Компонент	Технологія	Призначення	Версія
Мова програмування	Python	Основна мова реалізації	3.10+
Вебфреймворк	Flask	HTTP-сервер, маршрутизація	3.0+
База даних	SQLite	Зберігання прогресу та результатів	3.x
Шаблонізатор	Jinja2	Генерація HTML-сторінок	3.x
Керування VM	VBoxManage	CLI-утиліта VirtualBox	7.0+
Генерація PDF	reportlab	Створення звітів	4.x
Frontend	HTML5 + CSS3 + JS	Клієнтський інтерфейс	–
Формат сценаріїв	JSON	Опис структури сценаріїв	–

Вибір технологій. Python обрано за поширеність в освіті, багатство бібліотек автоматизації та зручність роботи з subprocess; Flask – за мінімалістичний, але достатній функціонал без надлишкової складності; SQLite – оптимальне локальне сховище для автономного застосунку. Від клієнтських JavaScript-фреймворків (React, Vue) вирішено відмовитися, оскільки обсяг клієнтського коду незначний і достатньо засобів HTML5 та стандартного JavaScript.

Отже, запропонована трирівнева архітектура забезпечує чітке розділення обов’язків між компонентами системи, базується на використанні перевірених та доступних технологій, а також повністю зберігає автономність тренажера, оскільки всі його елементи функціонують виключно у локальному середовищі на робочій станції користувача.2.3 Опис віртуалізованого середовища

2.3 Опис віртуалізованого середовища

Віртуалізоване середовище побудоване за принципом сегментації мережі й моделює типову корпоративну інфраструктуру з DMZ та внутрішньою

мережею, що дозволяє практикувати ключові техніки мережевого пентесту, зокрема латеральний рух (pivot) між сегментами [2, 9].

Структура віртуального середовища. Середовище складається з чотирьох віртуальних машин, кожна з яких виконує свою роль у навчальних сценаріях:

1) Kali Linux 2024.x – атакуюча машина (2048 МБ RAM, 2 ядра, 40ГБ), містить понад 600 інструментів пентесту (Nmap, Metasploit, Hydra, Wireshark тощо) [21]; студент працює безпосередньо в ній.

2) pfSense CE – міжмережевий екран і маршрутизатор (512 МБ, 1 ядро, 5 ГБ); сегментує мережу на DMZ та Internal і блокує прямий доступ з DMZ до Internal, що робить pivot обов'язковим для сценаріїв просунутого рівня [14].

3) Metasploitable 2 – ціль сценаріїв початкового та середнього рівня (512 МБ, 1 ядро, 8 ГБ) з множиною вразливих сервісів (vsftpd, Samba, UnrealIRCd, Apache Tomcat, MySQL) і задокументованими CVE [15]; містить вразливий вебдодаток DVWA на порту 80 для сценарію середнього рівня [6].

4) DC-1 (VulnHub) [31] – ціль просунутого-сценарію (512 МБ, 1 ядро, 7 ГБ): сервер з CMS Drupal 7 і критичною вразливістю Drupalgeddon2 (CVE-2018-7600) [35] та кількома шляхами підвищення привілеїв [11].

Сумарні вимоги всіх VM одночасно: 3584 МБ RAM, 5 ядер, 33 ГБ диску; на хості з 16 ГБ RAM лишається ~12 ГБ для ОС і тренажера. Ціль середнього-сценарію (DVWA) розгорнута не окремою VM, а сервісом на Metasploitable 2 – це знижує вимоги до ресурсів і точніше моделює реальний веб-сервер у складі інфраструктури.

Мережева топологія. Середовище побудоване на двох ізольованих VirtualBox-мережах, з'єднаних через pfSense. Підмережа DMZ (192.168.1.0/24): Kali Linux (192.168.1.10), Metasploitable 2 (192.168.1.100),(10.10.10.50/24), WAN-інтерфейс pfSense (192.168.1.1). Підмережа Internal (10.10.10.0/24): LAN-інтерфейс pfSense (10.10.10.1) та DC-1 (10.10.10.100). Файрвол pfSense блокує прямий доступ DMZ→Internal, змушуючи використовувати Техніки мережевого просування (pivot) через скомпрометовану Metasploitable 2.

Схему мережевої топології середовища представлено на рисунку 2.3.

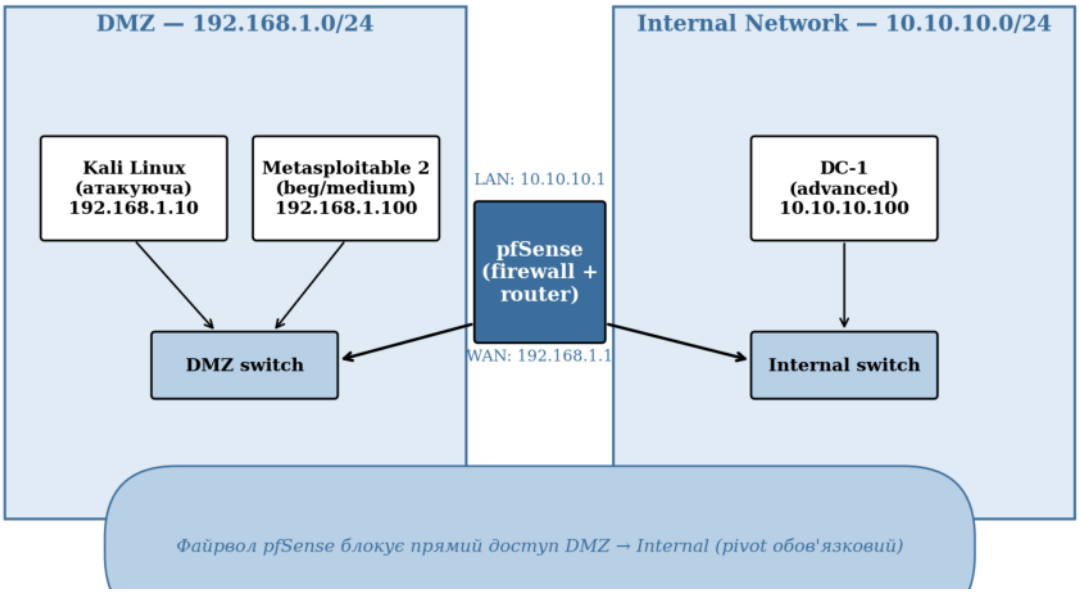


Рисунок 2.3 – Мережева топологія віртуального середовища

Конфігурація мережевих адаптерів. Кожна VM має один-два адаптери типу Internal Network відповідно до ролі: pfSense – два (dmz-net та internal-net), Kali Linux і Metasploitable 2 – лише в DMZ, DC-1 – лише в Internal. Повну конфігурацію наведено у таблиці 2.4.

Таблиця 2.4 – Мережева конфігурація віртуальних машин

VM	Адаптер 1	Адаптер 2	IP-адреса
Kali Linux	Internal (dmz-net)	–	192.168.1.10/24
pfSense	Internal (dmz-net)	Internal (internal-net)	WAN: 192.168.1.1, LAN: 10.10.10.1
Metasploitable 2	Internal (dmz-net)	–	192.168.1.100/24 10.10.10.50/24
DC-1	Internal (internal-net)	–	10.10.10.100/24

Ізоляція середовища. Тип Internal Network у VirtualBox не зв’язує віртуальну мережу зі стеком хостової ОС [16], що гарантує безпеку хоста й

зовнішньої мережі від навчальних атак, відсутність впливу зовнішнього трафіку та автономну роботу без інтернету.

Механізм snapshots. Кожна VM має заздалегідь підготовлений snapshot clean_state, створений під час налаштування стенду. Перед запуском сценарію він відновлює його (VBoxManage snapshot <name> restore clean_state), гарантуючи однаковий початковий стан для кожної спроби й повторне проходження без ручного скидання. Детальна процедура описана у підрозділі 2.12.

Отже, середовище моделює корпоративну мережу з двома сегментами, забезпечує повну ізоляцію та автоматизоване розгортання через snapshots [29], створюючи технічну основу для навчальних сценаріїв.

2.4 Модуль управління віртуальними машинами

Модуль управління ВМ (vm_manager) забезпечує взаємодію з гіпервізором через VBoxManage і реалізований як Python-клас VMManager, що інкапсулює команди VBoxManage в об’єктно-орієнтований інтерфейс [16].

Функціональні вимоги модуля: перевірка наявності зареєстрованих у VirtualBox машин; запуск у режимах gui, headless або separate; коректна зупинка; створення та відновлення snapshots; отримання статусу й IP-адреси VM для відображення в інтерфейсі.

Архітектура модуля. Клас VMManager реалізує шаблон Facade над API VBoxManage: кожна операція – окремий метод, що формує команду, виконує її через subprocess.run(), обробляє вивід і повертає Python-об’єкти. Помилки обробляються власними виключеннями (VMNotFoundError, VMError, VBoxNotInstalledError).

Основні методи класу VMManager наведено у таблиці 2.5.

Таблиця 2.5 – Основні методи класу VMManager

Метод	Команда VBoxManage	Призначення
list_vms()	list vms	Отримати перелік імпортованих VM
vm_exists(name)	showvminfo <name>	Перевірити наявність VM
start_vm(name, mode)	startvm <name> --type <mode>	Запустити VM у gui/headless/separate
stop_vm(name)	controlvm <name> acpipowerbutton	Коректна зупинка VM
force_stop(name)	controlvm <name> poweroff	Примусова зупинка VM
create_snapshot(name, snap)	snapshot <name> take <snap>	Створити snapshot
restore_snapshot(name, snap)	snapshot <name> restore <snap>	Відновити snapshot
get_vm_state(name)	showvminfo <name>	Отримати стан VM

Виявлення та контроль VM. При запуску тренажер зіставляє кожен профіль із переліком зареєстрованих у VirtualBox машин (VBoxManage list vms) за іменем. Реєстрація самих машин (імпорт OVA, встановлення, налаштування мережі та створення snapshot clean_state) виконується одноразово під час підготовки стенду (підрозділ 2.12). Якщо потрібної машини у VirtualBox немає, тренажер позначає її як недоступну; автоматичного імпорту образів тренажер не виконує.

Асинхронність операцій. Тривалі операції (запуск і зупинка VM, відновлення snapshot – 30 с–5 хв) виконуються в окремих потоках , а вебінтерфейс отримує статус через AJAX: студент бачить прогрес-бар, що оновлюється кожні 2 с, і поточну операцію.

Обробка помилок. VMManager опрацьовує відсутність VirtualBox, брак RAM, відсутність потрібної машини у VirtualBox, конфлікт імен VM і проблеми з мережевими адаптерами, формуючи для кожного випадку зрозуміле повідомлення українською з рекомендаціями.

Режими запуску VM. Модуль підтримує всі три режими VirtualBox: gui (VM у власному вікні – рекомендований для навчання, бо студент бачить стан цілі), headless (у фоні, для серверних цілей) і separate (фон з можливістю підключення вікна).

Діаграму станів VM у контексті тренажера представлено на рисунку 2.4.

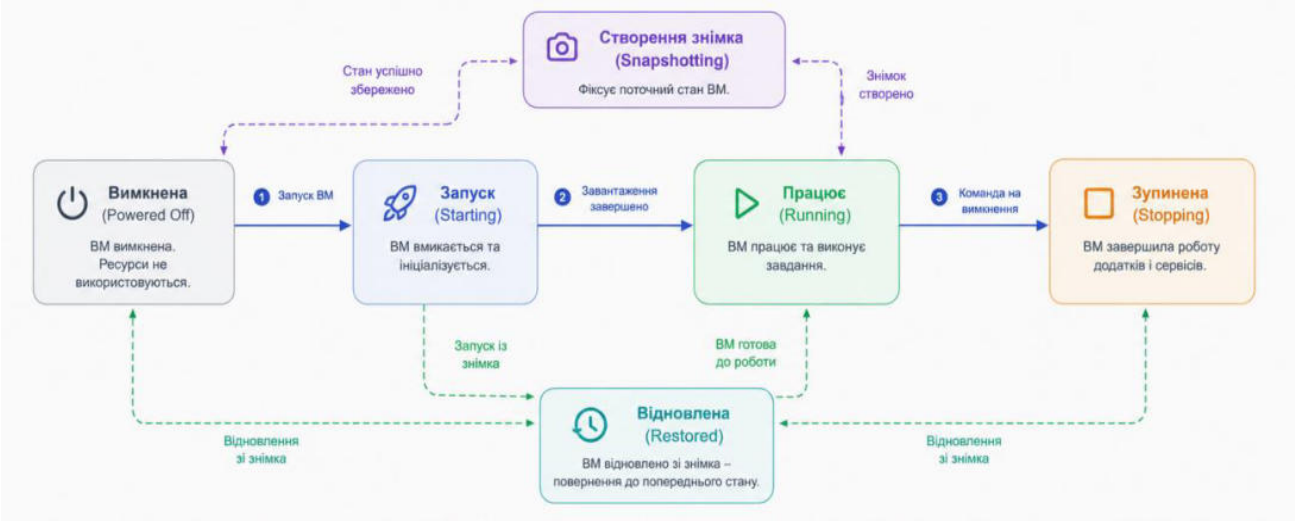


Рисунок 2.4 – Діаграма станів VM у тренажері

2.5 Модуль сценаріїв

Модуль сценаріїв (*scenario_loader*) завантажує, парсить і виконує сценарії. Ключове рішення – зберігання сценаріїв у JSON як окремих файлів, що дає змогу додавати контент без зміни коду.

Структура JSON-файлу сценарію. Кожен сценарій описується одним JSON-файлом з метаданими, переліком потрібних VM, послідовністю кроків з інструкціями й правильними відповідями та параметрами оцінювання. Опис основних полів наведено у таблиці 2.6.

Таблиця 2.6 – Поля JSON-файлу сценарію

Поле	Тип	Призначення	Обов’язкове
id	string	Унікальний ідентифікатор сценарію	Так
title	string	Назва сценарію для відображення	Так
level	enum	Рівень: beginner / medium / advanced	Так
duration_minutes	integer	Орієнтовна тривалість виконання	Так
required_vms	array	Перелік необхідних VM (id з vms/)	Так
required_snapshots	object	`VM → snapshot` для відновлення	Так
network	string	Id мережевої топології з networks/	Ні
learning_objectives	array	Навчальні цілі сценарію	Так
attck_techniques	array	Перелік технік MITRE ATT&CK	Так
steps	array	Кроки сценарію (об’єкти)	Так
theory_intro	string	Теоретичний вступ (HTML)	Ні
max_attempts	integer	Максимум спроб на крок	Ні

Кожен крок (об’єкт у масиві steps) має обов’язкові поля: number, title, instruction, flag_prompt, flag_answer, points, attck_technique; та опціональні: hint (знижує бали), expected_command, control_question, validator_type, case_sensitive, options (для multiple_choice).

Модуль підтримує п’ять типів валідаторів – від точного збігу рядка до числового діапазону – що дозволяє точно формулювати завдання. Перелік і характеристику наведено у таблиці 2.7.

Таблиця 2.7 – Типи валідаторів відповідей у модулі сценаріїв

validator_type	**Призначення**	**Приклад flag_answer**	**Що приймає валідатор**
text_flag	Точне співпадіння рядка (за замовчуванням)	"Apache"	Apache
regex	Регулярний вираз \	"^192\.168\.1\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)$$"	Будь-який IP у мережі 192.168.1.0/24
numeric_range	Число у діапазоні	"5-10"	Будь-яке в діапазоні 5-10
case_insensitive_substring	Підрядок без врахування регістру	"vsFTPd"	vsftpd , VSFTPD
multiple_choice	Вибір з options (підтримує і назву, і номер)	"MD5" + options=[bcrypt, MD5, ...]	MD5 або bdcrypt

Розширений набір валідаторів вирішує проблему залежності точної відповіді від snapshot-стану VM: numeric_range «5-10» задає завдання без прив’язки до конкретного числа (кількість користувачів у /etc/passwd), regex приймає будь-який IP з мережі , а *case_insensitive_substring – banner*-рядки сервісів незалежно від регістру.

Нижче наведено фрагмент JSON-файлу для одного з реалізованих сценаріїв, що ілюструє типову структуру кроку з нестандартним валідатором numeric_range:

```
{
  "id": "network_recon_deep",
  "title": "Сценарій 4: Глибока мережева розвідка та enumeration",
  "level": "medium",
  "duration_minutes": 120,
  "required_vms": ["kali", "metasploitable2"],
  "required_snapshots": {
    "kali": "clean_state",
    "metasploitable2": "clean_state"
  },
  "attck_techniques": ["T1595", "T1046", "T1018"],
  "steps": [
    {
      "number": 1,
      "title": "Швидке SYN-сканування топ-1000 портів",
      "instruction": "<p>Запустіть SYN-сканування проти цілі...</p>",
      "expected_command": "sudo nmap -sS -T4 192.168.1.100",
      "flag_prompt": "Скільки портів виявлено зі статусом open?",
      "flag_answer": "20-25",
      "points": 10,
      "validator_type": "numeric_range",
      "attck_technique": "T1046",
      "hint": "У зведеному рядку nmap...",
      "control_question": "Чим SYN-сканування відрізняється від Connect?"
    }
        ]
}
```

Формат JSON достатньо виразний для опису всіх аспектів сценарію. Поля theory_intro та instruction допускають HTML-теги для форматування тексту, посилань і зображень без зміни коду, що дає викладачу повний контроль над контентом без знання Python.

Життєвий цикл сценарію: студент обирає сценарій → тренажер перевіряє стан потрібних VM → автоматично розгортає, відновлює snapshot → показує паспорт сценарію → послідовно веде по кроках → валідатор перевіряє відповідь і нараховує бали → при помилці дозволяє повтор до max_attempts → після всіх кроків генерує PDF-звіт.

Діаграму процесу виконання сценарію представлено на рисунку 2.5.

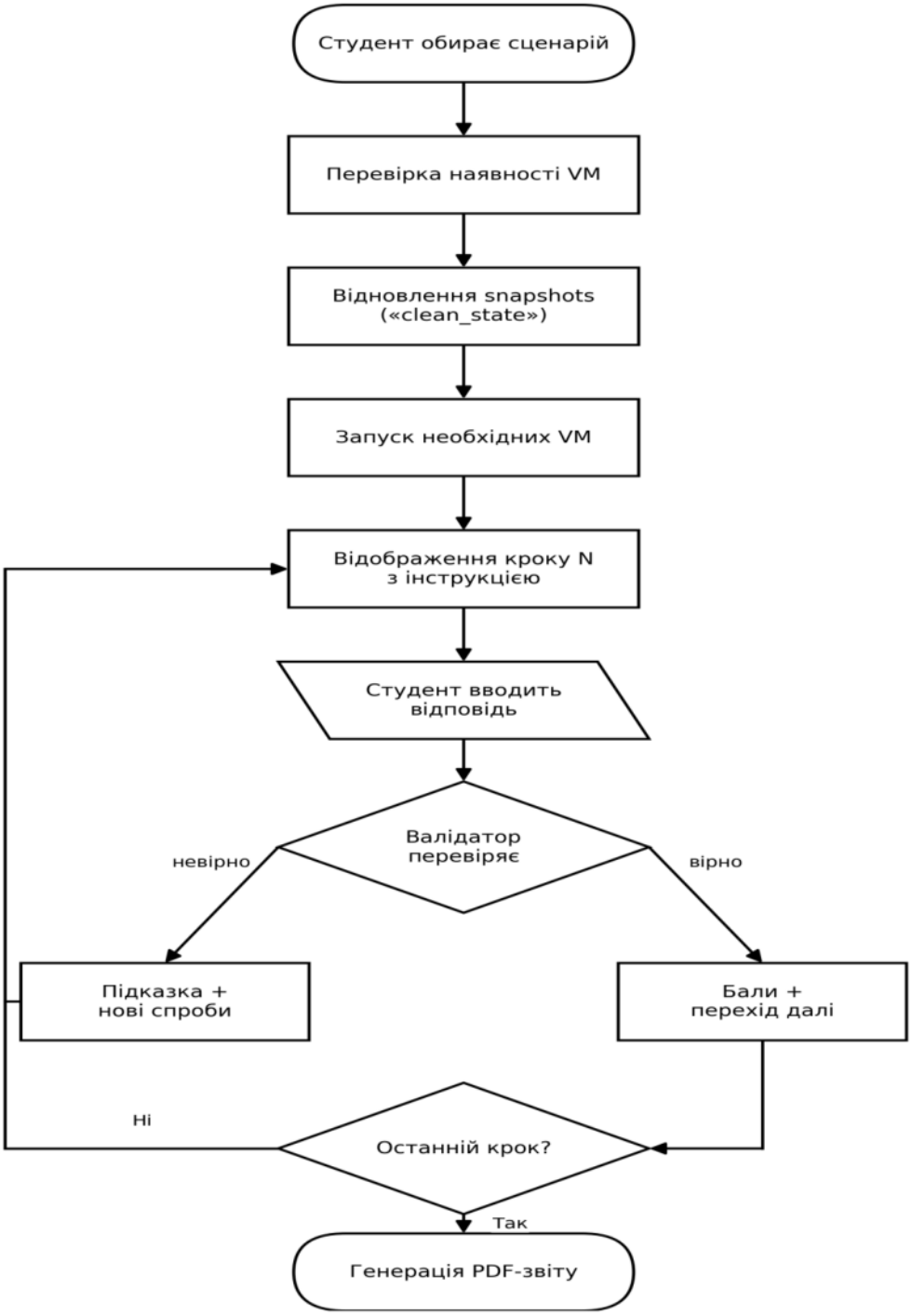


Рисунок 2.5 – Діаграма процесу виконання сценарію

Реалізовано п’ять сценаріїв, що покривають три рівні складності й сім фаз PTES. Спочатку планувалось три; після завершення модульної архітектури з CLI-інструментами обсяг розширено двома мережево-орієнтованими сценаріями (4 і 5), що демонструють модульність системи. Характеристики наведено у таблиці 2.8.

Таблиця 2.8 – Перелік реалізованих сценаріїв тренажера

№	Назва	Рівень	Тривалість	Кроків	Цільові VM
1	Базовий пентест Metasploitable 2	beginner	30 хв	10	Kali, Metasploitable2
2	Пентест веб-додатку DVWA	medium	50 хв	10	Kali, Metasploitable2
3	Багатоетапна атака з pivot DMZ→Internal	advanced	90 хв	15	Kali, Metasploitable2, pfSense, DC-1
4	Глибока мережева розвідка та enumeration	medium	30 хв	10	Kali, Metasploitable2
5	Просунуте сканування та обхід базової детекції	advanced	90 хв	10	Kali, Metasploitable2, pfSense

Сумарно реалізовано 55 кроків, що охоплюють 36 унікальних технік MITRE ATT&CK. Сценарії 1–3 покривають класичну прогресію (сервіс → веб → багатоетапна корпоративна атака), а 4 і 5 фокусуються на мережевих аспектах – глибокій розвідці (Nmap NSE, enum4linux, showmount/NFS, nikto/dirb) і обході детекції (фрагментація, decoy, idle scan, source-port spoofing, SSH dynamic forwarding).

Сумарне покриття тактик MITRE ATT&CK усіма п’ятьма сценаріями наведено у таблиці Г.3 (дод. Г). У сукупності реалізовані сценарії покривають 9 з 12 ключових тактик мережевого та системного рівнів.

Принципи проєктування сценаріїв. Зміст сценаріїв формувався не як довільний перелік завдань, а за визначеними дидактичними засадами. По-перше, дотримано послідовного ускладнення між сценаріями: від експлуатації одиничного вразливого сервісу (сценарій 1) через рівень веб-застосунку (сценарій 2) до багатоетапної атаки з транзитом до ізольованого сегмента мережі (сценарій 3), а далі — до поглибленої мережевої розвідки (сценарій 4) та ухилення від базових засобів виявлення (сценарій 5); така прогресія узгоджена з

послідовністю фаз PTES і поступово нарощує пізнавальне навантаження. По-друге, застосовано принцип гранулярності: кожен крок відповідає одній завершеній дії з однозначно перевірюваним результатом, що дає змогу локалізувати утруднення здобувача та нараховувати бали поетапно. По-третє, тип валідатора узгоджено з характером очікуваної відповіді: точний збіг — для незмінних значень (ідентифікатор модуля, версія сервісу), числовий діапазон — для величин, що залежать від стану цілі, регулярний вираз — для динамічних рядків (ідентифікатор сесії), входження підрядка без урахування регістру — для текстових ознак сервісів. По-четверте, передбачено запобігання розкриттю відповіді у підказках: формулювання підказок свідомо утримано на методологічному рівні, а їх додатково перевірено автоматизованим алгоритмом виявлення семантичних збігів між підказкою та еталонною відповіддю (підрозділ 2.10). Перелічені принципи покладено в основу методики побудови навчального сценарію, описаної нижче.

Методика побудови навчального сценарію. На основі наведених принципів сформовано відтворювану процедуру розроблення сценарію, що складається з дев’яти етапів:

1) Визначення навчальної мети та рівня складності. Обирають компетенцію, яку має сформувати сценарій, рівень складності (початковий, середній або просунутий) і перелік фаз PTES, що мають бути охоплені.

2) Побудова ланцюга атаки. На доступних вразливих цілях середовища (Metasploitable 2, DC-1 тощо) формують реалістичну послідовність дій, узгоджену з обраними фазами PTES.

3) Декомпозиція ланцюга на кроки. Ланцюг розбивають на окремі кроки за принципом гранулярності: один крок — одна завершена дія з однозначно перевірюваним результатом.

4) Відображення кроків на техніки MITRE ATT&CK. Кожному кроку ставлять у відповідність техніку матриці ATT&CK, що забезпечує покриття й подальшу звітність.

5) Вибір типу валідатора. Для кожного кроку обирають валідатор відповідно до характеру очікуваної відповіді: точний збіг — для сталих значень, числовий діапазон — для величин, що залежать від стану цілі, регулярний вираз — для динамічних рядків, входження підрядка без урахування регістру — для текстових ознак сервісів, множинний вибір — для теоретичних питань.

6) Формулювання інструкції та підказки. До кроку складають покрокову інструкцію та підказку, утримуючи останню на методологічному рівні, без прямого зазначення відповіді.

7) Перевірка на розкриття відповіді. Підказки автоматично перевіряють алгоритмом виявлення семантичних збігів (підрозділ 2.10), який фіксує ненавмисне розкриття еталонної відповіді.

8) Структурна валідація сценарію. Готовий JSON-файл перевіряють засобом validate_scenario.py на коректність полів, типів валідаторів і відповідність оголошених віртуальних машин та знімків стану.

9) Самотестування та калібрування. Сценарій проходять повністю з фіксацією часу й коректності кроків, за результатами чого уточнюють планову тривалість і складність (підрозділ 3.3).

Узагальнення проєктних рішень, отриманих застосуванням методики до кожного зі сценаріїв, наведено в таблиці 2.9.

Таблиця 2.9 – Проєктне обґрунтування навчальних сценаріїв

Сценарій	**Навчальна мета**	**Характеристика сценарію**	**Фази PTES**	**Тактики MITRE ATT&CK**
Базовий пентест Metasploitable 2	Формування навичок виконання повного циклу мережевого penetration testing окремого хоста	Розвідка мережі, ідентифікація сервісів, аналіз вразливостей, експлуатація мережевого сервісу та отримання облікових даних	Intelligence Gathering, Vulnerability Analysis, Exploitation, Post-Exploitation	Reconnaissance, Initial Access, Execution, Credential Access, Discovery, Collection
Пентест веб-застосунку DVWA	Формування практичних навичок тестування безпеки веб-застосунків	Виявлення та експлуатація веб-вразливостей, перевірка механізмів автентифікації та автоматизація атак на веб-рівні	Intelligence Gathering, Vulnerability Analysis, Exploitation, Post-Exploitation	Reconnaissance, Initial Access, Execution, Privilege Escalation, Credential Access, Discovery, Collection
Багатоетапна атака з транзитом DMZ→Internal	Формування навичок проведення багатоетапних атак та латерального руху між сегментами мережі	Компрометація вузла в DMZ, закріплення доступу, маршрутизація через скомпрометований вузол, проникнення до внутрішнього сегмента та підвищення привілеїв	Усі фази PTES	Reconnaissance, Initial Access, Execution, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Command and Control

Продовження таблиці 2.9

Поглиблена мережева розвідка	Формування навичок збору інформації про мережеву інфраструктуру та сервіси	Активна мережева розвідка, перелік ресурсів і сервісів, аналіз конфігурації мережевих служб	Intelligence Gathering, Vulnerability Analysis	Reconnaissance, Discovery, Collection
Ухилення від базового виявлення	Формування навичок застосування технік зниження помітності мережевої активності	Використання методів обходу базових засобів виявлення та аналіз мережевого трафіку з позиції захисника	Intelligence Gathering, Defense Evasion	Reconnaissance, Defense Evasion, Discovery, Collection, Command and Control

2.6 Модуль звітності та оцінювання

Модуль звітності (report_generator) автоматично генерує підсумкові PDF-звіти та нараховує бали за кроки забезпечуючи механізм оцінювання для інтеграції тренажера в навчальний процес.

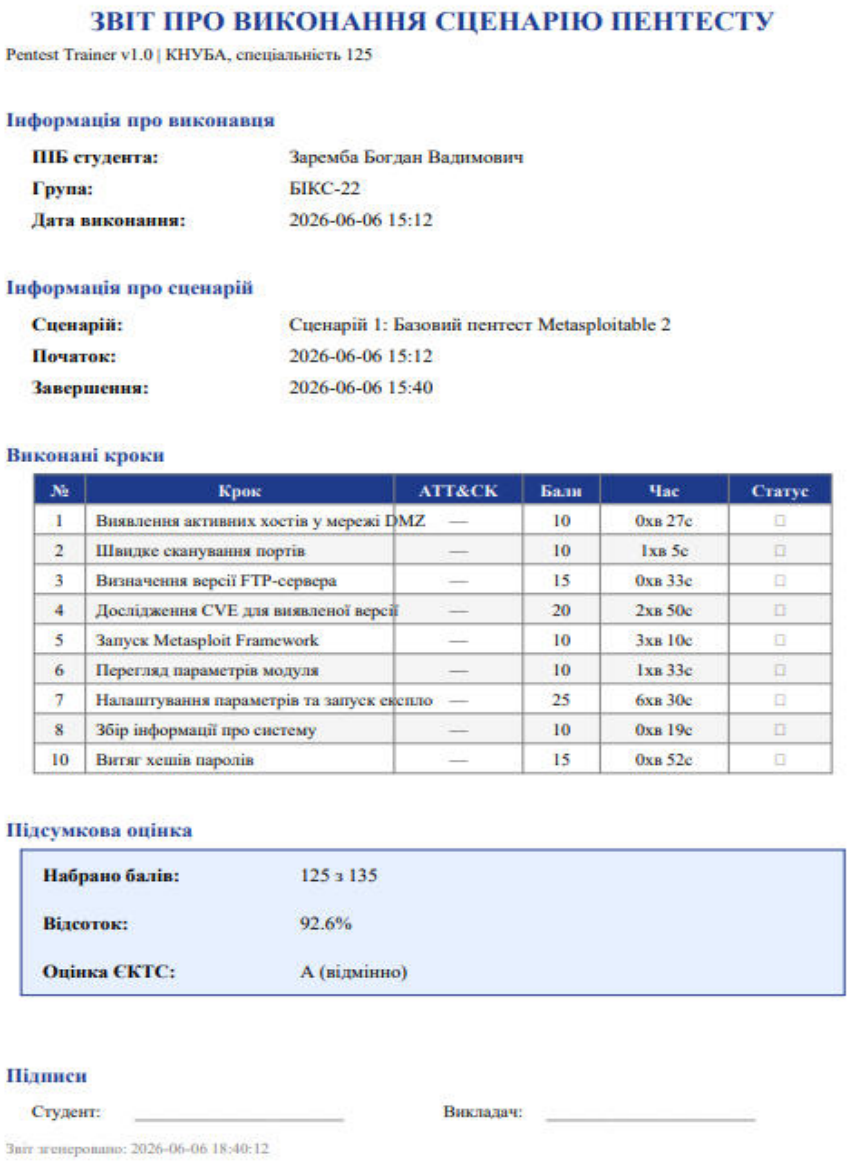
Критерії оцінювання. За кожен сценарій студент отримує 0–100 балів з урахуванням повноти виконання, кількості спроб на крок і використання підказок. Формула й коефіцієнти штрафу наведено у таблиці Г.4 (додаток Г)

Шкала оцінювання переводиться у ЄКТС: 90–100 – A, 82–89 – B, 74–81 – C, 64–73 – D, 60–63 – E, менше 60 – FX, що узгоджено з паспортом спеціальності 125 КНУБА.

Структура PDF-звіту (через reportlab): титульна сторінка з даними студента (ПІБ, група); назва сценарію й дата; паспорт сценарію (мета, складність,

тривалість, техніки ATT&CK); таблиця виконаних кроків зі статусом, балами й часом; маппінг дій на техніки ATT&CK; підсумкова оцінка в балах і ЄКТС.

Приклад структури сторінки PDF-звіту наведено на рисунку 2.6.



ЗВІТ ПРО ВИКОНАННЯ СЦЕНАРІЮ ПЕНТЕСТУ

Pentest Trainer v1.0 | КНУБА, спеціальність 125

Інформація про виконавця

ПІБ студента:	Заремба Богдан Вадимович
Група:	БІКС-22
Дата виконання:	2026-06-06 15:12

Інформація про сценарій

Сценарій:	Сценарій 1: Базовий пентест Metasploitable 2
Початок:	2026-06-06 15:12
Завершення:	2026-06-06 15:40

Виконані кроки

№	Крок	ATT&CK	Бали	Час	Статус
1	Виявлення активних хостів у мережі DMZ	—	10	0хв 27с	□
2	Швидке сканування портів	—	10	1хв 5с	□
3	Визначення версії FTP-сервера	—	15	0хв 33с	□
4	Дослідження CVE для виявленої версії	—	20	2хв 50с	□
5	Запуск Metasploit Framework	—	10	3хв 10с	□
6	Перегляд параметрів модуля	—	10	1хв 33с	□
7	Налаштування параметрів та запуск експло	—	25	6хв 30с	□
8	Збір інформації про систему	—	10	0хв 19с	□
10	Витяг хешів паролів	—	15	0хв 52с	□

Підсумкова оцінка

Набрано балів:	125 з 135
Відсоток:	92.6%
Оцінка ЄКТС:	A (відмінно)

Підписи

Студент: _____ Викладач: _____

Звіт згенеровано: 2026-06-06 18:40:12

Рисунок 2.6 – PDF-звіт про виконання сценарію

Зберігання результатів. Усі результати зберігаються у SQLite, що дозволяє переглядати історію сценаріїв студента, порівнювати спроби, формувати аналітику (середній час, найскладніші кроки) та перевіряти PDF-звіти. Схема БД має три таблиці: students (ПІБ, група), sessions (спроба з timestamp і балом), step_results (відповідь, час, використання підказки).

Схему бази даних представлено на рисунку 2.7.

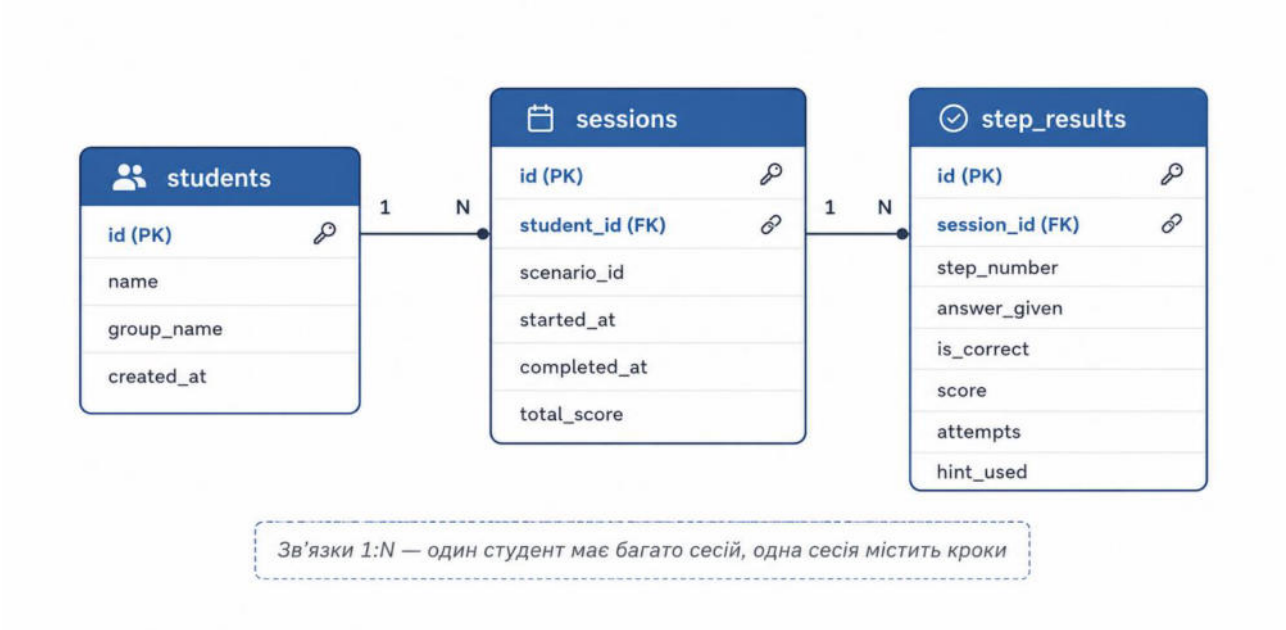


Рисунок 2.7 – ER-діаграма бази даних тренажера

2.7 Принципи модульності та розширюваності

Однією з ключових характеристик тренажера є модульність – здатність розширюватися новим контентом (сценаріями, VM, мережевими топологіями) без зміни коду. Викладач може самостійно адаптувати тренажер під свої дисципліни, додавати й змінювати матеріали.

Трирівнева модель модульності. Контент організовано у три незалежні рівні, кожен – окремими файлами: VM-профілі (описи окремих VM з прив'язкою до зареєстрованої у VirtualBox машини за іменем); мережеві профілі (варіанти мережевої організації середовища); сценарії (посилаються на VM- і мережеві профілі за ідентифікаторами). Це реалізує принцип Separation of Concerns(розмежування повноважень) і дозволяє повторно використовувати компоненти – наприклад, Metasploitable 2 задіяна у чотирьох із п'яти сценаріїв.

Структуру каталогів та зв'язки між компонентами модульної системи представлено на рисунку 2.8.

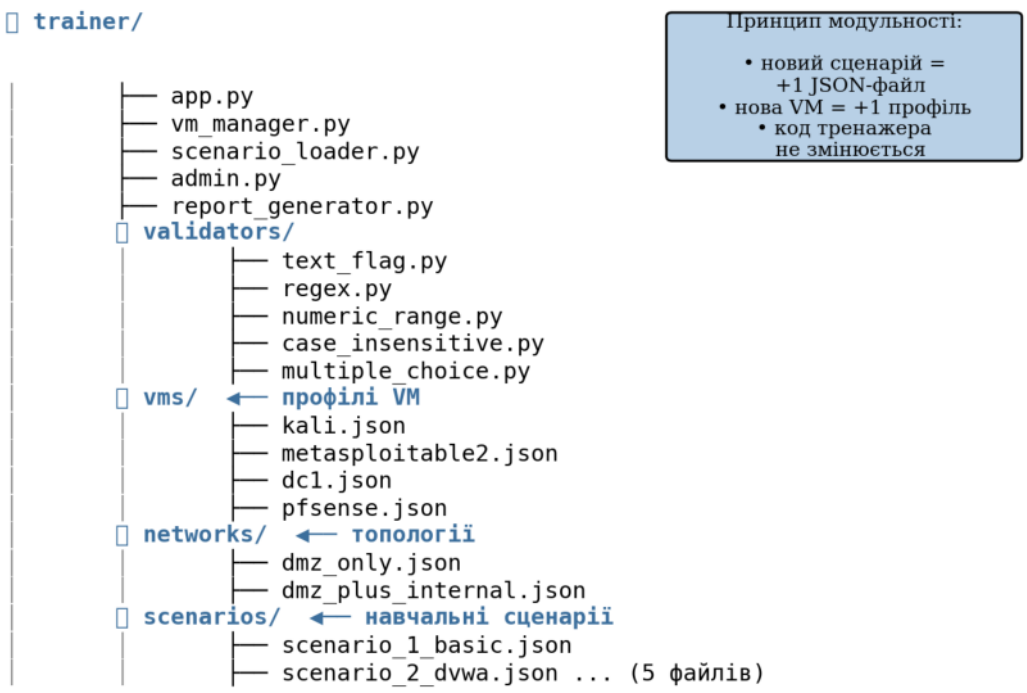


Рисунок 2.8 – Структура модульної системи тренажера

VM-профіль – JSON-файл з описом віртуальної машини: ім’я зареєстрованої у VirtualBox машини (для прив’язки за іменем), параметри ресурсів, мережеві налаштування за замовчуванням, облікові дані й категоризація. Профілі зберігаються у каталозі vms/. Основні поля наведено у таблиці Г.5 (дод. Г).

Додавання нової вразливої VM виконується у два кроки й не потребує жодного рядка коду. Спочатку машину реєструють у VirtualBox (імпортують OVA, встановлюють з ISO або беруть готовий образ) і роблять для неї snapshot clean_state. Потім у каталозі vms/ створюють JSON-профіль (через сторінку «Машини» підрозділу 2.11 або вручну), у якому ім’я машини збігається з її назвою у VirtualBox, – саме за цим іменем тренажер керує VM. При наступному запуску тренажер автоматично виявляє новий профіль і робить VM доступною у сценаріях за умови, що відповідна машина існує у VirtualBox.

Мережевий профіль описує топологію – підмережі, маршрутизатори й правила файрволу – що дозволяє визначати різні варіанти середовища (одна підмережа, дві з DMZ, три сегменти) для різних сценаріїв. Профілі зберігаються у каталозі networks/ як окремі JSON-файли.

Розглянемо чотири типові сценарії розширення тренажера новими користувачами (викладачами або іншими студентами), що демонструють переваги модульної архітектури.

Сценарій А – додавання нового сценарію на основі існуючих VM: створити JSON-файл у scenarios/ за структурою (або через генератор, підрозділ 2.10), вказати у required_vms наявні VM-профілі, описати кроки й перезапустити тренажер. Орієнтовно 30–60 хв на простий сценарій (10–15 кроків), без коду. Альтернативно – через веб-редактор (підрозділ 2.11).

Сценарій Б – додавання нової вразливої VM (наприклад, Mr-Robot з VulnHub): зареєструвати VM у VirtualBox (імпорт OVA або встановлення), зробити snapshot clean_state, створити JSON-профіль (через сторінку «Машини» чи вручну) і за потреби сценарій. Орієнтовно 15–30 хв на профіль плюс година на сценарій.

Сценарій В – нова мережева топологія (наприклад, три сегменти): створити JSON-профіль у networks/ з підмережами й маршрутизаторами, за потреби – VM-профіль маршрутизатора, і використати мережу у сценарії через поле network. Орієнтовно 30–60 хв, без коду.

Сценарій Г – додавання нового типу валідатора (якщо п’яти наявних бракує): додати тип у VALID_VALIDATOR_TYPES класу Step, логіку у check_answer() і валідацію формату у CLI-валідатор. Це єдиний сценарій розширення, що потребує навичок програмування.

Узагальнена характеристика різних варіантів розширення тренажера за рівнем складності та потребою у програмуванні наведена у таблиці Г.6 (дод. Г).

Найбільш простим та інтуїтивно зрозумілим способом модифікації, налаштування та додавання нових навчальних сценаріїв у тренажері є вбудований веб-редактор, детально описаний у підрозділі 2.11. Завдяки наявності цього інструменту, створення та редагування будь-якого практичного завдання можна реалізувати взагалі без знання синтаксису JSON або навичок програмування — процес відбувається виключно через заповнення інтерактивних форм у вікні браузера. Це суттєво знижує поріг входження та

робить платформу повністю придатною для самостійного використання викладачами без залучення сторонніх розробників. Консольні інструменти керування (розглянуті у підрозділі 2.10) виступають додатковим засобом автоматизації, тоді як уся базова робота із розширення контенту перенесена у зручний графічний вебінтерфейс.

Отже, розроблена архітектура забезпечує модульність системи на двох рівнях:

рівень контенту (покриває близько 90% потреб у розширенні лабораторної бази) — реалізується через редагування конфігураційних файлів формату JSON без необхідності програмування, із можливістю використання вбудованого веб-редактора;

рівень функціональності (охоплює приблизно 10% складніших випадків модифікації) — передбачає внесення невеликих змін до вихідного коду фіксованого обсягу.

Такий підхід робить програмний тренажер повністю придатним для довгострокового розвитку, масштабування та самостійної підтримки силами кафедри без залучення сторонніх фахівців.

2.8 Проєктування користувацького інтерфейсу

Користувацький інтерфейс реалізований як вебзастосунок з кількома логічно відокремленими екранами; навігація – через верхнє меню. Інтерфейс поділено на дві зони: студентську та викладацьку (доступ за паролем викладача).

Основні екрани студентського інтерфейсу. Студентська зона включає шість основних екранів, навігація між якими організована через верхнє меню.

1) Сторінка авторизації – при першому запуску студент вводить ПІБ і групу, що зберігаються у Flask-сесії на час роботи процесу. SECRET_KEY генерується випадково при кожному запуску, тому авторизація запитується разово на сесію.

2) Головна сторінка (Dashboard) – список сценаріїв картками (назва, складність, тривалість, кнопка «Запустити») та статистика: кількість виконаних сценаріїв, середній бал.

3) Сторінка сценарію (Scenario View) – основний робочий екран із трьома зонами: ліворуч поточний крок з інструкцією, у центрі поле відповіді й кнопки, праворуч прогрес-бар і перелік виконаних кроків. Поле відповіді адаптується під тип валідатора поточного кроку: для multiple_choice показується випадаючий список варіантів, для numeric_range — числове поле (студент вводить одне число), для решти типів — текстове поле.

4) Сторінка управління VM (VM Control) – перегляд стану всіх VM і ручні операції (запуск, зупинка, відкат до snapshot); корисна для налагодження та у разі збоїв.

5) Сторінка історії (History) – архів усіх виконаних спроб з можливістю перегляду результатів та завантаження PDF-звітів.

6) Сторінка налаштувань (Settings) – редагування даних студента (ПІБ, група) і режиму запуску VM (gui/headless/separate).

Окрім цих екранів, тренажер має панель адміністратора /admin/* – редактор сценаріїв для викладача (підрозділ 2.11), захищену окремою авторизацією за паролем (TEACHER_PASSWORD); маршрут /admin/* виключено зі студентської аутентифікації.

Макет головної сторінки тренажера представлено на рисунку 2.9.

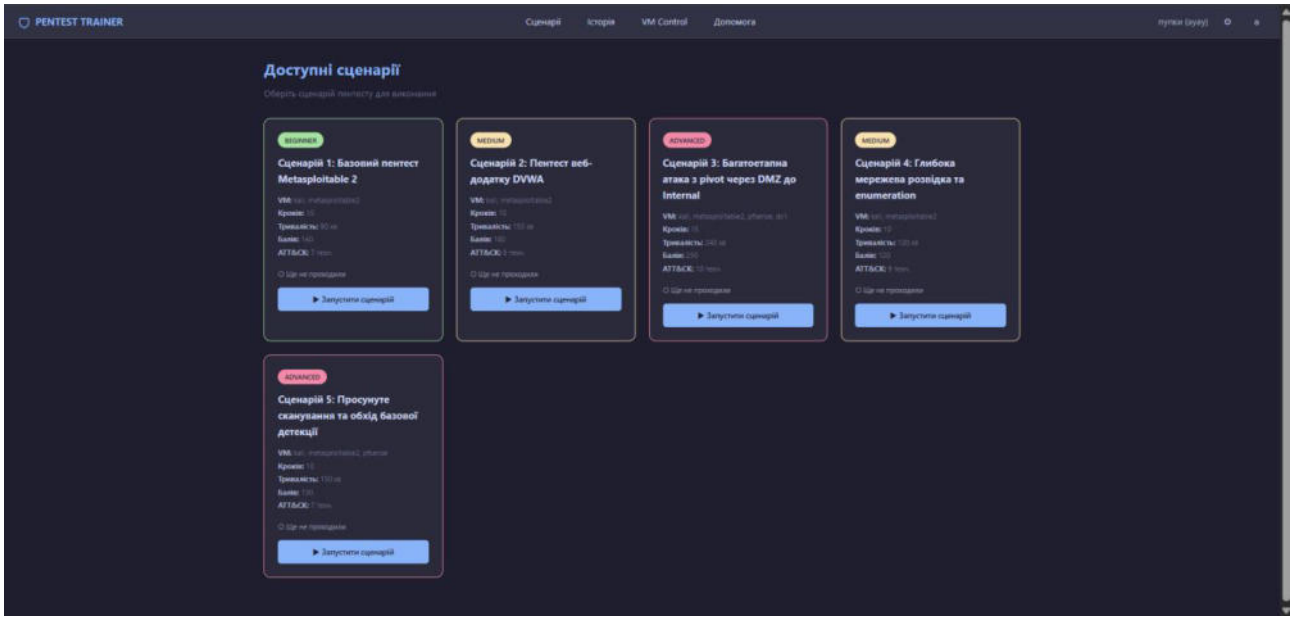


Рисунок 2.9 – Макет головної сторінки тренажера

Принципи дизайну: мінімалізм (без зайвих елементів); адаптивність (коректне відображення від 1366×768 до 4K); контраст (темна тема з читабельними кольорами для тривалої роботи).

Колірна схема. Темна тема із синьо-сірою палітрою, типовою для інструментів кібербезпеки (Kali Linux, Metasploit): фон *#1e1e2e*, текст *#cdd6f4*, акцент *#89b4fa*, успіх *#a6e3a1*, помилка *#f38ba8*, попередження *#f9e2af* – для комфортної тривалої роботи.

Перелік основних ендпоінтів REST API тренажера, що використовуються інтерфейсом, наведено у таблиці Г.7 (дод. Г).

Усі ендпоінти повертають JSON (окрім /api/report – PDF). Використовуються стандартні HTTP-коди: 200 (успіх), 400 (некоректні дані), 401 (потрібна авторизація), 404 (не знайдено), 409 (збереження заблоковано попередженнями валідатора), 500 (внутрішня помилка з описом у тілі).

2.9 Безпека та обмеження тренажера

Оскільки тренажер виконує реальні атаки (експлуатація вразливостей, Metasploit, брутфорс) у навчальному середовищі, особливу увагу приділено

безпеці хостової машини й зовнішніх мереж. Нижче розглянуто механізми ізоляції, ризики та обмеження функціоналу.

Багаторівнева ізоляція. Тренажер реалізує три рівні ізоляції, що в сукупності виключають вплив навчальних атак на хост і зовнішні мережі.

Перший рівень – ізоляція через гіпервізор. Усі вразливі VM запускаються у VirtualBox з апаратною ізоляцією (VT-x/AMD-V): шкідливий код усередині VM не має прямого доступу до пам’яті, файлів чи процесів хоста й не може вийти за межі гіпервізора (за відсутності VM escape, виправлених в актуальній версії VirtualBox) [16].

Другий рівень – ізоляція через Internal Network. Усі адаптери цільових VM налаштовані як Internal Network: трафік циркулює лише між VM однієї віртуальної мережі й не виходить на стек хостової ОС; VM не мають доступу до інтернету, локальної мережі студента й хоста; зовнішні зловмисники не можуть отримати до них доступ [16].

Третій рівень – ізоляція через pfSense. Між DMZ (де атакуюча Kali Linux) та Internal встановлено міжмережевий екран pfSense, що блокує прямий трафік між сегментами без явної експлуатації – додатковий захист на випадок неправильної конфігурації VM.

Аналіз потенційних загроз та контрзаходи. Перелік потенційних ризиків при використанні тренажера та механізми їх нейтралізації наведено у таблиці Г.8 (дод. Г).

Обмеження доступу вебсервера. Flask-сервер слухає виключно на localhost (127.0.0.1:5000), тому інтерфейс доступний лише з машини студента – інші користувачі мережі не можуть підключитися навіть знаючи його IP. Це унеможливлює віддалені атаки на тренажер.

Захист адмінінтерфейсу. Розділ /admin/* захищений окремим паролем TEACHER_PASSWORD (зі змінної середовища).. Маршрут /admin/* виключено зі студентської авторизації – обидва шари незалежні. Усі id в admin API валідуються регексом ^[a-z][a-z0-9_-]{2,63}$, що унеможливлює path traversal.

Обмеження команд VBoxManage. VMManager використовує whitelist дозволених команд, які формуються підстановкою параметрів у заздалегідь визначені шаблони; усі параметри валідуються регулярними виразами, що виключає command injection через, наприклад, ім’я VM у JSON-файлі.

Захист від некоректних JSON-сценаріїв. При завантаженні scenario_loader валідує файл за схемою: наявність обов’язкових полів, типи даних, допустимі значення перерахувань (level ∈ beginner/medium/advanced), формат validator_type. Файли, що не пройшли валідацію, не завантажуються, а помилка фіксується в логах. Та сама логіка доступна як CLI-інструмент validate_scenario.py (підрозділ 2.10) і через веб-інтерфейс (підрозділ 2.11).

Юридичні аспекти. Усі вразливі VM офіційно дозволені до навчального використання: Metasploitable 2 (Rapid7, освітня ліцензія) [15], DVWA (open source, GPL) [30], DC-1 (VulnHub, DCAU, дозвіл на навчання) [31]. Тренажер не містить шкідливого коду чи реальних експлойтів – усі вразливості існують лише у визначеному наборі ізольованих цільових VM.

Отже, тренажер забезпечує повну ізоляцію навчального середовища від хоста й зовнішніх мереж через комбінацію апаратної віртуалізації, ізоляції мережі та обмежень доступу вебсервера, що робить його використання безпечним.

2.10 CLI-інструменти для розробки сценаріїв

Окрім штатних механізмів динамічного завантаження даних (runtime-механізмів), програмний тренажер містить два додаткові консольні інструменти (CLI) — автоматизований валідатор сценаріїв та інтерактивний генератор контенту. Їх призначення полягає у спрощенні процесів розробки та підтримки конфігураційних файлів формату JSON, а також в автоматичному виявленні типових синтаксичних і логічних помилок на етапі створення завдань. Зазначені утиліти розташовані у системному каталозі scripts/ і можуть виконуватися як

автономно через інтерфейс командного рядка, так і викликатися автоматично безпосередньо з графічного вебінтерфейсу тренажера.

CLI-валідатор (validate_scenario.py) виконує статичний аналіз JSON-файлу й видає звіт про проблеми; підтримує перевірку одного файлу або пакетну обробку папки scenarios/. Перелік перевірок наведено у таблиці Г.9 (дод. Г).

Утиліта має три режими: стандартний (усі помилки й попередження, --all), strict (попередження → помилки, для CI/CD, --all --strict) і --errors-only (лише критичні). Коди виходу – 0 (коректно), 1 (помилки), 2 (невалідний JSON) – стандартизовані для інтеграції в скрипти.

Інтерактивний генератор сценаріїв (new_scenario.py) призначений для спрощення процесу створення нових завдань за допомогою послідовного діалогового майстра. Під час виконання скрипт у покроковому режимі запитує у користувача базові метадані: назву сценарію, рівень складності, перелік необхідних VM (доступні варіанти автоматично зчитуються з каталогу vms/), загальну кількість кроків та інші обов'язкові параметри. Результатом роботи утиліти є сформований за еталонною структурою файл формату JSON, який зберігається у директорії scenarios/. Він містить усі необхідні системні поля та тимчасові заповнювачі (TODO-плейсхолдери) для подальшого внесення контенту. Одразу після успішного завершення генерації файлу, скрипт автоматично ініціює виклик валідатора для первинної перевірки створеної структури.

Інтеграція з веб-редактором. Вся логіка валідатора реалізована у вигляді окремих модульних функцій, що дозволяє викликати їх як через консольний інтерфейс (CLI), так і всередині вебзастосунку. Основна функція validate_data(scenario_dict) приймає на вхід словник із даними сценарію та повертає об’єкт класу Report, який згодом серіалізується у формат JSON за допомогою методу to_dict().

Завдяки такій архітектурній побудові, консольні інструменти та графічний веб-редактор (детально описаний у підрозділі 2.11) виконують абсолютно ідентичні перевірки. Будь-яка інтерактивна форма у браузері не дозволить

зберегти сценарій, який відхиляється під час консольної перевірки. Це забезпечує суворе дотримання фундаментального принципу розробки програмного забезпечення DRY (Don't Repeat Yourself) на рівні бізнес-правил тренажера.

2.11 Веб-інтерфейс редактора сценаріїв

Хоча CLI-інструменти покривають технічні потреби, поріг входу для викладача без досвіду з JSON залишається відчутним. Для його зниження тренажер містить веб-інтерфейс редактора сценаріїв – створення, перегляд, редагування й валідація сценаріїв через форму у браузері. Нижче описано його архітектуру та екрани.

Авторизація викладача. Доступ захищено паролем TEACHER_PASSWORD (зі змінної середовища; за замовчуванням «teacher», для production рекомендовано змінити). Сторінка /admin/login приймає пароль; після успіху Flask-сесія отримує флаг teacher_authenticated, що перевіряється декоратором @teacher_required. Пароль порівнюється через hmac.compare_digest.

Дашборд адміністратора (/admin) показує всі сценарії з scenarios/ таблицею: статус валідації (OK / попередження / помилка), назва й id, складність, кількість кроків, потрібні VM, а також кнопки дій (редагування, дублювання, експорт у JSON, видалення). Угорі – статистика: загальна кількість сценаріїв, «чистих», з помилками й з попередженнями.

Макет дашборду адміністратора представлено на рисунку 2.10.

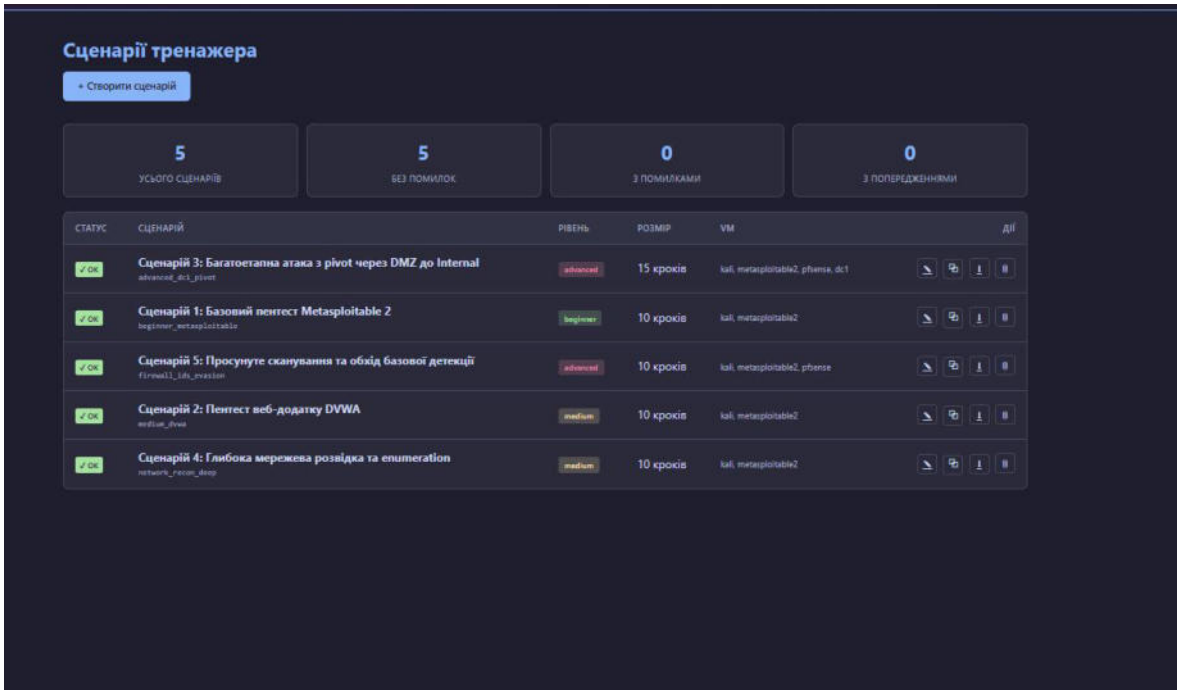


Рисунок 2.10 – Дашборд адміністратора з переліком сценаріїв

Сторінки /admin/scenarios/new і /admin/scenarios/<id>/edit використовують єдиний шаблон з шести секцій: метадані (id, назва, рівень, тривалість, потрібні VM, мережа); навчальні цілі; техніки MITRE ATT&CK (тег-стиль з автозаповненням); теоретичний вступ (HTML); кроки сценарію (повторюваний компонент з drag-handle сортуванням). Кожен крок – окрема карта з повним набором полів, що можуть згортатися.

Загальний вигляд форми редактора сценарію наведено на рисунку 2.11.

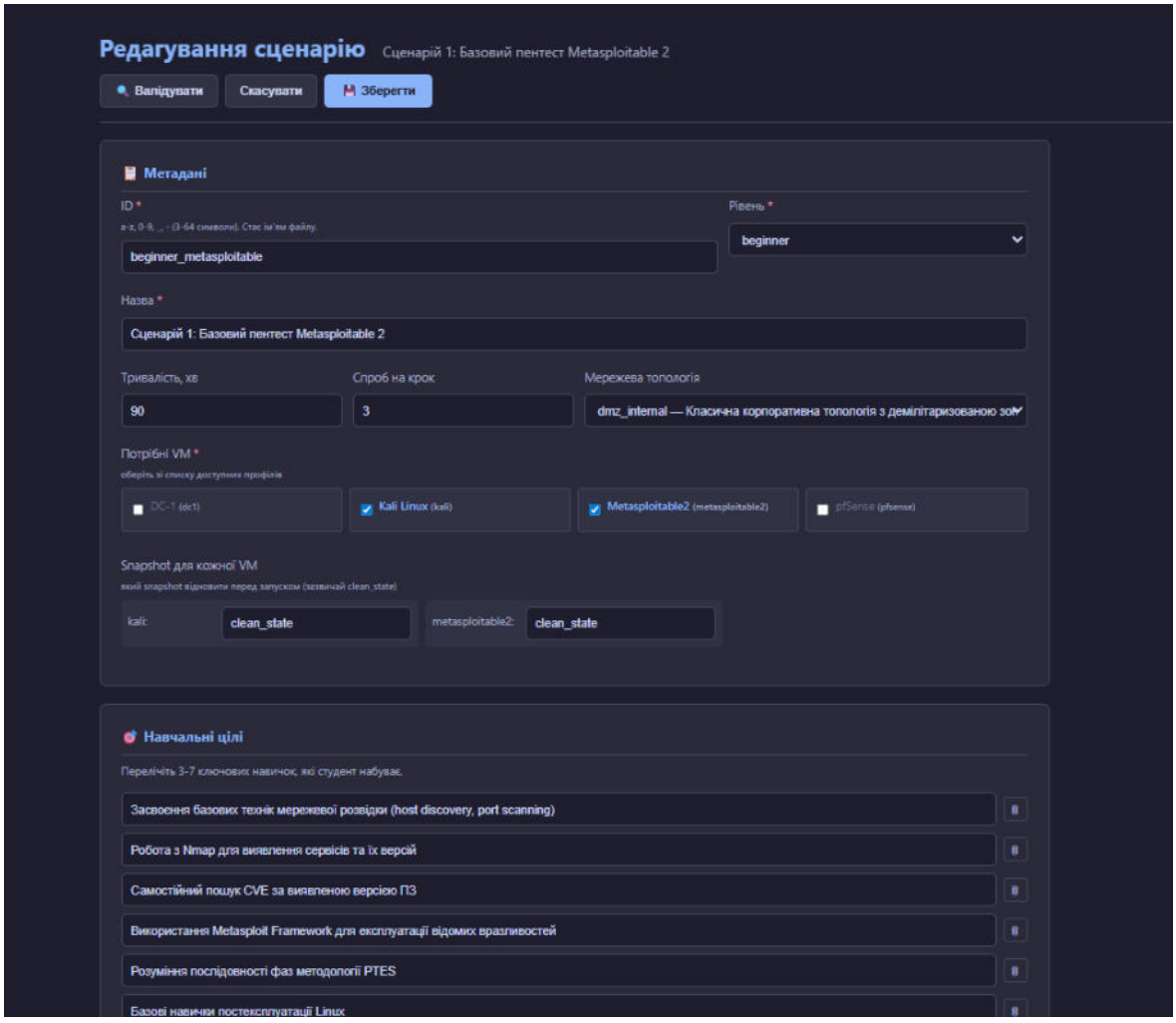


Рисунок 2.11 – Веб-форма редактора сценарію

Поле «Потрібні VM» – чекбокси, динамічно сформовані з vms/*.json; при виборі VM секція «Snapshot» автоматично додає рядок з дефолтом «clean_state». Поле «Мережева топологія» – список з networks/*.json. Поле «Тип валідатора» – список з п’яти типів; для multiple_choice з’являється поле «Варіанти відповіді». Це унеможливлює помилки на кшталт неіснуючого validator_type чи посилання на відсутню VM.

Автозаповнення технік MITRE ATT&CK. Поля ATT&CK-технік (рівня сценарію й кроку) реалізовано як автозаповнення з довідника trainer/static/data/attck_techniques.json (80 відібраних технік для мережевих пентест-сценаріїв). Кожен запис містить id (T1018), назву (Remote System Discovery) і тактику (Discovery); пошук – за id, назвою чи тактикою. Це знижує ризик помилки у форматі ідентифікаторів і час на пошук техніки.

Інтегрована валідація. Кнопка «Валідувати» викликає валідатор (підрозділ 2.10) через /api/admin/scenarios/validate; результат – у модальному вікні з переліком знахідок (severity, location, повідомлення). При збереженні валідація запускається автоматично: за наявності помилок збереження блокується, за наявності лише попереджень пропонується «Зберегти попри попередження» (?force=true).

Безпека операцій з файлами. Усі id в admin API валідуються регексом ^[a-z][a-z0-9_-]{2,63}$, що унеможливлює path traversal. Запис обмежено каталогом scenarios/ та іменем <id>.json; видалення вимагає підтвердження. Після кожного збереження/видалення викликається ScenarioLoader.load_all(), оновлюючи кеш – студент в іншій вкладці одразу бачить актуальний контент.

Типовий процес створення нового сценарію викладачем через веб-редактор виглядає наступним чином:

1) Викладач відкриває /admin/login та вводить пароль.

2) На дашборді /admin натискає «Створити сценарій» – відкривається порожня форма /admin/scenarios/new.

3) Заповнює базові метадані: id (наприклад, "smb_relay_attack"), назва, рівень, тривалість.

4) Відмічає чекбокси VM з пропонованого списку – секція snapshots заповнюється автоматично.

5) додає кроки кнопкою «Додати крок», заповнюючи заголовок, інструкцію, відповідь і бали; для нестандартних типів обирає validator_type зі списку.

6) Натискає «Валідувати» – у модальному вікні отримує перелік знахідок (якщо є). Виправляє проблеми безпосередньо у формі.

7) Натискає «Зберегти» – сценарій записується у scenarios/smb_relay_attack.json. Кеш ScenarioLoader оновлюється; сценарій одразу доступний студентам.

Весь процес від «Створити» до готового сценарію займає 15–45 хв залежно від кількості кроків. Контакту з командним рядком чи JSON-файлами не потрібно – викладач працює лише з графічним інтерфейсом.

Отже, веб-редактор реалізує найвищий рівень модульності: викладач без знання JSON може створювати повноцінні сценарії з усіма можливостями платформи. Інтегрована валідація гарантує якість перед збереженням, а спільна кодова база з CLI забезпечує консистентність перевірок.

Редактор підтримує повний цикл обміну сценаріями без ручного копіювання файлів. Експорт (GET /api/admin/scenarios/<id>/export) завантажує обраний сценарій як JSON-файл. Імпорт (POST /api/admin/scenarios/import, кнопка «Імпортувати» на дашборді) завантажує сценарій із JSON-файла назад у каталог scenarios/. Ключова особливість: перед збереженням імпортований файл проходить той самий валідатор (*validate_data*), що й при ручному збереженні через форму, — наявність помилок блокує імпорт повністю, а у разі попереджень або збігу id з уже наявним сценарієм інтерфейс запитує підтвердження (відповідно, імпорт попри попередження або перезапис). Передбачено базові перевірки розширення файла, його розміру та коректності структури JSON. Завдяки спільній з CLI логіці валідації (підрозділ 2.10) форма не може прийняти сценарій, який CLI відхиляє. Це дозволяє викладачеві готувати сценарії на одній машині й переносити їх на інші екземпляри тренажера, зберігаючи валідацію як обов’язковий контроль якості.

Керування профілями віртуальних машин. Окрім сценаріїв, веб-панель дозволяє керувати профілями VM на окремій сторінці (/admin/vms, пункт меню «Машини»), що усуває потребу писати JSON-профіль вручну. Викладач спочатку реєструє машину у VirtualBox (імпорт OVA, встановлення з ISO або готовий образ) і робить для неї snapshot clean_state, після чого на сторінці обирає цю машину зі списку зареєстрованих у VirtualBox (VBoxManage list vms) і задає поля, яких немає у гіпервізорі: категорію, мережу та IP, ім’я snapshot, облікові дані й короткий опис вразливостей. За цими даними тренажер генерує файл

vms/<id>.json. Образ при цьому не імпортується — профіль лише підключає вже зареєстровану машину за іменем (display_name).

Профілі підтримують експорт та імпорт. Кнопка експорту завантажує обраний профіль як JSON-файл, а імпорт (кнопка «Імпортувати JSON») завантажує готовий профіль назад у каталог vms/. Це дозволяє підготувати опис машини один раз і розповсюдити його на інші робочі станції без повторного заповнення форми — за умови, що відповідна машина зареєстрована в їхньому VirtualBox під тим самим іменем. Імпортований профіль перевіряється тими ж правилами, що й при ручному створенні (наявність обов'язкових полів, допустима категорія), а у разі збігу id з наявним профілем інтерфейс запитує підтвердження перезапису.

Сторінку керування профілями VM наведено на рис. 2.12.

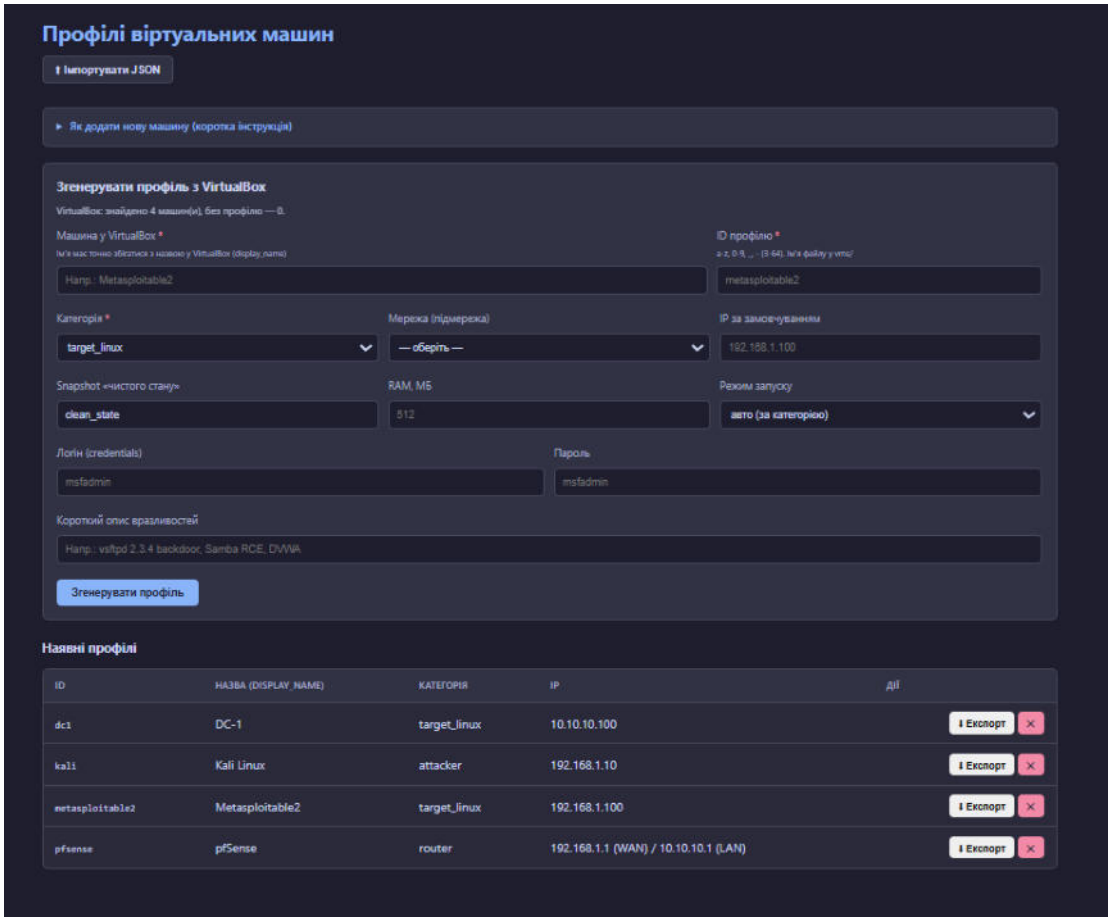


Рисунок 2.12 – Сторінка керування профілями VM (генерація з VirtualBox, експорт/імпорт)

2.12 Розгортання експериментального стенду

Цей підрозділ описує розгортання тренажера на робочій станції студента; покрокова інструкція гарантує відтворюваність стенду.

Хост має задовольняти вимоги Таблиці 2.2. Перед розгортанням встановлюють три компоненти: Oracle VirtualBox 7.0+ з Extension Pack; Python 3.10+ з pip з обов'язковим додаванням шляху до змінної PATH та розпаковувач архівів.

Етапи розгортання :

1) створення двох ізольованих мереж VirtualBox (dmz-net 192.168.1.0/24 та internal-net 10.10.10.0/24).
2) завантаження образів цільових VM з офіційних джерел (Kali з kali.org, Metasploitable 2 з SourceForge, DC-1 з VulnHub, pfSense ISO з pfsense.org).
3) Встановлення перейменуванням відповідно до JSON-профілів.
4) встановлення pfSense з ISO й налаштування двох інтерфейсів під зазначені вище підмережі
5) конфігурування статичних IP за Таблиця 2.4;
6) 6) створення зрізів «clean_state» на всіх VM.

Обов'язковим етапом верифікації є перевірка зв'язності віртуальних машин у межах створених підмереж. Для цього на маршрутизаторі pfSense необхідно виконати відправку ICMP-запитів (ping) на IP-адреси всіх хостів згідно з параметрами таблиці 2.4. Також здійснюється перевірка мережевої доступності з атакуючої машини Kali Linux до цільових систем Metasploitable2 та DC-1. Якщо всі ICMP-запити з pfSense пройшли успішно, а з боку Kali Linux фіксуються відповіді від Metasploitable2, але повністю відсутні від DC-1 (що обумовлено перебуванням машин у різних підмережах та блокуванням міжмережевого трафіку правилами фільтрації pfSense), конфігурацію мережевої інфраструктури вважають виконаною правильно.

Після підготовки середовища: розпакувати pentest_trainer.zip; створити venv (*python3 -m venv venv*) й активувати; встановити залежності (*pip install -r*

requirements.txt); опційно задати export TEACHER_PASSWORD='…'; запустити python run.py. Браузер автоматично відкриється на http://127.0.0.1:5000 зі сторінкою авторизації.

Опціональне розширення під MITM(man in the middle)-сценарії. Документ описує опціональний шостий VM-профіль для майбутніх сценаріїв канального/мережевого рівня (ARP-спуфінг, MITM, sniffing) – Ubuntu Desktop 22.04 LTS у ролі «жертви» зі статичним IP 192.168.1.50 в DMZ, що bash-скриптом кожні 30 с виконує cleartext FTP-вхід на Metasploitable для генерації перехоплюваного трафіку. Готовий JSON-профіль наведено в інструкції.

Документ містить розділ з типовими помилками: «VBoxManage not found» (додати шлях до PATH); «Internal Network не знайдена» (перевірити VBoxManage natnetwork list); проблеми з ping (підключення VM до однієї мережі); відсутність snapshot «clean_state» (точна назва).

Резервне копіювання. Перед оновленнями рекомендовано експортувати всі VM як OVA (VBoxManage export); розмір повного бекапу 15–20 ГБ. Зберігання архівів на зовнішньому носії дозволяє швидко відновити стенд на іншій робочій станції без повторного проходження всіх етапів.

Отже, процедуру розгортання детально задокументовано (DEPLOYMENT.md, ~500 рядків з командами VBoxManage, джерелами образів і таблицями налаштувань), що забезпечує відтворюваність стенду й дозволяє іншому досліднику самостійно повторити експериментальну частину.

2.13 Висновки до розділу

У другому розділі обґрунтовано концепцію, спроєктовано архітектуру, описано основні модулі та реалізовано програмний тренажер для навчання мережевому penetration testing. За результатами виконаної роботи:

1. Обґрунтовано концепцію тренажера як локального вебзастосунку, що автоматизує розгортання середовища, надає покрокові інструкції, перевіряє виконання й генерує звіти, поєднуючи зручність вебінтерфейсу з автономністю.

2. Обрано технологічний стек (Python 3.10+, Flask 3.x, SQLite, VBoxManage, reportlab) і спроєктовано трирівневу архітектуру з розділенням обов'язків.

3. Описано віртуалізоване середовище з двома сегментами (DMZ 192.168.1.0/24, Internal 10.10.10.0/24) через pfSense: Kali Linux (DMZ), pfSense(DMZ та Internal), Metasploitable 2(DMZ), DC-1(Internal).

4. Реалізовано модуль управління VM з трьома режимами запуску (gui/headless/separate), керуванням зрізами і асинхронним виконанням тривалих операцій.

5. Розроблено модуль сценаріїв з п'ятьма типами валідаторів (text_flag, regex, numeric_range, case_insensitive_substring, multiple_choice), що вирішує проблему варіативності відповідей між snapshot-станами VM.

6. Реалізовано п'ять сценаріїв із 55 кроків: базовий Metasploitable (10), DVWA (10), advanced pivot DMZ→Internal (15), мережева розвідка (10), сканування з обходом детекції (10); вони охоплюють 36 технік MITRE ATT&CK у 9 з 12 тактик.

7. Спроєктовано модуль оцінювання з трьома факторами (бали кроку, штраф за підказку 30%, штраф за додаткові спроби 10%) і автогенерацією PDF-звітів через reportlab.

8. Розроблено модульну архітектуру (VM-профілі, мережеві профілі, сценарії): ≈90% задач розширення виконуються через веб-редактор.

9. Реалізовано два CLI-інструменти: *validate_scenario.py* і *new_scenario.py* , що знижують поріг входу й автоматизують контроль якості.

10. Реалізовано веб-редактор сценаріїв (/admin) з паролем TEACHER_PASSWORD: дашборд зі статусом валідації, форма з автозаповненням ATT&CK (80 технік), drag-handle сортуванням кроків та інтегрованою валідацією.

11. Спроєктовано графічний користувацький інтерфейс тренажера, який включає шість спеціалізованих екранних форм для взаємодії зі студентами та окрему панель адміністратора для керування контентом. На рівні програмної

логіки розроблено та реалізовано архітектуру REST API, що налічує 14 ключових кінцевих точок (endpoints) для забезпечення стабільного обміну даними між клієнтською частиною застосунку, сервером Flask та підсистемою керування гіпервізором.

12. Проаналізовано безпеку: три рівні ізоляції (гіпервізор + Internal Network + pfSense), окрема авторизація викладача через hmac.compare_digest, захист від path traversal через regex-валідацію id, whitelist команд VBoxManage; підтверджено легітимність вразливих VM.

13. Підготовлено вичерпну технічну документацію у вигляді інструкції з розгортання DEPLOYMENT.md обсягом близько 500 рядків коду та текстового опису. Вона інтегрована до загального програмного пакету тренажера та містить детальні покрокові вказівки з конфігурації залежностей, що повністю забезпечує автономну відтворюваність експериментальної частини дослідження та полегшує подальше впровадження платформи у навчальний процес.У сукупності робота забезпечує функціонально завершений тренажер з модульною архітектурою, що готова до інтеграції у навчальний процес кафедри.

РОЗДІЛ 3. АПРОБАЦІЯ РОЗРОБЛЕНИХ СЦЕНАРІЇВ ТА ОЦІНЮВАННЯ МОЖЛИВОСТЕЙ ТРЕНАЖЕРА

3.1 Методика апробації

Третій розділ присвячено апробації розроблених сценаріїв та оцінюванню можливостей тренажера як середовища для їх створення й виконання: обґрунтовано методику апробації, наведено заміри продуктивності середовища, результати самотестування сценаріїв, перевірку можливості додавання нових сценаріїв і порівняльний аналіз з аналогами.

Апробацію виконано за п’ятьма напрямами, що сукупно дають всебічну оцінку готовності тренажера. Самостійне проходження всіх сценаріїв розробником забезпечує контрольований і відтворюваний базовий рівень вимірювань – із фіксованим часом і відомими правильними відповідями, що є коректною відправною точкою для подальшого випробування з цільовою аудиторією.

Перелік застосованих методів апробації та їх призначення наведено у таблиці 3.1.

Таблиця 3.1 – Застосовані методи апробації тренажера

Метод	Об’єкт оцінювання	Результат
Інструментальне вимірювання	Час запуску VM, споживання RAM, час відновлення snapshots, час старту сценаріїв	Отримання кількісних характеристик продуктивності та ресурсомісткості системи
Самотестування сценаріїв	Коректність роботи кожного з 5 реалізованих сценаріїв, тривалість виконання	Підтвердження технічної працездатності інфраструктури та її відповідності плановим часовим лімітам
Створення тестового сценарію	Оцінювання зручності модульної архітектури: порівняння часу розробки нового завдання через CLI, веб-редактор та шляхом ручного редагування JSON	Кількісне підтвердження ефективності зниження порогу входження для викладацького складу
Аналіз покриття	Відповідність розроблених сценаріїв фазам методології PTES, технікам фреймворку MITRE ATT&CK, категоріям OWASP Top 10 та стандарту NIST SP 800-115	Отримання якісної та кількісної оцінки дидактичної повноти навчального контенту
Порівняльний аналіз	Функціональні можливості та фінансова доступність розробленого тренажера порівняно з комерційними платформами (HackTheBox, TryHackMe)	Обґрунтоване позиціонування створеного рішення у ландшафті сучасних кіберполігонів

Таким чином, поточне дослідження фокусується на розробці методичного підґрунтя, реалізації функціонально завершеної архітектури програмного тренажера та експериментальному підтвердженні його технічної готовності. Отримані результати, що поєднують автоматизацію керування віртуальним середовищем із локалізованим супроводом завдань, повністю вирішують

поставлену задачу та є репрезентативними й достатніми для кваліфікаційного рівня бакалавра за спеціальністю 125 «Кібербезпека».

Отже, обрана методика забезпечує всебічну оцінку програмного тренажера за комплексом технічних, дидактичних та порівняльних критеріїв, що повністю відповідає вимогам до кваліфікаційних робіт бакалаврського рівня. У наступних підрозділах послідовно представлено та деталізовано результати практичних досліджень за кожним із визначених напрямів верифікації системи.

3.2 Технічні характеристики стенду як середовища виконання сценаріїв

Придатність стенду як середовища виконання сценаріїв визначається насамперед його ресурсними характеристиками. Інструментальне вимірювання продуктивності проводилось на робочій станції, що відтворює реальні умови використання. Конфігурацію тестової машини наведено у таблиці 3.2.

Таблиця 3.2 – Конфігурація апаратного та програмного забезпечення тестової машини

Компонент	Характеристика
Процесор	Intel Core i5-12450H (8 ядер 12 потоків) 2.0ГГц, 4.4 ГГц Turbo Boost
Оперативна пам’ять	16 ГБ DDR4 4800 МГц
Накопичувач	NVMe SSD 512 ГБ (SK Hynix BC901 , read 5000 МБ/с, write 4700 МБ/с)
Графіка	NVIDIA RTX 4060 Laptop (8gb)
Операційна система хоста	Windows 11 Pro 23H2 (build 22631)
VirtualBox	7.2.4r170995
Python	3.14.5
Браузер для тестування	Google Chrome 148.0.7778.218

Методика вимірювання. Час запуску VM (від VBoxManage startvm до стану «running») вимірювався Python-модулем time.perf_counter(); споживання RAM – через VBoxManage showvminfo --machinereadable після 60 с стабілізації; час старту сценарію – вбудованою аналітикою тренажера. Кожен замір виконувався тричі (середнє арифметичне); стандартне відхилення не перевищувало 10%.

Результати замірів часу запуску й відновлення snapshot наведено у таблиці 3.3. «Холодний старт» – запуск зі стану «poweroff», «snapshot restore» – відновлення з clean_state з подальшим завантаженням ОС.

Таблиця 3.3 – Час запуску та відновлення віртуальних машин (середнє з 3 замірів, с)

VM	Холодний старт	Snapshot restore	Стабілізація сервісів	Сумарно до готовності
Kali Linux 2024.1	52	9	12	21
Metasploitable 2	28	6	4	10
pfSense CE 2.7	22	5	3	8
DC-1 (VulnHub)	34	7	6	13

З таблиці випливає кілька спостережень: відновлення snapshot прискорює запуск у 3–6 разів проти холодного старту, що підтверджує доцільність механізму; найповільніша операція – підготовка Kali Linux (21 с) – прийнятна, бо за цей час встигає завантажитись вебінтерфейс.

Заміри споживання RAM кожною VM у стані «idle» наведено у таблиці 3.4; при активному використанні (Metasploit-консоль, паралельні nmap-сканування) споживання може зростати на 10–20%.

Таблиця 3.4 – Споживання оперативної пам’яті віртуальними машинами (idle-стан, МБ)

VM	Виділено (allocated)	Реально використано (used)
Kali Linux 2024.1	2048	830
Metasploitable 2	512	244
pfSense CE 2.7	1024	10
DC-1 (VulnHub)	1024	145
Сумарно (всі 4)	4608	1229

Сумарне реальне споживання RAM усіма чотирма VM – близько 1.2 ГБ (менше за виділені 4,6 ГБ, бо ОС завантажують лише потрібні модулі). На хості з 16 ГБ лишається ~14 ГБ для ОС і тренажера; навіть на мінімально дозволених 8 ГБ (підрозділ 2.1) лишається ~6 ГБ.

Найважливіший показник UX – час від натискання «Розпочати сценарій» до повної готовності середовища (відновлення snapshot і запуск усіх VM сценарію). Заміри для кожного з п’яти сценаріїв наведено у таблиці 3.5.

Таблиця 3.5 – Час старту сценаріїв (від кліку «Розпочати» до готовності, с)

Сценарій	Кількість VM	Час старту(с)
1. Базовий пентест Metasploitable	2	43
2. DVWA веб-пентест	2	43
3. Advanced pivot DMZ→Internal	4	87
4. Глибока мережева розвідка	2	43
5. Просунуте сканування з обходом	3	49

Сценарії 1, 2, 4 (дві VM: Kali + Metasploitable 2) готуються близько 43с – як старт важкого настільного додатка, без дискомфорту. Сценарій 5 з трьома VM (додано pfSense) – 49 с, найважчий сценарій 3 з чотирма VM – 87с. Для тривалих стартів тренажер показує прогрес-бар з поточною операцією (наприклад, «Відновлення snapshot для DC-1»).

Графік порівняння часу старту сценаріїв з різною кількістю задіяних VM представлено на рисунку 3.1.

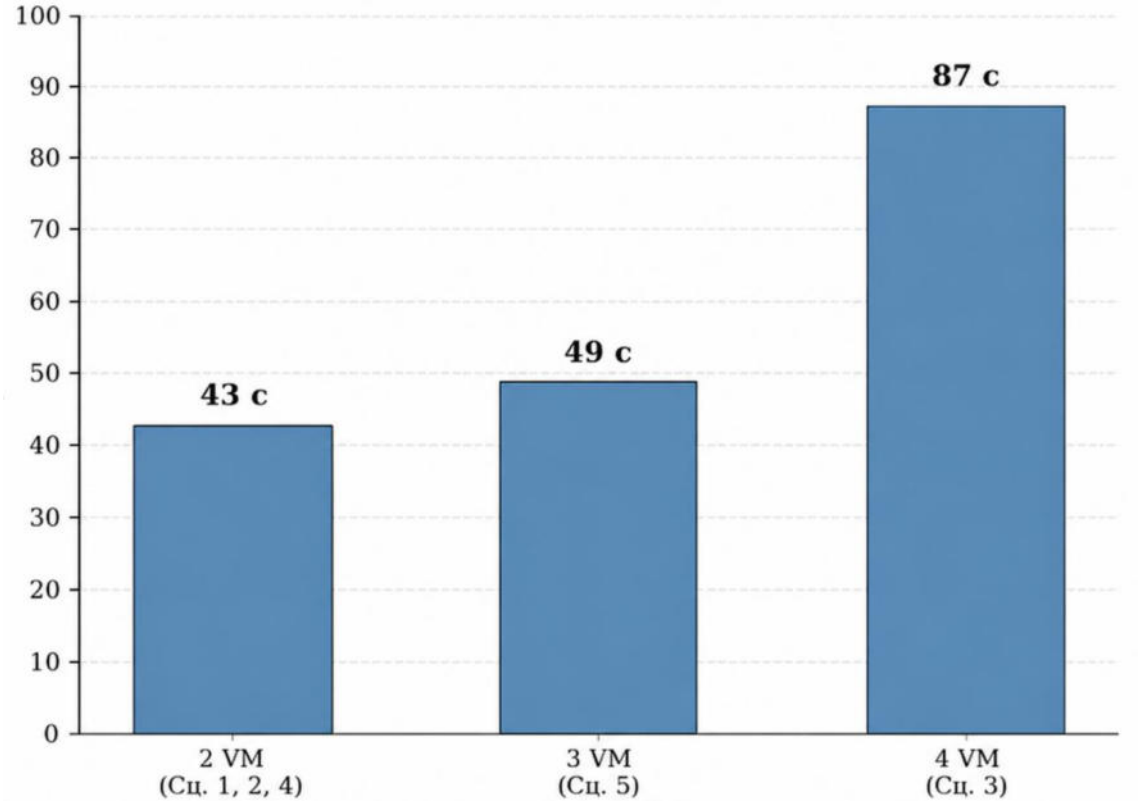


Рисунок 3.1 – Залежність часу старту сценарію від кількості задіяних віртуальних машин

Окрема характеристика – вимоги до дискового простору. Заміри реально займаного місця кожною VM у стані clean_state наведено у таблиці 3.6.

Таблиця 3.6 – Дисковий простір, виділений віртуальним машинам

VM	Розмір диска (виділено)	Реальний розмір на хості
Kali Linux 2024.1	80ГБ	15.2 ГБ
Metasploitable 2	8 ГБ	1.8 ГБ
pfSense CE 2.7	5 ГБ	1.5 ГБ
DC-1 (VulnHub)	4 ГБ	2 ГБ
Сумарно	97 ГБ	20.5 ГБ

Реальний розмір дисків значно менший за виділений завдяки динамічно розширюваним дискам VirtualBox. Сумарний дисковий слід стенду – близько 15ГБ, що відповідає мінімальній рекомендації у 30ГБ вільного місця (підрозділ 2.1) з резервом на тимчасові файли, логи й бекапи.

Підсумовуючи, стенд демонструє прийнятну для типового студентського ноутбука продуктивність: час старту сценаріїв (43–87 с), споживання RAM (1.2 ГБ) і диск (30 ГБ) відповідають проєктним характеристикам Розділу 2, що підтверджує технічну готовність тренажера до інтеграції.

3.3 Результати самотестування сценаріїв

Самотестування проводилось з фіксацією часу кожного кроку, факту використання підказок і коректності відповідей з першої спроби, що дозволяє оцінити реалістичність планової тривалості, складність кроків і якість підказок. Методику описано у підрозділі 3.1.

Сценарій 1: базовий пентест Metasploitable 2. 10 кроків, планова тривалість 30 хв, методологія PTES (Reconnaissance → Vulnerability Analysis → Exploitation → Post-Exploitation) на прикладі експлуатації vsftpd 2.3.4 backdoor (CVE-2011-2523). Фактичний час – 10 хв. Найдовші кроки: №4 «Дослідження CVE» (13 хв) і №7 «Налаштування експлойта» (10 хв, очікування Metasploit-сесії). Підказки не використовувались; контрольні питання прийнято з першої спроби.

Сценарій 2: пентест вебдодатку DVWA [30]. 10 кроків, планова тривалість 50 хв, охоплює веб-розвідку та експлуатацію SQL Injection, Command Injection, XSS і Brute Force через Hydra. Фактичний час – 30 хв. Найдовші кроки: №8 «Автоматизація через SQLmap» і №10 Brute Force з rockyou.txt проти gordonb.

Сценарій 3: багатоетапна атака з pivot DMZ→Internal – найскладніший (15 кроків, планова тривалість 90 хв, фактично 60хв). Складність – у комбінації дій: експлуатація Samba (CVE-2007-2447), upgrade до Meterpreter, autoroute, SOCKS-проксі, Drupalgeddon2 проти DC-1 через pivot, витяг хешів адміна Drupal, privesc через SUID find. Найдовші кроки: №10 Drupalgeddon2 (24 хв, з них 10 хв завантаження payload через pivot) і №14 privesc (17 хв, синтаксис find -exec).

Початківці можуть потребувати додаткових пояснень на кроках pivot і SOCKS – зафіксовано як побажання.

Сценарій 4: глибока мережева розвідка. 10 кроків, планова тривалість 30хв, фактично 20хв; фокус на широті розвідки – від типів Nmap-сканування до enum4linux, showmount (NFS), nikto, dirb. Тривалість залежить переважно від часу роботи утиліт.

Сценарій 5: просунуте сканування з обходом базової детекції. 10 кроків, планова тривалість 90хв, фактично 50хв – найбільш мережево-орієнтований: паралельна робота у двох терміналах (tcpdump + nmap), аналіз pcap-захоплень [32], idle/zombie-scan, SSH dynamic forwarding. Найдовший крок – №4 Slow scan -T1 (18 хв, особливість timing template T1). Крок №9 SSH ProxyJump потребував sshpass і proxychains. Валідатор numeric_range на кроці 1 («скільки пакетів у baseline-захопленні, очікувано 10–50») коректно прийняв фактичне значення 24 з tcpdump.

Сумарні результати самотестування всіх п’яти сценаріїв наведено у таблиці 3.7. Фактична тривалість виявилася на 33–67% меншою за планову (сумарно −44,8%), що свідчить про свідомо консервативне планування з часовим запасом (підрозділ 2.5).

Таблиця 3.7 – Сумарні результати самотестування сценаріїв

Сценарій	Кроків	Планова трив., хв	Фактична трив., хв
1. Metasploitable basic	10	30	10
2. DVWA web	10	50	20
3. Advanced pivot	15	90	60
4. Network recon	10	30	20
5. Evasion scanning	10	90	50

Під час самотестування підказки не використовувалися, що відображає верхню межу очікуваної ефективності, оскільки сценарії проходив їх розробник. Реальні студенти ймовірно звертатимуться до підказок, що згідно з системою

оцінювання (підрозділ 2.6) зменшує бали за крок на 30%; оцінка впливу підказок потребує окремого дослідження з цільовою аудиторією.

Графік фактичної та планової тривалості сценаріїв з відображенням похибки представлено на рисунку 3.2.

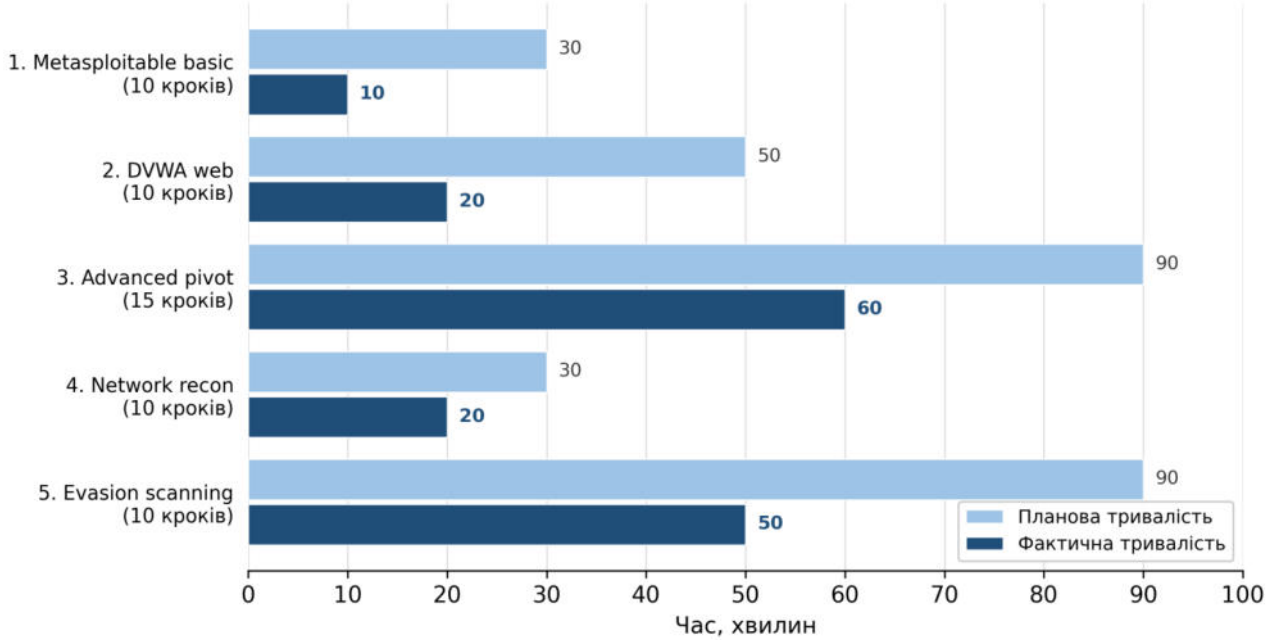


Рисунок 3.2 – Порівняння планової та фактичної тривалості виконання сценаріїв

Якість підказок і формулювань. У підрозділі 2.10 описано алгоритм виявлення витоків відповідей у підказках; його застосовано при доопрацюванні сценаріїв. У вихідних версіях деякі підказки містили дослівні відповіді (наприклад, «Стандартний пароль DVWA для admin – це слово password» у кроці 10 сценарію 2, де відповідь і була «password»). Після виправлення підказки змістилися на методологічний рівень («коротка комбінація з 6 символів, часто згадується у статтях про найпопулярніші слабкі паролі»), зберігши корисність без розкриття відповіді. Усі 55 кроків пройшли валідатор без жодного виявленого витоку.

Підсумовуючи, всі п’ять сценаріїв коректно функціонують, фактичні тривалості узгоджені з плановими підказки не розкривають відповідей, а нові

типи валідаторів (numeric_range, multiple_choice, case_insensitive_substring, regex) працюють як очікувалось.

3.4 Перевірка розширюваності: додавання нових сценаріїв

Одне з ключових рішень – модульність, що дозволяє додавати сценарії, VM-профілі й топології без зміни коду (підрозділ 2.7). Для перевірки заявлених у матриці розширюваності (таблиці Г.6 (дод. Г)) часових показників проведено експеримент: створення одного тестового сценарію трьома способами з фіксацією часу й зручності.

Опис експерименту. Тестовий сценарій – простий 5-кроковий beginner «SMB Anonymous Enumeration» (smbclient, enum4linux); навмисно простий, щоб різниця у часі відображала зручність інструменту, а не складність контенту. Створення проводилось трьома способами: ручне редагування JSON у VS Code, генерація через new_scenario.py із заповненням TODO і створення через веб-форму редактора (/admin). Кінцевий контент у всіх випадках ідентичний.

Результати замірів часу створення сценарію трьома способами наведено у таблиці 3.8.

Таблиця 3.8 – Час створення тестового сценарію різними способами (хв)

Спосіб	Налаштування	Структура	Контент кроків	Перевірка	Сумарно
Ручне редагування JSON	5 (вивчення схеми)	14	25	8 (debug помилок)	52
CLI new_scenario.py	2 (старт скрипта)	0 (генерується)	15(заповнення TODO)	3 (валідатор)	20
Веб-редактор	1 (вхід в адмін-панель)	0 (форма)	5я(через UI)	2 (validate-кнопка)	7

Найбільшу економію часу дає веб-редактор – 8 хв проти 52 при ручному JSON (−84,6%) за рахунок трьох факторів: відсутність потреби пам’ятати JSON-

схему (поля підставляються формою); динамічне підтягування опцій (VM, топології, ATT&CK-техніки з автозаповненням); інтегрована валідація (помилки видно до збереження). CLI-генератор займає проміжне положення – прибирає потребу пам’ятати схему, але контент заповнюється у звичайному редакторі.

Розподіл часу за етапами створення сценарію трьома методами візуалізовано на рисунку 3.3.

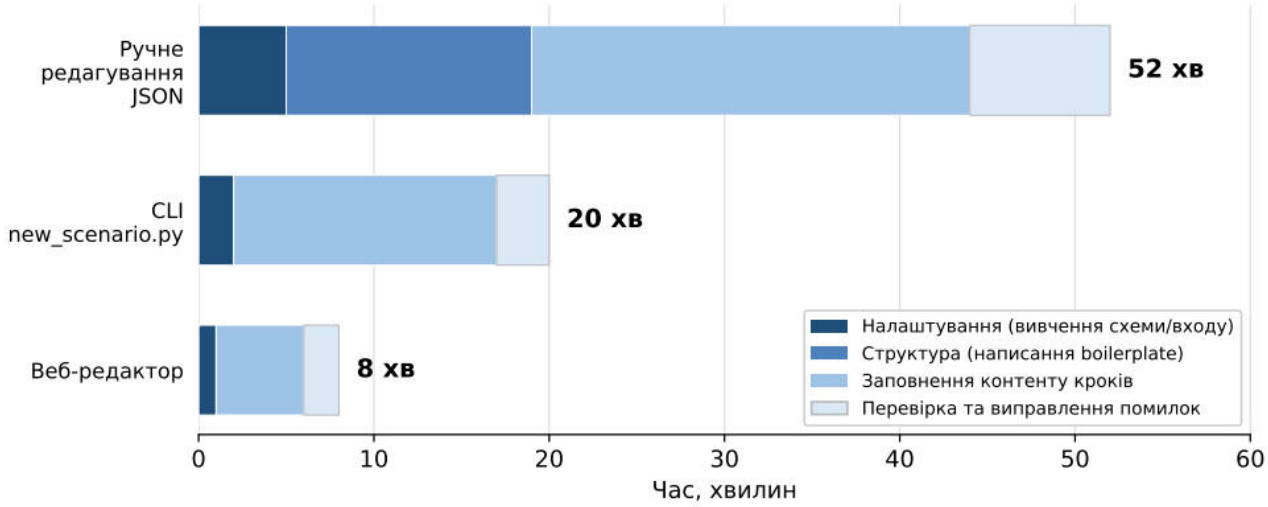


Рисунок 3.3 – Розподіл часу за етапами при створенні нового сценарію різними методами

Перевірка роботи валідатора. Підготовлено навмисно зламаний сценарій із 14 типовими помилками різних класів (невалідні поля, відсутні поля, неіснуюча VM, порожні та некоректні flag_answer, витоки відповідей у підказках, неправильні regex/numeric_range тощо) і протестовано через validate_scenario.py. Результати наведено у таблиці Г.10 (дод. Г).

Валідатор виявив усі 14 внесених помилок без хибно-позитивних і хибно-негативних результатів, що підтверджує ефективність алгоритму (підрозділ 2.10). Зокрема, для підказки «Пароль адміністратора – secret123» (де flag_answer = «secret123») валідатор коректно виявив дослівний збіг і згенерував повідомлення про необхідність переписати підказку.

Підсумовуючи, обидва шляхи створення сценарію (CLI і веб-редактор) істотно знижують час і поріг входу проти ручного JSON: CLI – на 61,5%, веб-редактор – на 84,6%. Валідатор виявляє 100% типових помилок (експеримент із 14 навмисно внесеними дефектами — підрозділ 3). Це підтверджує тезу підрозділу 2.7 про доступність ≈90% задач розширення без програмування.

3.5 Порівняльний аналіз з існуючими аналогами

Для позиціонування тренажера проведено порівняння з найвідомішими платформами навчання пентесту: HackTheBox (HTB), TryHackMe (THM), RangeForce та HTB Academy. Кожна позиціонується як рішення для навчання пентесту, але має істотні відмінності у моделі доступу, цільовій аудиторії й технічних особливостях.

Критерії порівняння. Обрано вісім критеріїв, що відображають технічні характеристики й дидактичну та організаційну придатність до інтеграції у навчальний процес ЗВО; вони випливають з вимог Розділу 1 (підрозділ 1.5): доступність (мова, ціна, ліцензія), якість методичного супроводу, технічна автономність, здатність до модифікації.

Зведене порівняння наведено у таблиці Г.11 (дод. Г) з позначеннями: + (повна підтримка), ± (часткова), – (відсутня) та короткими коментарями.

Порівняння виявляє переваги тренажера: повна автономність – на відміну від аналогів, що вимагають інтернету й VPN до cloud-інфраструктури, тренажер працює офлайн на локальній машині, що важливо для ЗВО з обмеженою мережею; безкоштовність – комерційні платформи коштують $10–14 на студента щомісяця ($3000–4200 на рік для групи з 25 осіб); україномовний інтерфейс і методичні матеріали – критичний фактор для студентів молодших курсів.

Слабкі сторони тренажера. Комерційні платформи переважають за обсягом контенту (HackTheBox – понад 300 машин, TryHackMe – понад 700 «rooms» проти 5 сценаріїв), актуальністю, спільнотою (форуми, Discord, CTF) і профілем користувача з рейтингами. Тренажер не претендує на конкуренцію за

обсягом контенту, а позиціонується як цільовий інструмент для ОП спеціальності 125, інтегрований у навчальний процес кафедри.

Унікальна ніша. Сукупність характеристик – україномовність, безкоштовність, автономність, модульність і можливість викладача самостійно створювати сценарії – формує нішу, не зайняту жодним аналогом. Найближчий за моделлю – RangeForce з академічними ліцензіями й кастомізацією, проте її вартість (від $50 тис. на рік для університету) робить її недоступною для більшості українських ЗВО. Тренажер заповнює саме цю прогалину.

Порівняльний аналіз підтверджує доцільність власної розробки: тренажер не конкурує за обсягом контенту, але істотно переважає аналоги в доступності, локалізації, модифіковуваності та інтеграції в освітній процес – найважливіших для ЗВО характеристиках.

3.6 Покриття стандартів і фреймворків реалізованими сценаріями

Дидактична цінність розроблених сценаріїв значною мірою визначається їх узгодженням із визнаними методологіями. У підрозділі проаналізовано покриття чотирьох стандартів: PTES, MITRE ATT&CK, OWASP Top 10 і NIST SP 800-115.

Покриття PTES. PTES визначає сім фаз [2]: Pre-engagement, Intelligence Gathering, Threat Modeling, Vulnerability Analysis, Exploitation, Post-Exploitation, Reporting. Кожен сценарій охоплює різну їх підмножину залежно від складності; покриття наведено у таблиці Г.12 (дод. Г).

Сценарій 3 (advanced pivot) покриває всі сім фаз PTES. Сценарії 1 і 2 – по 5–6 фаз з фокусом на експлуатації. Сценарії 4 і 5 як суто розвідувально-аналітичні не включають експлуатацію та постексплуатацію – свідома методична відмова заради фокусу на мережевих техніках. Reporting присутній у всіх сценаріях у вигляді автогенерованого PDF-звіту.

Покриття MITRE ATT&CK. Фреймворк [9, 10] складається з 14 тактик (версія v15) і понад 600 технік. Реалізовані сценарії покривають техніки у 9 з 14

тактик; розподіл унікальних технік за тактиками наведено у таблиці Г.13 (дод. Г).

Усього у сценаріях явно посилаються 36 унікальних технік MITRE ATT&CK з 9 тактик. Неохоплені тактики (Resource Development, Persistence, Collection, Exfiltration, Impact, ICS-тактики) – це або підготовчі фази, або post-engagement дії, або специфічні для промислових систем. Для мережевого пентесту бакалаврського рівня покрита підмножина достатня.

Тепло-карта покриття тактик MITRE ATT&CK реалізованими сценаріями представлено на рисунку 3.4.

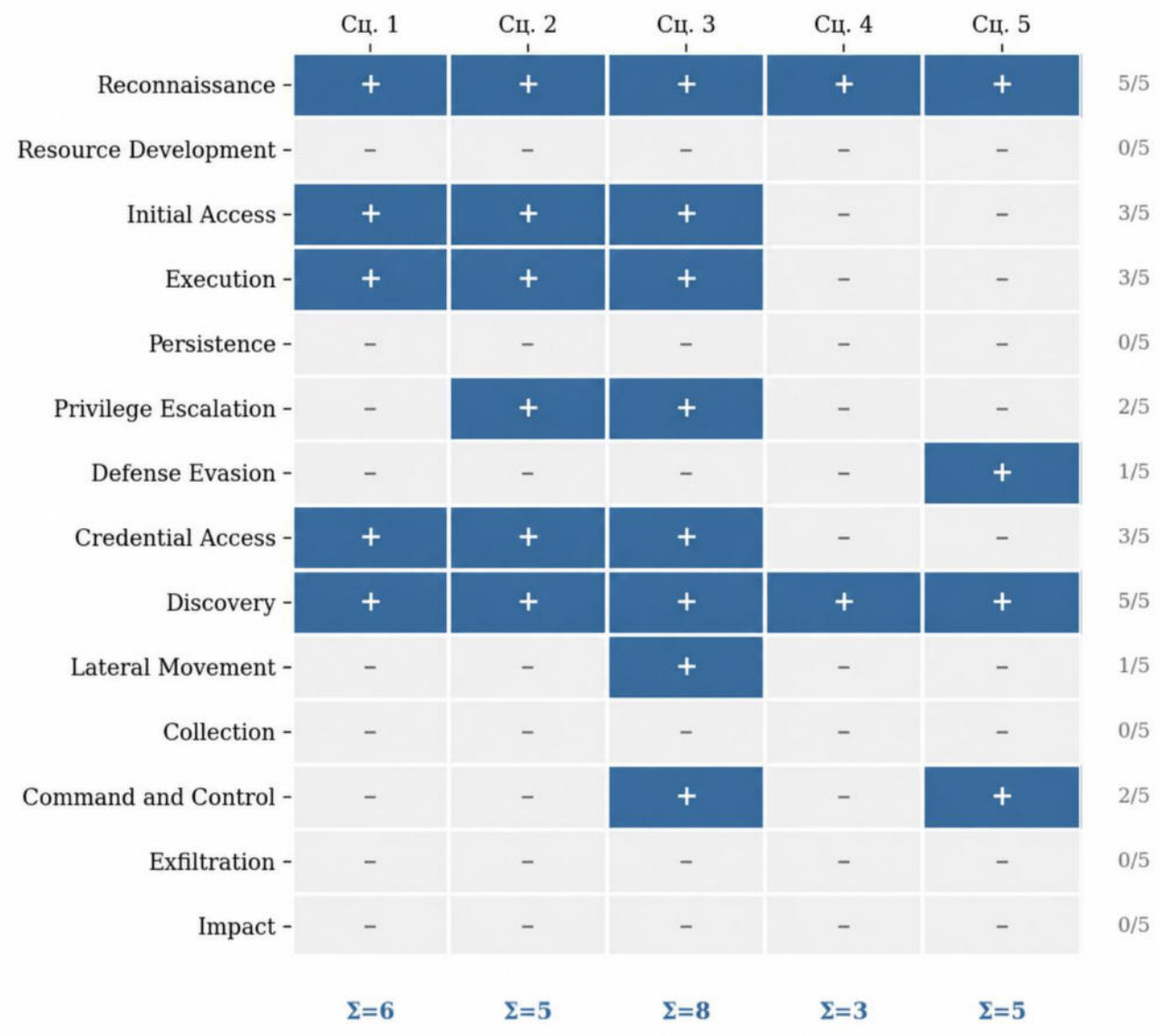


Рисунок 3.4 – Тепло-карта покриття тактик MITRE ATT&CK сценаріями

Покриття OWASP Top 10. OWASP Top 10 [6] – перелік найкритичніших ризиків вебдодатків, актуальний для сценарію 2 (DVWA). Покриття наведено у таблиці Г.14 (дод. Г).

Сценарій 2 покриває 7 з 10 категорій OWASP Top 10, причому A03 «Injection» – найповніше (5 кроків з 10), що відповідає її статусу найкритичнішої категорії за статистикою OWASP останніх 10 років. Неохоплені категорії (A01, A08, A09, A10) потребують спеціально підготовленого вразливого додатка – напрям майбутніх розширень.

Відповідність NIST SP 800-115. Це офіційне керівництво США [1] виділяє чотири фази: Planning, Discovery, Attack, Reporting, що зіставляються з кроками PTES, але формалізованіше щодо документації й повторюваності. Покриття NIST-фаз наведено у таблиці Г.15 (дод. Г).

Реалізовані сценарії покривають 7 з 8 підфаз NIST SP 800-115. Не реалізовану підфазу «Install Additional Tools» (persistent access) свідомо винесено за межі роботи як post-engagement дію red-team операцій, що виходить за базовий пентест бакалаврського рівня. Узгодження з NIST 800-115 підтверджує методологічну валідність тренажера.

Аналіз покриття підтверджує, що навчальний матеріал побудовано з прямим узгодженням з визнаними методологіями: PTES (до 7 фаз у найповнішому сценарії), MITRE ATT&CK (36 технік у 9 тактиках), OWASP Top 10 (7 з 10 категорій), NIST SP 800-115 (7 з 8 підфаз). Це робить набуті студентом навички перенесеними у реальну професійну діяльність.

3.7 Обмеження та напрями подальшого розвитку

Жоден освітній продукт не покриває всіх аспектів предметної галузі. У підрозділі чесно описано обмеження поточної версії й перспективні напрями розвитку – як орієнтир для майбутніх ітерацій та інших курсових і дослідницьких робіт кафедри.

Обмеження за обсягом контенту. П’ять сценаріїв охоплюють базові-середні-просунуті техніки мережевого пентесту, але не вичерпують предмет. Поза рамками лишились: атаки L2-рівня (ARP-спуфінг, MITM, sniffing) – заплановані як «Сценарій 6», профіль VM-жертви вже підготовлено; атаки на Active Directory (Kerberoasting, Pass-the-Hash, BloodHound) – потребують Windows Server і клієнта (+16 ГБ RAM, +30 ГБ диск); wireless-атаки – обмежені, бо VirtualBox не підтримує passthrough Wi-Fi; атаки на промислові протоколи (Modbus, DNP3, S7) – потребують спеціалізованих емуляторів (ConPot).

Обмеження апробації. Самотестування дає об’єктивну оцінку працездатності й узгодженості тривалостей, але не оцінює суб’єктивну дидактичну ефективність – сприйняття інструкцій студентами вперше; це об’єкт подальшої апробації.

Технічні обмеження архітектури. По-перше, single-user – тренажер працює на машині студента, що не дозволяє викладачу централізовано стежити за прогресом усіх. По-друге, фіксована залежність від VirtualBox – для інших гіпервізорів (VMware, KVM, Hyper-V) потрібні окремі VMDriver-плагіни . По-третє, відсутність вбудованого терміналу – студент тримає окремий термінал Kali; інтегрований термінал через WebSocket був би зручнішим.

Напрями подальшого розвитку, що випливають з виявлених обмежень, систематизовано у таблиці Г.16 (дод. Г) з оцінкою орієнтовного обсягу робіт.

Дослідницький потенціал. Тренажер створює основу для подальших досліджень – можливі теми магістерських робіт чи публікацій: аналіз ефективності навчання через гібридні сценарії; автоматизоване оцінювання якості сценаріїв; адаптивна траєкторія складності на основі результатів студента (machine learning); інтеграція з SIEM для тренування blue team команд.

Чесний аналіз обмежень показує, що поточна реалізація завершена для бакалаврського рівня й водночас формує основу для подальших ітерацій: напрями розвитку охоплюють і короткострокові доробки (нові сценарії), і архітектурні розширення (multi-user, інші гіпервізори), придатні для курсових, магістерських і дослідницьких робіт.

3.8 Висновки до розділу

У третьому розділі проведено апробацію розроблених сценаріїв та оцінено можливості тренажера як середовища для їх створення й виконання; отримані результати проаналізовано за кількома напрямами. За результатами виконаної роботи:

1. Обґрунтовано й реалізовано методику апробації через інструментальні вимірювання, самотестування сценаріїв, тестування модульної архітектури й порівняння з аналогами.

2. Проведено інструментальне вимірювання на типовій конфігурації (Intel i5, 16 ГБ RAM, NVMe SSD): час старту сценаріїв, споживання RAM), займаний дисковий простір – відповідають проєктним характеристикам і прийнятні для типової апаратної конфігурації кінцевого користувача.

3. Виконано повне самотестування всіх п’яти сценаріїв: сумарна фактична тривалість 160 хв проти 290 запланованих — фактичне проходження виявилося на 44,8% швидшим за планові оцінки, тобто планування було свідомо консервативним (із запасом). Усі 55 кроків пройшли валідатор без виявлення витоків відповідей у підказках.

4. Експериментально підтверджено ефективність модульної архітектури

створення тестового сценарію трьома способами – ручний JSON 52 хв, CLI new_scenario.py 20 хв (−61,5%), веб-редактор 8 хв (−84,6%). Валідатор виявив усі 14 з 14 навмисно внесених помилок (100%).

5. Проведено порівняння з чотирма аналогами (HackTheBox, TryHackMe, RangeForce, HTB Academy). Тренажер не конкурує за обсягом контенту, але істотно переважає у доступності, локалізації і відкритості для модифікації, формуючи унікальну нішу цільового інструменту для ОПП спеціальності 125.

6. Підтверджено узгодженість матеріалу з методологіями: PTES (5–7 фаз у різних сценаріях), MITRE ATT&CK (36 технік у 9 тактиках), OWASP Top 10 (7 з 10 категорій), NIST SP 800-115 (7 з 8 підфаз), що забезпечує перенесення набутих навичок у професійну діяльність.

7. Чесно описано обмеження: відсутність MITM- та AD-сценаріїв через ресурси стенду; single-user архітектура без централізованого моніторингу. Сформульовані напрямки розвитку, що можуть становити основу для подальших магістерських і дослідницьких робіт.

8. Результати апробації підтверджують, що тренажер є функціонально завершеним, технічно стабільним і методологічно обґрунтованим інструментом, готовим до інтеграції у навчальний процес кафедри для дисциплін мережевої безпеки й penetration testing.

У сукупності апробація демонструє завершеність тренажера на технічному, контентному й методологічному рівнях, а подальший розвиток визначено сформованими напрямами з урахуванням виявлених обмежень.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання – розроблено методику побудови навчальних сценаріїв мережевого penetration testing та програмний тренажер, що забезпечує їх виконання у віртуалізованому середовищі кіберполігону, готовий до інтеграції у навчальний процес кафедри.

Основні результати, отримані у роботі:

1. На основі систематичного аналізу теоретичних засад penetration testing, чотирьох методологій і моделей (PTES, NIST SP 800-115, OSSTMM, Cyber Kill Chain), фреймворку MITRE ATT&CK та існуючих кіберполігонів обґрунтовано вибір комбінованого підходу й сформульовано задачу розробки.

2. Запропоновано та реалізовано концепцію тренажера як локального вебзастосунку, що автоматизує запуск і керування віртуальним середовищем (Kali Linux, pfSense, Metasploitable 2, DC-1) – старт ВМ, відновлення snapshot та налаштування сегментованої топології – і проводить студента за покроковими сценаріями.

3. Реалізовано трирівневу архітектуру з розділенням рівнів представлення (вебінтерфейс Flask), логіки (модулі управління ВМ, сценаріїв і звітності) та даних (JSON-контент і SQLite).

4. Розроблено модульну систему контенту з трьома незалежними рівнями розширення (профілі ВМ, мережеві топології, сценарії у JSON), що дозволяє додавати контент без зміни коду тренажера.

5. Реалізовано розширений набір валідаторів відповідей (text_flag, regex, numeric_range, case_insensitive_substring, multiple_choice), що вирішує проблему невизначеності відповідей між різними snapshot-станами цілей.

6. Розроблено дворівневий набір засобів розширення: CLI-інструменти (validate_scenario.py та генератор шаблонів) і веб-редактор з довідником і автозаповненням MITRE ATT&CK для викладачів без досвіду роботи з JSON.

7. Розроблено методику побудови навчальних сценаріїв і за нею створено п’ять сценаріїв (сумарно 55 кроків) — базовий пентест Metasploitable 2, пентест веб-застосунку, багатоетапну атаку з транзитом до внутрішнього сегмента мережі, поглиблену мережеву розвідку та ухилення від базових засобів виявлення — з відображенням на техніки MITRE ATT&CK, градацією складності за фазами PTES і автоматизованою перевіркою підказок на розкриття правильної відповіді.

8. Підтверджено придатність розгорнутого середовища до виконання й подальшого розроблення сценаріїв: інструментальне вимірювання на типовій робочій станції показало прийнятні показники розгортання середовища та роботи вебінтерфейсу, а самотестування всіх п’яти сценаріїв — узгодженість їх фактичної та планової тривалості.

9. Підготовлено супровідну документацію з розгортання стенду (DEPLOYMENT.md).

Тренажер передбачений до інтеграції у навчальний процес кафедри як основний інструмент лабораторних робіт з мережевої безпеки та пентесту, засіб самостійної підготовки до CTF-змагань та олімпіад, основа для поповнення контенту в курсових і кваліфікаційних роботах, а також інструмент демонстрації принципів пентесту на лекціях. Завдяки модульній архітектурі та відкритості коду він може бути впроваджений і на споріднених кафедрах інших ЗВО.

На основі аналізу обмежень поточної версії визначено напрями розвитку: сценарії атак канального/мережевого рівня (MITM, ARP-спуфінг) і атак на Active Directory (Kerberoasting, Pass-the-Hash); інтеграція WebSSH-терміналу у вебінтерфейс; інтеграція з Moodle через REST API; розширення до multi-tenant режиму з сервером кафедри; проведення повноцінних студентських випробувань; поповнення набору сценаріями з відкритих CTF-змагань.

Сукупно у кваліфікаційній роботі представлено методику побудови навчальних сценаріїв і функціонально завершений програмний тренажер з модульною архітектурою, що підтримує як технічно підготовлених, так і нетехнічних користувачів-викладачів, методологічно узгоджений з провідними

галузевими стандартами та готовий до інтеграції у навчальний процес. Встановлена мета дослідження досягнута, усі сформульовані завдання виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Scarfone K. та ін. Technical Guide to Information Security Testing and Assessment. NIST Special Publication 800-115. Gaithersburg : National Institute of Standards and Technology, 2008. URL: https://csrc.nist.gov/pubs/sp/800/115/final (дата звернення: 18.03.2026).

2. Penetration Testing Execution Standard (PTES) Technical Guidelines / PTES Team. URL: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines (дата звернення: 15.03.2026).

3. Weidman G. Penetration Testing: A Hands-On Introduction to Hacking. San Francisco : No Starch Press, 2014. 528 p.

4. Wylie P. L., Crawley K. The Pentester BluePrint: Starting a Career as an Ethical Hacker. Hoboken : Wiley, 2020. 192 p.

5. Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 р. № 2163-VIII. Відомості Верховної Ради України. 2017. № 45. Ст. 403. URL: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення: 12.03.2026).

6. OWASP Top 10:2021. The Top 10 Web Application Security Risks / The Open Worldwide Application Security Project. URL: https://owasp.org/Top10/ (дата звернення: 22.03.2026).

7. Herzog P. The Open Source Security Testing Methodology Manual (OSSTMM) Version 3. Cardedeu : ISECOM, 2010. 213 p. URL: https://www.isecom.org/OSSTMM.3.pdf (дата звернення: 18.03.2026).

8. Hutchins E. M., Cloppert M. J., Amin R. M. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. Leading Issues in Information Warfare & Security Research. 2011. Vol. 1, No. 1. P. 80–106. URL: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf (дата звернення: 18.03.2026).

9. The MITRE ATT&CK Framework / The MITRE Corporation. URL: https://attack.mitre.org/ (дата звернення: 20.03.2026).

10. ATT&CK Matrix for Enterprise / The MITRE Corporation. URL: https://attack.mitre.org/matrices/enterprise/ (дата звернення: 20.03.2026).

11. Yamin M. M., Katt B., Gkioulos V. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. Computers & Security. 2020. Vol. 88. Article 101636. DOI: 10.1016/j.cose.2019.101636.

12. Davis J., Magrath S. A Survey of Cyber Ranges and Testbeds: Executive Summary. Edinburgh : DSTO, 2013. 35 p. URL: https://apps.dtic.mil/sti/citations/ADA594895 (дата звернення: 28.03.2026).

13. Cyber Defence Exercises: NATO CCDCOE Locked Shields / NATO Cooperative Cyber Defence Centre of Excellence. URL: https://ccdcoe.org/exercises/locked-shields/ (дата звернення: 25.03.2026).

14. Vykopal J., Ošlejšek R., Čeleda P., Vizváry M., Tovarňák D. KYPO Cyber Range: Design and Use Cases. Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017). Madrid : SciTePress, 2017. Vol. 1. P. 310–321. DOI: 10.5220/0006428203100321.

15. Metasploitable 2 Exploitability Guide / Rapid7 Inc. URL: https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/ (дата звернення: 05.04.2026).

16. Oracle VM VirtualBox User Manual: Version 7.0 / Oracle Corporation. URL: https://www.virtualbox.org/manual/UserManual.html (дата звернення: 25.03.2026).

17. Portnoy M. Virtualization Essentials. 2nd ed. Indianapolis : Sybex, Wiley, 2016. 336 p.

18. Lyon G. F. Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Sunnyvale : Nmap Project, 2009. 468 p.

19. ISC2 Cybersecurity Workforce Study 2024: Cybersecurity Workforce Gap Reaches 4.8 Million / International Information System Security Certification Consortium. Clearwater : ISC2 Research Insights, 2024. URL: https://www.isc2.org/research/workforce-study (дата звернення: 15.04.2026).

20. Metasploit Framework Documentation / Rapid7 Inc. URL: https://docs.metasploit.com/ (дата звернення: 10.04.2026).

21. Kali Linux Official Documentation / OffSec Services Limited. URL: https://www.kali.org/docs/ (дата звернення: 02.04.2026).

22. Petersen R. та ін. Workforce Framework for Cybersecurity (NICE Framework). NIST Special Publication 800-181 Revision 1. Gaithersburg : National Institute of Standards and Technology, 2020. 33 p. DOI: 10.6028/NIST.SP.800-181r1.

23. CompTIA PenTest+ Certification Exam Objectives (PT0-002) / The Computing Technology Industry Association. URL: https://www.comptia.org/certifications/pentest (дата звернення: 28.03.2026).

24. Rashid A. та ін. The Cyber Security Body of Knowledge (CyBOK) Version 1.1. Bristol : CyBOK Project, 2021. URL: https://www.cybok.org/ (дата звернення: 30.03.2026).

25. Vykopal J., Vizváry M., Ošlejšek R., Čeleda P., Tovarňák D. Lessons Learned From Complex Hands-on Defence Exercises in a Cyber Range. IEEE Frontiers in Education Conference (FIE 2017). Indianapolis : IEEE, 2017. P. 1–8. DOI: 10.1109/FIE.2017.8190713.

26. Про рішення Ради національної безпеки і оборони України від 14 травня 2021 року «Про Стратегію кібербезпеки України» : Указ Президента України від

26.08.2021 р. № 447/2021. URL: https://www.president.gov.ua/documents/4472021-40013 (дата звернення: 12.03.2026).

27. Flask Web Development Documentation: Version 3.0.x / Pallets Projects. URL: https://flask.palletsprojects.com/en/3.0.x/ (дата звернення: 02.04.2026).

28. The Python Language Reference: Release 3.12 / Python Software Foundation. URL: https://docs.python.org/3.12/reference/ (дата звернення: 02.04.2026).

29. Pham C., Tang D., Chinen K., Beuran R. CyRIS: A Cyber Range Instantiation System for Facilitating Security Training. Proceedings of the Seventh Symposium on Information and Communication Technology (SoICT 2016). Ho Chi Minh City : ACM, 2016. P. 251–258. DOI: 10.1145/3011077.3011087.

30. Wood R. Damn Vulnerable Web Application (DVWA). GitHub. URL: https://github.com/digininja/DVWA (дата звернення: 05.04.2026).

31. DC: 1. Vulnerable Machine for Penetration Testing Practice / DCAU. VulnHub. URL: https://www.vulnhub.com/entry/dc-1,292/ (дата звернення: 05.04.2026).

32. Wireshark User's Guide: Version 4.2 / The Wireshark Team. URL: https://www.wireshark.org/docs/wsug_html_chunked/ (дата звернення: 10.04.2026).

33. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. Київ : ДП «УкрНДНЦ», 2016. 24 с.

34. CVE-2011-2523: Backdoor in vsftpd 2.3.4 / The MITRE Corporation. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2523 (дата звернення: 12.04.2026).

35. CVE-2018-7600: Drupal Core Remote Code Execution (Drupalgeddon2) / The MITRE Corporation. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7600 (дата звернення: 12.04.2026).

36. Делембовський М. М., Гнатюк С. О., Корнійчук Б. В. Базові принципи розробки та застосування кіберполігонів для підготовки фахівців з кібербезпеки. Ukrainian Scientific Journal of Information Security. 2025. Т. 31, № 2. С. 104–111. DOI: 10.18372/2225-5036.31.20703.

ДОДАТОК А

Лістинги ключових модулів програмного коду

У цьому додатку наведено фрагменти ключових модулів програмного коду тренажера, що ілюструють основні архітектурні рішення, описані у розділі

А.1. Модуль управління віртуальними машинами реалізує шаблон проєктування Facade над командним інтерфейсом VBoxManage. Клас VMManager інкапсулює всі низькорівневі команди, надає типобезпечні методи запуску/зупинки/відновлення snapshots та реалізує захист від command injection через регулярний вираз перевірки імен VM.

Лістинг А.1 – фрагмент модуля vm_manager.py

```
"""
    Модуль управління віртуальними машинами через VBoxManage.

    Інкапсулює всі команди VBoxManage у зручний об'єктно-орієнтований інтерфейс.
"""
import subprocess
import re
import os
import time
import logging
from typing import Optional, List, Dict

from .config import VBOXMANAGE
class VMManager:
    """
        Менеджер віртуальних машин.

        Реалізує шаблон проєктування Facade над VBoxManage CLI.
    """

        # Допустимі символи в іменах VM (захист від command injection)
        _NAME_PATTERN = re.compile(r"^[a-zA-Z0-9_\-\. ]+$")

    def __init__(self, vboxmanage_path: str = VBOXMANAGE):
            self.vboxmanage = vboxmanage_path

            # Формат рядка: "VMName" {uuid}
```

```
            for line in result.stdout.splitlines():
                match = re.match(r'"([^"]+)"\s+\{', line)
                if match:
                    vm_names.append(match.group(1))
            return vm_names

    def vm_exists(self, name: str) -> bool:
        """Перевірити чи існує VM з вказаним іменем."""
            self._validate_name(name)
            return name in self.list_vms()

    def get_vm_state(self, name: str) -> str:
        """
            Отримати стан VM.
            Можливі значення: running, paused, stopped, aborted, saved.
        """
            self._validate_name(name)
            result = self._run(["showvminfo", name, "--machinereadable"])
            if result.returncode != 0:
                if "Could not find" in result.stderr:
                    raise VMNotFoundError(f"VM {name} не знайдено")
                raise VMError(f"showvminfo failed: {result.stderr}")

            for line in result.stdout.splitlines():
                if line.startswith("VMState="):
                    state = line.split("=", 1)[1].strip().strip('"')
                    return state
            return "unknown"

    def is_running(self, name: str) -> bool:
        """Чи запущена VM."""
            try:
                return self.get_vm_state(name) == "running"
```

```
                raise VMError(f"Не вдалось запустити VM {name}: {result.stderr}")

            # Чекаємо поки VM повністю запуститься
            for _ in range(30):
                if self.is_running(name):
                    logger.info(f"VM {name} запущена")
                    return True
                time.sleep(1)

            raise VMError(f"VM {name} не запустилась за 30 секунд")

    def stop_vm(self, name: str, force: bool = False) -> bool:
        """
            Зупинити VM.
            :param force: примусова зупинка (poweroff) замість коректної (acpipowerbutton)
        """
            self._validate_name(name)
            if not self.is_running(name):
                logger.info(f"VM {name} вже зупинена")
                return True

            action = "poweroff" if force else "acpipowerbutton"
            result = self._run(["controlvm", name, action], timeout=30)
            if result.returncode != 0:
                raise VMError(f"Не вдалось зупинити VM {name}: {result.stderr}")

            # Чекаємо поки VM зупиниться
            for _ in range(30):
                if not self.is_running(name):
                    logger.info(f"VM {name} зупинена")
```

```
                return True

    def create_snapshot(self, name: str, snapshot: str) -> bool:
        """Створити snapshot VM."""
            self._validate_name(name)
            self._validate_name(snapshot)

            # Якщо VM запущена – зупиняємо
            if self.is_running(name):
                self.stop_vm(name)

            result = self._run(["snapshot", name, "take", snapshot], timeout=60)
            if result.returncode != 0:
                raise VMError(f"Не вдалось створити snapshot:
{result.stderr}")
            return True

    def restore_snapshot(self, name: str, snapshot: str) -> bool:
        """Відновити VM до snapshot."""
            self._validate_name(name)
            self._validate_name(snapshot)

            if not self.has_snapshot(name, snapshot):
                raise VMError(f"Snapshot {snapshot} не існує для {name}")

            # VM має бути зупинена
            if self.is_running(name):
                self.stop_vm(name, force=True)
```

А.2. Модуль завантаження сценаріїв читає JSON-файли з каталогу scenarios/ та парсить їх у внутрішнє представлення. Клас Step реалізує валідацію відповіді користувача з підтримкою регістронезалежного порівняння, а метод to_dict() забезпечує серіалізацію без витоку правильних відповідей (флаг include_answer передається лише при генерації звіту викладача).

Лістинг А.2 – фрагмент модуля scenario_loader.py

```
"""
        Завантажувач сценаріїв з JSON-файлів.
"""
import os
import json
import logging
from typing import Dict, List, Optional

from .config import SCENARIOS_DIR

        logger = logging.getLogger(__name__)


class Step:
    """Один крок сценарію."""

            REQUIRED = ["number", "title", "instruction", "flag_answer", "points"]


    def __init__(self, data: dict):
                for field in self.REQUIRED:
                    if field not in data:
                        raise ValueError(f"Step missing field: {field}")

                self.number = data["number"]
                self.title = data["title"]
                self.instruction = data["instruction"]
                self.expected_command = data.get("expected_command", "")
                self.flag_prompt = data.get("flag_prompt", "Введіть відповідь:")
                self.flag_answer = str(data["flag_answer"])
                self.points = int(data["points"])
                self.attck_technique = data.get("attck_technique", "")
                self.hint = data.get("hint", "")
                self.control_question = data.get("control_question", "")
                self.validator_type = data.get("validator_type", "text_flag")
                self.case_sensitive = data.get("case_sensitive", False)
```

```
    def check_answer(self, user_answer: str) -> bool:
        """Перевірити відповідь користувача."""
            user = user_answer.strip()
            correct = self.flag_answer.strip()
            if not self.case_sensitive:
                return user.lower() == correct.lower()
            return user == correct

    def to_dict(self, include_answer: bool = False) -> dict:
        """Серіалізувати. include_answer=True тільки для звітів!"""
            d = {
            "number": self.number,
            "title": self.title,
            "instruction": self.instruction,
            "expected_command": self.expected_command,
            "flag_prompt": self.flag_prompt,
            "points": self.points,
            "attck_technique": self.attck_technique,
            "hint": self.hint,
            "control_question": self.control_question,
        }
            if include_answer:
                d["flag_answer"] = self.flag_answer
            return d


class Scenario:
    """Один навчальний сценарій."""

        REQUIRED = ["id", "title", "level", "required_vms", "steps"]
        VALID_LEVELS = ["beginner", "medium", "advanced"]

    def __init__(self, data: dict):
            for field in self.REQUIRED:
                if field not in data:
                    raise ValueError(f"Scenario missing field: {field}")
```

```
            if data["level"] not in self.VALID_LEVELS:
                raise ValueError(f"Invalid level: {data['level']}")
            self.steps = [Step(s) for s in data["steps"]]
            if not self.steps:
                raise ValueError("Scenario must have at least one step")

    @property
    def total_points(self) -> int:
        return sum(s.points for s in self.steps)

    def get_step(self, number: int) -> Optional[Step]:
        for step in self.steps:
            if step.number == number:
                return step
        return None

    def to_dict(self, include_answers: bool = False) -> dict:
        return {
            "id": self.id,
            "title": self.title,
            "level": self.level,
            "duration_minutes": self.duration_minutes,
            "max_attempts": self.max_attempts,
            "network": self.network,
            "required_vms": self.required_vms,
            "learning_objectives": self.learning_objectives,
            "attck_techniques": self.attck_techniques,
            "theory_intro": self.theory_intro,
            "total_points": self.total_points,
            "total_steps": len(self.steps),
            "steps": [s.to_dict(include_answers) for s in self.steps],
        }


class ScenarioLoader:
    """Завантажувач сценаріїв з каталогу scenarios/."""

    def __init__(self, scenarios_dir: str = SCENARIOS_DIR):
        self.scenarios_dir = scenarios_dir
```

```
        self._scenarios: Dict[str, Scenario] = {}
        self.load_all()
```

ДОДАТОК Б

Приклад JSON-сценарію

У цьому додатку наведено фрагмент JSON-файлу одного з реалізованих навчальних сценаріїв – «Сценарій 1: Базовий пентест Metasploitable 2» (beginner-рівень, 10 кроків, 90 хвилин). Через обсяг приведено метадані сценарію та повний опис перших двох кроків. Структура інших кроків аналогічна.

Лістинг Б.1 – beginner_metasploitable.json (метадані + 2 кроки)

"id": "beginner_metasploitable",

"learning_objectives": [

"Засвоєння базових технік мережевої розвідки (host discovery, port scanning)",

"Робота з Nmap для виявлення сервісів та їх версій",

"Самостійний пошук CVE за виявленою версією ПЗ",

"Використання Metasploit Framework для експлуатації відомих вразливостей",

"Розуміння послідовності фаз методології PTES",

"Базові навички постексплуатації Linux",

"Відновлення паролів за хешами (offline cracking) за допомогою John the Ripper"

],

"level": "beginner",

"max_attempts": 3,

"network": "dmz_internal",

"required_snapshots": {

"kali": "clean_state",

"metasploitable2": "clean_state"

},

"required_vms": [

"kali",

```
      "metasploitable2"
    ],
    "steps": [
      {
        "attck_technique": "T1018",
        "control_question": "Чому ARP-сканування ефективніше за ICMP-сканування у локальній мережі?",
        "expected_command": "sudo netdiscover -r 192.168.1.0/24",
        "flag_answer": "192.168.1.100",
        "flag_example": "192.168.1.50",
        "flag_prompt": "Введіть IP-адресу цільової машини Metasploitable 2:",
        "hint": "У виводі netdiscover буде кілька рядків IP-MAC. Приберіть IP, які вже відомі (Kali 192.168.1.10 та шлюз 192.168.1.1). Хост, що лишився, — ваша ціль.",
        "instruction": "<p>Будь-який пентест починається з розвідки: спершу треба зрозуміти, <em>хто взагалі є</em> у мережі. Знайдемо «живі» хости у сегменті DMZ (діапазон 192.168.1.0/24).</p><p>Найшвидший спосіб у локальній мережі — ARP-сканування. Воно працює на канальному рівні, тому спрацьовує навіть тоді, коли хости ігнорують ping (ICMP):</p><pre>sudo netdiscover -r 192.168.1.0/24</pre><p>Розбір команди: <code>netdiscover</code> розсилає ARP-запити; <code>-r 192.168.1.0/24</code> задає діапазон сканування. Зачекайте 15–30 с, поки зберуться відповіді. У виводі будуть пари IP ↔ MAC.</p><p>Викресліть ті, що вже відомі з опису (Kali — 192.168.1.10, шлюз pfSense — 192.168.1.1). Те, що залишиться, — і є вашою ціллю.</p>",
        "number": 1,
        "points": 10,
        "title": "Виявлення активних хостів у мережі DMZ"
      },
```

{
"attck_technique": "T1046",
"control_question": "У чому різниця між -sS (SYN scan) та -sT (TCP connect scan)?",
"expected_command": "nmap -sS 192.168.1.100 | grep open | wc -l",
"flag_answer": "23",
"flag_example": "число",
"flag_prompt": "Скільки відкритих портів виявлено? Введіть число:",
"hint": "Виконайте 'nmap -sS <ip> | grep open | wc -l' — команда сама виведе кількість відкритих портів. Це число й є відповіддю.",
"instruction": " Ціль знайдено — тепер подивимось, які сервіси на ній працюють. Запустіть SYN-сканування проти знайденого IP: <pre>nmap -sS 192.168.1.100</pre> Розбір команди: <code>nmap</code> — сканер мережі/портів; <code>-sS</code> — SYN-скан (напіввідкритий): nmap надсилає лише SYN і не завершує з'єднання — швидко й залишає менше слідів у логах; далі — IP цілі (підставте свій із кроку 1). Щоб побачити лише відкриті порти у зручному вигляді, відфільтруйте вивід: <pre>nmap -sS 192.168.1.100 | grep open</pre> <code>grep open</code> залишить рядки відкритих портів (по одному на рядок). А щоб не рахувати руками — додайте підрахунок рядків: <pre>nmap -sS 192.168.1.100 | grep open | wc -l</pre> <code>wc -l</code> рахує кількість рядків, тобто кількість відкритих портів. Саме це число й потрібно ввести. ",
"number": 2,
"points": 10,
"title": "Сканування відкритих портів"
},
{
"attck_technique": "T1046",

"control_question": "Чому застаріле ПЗ є основним джерелом вразливостей у системах?",

"expected_command": "nmap -sV -p 21 192.168.1.100",

"flag_answer": "2.3.4",

"flag_example": "1.2.3",

"flag_prompt": "Введіть точну версію FTP-сервера (формат X.Y.Z):",

"hint": "Знайдіть рядок з портом 21/tcp. У колонці VERSION буде назва ПЗ і три числа через крапку. Введіть лише числа з крапками, без назви.",

*"instruction": " Знати відкриті порти замало — потрібні точні версії сервісів, бо саме за версією шукають відомі вразливості (CVE). Просканіруємо FTP-порт із визначенням версії: <pre>nmap -sV -p 21 192.168.1.100</pre> Розбір команди: <code>-sV</code> — визначення версій: nmap під'єднується до сервісу й читає його banner (рядок із назвою ПЗ та номером версії);
<code>-p 21</code> — сканувати лише порт 21 (FTP). У колонці VERSION побачите щось на кшталт <code>vsftpd 2.X.Y</code>. Потрібні саме три числа версії. ",*

"number": 3,

"points": 15,

"title": "Визначення версії FTP-сервера"

},

ДОДАТОК В

Конфігураційні файли VM та мережевої топології

Модульна архітектура тренажера передбачає, що додавання нової віртуальної машини або мережевої топології виконується через створення JSON-файлу у відповідному каталозі без зміни програмного коду. Нижче наведено приклади таких файлів.

В.1. Профіль атакуючої машини Kali Linux містить ім'я VM у VirtualBox, мережеві адаптери, очікувану IP-адресу та назву базового snapshot для відновлення початкового стану.


```
{
  "id": "kali",
  "display_name": "Kali Linux",
  "ova_file": "kali-linux-2024.ova",
  "category": "attacker",
  "default_resources": {
    "ram_mb": 2048,
    "cpu_cores": 2,
    "disk_gb": 40
  },
  "default_network": "dmz",
  "default_ip": "192.168.1.10",
  "snapshot_name": "clean_state",
  "credentials": {
    "username": "kali",
    "password": "kali"
  },
  "vulnerabilities_summary": "Атакуюча машина — інструменти пентесту (Nmap, Metasploit, Hydra, Wireshark)",
  "min_ram_mb": 1024,
```

```
  "start_mode": "gui"
}
```

В.2. Файл мережевої топології задає сегментацію кіберполігону. Топологія dmz_internal об'єднує два сегменти (DMZ та Internal Network) через pfSense-роутер, що забезпечує необхідність pivot для атак на внутрішні цілі.

Лістинг В.2 – networks/dmz_internal.json

```
{
  "id": "dmz_internal",
  "display_name": "DMZ + Internal через pfSense",
  "description": "Класична корпоративна топологія з демілітаризованою зоною та внутрішньою мережею, розділеними міжмережевим екраном pfSense.",
  "subnets": [
    {
      "name": "dmz-net",
      "cidr": "192.168.1.0/24",
      "gateway": "192.168.1.1",
       "description": "DMZ — публічна зона з атакуючою машиною та зовнішніми сервісами"
    },
    {
      "name": "internal-net",
      "cidr": "10.10.10.0/24",
      "gateway": "10.10.10.1",
      "description": "Внутрішня мережа — захищена від прямого доступу з DMZ"
    }
  ],
  "router": {
```

```
    "vm_id": "pfsense",
    "interfaces": {
      "wan": {"network": "dmz-net", "ip": "192.168.1.1"},
      "lan": {"network": "internal-net", "ip": "10.10.10.1"}
    }
  },
  "firewall_rules": [
    "block from dmz-net to internal-net",
    "allow established connections",
    "allow all from internal-net (LAN)"
  ]
}
}
```

ДОДАТОК Г. Допоміжні таблиціТаблиця

Г.1 – Порівняння платформ віртуалізації для кіберполігону

Платформа	Тип гіпервізора	Модель ліцензування	Знімки стану	Рівень мережевої ізоляції	Складність розгортання
VMware ESXi	1-й тип (bare-metal)	Комерційна	так	Високий	Середня
Proxmox VE	1-й тип (bare-metal)	Відкритий код	так	Високий	Середня
Oracle VirtualBox	2-й тип (hosted)	Безкоштовна	так	Середній	Низька
VMware Workstation	2-й тип (hosted)	Комерційна	так	Середній	Низька

Таблиця Г.2 – Інструменти мережевого пентесту за фазами PTES та їх маппінг на MITRE ATT&CK

Фаза стандарту PTES	Інструментальний засіб	Функціональне призначення у тренажері	Техніка матриці MITRE ATT&CK
Intelligence Gathering (Збір інформації)	Nmap	сканування активних портів, визначення версій мережевих служб та операційних систем	T1046, T1018
Intelligence Gathering (Збір інформації)	Netdiscover	оперативне виявлення хостів у локальній мережі на основі ARP-запитів	T1018
Vulnerability Analysis (Аналіз уразливостей)	OpenVAS	автоматизований пошук відомих уразливостей (дефектів безпеки) за реєстрами CVE	T1595
Vulnerability Analysis (Аналіз уразливостей)	Nikto	аудит безпеки конфігурацій вебсерверів та виявлення небезпечних файлів	T1595
Vulnerability Analysis (Аналіз уразливостей)	Enum4linux	збір та перелічення системної інформації через мережевий протокол SMB	T1087, T1135
Exploitation (Експлуатація)	Metasploit Framework	Frameworkреалізація модульних експлойтів для експлуатації виявлених дефектів безпеки	T1210, T1190
Exploitation (Експлуатація)	Hydra	автоматизований перебір автентифікаційних даних (brute-force) за словником	T1110
Exploitation (Експлуатація)	SQLmap	автоматизоване виявлення та експлуатація SQL-ін'єкцій у вебзастосунках	T1190
Post-Exploitation (Постексплуатація)	Meterpreter	організація інтерактивного керування, вивантаження хешів, логування клавіш та реалізація латерального руху (pivoting)	T1021, T1003
Post-Exploitation (Постексплуатація)	LinPEAS / WinPEAS	автоматизований пошук векторів локального підвищення привілеїв в ОС Linux та Windows	T1548
Post-Exploitation (Постексплуатація)	Mimikatz	зчитування та вивантаження автентифікаційних даних (хешів, квитків Kerberos) з пам'яті Windows	T1003
Аналіз мережевого трафіку	Wireshark	перехоплення, глибокий аналіз та візуалізація пакетів мережевих протоколів	T1040
Аналіз мережевого трафіку	Responder	імітація відповідей на запити протоколів LLMNR, NBT-NS та mDNS (poisoning)	T1557

Таблиця Г.3 – Покриття тактик MITRE ATT&CK сценаріями тренажера

Тактика ATT&CK	Сц. 1	Сц. 2	Сц. 3	Сц. 4	Сц. 5
Reconnaissance (TA0043)	+	+	+	+	+
Initial Access (TA0001)	+	+	+	–	–
Execution (TA0002)	+	+	+	–	–
Persistence (TA0003)	–	–	–	–	–
Privilege Escalation (TA0004)	–	+	+	–	–
Defense Evasion (TA0005)	–	–	–	–	+
Credential Access (TA0006)	+	+	+	–	–
Discovery (TA0007)	+	+	+	+	+
Lateral Movement (TA0008)	–	–	+	–	–
Collection (TA0009)	+	+	+	+	+
Command and Control (TA0011)	–	–	+	–	+
Покрито тактик з 12	6	7	9	3	5

Таблиця Г.4 – Критерії оцінювання виконання сценарію

Фактор	Вплив на бали	Опис
Використання підказки	×0.7 (–30%)	Бали за крок зменшуються на 30% при перегляді hint
Кожна додаткова спроба	×(1 – 0.1·n)	За n-у додаткову спробу зменшення на 10%
Невиконаний крок	Незарахування балів	Якщо студент не пройшов крок

Таблиця Г.5 – Поля JSON-профілю віртуальної машини

Поле	Тип	Призначення	Обов'язкове
id	string	Унікальний ідентифікатор VM	Так
display_name	string	Назва для відображення в UI	Так
ova_file	string	Ім'я вихідного образу VM (довідково; індикатор наявності образу в каталозі vms/)	Так
category	string	attacker / target_linux / target_windows / router	Так
default_resources	object	RAM, CPU та диск за замовчуванням	Так
default_network	string	Мережа за замовчуванням	Так
default_ip	string	IP-адреса у мережі	Так
snapshot_name	string	Назва snapshot для clean state	Так
credentials	object	Стандартні облікові дані VM	Ні
vulnerabilities_summary	string	Короткий опис вразливостей	Ні
min_ram_mb	integer	Мінімально необхідна RAM	Ні

Таблиця Г.6 – Матриця розширюваності тренажера

Що додати/змінити	Час	Потребує редагування логіки	Хто може виконати
Новий сценарій на існуючих VM (через JSON)	30–60 хв	Ні	Викладач
Новий сценарій через веб-редактор	15–45 хв	Ні	Викладач
Нова VM	15–30 хв	Ні	Викладач
Нова мережева топологія	30–60 хв	Ні	Викладач
Зміна параметрів існуючої VM	5 хв	Ні	Викладач
Локалізація на іншу мову	2–4 год	Ні	Викладач
Новий тип валідатора відповіді	1–2 год	Так (мінімум)	Викладач зі знанням мови програмування Pyton

Таблиця Г.7 – Основні ендпоінти REST API тренажера

Ендпоінт	Метод	Призначення	Відповідь
/api/auth	POST	Авторизація студента (ПІБ, група)	JSON: {status}
/api/scenarios	GET	Отримати список сценаріїв	JSON: [{id, title, level, ...}]
/api/scenario/<id>/start	POST	Запустити сценарій	JSON: {session_id, operation_id}
/api/session/<id>/step/<n>/submit	POST	Надіслати відповідь на перевірку	JSON: {correct, points}
/api/vm/<name>/start	POST	Запустити VM	JSON: {status, mode}
/api/vm/<name>/restore	POST	Відновити snapshot	JSON: {status}
/api/history	GET	Історія виконаних сценаріїв	JSON: [{session_id, date, score}]
/api/report/<session_id>	GET	Отримати PDF-звіт	PDF-файл
/api/admin/scenarios	GET/POST	Список/збереження сценаріїв (admin)	JSON
/api/admin/scenarios/validate	POST	Валідація dict без збереження	JSON: {findings, errors_count}

Таблиця Г.8 – Аналіз ризиків безпеки тренажера

Ризик	Ймовірність	Контрзахід
Шкідливий код виходить з VM (VM escape)	Дуже низька	Використання актуальної версії VirtualBox 7.x з патчами
Вразлива VM атакує мережу студента	Виключено	Internal Network ізолює трафік від хостової ОС
Експлойти Metasploit вражають реальні системи	Виключено	Усі цілі – у Internal Network, без доступу до інтернету
Студент пошкоджує цільову VM	Низька	Snapshots дозволяють миттєво відновити стан
Витік даних з хосту через спільні папки	Виключено	Спільні папки не використовуються
Захоплення тренажера через вебінтерфейс	Низька	Flask слухає лише на localhost (127.0.0.1)
Несанкціонована зміна сценаріїв	Низька	Admin-зона захищена паролем (TEACHER_PASSWORD)
Path traversal через id сценарію	Виключено	Регулярний вираз ^[a-z][a-z0-9_-]{2,63}$ блокує невалідні id
Запуск шкідливих команд через тренажер	Виключено	викликається через жорсткий whitelist

Таблиця Г.9 – Перевірки CLI-валідатора сценаріїв

Категорія перевірки	Що перевіряється	Серйозність
JSON-схема	Наявність обов’язкових полів, коректність типів даних	Error
Значення level	Має бути одним з beginner / medium / advanced	Error
Посилання на VM	Кожний id у required_vms існує у vms/*.json	Error
Посилання на snapshots	Snapshot оголошено для VM зі списку required_vms	Warning
Посилання на мережу	Поле network посилається на наявний файл у networks/	Warning
Формат MITRE ATT&CK	Ідентифікатори відповідають шаблону T\d{4}(\.\d{3})?	Warning
Послідовність кроків	Номери йдуть як 1, 2, 3, ... без пропусків та дублів	Warning
Семантика validator_type	regex компілюється, numeric_range має формат "X" або "X-Y"	Error
Multiple choice options	options присутні і непорожні, flag_answer серед них	Error
Витоки відповідей у hint	Відповідь кроку не з’являється дослівно у його підказці	Error
Витоки відповідей у theory_intro	Відповідь будь-якого кроку не з’являється у вступі	Warning
Cross-step витоки	Відповідь кроку N не з’являється в instruction кроку M > N	Info

Таблиця Г.10 – Виявлення типових помилок CLI-валідатором

Категорія помилки	Внесено	Виявлено	Severity
Невалідне значення level	1	1	Error
Некоректна duration_minutes	1	1	Error
Посилання на неіснуючу VM	1	1	Error
required_snapshots не словник	1	1	Error
Відсутнє обов'язкове поле кроку	1	1	Error
Недодатні points	1	1	Error
Невідомий validator type	1	1	Error
Порожня flag_answer	1	1	Error
Невалідний numeric_range	1	1	Error
Невалідний regex у flag_answer	1	1	Error
Multiple_choice без options	1	1	Error
flag_answer поза options	1	1	Error
Витік відповіді у hint	2	2	Error

Таблиця Г.11 – Порівняння розробленого тренажера з комерційними аналогами

Критерій	Pentest Trainer	HackTheBox	TryHackMe	RangeForce
Модель доступу	Локально (offline)	Online (cloud VPN)	Online (browser+VPN)	Online (cloud)
Вартість для студента	0	$14/міс	$10/міс	Корпоративна (від $50k/рік)
Мова інтерфейсу	Українська	Англійська	Англійська	Англійська
Українські методичні матеріали	+	–	–	–
Структуровані сценарії	+ (5 з 55 кроків)	± (через Academy)	+ (Rooms)	+ (Modules)
Автономність (без інтернету)	+	–	–	–
Модифікація сценаріїв ЗВО	+ (повна, FOSS)	–	–	± (за окрему плату)
Інтеграція з оцінюванням	+ (PDF-звіти, ЄКТС)	– (тільки внутрішні бали)	– (профіль користувача)	+ (LMS-інтеграція)
Покриття мережевого пентесту	+ (3 з 5 сценаріїв)	+ (Pro Labs)	± (через Networks)	± (через модулі)
Веб-редактор сценаріїв	+ (вбудований)	–	–	–
Захист від витоку відповідей	+ (автовалідатор)	Не застосовно	Не застосовно	Не застосовно
Підтримка ATT&CK	+ (явний mapping)	± (для деяких)	± (для деяких)	+ (системний mapping)

Таблиця Г.12 – Покриття фаз методології PTES реалізованими сценаріями

Фаза PTES	Сц.1	Сц.2	Сц.3	Сц.4	Сц.5
1. Pre-engagement (підготовка)	–	–	+	–	+
2. Intelligence Gathering (розвідка)	+	+	+	+	+
3. Threat Modeling	–	+	+	–	+
4. Vulnerability Analysis (аналіз вразливостей)	+	+	+	+	+
5. Exploitation (експлуатація)	+	+	+	–	–
6. Post-Exploitation (постексплуатація)	+	+	+	–	–
7. Reporting (звітність)	+	+	+	+	+
Покрито фаз з 7	5	6	7	3	4

Таблиця Г.13 – Розподіл унікальних технік MITRE ATT&CK по тактиках

Тактика ATT&CK	Кількість технік	Приклади
Reconnaissance (TA0043)	4	T1595 Active Scanning, T1595.001 IP Blocks, T1590, T1592
Initial Access (TA0001)	3	T1190 Public-Facing App, T1133, T1078
Execution (TA0002)	4	T1059 Command Interpreter, T1059.004 Unix Shell, T1106, T1203
Privilege Escalation (TA0004)	3	T1068, T1548.001 SUID/GUID, T1548.003 Sudo
Defense Evasion (TA0005)	3	T1027 Obfuscation, T1562, T1070
Credential Access (TA0006)	5	T1003 OS Credential Dumping, T1110 Brute Force, T1552.001
Discovery (TA0007)	7	T1018, T1046 Network Service Discovery, T1083, T1082, T1087.001, T1087.002, T1135
Lateral Movement (TA0008)	3	T1021 Remote Services, T1021.001 RDP, T1021.004 SSH
Command and Control (TA0011)	4	T1090 Proxy, T1090.001, T1572 Tunneling, T1071
Сумарно унікальних технік	36	–

Таблиця Г.14 – Покриття OWASP Top 10 (2021) сценарієм DVWA

Категорія OWASP Top 10	Покрито у сценарії 2	Крок сценарію
A01 Broken Access Control	–	–
A02 Cryptographic Failures	+ (MD5 для паролів)	Крок 7
A03 Injection (SQL, Command, тощо)	+ (SQLi, Command Injection)	Кроки 3, 5, 6, 7, 8
A04 Insecure Design	+ (Brute force без lockout)	Крок 10
A05 Security Misconfiguration	+ (default credentials)	Крок 2
A06 Vulnerable Components	+ (через Nikto-розвідку)	Крок 1
A07 Authentication Failures	+ (Brute force admin/gordonb)	Крок 10
A08 Software/Data Integrity Failures	–	–
A09 Logging/Monitoring Failures	–	–
A10 SSRF	–	–

Таблиця Г.15 – Покриття фаз NIST SP 800-115 розробленим тренажером

Фаза NIST 800-115	Реалізація у тренажері
1. Planning	Реалізовано через theory_intro кожного сценарію, у якому формулюється мета та умови (scope тестування)
2. Discovery: Network Discovery	Кроки розвідки у сценаріях 1, 3, 4, 5 – Nmap, netdiscover, NSE-скрипти
2. Discovery: Vulnerability Identification	Кроки 3-5 сценаріїв з searchsploit, NSE vuln-скриптами, аналізом CVE
3. Attack: Gaining Access	Експлуатація через Metasploit (сценарії 1, 3), Command Injection (2), SQL Injection (2)
3. Attack: Escalating Privileges	Privilege escalation через SUID find у сценарії 3
3. Attack: System Browsing	Кроки 8-10 сценаріїв 1 та 2 – дослідження файлової системи цілі
3. Attack: Install Add. Tools	Не реалізовано (не входить у scope бакалаврської роботи)
4. Reporting	Автоматична генерація PDF-звіту після кожного сценарію

Таблиця Г.16 – Напрями подальшого розвитку тренажера

Напрям	Очікувана вигода	Обсяг робіт
Сценарій 6 – L2/L3 атаки (MITM)	Покриття атак канального рівня	1 нова VM + 1 сценарій, ~2 тижні
Сценарій 7 – Active Directory	Покриття атак на корпоративні мережі	2-3 нові VM + 1 сценарій, ~4 тижні
Інтегрований WebSSH-термінал	Робота без зовнішнього терміналу	Окрема Python+JS компонента, ~3 тижні
LMS-інтеграція (Moodle)	Автоматична передача оцінок у журнал	Plugin Moodle + API розширення, ~2 тижні
Multi-tenant режим	Сервер на кафедрі для всіх студентів	Архітектурне переосмислення, ~6+ тижнів
Підтримка KVM/libvirt	Робота на Linux-серверах кафедри	VMDriver-плагін + тестування, ~3 тижні
Студентські випробування	Емпіричні дані для покращення підказок	Координація з кафедрою, ~1 семестр
Сценарії з реальних CTF	Підготовка до участі студентів у змаганнях	По одному сценарію за 1-2 тижні