Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет Інфокомунікацій
(повна назва)
Кафедра Інформаційно-вимірювальні технології
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
Дослідження методів тестування безпеки веб-додатків та впровадження інструментів для виявлення вразливостей
(тема)

Виконав:
здобувач 2 року навчання,
групи ЗЯм-23-1
Панасенко Д.О.
(прізвище, ініціали)

Спеціальність 175 Інформаційно-вимірювальні технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма забезпечення якості
(повна назва освітньої програми)

Керівник Єгоров А.Б.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри Захаров І.П.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-вимірювальні технології

Рівень вищої освіти другий (магістерський)

Спеціальність 175 Інформаційно-вимірювальні технології
(код і повна назва)

Тип програми освітно-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма забезпечення якості
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Панасенко Д.О.

(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів тестування безпеки веб-додатків та впровадження інструментів для виявлення вразливостей
затверджена наказом університету від 12 11 2024 р. № 1202Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 10 01 2025 р.
3. Вихідні дані до роботи
3.1 Типові загрози та види вразливостей: SQL-ін'єкція, недоліки автентифікації (Broken Authentication), недоліки контролю доступу (Broken Access Control), міжсайтовий скриптинг (Cross-Site Scritping, XSS), підробка міжсайтових запитів (Cross-Site Request Frogery, CSRF);
3.2 Методи тестування: статичний аналіз, динамічний аналіз, тестування методом проникнення
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Основи безпеки веб-додатків;
4.2 Методи та інструменти тестування безпеки веб-додатків;
4.3 Практична реалізація тестування веб-додатків;
4.4 Аналіз отриманих даних
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
5.1 Тема дослідження;

5.2 Мета та актуальність;
5.3 Основні поняття безпеки (конфіденційність, цілісність, доступність);
5.4 Типові загрози (SQL-ін’єкція, недоліки автентифікації, недоліки контролю доступу, міжсайтовий скриптинг, підробка міжсайтових запитів);
5.5 Методи тестування безпеки (статичний аналіз, динамічний аналіз, тестування методом проникнення);
5.6 Практичне тестування (SonarQube, OWASP ZAP, Metasploit);
5.7 Аналіз результатів;
5.8 Висновки
6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основний	Проф. Єгоров А.Б.		10.01.20255

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів ро-	Примітка
1	Огляд літератури	15.11.2024	Вик
2	Формування завдань та дослідження	17.11.2024	Вик
3	Розробка основної частини тексту	20.12.2024	Вик
4	Формування висновків	27.12.2024	Вик
5	Оформлення тексту	30.12.2024	Вик
6	Побудова презентації	30.12.2024	Вик
7	Представлення тексту виступу	05.01.2025	Вик
8	Подання роботи на перевірку	10.01.2025	Вик

Дата видачі завдання 30 11 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ Проф. Єгоров А.Б.
(підпис) (посада, прізвище, ініціали)

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка: 59 с., 8 табл., 35 рис.
ВЕБ-ДОДАТКИ, ВРАЗЛИВОСТІ, БЕЗПЕКА, МЕТОДИ, ІНСТРУМЕНТИ

Об'єктом дослідження є процес забезпечення інформаційної безпеки веб-додатків за допомогою тестування на вразливості.

Предметом дослідження є методи тестування безпеки веб-додатків та інструменти, що дозволяють виявляти потенційні вразливості в коді та архітектурі веб-додатків, зокрема, такі як статичний та динамічний аналіз, а також тестування методом проникнення.

Методи дослідження включають аналіз існуючих методів тестування, порівняння різних інструментів безпеки для виявлення вразливостей, впровадження практичних рішень для їх використання та оцінку ефективності методів для забезпечення безпеки.

Результатом кваліфікаційної роботи є впровадження та тестування інструментів, що дозволяють ефективно виявляти та усувати вразливості у веб-додатках. В рамках роботи було проведено:

- дослідження та аналіз загроз безпеці для веб-додатків;
- порівняння ефективності сучасних інструментів для тестування безпеки;
- практичне застосування методів тестування безпеки на вибраному веб-додатку.

Ці результати спрямовані на покращення процесу забезпечення безпеки у веб-розробці за рахунок комплексного підходу до аналізу та усунення вразливостей.

ABSTRACT

Explanatory note: 59 p., 8 tables, 35 figures.
WEB APPLICATIONS, VULNERABILITIES, SECURITY, METHODS, TOOLS

The object of research is the process of ensuring the information security of web applications through vulnerability testing.

The subject of the study is web application security testing methods and tools that allow detecting potential vulnerabilities in the code and architecture of web applications, in particular, such as static and dynamic analysis, as well as penetration testing.

The research methods include analyzing existing testing methods, comparing various security tools to identify vulnerabilities, implementing practical solutions for their use, and evaluating the effectiveness of security methods.

The result of the qualification work is the implementation and testing of tools that allow you to effectively identify and eliminate vulnerabilities in web applications. The work included:

- research and analysis of security threats to web applications;
- comparison of the effectiveness of modern security testing tools;
- practical application of security testing methods on a selected web application.

These results are aimed at improving the security process in web development through an integrated approach to analyzing and eliminating security threats.

ВСТУП

У сучасному світі веб-додатки широко використовуються в різних сферах - від електронної комерції до державних послуг - і містять значні обсяги конфіденційних даних користувачів. Безпека таких додатків є критично важливою, оскільки навіть невеликі вразливості можуть мати серйозні наслідки, такі як крадіжка персональних даних, шахрайство і навіть порушення роботи організації. Тому важливість дослідження методів тестування безпеки веб-додатків стрімко зростає.

Веб-додатки можуть містити різні вразливості, які потребують ефективних методів тестування та інструментів для їх виявлення та усунення. Сучасні методи тестування безпеки включають статичний і динамічний аналіз коду, а також тестування на проникнення, яке імітує атаку на додаток для виявлення вразливостей. Крім того, існують різні автоматизовані інструменти, які значно полегшують і прискорюють процес виявлення вразливостей, дозволяючи розробникам виявляти слабкі місця на ранніх стадіях розробки.

Метою даного дослідження є аналіз способів тестування безпеки веб-додатків та застосування інструментів для виявлення вразливостей з метою підвищення безпеки всієї інформаційної системи. Використання комплексного підходу до тестування, що включає статичний і динамічний аналіз, а також автоматизовані інструменти, може знизити ризик атак і підвищити стабільність роботи веб-додатків. В результаті дослідження будуть розроблені рекомендації щодо впровадження підходів до тестування безпеки та отримано практичний досвід використання різних інструментів для виявлення та усунення вразливостей. Використання таких інструментів може сприяти підвищенню рівня безпеки веб-додатків, підвищенню довіри користувачів до інформаційних систем та зменшенню потенційних загроз і втрат.

1 ОСНОВИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

1.1 Основні поняття інформаційоної безпеки

Інформаційна безпека (ІБ) є важливим аспектом забезпечення стабільної роботи інформаційних систем та захисту даних від зовнішніх і внутрішніх загроз. Зі збільшенням кількості кібератак та розвитком цифрових технологій забезпечення ІБ стало пріоритетом для організацій. ІБ охоплює комплекс заходів, процесів та політик, спрямованих на захист інформаційних активів від несанкціонованого доступу, модифікації та погіршення доступності. Конфіденційність, цілісність та доступність є основними поняттями ІБ та визначають три фундаментальні принципи.

Конфіденційність - це обмеження доступу до інформації лише тими користувачами або системами, які мають відповідні повноваження. Основними методами забезпечення конфіденційності є шифрування даних та контроль доступу; шифрування на основі алгоритму Advanced Encryption Standard (AES) забезпечує високий рівень захисту даних і широко використовується в сучасних веб-додатках (Rescorla, 2018). Крім того, контроль доступу досягається завдяки використанню багаторівневих систем автентифікації, таких як паролі, сертифікати та біометрія.

Цілісність гарантує, що інформація не змінюється і захищена від випадкових або навмисних змін. Для цього використовуються методи хешування (наприклад, алгоритм SHA-256) для створення унікального хеш-коду для даних. Зміни в даних змінюють хеш-код, що дозволяє швидко виявити втручання (Stallings, 2017). Контроль версій - ще один важливий компонент, який допомагає відстежувати зміни в даних і відновлювати їх у разі порушення цілісності. Принцип доступності означає, що інформація має бути доступною тоді, коли вона потрібна. Доступність досягається завдяки використанню балансування навантаження, резервування системи та планів аварійного відновлення. Такі технології,

як хмарні обчислення, забезпечують високий рівень доступності за рахунок розподілених обчислень і зберігання даних. Розширені аспекти інформаційної безпеки аутентифікація та авторизація. Аутентифікація використовує паролі, одноразові коди, сертифікати та біометричні дані для перевірки особи користувача.

Багатофакторна автентифікація (MFA) підвищує безпеку, вимагаючи поєднання декількох факторів, таких як паролі та біометричні дані. Авторизація контролює доступ до ресурсів, гарантуючи, що користувачі можуть виконувати лише ті дії, на які вони уповноважені. Просунуті системи авторизації використовують моделі контролю доступу, такі як моделі контролю доступу на основі ролей (RBAC), щоб забезпечити гнучке управління правами доступу та підвищити рівень безпеки. Журналювання та моніторинг безпеки. Системи реєстрації дозволяють відстежувати поведінку користувачів і швидко виявляти підозрілі дії.

Моніторинг у реальному часі за допомогою інструментів штучного інтелекту виявляє аномалії та потенційні атаки і забезпечує швидке реагування. Управління вразливостями та оновлення. Регулярний аудит та оновлення програмного забезпечення мають важливе значення для підтримки безпеки. Системи управління вразливостями забезпечують своєчасне виявлення та усунення потенційних вразливостей. Сюди також входить встановлення оновлень безпеки, що знижує ризик успішних атак.

Практичне значення ІБ у веб-додатках ІБ має вирішальне значення для веб-додатків, які обробляють особисту та фінансову інформацію користувачів. Захист таких додатків вимагає не тільки впровадження технічних заходів, а й дотримання політики безпеки відповідно до таких стандартів, як ISO/IEC 27001 та NIST Cybersecurity Framework. Також важливими є регулярне навчання персоналу та впровадження протоколів безпеки для мінімізації ризиків, пов'язаних з людським фактором. Інтегруючи заходи безпеки на всіх етапах життєвого циклу розробки, компанії можуть значно знизити ймовірність витоку даних, втрати конфіденційної інформації та інших кібератак.

1.2 Типові загрози безпеці веб-додатків та види вразливостей

У сучасному світі веб-додатки стали невід'ємною частиною діяльності підприємств, державних установ та індивідуальних користувачів, які обробляють великі обсяги інформації, в тому числі конфіденційні дані. Однак ці додатки піддаються впливу низки загроз, які становлять серйозні ризики для безпеки та стабільності їхньої діяльності. Глибоке розуміння типових загроз може допомогти розробити ефективні стратегії захисту та запобігти можливим атакам.

SQL-ін’єкція

SQL-ін'єкція (SQLi) - це уразливість веб-безпеки, яка дозволяє зловмиснику втручатися в запити, які додаток робить до своєї бази даних. Це може дозволити зловмиснику переглянути дані, які він зазвичай не може отримати. Це можуть бути дані, які належать іншим користувачам, або будь-які інші дані, до яких програма має доступ. У багатьох випадках зловмисник може змінити або видалити ці дані, спричиняючи стійкі зміни у вмісті або поведінці програми.

У деяких ситуаціях зловмисник може розширити атаку SQL-ін'єкцій, щоб скомпрометувати базовий сервер або іншу внутрішню інфраструктуру. Це також може дозволити їм виконувати атаки на відмову в обслуговуванні.

До чого призводить успішна атака SQL-ін'єкція.

Успішна SQL-атака може призвести до несанкціонованого доступу до конфіденційних даних, таких як:

- паролі;
- дані кредитних карток;
- особиста інформація користувача.

Протягом багатьох років SQL-ін'єкції використовувались у багатьох гучних витоках даних. Вони завдали шкоди репутації та призвели до штрафів від регуляторних органів. У деяких випадках зловмисник може отримати постійний бекдор в системи організації, що призводить до довготривалої компрометації, яка може залишатися непоміченою протягом тривалого періоду.

Як виявити SQL ін'єкційні уразливості.

Можна виявити SQL ін'єкції вручну, використовуючи систематичний набір тестів для кожної точки входу в додаток. Для цього, як правило, потрібно ввести:

- одинарні лапки ' що дає змогу шукати помилки або інші аномалії.
- деякий специфічний для SQL синтаксис, який обчислює базове (початкове) значення точки входу та інше значення, і шукає систематичні відмінності у відповідях програми;
- булеві умови, такі як OR і OR 1=2, і шукають відмінності у відповідях програми;
- корисне навантаження, призначене для ініціювання часових затримок при виконанні SQL-запиту, і пошук відмінностей у часі, необхідному для відповіді;
- OAST, призначені для запуску позасмугової мережевої взаємодії при виконанні в SQL-запиті та моніторингу будь-якої взаємодії, що виникла в результаті.

Крім того, ви можете швидко і надійно знайти більшість SQL ін'єкцій за допомогою Burp Scanner.

SQL-ін'єкції в різних частинах запиту.

Більшість SQL-уразливостей виникають в операторі WHERE запиту SELECT. Більшість досвідчених тестувальників знайомі з цим типом SQL ін'єкцій.

Однак, уразливості SQL ін'єкцій можуть виникати в будь-якому місці запиту, а також в різних типах запитів. Ось деякі інші поширені місця, де може виникнути SQL-ін'єкція:

- в операторах UPDATE, в оновлюваних значеннях або в реченні WHERE;
- в операторах INSERT – у значеннях, що вставляються;
- в операторах SELECT, в назві таблиці або стовпця;
- в операторах SELECT в реченні ORDER BY.

Приклади SQL ін'єкцій

Існує багато уразливостей, атак та методів SQL ін'єкцій, які виникають в різних ситуаціях.

Отримання прихованих даних

Це дії, при яких можна змінити SQL-запит, щоб повернути додаткові результати. Зловмисник змінює SQL-запит, щоб повернути додаткову інформацію, яка зазвичай не відображається. Наглядно можемо побачити це на рис 1.1

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

Рисунок 1.1 – Приклад SQL ін’єкції

Результатом є те, що усі записи таблиці users повертаються, оскільки '1'='1' завжди істинне.

Підрив логіки програми

Можливість зміни запиту, щоб втрутитися в логіку програми. Зловмисник змінює запит, щоб обійти автентифікацію. Наглядно можемо побачити це на рис 1.2

```
SELECT * FROM users WHERE username = 'admin' -- AND password = 'password123';
```

Рисунок 1.2 – Приклад підрива логіки програми

Результатом цієї ін’єкції є те, що у запиті перевірка пароля виключається, і автентифікація проходить успішно.

UNION-атаки

Цей спосіб дозволяє отримати дані з різних таблиць бази даних.

Сліпа SQL-ін'єкція, коли результати контрольованого вами запиту не повертаються у відповідях програми. На рис 1.3 можемо побачити запит з уразливістю.

```sql
SELECT id, username FROM users WHERE username = 'guest';
```

Рисунок 1.3 – SQL запит

Надалі, зловмисник вводить ін’єкцію, яку ми можемо побачити на рис. 1.4

```sql
' UNION SELECT credit_card_number, cvv FROM credit_cards --
```

Рисунок 1.4 – Приклад ін’єкції

В результаті зловмисник отримує дані з таблиці credit _cards, які могли б бути конфіденційними.

Сліпа SQL ін’єкція

Дії зловмисника, при яких він вводить шкідливий код для перевірки логіки запиту, навіть якщо відповідь не містить даних. Наглядний приклад можна побачити на рис. 1.5

```sql
SELECT * FROM users WHERE username = 'guest' AND 1=1 -- AND password = 'password123';
```

Рисунок 1.5 – Приклад сліпої SQL ін’єкції

В результаті запит виконується успішно, а зловмисник отримує інформацію, що вразливість існує.

Недоліки автентифікації (Broken Authentication)

Порушення автентифікації стосується будь-яких вразливостей, коли зловмисники видають себе за справжніх користувачів додатків. Іншими словами, автентифікація порушується, коли зловмисники можуть видавати себе за користувачів, компрометуючи паролі, токени сеансів, інформацію про облікові записи користувачів та інші дані. Основними причинами порушення автентифікації є погано реалізоване управління сеансами і слабкі політики паролів або інші слабкі заходи безпеки, що призводять до викрадення або компрометації облікових даних.

Погано реалізоване керування сеансами

Перш ніж розглянути, як погано реалізоване керування сеансами призводить до порушень автентифікації, необхідно пояснити кілька термінів.

Коли користувачі переглядають більшість веб-додатків, їм потрібно отримати доступ до онлайн-облікових записів. У більшості випадків вони входять, використовуючи ім'я користувача та пароль. Як тільки вони отримують доступ до онлайн-акаунту, додаток присвоює їм унікальний ідентифікатор сеансу, який діє як ідентифікаційний ключ. Наглядно процес можна побачити на рис. 1.6

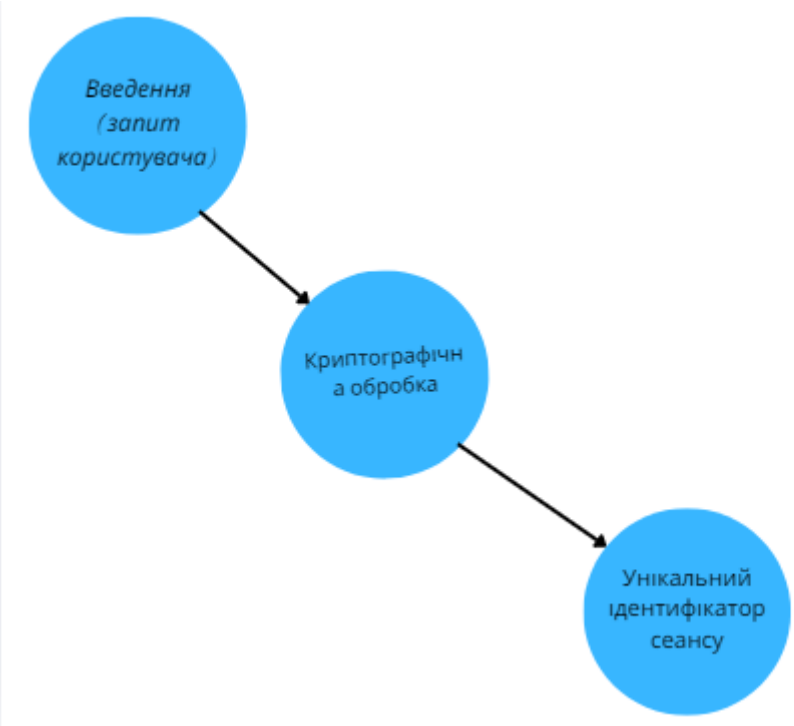

Рисунок 1.6 – Схематичне зображення процесу створення унікального ідентифікатору сеансу

Цей процес створює сесію - серію запитів, пов'язаних з користувачем, які відстежуються разом. Сесії використовуються для зберігання інформації про стан вашої взаємодії з додатком. Ідентифікатори сесії зазвичай існують у вигляді файлів cookie або заголовка авторизації.

Наприклад, сесія створюється, коли користувачі входять на веб-сайт. Ця сесія відстежує всі запити, які користувачі роблять під час входу. Як тільки користувач виходить з системи або закінчується час сеансу, сеанс знищується.

Керування сесіями пов'язане з тим, як користувачі вашого веб-сайту і додатків визначають параметри сесії. Йдеться про такі аспекти, як тривалість сеансу, перш ніж користувач вийде з системи, або про те, як ви видаєте ідентифікатори сеансу. Це також пов'язано з безпекою даних сесій при підключенні до IP-адрес різних користувачів.

Потрібно звернути увагу, що користувачі створюють сеанс з додатком або веб-сайтом кожного разу, коли вони входять в систему як користувачі. Цей сеанс є автентифікованим, тобто користувачам потрібні облікові дані для входу в сеанс.

З цієї причини OWASP визнає, що «ідентифікатор сеансу автентифікованого сеансу тимчасово еквівалентний найнадійнішому методу автентифікації, що використовується програмою». Методом автентифікації може бути ім'я користувача та пароль, одноразові паролі (OTP) або біометрія.

Існують різні типи атак на порушення автентифікації, пов'язані з керуванням сеансами. До них відносяться:

- викрадення сесії;
- перезапис URL-адреси ідентифікатора сеансу;
- фіксація сесії;
- вразлива політика паролів та викрадені/компрометовані облікові дані;

Викрадення сесії

Цей тип атак на управління сеансами відбувається, коли зловмисник захоплює сеанс користувача, викрадаючи його ідентифікатор сеансу. Зловмисник може зробити це кількома способами, наприклад, перехопивши ідентифікатор сеансу, що передається між користувачем і сервером.

Зловмисник також може скористатися ситуацією, коли користувач не виходить з системи після використання програми і залишає свій пристрій. Тоді кіберзлочинці зможуть отримати доступ до пристрою і використати ту саму сесію, яка все ще активна.

Перезапис URL-адреси ідентифікатора сеансу

Перезапис URL-адреси за ідентифікатором сеансу відбувається, коли ідентифікатор сеансу користувача відображається в URL-адресі веб-сайту. Будь-хто, хто отримує доступ до URL-адреси через незахищену мережу Wi-Fi, може продовжити сеанс.

Атака зазвичай відбувається, коли ідентифікатор сеансу вставляється в URL-адресу замість того, щоб зберігатися в файлі cookie. Користувачі можуть ненавмисно поділитися своїм ідентифікатором сеансу, коли надсилають посилання іншим людям. Люди з посиланнями можуть видавати себе за справжніх користувачів. Цей тип атаки поширений в додатках, які використовують параметри URL для зберігання ідентифікаторів сеансу.

Фіксація сесії

Ця атака відбувається, коли веб-додаток не генерує новий ідентифікатор сеансу після входу користувача в систему. У цьому випадку додаток видає користувачам однакові ідентифікатори до і після аутентифікації. Це також може статися, коли додаток генерує статичні або легко вгадувані ідентифікатори сеансу.

Вразлива політика паролів та викрадені/компрометовані облікові дані

Кіберзлочинці також можуть скомпрометувати ваш процес автентифікації, якщо у ваших додатках немає суворих політик щодо паролів. Ваші користувачі можуть бути схильні обирати паролі, які легко вгадуються і які кіберзлочинці можуть використовувати для доступу до їхніх облікових записів.

Ось деякі з атак, пов'язаних з викраденими або скомпрометованими обліковими даними:

- наповнення облікових даних;
- розпилення паролів;

– фішингові атаки.

Наповнення облікових даних

Наповнення облікових даних передбачає автоматичне введення викрадених пар імен користувачів і паролів у форми входу на веб-сайти. Зловмисники отримують списки скомпрометованих облікових даних користувачів і використовують ботів для автоматичного входу в різні системи. Це базується на припущенні, що багато користувачів повторно використовують логіни та паролі, які легко вгадати або які були скомпрометовані.

Якщо стався витік даних, надання викрадених облікових даних на інших сайтах полегшить зловмисникам компрометацію інших облікових записів. Зловмисники можуть продати викрадені облікові дані або віддати їх своїм колегам-кіберзлочинцям. Це означає, що більше хакерів намагатимуться використати облікові дані ваших користувачів у різних акаунтах, що збільшує ризик успішних атак.

Розпилення паролів

У цьому типі атаки кіберзлочинці намагаються вгадати паролі багатьох облікових записів, використовуючи поширені паролі. Прикладами таких паролів є 123456, лайливі слова, спортивні назви та термін «пароль». Насправді, 2,5 мільйона людей все ще використовують «123456» в якості пароля. Зазвичай зловмисники націлені на великий список користувачів, а не намагаються зламати один обліковий запис протягом якогось одного періоду.

Проблема з такими типами атак полягає в тому, що вони легко залишаються непоміченими. Це відбувається тому, що більшість організацій не відстежують невдалі спроби входу.

Фішингові атаки

Фішингові атаки відбуваються, коли кіберзлочинці надсилають шкідливі електронні листи, які обманом змушують користувачів розкрити свої облікові дані. Вони також можуть використовувати цей метод для встановлення шкідливого програмного забезпечення на пристрій жертви або перенаправлення на фальшивий веб-сайт.

Фішингові атаки можуть викрити облікові дані користувачів, які потім можуть бути використані для доступу до їхніх акаунтів на інших веб-сайтах.

Атаки можуть бути широкими, спрямованими на всіх користувачів за допомогою одного шахрайського електронного листа, або фішинговими атаками, спрямованими на конкретну особу. Останній тип атак є поширеним, оскільки зловмисникам легко маніпулювати емоціями користувача на основі доступної особистої інформації.

Недоліки контролю доступу (Broken Access Control)

Недоліки контролю доступу – це обмеження доступу до додатків і даних. Організації контролюють доступ шляхом аутентифікації запитів на з'єднання та авторизації користувачів для доступу до необхідної функціональності. Користувачі мають доступ лише до тих ресурсів, які їм потрібні. Всі інші дані та додатки обмежуються за принципом найменших привілеїв.

Порушення контролю доступу відбувається, коли цей процес скомпрометований. Системи можуть не змогти безпечно автентифікувати користувачів. Неправильно автентифіковані користувачі також можуть підвищувати привілеї, отримуючи доступ до ресурсів, які повинні бути закриті.

Багато витоків даних починаються з порушення контролю доступу, що призводить до шкоди репутації, втрати бізнес-даних і величезних штрафних санкцій з боку регуляторних органів.

Проста помилка в коді на веб-сторінці може відкрити двері для скриптових атак. З невеликими знаннями зловмисники можуть видавати себе за законних користувачів, отримуючи контроль над покинутими обліковими записами, які більше не повинні бути активними. Коли це відбувається, зловмисникам не потрібно багато часу, щоб отримати доступ до фінансових або персональних даних.

Перш ніж зрозуміти, як відбувається злам систем контролю доступу, необхідно розібратися з трьома основними типами систем контролю доступу:

- вертикальний контроль доступу;
- горизонтальний контроль доступу;

– контекстно-залежний контроль доступу.

Вертикальний контроль доступу обмежує доступ до програм або даних для різних типів користувачів. Доступ користувачів залежить від їхньої ролі або класифікації. Наприклад, користувач-адміністратор може додавати, видаляти або змінювати профілі користувачів.

Звичайні працівники не мають повноважень для виконання цих функцій. А оскільки функції адміністратора несуть у собі дуже багато влади, рекомендується обмежувати адміністративні привілеї настільки, наскільки це можливо.

Горизонтальний контроль доступу

Горизонтальний контроль доступу обмежує доступ до додатків або даних користувачам з необхідними дозволами. Обмеження доступу користувачів не ґрунтуються на старшинстві. Замість цього, користувачі з однаковим рівнем привілеїв мають різні рівні доступу до одних і тих самих ресурсів.

Це поширений інструмент у фінансових компаніях, які прагнуть відповідати вимогам PCI-DSS. Клієнти матимуть доступ до своїх рахунків, де вони зможуть перевіряти баланс і здійснювати перекази. Але вони не матимуть доступу до рахунків інших користувачів.

Контекстно-залежний контроль доступу

Контекстно-залежний контроль доступу обмежує доступ на основі атрибутів програми або користувача, що підключається до неї. Користувачі можуть виконувати дії, якщо вони відповідають певним умовам. В іншому випадку доступ заборонено. Наприклад, на веб-сайтах електронної комерції важливо, щоб клієнти дотримувалися правильного порядку замовлення. Оплата повинна бути завершальним етапом, і клієнти не повинні мати можливості змінити своє замовлення після цього моменту. Контекстні елементи керування роблять це можливим. Магазини можуть перевіряти стан користувача на кожному етапі, регулюючи процес покупки.

Підвищення привілеїв є найпоширенішим типом вразливостей контролю доступу. Існує два основних типи:

– вертикальна ескалація привілеїв;

- горизонтальна ескалація привілеїв.

Вертикальна ескалація привілеїв

Вертикальна ескалація привілеїв передбачає, що користувачі отримують доступ до функцій або даних без необхідних привілеїв. Вертикальна ескалація привілеїв може відбуватися через:

- маніпуляції з URL-адресами – це означає що цей метод передбачає отримання доступу до привілейованих сторінок або функцій шляхом безпосереднього маніпулювання URL-адресами/параметрами без належних перевірок авторизації;
- використання інтерфейсів прикладного програмування (API) – це означає, що деякі кінцеві точки API можуть не забезпечувати належний контроль доступу, що дозволяє непривілейованим користувачам виконувати привілейовані дії шляхом прямого виклику API.

Зловмисники або інсайдери можуть використовувати стандартні облікові записи клієнтів або співробітників для отримання адміністративних привілеїв. Це дає їм набагато більше можливостей для маніпулювання налаштуваннями програми та вилучення конфіденційних даних.

Горизонтальна ескалація привілеїв

Горизонтальна ескалація привілеїв передбачає отримання користувачами доступу до облікових записів інших користувачів на тому ж рівні привілеїв. Горизонтальна ескалація привілеїв може відбуватися за допомогою:

- Маніпуляції з сеансом – це означає, що отримання доступу до сеансу іншого користувача шляхом перехоплення/вгадування токенів/кукі-файлів сеансу або використання незахищеного керування сеансом;
- Помилки перемикання контексту – це обхід засобів контролю доступу шляхом зміни контексту/ідентифікаторів користувача в URL-адресі або через токени підміни без повторної аутентифікації.

Зловмисники можуть використовувати техніку «передати хеш», щоб отримати хеші паролів і отримати доступ, видаючи себе за іншого користувача.

Або ж вони можуть вгадати ідентифікатори інших користувачів. У найпростіших випадках зловмисники можуть просто підвищити свої привілеї, змінивши ідентифікатор цільової URL-адреси.

Горизонтальне ескалацію привілеїв зазвичай важче виявити, ніж вертикальну. Команди безпеки зазвичай бачать, коли звичайні користувачі починають використовувати привілеї адміністратора. Однак аномалії набагато складніше виявити, коли користувачі імітують облікові записи з однаковим рівнем привілеїв.

Більш детально побачити приклад атак зі зламаним контролем доступу можна на рис. 1.7

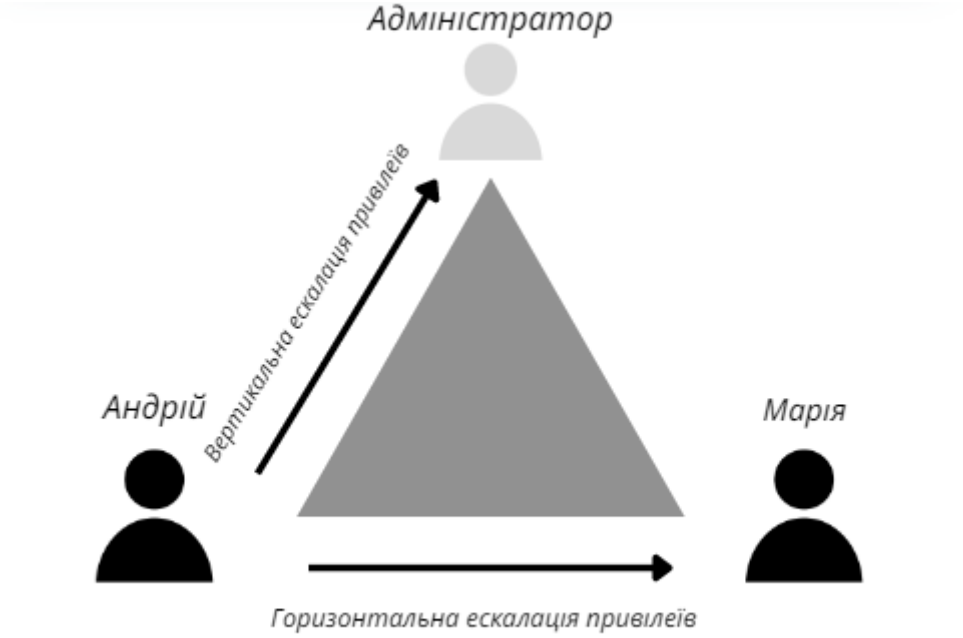

Рисунок 1.7 – Приклад атак зі зламаним контролем доступу

Два основні приклади порушення контролю доступу не є взаємовиключними. Горизонтальна ескалація привілеїв може також призвести до вертикальної ескалації, якщо зловмисники націлені на користувачів з додатковими привілеями. Тому командам безпеки потрібно залишатися пильними щодо обох різновидів порушень контролю доступу.

Ескалація привілеїв є основною формою вразливостей, пов'язаних зі зламаним контролем доступу. Але важливо зазначити, що ескалація може відбуватися різними способами.

Міжсайтовий скриптинг (Cross-Site Scripting, XSS)

Міжсайтовий скриптинг (також відомий як XSS) - це вразливість веб-безпеки, яка дозволяє зловмиснику скомпрометувати взаємодію користувачів з вразливим додатком. Вона дозволяє зловмиснику обійти єдину політику походження, яка призначена для відокремлення різних веб-сайтів один від одного. Уразливості міжсайтового скриптингу зазвичай дозволяють зловмиснику маскуватися під користувача-жертву, виконувати будь-які дії, які може виконувати користувач, і отримувати доступ до будь-яких даних користувача. Якщо користувач-жертва має привілейований доступ у додатку, зловмисник може отримати повний контроль над усіма функціями та даними додатку.

Міжсайтовий скриптинг працює, маніпулюючи вразливим веб-сайтом так, щоб він повертав користувачам шкідливий JavaScript. Коли шкідливий код виконується в браузері жертви, зловмисник може повністю скомпрометувати її взаємодію з додатком. На рис. 1.8 можна побачити приклад такої атаки.

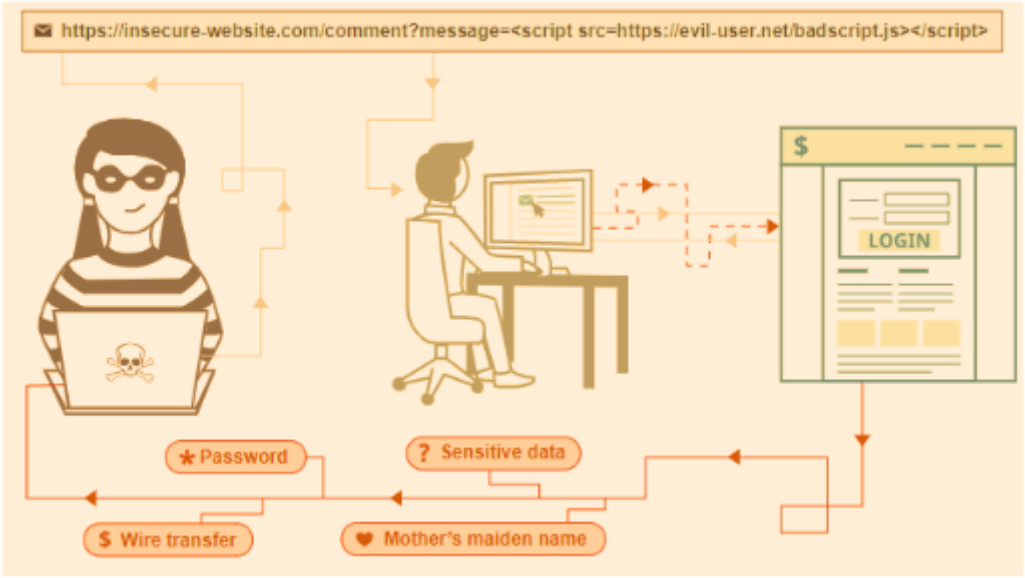

Рисунок 1.8 – Приклад атаки міжсайтовим скриптингом

Існує три основні типи XSS-атак:

- Відображений XSS, коли шкідливий скрипт отримується з поточного HTTP-запиту;
- Збережені XSS, коли шкідливий скрипт надходить з бази даних веб-сайту;

– DOM-орієнтована XSS, коли вразливість існує в коді на стороні клієнта, а не на стороні сервера.

Відображення XSS

Відображений XSS є найпростішим різновидом міжсайтових сценаріїв. Він виникає, коли додаток отримує дані в HTTP-запиті і включає ці дані в негайну відповідь небезпечним способом. На рис. 1.9 можемо побачити простий приклад відображеної вразливості

```
https://insecure-website.com/status?message=All+is+well.
<p>Status: All is well.</p>
```

Рисунок 1.9 – Приклад відображеної вразливості

Програма не виконує жодної іншої обробки даних, тому зловмисник може легко побудувати атаку.

Якщо користувач відвідує URL-адресу, побудовану зловмисником, то сценарій зловмисника виконується в браузері користувача, в контексті сеансу цього користувача з додатком. На цьому етапі скрипт може виконувати будь-які дії і отримувати будь-які дані, до яких користувач має доступ.

Збережений XSS (також відомий як постійний або XSS другого порядку) виникає, коли програма отримує дані з ненадійного джерела і включає ці дані в свої пізніші відповіді HTTP небезпечним способом.

Дані, про які йде мова, можуть бути передані в додаток через HTTP запити; наприклад, коментарі до блогу, прізвиська користувача в чаті або контактні дані про замовлення клієнта. В інших випадках дані можуть надходити з інших ненадійних джерел; наприклад, програма для веб-пошти, що відображає повідомлення, отримані через SMTP, маркетингова програма, що відображає повідомлення в соціальних мережах, або програма для моніторингу мережі, що відображає пакетні дані з мережевого трафіку.

Ось простий приклад збереженої уразливості XSS. На рис 1.10 додаток дошки повідомлень, який дозволяє користувачам надсилати повідомлення, що відображаються іншим користувачам.

```
<p>Hello, this is my message!</p>
```

Рисунок 1.10 – Приклад додатку

Програма не виконує жодної іншої обробки даних, тому зловмисник може легко надіслати повідомлення, яке атакує інших користувачів. На рис 1.11 можемо побачити концепцію такого повідомлення.

```
<p><script>/* Bad stuff here... */</script></p>
```

Рисунок 1.11 – Концепія збереженого XSS повідомлення

DOM-орієнтована XSS

XSS на основі DOM (також відомий як DOM XSS) виникає, коли програма містить якийсь клієнтський JavaScript, який обробляє дані з ненадійного джерела небезпечним способом, зазвичай записуючи дані назад до DOM.

У наступному прикладі на рис 1.12 програма використовує деякий JavaScript для читання значення з поля вводу та запису цього значення до елемента в HTML.

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

Рисунок 1.12 – Приклад коду для демонстрації XSS атаки

Якщо зловмисник може контролювати значення поля вводу, він може легко побудувати шкідливе значення, яке викликає виконання власного скрипта. Наглядно це можна побачити на рис. 1.13

```
You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

Рисунок 1.13 – Приклад виконання власного скрипта

У типовому випадку поле вводу буде заповнено з частини запиту HTTP, наприклад, параметр рядка запиту URL, що дозволяє зловмиснику доставити атаку за допомогою шкідливої URL-адреси так само, як відображається XSS.

Підробка міжсайтових запитів (Cross-Site Request Forgery, CSRF)

Підробка міжсайтових запитів (англ. cross-site request forgery, CSRF), також відома як XSRF, Session Riding або атака одним кліком, є вразливістю веб-безпеки, яка змушує веб-браузер виконувати небажані дії на надійному сайті. Зловмисник зловживає довірою, яку веб-додаток має до браузера жертви, використовуючи довіру, яку веб-додаток має до автентифікованого користувача. Це може статися, коли шкідливий веб-сайт, блог, повідомлення електронної пошти, миттєве повідомлення або веб-додаток змушує веб-браузер користувача виконувати небажані дії на надійному сайті.

Під час атаки CSRF зловмисник приймає особу жертви та використовує її для виконання дій від імені користувача. Наприклад, успішна атака CSRF може змусити користувача переказати кошти, змінити адресу електронної пошти або змінити пароль. Зловмисники зазвичай використовують схеми соціальної інженерії, щоб обдурити користувачів у виконанні цих атак. Наприклад, користувач може отримати електронний лист або текстове повідомлення з посиланням, яке закликає його негайно вжити заходів.

CSRF атаки дозволяють зловмиснику частково обійти політику одного походження, яка призначена для запобігання різних веб-сайтів від втручання один в одного. Щоб атака CSRF була можливою, необхідно встановити три ключові умови:

- операція в веб-додатку, яка надає значення зловмиснику;
- обробка сеансів на основі файлів cookie;
- немає непередбачуваних параметрів запиту.

Успішна атака CSRF може бути руйнівною як для власника веб-додатка, так і для його користувачів. Вплив може включати:

- повний компроміс веб-додатку, якщо потерпілий є адміністративним обліковим записом;
- отримання привілеїв або прийняття ідентичності;
- обхід механізму захисту;
- читання або зміна даних програми;
- відмова в обслуговуванні (збій, вихід або перезапуск).

На рис. 1.14 можна побачити типовий алгоритм роботи CSRF атаки

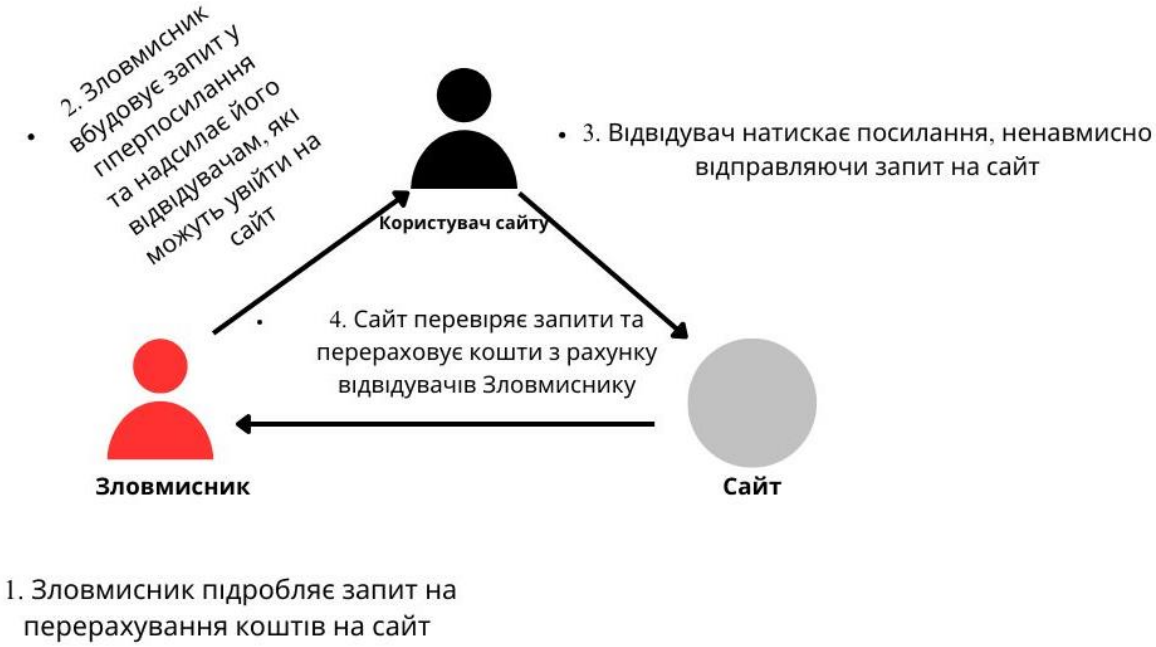

Рисунок 1.14 – Атака CSRF запитом

Коли веб-сайт надсилає запит на дані на інший веб-сайт від імені користувача разом із cookie-файлом сеансу користувача, зловмисник може запустити атаку підробки запиту на крос-сайт, яка зловживає довірчими відносинами між браузером жертви та веб-сервером.

У деяких випадках, залежно від типу дії, зловмисник може отримати повний контроль над обліковим записом користувача. Якщо скомпрометований користувач має привілейовану роль у програмі, зловмисник може мати можливість повністю контролювати всі функціональні можливості та дані програми, що є руйнівним як для бізнесу, так і для користувача. Результатом може бути крадіжка даних, несанкціоновані перекази коштів, пошкоджені відносини з клієнтами, змінені паролі та багато іншого.

Важливим аспектом безпеки є управління сеансами. Погане управління сеансами створює ризики, пов'язані з перехопленням і використанням сеансів користувачів. Наприклад, якщо файли cookie не мають функцій безпеки, таких

як HttpOnly або Secure, зловмисник може перехопити маркери сеансу. Щоб запобігти цьому, слід використовувати безпечні маркери сеансу, обмежувати тривалість сеансу та використовувати протоколи шифрування. Недостатньо безпечне зберігання даних - ще одна серйозна загроза. Зберігання конфіденційних даних у незашифрованому вигляді може призвести до крадіжки в разі зламу системи; використання алгоритмів шифрування, таких як AES і хешування паролів за допомогою алгоритмів bcrypt або PBKDF2, є важливими заходами для забезпечення захисту даних. З огляду на ці загрози, розробники та адміністратори веб-додатків повинні впроваджувати сучасні інструменти безпеки та дотримуватися рекомендацій таких стандартів, як OWASP Top 10 та ISO/IEC 27001. Регулярне тестування на проникнення, використання автоматизованих інструментів аналізу коду та моніторинг активності є важливими елементами ефективного управління інформаційною безпекою. Принципи безпеки на різних етапах розробки веб-додатків Захист веб-додатків - це багатогранний процес, який охоплює різні етапи життєвого циклу розробки програмного забезпечення (SDLC). Щоб забезпечити ефективний захист даних і функціональності програми, важливо, щоб безпека була інтегрована в кожну фазу розробки. Цей підхід відомий як «безпека за задумом» і включає наступні принципи на ключових етапах розробки.

Першим етапом розробки веб-додатків є аналіз вимог безпеки. На цьому етапі важливо зробити наступне:

- ідентифікація критичних даних (необхідно визначити, які дані, наприклад, персональні дані, фінансова інформація, будуть оброблятися, а також рівень їх конфіденційності);
- аналіз загроз (розробляється модель загроз для визначення потенційних векторів атак, моделі дозволяють передбачити можливі сценарії атак та відповідні заходи безпеки для їх нейтралізації);
- оцінка ризиків (визначення ймовірності кожної загрози та наслідків, до яких вона може призвести, це дозволяє розробити перелік пріоритетних заходів

захисту для наступних етапів, цей етап формує основу для всього циклу розробки з точки зору безпеки, оскільки допомагає зрозуміти, які вимоги безпеки повинні бути інтегровані в систему (NIST, 2018)).

Розробка веб-додатку з точки зору безпеки вимагає проектування архітектури, яка може протистояти можливим атакам. Основні принципи, яких слід дотримуватися:

- принцип найменших привілеїв;
- розділення компонентів системи;
- використання захищених протоколів.

Принцип найменших привілеїв.

Надавайте користувачам доступ лише до необхідних функцій і даних, щоб обмежити потенційний вплив у разі компрометації облікового запису.

Розділення компонентів системи.

Кожен компонент системи повинен мати чітко визначені функції. Наприклад, розділення функцій клієнта та сервера може мінімізувати загрозу компрометації сервера через вразливості на стороні клієнта.

Використання захищених протоколів.

Щоб мінімізувати ризик підслуховування, обирайте протоколи, які забезпечують шифрування передачі даних, наприклад, HTTPS. Дотримуючись цих принципів, можна підвищити безпеку системи на етапі проектування та створити гнучку архітектуру, яка зменшує ризик вразливостей (OWASP, 2021).

Впровадження безпечного кодування. Безпечні практики кодування включають:

- перевірка введених даних (всі дані, що надсилаються користувачами, мають бути перевірені та підтверджені відповідно до очікуваного формату, щоб запобігти SQL-ін'єкціям, міжсайтовому скриптингу (XSS) та іншим атакам);
- використання параметризованих запитів (уникайте прямих SQL-запитів та використовуйте параметризовані запити та ORM (об'єктно-реляційне відображення) для захисту від SQL-ін'єкцій);

- розмежування рівнів доступу (логування та моніторинг: скрипти повинні включати механізм реєстрації подій (логів) для відстеження потенційно шкідливих дій та швидкого реагування на події. Для виявлення помилок під час кодування слід використовувати інструменти статичного аналізу коду, які автоматично перевіряють наявність таких вразливостей, як SQL-ін'єкції та XSS.

Тестування безпеки.

Етап тестування повинен включати ретельну перевірку на наявність вразливостей додатку. Основні підходи до тестування безпеки:

- статичний аналіз коду – це виявлення вразливостей в структурі коду без запуску додатку;
- динамічний аналіз – це тестування додатку під час виконання для перевірки його поведінки в реальних ситуаціях;
- тестування на проникнення (pen testing) – це імітація реальних атак для виявлення та оцінки ефективності захисту додатків;
- ручне тестування та перегляд коду – це говорить про те, що окрім автоматизованих методів, важливо також вручну перевіряти код на відповідність принципам безпеки, такі інструменти, як OWASP ZAP і Burp Suite для тестування безпеки, широко використовуються для автоматизації перевірок на наявність поширених загроз.

Моніторинг і підтримка безпеки на етапі експлуатації. Етап експлуатації є останнім, але критично важливим етапом забезпечення безпеки. Після розгортання веб-додатків їх необхідно постійно контролювати, щоб своєчасно виявляти загрози та підтримувати стабільність. Ключові аспекти підтримки безпеки:

- регулярний моніторинг (систематичний моніторинг дій користувачів і журналів для виявлення аномалій);
- оновлення та виправлення вразливостей (регулярне виправлення компонентів, бібліотек та серверних систем для захисту від відомих вразливостей);

- аудит безпеки (регулярні перевірки систем на відповідність новим вимогам безпеки та оновленням політик);
- управління інцидентами (впровадження протоколів у разі виникнення інциденту безпеки, включаючи швидке реагування, оцінку збитків та усунення наслідків для зменшення ризику порушення безпеки, моніторинг і підтримка безпеки є важливими для підтримки надійності веб-додатків в умовах постійно зростаючих загроз і мінливого кіберсередовища).

2 МЕТОДИ ТА ІНСТРУМЕНТИ ТЕСТУВАННЯ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

2.1 Статичний аналіз

Статичний аналіз коду - це фундаментальний метод інформаційної безпеки, який дозволяє знайти потенційні вразливості програмного забезпечення ще до його розгортання. Цей підхід особливо ефективний на ранніх стадіях розробки, оскільки базується на аналізі вихідного коду та байткоду без запуску програми. Статичний аналіз спрямований на виявлення системних помилок, які можуть призвести до таких атак, як SQL-ін'єкції, XSS, неправильна обробка винятків і порушення політики доступу. Статичний аналіз можна використовувати не тільки для зменшення ризиків, але й для забезпечення відповідності програмного забезпечення вимогам стандартів безпеки, таких як OWASP Secure Coding Practices, ISO/IEC 27034 та NIST SP 800-53. Методи статичного аналізу базуються на алгоритмічному аналізі вихідного коду. Такі інструменти застосовують кілька основних підходів до аналізу:

- Лексичний аналіз (розбиває код на базові компоненти і виявляє синтаксичні помилки);
- Семантичний аналіз (визначає логічну правильність програмного забезпечення та перевіряє код на відповідність стандартам);
- Формальний аналіз (використовує математичні моделі для перевірки властивостей безпеки).

Однією з головних особливостей статичного аналізу є те, що він може виявити потенційні помилки до того, як код буде виконаний; Boehm (2006) виявив, що виправлення помилок на етапі розробки в 15 разів дешевше, ніж їх виправлення в готовому програмному забезпеченні. Типи вразливостей, виявлених за допомогою статичного аналізу може бути SQL-ін'єкція. Цей тип атаки дозволяє зловмисникам маніпулювати SQL-запитами, вставляючи шкідливий код у поля введення. На рис. 2.1 наведений приклад атаки SQL-ін’єкцією.

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

Рисунок 2.1 – Приклад атаки SQL- ін’єкцією

Якщо $username не обробляється належним чином, зловмисник може вставити код, який виконує небажані операції, такі як видалення бази даних.

Міжсайтовий скриптинг (XSS)

Цей вид вразливості дозволяє виконати шкідливий JavaScript-код у браузері користувача, що може призвести до небажаних дій, таких як перехоплення сеансу.

Несанкціонована обробка виключень

Неправильна обробка помилок може надати зловмиснику інформацію про структуру додатку або конфігурацію сервера.

Порушення контролю доступу

Неправильна реалізація політик доступу може дозволити неавторизованим користувачам отримати доступ до конфіденційних даних і функціональності.

Інтеграція статичного аналізу в процес CI/CD гарантує автоматичне виявлення вразливостей на кожному етапі розробки. Наприклад, SonarQube можна налаштувати на перевірку коду при кожному перенесенні до сховища, забезпечуючи безперервний контроль якості. Поширені інструменти для статичного аналізу В сучасних додатках використовується велика кількість інструментів для статичного аналізу, кожен з яких має свої особливості. У таблиці 2.1 наведено порівняння найпоширеніших інструментів.

Таблиця 2.1 – Порівняння інструментів статистичного аналізу

Інструмент	Підтримувані мови	Особливості	Ціна	Подходить для

SonarQube	Java, Python, C#, PHP	Відкритий код, інтеграція з CI/CD	Безкоштовний / Pro	Малі та середні команди
Checkmarx	Java, .NET, PHP	Рекомендації щодо виправлення помилок	Комерційна ліцензія	Великіі організації
Forfify SCA	Java, C/C++, Python	Підтримка великих проєктів	Висока	Корпорації

2.2 Динамічний аналіз

Динамічний аналіз коду – це процес тестування веб-додатку під час його роботи в реальному або тестовому середовищі. На відміну від статичного аналізу, який досліджує лише вихідний код, динамічний аналіз фокусується на поведінці програми та взаємодії її компонентів. Цей метод може виявити проблеми, які залежать від конфігурації сервера, його взаємодії з базою даних або поведінки під навантаженням.

Характеристики динамічного аналізу:

- реалістичність середовища (тести виконуються в умовах, максимально наближених до реального використання програми. Це дозволяє виявити проблеми з конфігурацією сервера або помилкові запити до API);
- тестування бізнес-логіки (цей метод дозволяє виявити помилки в логіці програми, які можуть бути спричинені некоректною обробкою даних або ненадійним контролем доступу);
- застосування інструментів (динамічний аналіз базується на використанні спеціалізованих інструментів, які автоматично генерують запити до додатку для виявлення потенційних вразливостей).

Динамічний аналіз особливо корисний для виявлення наступних проблем:

- міжсайтовий скриптинг (вставка шкідливого JavaScript коду);

- CSRF (Cross-Site Request Forgery) підробка міжсайтових запитів (атаки, що змушують користувачів виконувати небажані дії);
- недостатній контроль доступу (помилки, що дозволяють обійти механізми авторизації);
- вразливий API (некоректна обробка запитів до сервера).

У сучасній практиці використовується низка інструментів для динамічного аналізу, кожен з яких має свої особливості. У таблиці 2.2 наведено порівняння найпоширеніших інструментів.

Таблиця 2.2 – Порівняння інструментів динамічного аналізу

Інструмент	Тип вразливості	Особливості	Вартість	Рівень користувача
OWASP ZAP	XSS, SQLi, CSRF	Відкритий код, інтеграції з браузерами	Безкоштовний	Початковий
Burp Suite	XSS, SQLi, CSRF, DoS	Ручне та автоматизоване тестування	Безкоштовний / Pro	Середній
Nessus	Конфігурації серверів, DoS	Сканування серверів та додатків	Комерційна ліцензія	Досвідчений

2.3 Тестування методом проникнення

Тестування на проникнення (пентестування) - це процес моделювання, який дозволяє виявити вразливості у веб-додатку шляхом імітації атаки. Його

основною метою є не тільки виявлення вразливостей, але й перевірка їх фактичного використання та оцінка їх можливого впливу на систему. Пентестування фокусується на захист конфіденційних даних, стійкість до атак на бізнес-логіку, правильність конфігурації серверів та API, надійність механізмів аутентифікації та авторизації.

Цей метод є одним з найефективніших способів оцінки безпеки, оскільки враховує не тільки окремі вразливості, а й комбінації, які можуть бути використані в складних атаках. Кожна фаза відіграє певну роль у процесі оцінки безпеки веб-додатку:

- розвідка;
- сканування;
- експлуатація.

Розвідка (збір інформації)

На цьому етапі збираються дані про цільову систему, які потім використовуються для планування атаки. Основними завданнями є аналіз загальної інформації (IP-адреси, доменів і субдоменів), виявлення відкритих портів і сервісів та визначення версій програмного забезпечення. Використані інструменти: Nmap, Shodan, Recon-ng.

Сканування

Цей етап спрямований на виявлення відомих вразливостей за допомогою автоматизованих інструментів. Результатом є список потенційно небезпечних компонентів, які потребують детального аналізу. Інструменти Nessus, OpenVAS, Burp Suite Pro.

Експлуатація

Фаза, на якій тестуються фактичні експлойти виявлених вразливостей. Сюди відносяться і атаки на механізми автентифікації (груба сила, підміна облікових даних); і SQL-ін'єкції для отримання доступу до баз даних; і використання міжсайтового скриптингу (XSS) для виконання шкідливого коду. Інструменти Metasploit Framework, SQLmap, BeEF. Після успішної атаки оцінюють масштаб її впливу.

Основні завданнями є виявлення конфіденційних даних, які могли бути скомпрометовані; аналіз можливостей ескалації привілеїв; дослідження шляхів горизонтального руху в мережі (Lateral Movement) та звітування, саме підготовка детального звіту, що включає в собі опис виявлених вразливостей і того, як вони можуть бути використані, можливі наслідки атаки та рекомендації щодо вирішення проблеми.

В залежності від типу уразливості, що перевіряється, в пентесті використовуються різні техніки атак:

- атаки на авторизацію;
- грубий перебір (автоматичний підбір паролів для доступу до облікових записів);
- вкидання облікових даних (використання скомпрометованих паролів з інших систем);
- перехоплення сеансу.

Тестування на проникнення є одним з найефективніших методів забезпечення безпеки і має багато переваг:

- виявлення реальних ризиків, якими можуть скористатися зловмисники;
- оцінка ефективності існуючих механізмів безпеки;
- можливість виявити помилки в бізнес-логіці, які часто не враховуються іншими методами;
- зниження ризиків за рахунок генерації рекомендацій щодо виправлення проблем.

Незважаючи на свої переваги, ручне тестування також має певні проблеми:

- складність налаштування (для проведення ручного тестування потрібні спеціалізовані знання та навички);

- часові обмеження (ручне тестування може зайняти багато часу, особливо для великих систем);

- ризик простою системи (неналежне використання вразливостей може призвести до збою в роботі додатків);

- Необхідність комплексного підходу (одних лише методів тестування може бути недостатньо для виявлення складних загроз).

Інструменти, що використовуються для пентестування, можна розділити на кілька категорій. У таблиці 2.3 представлено найпоширеніші з них.

Таблиця 2.3 – Інструменти для пентестингу

Інструмент	Типи атак	Особливості	Рівень користувача
Metasploit	Експлуатація, DoS	Багата база експлойтів, інтеграція з API	Досвідчений
Kali Linux	Сканування, експлуатація	Комплекс інструментів для тестування	Початковий
Burp Suite Pro	SQLi, XSS, логіка	Ручне й автоматизоване тестування	Середній

2.4 Автоматизовані інструменти для пошуку вразливостей

Автоматизовані інструменти для пошуку вразливостей є основним елементом сучасного забезпечення безпеки веб-додатків. Вони надають можливість ефективно і швидко виявляти потенційні загрози в програмному забезпеченні та конфігурації інфраструктури, що є особливо важливим у великих проєктах. За-

вдяки автоматизації тестувань компанії можуть значно зменшити ризики, забезпечуючи регулярні перевірки безпеки без надмірного залучення людських ресурсів.

Основні функції автоматизованих інструментів.

Сканування коду та конфігурацій серверів.

Автоматизовані інструменти аналізують програмний код і конфігурації серверів для виявлення найпоширеніших вразливостей. Це включає:

- SQL-ін’єкції (SQL Injection);
- міжсайтовий скриптинг (XSS);
- помилки конфігурації серверів (наприклад, відсутність SSL/TLS).

Перевірка API-запитів

API є однією з основних мішеней для атак, тому інструменти дозволяють перевіряти:

- відповідність стандартам безпеки (наприклад, OAuth 2.0);
- неправильну обробку даних у JSON або XML-запитах;
- захищеність від DoS-атак через некоректні запити.

Автоматизація тестувань у CI/CD

Інтеграція інструментів із CI/CD дозволяє автоматично перевіряти код після кожної зміни. Це знижує ймовірність впровадження нових вразливостей у релізи.

Формування звітів

Генерація детальних звітів із переліком виявлених вразливостей, їхнім рівнем ризику та рекомендаціями щодо усунення дозволяє розробникам швидко вносити зміни.

Аналіз змін у коді

Інструменти здатні порівнювати новий код із попередніми версіями, виявляючи вразливості, що з’явилися під час внесення змін.

Автоматизовані інструменти відрізняються за функціональністю, типами виявлених вразливостей та можливостями інтеграції. Таблиця 2.4 представляє ключові з них.

Таблиця 2.4 – Інструменти для автоматизованого тестування

Інструмент	Тип вразливостей	Особливості	Ціна	Цільова аудиторія
Acunetix	SQLi, XSS, API	Інтеграція з CI/CD, хмарна версія	Комерційна ліцензія	Великі компанії
Nessus	Конфігурація серверів, DoS	Сканування мереж і серверів	Безкоштовний / Pro	Малі та середні команди
Burp Suite Pro	SQLi, XSS, бізнес-логіка	Ручне та автоматизоване тестування	Комерційна ліцензія	Середній бізнес
Qualys Scanner	API, незахищені дані	Хмарна платформа для швидкого сканування	Висока	Корпорація

Переваги автоматизованих інструментів:

- швидкість тестування (інструменти дозволяють проводити сканування за лічені хвилини навіть для великих проєктів);
- масштабованість (автоматизовані рішення можуть адаптуватися до потреб великих систем із тисячами компонентів);
- зменшення людського фактора (завдяки автоматизації зменшується ймовірність помилок, викликаних людським фактором);
- регулярність (інструменти дозволяють проводити регулярні перевірки безпеки без значного навантаження на команду розробників).

Виклики використання автоматизованих інструментів:

- хибнопозитивні результати (інструменти можуть генерувати звіти з великою кількістю хибних спрацьовувань, що вимагає додаткового аналізу вручну);

– вартість ліцензій (деякі інструменти мають високу вартість, що може бути недосяжним для малих компаній);
– складність налаштування (великі системи потребують ретельного налаштування інструментів для точного сканування).

2.5 Порівняння популярних інструментів для тестування безпеки

Вибір інструментів для тестування безпеки веб-додатків є критично важливим етапом у забезпеченні інформаційної безпеки. Основними критеріями вибору є:

- типи вразливостей, які може виявити інструмент;
- інтеграція з процесами розробки (CI/CD);
- зручність використання;
- вартість і ліцензування;
- підтримка актуальних стандартів безпеки (OWASP, PCI DSS, ISO 27001).

Кожен із цих критеріїв враховує потреби команди розробників, масштаби проєкту та специфіку інфраструктури.

Інструменти тестування безпеки поділяються на кілька категорій:

- для статичного аналізу коду (SAST);
- для динамічного аналізу (DAST);
- для інтерактивного тестування (IAST);
- для тестування API;
- інструменти для тестування методом проникнення.

У таблиці 2.5 представлено порівняння основних інструментів за ключовими параметрами.

Таблиця 2.5 – Порівняння популярних інструментів для тестування

Інструмент	Категорія	Типи вразливостей	Особливості	Ціна	Цільова аудиторія

Acunetix	DAST	SQLi, XSS, API	Хмарна версія, інтеграція з CI/CD	Комерційна ліцензія	Великі компанії
Nessus	Сканування серверів	Конфігурації DoS	Перевірка конфігурації і мереж	Безкоштовний / Pro	Середній бізнес
Burp Suite Pro	DAST	SQLi, XSS, бізнес-логіка	Автоматизація та ручне тестування	Комерційна ліцензія	Розробники
SonarQube	SAST	XSS, SQLi, логіка коду	Статичний аналіз, інтеграція з CI/CD	Безкоштовний / Pro	Початковий рівень
Qualys Scanner	Хмарна платформа	API, сервери	Масштабованість, підтримка стандартів	Висока	Великі організації
Metasploit	Пентестинг	Експлуатація вразливостей	Автоматизація атак, база експлойтів	Досвідчений	Тестувальники безпеки

Аналіз інструментів

Acunetix спеціалізується на тестуванні веб-додатків і API. Його перевагами є потужна інтеграція з CI/CD, широкий спектр виявлених вразливостей (SQL-ін’єкції, XSS, CSRF) та зручний інтерфейс. Однак висока вартість ліцензії обмежує його використання в малих компаніях.

Nessus орієнтований на перевірку конфігурацій серверів і мереж. Він виявляє помилки в налаштуваннях SSL/TLS, слабкі паролі, відкриті порти та застарілі версії ПЗ. Nessus є одним із найдоступніших інструментів, що робить його популярним серед малих і середніх компаній.

Burp Suite поєднує автоматичне і ручне тестування веб-додатків. Він підтримує перевірку бізнес-логіки, SQL-ін'єкцій та XSS. Вартість професійної версії є помірною, що робить цей інструмент популярним серед середніх компаній.

SonarQube забезпечує статичний аналіз коду, інтегруючись із CI/CD для автоматизації процесу. Його перевагою є безкоштовна версія з базовими функціями, що дозволяє використовувати інструмент для невеликих проєктів.

Qualys Scanner є хмарною платформою, яка підходить для великих організацій із розгалуженою інфраструктурою. Вона забезпечує перевірку серверів, API і відповідність стандартам безпеки, таким як PCI DSS та ISO 27001.

Metasploit є найпотужнішим інструментом для тестування методом проникнення. Він надає велику базу експлойтів, підтримує автоматизацію атак і дозволяє проводити реальні перевірки систем безпеки.

Виклики вибору інструментів:

- вартість;
- складність інтеграції;
- хибнопозитивні результати.

Вартість

Інструменти, такі як Acunetix та Qualys, мають високу вартість ліцензій, що може обмежити їх використання в малих компаніях.

Складність інтеграції

Деякі інструменти вимагають складного налаштування для роботи з CI/CD.

Хибнопозитивні результати:

Автоматизовані інструменти можуть генерувати зайві попередження, що вимагає ручного аналізу.

Порівняння показує, що вибір інструмента залежить від масштабів проєкту, типу вразливостей і доступного бюджету. Для малих проєктів підходять безкоштовні або недорогі інструменти, такі як SonarQube або Nessus. Великі компанії потребують комплексних рішень, таких як Acunetix чи Qualys, для забезпечення відповідності стандартам.

Проведене порівняння допомагає зрозуміти сильні та слабкі сторони кожного інструмента, що дозволяє обрати оптимальне рішення для конкретних завдань безпеки.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

3.1 Опис тестового середовища

SafeAndAttack - це освітній проект, покликаний показати, як захистити веб-додатки і як проводити тести на атаки, що складається з двох частин:

Захищена версія додатку, вона включає інтегровані функції безпеки, такі як JWT-аутентифікація, двофакторна автентифікація (2FA), Google reCAPTCHA v2, обмеження швидкості, CSRF-захист та ін.

Незахищена версія програми: призначена для тестування поширених уразливостей, таких як SQL-ін'єкція, XSS та CSRF.

3.2 Налаштування середовища

Спочатку потрібно скопіювати репозиторій. Наглядно це можна побачити на рис 3.1

```bash
git clone https://github.com/valentynvovchak/SafeAndAttack.git
cd SafeAndAttack
```

Рисунок 3.1 – Клонування репозиторію

Надалі, налаштування бекенду. Для цього буде використовуватися середовище Python. На рис. 3.2 можна побачити практичне застосування

```bash
python -m venv venv
source venv/bin/activate  # Для Linux/Mac
venv\Scripts\activate     # Для Windows
```

Рисунок 3.2 – Створення віртуального середовища Python

Наступним кроком є встановлення залежності. На рис. 3.3 це можна наглядно побачити.

```bash
pip install -r requirements.txt
```

Рисунок 3.3 – Встановлення залежності

Далі буде застосування міграції, на рис. 3.4 можна побачити як з технічної точки зору це виглядає.

```bash
python manage.py makemigrations
python manage.py migrate
```

Рисунок 3.4 – Застосування міграції

Останнім кроком налаштування фронтенду є запуск серверу. На рис. 3.5 зображений цей процес.

```bash
python manage.py runserver
```

Рисунок 3.5 – Запуск серверу

Налаштування фронтенду
Перехід до директорії frontend можна побачити на рис. 3.6

```bash
cd frontend
```

Рисунок 3.6 – Перехід до директорії frontend

Встановлення залежності, на рис. 3.7 представлене практичне застосування

```bash
npm install
```

Рисунок 3.6 – Залежність у frontend

Запуск frontend технічно зображено на рис. 3.7

```bash
npm start
```

Рисунок 3.7 – Запуск frontend

Отримуємо доступ до додатка:

- бекенд захищеного сайту (http://localhost:8000);
- фронтенд захищеного сайту (http://localhost:3000);
- незахищений сайт (http://localhost:5000).

3.3 Статичний аналіз коду за допомогою SonarQube

Статичний аналіз кодудопомагаєвиявити потенційні вразливості та баги-без запуску програми. Для цього використовується SonarQube, який забезпечує автоматизований аналіз коду та інтеграцію з процесами CI/CD.

Налаштування проекту:

- створення нового проекту в SonerQube;
- генерація токена для автентифікації;
- налаштування файлу sonar-project.properties у кореневій директорії проекту.

Наглядно це можна побачити на рис 3.8

```
arduino

sonar.projectKey=SafeAndAttack
sonar.sources=.
sonar.host.url=http://localhost:9000
sonar.login=your_generated_token
```

Рисунок 3.8 – Процес налаштування проекту

Для запуску аналізу використовуємо SonarScanner. На рис. 3.9 можна побачит як програмно це записується.

```
bash

sonar-scanner
```

Рисунок 3.9 – Запуск аналізу за допомогою SonarScanner

Після завершення аналізу в SonarQube виявлені наступні проблеми:

- SQL-ін'єкції у функції обробки запитів до бази даних;
- недостатня валідація користувацького вводу;
- використання застарілих бібліотек з відомими уразливостями;
- потенційні проблеми з керуванням пам'яттю;
- недотримання стандартів кодування;
- відсутність коментарів у критичних частинах коду.

Детальніше, результати можна побачити на рис 3.10

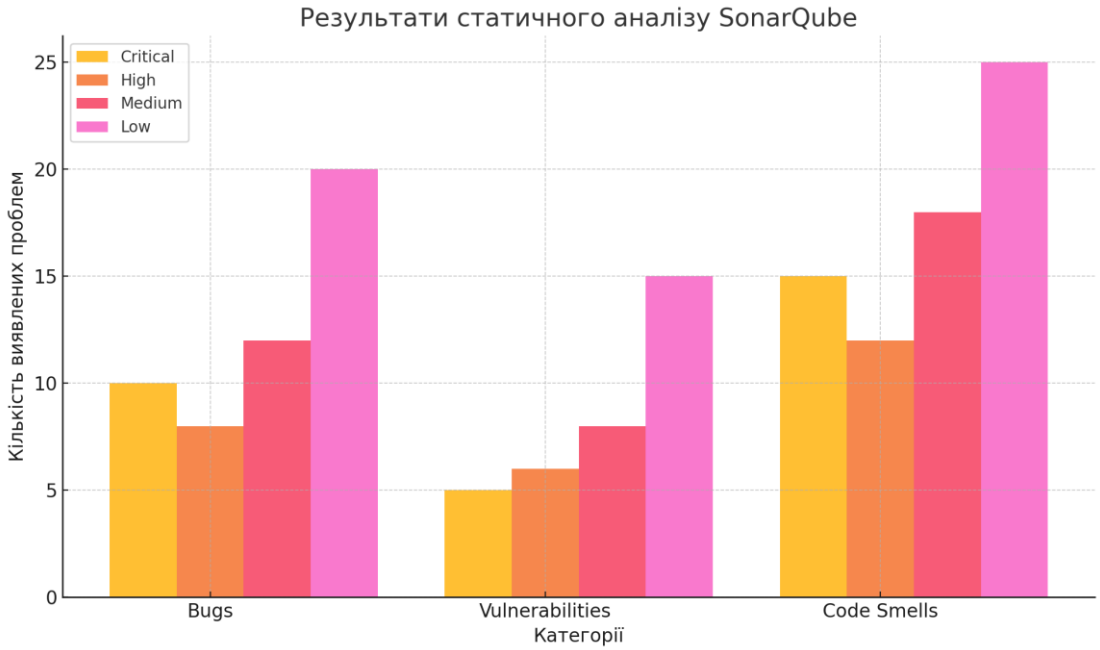

Рисунок 3.10 – Результати статистичного аналізу SonarQube

У результатах статистичного аналізу можемо побачити три категорії:

– Bugs (помилки);
– Vulnerabilities (вразливості);
– Code smells (недоліки коду).

Bugs

Bugs – це дефект у програмному забезпеченні,який призводить до неправильної роботи програми або докритичної помилки. Помилки можуть призвести доаварійного завершення роботи системи або некоректної поведінки системи. Наслідки можуть призвести до аварійного завершення роботи програми, некоректних обчислень та некоректного представлення даних.

Vulnerabilities

Vulnerabilities – це потенційні загрози безпеці, які зловмисник може використати для отримання несанкціонованого доступу до систем та даних. Наслідками цього є те, що зловмисники можуть використовувати ці вразливості для отримання конфіденційної інформації, обходу автентифікації або модифікації даних у системі.

Code Smells – це дефекти коду, які не обов'язково є помилками, але які можуть ускладнити підтримку та розвиток проекту. Вони вказують на погану архітектуру або надто складний код. Дефекти коду ускладнюють рефакторинг і знижують продуктивність команди. Вони також підвищують ризик появи нових помилок під час обслуговування коду.

3.4 Динамічний аналіз за допомогою OWASP ZAP

Динамічний аналіз є одним з основних методів тестування безпеки веб-додатків і виконується в процесі виконання самої програми та передбачає взаємодію з сервером за допомогою HTTP-запитів. OWASP ZAP (Zed Attack Proxy) – один з найпопулярніших інструментів динамічного аналізу, який може автоматично і вручну сканувати веб-додатки на наявність вразливостей.

Налаштування OWASP ZAP:

1. Налаштування проксі-сервера OWASP ZAP для перехоплення HTTP-запитів.
2. Запуск веб-додатка (незахищеної версії SafeAndAttack) на http://localhost:5000.

Надалі запустимо активне сканування (Active Scan), яке автоматично проведе сканування всіх доступних URL-адрес веб-додатка та виконає атаки на виявлення поширених вразливостей, а саме SQL-ін’єкції, Cross-Site Scripting (XSS); незахищені заголовки безпеки.

Після завершення тестування формується звіт із переліком знайдених вразливостей. У таблиці 3.1 представлені результати тестування веб-додатка.

Таблиця 3.1 – Результат динамічного аналізу OWASP ZAP

Тип вразливості	Кількість виявлених випадків	Рівень критичності	Опис

SQL Injection	3	Високий	Уразливості дозволяють виконувати SQL-запити на сервері.
Cross-Site Scripting (XSS)	5	Середній	Дозволяє виконувати зловмисний JavaScript-код у браузері користувача
Незахищені куки	4	Низький	Відсутній прапорець Secure у файлах кукі
Відсутні заголовки безпеки	2	Низький	Відсутній Content-Security-Policy(CSP)
Міжсайтова підробка запитів (CSRF)	2	Високий	Змушує користувача виконати небажану дію на веб-додатку, де він вже авторизований

3.5 Тестування методом проникнення за допомою Metasploit

Підготовка до тестування.

Запуск веб-додатку, який працює на порту localhost: 5000. На рис. 3.11 можемо це наглядно побачити.

```bash
npm run dev
```

Рисунок 3.11 – Запуск веб-додатку

Запуск Metasploit Framework. На рис. 3.12 можна побачити за допомогою якої команди була запущена консоль Metasploit.

```bash
msfconsole
```

Рисунок 3.12 – Консольна команда запуску Metasploit

Налаштування параметрів.

RHOST: 127.0.0.1 – локальний сервер;

PORT: 5000 – порт веб-додатку.

Проведення тестування.

Для перевірки активних портів і служб було використано модуль TCP-сканування. На рис. 3.13 можемо побачити саму перевірку.

```bash
use auxiliary/scanner/portscan/tcp
set RHOSTS 127.0.0.1
run
```

Рисунок 3.13 – Перевірка активних портів і служб

У результаті отримаємо, що порт 5000 (HTTP) – активний, порт 22 (SSH) – активний, але доступ обмежений для авторизованих IP, порт 3000 – неактивний.

Перевірка на SQL-ін’єкції

Використуємо модуль на сторінці “/login”, який зображено на рис. 3.14, для перевірки на уразливі параметри.

```
bash

use auxiliary/scanner/http/sql_injection
set RHOSTS 127.0.0.1
set TARGETURI /login
run
```

Рисунок 3.14 – Модуль перевірки на уразливі параметри

Результатом є те, що Metasploit виявив уразливий параметр у полі імені користувача. Зловмисник може вставити SQL-запити у форму входу в систему.

Надалі спробуємо використати наступний експлойт для того щоб перевірити можливість отримання бази даних. На рис. 3.15 можемо побачити технічну частину експлойту.

```
bash

use exploit/unix/webapp/sqlmap_sqli
set RHOSTS 127.0.0.1
set TARGETURI /login
set PAYLOAD generic/shell_reverse_tcp
run
```

Рисунок 3.15 – Структура експлойту

В результаті отримуємо доступ до таблиці 3.2, users, у якій зберігаються облікові записи.

Таблиця 3.2 – Результат роботи експлойту

ID користувача	Ім’я користувача	Email	Тип облікового запису	Хеш пароля (MD5)
1	admin	admin@gmail.com	Адміністратор	5f4dcg3b5aa765d61d8327deb882cf99
2	dakota.gibs	dakota.gibs@gmail.com	Користувач	25d55at283aa400af464c76d713c07ad
3	andrew.twen	andrew.twen@gmail.com	Користувач	098f6bcd4621b373cade4e832627b4f6
4	support	support@gmail.com	Підтримка	72b302bn297a228a75730123efef7c41
5	tester	tester@gmail.com	Тестувальник	6f1ed012ab5595859014ebf0951522d9

Для отримання результатів тестування на XSS-уразливість використуємо наступний код, який можна побачити на рис. 3.16

```
bash

use auxiliary/scanner/http/xss_fuzzer
set RHOSTS 127.0.0.1
set TARGETURI /search
run
```

Рисунок 3.16 – Структура тестування на XSS-уразливість

Результатом є виявлення можливості введення JavaScript-коду в поле пошуку. Прикладом такого впровадження може бути рис. 3.17

```javascript
<script>alert('XSS атака');</script>
```

Рисунок 3.17 – Приклад впровадження скрипту

Скрипт виконувався в браузері користувача, тому введені дані не фільтрувалися.

Наступним кроком є перевірка доступу до API. Для цього тестування було використано модуль, який зображений на рис. 3.18

```bash
use auxiliary/scanner/http/api_fuzz
set RHOSTS 127.0.0.1
set TARGETURI /api/v1/products
run
```

Рисунок 3.18 – Модуль для тестування API-запитів

Результатом є те, що запит GET / api / v1 / products повертає дані без авторизації.

Підсумком тестування за допомогою Metasploit є таблиця 3.3, у якій демонструються всі вразливості виявлені за допомогою цього методу

Таблиця 3.3 – Результати тестування за допомогою Metasploit

Тип вразливості	Об’єкт атаки	Модуль/експлойт	Результат

SQL-ін’єкція	/login	exploit/unix/webapp/sqlmap_sqli	Отримано доступ до бази даних
XSS(Cross-Site Scripting)	/search	auxiliary/scanner/http/xss_fuzzer	Виконання шкідливого JavaScript-коду у браузері
Відкритий API	/api/v1/products	uxiliary/scanner/http/api_fuzz	Виявлено доступ до даних без авторизації

3.6 Аналіз отриманих результатів

У цьому розділі веб-додаток SafeAndAtack було детально протестовано трьома основними методами:

- статичний аналіз (за допомогою інструмента SonarQube у вихідному коді виявлено критичні помилки, зокрема SQL-ін’єкції);
- динамічний аналіз (за допомогою інструменту OWASP ZAP виявлено 5 уразливостей, зокрема й XSS та CSRF);
- тестування методом проникнення (інструмент Metasploit дозволив отримати реальна демонстрація модулів та експлойтів , зокрема процес доступу до бази даних за допомогою SQL-ін’єкції).

ВИСНОВКИ

В магістерській роботі розглянуто питання дослідження методів тестування безпеки веб-додатків та впровадження інструментів для виявлення вразливостей.

Проведено детальний аналіз видів вразливостей та методів тестування безпеки веб-додатків, що дозволяють виявляти потенційні вразливості в коді та архітектурі веб-додатків, зокрема, такі як статичний та динамічний аналіз, а також тестування методом проникнення.

Також, описані інструменти для тестування, порівняння їх за ключовими характеристиками та розбір переваг та недоліків окремого виду.

Поставлено практичний експеримент з використанням існуючого відкритого освітнього проекту SafeAndAtack, який створений для того, щоб показати, як захистити веб-додатки і як проводити тести на атаки, що складається з двох частин, це захищена версія додатку та незахищена версія програми: призначена для тестування поширених уразливостей.

Застосовані інструменти SonarQube, OWASP ZAP, Metaspoil дозволили імітувати реальні загрози, що дає можливість оцінити рівень захищенності веб-додатку.

Актуальність цього дослідження підтверджується великою кількістю атак на веб-додатки та зростанням ролі кібербезпеки в сучасному суспільстві. Результати цього дослідження можуть бути використані компаніями-розробниками програмного забезпечення для створення більш надійних і безпечних додатків.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. OWASP Foundation. OWASP Top 10 Vulnerabilities (2021) [Електронний ресурс]. – Режим доступу: https://owasp.org.
2. National Institute of Standards and Technology. SP 800-53: Security and Privacy Controls for Information Systems and Organizations [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov.
3. SonarSource Documentation: Документація до SonarQube: статичний аналіз коду [Електронний ресурс]. – Режим доступу: https://docs.sonarqube.org.
4. PortSwigger. SQL Injection: How to Prevent SQL Injection [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/sql-injection#how-to-prevent-sql-injection.
5. Authgear. Broken Authentication: What is it and How to Prevent It? [Електронний ресурс]. – Режим доступу: https://www.authgear.com/post/broken-authentication-what-is-it-and-how-to-prevent-it#definition.
6. NordLayer. Broken Access Control: Understanding the Threat and How to Prevent It [Електронний ресурс]. – Режим доступу: https://nordlayer.com/learn/access-control/broken-access-control/.
7. PortSwigger. Cross-Site Scripting (XSS): What is Cross-Site Scripting? [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/cross-site-scripting#what-is-cross-site-scripting-xss.
8. BrightSec. Cross-Site Request Forgery (CSRF): What is CSRF and How to Prevent It? [Електронний ресурс]. – Режим доступу: https://brightsec.com/blog/cross-site-request-forgery-csrf/.
9. PortSwigger Documentation: Документація до Burp Suite [Електронний ресурс]. – Режим доступу: https://portswigger.net.

10. Acunetix Vulnerability Scanner: Опис функціональних можливостей інструмента Acunetix [Електронний ресурс]. – Режим доступу: https://www.acunetix.com.
11. Tenable Documentation: Nessus Vulnerability Scanner [Електронний ресурс]. – Режим доступу: https://www.tenable.com.
12. Shostack, A. Threat Modeling: Designing for Security. – Wiley, 2014. – 624 с.
13. Chess, B., & McGraw, G. Static Analysis for Security // IEEE Security & Privacy, 2004. – Т. 2, № 6. – С. 76-79.
14. OWASP ZAP User Guide: Інструкції для налаштування та використання OWASP ZAP [Електронний ресурс]. – Режим доступу: https://www.zaproxy.org.
15. Metasploit Framework Documentation: Інструмент для пентестингу [Електронний ресурс]. – Режим доступу: https://www.metasploit.com.
16. Kali Linux Tools Documentation: Документація до інструментів пентестингу [Електронний ресурс]. – Режим доступу: https://www.kali.org/tools.
17. Qualys Vulnerability Management: Хмарний сканер вразливостей [Електронний ресурс]. – Режим доступу: https://www.qualys.com.
18. Grossman, J. Cross-Site Scripting Attacks and How to Prevent Them // IEEE Computer Security, 2011. – Т. 44, № 7. – С. 34-38.
19. OWASP Testing Guide v4: Документ з рекомендаціями щодо тестування безпеки [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-testing.
20. PCI DSS: Стандарт захисту даних платіжних карток [Електронний ресурс]. – Режим доступу: https://www.pcisecuritystandards.org.
21. ISO/IEC 27001: Стандарт систем управління інформаційною безпекою [Електронний ресурс].
22. Панасенко Д.О., Захватова Т.Є., Сидоренко Г.М. Дослідження методів тестування безпеки Веб додатків // Collection of Scientific Papers 1

International Scientific and Practical Conference «Global Trends in the Development of Information Technology and Science» (January 8-10, 2025) Stockholm, Sweden. International Scientific Unity, 2025. P. 57 - 59. Sweden_1_08.01.2025.pdf
ISBN 979-8-89704-992-9 (series)
DOI 10.70286/ISU-08.01.2025

23. Вовчак В. SafeAndAttack: демонстраційний веб-додаток [Електронний ресурс]. – Режим доступу: https://github.com/valentynvovchak/SafeAndAttack