

АВТОМАТИЗОВАНЕ РОЗГОРТАННЯ ВРАЗЛИВИХ НАВЧАЛЬНИХ СЕРЕДОВИЩ ІЗ ЗАСТОСУВАННЯМ МОДЕЛІ МІНІМАЛЬНИХ ПРИВІЛЕЇВ

Зростання числа та складності кібератак висуває все жорсткіші вимоги до практичної підготовки фахівців, які працюють у сфері кібербезпеки. Водночас традиційна лекційна форма навчання часто не здатна забезпечити достатнього рівня прикладних навичок. Певним чином цю проблему вирішують навчальні кіберполігони, але наявні рішення не часто пропонують автоматизоване розгортання окремого, повністю ізольованого середовища для кожного учасника. Тож головна мета нашого дослідження полягає в обґрунтуванні, подальшому проєктуванні та, зрештою, програмній реалізації платформи. Ця платформа має забезпечувати автоматизоване розгортання вразливих стендів, які точно відтворюють задокументовані техніки, описані у фреймворку MITRE ATT&CK, але в контрольованому багатокористувацькому просторі, гарантуючи при цьому повну ізоляцію між користувачами за допомогою технологій контейнеризації.

У процесі роботи застосовано різноманітні методи. Це й системний аналіз існуючих кіберполігонів та вразливих стендів, і ретельний порівняльний аналіз механізмів ізоляції як на рівні контейнерів, так і віртуальних машин. Окрім того, були використані методи об'єктно орієнтованого проєктування архітектури програмного забезпечення, а також детально опрацьована модель мінімальних привілеїв для конфігурування контейнерів. Платформу побудовано за принципом багаторівневої архітектури (Nginx, Django/Daphne, PostgreSQL, Redis) з динамічною оркестрацією Docker-контейнерів, керованою через спеціалізований програмний інтерфейс.

Спроєктовано та реалізовано платформу, що на вимогу автоматично створює, а потім і знищує ізольовані вразливі стенди. Вона здатна зіставляти кожен такий стенд із конкретними техніками MITRE ATT&CK і, що важливо, підтримує додавання абсолютно нових лабораторних сценаріїв без потреби вносити зміни до її основного ядра. Запропоновано певну конфігурацію безпеки для контейнерів, засновану на принципі відмови від зайвих привілеїв, водночас зберігаючи можливість точкового послаблення цих обмежень для коректного відтворення певних технік. Порівняння обраного нами контейнерного підходу з традиційною віртуалізацією переконливо продемонструвало значно менший час розгортання та суттєво нижче споживання ресурсів.

Поєднання контейнеризації, автоматизованої оркестрації та інтеграції сценаріїв безпосередньо з фреймворком MITRE ATT&CK дає нам можливість створити масштабоване й, що головне, відтворюване середовище, що придатне для практичної підготовки у сфері кібербезпеки. Практична цінність роботи полягає в тому, що платформа може бути легко інтегрована як в освітній процес закладів вищої освіти, так і в системи підвищення кваліфікації для фахівців із захисту інформації.

Ключові слова: кібербезпека; MITRE ATT&CK; кіберполігон; вразливе середовище; контейнеризація; Docker; ізоляція середовищ; моделювання.

Вступ та постановка проблеми. Кількість і складність кібератак щорічно зростає, що висуває підвищені вимоги до підготовки фахівців з інформаційної безпеки. Традиційний лекційний підхід до навчання не дає достатнього рівня практичних навичок, які потрібні для виявлення, аналізу та протидії реальним загрозам. Щоб компенсувати цей розрив, в освітніх та корпоративних середовищах почали активно використовувати навчальні кіберполігони –

контрольовані середовища, де студенти і фахівці можуть безпечно відтворювати сценарії атак та відпрацьовувати захисні дії.

Одним із ключових кроків у стандартизації знань про кіберзагрози стало створення фреймворку MITRE ATT&CK, який систематизує тактики, техніки та процедури (TTP) реальних зловмисників. Завдяки йому з'явилась можливість структуровано описувати поведінку кіберзлочинців і будувати на цій основі сценарії емуляції загроз. Проте на практиці використання MITRE ATT&CK у навчальному процесі вимагає відповідної інфраструктури – вразливих стендів, де студенти зможуть відтворювати задокументовані техніки в ізолюваному середовищі.

Існуючі рішення для практичного навчання кібербезпеці мають низку обмежень. Комерційні платформи, такі як Hack The Box або TryHackMe, не завжди доступні в академічному середовищі через вартість або обмеження ліцензій. Окрім того, більшість цих рішень не передбачають автоматизованого розгортання окремого ізолюваного інстансу для кожного студента, що суттєво ускладнює їх застосування при роботі з навчальними групами.

Актуальність теми зумовлена потребою у платформі, яка б автоматизувала розгортання вразливих стендів з використанням конкретних технік MITRE ATT&CK, забезпечувала ізоляцію між користувачами та дозволяла додавати нові лабораторні сценарії. Контейнеризація добре підходить для розв'язання цієї задачі, оскільки дає змогу швидко створювати і видаляти ізолювані середовища, гарантувати відтворюваність конфігурацій та ефективно використовувати обчислювальні ресурси. При цьому окремої уваги заслуговує безпека самої платформи: вона за своєю природою містить навмисно вразливі компоненти, які мають бути надійно ізолювані від хост-системи та від середовищ інших користувачів.

Метою статті є документування підходів до автоматизації розгортання вразливих стендів, створення платформи для демонстрації MITRE ATT&CK у навчальних цілях на ізолюваних стендах, що не взаємодіють з іншими користувачами, з можливістю додавання нових лабораторних сценаріїв за допомогою технологій контейнеризації.

Огляд літератури (аналіз останніх досліджень і публікацій). Проблематика практичної підготовки фахівців з кібербезпеки та створення навчальних кіберполігонів стала активно досліджуватися в науковій літературі. Наприклад, у [1] подано систематизацію підходів до розробки кіберполігонів, тестових середовищ, а також до класифікації функцій, сценаріїв, архітектурних рішень тощо. У той же час основними характеристиками якісного кіберполігону є реалістичність сценаріїв, його ізолюваність і можливість автоматизованого розгортання середовища.

Прикладом академічного кіберполігону є KYPO, що детально описана в [2]. Автори показали, що шляхом використання хмари та віртуалізації можна масштабувати сценарії навчання, але для цього потрібно багато обчислювальних потужностей. Інші академічні кіберполігони, призначені для навчання, такі як CyTrONE або інтегровані середовища навчання, обговорювалися в [3] з акцентом на автоматизацію підготовки навчального контенту.

Фреймворк MITRE ATT&CK може бути використаний для того, щоб сформалізувати дії супротивника. Техніки ATT&CK виявили статистичний зв'язок в роботі [4], що робить їх відповідними для навчання за допомогою послідовних атак. Це можна вважати підтвердженням ідеї про прив'язку конкретних стендів до технік, закладеною в основі представленого фреймворку.

Контейнеризація є технологічним підґрунтям ізоляції навчальних середовищ. Перевагу контейнерів над повноцінною віртуалізацією за швидкістю запуску та щільністю розміщення проаналізовано в роботах [5, 6]. При цьому безпековий аспект контейнеризації завжди залишається критично важливим. Так, у своїй праці Комб з колегами [7] вже докладно проаналізували ризики, пов'язані з використанням Docker, а Султан та інші дослідники [8] систематизували можливі загрози та розробили захисні механізми для контейнерів, серед яких простори імен, контрольні групи та обмеження привілеїв. Огляд застосування імітаційного моделювання в контексті кібербезпеки, здійснений Каваком та його командою [9], додатково

підтверджує нагальну потребу в таких відтворюваних і керованих середовищах.

Хоча у науковій літературі існує безліч праць на цю тему, однак надзвичайно важливе завдання досі залишається поза належною увагою. Йдеться про здатність ефективно інтегрувати автоматизоване створення відокремленого, індивідуального середовища для кожного користувача. Цей процес має бути тісно пов'язаний з методиками MITRE ATT&CK, а також передбачати контрольоване зменшення привілеїв контейнерів, щоб мати змогу точно відтворювати певні техніки кібербезпеки. Власне, саме у заповненні цієї методологічної прогалини й полягає як головна мета, так і новизна цього дослідження.

Підготовка фахівців із кібербезпеки вимагає не лише теоретичних знань про вразливості та механізми захисту, а й практичного розуміння того, як зловмисники діють у реальних системах. Щоб систематизувати ці знання, було створено кілька спеціалізованих фреймворків, серед яких найбільш поширеним став MITRE ATT&CK.

MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) – це загальнодоступна база знань про тактики та техніки зловмисників, побудована на основі спостережень за реальними кібератаками [10]. Фреймворк розроблений некомерційною корпорацією MITRE у 2013 році як внутрішній дослідницький проєкт, а у 2015 році був відкритий для широкого загалу. З того часу він став стандартним інструментом для команд реагування на інциденти, аналітиків загроз, red team спеціалістів та розробників засобів захисту [11].

Фреймворк організований у вигляді матриці, де стовпці відповідають тактикам, а рядки – технікам. Тактика описує мету зловмисника на певному етапі атаки, тобто відповідає на запитання "чому" він виконує ту чи іншу дію. Техніка визначає конкретний спосіб досягнення цієї мети – відповідає на питання "як" [10]. Багато технік додатково розділені на підтехніки, що деталізують окремі варіанти виконання. Процедури фіксують конкретні випадки використання технік реальними групами зловмисників [12].

Крім каталогізації загроз, MITRE ATT&CK виконує важливу навчальну функцію. На його основі будуються сценарії емуляції атак (adversary emulation), де red team відтворює поведінку реальних груп зловмисників, дотримуючись задокументованих технік [10]. Це дає можливість тренувати захисників у наближених до реальних умовах, а не лише працювати з абстрактними вразливостями. Фреймворк також формує спільну термінологію, яка допомагає фахівцям з кібербезпеки однозначно описувати виявлені інциденти та обмінюватися інформацією [12].

Практична підготовка фахівців з кібербезпеки потребує середовищ, де можна безпечно відтворювати реальні атаки без ризику для діючих систем. Для цього створюються навчальні кіберполігони (cyber ranges) та окремі вразливі стенди – спеціально підготовлені середовища з відомими вразливостями, призначені для відпрацювання навичок виявлення та експлуатації [1]. Наукові дослідження підтверджують, що практичні вправи у кіберполігонах значно підвищують рівень підготовки порівняно з традиційним лекційним підходом [2]. Існуючі рішення у цій сфері можна умовно поділити на відкриті локальні проєкти, комерційні навчальні платформи та інструменти для емуляції атак.

Серед відкритих локальних рішень найбільш відомими є VulnHub та Metasploitable. VulnHub надає колекцію готових образів віртуальних машин з різними вразливостями, які користувач завантажує та запускає на своєму комп'ютері [13]. Metasploitable – це спеціально підготовлена віртуальна машина з набором вразливих сервісів, призначена для тестування інструментів на кшталт Metasploit Framework [14]. Основним недоліком таких рішень полягає у їх локальній природі: кожен студент має самостійно завантажити образ, налаштувати гіпервізор, розгорнути віртуальну машину та сконфігурувати мережу. У групі з десятків студентів це перетворюється на суттєву організаційну проблему, а будь-які відмінності у конфігурації середовища можуть призводити до ситуацій, коли у когось лабораторна робота просто не працює. Централізоване управління інстансами або відстеження прогресу у таких рішеннях відсутнє.

Комерційні платформи Hack The Box та TryHackMe вирішують ці проблеми по-іншому. TryHackMe орієнтований на структуроване навчання з покроковими інструкціями та навчальними шляхами (learning paths), що робить його зручним для початківців [15]. Hack The Box надає вразливі машини без покрокових підказок, імітуючи реальні умови тестування на проникнення, і пропонує власні сертифікації [16]. Обидві платформи забезпечують ізольовані середовища для кожного користувача і знімають з нього необхідність розгортання. Проте вони працюють за моделлю платної підписки, що обмежує їх доступність [15, 16]. Але найважливішим обмеженням для навчального закладу є закритість цих платформ: адміністратор не може створювати та додавати власні лабораторні сценарії, адаптовані під конкретний курс чи програму. Навчальний процес повністю залежить від контенту, який вирішить опублікувати провайдер платформи.

Проведений аналіз дозволяє виділити ключові обмеження існуючих рішень (табл. 1). Локальні проекти (VulnHub, Metasploitable) вимагають від кожного студента самостійного розгортання та налаштування середовища, що створює значний організаційний бар'єр при роботі з групами. Комерційні платформи (Hack The Box, TryHackMe) знімають цю проблему, але є закритими – адміністратор навчального процесу не має можливості додавати власні лабораторні сценарії.

Таблиця 1 – Порівняння наявних платформ практичної підготовки з кібербезпеки

Платформа	Тип доступу	Ізоляція на користувача	Автоматизоване розгортання	Прив'язка до MITRE ATT&CK	Розширюваність сценаріїв
VulnHub	Образи ВМ, безкоштовно	Локально, вручну	Ні	Опосередкова на	Обмежена
Metasploitable	Образ ВМ, безкоштовно	Локально, вручну	Ні	Опосередкова на	Ні
TryHackMe	Хмара, підписка	Так (кімнати)	Частково	Частково	Закрита
Hack The Box	Хмара, підписка	Так	Частково	Частково	Закрита
MITRE CALDERA	Відкритий код	Ні (емуляція)	Частково	Пряма	Так
KYPO Cyber Range	Відкритий код, ВМ	Так	Так	Опосередкова на	Так
Запропонована платформа	Відкрите розгортання, контейнери	Так (окремих контейнер)	Так	Пряма (за технікою)	Так (додавання стендів)

Виклад основного матеріалу (методи, результати). Одним із основних обмежень локальних вразливих стендів є необхідність розгортання повноцінних віртуальних машин для кожного сценарію. Традиційна віртуалізація на базі гіпервізора передбачає запуск окремої гостьової операційної системи для кожного екземпляра, що вимагає значних обчислювальних ресурсів і часу на запуск [6]. Контейнеризація пропонує принципово інший підхід до ізоляції, який краще відповідає задачі масового розгортання навчальних стендів.

Контейнер – це ізольований процес (або група процесів), що виконується на хост-системі та використовує її ядро, але має власний відокремлений простір файлової системи, мережі, ідентифікаторів процесів та інших системних ресурсів [17]. На відміну від віртуальної машини, контейнер не потребує окремого ядра операційної системи, що робить його значно легшим – типовий образ контейнера займає від десятків до сотень мегабайт замість гігабайтів у випадку VM, а запуск відбувається за секунди, а не хвилини [18].

Ізоляція контейнерів у Linux забезпечується двома механізмами ядра. Простори імен

(namespaces) розділяють видимість системних ресурсів між процесами: кожен контейнер бачить тільки свої процеси (PID namespace), свою файлову систему (mount namespace), свій мережевий стек (network namespace) та свої ідентифікатори користувачів (user namespace) [17]. Контрольні групи (cgroups) обмежують кількість ресурсів, які контейнер може споживати: оперативну пам'ять, процесорний час, кількість процесів та інші параметри [17]. Саме ці два механізми перетворюють звичайний процес на ізольоване середовище виконання.

Додатковий рівень безпеки забезпечується системою Linux capabilities. Традиційно процеси у Linux мали лише два рівні привілеїв – це звичайний користувач або root з повним доступом. Capabilities дозволяють розбити права root на окремі дрібні привілеї і надавати процесу тільки ті, які йому дійсно потрібні [19]. Docker за замовчуванням запускає контейнери з обмеженим набором capabilities, а за допомогою параметрів `cap_drop` та `cap_add` дозволяє точно контролювати, які привілеї доступні всередині контейнера [20]. Цей механізм є особливо важливим для платформи з вразливими стендами, де необхідно забезпечити баланс між двома вимогами: контейнер має бути достатньо обмеженим, щоб не становити загрозу для хост-системи, але водночас деякі лабораторні роботи (наприклад, на підвищення привілеїв) потребують певних додаткових capabilities для коректної роботи сценарію.

Найпоширенішою платформою контейнеризації є Docker, який надає зручний інтерфейс для створення, запуску та управління контейнерами [20]. Docker використовує концепцію образів (images) – шаблонів файлової системи, з яких створюються контейнери. Образи описуються через Dockerfile, де покроково вказуються базовий образ, встановлення пакетів, копіювання файлів та інші операції [20]. Це забезпечує повну відтворюваність середовища: один і той самий Dockerfile завжди дає ідентичний результат, незалежно від того, на якій машині його зібрано.

Для задачі автоматизованого розгортання вразливих стендів контейнеризація має кілька принципових переваг перед віртуалізацією. Швидкість запуску дозволяє створювати новий інстанс для студента за лічені секунди, а не чекати хвилинами на завантаження віртуальної машини. Легковагість образів дає змогу тримати на одному сервері десятки одночасних інстансів, тоді як кількість повноцінних VM на тому ж обладнанні була б значно меншою. Відтворюваність гарантує, що кожен студент отримує ідентичне середовище. А одноразовість контейнерів означає, що після завершення сесії інстанс просто знищується і не потрібно відкочувати стан системи або очищати артефакти попереднього використання (рис. 1).

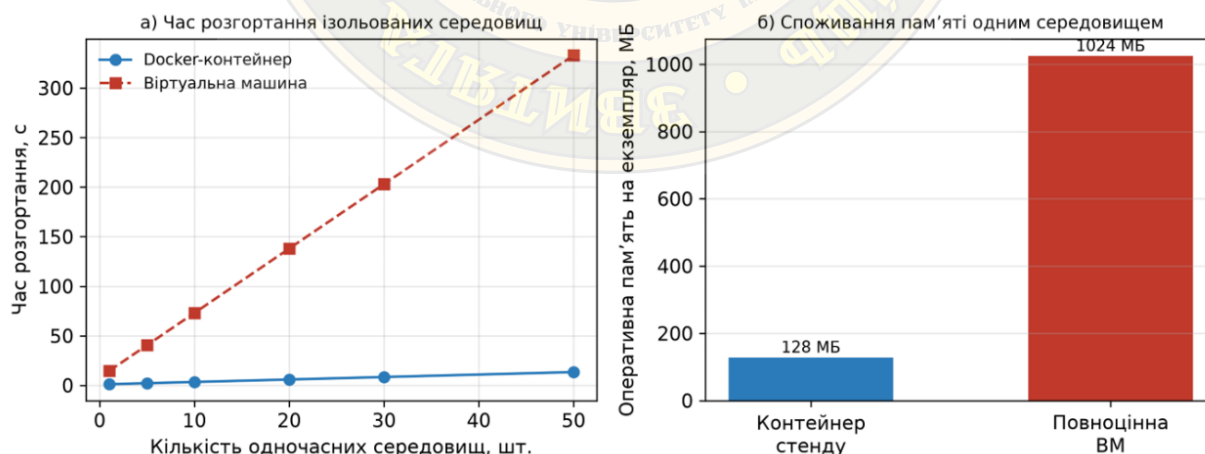


Рисунок 1 – Порівняння контейнерної та повної віртуалізації за швидкодією та ресурсами

Водночас контейнерна ізоляція є менш жорсткою, ніж у віртуальних машин, оскільки контейнери використовують спільне ядро хост-системи [18]. Це вимагає уважного підходу до налаштування безпеки, особливо коли контейнери містять навмисно вразливе програмне забезпечення. Правильна конфігурація capabilities, cgroups та мережевої ізоляції дозволяє мінімізувати ці ризики, що детально розглядається у практичній частині роботи.

Для реалізації платформи автоматизованого розгортання вразливих стендів необхідно обрати технології, які забезпечать управління контейнерами, взаємодію з користувачем у реальному часі, надійне зберігання даних та безпечний доступ до лабораторних середовищ.

Мова програмування та веб-фреймворк. Як основну мову обрано Python, яка має розвинену екосистему бібліотек для роботи з Docker і широко використовується у сфері кібербезпеки. Серед Python-фреймворків розглядалися Django та Flask. Flask є мінімалістичним мікрофреймворком, який дає розробнику свободу у виборі компонентів, але вимагає ручного підключення ORM, автентифікації та адміністративного інтерфейсу. Django надає все це з коробки, включаючи повноцінну адмінпанель [21].

Підтримка WebSocket. Термінальні лабораторні роботи потребують двосторонньої взаємодії у реальному часі між браузером користувача та терміналом всередині контейнера. Стандартний HTTP працює за моделлю запит-відповідь і не підтримує постійне з'єднання, тому для цієї задачі використовується протокол WebSocket. У Django підтримка WebSocket реалізується через бібліотеку Django Channels разом з ASGI-сервером Daphne [22]. Альтернативою було використання окремого WebSocket-сервера, але це ускладнило б архітектуру і вимагало б синхронізації автентифікації між двома сервісами.

Оркестрація контейнерів. Для програмного управління Docker-контейнерами використовується Docker SDK for Python – офіційна бібліотека, що надає Python-інтерфейс до Docker Engine API [23]. Вона дозволяє створювати, запускати, зупиняти та видаляти контейнери з коду Django. Розглядався також Kubernetes, який надає потужніші засоби оркестрації, проте для навчальної платформи, розрахованої на десятки одночасних користувачів, він вносить надмірну складність у розгортання та адміністрування.

База даних та кешування. Для зберігання даних про лабораторні роботи, інстанси та прогрес користувачів обрано PostgreSQL [24]. Як брокер повідомлень для Django Channels використовується Redis, який забезпечує обмін даними між WebSocket-з'єднаннями через механізм channel layers [22].

Зворотний проксі-сервер. Перед Django-застосунком працює Nginx, який роздає статичні файли, проксує HTTP-запити до Django, WebSocket-з'єднання до Daphne, а також забезпечує авторизований доступ до веб-лабораторій та ізоляцію між користувачами на мережевому рівні [25].

Фронтенд. Для інтерфейсу користувача обрано серверний рендеринг шаблонів Django у поєднанні з HTMX та Alpine.js. HTMX дозволяє оновлювати частини сторінки без написання JavaScript-коду, а Alpine.js забезпечує реактивність для невеликих інтерактивних елементів [26, 27]. Такий стек значно простіший за повноцінний SPA-фреймворк на кшталт React чи Vue, не потребує окремого API та процесу збірки, і при цьому дає зручний інтерактивний інтерфейс.

Розроблена платформа побудована як багатокомпонентна система (рис. 2), що складається з постійно працюючих інфраструктурних сервісів та динамічно створюваних контейнерів з вразливими стендами. Усі інфраструктурні компоненти описані у файлі `docker-compose.yml` і запускаються як єдине середовище.

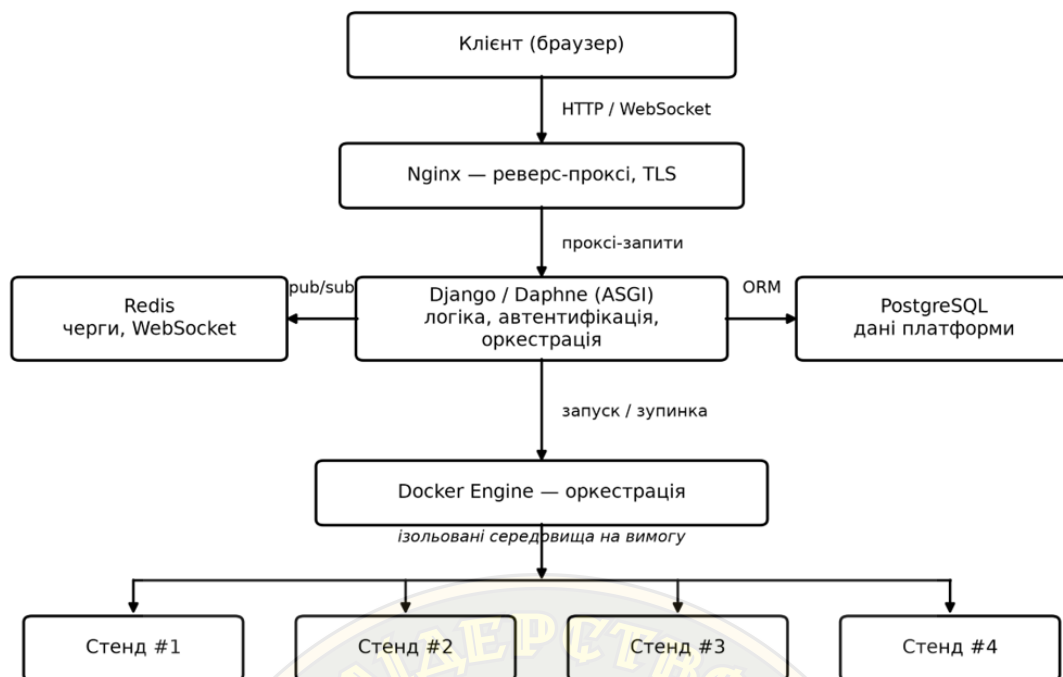


Рисунок 2 – Узагальнена архітектура платформи автоматизованого розгортання вразливих стендів

Інфраструктурне ядро платформи включає чотири сервіси. PostgreSQL відповідає за зберігання даних про лабораторні роботи, їх інстанси, користувачів та прогрес проходження. Redis використовується як брокер повідомлень для Django Channels і забезпечує роботу WebSocket-з'єднань. Веб-застосунок на Django запускається через ASGI-сервер Daphne і обробляє як звичайні HTTP-запити, так і WebSocket-з'єднання для термінальних лабораторій. Nginx працює як зворотний проксі-сервер, розподіляючи запити між компонентами системи та забезпечуючи доступ до лабораторних середовищ.

Окремо від інфраструктурних сервісів існують контейнери лабораторних стендів. Вони не описані у `docker-compose.yml`, а створюються динамічно через Docker SDK for Python у момент, коли студент запускає лабораторну роботу. Кожен такий контейнер містить вразливе середовище – веб-застосунок або Linux-систему з навмисно допущеними вразливостями. Після завершення сесії контейнер зупиняється і видаляється. Цей підхід дозволяє ефективно використовувати ресурси сервера: контейнери існують лише протягом активної сесії студента.

Взаємодія користувача з платформою відбувається за одним з двох сценаріїв залежно від типу лабораторної роботи.

Для термінальних лабораторій (наприклад, на підвищення привілеїв у Linux) використовується WebSocket-з'єднання. Всередині контейнера працює `ttyd` – легковагий інструмент, який надає доступ до терміналу через WebSocket-протокол. Коли студент відкриває лабораторію у браузері, JavaScript-клієнт встановлює WebSocket-з'єднання з Django через Nginx. Django Channels consumer отримує це з'єднання, знаходить IP-адресу відповідного контейнера і відкриває другий WebSocket до `ttyd` всередині нього. Далі consumer працює як міст, передаючи введення користувача до `ttyd` і повертаючи вивід терміналу назад у браузер. Таким чином студент отримує повноцінний термінал у веб-інтерфейсі без необхідності встановлення SSH-клієнта або VPN.

Для веб-лабораторій (наприклад, на SQL-ін'єкцію або `command injection`) використовується інший механізм. Всередині контейнера працює вразливий веб-застосунок на Flask (або інший фреймворк), який слухає на певному порті. Цей порт прокидається на хост-систему, і Nginx проксує запити студента безпосередньо до контейнера. Перед проксуванням Nginx виконує авторизаційну перевірку через механізм `auth_request`: він надсилає внутрішній

підзапит до Django, який перевіряє, чи належить контейнер на даному порті поточному автентифікованому користувачу. Якщо перевірка пройдена – запит проксується до вразливого застосунку, якщо ні – повертається помилка доступу. Вразливий веб-застосунок відображається у вбудованому iframe на сторінці лабораторії, що дозволяє студенту одночасно бачити опис завдання, працювати з вразливим сайтом та вводити знайдений флаг.

Обидва потоки мають спільну рису: студент взаємодіє з платформою виключно через браузер, а вся інфраструктура розгортання, ізоляції та проксування прихована від нього. З точки зору користувача, він натискає кнопку "Start Lab", через кілька секунд отримує робоче середовище і може починати працювати.

Модель даних платформи побудована навколо трьох основних сутностей: лабораторна робота, інстанс лабораторної роботи та профіль користувача. Така структура відокремлює опис лабораторного сценарію від його конкретного запуску для певного студента, що дозволяє одній лабораторній роботі мати довільну кількість одночасних інстансів.

Модель LabInstance представляє конкретний запуск лабораторної роботи певним користувачем. Вона пов'язана з моделями Lab та User і зберігає ідентифікатор створеного Docker-контейнера (container_id), його ім'я (container_name), порт на хост-системі (host_port) та поточний статус. Інстанс може перебувати в одному з трьох станів: running (контейнер запущений і доступний), stopped (зупинений користувачем або системою) або expired (час сесії вичерпано). Кожен інстанс має обмежений час життя – при створенні автоматично встановлюється термін дії тривалістю дві години, після якого система зупиняє контейнер і позначає інстанс як expired.

Модель UserProfile розширює стандартну модель користувача Django і зберігає накопичені бали, список розв'язаних лабораторних робіт (зв'язок many-to-many з Lab) та налаштування видимості у таблиці результатів. Профіль створюється автоматично при реєстрації нового користувача через механізм Django signals.

Ключовою проєктною вимогою платформи є простота додавання нових лабораторних стендів. Процес складається з двох кроків. Спочатку потрібно підготувати Docker-образ з вразливим середовищем: написати Dockerfile, який встановлює потрібне програмне забезпечення, створює вразливу конфігурацію та розміщує флаг. Потім адміністратор реєструє нову лабораторну роботу через стандартну адмінпанель Django, заповнюючи поля моделі Lab: вказує назву образу, тип лабораторної роботи, порт, необхідні capabilities та флаг. Після збереження лабораторна робота одразу стає доступною студентам на головній сторінці. Код платформи при цьому не змінюється – вся логіка розгортання, ізоляції та верифікації флагів працює однаково для будь-якого стенду незалежно від того, яку вразливість він моделює.

Центральним компонентом платформи є модуль оркестрації контейнерів, реалізований у файлі docker_service.py. Він відповідає за повний життєвий цикл лабораторного інстансу – від створення контейнера до його зупинки та видалення. Взаємодія з Docker Engine відбувається через Docker SDK for Python, який підключається до Docker-демона на хост-системі через прокинутий Docker socket.

Процес запуску лабораторної роботи починається, коли студент натискає кнопку "Start Lab" в інтерфейсі платформи. Django створює запис LabInstance у базі даних і передає його модулю оркестрації. Перш ніж створити новий контейнер, система перевіряє, чи є у користувача вже запущений інстанс будь-якої іншої лабораторної роботи. Якщо такий існує – він автоматично зупиняється. Це обмеження (один активний інстанс на користувача) є свідомим проєктним рішенням, яке спрощує управління ресурсами і запобігає ситуації, коли один студент займає кілька портів та споживає надмірну кількість ресурсів сервера.

Після виділення порту модуль оркестрації видаляє можливий залишковий контейнер з тим самим ім'ям (ситуація може виникнути, якщо попередній процес зупинки не завершився коректно), Ключовий фрагмент функції start_lab, що відповідає за створення контейнера з параметрами ізоляції.

Зупинка інстансу може відбуватися трьома шляхами. Студент може натиснути кнопку "Stop" в інтерфейсі – тоді Django викликає функцію stop_lab, яка зупиняє контейнер з

таймаутом 5 секунд і потім видаляє його. Другий варіант – автоматичне витіснення при запуску нової лабораторної роботи, коли система зупиняє попередній інстанс перед створенням нового. Третій варіант – закінчення терміну дії сесії. Функція `expire_old_instances` знаходить у базі всі інстанси зі статусом `running`, у яких час `expires_at` вже минув, зупиняє їх контейнери та позначає як `expired`. Ця функція може викликатися періодично або при перевірці статусу інстансу.

У всіх трьох випадках контейнер не просто зупиняється, а повністю видаляється з Docker. Це гарантує, що після завершення сесії на сервері не залишається жодних артефактів попереднього запуску, а при наступному старті тієї ж лабораторної роботи студент отримує чисте середовище.

Важливою особливістю модуля є його універсальність. Логіка оркестрації не містить жодних знань про конкретні лабораторні стенди – вона працює виключно з параметрами, описаними у моделі Lab. Завдяки цьому додавання нового стенду не потребує змін у коді оркестрації: достатньо створити Docker-образ та зареєструвати лабораторну роботу через адмінпанель.

Платформа побудована за принципом `secure by default` з можливістю точкового послаблення. Як було описано вище, кожен контейнер за замовчуванням запускається з мінімальними привілеями. Проте різні типи лабораторних робіт потребують різного рівня доступу. Наприклад, веб-лабораторна робота на SQL-ін'єкцію працює без жодних додаткових `capabilities` і з увімкненим `no-new-privileges`, оскільки вразливість знаходиться виключно на рівні застосунку. Натомість лабораторна робота на `sudo misconfiguration` потребує `CAP_SETUID`, `CAP_SETGID`, `CAP_DAC_OVERRIDE` та `CAP_DAC_READ_SEARCH`, а лабораторна робота на SUID-бінарний файл – `CAP_SETUID` і `CAP_SETGID` з вимкненим `no-new-privileges`. Адміністратор налаштовує ці параметри індивідуально для кожного стенду через адмінпанель, і послаблення стосується лише конкретної лабораторної роботи, не впливаючи на решту.

Інтерфейс платформи побудований на серверному рендерингу шаблонів Django з використанням HTMX та Alpine.js. Такий підхід дозволяє реалізувати інтерактивну взаємодію без повного перезавантаження сторінки і при цьому зберегти простоту архітектури без окремого API та фронтенд-збірки (рис. 3).

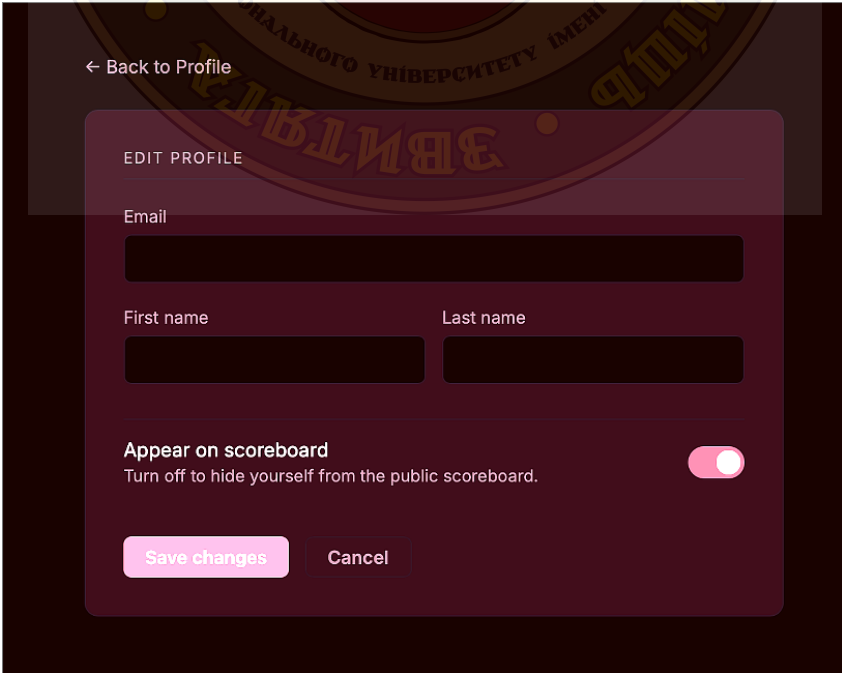


Рисунок 3 – Редагування профілю користувача та керування видимістю у таблиці результатів

Головна сторінка відображає каталог доступних лабораторних робіт, згрупованих за категоріями (рис. 3-4). Кожна картка лабораторної роботи містить назву, опис, ідентифікатор техніки MITRE ATT&CK, рівень складності та кількість балів. Якщо студент вже розв’язав лабораторну роботу, картка позначається відповідним індикатором. У верхній частині сторінки відображається статистика користувача – кількість розв’язаних лабораторних робіт та накопичені бали. Якщо у студента є активна сесія, з’являється банер з назвою запущеної лабораторної роботи, часом, що залишився, та кнопками для переходу до сесії або її зупинки.

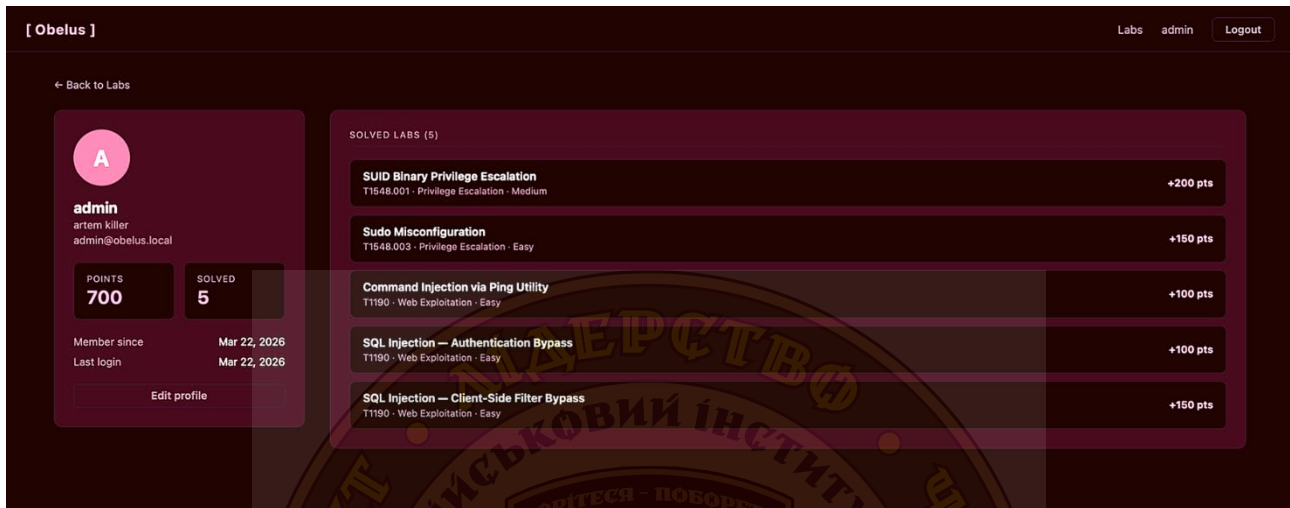


Рисунок 4 – Профіль користувача з переліком розв’язаних робіт та прив’язкою до технік MITRE ATT&CK

Сторінка лабораторної роботи має двоколонкову компоновку. Ліва панель містить опис завдання, категорію, рівень складності та форму для введення флага. Права частина займає робочу область – термінал або вбудований iframe з вразливим веб-застосунком залежно від типу лабораторної роботи. Запуск та зупинка інстансу відбуваються через HTMX-запити, які оновлюють лише панель сесії без перезавантаження всієї сторінки. Alpine.js використовується для відліку часу сесії у реальному часі та для відображення стану кнопки запуску під час створення контейнера.

Верифікація проходження побудована на механізмі флагів – унікальних рядків формату FLAG{...}, які студент має знайти в процесі експлуатації вразливості. Форма введення флагу надсилає HTMX POST-запит до Django, де введений рядок хешується алгоритмом SHA-256 і порівнюється зі збереженим хешем. Результат перевірки (правильно, неправильно або вже розв’язано) відображається одразу під формою без перезавантаження сторінки. При першому правильному введенні флагу система додає лабораторну роботу до списку розв’язаних у профілі користувача та нараховує відповідну кількість балів (рис. 5).

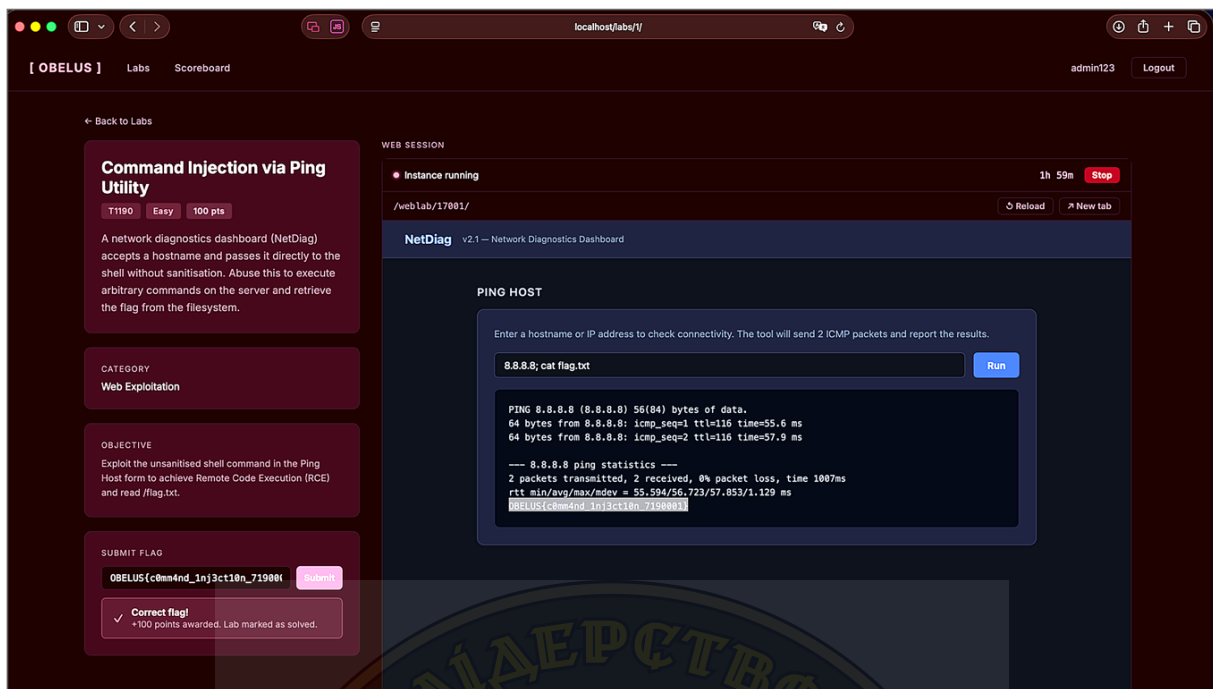


Рисунок 5 – Проходження веб-лабораторної роботи (Command Injection, T1190) з верифікацією флагу

Елементи гейміфікації включають систему балів, рівні складності та таблицю результатів (scoreboard). Кожна лабораторна робота має визначену кількість балів залежно від складності. Таблиця результатів відображає рейтинг користувачів, відсортований за кількістю балів. Участь у таблиці результатів є добровільною – кожен користувач може увімкнути або вимкнути відображення свого профілю у налаштуваннях (рис. 6).

#	USER	PROGRESS	POINTS
1	admin	5/5	700
2	admin123 (you)	2/5	300
3	mega_admin	1/5	200
4	admin1	1/5	150

Рисунок 6 – Таблиця результатів (scoreboard)

Для перевірки коректності роботи платформи було проведено тестування основних сценаріїв: запуск та зупинка лабораторних робіт, ізоляція між користувачами, дотримання ресурсних обмежень, автоматичне закінчення сесій та верифікація флагів.

Для демонстрації можливостей платформи було пройдено лабораторні роботи обох типів. У веб-лабораторії на SQL-ін'єкцію введення класичного payload ' OR 1=1-- у форму авторизації дозволило обійти перевірку пароля та отримати доступ до адміністративної панелі, де відображався флаг. У термінальній лабораторії на sudo misconfiguration команда sudo -l показала, що користувач student може запускати /usr/bin/python3 з правами root без пароля, після чого виконання sudo python3 -c 'import os; os.system("/bin/bash")' дало root-доступ і можливість прочитати флаг з /root/flag.txt. (рис. 7).

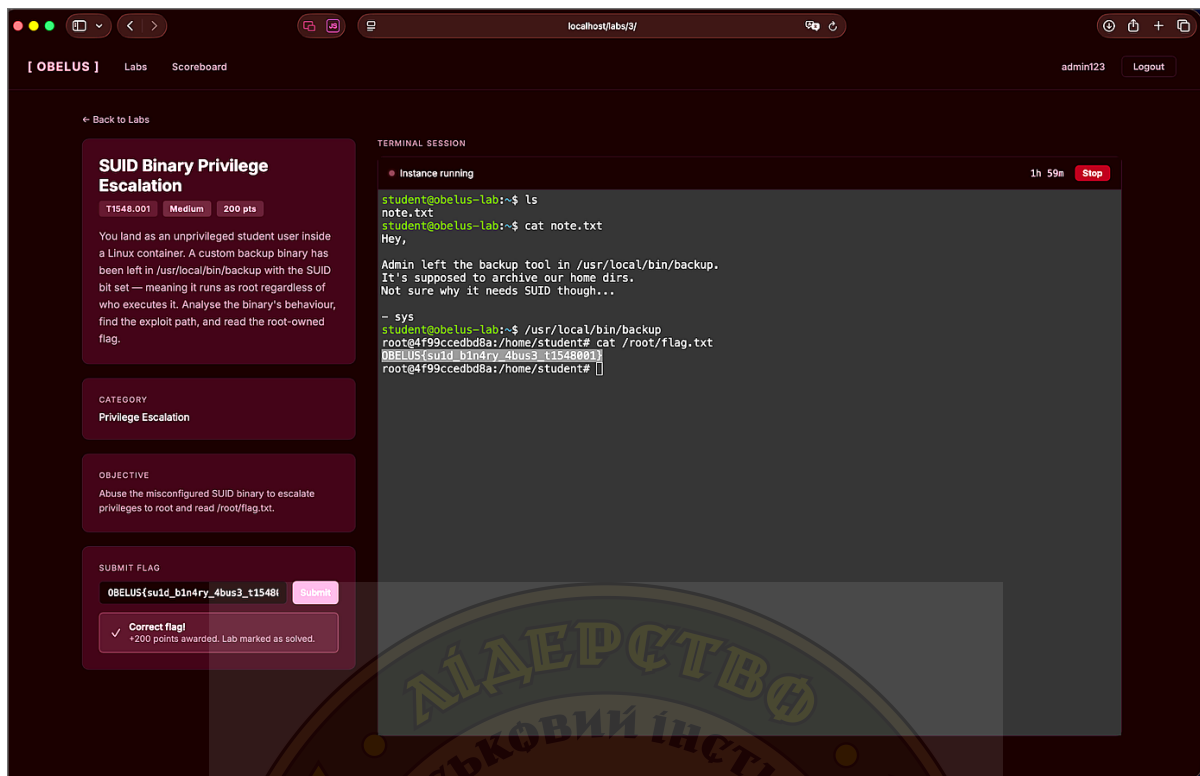


Рисунок 7 – Проходження термінальної лабораторної роботи

Дискусія і висновки. Досліджено підходи до автоматизованого розгортання вразливих стендів, що спеціально призначені для навчання кібербезпеці та успішно реалізовано платформу, що легко відтворює техніки MITRE ATT&CK в ізольованих контейнерних середовищах.

Проаналізовано структуру фреймворку MITRE ATT&CK, проведено огляд існуючих навчальних кіберполігонів та, що особливо важливо, виявлено їхні ключові обмеження. Наприклад, локальні рішення потребують від кожного студента ручного розгортання, тоді як комерційні платформи часто не надають можливості додавати власні, унікальні сценарії. Обґрунтовано вибір контейнеризації як технологічної основи та чітко окреслено технологічний стек для платформи.

У практичній частині авторами спроектовано й реалізовано платформу, яка автоматично розгортає ізольований контейнер для кожного користувача окремо. Платформа надає два типи доступу до лабораторних середовищ: термінальний, реалізований через WebSocket, та веб-доступ через Nginx-проксі, а також впроваджує модель безпеки, засновану на мінімальних привілеях за замовчуванням з можливістю точкового послаблення для певних стендів. Архітектуру платформи розроблено таким чином, що адміністратор може додавати нові лабораторні роботи, не модифікуючи при цьому вихідний код. Тому достатньо лише підготувати Docker-образ і зареєструвати відповідний стенд через адмінпанель Django.

Проведене тестування чітко підтвердило коректну роботу механізмів ізоляції між користувачами, суворе дотримання всіх ресурсних обмежень та ефективне автоматичне завершення сесій. Працездатність платформи продемонстровано на прикладі успішного проходження лабораторних робіт різних типів.

Отримані результати наочно демонструють, що поєднання контейнеризації з автоматизованою оркестрацією змінює весь підхід до створення навчальних рішень, усуваючи їхнє найголовніше обмеження – постійну потребу в ручному розгортанні окремого робочого середовища для кожного учасника. Запропонована платформа здатна розгорнути абсолютно ізольований тренувальний, а після успішного завершення сесії вона самостійно, без будь-якого втручання, звільняє абсолютно всі використані системні ресурси, що робить її досить гнучкою та ефективною.

На відміну від переважної більшості комерційних кіберполігонів, таких як популярні Hack The Box чи TryHackMe, які функціонують як закриті сервіси з чітко фіксованим, заздалегідь визначеним набором завдань, ключовою та принциповою перевагою розробленої нами платформи є її відкритість до подальшого, практично необмеженого розширення. Додавання нового навчального стенда не потребує жодних, навіть найменших, змін у ядрі системи й зводиться до порівняно простої підготовки відповідного Docker-образу та реєстрації потрібного запису в адміністративній панелі. Окрім цього, пряма й чітка прив'язка кожного стенда до конкретної техніки MITRE ATT&CK, якої так бракує більшості проаналізованих нами аналогів, дає змогу ідеально узгодити практичні завдання з формалізованою та загально визнаною моделлю дій потенційного супротивника, тим самим значно полегшуючи процес планування як індивідуальних, так і групових навчальних траєкторій.

Водночас необхідно зазначити, що проведені дослідження, має низку невід'ємних обмежень. Усі кількісні показники, що детально стосуються часу розгортання середовища та загального споживання пам'яті, були отримані в контрольованому лабораторному середовищі й, безумовно, вимагають додаткової, всебічної перевірки та валідації за умов реального промислового навантаження, коли з системою одночасно працюватимуть десятки, а то й сотні користувачів. Варто також пам'ятати, що контейнерна ізоляція, на противагу повноцінній апаратній віртуалізації, використовує спільне ядро хост-системи. Це означає, що для специфічних сценаріїв, які передбачають експлуатацію вразливостей саме на рівні ядра, є абсолютно необхідними додаткові, значно посилені механізми захисту хоста-сервера.

Щодо перспективних напрямів подальших досліджень, то актуальним завданням стає перенесення механізмів оркестрації на потужний, промисловий рівень Kubernetes. Це дозволить ефективно розподіляти та балансувати навантаження між численними серверами, забезпечуючи таким чином максимальну відмовостійкість та значну продуктивність системи в цілому. Також варто спланувати суттєве розширення вже наявної бібліотеки доступних лабораторних сценаріїв, охопивши ними додаткові, найактуальніші тактики та техніки з фреймворку MITRE ATT&CK, аби зробити процес навчання ще більш всеосяжним, глибоким та актуальним для сучасних реалій кібербезпеки. Треба визнати: не менш важливою є й інтеграція інноваційної системи автоматизованого оцінювання та виявлення типових помилок студентів, яка б базувалася на глибокому, контекстному аналізі їхніх дій безпосередньо в навчальному середовищі.

Практична цінність нашої роботи полягає, перш за все, у перспективі інтеграції платформи в освітній процес закладів вищої освіти, пропонуючи студентам унікальний та незамінний практичний досвід. Крім того, вона стане надзвичайно цінним інструментом для програм підвищення кваліфікації спеціалістів з кібербезпеки, дозволяючи їм опрацьовувати найактуальніші техніки захисту інформації в контрольованому, але реалістичному середовищі.

ЛІТЕРАТУРА

1. Yamin M. M., Katt B., Gkioulos V. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers & Security*. 2020. Vol. 88. Art. 101636. DOI: <https://doi.org/10.1016/j.cose.2019.101636>
2. 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)*. Madrid, Spain, 2017. P. 310–321. DOI: <https://doi.org/10.5220/0006428203100321>
3. Beuran R., Tang D., Pham C., Chinen K., Tan Y., Shinoda Y. Integrated framework for hands-on cybersecurity training: CyTrONE. *Computers & Security*. 2018. Vol. 78. P. 43–59. DOI: <https://doi.org/10.1016/j.cose.2018.06.001>
4. Al-Shaer R., Spring J. M., Christou E. Learning the associations of MITRE ATT&CK adversarial techniques. *2020 IEEE Conference on Communications and Network Security (CNS)*. 2020. P. 1–9. DOI: <https://doi.org/10.1109/CNS48642.2020.9162207>

5. Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*. 2014. Vol. 1, No. 3. P. 81–84. DOI: <https://doi.org/10.1109/MCC.2014.51>
6. Merkel D. Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*. 2014. No. 239. P. 2.
7. Combe T., Martin A., Di Pietro R. To Docker or not to Docker: A security perspective. *IEEE Cloud Computing*. 2016. Vol. 3, No. 5. P. 54–62. DOI: <https://doi.org/10.1109/MCC.2016.100>
8. Sultan S., Ahmad I., Dimitriou T. Container security: Issues, challenges, and the road ahead. *IEEE Access*. 2019. Vol. 7. P. 52976–52996. DOI: <https://doi.org/10.1109/ACCESS.2019.2911732>
9. Kavak H., Padilla J. J., Vernon-Bido D., Diallo S. Y., Gore R., Shetty S. Simulation for cybersecurity: State of the art and future directions. *Journal of Cybersecurity*. 2021. Vol. 7, No. 1. Art. tyab005. DOI: <https://doi.org/10.1093/cybsec/tyab005>
10. MITRE. MITRE ATT&CK®. URL: <https://attack.mitre.org/>
11. MITRE. Getting started with ATT&CK. URL: <https://attack.mitre.org/resources/>
12. IBM. What is the MITRE ATT&CK framework? URL: <https://www.ibm.com/think/topics/mitre-attack>
13. VulnHub. VulnHub. URL: <https://www.vulnhub.com/>
14. Rapid7. Metasploitable 2 exploitability guide. URL: <https://docs.rapid7.com/metasploit/metasploitable-2/>
15. TryHackMe. URL: <https://tryhackme.com/>
16. Hack The Box. Hack The Box. URL: <https://www.hackthebox.com/>
17. Namespaces. Linux man-pages. URL: <https://man7.org/linux/man-pages/man7/namespaces.7.html>
18. Docker. Docker overview. URL: <https://docs.docker.com/get-started/overview/>
19. Capabilities. Linux man-pages. URL: <https://man7.org/linux/man-pages/man7/capabilities.7.html>
20. Docker. Docker security. URL: <https://docs.docker.com/engine/security/>
21. Django Software Foundation. Django documentation (Version 5.1). URL: <https://docs.djangoproject.com/en/5.1/>
22. Django Channels. Django Channels documentation. URL: <https://channels.readthedocs.io/en/stable/>
23. Docker SDK for Python. Docker SDK for Python documentation. URL: <https://docker-py.readthedocs.io/en/stable/>
24. PostgreSQL Global Development Group. PostgreSQL documentation. URL: <https://www.postgresql.org/docs/>
25. Nginx. Module ngx_http_auth_request_module. URL: https://nginx.org/en/docs/http/ngx_http_auth_request_module.html
26. htmx. htmx documentation. URL: <https://htmx.org/docs/>
27. Alpine.js. Alpine.js documentation. URL: <https://alpinejs.dev/>

REFERENCES

1. Yamin, M.M., Katt, B. and Gkioulos, V. (2020) 'Cyber ranges and security testbeds: Scenarios, functions, tools and architecture', *Computers & Security*, 88, 101636. Available at: <https://doi.org/10.1016/j.cose.2019.101636>
2. Vykopal, J., Ošlejšek, R., Čeleda, P., Vizváry, M. and Tovarňák, D. (2017) 'KYPO Cyber Range: Design and use cases', in *Proceedings of the 12th International Conference on Software Technologies (ICSOFT)*, pp. 310–321. Available at: <https://doi.org/10.5220/0006428203100321>
3. Beuran, R., Tang, D., Pham, C., Chinen, K., Tan, Y. and Shinoda, Y. (2018) 'Integrated framework for hands-on cybersecurity training: CyTrONE', *Computers & Security*, 78, pp. 43–59. Available at: <https://doi.org/10.1016/j.cose.2018.06.001>
4. Al-Shaer, R., Spring, J.M. and Christou, E. (2020) 'Learning the associations of MITRE ATT&CK adversarial techniques', in *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9. Available at: <https://doi.org/10.1109/CNS48642.2020.9162207>
5. Bernstein, D. (2014) 'Containers and cloud: From LXC to Docker to Kubernetes', *IEEE Cloud Computing*, 1(3), pp. 81–84. Available at: <https://doi.org/10.1109/MCC.2014.51>
6. Merkel, D. (2014) 'Docker: Lightweight Linux containers for consistent development and deployment', *Linux Journal*, 2014(239), p. 2.
7. Combe, T., Martin, A. and Di Pietro, R. (2016) 'To Docker or not to Docker: A security perspective', *IEEE Cloud Computing*, 3(5), pp. 54–62. Available at: <https://doi.org/10.1109/MCC.2016.100>

8. Sultan, S., Ahmad, I. and Dimitriou, T. (2019) 'Container security: Issues, challenges, and the road ahead', *IEEE Access*, 7, pp. 52976–52996. Available at: <https://doi.org/10.1109/ACCESS.2019.2911732>
9. Kavak, H., Padilla, J.J., Vernon-Bido, D., Diallo, S.Y., Gore, R. and Shetty, S. (2021) 'Simulation for cybersecurity: State of the art and future directions', *Journal of Cybersecurity*, 7(1), tyab005. Available at: <https://doi.org/10.1093/cybsec/tyab005>
10. MITRE (2024) *MITRE ATT&CK®*. Available at: <https://attack.mitre.org/>
11. MITRE (2023) *Getting started with ATT&CK*. Available at: <https://attack.mitre.org/resources/>
12. IBM (2024) *What is the MITRE ATT&CK framework?* Available at: <https://www.ibm.com/think/topics/mitre-attack>
13. VulnHub (2024) *VulnHub*. Available at: <https://www.vulnhub.com/>
14. Rapid7 (2024) *Metasploitable 2 exploitability guide*. Available at: <https://docs.rapid7.com/metasploit/metasploitable-2/>
15. TryHackMe (2024) *TryHackMe*. Available at: <https://tryhackme.com/>
16. Hack The Box (2024) *Hack The Box*. Available at: <https://www.hackthebox.com/>
17. Namespaces (7) (2023) *Linux man-pages*. Available at: <https://man7.org/linux/man-pages/man7/namespaces.7.html>
18. Docker (2024) *Docker overview*. Available at: <https://docs.docker.com/get-started/overview/>
19. Capabilities (7) (2023) *Linux man-pages*. Available at: <https://man7.org/linux/man-pages/man7/capabilities.7.html>
20. Docker (2024) *Docker security*. Available at: <https://docs.docker.com/engine/security/>
21. Django Software Foundation (2024) *Django documentation (Version 5.1)*. Available at: <https://docs.djangoproject.com/en/5.1/>
22. Django Channels (2024) *Django Channels documentation*. Available at: <https://channels.readthedocs.io/en/stable/>
23. Docker SDK for Python (2024) *Docker SDK for Python documentation*. Available at: <https://docker-py.readthedocs.io/en/stable/>
24. PostgreSQL Global Development Group (2024) *PostgreSQL documentation*. Available at: <https://www.postgresql.org/docs/>
25. Nginx (2024) *Module ngx_http_auth_request_module*. Available at: https://nginx.org/en/docs/http/ngx_http_auth_request_module.html
26. htmx (2024) *htmx documentation*. Available at: <https://htmx.org/docs/>
27. Alpine.js (2024) *Alpine.js documentation*. Available at: <https://alpinejs.dev/>

Babych T., PhD Sashnova M., Radziievska O., Dyakonenko D.

AUTOMATED DEPLOYMENT OF VULNERABLE TRAINING ENVIRONMENTS USING THE LEAST-PRIVILEGE MODEL

The growing number and complexity of cyberattacks impose increasingly stringent requirements on the practical training of cybersecurity professionals. At the same time, traditional lecture-based instruction is, unfortunately, often unable to provide a sufficient level of applied skills. Training-oriented cyber ranges address this problem to a certain extent, but existing solutions rarely offer automated deployment of a separate, fully isolated environment for each participant. Moreover, they frequently fail to align training scenarios with formalized descriptions of adversary techniques. The main goal of our study was therefore to substantiate, design, and ultimately implement in software a platform capable of automated deployment of vulnerable testbeds that accurately reproduce documented techniques from the MITRE ATT&CK framework, within a controlled multi-user space, while ensuring complete isolation between users through containerization technologies.

A variety of methods were employed in the course of this work, including a systematic analysis of existing cyber ranges and vulnerable testbeds, as well as a thorough comparative analysis of isolation mechanisms at both the container and virtual machine levels. In addition, object-oriented software architecture design methods were applied, together with a carefully elaborated least-privilege model for container configuration. The platform is built on a multi-tier architecture (Nginx, Django/Daphne, PostgreSQL, Redis) with dynamic orchestration of Docker containers, managed through a dedicated application programming interface.

A platform was designed and implemented that automatically creates and, subsequently, destroys isolated vulnerable testbeds on demand. It is capable of mapping each testbed to specific MITRE ATT&CK

techniques and, importantly, supports the addition of entirely new laboratory scenarios without requiring changes to its core. A security configuration for containers was proposed, based on the principle of dropping unnecessary privileges, while preserving the ability to selectively relax these restrictions in order to correctly reproduce certain techniques. A comparison of the chosen container-based approach with traditional virtualization convincingly demonstrated substantially shorter deployment times and significantly lower resource consumption.

Combining containerization, automated orchestration, and scenario integration directly with the MITRE ATT&CK framework enables the creation of a scalable and, most importantly, reproducible environment suitable for practical cybersecurity training. The practical value of the platform lies in the fact that it can be readily integrated both into the educational process of higher education institutions and into professional development systems for information security specialists.

Keywords: cybersecurity; MITRE ATT&CK; cyber range; vulnerable environment; containerization; Docker; environment isolation; modeling.

