Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Впровадження методів розробки системи безпеки у веб-додатках
(тема)

Виконав:
студент 2 курсу, групи АМСЗІм-20-1
Мазепа А.Д.
(прізвище, ініціали)
Спеціальність: 125 Кібербезпека
(код і повна назва спеціальності)
Тип програми: освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма: Адміністративний менеджмент у сфері захисту інформації
(повна назва освітньої програми)
Керівник: професор кафедри ІКІ ім. В.В.Поповського
Радівілова Т.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Лемешко О.В.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 125 Кібербезпека
(код і повна назва)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітньої програма Адміністративний менеджмент у сфері захисту інформації
(повна назва)

ЗАТВЕРДЖУЮ
Зав. кафедри_____
(підпис)
«____» _____2022р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Мазепі Артему Дмитровичу .
(прізвище, ім'я, по-батькові)

1. Тема роботи: Впровадження методів розробки системи безпеки у веб-додатках
затверджена наказом по університету від «24» березня 2022р. №410 Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 11.05.2022р.
3. Вихідні дані до роботи: методи розробки системи безпеки у веб-додатках, використання провідних рішень безпеки у веб-додатках, методи побудови комплексної системи безпеки для веб-додатку, токени, методи автентифікації
4. Перелік питань, що потрібно опрацювати в роботі:
1) Огляд поточного стану сучасних веб-додатків
2) Аналіз основних загроз та вразливостей у сучасних веб-додатках
3) Побудова комплексної системи безпеки навколо вразливого веб-додатку
4) Аналіз ефективності впровадженої системи безпеки

5. Перелік графічного матеріалу із зазначенням креслень, плакатів, комп'ютерних ілюстрацій: Демонстраційний матеріал у вигляді ppt-презентації;

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		(підпис)	(дата)
Основна частина	професор Радівілова Тамара Анатоліївна		23.02.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	17.01.2022	Виконано
2	Збір матеріалів для дослідження	01.02.2022	Виконано
3	Розробка 1 розділу	07.02.2022	Виконано
4	Розробка 2 розділу	12.02.2022	Виконано
5	Розробка 3 розділу	16.02.2022	Виконано
6	Розробка 4 розділу	23.02.2022	Виконано
7	Оформлення атестаційної роботи	28.04.2022	Виконано

Дата видачі завдання 17 лютого 2022 року

Студент _____ Мазепа А.Д.
(підпис) (прізвище, ініціали)

Керівник роботи _____ професор Радівілова Т.А.
(підпис) (посада, прізвище, ініціали)

Робота не містить відомостей заборонених до відкритого опублікування

Студент **Мазепа А.Д.**

Керівник **Радівілова Т.А.**

РЕФЕРАТ

Пояснювальна записка: 83 с., 45 рис., 7 табл., 10 джерел.

КОМПЛЕКСНА СИСТЕМА БЕЗПЕКИ, МЕТОДИ РОЗРОБКИ СИСТЕМИ БЕЗПЕКИ, ТОКЕН, ВЕБ-ДОДАТОК, ДВУФАКТОРНА АВТЕНТИФІКЦАІЯ.

Об’єкт дослідження – процес розробки системи безпеки для захисту веб-додатку.

Предмет дослідження – методи розробки системи безпеки.

Мета роботи – впровадження методів розробки системи безпеки для реалізації комплексної системи захисту веб-додатку.

Методи досліджень – емпіричний аналіз.

В першому розділі кваліфікаційної роботи був зроблений огляд поточного стану сучасних веб-додатків, та проблем забезпечення належної безпеки для них.

У другому розділі були досліджені основні вразливості, які зловмисники використовують для компрометації сучасних веб-додатків.

У третьому розділ була зроблена якісна оцінка існуючого додатку, який потребує побудови системи безпеки. На основі результатів якісної оцінки був наведений процес побудови цієї системи безпеки, використовуючи провідні методи забезпечення безпеки у веб-додатках.

У четвертому розділі були представлені показники ефективності побудованої системи безпеки з третього розділу.

ABSTRACT

The report contains: 83 p., 45 fig., 7 tables., 10 sources.

COMPLEX SECURITY SYSTEM, METHODS FOR SECURITY SYSTEMS DEVELOPMENT, TOKEN, WEB-APPLICATION, TWO-FACTOR AUTHENTICATION.

The object of research is the process of developing a security system for web-application.

The subject of research - methods for security system development.

An aim of work is an introduction of methods for developing a security system for subsequent creation of a comprehensive system of web-application security.

The method of research is empirical analysis.

The first part of the qualification work reviewed the current state of modern web applications and the problems of ensuring proper security for them.

The second part explores the main vulnerabilities that attackers use to compromise modern web applications.

In the third part, a qualitative assessment of the existing application, which requires the construction of a security system, was performed. Based on the results of the qualitative assessment, the creation process of said security system was described, using the leading security methods in web applications.

In the fourth part, the performance indicators of the constructed security system from the third section were presented.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення
ЦС – центром сертифікації
API – application programming interface
CORS – cross-origin resource sharing
CRM – customer relationship management
CSRF – cross-site request forgery;
CSS – cascading style sheets
DNS – domain name system
DOM – document object model
GPU – graphics processing unit
GTX – giga texel shader extreme
HTML5 – hypertext markup language 5
HTTP – hypertext transfer protocol
HTTPS – hypertext transfer protocol secure
IP – identity provider
JWT – json web token
MD5 – message digest 5
ORM – object relation mapping
OWASP – open web application security project
QR – quick response
SAAS – software as a service
SHA – secure hash algorithm
SPA – single page applications
SQL – structured query language
SSL – secure sockets layer
TLS – transport layer security
URL – uniform resource locator
VPN – virtual private network
XSS – cross-site scripting

ВСТУП

Інтернет став одним з найважливіших відкриттів в сучасному суспільстві. Він вніс значущі зміни у повсякденне життя людей, політику, стосунки, новини, науку, навчання і розваги. Інтернет перевернув наше існування з ніг на голову, та зробив революцію у комунікаційних системах. Тепер він вважається нашим основним засобом для повсякденного спілкування.

Але у всього є і свої мінуси та нюанси, на які треба звернути увагу. Так як люди вже звикли до користування інтернетом на денній основі, велика кількість даних, у тому числі і конфіденційних, постійно циркулюють по мережі. Незважаючи на те, що Інтернет в основному приватний і безпечний канал передачі інформації, він також може бути небезпечним. Проблема в тому, що чим більше розвивається світ технологій, тим більше росте кіберзлочинність. Наприклад, кібератаки – це одна з причин, по якій багато компаній зазнали великих збитків в процесі ведення свого бізнесу. Тільки за минулий рік, 53 відсотки кібератак привели до збитку у розмірі 500 000 доларів і більше. Сама кібератака може зловмисно відключити комп'ютер, вкрасти дані, або використати зламаний комп'ютер в якості точки запуску для інших атак. Кіберзлочинці використовують різні методи для запуску кібератак, у тому числі шкідливі програми, фішинг, програми-вимагачі, атака відмови в обслуговуванні та інші методи.

Нажаль, кіберзлочинність продовжує рости з кожним роком, оскільки люди намагаються знайти все нові і нові способи отримання вигоди з уразливих бізнес-систем. Враховуючи усі ці фактори, інтернет-безпека є головним пріоритетом як для приватних осіб, так і для великих компаній.

Метою цієї роботи є створення комплексної системи захисту для сучасного веб-додатку.

Для виконання поставленої мети, в першому розділі кваліфікаційної роботи був зроблений огляд поточного стану сучасних веб-додатків, та проблем забезпечення належної безпеки для них.

У другому розділі були досліджені основні вразливості, які зловмисники використовують для компрометації сучасних веб-додатків

У третьому розділ була зроблена якісна оцінка існуючого додатку, який потребує побудови системи безпеки. На основі результатів якісної оцінки був наве-

дений процес побудови цієї системи безпеки, використовуючи провідні методи забезпечення безпеки у веб-додатках.

У четвертому розділі були представлені показники ефективності побудованої системи безпеки з третього розділу.

Окремі результати роботи доповідались на Міжнародній науковій конференції [1, 2, 3].

1 ОГЛЯД СУЧАСНИХ ВЕБ-ДОДАТКІВ

1.1 Поточний стан веб-додатків

У ранні часи інтернету, ще не було такого поняття, як веб-додаток. Замість цього люди створювали статичні веб-сайти. Статичний веб-сайт не вимагав веб-програмування або проектування бази даних. Такі сайти вважалися золотим стандартом на протязі тривалого періоду часу, але з подальшим розвитком інтернету, такі сайти стали не дуже практичними у використанні. Підтримка великої кількості статичних сторінок дуже швидко стало непрактичним зайняттям, тому розробники почали шукати нові способи оптимізації процесу. Таким чином і були створені веб-додатки.

Як відомо, веб-додатки – це свого роду комп’ютерні програми. Вони використовують онлайн-технології, такі як браузери, для виконання величезної кількості різних завдань. Зараз, багато з них використовуються для цілей онлайн-рітейлу. Проте, вони можуть служити для самих різних цілей, від замовлення їжі на виніс до проведення грошових операцій. Веб-додатку не обов’язково бути великим, він може бути настільки простим, як контактні форми на веб-сайті або онлайн-калькулятор.

Більш складні веб-додатки зберігають інформацію за допомогою мови програмування, та бази даних на стороні сервера, а скрипти на стороні клієнта витягують, та представляють цю інформацію в призначеному для користувача інтерфейсі. Ці інтерфейси можуть набувати будь-якої форми.

У веб-додатків є безліч переваг. Зокрема, вони допомагають скоротити витрати для підприємств і окремих користувачів. Це пов'язано з тим, що вони вимагають менше обслуговування, а також мають нижчі вимоги до комп'ютерів користувачів з точки зору обчислювальної потужності. Вимоги нижчі тому, що обробка інформації, фактично, відбувається у іншому місці. Також веб-додатки можна запускати у будь якому веб-браузері. Наприклад, Mozilla Firefox, Safari або Google Chrome. Це робить їх доступними для кожного.

Веб-додатки на основі підписки, такі як software as a service (SAAS), також допомагають зменшити піратство програмного забезпечення в Інтернеті. SAAS можна використати тільки через хмару, тому користувачі можуть отримати до нього доступ тільки після того, як заплатять за нього. Такі додатки не треба вста-

новлювати на жорсткий диск, тому що до них можна отримати повний доступ онлайн. Таким чином, користувачі можуть заощадити місце на девайсі і уникнути клопоту стосовно видалення, коли їм починає не вистачати місця, і повторної установки програми. Також немає проблем з сумісністю, оскільки усі користувачі мають доступ до однієї і тієї ж версії додатку.

Як можна побачити, при нинішньому стані веб-розробки, веб-додатки стали дуже популярними. Бізнес більше не може досягти пікового зростання без належного веб-додатку. Вони популярні, тому що грають вирішальну роль в процесі брендінгу. Використовуючи цю технологію, бізнес може підтримувати належний канал зв'язку між потенційними клієнтами і іншими бізнес-організаціями. Веб-додатки також збільшують взаємодію користувача з бізнесом або продуктом, що примушує користувача почувати себе більше залученим і може поліпшити коефіцієнт конверсії, утримання користувачів і отримання доходу.

Веб-додатки тепер стали основним продуктом для великого бізнесу або брендів. Хоча статичні сайти все ще розроблюються по причині дешевизни та швидкості самого процесу розробки, вони поступово витісняються веб-додатками, які все швидше стають провідним рішенням для демонстрації бізнес-можливостей продукту, або компанії.

На рис. 1.1 була наведена діаграма співвідношення кількості веб-додатків до статичних сайтів на 2022 рік [4].

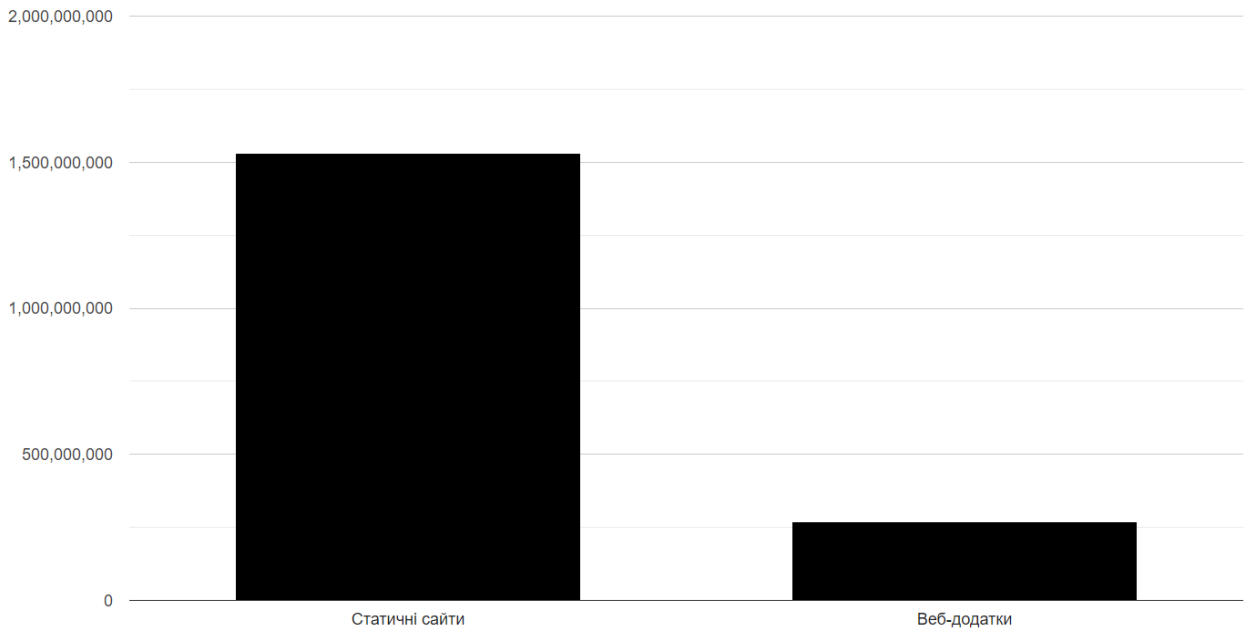

Рисунок 1.1 – Співвідношення кількості веб-додатків до статичних сайтів

Як бачимо з рис. 1.1, на даний момент вже, приблизно, 15 відсотків від усіх веб-сайтів складають веб-додатки, що само по собі вражає своєю масштабність, та, зважаючи на відносну новизну веб-додатків, дає ідею, на скільки швидко іде заміна статичних сайтів на веб-додатки.

1.2 Проблема безпеки сучасних веб-додатків

Незважаючи на всю популярність сучасних веб-додатків, і на те, що вони приносять значущу вигоду для усіх видів бізнесу, можна спостерігати негативний тренд у якості розробки цих самих додатків. Зокрема, у якості розробки частин, які відповідають за безпеку. Через те, що зараз більшість маленьких та середніх бізнесів концентруються на отриманні як можна швидшого і більшого прибутку, вони не приділяють увазі безпеці у своїх веб-додатках.

Ще одною причиною такої ситуації з безпекою є неправильно реалізований код самої програми, який був написаний недосвідченими розробниками. Оскільки зараз професія веб-розробника дуже затребувана, багато хто з нових розробників йде працювати в компанії маючи маленький досвід або без нього взагалі. Але навіть ті розробники, які вже мають доволі великий досвід можуть допустити помилки за елементарною випадковістю або халатністю.

Така тенденція сама по собі наражає на небезпеку не лише конфіденційні дані користувачів, які у більшості випадків неминуче будуть зберігатися у веб-додатку, але ще і репутацію самої компанії, яка зобов’язана відповідати за безпеку даних, які були довірені їй користувачами.

Щоб продемонструвати увесь масштаб проблеми, далі буде наведені статистичні данні по атакам на компанії протягом останніх трьох років.

На рис. 1.2 була наведена діаграма відсотку компаній, які зазнали хоча б одну вдалу атаку за останні три роки [5].

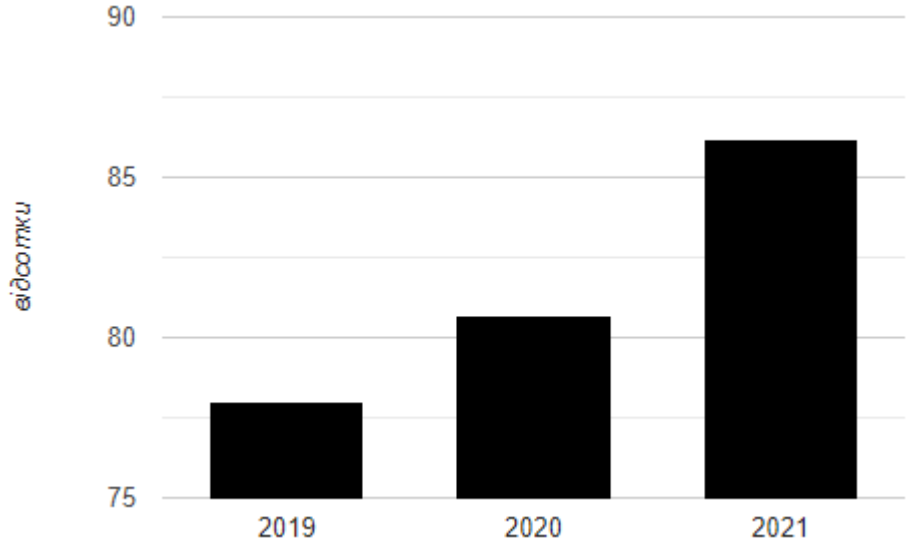

Рисунок 1.2 – Статистика відсотку компаній зазнавших вдалі атаки

З рис. 1.2 можна побачити, що відсоток компаній, які зазнали хоч б одну атаку збільшується з кожним роком на, приблизно, 5 відсотків.

Такий негативний тренд ще раз підкреслює той факт, що багато компаній не приділяють безпеці достатньої уваги. Але навіть ті компанії, що піклуються про своїх клієнтів, та безпеку їх даних, як вже було згадано, не застраховані від неякісно написаного коду, та неправильно зроблених рішеннях при організації системи захисту від розробників.

На рис. 1.3 була наведена діаграма аналізу причин виникнення вразливостей у веб-додатках за 2019 рік [6].

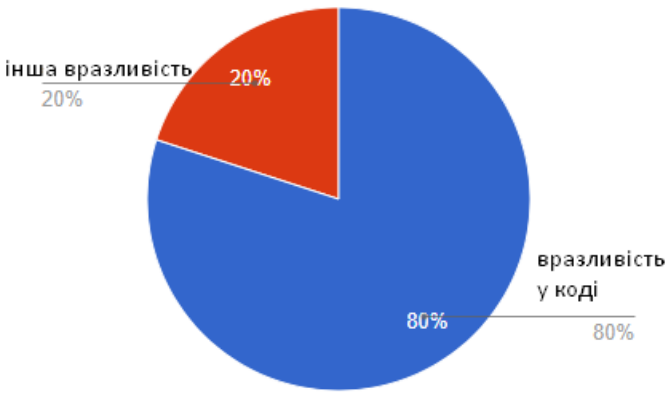

Рисунок 1.3 – Аналіз причин виникнення вразливостей у веб-додатках

Як бачимо з рис. 1.3, приблизно 80% проблем було виявлено в самому коді додатку.

В середньому кількість вразливостей на один веб-додаток досягає 22. Як правило, з них 4 критичних вразливості.

Зрозуміло, що уся ця статистика, та цифри лише приблизні, але навіть так можна побачити велику проблему зі поточним станом безпеки веб-додатків. У частості з написанням якісного, та, насамперед, безпечного коду.

2 ВРАЗЛИВОСТІ У ВЕБ-ДОДАТКАХ

2.1 Основні вразливості у сучасних веб-додатках

Для того щоб прийняти правильні рішення при розробці безпечного веб-додатку, варто спочатку розглянути основні вразливості, про які часто забувають розробники при написанні коду, та загальному створені програми.

Цими вразливостями можна назвати:

- атака structured query language (SQL) injection;
- розкриття конфіденційних даних;
- порушена автентифікація;
- порушення контроль доступу;
- cross-site scripting (XSS);
- cross-site request forgery (CSRF);
- недостатнє логування та моніторинг.

Далі будуть розглянуті усі вразливості приведені вище, та приклади того, як вони уникають у програмі.

2.2 Порушення безпеки додатку атакою SQL injection

SQL – це мова запитів, розроблена для управління даними, що зберігаються в реляційних базах даних. Вона може бути використана для доступу, зміни і видалення даних. Багато веб-додатки зберігають усі дані у базах даних SQL. В деяких випадках команди SQL можуть бути використаними для запуску команд операційної системи. Тому успішна атака SQL injection може мати дуже серйозні наслідки.

SQL injection – це уразливість веб-безпеки, яка дозволяє зловмисникові втручатися в запити, які додаток робить до своєї бази даних. Зазвичай це дозволяє зловмисникові переглядати дані, які він зазвичай не може отримати. Це можуть бути дані, що належать іншим користувачам, або будь-які інші дані, до яких може отримати доступ додаток. У багатьох випадках зловмисник може змінити або видалити ці дані, що приведе до постійних змін вмісту або поведінки додатка. У деяких ситуаціях зловмисник може ескалювати атаку шляхом впровадження SQL

коду, що скомпрометує базовий сервер або іншу внутрішню інфраструктуру. Також він зможе виконати атаку типу "відмова в обслуговуванні".

Уразливість SQL injection може торкнутися будь-якого веб-додатку, що використовує базу даних SQL, наприклад, таку, як MySQL, Oracle, SQL Server або інші. Атаки SQL injection являє собою одну із найстаріших, найбільш поширених і найбільш небезпечних вразливостей веб-додатків.

Як вже було згадано, успішна атака з впровадженням SQL коду може привести до несанкціонованого доступу до конфіденційних даних, таким як паролі, дані кредитної карти або особиста інформація користувача. Багато гучних витоків даних останніми роками були результатом атак з використанням SQL injection, що привело до збитку для репутації і штрафів з боку регулюючих органів. В деяких випадках зловмисник може отримати постійний чорний хід в системах організації, що приведе до довгострокової компрометації, яка може залишатися непоміченою впродовж тривалого періоду часу.

Щоб вчинити атаку SQL injection, зловмисник повинен спочатку знайти уразливі поля вводу призначені для користувача на веб-сторінці або у веб-додатку. Зловмисник може створити вхідний контент через ці поля вводу. Такий контент часто називають шкідливим корисним навантаженням, і він є ключовою частиною атаки. Після того, як зловмисник відправляє цей контент на сервер, у базі даних виконуються шкідливі SQL команди. Далі дії зловмисника можуть різнитися, базуючись на його кінцевій меті.

SQL injection може використовуватися різними способами, для отримання інформації про базу даних, та, у подальшому, викликати серйозні проблеми.

Розглянемо три основні категорії впровадження SQL injection.

1) In-band SQL injection вважається найбільш поширеною і простою у використанні SQL injection. In-band SQL injection відбувається, коли зловмисник може використати один і той же канал зв'язку як для запуску атаки, так і для збору результатів. Одним з методів цього підходу є SQL injection на основі помилок. Цей метод ґрунтований на використанні повідомлень про помилки, що видаються сервером бази даних, для отримання інформації про структуру бази даних. В деяких випадках зловмисникові досить однієї SQL injection на основі помилок, щоб зібрати усю інформацію про базу даних. Хоча помилки дуже корисні на етапі розробки веб-додатків, їх слід відключати на працюючому сайті або замість цього записувати у файл з обмеженим доступом. Іншим методом цієї категорії можна вважати SQL injection на основі об'єднання. Це метод, який використовує оператор SQL

UNION для об'єднання результатів двох або більше операторів SELECT в один результат, який потім повертається як частина відповіді hypertext transfer protocol (HTTP).

2) Inferential SQL injection, на відміну від in-band-SQL injection, може зайняти більше часу у зловмисника, проте вона так само небезпечна, як і будь-яка інша форма SQL injection. При inferential SQL injection атаці ніякі дані фактично не передаються через веб-додаток, і зловмисник не зможе побачити результат атаки усередині каналу. Тому такі атаки зазвичай називають сліпими атаками SQL injection. Замість цього зловмисник може реконструювати структуру бази даних, відправляючи корисні дані, спостерігаючи за відповіддю веб-додатку і результуючою поведінкою сервера бази даних. Одним з методів цього підходу є впровадження SQL injection на основі логічних значень, який ґрунтований на відправці SQL-запиту у базу даних, який примушує додаток повертати різні результати залежно від того, чи повертає запит результат TRUE або FALSE. Залежно від результату вміст відповіді HTTP зміниться або залишиться тим самим. Це дозволяє зловмисникові зробити висновок, чи повернуло використовуване корисне навантаження значення TRUE або FALSE, навіть якщо дані з бази даних не повертаються. Ця атака, як правило, повільна, особливо на великих базах даних, оскільки зловмисникові необхідно перебирати базу даних посимвольно. Іншим методом цієї категорії можна назвати SQL injection на основі часу. Цей метод SQL injection, ґрунтований на відправці SQL-запиту у базу даних, який примушує базу даних чекати певний час у секундах перед відповіддю. Час відгуку вкаже зловмисникові, чи являється результат запиту TRUE або FALSE. Залежно від результату, відповідь HTTP буде повернена із затримкою або повернена негайно. Це дозволяє зловмисникові зробити висновок, чи повернуло використовуване корисне навантаження значення TRUE або FALSE, навіть якщо дані з бази даних не повертаються.

3) Out-of-band SQL injection не дуже поширена, в основному тому, що вона залежить від функцій, включених на сервері бази даних, які використовуються веб-додатком. Out-of-band SQL injection відбувається, коли зловмисник не може використати один і той же канал для запуску атаки і збору результатів. Методи цієї категорії пропонують зловмисникові альтернативу методам, ґрунтованим на часі, особливо якщо відповіді сервера не дуже стабільні, що робить атаку на основі виведення, ґрунтованою на часі, ненадійною. Такі методи покладатимуться на здатність сервера бази даних робити запити domain name system (DNS) або HTTP

для доставки даних зловмисникові. Прикладом може слугувати команда з Microsoft SQL Server xp_dirtree, яку можна використати для виконання DNS-запитів до сервера, контрольованого зловмисником, а також пакет UTL_HTTP Oracle Database, який можна використати для відправки HTTP-запитів з SQL на сервер, контрольований зловмисником.

Як можна побачити, методів впровадження SQL injection дуже багато, але по своїй природі, вони майже всі зав’язані на відправці шкідливих запитів до бази даних серверу. Для повного розуміння даної вразливості, далі буде розглянуто приклад, того, чому така вразливість можлива, та як її використовують зловмисники.

Розглянемо, як зловмисник може використати уразливість SQL injection, щоб обійти безпеку додатка і аутентифікуватися як адміністратор. У даному прикладі буде використаний простий приклад автентифікації за допомогою імені користувача і пароля, використовуючи псевдокод. У прикладі бази даних є таблиця з ім'ям users, у якій є стовпці username та password. Варто зауважити, що на стороні клієнта використовуються звичайні поля вводу без перевірки та серіалізації.

На рис. 2.1 був наведений вразливий, до SQL injection, код серверу, який робить запити до SQL бази даних.

```
....
const username = req.body.username;
const password = req.body.password;

sql = “SELECT id FROM users WHERE username=’” + username + “’ AND password=’” + password + “’”

sqlDB.getByQuery(sql)

....
```

Рисунок 2.1 – Приклад вразливого, до SQL injection, коду сервера

Як можна побачити з рис. 2.1, інформація, яка приходить з полей вводу username та password не сереалізована, а також ці поля без перевірки одразу використовуються для отримання інформації з бази даних. Така реалізація запитів до бази даних уразлива для SQL injection. Зловмисник може використати команди SQL у вхідних даних так, щоб змінити інструкцію SQL, що виконується сервером

бази даних. Наприклад, він може використати підхід з одинарною лапкою і встановити в полі password значення “password’ OR 1=1”.

На рис. 2.2 була наведена результуюча інструкція при зміні значення поля password на “password' OR 1=1”.

```
SELECT id FROM users WHERE username='username' AND password='password' OR 1=1'
```

Рисунок 2.2 – Результуюча інструкція

На рис. 2.2 можна спостерігати, що через оператор “OR 1=1”, значення WHERE повертає перший ідентифікатор з таблиці користувачів незалежно від імені користувача і пароля. Перший ідентифікатор користувача у базі даних дуже часто є адміністратором. Таким чином зловмисник не лише обходить автентифікацію, але і отримує права адміністратора.

Як можна побачити з цього прикладу, що сама по собі SQL injection доволі проста з точки зору логіки виконання, але така атак ні у якому разі не повинна бути недооцінена. Саме через те, що багато розробників забувають про її існування, або просто ігнорують, ця атака все ще знаходиться у топ 10 самих небезпечних веб вразливостей на сьогоднішній день.

2.3 Вразливість розкриття конфіденційних даних

Розкриття конфіденційних даних відбувається, коли додаток, компанія або інша особа ненавмисно розкриває особисті дані. Наприклад, коли розробник помилково завантажує дані в неправильну базу даних. Варто зазначити, що розкриття, та виток конфіденційних даних, являють собою різні речі.

Виток даних – це інцидент безпеки, при якому доступ до інформації здійснюється без авторизації. Хакери шукають особисту інформацію і інші дані, щоб вкрасти гроші, скомпрометувати особу або продати цю інформацію через deep web. Як можна побачити, виток кардинально відрізняється від, частіше за все, випадкової, або помилкової природи розкриття конфіденційних даних.

Окрім випадковості, розкриття конфіденційних даних може відбутися в результаті недостатнього захисту бази даних, в якій зберігається така інформація,

через слабке шифрування, або відсутність шифрування. У деяких випадках через недоліки програмного забезпечення.

Причина розкриття даних може бути пов'язана з тим, як компанія обробляє певну інформацію. Іноді конфіденційні дані можуть зберігатися в текстових документах. Якщо веб-додатки не використовують secure sockets layer (SSL) і не забезпечують безпеку hypertext transfer protocol secure (HTTPS) на веб-сторінках, на яких зберігається інформація, дані також можуть бути схильними до ризику розкриття.

Інші способи розкриття включають:

- зберігання даних у базі даних, яка може бути скомпрометована шляхом впровадження SQL injection або інших типів атак;
- використання слабких криптографічних алгоритмів або ключів;
- відмова від використання методів хешування паролів;
- відмова від використання солі при хешуванні паролів;
- використання інших незахищені сховищ даних.

Потрібно також не забувати про найбільший фактор розкриття конфіденційних даних, яким є людський чинник. Це може бути фішинг, соціальна інженерія або, знову таки, проста випадковість.

До даних, які можуть бути класифікованими як конфіденційні відносяться:

- номери банківських рахунків;
- номера кредитних карт;
- медичні дані;
- токени сеансів;
- номер соціального страхування;
- домашня адреса;
- номери телефонів;
- дати народження і інформація про обліковий запис користувача, такі як імена користувачів і паролі.

2.4 Вразливість порушення автентифікації

Порушена автентифікація зазвичай викликано поганою реалізацією функцій перевірки достовірності і управління сеансами. Атаки порушення автентифікації спрямовані на захоплення однієї або декількох облікових записів, надаючи зловмисникові ті ж привілеї, що і у користувача, що атакується. Автентифікація по-

рушується, коли зловмисники можуть скомпрометувати паролі, ключі сеансу, інформацію про обліковий запис користувача та інші дані, щоб видати себе за цього користувача.

До наслідків успішного порушення автентифікації можна віднести:

- крадіжка важливих бізнес-даних;
- крадіжка особистих даних;
- відправка шахрайських дзвінків або електронних листів;
- створення шкідливих програм для порушення роботи мереж;
- кібертероризм;
- кібесталкінг;
- продаж нелегальних предметів в даркнеті;
- поширення фейкових новин в соціальних мережах.

В останні роки атаки з порушенням автентифікації стали причиною багатьох найсерйозніших витоків даних, і експерти по безпеці б'ють тривогу з приводу цієї недооціненої загрози. У 2017 року open web application security project (OWASP) включив його у свій список 10 найбільших загроз безпеки веб-додатків.

Розглянемо основні фактори ризику, які належать до порушення автентифікації.

1) Передбачувані облікові дані для входу. Під передбачуваними розуміються слабкі паролі, наприклад "12345" або "pass123". Хакер може використати різні методи злому паролів, такі як веселкові таблиці і словники, щоб отримати доступ до системи.

2) Облікові данні користувача, які не захищені при збереженні. Цей пункт більше відноситься до криптографії. Використання слабких методів шифрування, таких як base64, і слабких алгоритмів хешування, таких як secure hash algorithm (SHA) і message digest 5 (MD5), робить облікові дані уразливими. Цей пункт вже згадувався у розділі про розкриття конфіденційних даних, але він вважається одним із основних факторів риску при атаці направленої на порушення автентифікації.

3) Ідентифікатори сеансу, представлені в uniform resource locator (URL). Наприклад, перезапис URL-адресу.

4) Ідентифікатори сеансів уразливі для атак фіксації сеансу.

5) Значення сеансу, яке не витікає по тайм-ауту або не стає недійсним після виходу з системи.

6) Ідентифікатори сеансів, які не міняються після успішного входу в систему.

7) Паролі, ідентифікатори сеансів і інші облікові дані, відправлені по незашифрованих з'єднаннях.

Усі ці чинники наражають на небезпеку веб-додаток, та привертають увагу потенційних зловмисників, які можуть здійснити атаку, та вкрасти конфіденційну інформацію користувачів.

Розглянемо ці атаки у деталях.

1) Перехоплення сеансу. Без відповідних заходів безпеки веб-застосування уразливі для перехоплення сеансу, коли зловмисники використовують вкрадені ідентифікатори сеансу, щоб видавати себе за особу користувача. Найпростіший приклад перехоплення сеансу – це користувач, який забуває вийти з додатка на пристрої, яким потім користується хакер. Далі хакер може просто продовжити сеанс на цьому пристрої.

2) Перезапис URL-адреси ідентифікатора сеансу. Ще одним поширеним способом перехоплення сеансу є перезапис URL. В цьому випадку індивідуальний ідентифікатор сеансу відображається в URL-адресі веб-сайту. Усі, хто може його побачити, можуть підключитися до сеансу. Наприклад, через незахищене з'єднання Wi-Fi.

3) Фіксація сеансу. Вважається варіацією атаки перехоплення сеансу. У цій атаці використовується обмеження в тому, як веб-застосування управляє ідентифікатором сеансу. У тих випадках, коли при автентифікації користувачеві не призначається новий ідентифікатор сеансу, можна реалізувати цю атаку.

4) Credential stuffing – це атака, коли зловмисники дістають доступ до бази даних, заповненої незашифрованими електронними листами і паролями, які вони часто перепродають для використання іншим зловмисниками. Потім ці зловмисники використовують мережі ботів для атак методом грубої сили, які перевіряють ці вкрадені облікові дані з одного сайту, на різних облікових записах інших сайтів. Ця тактика часто працює, тому що люди часто використовують один і той же пароль в різних додатках.

5) Password spraying – це атака, яка трохи схожа на credential stuffing, але замість того, щоб працювати з базою вкрадених облікових даних, вона використовує набір слабких або поширених паролів для злому облікового запису користувача. До таких паролів можна віднести “123456”, “password”, та інші.

6) Фішинг. Зловмисники, як правило, здійснюють фішинг, відправляючи користувачам електронні листи, що видаються за надійне джерело, а потім обманом примушуючи користувача ділитися своїми обліковими даними або іншою пов'язаною інформацією. Це може бути широкомасштабна атака, яка вражає усіх в організації за допомогою одного і того ж фальшивого електронного листа, або вона може набувати форми атаки цільового фішингу, орієнтованої на конкретну мету, або особу. Цільовий фішинг може бути особливо корисний зловмисникам. Вони можуть використати цю техніку атаки, щоб маніпулювати чиїмись емоціями на основі їх особистої інформації.

2.5 Вразливість порушення контролю доступу

Контроль доступу – це накладення обмежень на те, хто може виконувати певні дії, або отримувати доступ до запрошених ресурсів. У контексті вебзастосувань контроль доступу залежить від автентифікації і управління сеансом, які були розглянуті у попередньому розділі [7].

Порушений контроль доступу – це уразливість системи безпеки, що часто зустрічається і є критичною. Розробка і управління засобами контролю доступу – це складна і динамічна проблема, яка накладає ділові, організаційні і юридичні обмеження на технічну реалізацію. Рішення по проектуванню управлінням доступу повинні прийматися людьми, а не технологіями. Саме тому вірогідність помилок дуже висока.

З точки зору користувача сам контроль доступу можна розділити на декілька категорій, які будуть розглянуті далі.

1) Вертикальний контроль доступу – це механізми, які обмежують доступ до конфіденційних функцій, недоступних для інших типів користувачів. Завдяки вертикальному управлінню доступом різні типи користувачів мають доступ до різних функцій додатка. Наприклад, адміністратор може змінити або видалити обліковий запис будь-якого користувача, тоді як звичайний користувач не має доступу до цих функцій. Вертикальні засоби управління доступом можуть бути більш деталізованими реалізаціями моделей безпеки, розробленими для забезпечення дотримання бізнес-політик, таких як розділення обов'язків і мінімальні привілеї.

2) Горизонтальний контроль доступу – це механізми, які обмежують доступ до ресурсів. При горизонтальному управлінні доступом різні користувачі мають

доступ до підмножини ресурсів одного типу. Наприклад, банківське застосування дозволить користувачеві переглядати транзакції і здійснювати платежі зі своїх власних рахунків, але не з рахунків будь-якого іншого користувача.

Самі уразливості порушення контролю доступу існують, коли користувач фактично може отримати доступ до деякого ресурсу або виконати яку-небудь дію, доступ до якого у нього не має бути. Якщо користувач може отримати доступ до функцій, до яких у нього не має бути доступу, то це вертикальне підвищення привілеїв. Наприклад, коли користувач без прав адміністратора може отримати доступ до сторінки адміністратора, де він може видалити облікові записи інших користувачів.

На рис. 2.3 був наведений приклад спроби зловмисника підвищити свої права з клієнта, до адміністратора у веб-додатку з порушеним контролем доступу.

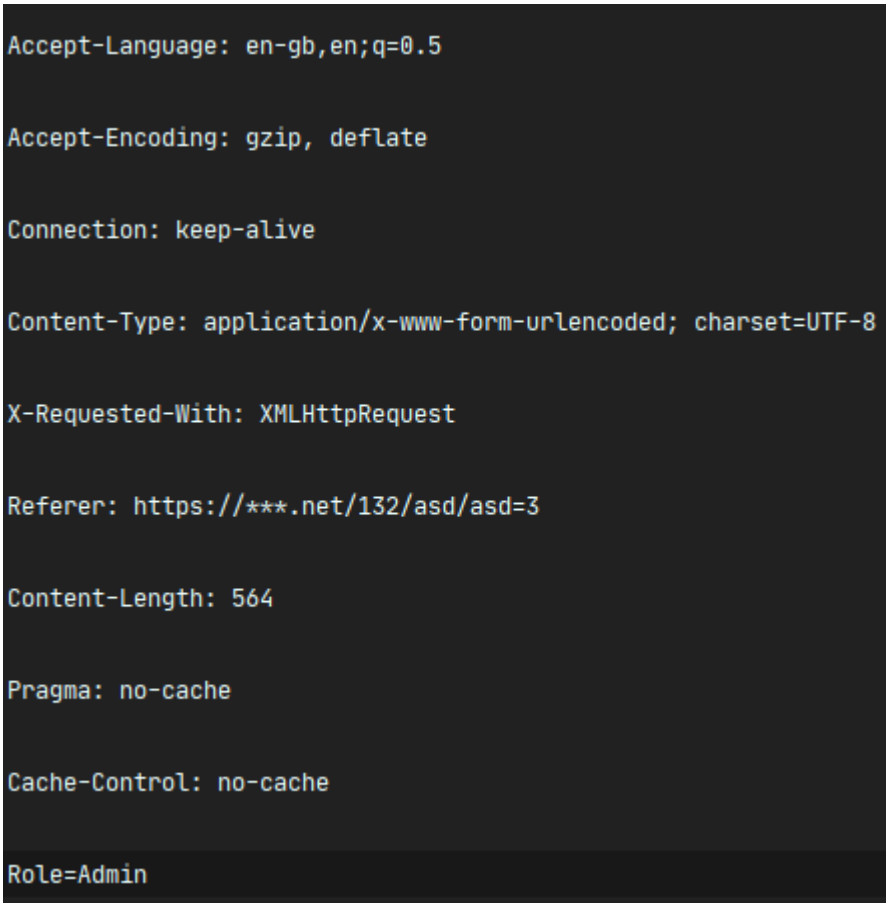

Рисунок 2.3 – Приклад підвищення прав шкідливим HTTP запитом

Як можна побачити з рис. 2.3, змінивши параметр "роль" на “Admin”, зловмисник може підвищити свої привілеї. Це дасть зловмиснику доступ до усіх фун-

кцій, доступних для вибраної ролі, що призводить до вертикального підвищення привілеїв.

Ще один випадок вертикального злому, та неправильного налаштування контролю доступу можна спостерігати у тих випадках, коли деякі додатки застосовують контроль доступу на рівні платформи, обмежуючи доступ до певних URL-адрес і методів HTTP залежно від ролі користувача. Наприклад, додаток може заборонити користувачам видаляти інших користувачів з використанням методу POST, та кінцевою точкою “/admin/deleteUser”. На перший погляд таке рішення виглядає доволі непогано, але іноді деякі платформи підтримують різні нестандартні заголовки HTTP, які можна використати для перевизначення URL-адреси в початковому запиті. Наприклад X-Original-URL і X-Rewrite-URL. Якщо на вебсайті використовуються строгі зовнішні елементи управління для обмеження доступу на основі URL-адреси, але додаток дозволяє перевизначити URL-адресу за допомогою заголовка запиту, то можна обійти елементи управління доступом, використовуючи запит, подібний до “ X-Original-URL: /admin/deleteUser ”.

У свою чергу, горизонтальне підвищення привілеїв виникає, коли користувач може отримати доступ до ресурсів, що належать іншому користувачеві, замість своїх власних ресурсів. Наприклад, якщо співробітник повинен мати доступ тільки до своїх власних записів про зайнятість і заробітну плату, але фактично може також отримати доступ до записів інших співробітників, то це горизонтальне підвищення привілеїв.

Атаки з горизонтальним підвищенням привілеїв можуть використати ті ж типи методів эксплойта, що і вертикальне підвищення привілеїв. Наприклад, зазвичай користувач може отримати доступ до сторінки свого облікового запису, використовуючи URL-адреса, подібна до “https://abc.com/account?id=123”. Тепер, якщо зловмисник змінить значення параметра id на значення іншого користувача, він може отримати доступ до сторінки облікового запису іншого користувача з відповідними даними і функціями.

У деяких застосуваннях такий параметр не має передбачуваного значення. Наприклад, замість зростаючого числа додаток може використати глобальні унікальні ідентифікатори для ідентифікації користувачів. Тут зловмисник може бути не в змозі вгадати або передбачити ідентифікатор іншого користувача. Проте такі ідентифікатори, що належать іншим користувачам, все ще можуть бути розкриті у іншому місці додатку, де згадуються користувачі. Наприклад, в повідомленнях або відгуках користувачів.

В деяких випадках додаток виявляє, що користувачеві не дозволений доступ до ресурсу, і повертає перенаправлення на сторінку входу. Проте відповідь, що містить перенаправлення, може як і раніше містити деякі конфіденційні дані, що належать цільовому користувачеві.

Такі, майже нереалістична на перший погляд, атаки у наш час все ще можуть бути успішно використані, якщо контроль доступу у додатку реалізован неправильно, або не реалізован взагалі. Нажаль у реальному житті, такі випадки все ще можна зустріти на ранніх етапах розробки додатків.

2.6 Порушення безпеки додатку атакою XSS

XSS – це атака з впровадженням коду на стороні клієнта. Зловмисник прагне виконати шкідливий скрипт у веб-браузері жертви, ввівши шкідливий код у клієнтську частину веб-додатку. Фактична атака відбувається, коли жертва відвідує веб-додаток, що виконує шкідливий код. Веб-додатки стають засобом доставки шкідливого script у браузер користувача. Уразливі засоби, які зазвичай використовуються для атак з використанням XSS це веб-форуми, електроні дошки оголошень і веб-додатки, на яких дозволені коментарі [8].

Загалом, веб-додатки уразливі для XSS, якщо вони використовують поля вводу без додаткової перевірки, які призначені для користувача. Ці поля вводу у свою чергу можуть генерувати небезпечні строки тексту. Такий небезпечний текст вводить сам зловмисник. Потім цей текст має бути проаналізований браузером жертви. Атаки XSS можливі у декількох язиках програмування, та навіть у cascading style sheets (CSS), проте, вони найбільш поширені в JavaScript, передусім тому, що JavaScript є основою для більшості таких полів вводу, які являють собою елемент інтерфейсу користувача.

XSS відрізняється від інших векторів веб-атак, тим, що не націлений безпосередньо на саме застосування. Замість цього користувачі веб-додатку піддаються ризику. Залежно від серйозності атаки облікові записи користувачів можуть бути скомпрометовані, активовані троянські програми і змінений вміст сторінок, що вводить користувачів в оману, перенаправляючи браузер на іншу веб-сторінку, що наприклад, містить шкідливий код, який примушує добровільно передавати свої особисті дані. Нарешті, сеансові файли cookie можуть бути розкриті, що дозволить зловмисникові видавати себе за дійсних користувачів і зловживати їх особистими обліковими записами, та привілеями.

Дуже часто уразливості XSS сприймаються як менш небезпечні ніж, наприклад, уразливості SQL injection. Наслідки можливості запуску шкідливої команди JavaScript на веб-сторінці спочатку можуть не здатися такими жахливими. Більшість веб-браузерів запускають JavaScript в дуже строго контрольованому середовищі. JavaScript має обмежений доступ до операційної системи користувача і його файлової системи. Проте, незважаючи на все, JavaScript все ще може бути небезпечним, якщо його використати як частину шкідливого контенту.

Розглянемо чинники, які можуть перетворити функціонал JavaScript у повноцінну атаку.

1) JavaScript має доступ до файлів cookie користувача. Файли cookie часто використовуються для зберігання токенів сеансу. Якщо зловмисник може отримати файл cookie сеансу користувача, він може видати себе за цього користувача, виконувати дії від його імені, та отримати доступ до його конфіденційних даних.

2) JavaScript може читати document object model (DOM) браузеру і вносити в нього довільні зміни, що грає на руку потенційному зловмиснику.

3) JavaScript може використати об'єкт XMLHttpRequest для відправки HTTP-запитів з довільним вмістом в довільні пункти призначення.

4) JavaScript в сучасних браузерах може використати application programming interface (API), який належить до hypertext markup language 5 (HTML5). За допомогою нього, він може отримати доступ до геолокації користувача, вебкамери, мікрофону і навіть певним файлам з файлової системи. Для більшості цих API потрібно згоду користувача, але зловмисник завжди може використати соціальну інженерію, щоб обійти це обмеження.

Усі ці чинники дозволяють злочинцям проводити складні атаки, включаючи впровадження троянів, кейлоггінг, фішинг і крадіжку особистих даних. Уразливості XSS забезпечують ідеальну основу для ескалації атак до серйозніших вразливостей. Міжсайтовий скриптинг також можна використати у поєднанні з іншими типами атак, наприклад, з атакою CSRF.

Існує декілька типів XSS атак:

- stored XSS;
- reflected XSS;
- атака XSS на основі DOM.

Для реалізації Stored XSS зловмисник має запустити шкідливий код через уразливе поле вводу, яке призначене для користувача. Таке поле повинно бути хоча б на одній з сторінок веб-додатку. Потім зловмисник може вставити шкідли-

вий рядок, який використовуватиметься у веб-додатку і розглядатиметься браузером жертви як код скрипту. Деякі типи веб-сайтів більше схильні до таких вразливостей, оскільки вони дозволяють користувачам обмінюватися контентом частіше за інші.

Такими сайтами можна назвати:

- форуми або дошки оголошень;
- сайти блогів;
- соціальні мережі;
- веб-інструменти для спільної роботи;
- веб-системи customer relationship management (CRM);
- веб-консолі сервера електронної пошти і веб-клієнти електронної пошти;
- будь-які сайти з полями для коментарів відвідувачів.

Такий тип XSS не вимагає етапу соціальної інженерії. Жертв цієї атаки не треба заманювати переходом по створеному посиланню. Проте при використанні вразливості stored XSS зловмисники часто намагаються змусити більше жертв відвідати уразливу веб-сторінку.

На рис. 2.4 був наведений приклад спроби зловмисника провести атаку stored XSS у соціальній мережі.

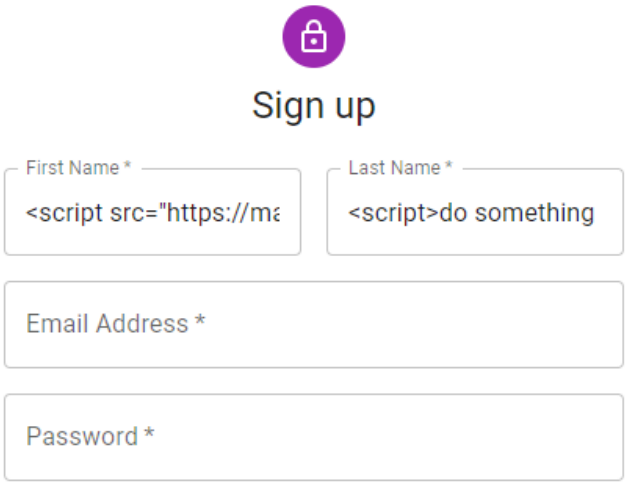

Рисунок 2.4 – Приклад атаки stored XSS у вразливому веб-додатку

Як можна побачити з рис. 2.4, зловмисник може скористатися вразливими полями вводу у соціальній мережі, для впровадження шкідливого коду JavaScript.

Коли інші користувачі соціальної мережі відвідують шкідливий профіль, корисне навантаження доставляється в їх веб-браузер і виконується.

У свою чергу при реалізації reflected XSS шкідливий код виконується браузером жертви, а корисне навантаження ніде не зберігається. Скрипт виконується через посилання, яке відправляє запит на веб-додаток з уразливістю, що дозволяє виконувати шкідливий код. Уразливість зазвичай виникає із-за недостатньої перевірки запитів, що надходять до серверу. Для поширення шкідливого посилання зловмисник зазвичай вбудовує його в електронний лист або на сторонній веб-сайт. Наприклад, в розділ коментарів або в соціальні мережі. Посилання вбудоване в якірний текст, провокує користувача клацнути по ньому, що ініціює XSS-запит до вразливого веб-сайту, відбиваючи атаку назад у браузер користувача.

На відміну від stored XSS, коли зловмисник повинен знайти вразливий веб-сайт, який дозволяє виконувати шкідливі script, reflected XSS вимагають тільки вбудовування шкідливого script у посилання. При цьому для того, щоб атака пройшла успішно, користувачеві необхідно натиснути на це шкідливе посилання.

Крім цього між reflected XSS, та stored XSS, існує ряд ключових відмінностей:

- reflected XSS зустрічаються частіше;
- reflected XSS не мають такого ж охоплення, як stored XSS;
- пильні користувачі можуть уникнути reflected XSS.

З reflected XSS зловмисник намагається відправити шкідливе посилання як можна більшій кількості користувачів, тим самим підвищуючи свої шанси на успішне виконання атаки.

На рис. 2.5 був наведений приклад спроби зловмисника провести атаку reflected XSS через шкідливе посилання.

```
//www.malicious.com/search?value=<script>SomeFunction()</script>
```

Рисунок 2.5 – Приклад атаки reflected XSS через шкідливе посилання

Як можна побачити з рис. 2.5, у посиланні був закладений пошуковий запит з шкідливим значенням.

Останній тип XSS, це XSS на основі DOM у браузері. У цьому типі корисне навантаження самої атаки виконується шляхом зміни DOM у браузері жертви. Це примушує клієнт запускати код без згоди користувача. Сама сторінка не змінить-

ся, але зловмисна зміна в середовищі DOM приведе до того, що клієнтський код, що міститься на сторінці, виконуватиметься по-іншому.

Кроки для виконання XSS атаки на основі DOM виглядають так:

- зловмисник вбудовує шкідливий скрипт в URL-адрес. Наприклад, http://www.asd.com/context=<script>SomeFunction(somevariable)</script>;
- браузер жертви отримує цей URL, відправляє HTTP-запит на http://www.asd.com, і отримує HTML-сторінку;
- браузер починає будувати DOM сторінки і заповнює властивість “document.URL” URL-адресою з кроку 1;
- браузер аналізує HTML-сторінку, знаходить скрипт і запускає його, витягаючи шкідливий вміст з властивості “document.URL”;
- браузер оновлює необроблений HTML-текст сторінки, щоб він містив <script>SomeFunction()</script>;
- браузер знаходить код JavaScript в тілі HTML і виконує його.

Важливо зазначити, що деякі браузери можуть кодувати символи “<” і “>” у URL-адресі, що призводить до збою атаки. Проте є і інші сценарії атаки, в яких ці символи не потрібні.

2.7 Порушення безпеки додатку атакою CSRF

CSRF – це уразливість веб-безпеки, яка дозволяє зловмисникові спонукати користувачів виконувати дії, які вони не мають наміру виконувати. Це дозволяє зловмисникові частково обійти same origin policy, яка призначена для відвертання взаємодії різних веб-сайтів один з одним.

При успішній атаці CSRF зловмисник примушує жертву ненавмисно виконати деякі дії. Наприклад, це може бути зміна адресу електронної пошти в їх обліковому записі, зміна пароля або переказ коштів. Залежно від характеру дії зловмисник може отримати повний контроль над обліковим записом користувача. Якщо скомпрометований користувач має привілейовану роль в додатку, зловмисник може отримати повний контроль над усіма даними і функціями додатка.

Механізми реалізації CSRF-атак практично такі ж, як і для reflected XSS. Як правило, зловмисник розміщує шкідливий HTML-код на контрольованому їм веб-сайті, а потім спонукає жертв відвідати цей веб-сайт. Це можна зробити, надавши користувачеві посилання на веб-сайт по електронній пошті або в повідомленні в соціальних мережах. Якщо атака розміщена на популярному веб-сайті, наприклад,

в коментарях користувачів, вони можуть просто почекати, поки користувачі не відвідають веб-сайт.

Перш ніж вчинити напад, зловмисник зазвичай вивчає додаток.

Розглянемо три ключові умови, які потрібні зловмиснику для реалізації CSRF-атака.

1) У додатку має бути дія, з якої зловмисник отримає вигоду виконав її. Це може бути привілейована дія. Наприклад, зміна дозволів для інших користувачів, або будь-яка дія з даними користувача. Наприклад, зміна власного пароля користувача.

2) Виконання дії включає відправку одного або декількох HTTP-запитів, і додаток покладається виключно на файли cookie сеансу для ідентифікації користувача, що відправив запити. Іншого механізму для відстежування сеансів або перевірки призначених для користувача запитів не існує.

3) Запити, що виконують дію, не містять параметрів, значення яких зловмисник не може визначити або вгадати.

Коли зловмисник зібрав достатню кількість інформації про вразливий додаток, він може взяти структуру реального запиту до додатку, та змінити його, щоб він робив те, що потрібно зловмиснику. Наприклад, переказати 10000 доларів на його рахунок. Далі, такий запит вставляють у посилання, на яке повинна натиснути жертва.

На рис. 2.6 був наведений приклад такого шкідливого посилання, яке зробив зловмисник.

```
<a href="http://asd.com/transfer?account=attackeraccount&amount=$10000">Useful link</a>
```

Рисунок 2.6 – Приклад шкідливого посилання для реалізації CSRF

Як можна побачити з рис. 2.6, у посилання було закладено шкідливий запит. Потім зловмисник може розіслати це посилання по електронній пошті великій кількості клієнтів банку. Ті, хто натисне на посилання, увійшовши до свого банківського рахунку, ненавмисно ініціюють переказ грошей зі свого рахунку.

Якщо веб-сайт банку використовує тільки POST-запити, то у такому разі буде неможливо створити шкідливі запити за допомогою тегу href <a>. Проте є інший спосіб, який використовує тег <form> з автоматичним виконанням вбудованого JavaScript на сайті зловмисника.

Процес самої атаки через тег <form> відбувається наступним чином:

- сторінка зловмисника ініціює HTTP-запит до уразливого веб-додатку шляхом підтвердження скритої на сторінці форми;
- якщо користувач авторизувався на уразливому веб-сайті, його браузер автоматично включить файл cookie сеансу в запит;
- уразливий веб-сайт обробить запит звичайним способом, розцінить його як зроблений користувачем і змінить його адресу електронної пошти.

На рис. 2.7 був наведений приклад такої скритої форми вводу на сторінці зловмисника.

```
<form action="https://asd.com/email/change" method="POST">
    <input type="hidden" name="email" value="asd" />
</form>
<script>
   document.forms[0].submit();
</script>
```

Рисунок 2.7 – Приклад скритої форми для реалізації CSRF

З рис. 2.7 видно, єдине поле вводу у самій формі скрито, та у тегу script є тільки одна інструкція, яка відправляє значення у самому полі до вразливого веб-додатку.

2.8 Наслідки недостатнього логування та моніторингу

Багато серйозних інцидентів безпеки відбуваються із-за недостатнього логування, та моніторингу. Підприємства, що використовують додатки з недостатньою кількістю логування або без нього, ризикують, тим, що атака завдасть набагато більшої шкоди, та потребує набагато більше часу на рішення проблем, які вона створила.

Механізми логування і моніторингу надають адміністраторам і групам безпеки оброблені дані по загальному стану додатку, та трафіку, які допомагають виявляти потенційні загрози, або відхилення від норм у роботі. Ці механізми є основними стовпами безпеки усього додатку. У відсутність ретельно спланованих ме-

ханізмів логування, організація упускає можливості для аналізу безпеки, тим самим дозволяючи атакам мати досить часу для подальшого проникнення в екосистему додатку.

Основні показники недостатнього логування системи:

- незареєстровані події і транзакції;
- відсутні резервні копії журналів;
- прихована реєстрація помилок;
- відсутні плани ескалації порушень;
- погане управління автентифікацією;
- неефективний моніторинг;
- відсутність експорту для аналізу даних логів;
- неправильні налаштування програмного забезпечення.

Без реєстрації критично важливої інформації про безпеку, адміністратори не отримують попереджень про незвичайні події у системі, що перетворює кожну уразливість на потенційну компрометацію усього додатку, що відкриває можливість реалізації подальшої атаки з підвищенням привілеїв. Така ситуація надає значну перевагу зловмиснику.

Отримавши доступ до системи, зловмиснику буде набагато легше приховати свою присутність. У системах, в яких відсутнє комплексне управління журналами, зловмисник навіть може стерти журнали подій, що ще більше ускладнити відстеження вторгнення у систему.

Далі зловмисник може розпочати реалізацію активних атак. Типові активні атаки розпочинаються з того, що хакер аналізує систему на наявність вразливостей в області безпеки. Оскільки час реагування на атаки у системі з недостатнім логуванням і моніторингом може перевищувати 150-200 днів, у зловмисника є досить часу на реалізацію усіх його планів. Зловмисники зазвичай використовують добре відомі стратегії атак, щоб охопити більше території після того, як вони отримали первинний доступ до додатку.

Розглянемо деякі наслідки недостатнього, або неправильного логування і моніторингу.

1) Недоступність системи. Зловмисники, що бажають здійснити атаку типу "відмова в обслуговуванні", зазвичай наводняють цільовий сервер трафіком до тих пір, поки сервер не вийде з ладу або не перестане відповідати. У результаті цієї атаки, сервер буде перевантажено, і його служби стануть недоступними для користувачів. Зловмисники також можуть ініціювати недоступність серверу так,

що атака буде схожа на звичайне перевантаження, що не є зловмисним. Такий вид цієї атаки ще більше ускладнює їх відстеження.

2) Скомпрометована конфіденційність. Журнали подій зазвичай містять конфіденційну інформацію про користувачів і систему. Зловмисники, що мають доступ до системних журналів, мають необмежений доступ до цих даних, які вони можуть використати для інших зловмисних цілей. Неправильне логування і моніторинг дозволяють зловмисникам отримати доступ до приватної інформації, що коштує компаніям грошей і репутації.

3) Зниження цілісності даних. Важко встановити належні засоби контролю для різних етапів життєвого циклу даних, коли відсутні адекватні механізми логування і моніторингу. Зловмисники, що отримали незаконний доступ до системи, можуть легко змінювати дані журналу, змінювати записи і вводити в систему несподівані вхідні дані.

4) Відсутність довіри. Важко довіряти готовності організації до забезпечення безпеки, коли немає можливості відстежувати безпеку користувачів і мережі. Механізми логування і моніторингу служать гарантією того, що усі події, пов'язані з системою, можуть бути відстеженні і перевірені.

3 ВПРОВАДЖЕННЯ МЕТОДІВ ПОБУДОВИ СИСТЕМИ БЕЗПЕКИ ДЛЯ ІСНУЮЧОГО ВЕБ-ДОДАТКУ

3.1 Загальний огляд існуючого веб-додатку

Існуючий веб-додаток представляє з себе типовий інтернет магазин, який продає електроні товари різних видів.

По типу додатку клієнтська частина являє собою single page applications (SPA). В даному випадку вибір такого типу додатку було оправдано тим, що вимоги самого додатку включають багату функціональність, що виходить за рамки того, що пропонують звичайні HTML-форми.

Перелік технологій які були використані при розробці клієнтської частини додатка були наведені у таблиці 3.1.

Таблиця 3.1 – Перелік технологій, використаних на клієнтській частині

Назва	Опис технології
React	Бібліотека яка використовується як основа для створення SPA додатків
MobX	Бібліотека яка дозволяє клієнтській частині зберігати дані в централізованому сховищі. Таке сховище спрощує розробку додатку
MobX-router	Бібліотека яка підключається до централізованого сховища MobX, та надає легкий спосіб навігації по додатку
MobX-react	Бібліотека яка робить можливим роботу react, та MobX у комбінації
antd	Бібліотека яка надає велику кількість готових рішень для реалізації візуальної частини додатку
axios	Бібліотека для полегшення роботи з мережевими запитами
webpack	Бібліотека для полегшення загальної розробки, та ефективного збору коду клієнтської частини для відправлення її до хостінгу

Серверна частина цього додатку використовує середовище розробки nodejs, з невеликою допоміжною бібліотекою express. Таке рішення було зроблено тому, що серверна частина не включає складного функціоналу у роботі з даними, які приходять на сервер.

Перелік технологій які були використані при розробці серверної частини додатка були наведені у таблиці 3.2.

Таблиця 3.2 – Перелік технологій, використаних на серверній частині

Назва	Опис технології
express	Бібліотека яка використовується як основа для побудування серверної частини
body-parser	Бібліотека потрібна для обробки, та розуміння даних які поступають до серверу
axios	Бібліотека для полегшення роботи з мережевими запитами
http-error	Бібліотека для стандартизованого створення http помилок
moment	Бібліотека для стандартизованої роботи з датами
lodash	Бібліотека яка надає велику кількість корисних функцій для роботи з різними типами даних
dotenv	Допоміжна бібліотека для обробки файлів з розширенням .env
PayPal-node	Бібліотека яка взаємодіє з сервісом грошових транзакцій PayPal

Розглядаємий додаток включає у себе базову реалізацію стандартного функціоналу сучасного веб-додатку.

До такого функціоналу можна віднести:

- механізм автентифікації;
- механізм контролю доступу з розділенням на типи користувача клієнта та адміністратора;
- механізм роботи з базою даних SQL;
- механізм роботи з сервісом paypal.

Через те, що зазначений вище функціонал реалізований на базовому рівні, загальна безпека додатку не відповідає сучасним нормам.

Для представлення повної картини усіх проблем з безпекою, був зроблений якісний аналіз розглядаємого додатку. Усі результати були наведені у таблиці 3.3.

Таблиця 3.3 – Якісна оцінка розглядаємого додатку

Назва ризику	Тип вразливості	Збиток	Вірогідність виникнення	Результуючий ризик
Відправлення шкідливого коду на сервер з клієнта через незахищені поля вводу	SQL injection	Дуже високий	Висока	Високий
Отримання, та обробки сервером шкідливого коду	SQL injection	Дуже високий	Дуже висока	Дуже високий
Злому паролю	Порушення автентифікації	Високий	Дуже висока	Дуже високий
Викрадення токену сесії	Порушення автентифікації	Високий	Дуже висока	Дуже висока
Розкриття паролів при отриманні несанкціонованого доступу до бази даних	Розкриття конфіденційних даних	Дуже високий	Дуже висока	Дуже високий
Викрадення конфіденційної інформації при перехоплені трафіку	Розкриття конфіденційних даних	Дуже високий	Дуже висока	Дуже високий
Отримання несанкціонованого доступу до облікового запису адміністратора	Порушення контролю доступу	Високий	Висока	Високий
Отримання несанкціонованого доступу до функцій адміністратора	Порушення контролю доступу	Високий	Висока	Високий
Введення шкідливого коду у клієнтську частину додатку	XSS	Високий	Низька	Середня
Вразливість відправлення конфіденційних даних через веб-сайт зловмисника	CSRF	Високий	Середня	Високий
Несвоєчасне знаходження вразливостей у системі	Недостатнє логування та моніторингу	Дуже високий	Дуже висока	Дуже високий

Як можна побачити з таблиці 3.3, розглядаємий додаток містить велику кількість серйозний вразливостей, шанси на виникнення яких дуже високі.

Саме тому, навколо цього додатку буде побудовано систему безпеки, яка використовує провідні методи боротьби з такими вразливостями. Така система зведе ризик нападу зловмисників до мінімуму, та збільшить стійкість додатку, у тих випадках, коли така атака була зроблена.

3.2 Усунення вразливості SQL injection

У розглядаємому додатку є дві проблеми, які можуть призвести до успішної атаки SQL injection.

До цих проблем можна віднести такі проблеми:

- поля вводу на стороні клієнта не перевіряються перед відправкою до серверу;
- при зверненні до бази даних, параметри, які прийшли з клієнта вставляються у запит бази даних без додаткових мір захисту, або перевірки на стороні серверу.

Спочатку розглянемо проблему зі сторони клієнту.

На рис. 3.1 був наведений один з сегментів додатку у якому в полях введення не відбувається перевірки введених значень.

```
export function RegistrationForm() {
    const sendData = () => {
        .... sending data logic
    };

    return (
        <form onSubmit={sendData}>
            <input type="text" id="email" name="email" value="Asd" />
            <input type="text" id="password" name="password" value="Zxc" />
            <input type="text" id="repeatPassword" name="repeatPassword" value="Zxc" />
            <input type="text" id="country" name="country" value="Ukraine" />
        </form>
    );
}
```

Рисунок 3.1 – Форма реєстрації у якій не відбувається перевірка даних

Як можна побачити з рис. 3.1, у даній формі є тільки одна функція, яка відповідає за відправлення даних до серверу. Функція перевірки полів відсутня. Немає навіть мінімальної перевірки, яка досягається шляхом додавання атрибутів потрібного типу до рідних полів вводу. У даному випадку усі поля вводу мають тип text.

Така ситуація наражає на небезпеку тим, що потенційний зловмисник може відправити шкідливий запит через ці поля, та скомпрометувати весь додаток.

Цю ситуацію можна виправити декількома способами, які включають:

- використання перевірочної функції;
- regex вирази;
- використання зумовлених значень які не дадуть ввести своє значення.

Кожен з цих способів маже бути використаний незалежно один від одного, або у комбінації. Розглянемо перший спосіб.

Перевірочна функція – це функція у якій процес перевірки спрямований на перевірку того, чи дозволено введення того чи іншого значення, яке було представлено користувачем. Перевірка введення гарантує, що у поле був введений допустимий тип, довжина або формат даних. Тільки значення, що пройшло перевірку, може бути оброблене. Це допомагає протидіяти будь-яким шкідливим командам, які були вставлені у поле введення.

Наступний метод дуже часто використовується у комбінації з першим. Regex вирази – це рядок тексту, що дозволяє створювати шаблони, що допомагають знаходити та управляти текстом у документі або строчці. Кажучи про усунення SQL injection, regex може допомогти у знаходженні і розпізнаванні шкідливих команд, які намагається ввести зловмисник.

Останній спосіб буде корисним у разі, коли можна перетворити строчку вводу у фіксований набор значень. Наприклад, реалізувати випадаючий список або кнопки перемикачі, які будуть визначати, які значення будуть повертатися.

У даному випадку, буде використано усі три способи у комбінації, що забезпечить максимальний захист.

Перш за все, потрібно убезпечити поля введення, які призначені для користувачів типу клієнт тому, що з цієї сторони є більший шанс зазнати атаку від зловмисників. Далі будуть захищені поля для адміністраторів, що зменшить шанс на випадкову помилку при введені динних.

Кожне поле буде перевірятися по різним критеріям. Наприклад, для поля email, неможливо буде написати значення, яке не містить типові символи, які зу-

стрічаються у назві поштової скриньки. При введені неправильного значення, буде відображатися помилка, яка буде говорити про неправильність вводу для користувача.

Сама перевірка буде використовувати функцію перевірки з regex виразами, а усі поля, які можна згрупувати, будуть перетворені на випадаючі списки.

На рис. 3.2 був наведений проблемний сегмент з попереднього рисунку, до якого було додано описані вище покращення.

```
const sendData = () => {
    if(validateForm()) {
         .... sending data logic
    }

};
function validateForm() {
    let email = document.forms["form"]["email"].value;
    let password = document.forms["form"]["password"].value;
    let repeatPassword = document.forms["form"]["repeatPassword"].value;
    const re = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]+

    if (email == "") {
        alert("Email must be filled out");

        return false;
    } else if (
    !re.test(String(email).toLowerCase()
    && !re.test(String(password).toLowerCase()
    && !re.test(String(repeatPassword).toLowerCase()) == false
    ) {
        alert("Email must be valid");

        return false;
    }
    return true;
}

return (
    <form onSubmit={sendData}>
        <input type="email" id="email" name="email" value="Asd" />
        <input type="password" id="password" name="password" value="Zxc" />
        <input type="password" id="repeatPassword" name="repeatPassword" value="Zxc" /
        <select>
           <option value="Afghanistan">Afghanistan</option>
           <option value="Albania">Albania</option>
           <option value="Algeria">Algeria</option>
```

Рисунок 3.2 – Форма реєстрації з реалізованими перевірками

Як можна побачити з рис. 3.2, тепер функція відправки даних спочатку перевіряє данні у формі, на небажані символи шляхом виклику перевірочної функції. У самій функції, ця перевірка відбувається через використання шаблону regex. Якщо email, або паролі не пройшли перевірку regex, то данні на сервер відправлятися не будуть. У самій формі, поле для вводу країн було замінено на випадаючий список, який не потребує прямого вводу від користувача. Можна також відмітити, що типи самих полів вводу були замінені на типи, які підходять кожному компоненту.

Ця перевірка стане першим рівнем захисту від зловмисників, які намагаються перевірити клієнтську частину на наявність вразливостей для запуску SQL injection. Зрозуміло, що такий захист не зупинить більш досвідчених зловмисників, тому також потрібно захистити і серверну частину.

Розглянемо проблему на стороні серверу.

На рис. 3.3 була наведена вразлива частина додатку, яка відповідає за реєстрацію нового користувача у системі.

```
app.post('/api/auth/registration', async (req, res) => {
    const {email, password, country} = req.params;
    const existingEmail = await db.query(`SELECT * FROM users WHERE email='${email}'`);

    if(existingEmail) {
        return res.sendStatus(400);
    }
    ...
})
```

Рисунок 3.3 – Вразлива до SQL injection частина додатку на стороні сервера

Як можна побачити з рис. 3.3, поштова скринька, яка приходить до серверу перевіряється на те, чи існує вже така у системі. На перший погляд усе нормально, але проблема скривається у передачі параметра email, до запиту бази даних напряму. Така реалізація дуже ненадійна. Через те, що параметр одразу потрапляє у запит без перевірки, дуже легко організувати SQL injection шляхом відправки шкідливої команди замість нормально вводу.

Таку вразливість у коді можна виправити наступними методами:

- використання так званого query placeholder;

- використання object relation mapping (ORM), яка сама буде слідкувати за небажаними параметрами.

У першому випадку, нам потрібно додати query placeholder, до самого запиту. Query placeholder – це технологія, яка дозволяє відмічати, де в запиті треба вставити значення з параметра. При такому підході значення, які надходять до запиту будуть безпечно вставлені у сам запит з додатковою перевіркою.

У другому випадку, можна встановити ORM. ORM це спеціальна програма, яка взаємодіє з базою даних через набір функцій та схем. Загалом, такі функції доволі безпечні, та не потребують додаткового захисту, але якщо використовувати ORM, то про всяк випадок, перед відправленням значення краще додатково перевірити його самостійно.

Для виправлення вразливості на сервері, було вибрано перший варіант, оскільки, сам варіант доволі простий та ефективний. Також він не потребує встановлення сторонніх бібліотек, та додаткових перевірок.

На рис. 3.4 була наведена реалізація мір безпеки для усунення вразливості SQL injection на стороні серверу.

```
app.post('/api/auth/registration', async (req, res) => {
    const {email, password, country} = req.params;
    const existingEmail = await db.query(`SELECT * FROM users WHERE email=?`, [email]);

    if(existingEmail) {
        return res.sendStatus(400);
    }
    ...
})
```

Рисунок 3.4 – Частина додатку з усуненою вразливість SQL injection

Як можна побачити з рис. 3.4, тепер запит до бази даних використовує query placeholder, що заміняє параметр email на знак запитання. Як другий параметр, запит приймає саме значення email, що буде підставлена замість знаку питання, коли запит почне виконуватися. Сама вставка безпечна, адже SQL слідкує за тим, щоб у значенні не було шкідливого коду, який може призвести до успішного виконання SQL injection.

Так усунення вразливості буде застосовано до всіх функцій цього додатку, де значення вставляються прямо у запит бази даних.

Таким чином додаток був захищений від SQL injection вразливості.

3.3 Усунення вразливості порушення автентифікації

На даний момент у додатку присутня такі проблеми за автентифікаціє:

- відсутня політика створення паролів;
- немає протидії атаці підбору паролів грубою силою;
- токени сесії передаються у запиті;
- не змінює id-сесії при повторному вході;
- неправильне інвалідування id сесії при виході з додатку;
- відсутня двофакторна автентифікація.

Почнемо розбір цих проблем з політики паролів.

Наявність політики паролів являє собою один з найважливіших компонентів системи автентифікації. У даному випадку політика паролів відсутня, тому користувачі можуть вводити не надійні паролі для реєстрації. Наприклад, користувач може ввести такі паролі, як password, 12345678, qwerty. Такі паролі дуже легко підібрати простим перебором, або атакою за допомогою словника. Саме тому потрібно створити політику паролів.

У цю політику будуть включатися такі правила:

- пароль не може бути меншим ніж 8 символів. Чим більше символів, тим краще;
- суміш прописних і рядкових букв;
- суміш букв і цифр;
- включення хоч б одного спеціального символу.

Для перевірки ефективності, далі буде розрахована ентропія мінімального доступного паролю з цими правилами.

Для розрахунку ентропії паролів буде використана така формула [9]:

$$e = L \cdot \frac{\log(C)}{\log(2)}, \tag{3.1}$$

де L – мінімальна довжина паролю;

C – розмір набору символів для нової політики паролів.

Підставляємо потрібні значення та розраховуємо ентропію мінімального паролю використовуючи формулу (3.1):

$$e = 8 \cdot \frac{\log(94)}{\log(2)} = 5{,}243$$

Як правило, чим більше виходить значення ентропії, тим складніше буде зловмиснику підібрати пароль. Як можна побачити, у даному випадку значення дорівнює 5,243 бітів на символ, що вважається сильним результатом.

Для перевірки надійності паролів, буде створена нова перевірочна функція на сторінці реєстрації, яка буде перевіряти кожен пароль на дотримання правил створення. Ті паролі, які не будуть відповідати поставленим правилам, будуть відкидатися, а сам запит до сервера надходити не буде. Такий підхід змусить користувачів придумувати надійніші паролі.

На рис. 3.5 був наведений сегмент коду з форми реєстрації, у який було добавлено перевірку пароля надійність.

```
    alert("Email must be valid");

    return false;
}

const isPasswordValid = validatePassword(password);

if(!isPasswordValid) {

    return false;
 }

return true;
```

Рисунок 3.5 – Реалізація перевірки пароля на формі реєстрації

Як можна побачити з рис. 3.5, тепер данні на формі реєстрації не будуть відіслані, якщо правила паролів не були дотримані.

Перевірки пароля на стороні клієнта не вистачить, тому, що ненадійний пароль все ще можна відправити напряму до серверу, минаючи інтерфейс користувача. Саме тому, необхідно реалізувати перевірку на стороні серверу. Досягти цього можна декількома способами.

Цими способами є:

- створення ще однієї перевірочної функції на стороні сервера;
- якщо використовується ORM, то можна налаштувати схему даних на збереження тільки надійних паролів;
- скористатися спеціальною бібліотекою, яка сама зробить перевірку.

У даному випадку, кращим варіантом буде реалізувати ще одну функцію перевірки. Так як пароль перевіряється тільки при створенні нового користувача, перевірка через ORM буде зайвою, так само як і використання сторонніх бібліотек.

Для перевірки самих паролів у функції буде використовуватися regex вирази, так само, як і у випадку з перевіркою на SQL injection.

На рис. 3.6 був наведений сегмент коду з серверної частини, де перед збереженням нового користувача до бази даних, перевіряється його пароль на надійність.

```
app.post('/api/auth/registration', async (req, res) => {
    const {email, password, country} = req.params;
    const isPasswordValid = validatePassword(password);

        if(!isPasswordValid) {

            return res.sendStatus(400);;
         }
```

Рисунок 3.6 – Перевірка паролю на надійність з серверної сторони

Як можна побачити з рис. 3.6, тепер данні на стороні серверу теж перевіряються, та усі ненадійні паролі відкидаються, відправляючи помилку з кодом 400.

Тепер політика паролів налаштована правильно і користувачі не зможуть придумати собі ненадійні паролі.

Нажаль, обмежитись тільки політикою паролів не вийде. Оскілки зараз у додатку не встановлено ніяких додаткових мір захисту проти атаки грубої сили, ніщо не заважає зловмисникові намагатися підібрати пароль стільки разів, скільки йому буде потрібно. Саме тому, наступним кроком до поліпшення системи автентифікації буде встановлення захисту від атаки грубою силою.

Такий захист можна реалізувати за допомогою так званої техніки rate limiting. Ця техніка використовується для управління потоком даних на сервер і з серверу шляхом обмеження кількості взаємодій користувача з API. Це не дозволяє одному користувачеві використати надто багато ресурсів серверу.

Щоб контролювати цей потік даних, нам спочатку треба визначити критерії користувачів, за якими треба буде блокувати.

Розглянемо ідентифікатори, які використовуються для цього завдання.

1) Ідентифікатор користувача. Ідентифікатор користувача це найбільш ефективний варіант для серверу, який авторизує користувачів і привласнює їм унікальні ідентифікатори. Будь-яке унікальне значення може служити ідентифікатором користувача, включаючи закриті та секретні ключі, а також імена користувачів.

2) Identity provider (IP) адреса. Цей метод працює краще всього, якщо у додатку немає іншого способу однозначно ідентифікувати користувача. Хоча в деяких випадках це працює дуже добре, потрібно бути обережним, щоб випадково не заблокувати користувачів із загальними IP-адресами.

3) Геолокація. Через те, що більшість атак надходять з декількох країн. Ці країни можуть бути ізольовані або заблоковані, не зачіпаючи користувачів з інших регіонів та країн.

Для розглядаємого додатку, ідеальним варіантом буде слідкувати за id користувача, оскільки до усіх записів у базі даних присвоюється унікальний ідентифікатор, у тому числі і до кожного запису користувача. Як вже було згадано, блокування по IP може зачепити користувачів з загальними IP-адресами, а у випадку з геолокацією немає гарантії, що зловмисник не буде використовувати virtual private network (VPN), щоб обійти цей механізм захисту.

Для реалізації захисту від атаки грубою силою можна скористатися наступними методами:

- реалізувати один з алгоритмів перевірки, на кількість входів у систему;

- використати готову бібліотеку для перевірки на кількість входів.

У данному випадку реалізація такого алгоритм з нуля буде зайвою, тому краще використати вже готову бібліотеку для цього.

Бібліотека, яка буде використовуватися називається express-brute. Ця бібліотека дозволяє визначити час та кількість запитів, які розраховані на одного користувача при спробі авторизації. Таким чином на сервер було добавлено нову частину коду.

На рис. 3.7 була наведена ця частина коду з реалізацією мір безпеки для усунення можливості скоєння атаки шляхом грубої сили на стороні серверу.

```
module.exports = new ExpressBrute(store, {
    freeRetries: 5,
    minWait: ms('10s'),
    maxWait: ms('1 hour'),
    handleStoreError
});
```

Рисунок 3.7 – Частина додатку з усуненням шансу нанесення атаки грубої сили

Як можна побачити з рис. 3.7, на сервері були введені обмеження на кількість спроб авторизації.

Поля, які прописані у даній функції означають наступне:

- freeRetries - кількість спроб, яку користувач робить до того, як він потрапить в чорний список;
- minWait - час, який користувач повинен чекати, коли у нього закінчуються повторні спроби вперше;
- maxWait - максимальна кількість часу, яку користувачеві доведеться чекати;
- handleStoreError – функція обробки помилок.

Це обмеження працює наступним чином. Користувач зможе зробити тільки 5 невдалих спроб автентифікації. Далі йому доведеться зачекати 10 секунд, щоб спробувати знову. Після ще одних 5 невдалих спроб, час очікування збільшиться до 20 секунд и так далі до 1 часу.

Варто зазначити, що такі параметри було вибрано для тесту даної функції. Перед випуском додатку у реальне оточення, треба змінити ці параметри на 3 спроби з мінімальним інтервалом 1 час, та максимальним 1 рік.

Такі обмеження на спроби входу не гарантують, що у зловмисника не вийде підібрати пароль на якусь з спроб, але шанси на такий розклад з оновленою політику паролів мінімальний.

Наступна проблема, яку треба розглянути, це реалізація безпечної системи управління сесіями.

На даний момент ця система не відповідає стандартам безпеки, адже має такі проблеми:

- токени сесії передаються у запиті;
- не змінюється id сесії при повторному вході;
- неправильне інвалідування id сесії при виході з додатку.

Через те, що кількість проблем та вразливостей перевищує усі допустимі межі, було вирішено переробити цю частину додатку з нуля.

Для реалізації такої системи за основу було взято технологію або json web token (JWT). Такий вибір було оправдано надійністю самої технології, та гнучкістю з якою можна її інтегрувати.

JWT являє собою відкритий стандарт, використовуваний для обміну інформацією між клієнтом і сервером. JWT відрізняються від інших веб-токенів тим, що містять набір тверджень. Твердження використовуються для передачі інформації між двома сторонами. Що є цими твердженнями, залежить від варіанту використання. Наприклад, в твердженні може бути вказано, хто видав токен, як довго він дійсний, або які дозволи були надані користувачеві. JWT підписуються з використанням криптографічного алгоритму, щоб гарантувати, що твердження не можуть бути змінені після випуску токену. Сам підпис – це хеш кодованого заголовка, закодованого корисного навантаження та "секретний" ключ, який безпечно зберігається на сервері, використовуючи алгоритм, визначений у заголовку [1].

Розглянемо види токенів, які будуть включені у реалізацію системи сесій.

1) Access токен, який містить усю інформацію, яку повинен знати сервер, чи може користувач отримати доступ до запитуваного ресурсу або ні. Зазвичай це токен з коротким терміном дії.

2) Refresh токен, який використовується для створення нового access токена. Як правило, якщо у access токена є дата закінчення терміну дії, після закінчення цього терміну користувачеві доведеться знову пройти автентифікацію, щоб отри-

мати новий access токен. З refresh токеном цей крок можна пропустити і запитом до API отримати новий access токен, який дозволяє користувачеві продовжити доступ до ресурсів додатка.

Ці токени будуть використовуватися у комбінації один з одним, щоб сформувати нову та безпечну систему управління сесіями.

Сама система буде складатися з 3 частин, які являють собою:

- реалізація видачі токенів при успішній автентифікації;
- перевірка токенів при спробі доступу до конфіденційних даних або інтерфейсу взаємодії з сервером;
- інвалідування токенів при виході з додатку.

Перша частини буде реалізована при авторизації до платформи. Коли користувач введе свій логін і пароль, йому, після перевірки на автентичність, буде видано access та refresh токени.

На рис. 3.8 був наведений код з реалізацією видачі двух JWT при успішній автентифікації у додаток.

```
const body = req.body;
const user = {
    "email": body.email,
}

const token = jwt.sign(user, config.accessTokenSecret, { expiresIn: config.tokenExpiration})
const refreshToken = jwt.sign(user, config.refreshTokenSecret, { expiresIn: config.refreshTokenExpiration})
const response = {
    "status": "Logged in",
    "token": token,
    "refreshToken": refreshToken,
}
res.status(200).json(response);
```

Рисунок 3.8 – Перша частина реалізації нової системи управління сесіями

Як можна побачити з рис. 3.8, після вдалої автентифікації користувача, сервер бере його email, та закодовує його у обидва JWT. По цій інформації можна однозначно ідентифікувати користувача при перевірці токену, так як його контент змінити неможливо, через електронний підпис, яким був підписаний даний токен.

Обидва токени використовують секретний рядок тексту для генерації електронного підпису. Для кожного з них рядок різний.

Далі встановлюється час на життя токенів. У даному випадку access токен дійсний тільки 5 хвилин, а refresh протягом 1 дня. Це означає, що користувачеві потрібно буде автентифікуватись тільки раз у день.

Нарешті, коли усі токени були створені, вони відправляються до клієнту, який буде зберігати їх.

Зберігатися на стороні клієнту вони можуть у таких місцях:

- local storage;
- session storage;
- cookie storage.

У даному випадку було вибрано cookie storage. Це оправдано тим, що local storage та session storage вразливі до атаки XSS. Обов’язково використовувати cookie файли з параметрами httpOnly, та secure. Параметр secure не дозволить браузеру відправити cookie файли по відкритому HTTP. Тільки ті cookie, які будуть відправлятися по HTTPS будуть пропускатися. Доповнюючи захист cookie файлів, параметр httpOnly зробить неможливим усі спроби звернутися до файлів cookie через скрипти у браузері. Таке обмеження не дозволить провести атаку XSS, що робить cookie storage самим надійним місцем для зберігання токенів.

На рис. 3.9 була наведена частина коду з серверу, де токени поміщаються у cookie файли.

```
const body = req.body;
const user = {
    "email": body.email,
}

const token = jwt.sign(user, config.accessTokenSecret, { expiresIn: config.tokenExpiration})
const refreshToken = jwt.sign(user, config.refreshTokenSecret, { expiresIn: config.refreshTokenExpiration})
res.cookie('token', token, { httpOnly: true });
res.cookie('refreshToken', refreshToken, { httpOnly: true, secure: true })
res.status(200);
```

Рисунок 3.9 – Створення cookie файлів з токенами на стороні серверу

Як можна побачити з рис. 3.9, код автентифікації було відредаговано, і тепер замість токенів у тілі відповіді, вони прикріпляються до cookie. Також видно, що параметри httpOnly та secure також були виставлені.

Після того, як ці токени прийшли з серверу, клієнт буде зберігати їх у cookie файлах, та відправляти їх з кожним запитом назад до серверу.

Потім сервер буде перевіряти ці токени, які надійшли у cookie файлах з клієнту, та вирішувати, чи надавати доступ до того чи іншого ресурсу.

На рис. 3.10 була наведена частина коду з реалізацією перевірки токенів на сервері.

```
if (token) {
    jwt.verify(token, config.accessTokenSecret, function(err, decoded) {
        if (err) {
            return res.status(401).json({"error": true, "message": 'Unauthorized access.' });
        }
        req.decoded = decoded;
        next();
    });
} else {
    return res.status(403).send({
        "error": true,
        "message": 'No token provided.'
    });
}
```

Рисунок 3.10 – Реалізацією коду перевірки токенів на сервері

Як можна побачити з рис. 3.10, перш за все, токен дешифрується. Якщо дешифровка пройшла успішно, то користувачеві дається доступ до запитуваного ресурсу або функції. У іншому випадку, токен відкидається, та до користувача надходить повідомлення з помилкою.

Тепер залишилося лише розрішити проблему з виходом системи. Для правильного виходу системи, усі токени користувача повинно бути інвалідовано.

На рис. 3.11 було наведена частина коду з реалізацією інвалідування токенів користувача.

```
jwt.destroy(token)
jwt.destroy(refreshToken)
res.status(200);
```

Рисунок 3.11 – Інвалідування токенів користувача

Як можна побачити з рис. 3.11, токени, які були видані сервером знищуються, а у якості відповіді відправляється статус 200, що означає вдало виконану операцію. Тепер, коли користувач захоче вийти з додатку, він відправить токени до серверу, для інвалідування, щоб зловмисник у випадку перехоплення цих токенів не зміг продовжити сесію користувача, та видати себе за нього.

Таким чином нова система управління сесіями, яка використовує access та refresh токени, вирішує усі проблемні та неправильні рішення, які були у попередній системі.

Останнім кроком усунення вразливості порушення автентифікації буде реалізація двуфакторної автентифікації.

Двуфакторна автентифікація забезпечує додатковий рівень безпеки поверх стандартного процесу автентифікації. Рівень двуфакторної автентифікації вимагає від користувача введення додаткових даних для доступу до його облікового запису.

Ці дані можуть надходити з таких джерел:

- фізичний пристрій, наприклад смартфон або карта доступу;
- біологічний атрибут, наприклад біометричні дані, такі як відбитки пальців або сітківка ока.

Найбільш поширені форми двуфакторної автентифікації включають введення коду, відправленого на мобільний телефон, або введення коду, отриманого з додатку для автентифікації.

Розглянемо достоїнства двуфакторної автентифікація.

1) Вона забезпечує надійніший захист від атак і додатковий рівень безпеки для облікового запису користувача.

2) У більшості випадків двуфакторна автентифікація не додає додаткових витрат з боку користувача.

3) Налаштування двуфакторної автентифікації відносно просте для більшості сервісів. Для більшості реалізацій все, що треба зробити користувачеві, це включити двуфакторну автентифікацію і відсканувати quick response (QR) код, або ввести код, який прийшов на його мобільний телефон.

У даному випадку, для реалізації двуфакторної автентифікації буде використана бібліотека speakeasy на стороні серверу.

Також, для цього функціоналу буде створено простий мобільний додаток, до якого будуть приходити коди, які будуть обмежені по часу. Ці коди користувач повинен ввести у веб-додаток для проходження двуфакторної автентифікації.

Тільки після цього, користувач може бути допущений до основних функцій веб-додатку.

Першим кроком до включення двуфакторної автентифікації є створення ключа підтвердження для зв'язку з сервером, мобільного додатка та веб-додатку.

На рис. 3.12 була наведена частина коду з функцією запиту користувача на підключення двуфакторної автентифікації з новим параметром secret.

```
try {
    ...
    const temp_secret = speakeasy.generateSecret();
    await db.query("UPDATE users SET temp_secret=? WHERE id=?", [temp_secret, user.id ]);
    await sendTokenToEmail(user.email, temp_token);
    ...
} catch(e) {
    console.log(e);
    res.status(500).json({ message: e.message})
}
```

Рисунок 3.12 – Нова функція запиту підключення двуфакторної автентифікації

Як можна побачити з рис. 3.12, до запису користувача у базі даних додається параметр temp_secret. Він потрібен для початкового встановлення двуфакторної автентифікації. Варто зазначити, що temp_secret встановлюється туди тимчасово. Після закінчення встановлення двуфакторної автентифікації, цей тимчасовий секретний ключ зміниться на постійний.

Для генерації цього ключа, було використано функцію generateSecret з бібліотеки speakeasy. Виконання даної функції повертає об'єкт з секретом у форматах ascii, hex, base32 та otpauth_url.

Otpauth_url являє собою QR-код, в якому секрети закодовані у вигляді URL-адреси. Otpauth_url можна використати для створення QR-коду, який користувач може відсканувати для налаштування двуфакторної автентифікації. Оскільки для поточної реалізації QR-код не потрібен, буде використовуватися тільки рядок base32.

Після цього, сервер відправляє на email цього користувача, параметр temp_token, який був зроблений на основі temp_secret.

На рис. 3.13 був наведений код функції, яка здійснює саму відправку.

```
async function sendTokenToEmail(email, temp_token) {
    let testAccount = await nodemailer.createTestAccount();

    let transporter = nodemailer.createTransport({
        host: "smtp.ethereal.email",
        port: 587,
        secure: true,
        auth: {
            user: testAccount.user,
            pass: testAccount.pass,
        },
    });

    let info = await transporter.sendMail({
        from: '"Secure Application" <foo@example.com>',
        to: "artem@example.com",
        subject: "Test",
        text: `Here is your code to set up your 2FA - ${temp_token}`,
    });
}
```

Рисунок 3.13 – Функція яка відправляє код підтвердження на email

Як можна побачити з рис. 3.13, у функцію поступають параметри email та temp_token. Далі визначається, як буде відправлятися електронний лист, порт та облікові дані поштового сервісу, який буде використовуватися. При відправці вказується тема та контент електронного листа з вказанням коду підтвердження. Як провідника поштового сервісу було використано google.

На рис. 3.14 був наведений приклад електронного листа, який буде отримувати користувач.

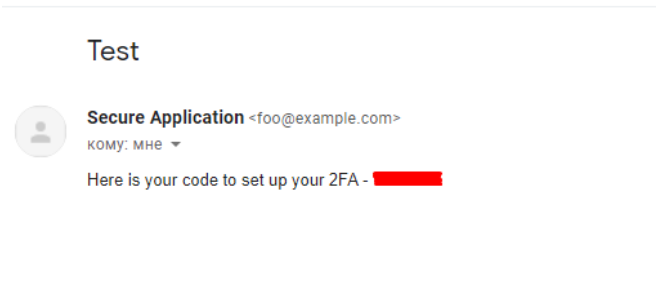

Рисунок 3.14 – Приклад електронного листа, який буде отримувати користувач

Як можна побачити з рис. 3.14, на пошту був присланий temp_token, який треба буде відправити назад на сервер для продовження активації.

Наступним кроком буде реалізація функції, яка буде перевіряти temp_token на сервері, та прив’язувати двуфакторну автентифікацію до облікового запису користувача. Сама функція буде знаходитися, на endpoint, який буде отримувати ідентифікатор користувача і temp_token. Потім функція звіряє temp_token з збереженим тимчасовим секретом, і якщо все сходиться, сервер заміняє секретний ключ на постійний, що означає закінчення активації двуфакторної автентифікації.

На рис. 3.15 була наведена функція, яка буде підтверджувати temp_token, який користувач буде отримувати з електронного листа.

```
...
app.post("/api/verify", (req,res) => {
    const { userId, tempToken } = req.body;
    try {
        const user =  await db.query("SELECT * FROM users WHERE id=?", [userId]);
        const { base32: secret } = user.temp_secret;
        const verified = speakeasy.totp.verify({
            secret,
            encoding: 'base32',
            tempToken
        });
        if (verified) {
            db.query("UPDATE users SET secret=? WHERE id=?", [temp_secret, user.id ]);
            res.json({ verified: true })
        } else {
            res.json({ verified: false})
        }
    } catch(e) {
        console.error(e);
        res.status(500).json({ message: e.message})
    };
})
...
```

Рисунок 3.15 – Приклад перевірочної функції для temp_token

Як можна побачити з рис. 3.15, після отримання userId, та tempToken, сервер шукає такого користувача, та після знаходження бере секретний ключ, який було збережено у параметр temp_secret раніше. Далі, за допомогою speakeasy буде виконано порівняння temp_secret з tempToken, який має бути зроблений на його основі. Якщо це так, то сервер зберігає temp_secret у базу даних, як параметр secret, що означає, що двуфакторна автентифікація була активована.

Наступним кроком буде реалізація функції, яка буде перевіряти токени зроблені на основі параметру secret, які користувач буде вводити з мобільного додатку. Для цього потрібно додати ще один ендпоінт з функцією, який підтверджуватиме, що токени, введені користувачем, дійсні. Цей ендпоінт буде дуже схожим на структуру підтвердження temp_token. Як і у випадку з temp_token, перевірка буде відбуватися через функцію від Speakeasy яка називається totp.

На рис. 3.16 був наведений код перевірки одноразових токенів.

```
...
app.post("/api/validate", (req,res) => {
    const { userId, token } = req.body;
    try {
        const user =  await db.query("SELECT * FROM users WHERE id=?", [userId]);
        const { base32: secret } = user.secret;
        const tokenValidates = speakeasy.totp.verify({
            secret,
            encoding: 'base32',
            token,
            window: 1
        });
        if (tokenValidates) {
            res.json({ validated: true })
        } else {
            res.json({ validated: false})
        }
    } catch(e) {
        console.error(e);
        res.status(500).json({ message: e.message})
    };
})
...
```

Рисунок 3.16 – Приклад перевірочної функції для одноразових токенів

Як можна побачити з рис. 3.16, ця функція майже ідентична до перевірки temp_token. Відрізняється вона лише тим, що ключ з яким повинен перевірятися отриманий токен береться з параметру secret замість temp_secret, та до списку параметрів функції перевірки totp додався ще один, який називається window.

Параметр window відноситься до періоду часу, впродовж якого токен дійсний після закінчення його терміну придатності. Якщо кожен токен дійсний протягом 30 секунд, параметр window множить ці 30 секунд, та своє значення. Таким чином, якщо параметр вікна дорівнює 2, а термін життя токена 30 секунд, то результуюча величина буде дорівнювати 60 секунд.

У даному випадку, для додатку було встановлено 30 секунд для кожного токену, та параметр window був встановлений на 1.

Останнім кроком буде реалізація функції ротації токенів для активованої, двуфакторної автентифікації.

Для цього буде потрібен endpoint, який буде видавати одноразові токени для користувачів, які хочуть автентифікуватися у веб-додаток.

Як і раніше, створення токену відбуватиметься через бібліотеку Speakeasy. Буде використана функція з цієї бібліотеки, яка на основі параметру secret, буде створювати ці тимчасові токени.

На рис. 3.17 був наведений код функції створення тимчасового токену.

```
...
app.get("/api/auth/new_token", (req,res) => {
    try {
        const user =  await db.query("SELECT * FROM users WHERE id=?", [userId]);
        const { base32: secret } = user.secret;

        const verificationToken = speakeasy.totp({
             secret: secret.base32
        });

        res.json({ verificationToken })
    } catch(e) {
        console.error(e);
        res.status(500).json({ message: e.message})
    };
})
...
```

Рисунок 3.17 – Код функції створення тимчасового токену

Як можна побачити з рис. 3.17, за основу тимчасового токену використовується base32 версія секрету. Далі цей токен відправляється до мобільного додатку користувача.

Тепер, коли з серверною частиною було закінчено, потрібно розробити сам мобільний додаток, який буде зв’язано як з серверною частиною, так і з веб-додатком.

Мобільний додаток буде написаний на технології React Native, яка по факту є бібліотекою React для написання додатків на мобільні телефони. Мобільний додаток буде складатися з 2 сегментів.

Цими сегментами є:

- сегмент з формою логіну;
- сегмент який показує тимчасові токени, які приходять до серверу.

Почнемо з сегменту форми логіну. Вона являє собою просту форму з усіма перевірками, та обмеження, що були описані на формі у веб-додатку для запобігання різноманітних атак.

На рис. 3.18 був наведений сегмент з формою логіна у мобільному додатку.

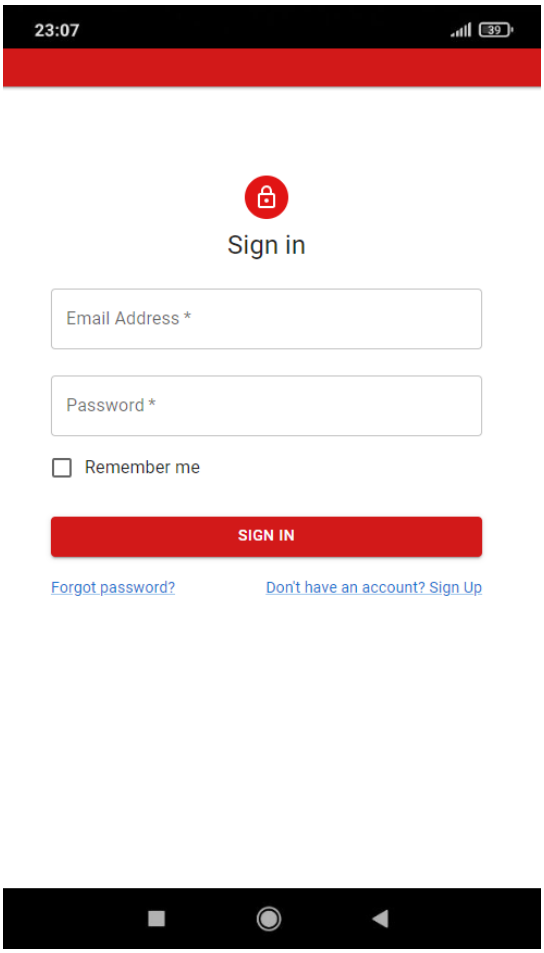

Рисунок 3.18 – Форма логіну у мобільному додатку

Як можна побачити з рис. 3.18, було створено форму логіну, яка дозволяє користувачу авторизуватися, до мобільного додатку для отримання коду підтвердження, який використовується для входу до веб-додатку. Варто відмітити, що при першому заході у мобільний додаток, користувач автоматично робить запит на сервер, який починає процес підключення двухфакторної автентифікації. Використовується функція, яка була показана на рис. 3.12.

Після вдалої авторизації до мобільного додатку, наступне, що бачить користувач, це сегмент з полем підтвердження, у яке він повинен ввести код підтвердження, який був висланий йому на електронну пошту. Приклад такого електронного листа можна розглянути на рис. 3.14.

На рис. 3.19 була наведена частина інтерфейсу з полем для введення коду підтвердження.

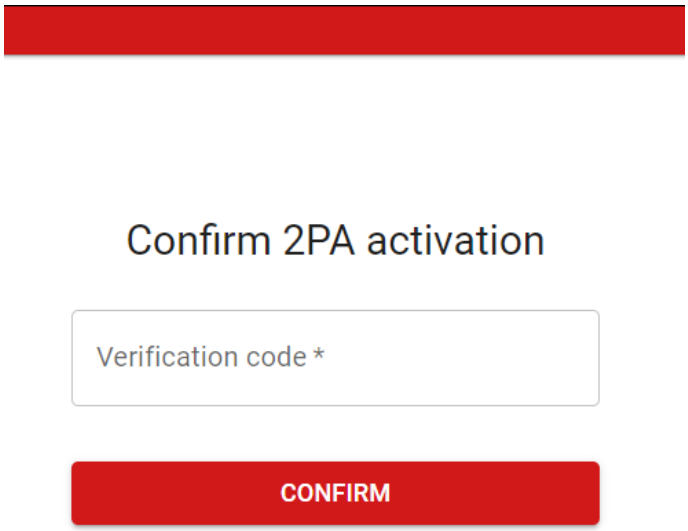

Рисунок 3.19 – Форма для введення коду підтвердження

Як можна побачити з рис. 3.19, мобільний додаток запрошує тимчасовий код, який був висланий до користувача через електрону пошту сервером. Після введення цього коду, мобільний додаток робить запит на функцію валідації, яка була наведена на рис. 3.15, у якій і відбувається підключення мобільного додатку до облікового запису користувача.

Далі інтерфейс на мобільному додатку міняється на ротацію кодів верифікації двуфакторної автентифікації, які користувач повинен вводити при кожному вході до облікового запису веб-додатку.

На рис. 3.20 була наведений інтерфейсу цієї ротації кодів верифікації.

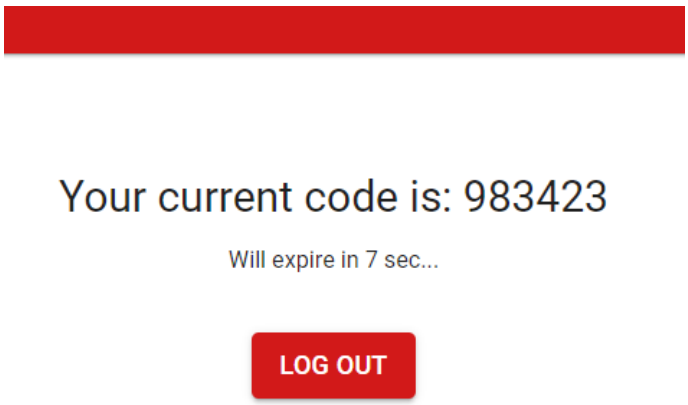

Рисунок 3.20 – Інтерфейс ротації кодів верифікації

Як можна побачити з рис. 3.20, до закінчення терміну придатності поточного коду залишилось 7 секунд. Після того, як цей код стане недійсним, мобільний додаток зробить запит для отримання нового коду верифікації. Сам запит буде зроблено на функцію, яка була зображена на рис. 3.17. Після вдалого запиту, новий код знову буде дійсним 30 секунд. Потім процес повториться.

У цей час, користувачеві з активованою двуфакторною автентифікацією потрібно буде ввести дійсний код з мобільного додатку після проходження стандартної автентифікації у веб-додатку.

На рис. 3.21 був наведена приклад спливаючого вікна, куди потрібно ввести код у веб-додатку.

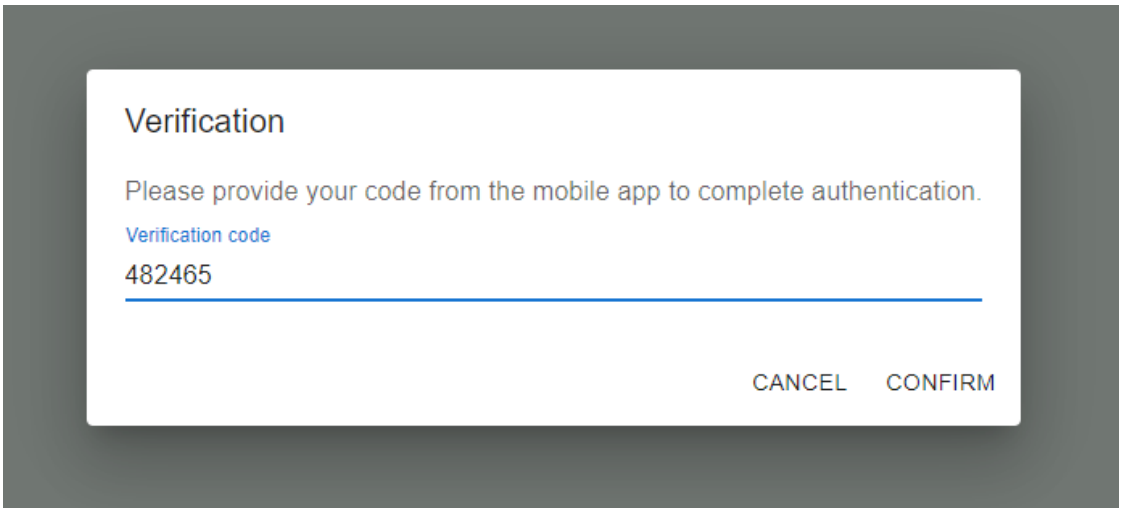

Рисунок 3.21 – Форма для введення коду підтвердження

Як можна побачити з рис. 3.21, у веб додаток був введений код підтвердження з мобільного додатку для закінчення другого кроку автентифікації. Коли користувач натискає на кнопку для відправки, то запит відправляється на сервер до функції перевірки тимчасових кодів, яка була зображена на рис. 3.17. Якщо код дійсний, то користувач допускається до додатку.

Таким чином була реалізована двуфакторна автентифікація у розглядаємому додатку. У сукупності з іншими методами, які були описані вище, шанс на експлуатацію вразливість порушення автентифікації був значно знижений.

3.4 Усунення вразливості розкриття конфіденційних даних

У розглядаємому додатку на разі є 2 проблеми зв’язаних з вразливістю розкриття конфіденційних даних.

Цими проблемами є:

- не використання алгоритму хешування для зберігання конфіденційної інформації у базі даних;
- для передачі інформації по мережі, використовується HTTP протокол.

Для усунення першої проблеми, треба спочатку зрозуміти, які данні є конфіденційними у нашому додатку.

На рис. 3.22 був наведений типовий об’єкт типу user у базі даних додатку.

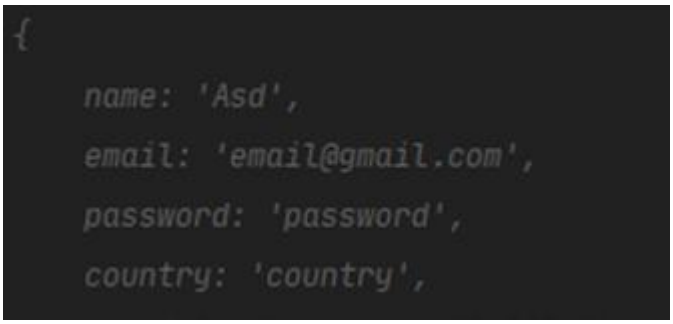

Рисунок 3.22 – Типовий об’єкт типу user у базі даних додатку

З рис. 3.22, стає очевидно, що поле password не повинне зберігатися у форматі відкритого тексту. Така ситуація являє собою найтиповіший приклад вразливості розкриття конфіденційних даних.

До варіантів вирішення цієї проблеми можна віднести такі алгоритми хешування:

- argon2;
- scrypt;
- bcrypt;
- pbkdf2.

Хоча більшість алгоритмів з цього списку вважаються достатньо безпечними, далеко не всі вони підійдуть для цього завдання.

Критерії, які повинен задовольняти алгоритм для цього завдання:

- алгоритм повинен бути стійким;
- алгоритм повинен бути повільним;
- у процесі хешування повинен використовуватися salt;
- алгоритм повинен бути перевіреним часом;
- алгоритм повинен адаптуватися до нових потужностей обчислювальної техніки;
- алгоритм повинен використовувати якомога меншу кількість пам’яті обчислювальної машини, на якому вона виконується.

Незважаючи на очевидну стійкість, повільність алгоритму оправдана тим, що зловмиснику при спробі реалізації атаки грубою силою буде потрібно витрати величезну кількість часу для підбору захешованого значення. Наприклад, саме тому алгоритми, сім’ї SHA не використовуються для хешування конфіденційних даних. Вони з самого початку були сконструйовані так, щоб виконуватися якомога швидше.

Для подальшого укріплення безпеки проти злому, алгоритм повинен використовувати процес додавання salt. Процес додавання salt – це додавання випадкових значень до хеш-функції для отримання унікального хешу. Навіть при використанні одних і тих же вхідних даних можна отримати різні і унікальні хеши. Ці хеши призначені для посилення загальної безпеки, захисту від атак по словнику, атак методом грубої, атак веселковими таблицями та інших.

Перевірка часом являє собою не менш важливий показник. Алгоритм може бути кращим по усім параметрам, але його новизна може призвести до непередбачуваних наслідків. Наприклад, новизна приведе до підвищеного інтересу до цього алгоритму, а з підвищеним інтересом прийдуть зловмисники, які почнуть його тестувати. Одна або дві знайдені вразливості у алгоритмі і усі конфіденційні дані, які використовують цей алгоритм будуть знаходитися у небезпеці. Вибір перевіреного часом алгоритму знизить фактор непередбачуваності, що підвищить впевненість у захисті системи.

Алгоритм повинен адаптуватися до нових потужностей обчислювальної техніки. Наприклад, одна з таких адаптацій, це можливість збільшити кількість разів проходження циклу при додаванні salt. Якщо алгоритм не зможе цього робити, то його буде дуже легко скомпрометувати завдяки потужним обчислювальним машинам, що просто скористаються атакою підбору.

І нарешті, використання пам’яті. Алгоритм повинен використовувати якомога менше пам’яті через можливість проведення атак на основі graphics processing unit (GPU).

Враховуючи усі ці критерії, був вибраний bcrypt. Стійкий, та перевірений часом, алгоритм bcrypt має можливість як додавати salt до хешу, так і адаптуватися. Сам по собі він являє собою повільний алгоритм, що ідеально підходить до нашого випадку.

У свою чергу argon2 не підходить для даного завдання, через швидший час виконання ніж bcrypt. Крім того, argon2 вважається відносно новим алгоритмом, який ще не пройшов таке випробування часом, як bcrypt.

Scrypt сильно програє bcrypt по використанню пам’яті. Scrypt вимагає приблизно в 1000 разів більше пам'яті, ніж bcrypt, для досягнення порівнянного рівня захисту від атак на основі GPU.

В основі pbkdf2 лежить SHA алгоритм, про який вже було згадано раніше. Через швидкість виконання, алгоритми SHA не підходять для даного завдання.

Тепер, коли сам алгоритм було вибрано, та дані для хешування були визначені, наступним кроком буде налаштування часу виконання алгоритму. Незважаючи на те, що bcrypt є повільним по стандарту, цієї повільності може не вистачити при атаці на нього з більш потужних обчислювальних приладів. Для таких випадків bcrypt може бути адаптований шляхом вибору правильного коефіцієнту роботи.

Коефіцієнт роботи – це, по суті, кількість ітерацій алгоритму хешування, які виконуються для кожного пароля. Мета коефіцієнта роботи полягає в тому, щоб зробити створення хешу більш витратним в обчислювальному відношенні, що, у свою чергу, знижує швидкість створення і збільшує час, за який зловмисник буде зламувати хеш пароля. Коефіцієнт роботи зазвичай зберігається у вихідних даних хеш-функції.

Варто зазначити, що не можна просто так поставити велике значення для коефіцієнту роботи. У такому випадку, час виконання буде занадто великим, що буде негативно позначатися на користувачах які використовують додаток. Напри-

клад, при автентифікації до додатку, користувач відправляє свій пароль до серверу, який порівнює його з хешом у базі даних. Процес порівняння при великому значенні коефіцієнту роботи може зайняти від 1 хвилини. У такому разі рівень зручності використання додатку буде знижений, що приведе до невдоволення від користувачів. Як правило, такий процес повинен займати максимум 2 секунди.

Саме тому потрібно вибрати збалансоване значення коефіцієнту роботи. Для цього потрібно буде розрахувати приблизний час злому хешу на основі коефіцієнту роботи.

Першим кроком буде, знаходження кількості хешів які може спробувати зловмисник за одну секунду, з робочим фактором у діапазоні від 10 до 19. За відсутністю обчислювальної машини з достатньою потужністю для відображення адекватного показника підбору хешів у секунду, як орієнтир було взято показники тестування bcrypt на основі обчислювальної потужності восьми відео карт giga texel shader extreme (GTX) 1080, та утиліти для відновлення паролів Hashcat [10].

Варто зазначити, що на момент цього тесту bcrypt використовував робочий фактор 5, тому інші фактори потрібно буде обчислити самостійно. Усі обчислені значення представляють лише приблизні показники, але достане наближені до реальності.

Для обчислювання інших значень робочого фактору у діапазоні від 10 до 19 буде використана наступна формула:

$$Hs = \frac{m}{2^{n-5}}, \tag{3.2}$$

де m – кількість спробуваних хешів за секунду, використовуючи 8 відео карт, з робочим фактором 5;

n – робочий фактор, який потрібно знайти.

Приклад розрахунку значення кількості спробуваних хешів у секунду з робочим фактором 10 використовуючи формулу (3.2):

$$Hs = \frac{105708}{2^{10-5}} = 3303.37\ h/s$$

Таким чином буде розраховано значення для інших робочих факторів. Результати обчислень були представлені у таблиці 3.4.

Таблиця 3.4 – Значення підбору хешів у секунду

Значення робочого фактору	Кількість хешів у секунду
10	3003.37 H/s
11	1651.51 H/s
12	825.84 H/s
13	412.92 H/s
14	206.46 H/s
15	103.23 H/s
16	51.61 H/s
17	25.80 H/s
18	12.90 H/s
19	6.40 H/s

Тепер, коли стало відомо скільки хешів в секунду може підставити зловмисник з різними факторами роботи, потрібно обчислити час підбору одного пароля.

Для обчислювання часу підбору одного пароля для кожного значення робочого фактору у діапазоні від 10 до 19 буде використана наступна формула:

$$t = \frac{C^L}{Hs}, \tag{3.3}$$

де C – розмір набору символів для паролів, враховуючи нову політику паролів;

L – мінімальна довжина паролю, враховуючи нову політику створення паролів;

Hs – значення підбору хешів у секунду.

Приклад розрахунку часу на підбор одного паролю у секундах з робочим фактором 10 використовуючи формулу (3.3):

$$t = \frac{94^8}{3003.37} \equiv 2\ 029\ 616\ 525\ 906\,\text{s}$$

Таким чином ми отримуємо час, який потрібно витратити зловмиснику для злому хешу одного пароля у секундах. Далі ці секунди будуть переведені у роки та занесені у таблицю 3.5.

Таблиця 3.5 – Час злому хешу одного пароля

Значення робочого фактору	Час злому одного пароля
10	64358 років
11	117076 років
12	2147700 років
13	4295400 років
14	8786047 років
15	17572094 років
16	38658608 років
17	64431014 років
18	96646521 років
19	276132917 років

Як можна побачити, нова політика паролів у комбінації з bcrypt надають відмінні результати при захисту від атаки грубою силою. Підібрати такі хеши буде майже неможливо.

Останнім кроком для вибору оптимального робочого фактору буде вимірняння часу, який був витрачений на хешування паролів за допомогою bcrypt у діапазоні робочого фактору від 10 до 19. Вимірювання було зроблено за допомогою таймера, який надається оточенням розробки nodejs.

Результати замірів були занесені у таблицю 3.6.

Таблиця 3.6 – Час хешування паролю

Значення робочого фактору	Час використаний на хеш паролю
10	72.43 ms
11	143.23 ms
12	284.71 ms
13	570.54 ms
14	1134.81 ms
15	2277.08 ms
16	4543.10 ms
17	9110.29 ms
18	18220.64 ms
19	36447.95 ms

Маючи дві таблиці з даними по кількості часу для злому, та кількості часу для хешування одного паролю, можна зробити висновок, що завдяки сильній політиці паролів, будь-який робочий фактор у діапазоні від 10 до 19 буде задовільним. Тому було зроблено рішення використати значення 12. Таке значення було вибрано тому, що воно надає максимальний захист даних, та одночасно не оказує ніякого впливу на швидкість роботи додатку.

На рис. 3.23 був наведений приклад реалізації функції хешування паролю за допомогою bcrypt.

```
const hashPassword = async (password, rounds) => {
    await bcrypt.hash(password, rounds)
    return await bcrypt.hash(password, rounds)
}
```

Рисунок 3.23 – Частина додатку з реалізацією хешування паролів

Як можна побачити з рис. 3.23, функція приймає пароль, який був створений користувачем, та робочий фактор, що позначається параметром rounds. Тепер, цю функцію можна використовувати при процесі створення нового користувача.

На рис. 3.24 був наведений приклад використання функції хешування паролю.

```
        return res.sendStatus(400);;
    }

const existingEmail = await db.query(`SELECT * FROM users WHERE email=?`, [email]);

if(existingEmail) {
    return res.sendStatus(400);
}

hashPassword(password, 14);

...
```

Рисунок 3.24 – Використання функції хешування паролів

Як можна побачити з рис. 3.24, після перевірки пошти, починається процес хешування.

Останнім кроком буде реалізація перевірки паролю при автентифікації. Пароль, який буде вислано до серверу від користувача, буде порівняно з хешом, який лежить у базі даних. Якщо пароль і хеш співпадають, то користувача буде автентифіковано. В іншому випадку буде відправлена помилка 401.

На рис. 3.25 була наведена частина коду з перевіркою пароля та хешу у функції автентифікації.

```
const {email, password, country} = req.params
const user = await db.query(`SELECT * FROM users WHERE email=?`, [email]);
const isValid = await bcrypt.compare(password, user.password)
if(!isValid) {
 res.sendStatus(401)
 else {
```

Рисунок 3.25 – Функція перевірки хешу паролів

Як можна побачити з рис. 3.25, значення порівняння призначається у змінну isValid, що потім перевіряється, та відкидається при значенні false.

Тепер, коли усі конфіденційні данні були правильно захешовані, можна перейти до наступної проблеми, яка полягає у використані HTTP протоколу для обміну даних по мережі.

Перед тим, як данні будуть захешовані, вони все одно мають бути відправлені до серверу у відкритому вигляді. Наприклад, при кожній авторизації, користувач відправляє пароль і логін для входу. Усі ці данні ніяк не захищені, а протокол передачі HTTP тільки ускладнює ситуацію. Передача конфіденціальних даних по протоколу HTTP не дуже безпечна тому, що зловмисник може перехопити трафік, та отримати ці данні у відкритому вигляді.

Для рішення цієї проблем, треба змінити протокол передачі даних на HTTPS. HTTPS використовує transport layer security (TLS) або SSL для шифрування HTTP-запитів і відповідей, тому замість відкритого тексту зловмисник побачить ряд, здавалося б, випадкових символів.

TLS використовує технологію, що називається шифруванням з відкритим ключем. У цій технології є два ключі, відкритий ключ і закритий ключ. Відкритий ключ передається клієнтським пристроям через SSL-сертифікат сервера. Сертифі-

кати криптографічно підписані центром сертифікації (ЦС), і у кожного браузеру є список ЦС, яким він довіряє. Будь-який сертифікат, підписаний ЦС зі списку довірених, позначається зеленим замком в адресному рядку браузеру, оскільки доведено, що він є довіреним і належить цьому домену.

Коли клієнт відкриває з'єднання з сервером, кожній машині потрібно підтвердження особи. Таким чином, два пристрої використовують відкритий і закритий ключі для узгодження нових ключів, що називаються ключами сеансу, для шифрування подальшого обміну даними між ними. Потім усі HTTP-запити і відповіді шифруються за допомогою цих сеансових ключів, так що будь-хто, хто перехоплює повідомлення, може бачити тільки випадковий рядок символів, а не відкритий текст.

Таким чином була усунута вразливість розкриття конфіденційних даних у додатку.

3.5 Усунення вразливості порушення контролю доступу

На даний момент, більшість вразливостей зв’язаних з порушенням контролю доступу були виправлені шляхом впровадження JWT токену. Завдяки природі побудування JWT токенів, користувачі можуть бути однозначно ідентифіковані по значенню, яке знаходиться у самому токені.

Як вже було згадано раніше, у JWT токен можна закодувати любі значення, та завдяки криптографічному підпису, ці значення неможливо змінити без компрометації всього токену. Компрометовані токени не пройдуть перевірку на сервері, та користувач не отримає даних, які йому не належать. Наприклад, користувач змінивши id у своєму access token не отримає дані користувача з цим id.

Наважаючи на все, одна проблема все ще залишилася, і це нерегульований доступ до функцій облікового запису адміністратора.

До таких функцій належать:

- додавання, редагування та видалення користувача;
- додавання, редагування та видалення продуктів.

Фактично, це функції контролю усього додатку. На даний момент, усі хто мають дійсний токен, та знають назви endpoint можуть скористатися цими функціями адміністратора.

Для того, щоб виправити цю проблему потрібно:

- додати параметр role, до access JWT, який буде визначати роль користувача, на основі якої і буде регулюватися доступ до приватних функцій додатку;
- створити додаткову перевірочну функцію на упізнавання адміністративних прав.

Почнемо з впровадження параметру role. Для цього потрібно змінити процес створення токену при успішному проходженні автентифікації яка була наведена на рис. 3.8.

На рис. 3.26 був наведений код з реалізацією видачі JWT з новим параметром role.

```
...
const user =  await db.query("SELECT * FROM users WHERE id=?", [userId]);
const { role, email } = user;
const user = {
    email,
    role,
}

const token = jwt.sign(user, config.accessTokenSecret, { expiresIn: config.tokenExpiration})
const refreshToken = jwt.sign(user, config.refreshTokenSecret, { expiresIn: config.refreshTokenExpiration})
res.cookie('token', token, { httpOnly: true });
res.cookie('refreshToken', refreshToken, { httpOnly: true, secure: true })
res.status(200);
..
```

Рисунок 3.26 – Впровадження параметру role у функцію видачі токенів

Як можна побачити з рис. 3.26, до закодованих значень у JWT був добавлений параметр role. Параметр може приймати значення user, та admin. Тепер по токену можна зрозуміти, які привілеї повинен мати користувач.

Тепер потрібно додати функцію перевірки прав адміністратора, шляхом слідкування за параметром role, який було закодовано у JWT.

На рис. 3.27 був наведений код з такої перевірочної функції.

```
router.use((req, res, next) => {
    if (!req.user || !(req?.user?.role === admin)) {
        res.status(401).json({ error: 'Unauthorized' });
        return;
    }

    next();
});
```

Рисунок 3.27 – Впровадження перевірочної функції на адміністратора

Як можна побачити з рис. 3.27, якщо користувач не адміністратор, то сервер поверне помилку 401, яка означає, що користувач не має достатніх привілеїв для адміністративної функції.

Таку перевірочна функція буде встановлена перед виконанням усіх функцій адміністратора, що не дасть звичайним користувачам доступу до них.

Таким чином була усунена вразливість порушення контролю доступу.

3.6 Усунення вразливості XSS

Завдяки тому, що додаток використовує бібліотеку React, у якій вже реалізовані методи боротьби з вразливістю XSS шляхом перевірки усіх елементів сторінки на потенційно шкідливий код, шанси на вдалу реалізацію XSS атаки невеликі. Проте, така перевірка не являє собою повний захист, та не виключає можливості пропуску шкідливого коду у рідких випадках. Саме тому буде реалізована додаткова перевірка коду.

Така перевірка буде виконана за допомогою бібліотеки перевірки елементі сторінки dompurify. Усі місця, де користувач може потенціально спробувати ввести вразливий код будуть додатково перевірені та відфільтровані.

На рис. 3.28 була наведена частина коду, де виконується додаткова перевірка.

```
const sanitizedContent=DOMPurify.sanitize(content)

<div>
    <div>
        {sanitizedContent}
    </div>
</div>
```

Рисунок 3.28 – Приклад додаткової перевірки на XSS атаку

Як можна побачити з рис. 3.28, перед вставкою контенту у сторінку, це наповнення додатково проходить перевірку, що повинно мінімізувати шанси скоєння XSS атаки.

Таким чином, був вдосконалений захист від XSS у роглядаємому додатку.

3.7 Усунення вразливості CSRF

Для захисту розглядаємого додатку від вразливості CSRF, потрібно реалізувати наступні методи захисту:

- додатковий захист для cookie файлів з JWT токенами;
- реалізувати cross-origin resource sharing (CORS) політики на стороні серверу;
- реалізація CSRF токенів.

Почнемо з додаткового захисту для cookie файлів. Для забезпечення такого захисту потрібно додати новий параметр при створені нового cookie. Цим параметром є Same-site. Такий параметр потрібен для визначення сайтів з яких cookie будуть прикріплятися автоматично до запитів на сервер.

Розглянемо значення який може приймати цей параметр.

1) None. При такому параметрі, cookie будуть прикріплятися при запиту з любого сайту, що є небезпечною практикою, та надає зловмисникам можливість реалізувати атаку CSRF.

2) Strict. При такому параметрі cookie будуть прикріплятися при запиту тільки з сайту, який видав ці cookie.

3) Lax. При такому параметрі cookie будуть прикріплятися при запиту з сайту, який видав ці cookie, та при переході по посиланню до цього сайту з іншого сайту.

У даному випадку був вибраний параметр Strict, який не дозволить зробити прямих запитів з прикріпленням cookie, з іншого сайту.

На рис. 3.29 був наведений код з додатковим параметром Same-site при створені cookie.

```
const token = jwt.sign(user, config.accessTokenSecret, { expiresIn: config.tokenExpiration})
const refreshToken = jwt.sign(user, config.refreshTokenSecret, { expiresIn: config.refreshTokenExpiration})
res.cookie('token', token, { httpOnly: true, secure: true, sameSite: 'Strict' });
res.cookie('refreshToken', refreshToken, { httpOnly: true, secure: true, sameSite: 'Strict' })
res.status(200);
```

Рисунок 3.29 – Додатковий параметр Same-site при створені cookie

Як можна побачити з рис. 3.29, до вже встановлених параметрів httpOnly та secure було добавлено SameSite. Тепер зловмисник не зможе зробити запит до розглядаємого додатку через інший веб-сайт.

Наступним кроком, буде реалізація політики CORS.

Політика CORS – це протокол, який дозволяє серверу вибірково приймати запити від визначених клієнтів та відкидати запити від усіх інших клієнтів. Таким чином, у зловмисника значно знизяться шанси провести вдалу атаку CSRF.

На рис. 3.30 був наведений приклад коду з серверу, де була налаштована політика CORS, що дозволяє тільки клієнту цього веб-додатку робити запити до поточного серверу.

```
...
app.use(cors({
    origin: 'http://mywebsite.com'
}));
...
```

Рисунок 3.30 – Налаштована політика CORS

Як можна побачити з рис. 3.30, тепер тільки ‘mywebsite’, який являє собою клієнтську частину цього додатку, може звернутися до цього серверу.

Нажаль, додаванням політики CORS буде мало для подолання усіх типів CSRF атак. Наприклад підтип DNS-rebinding, головною особливістю якого є факт того, що він може подолати границі CORS політик браузера [2]. Тому наступним кроком, буде реалізація так званого CSRF токену.

CSRF токен – це унікальне, секретне, непередбачуване значення, яке генерується додатком на стороні сервера і передається клієнтові. Далі це значення включається в усі подальші HTTP-запити, які були зроблені клієнтом. Після цього сервер перевіряє чи було включене це значення до запиту. Якщо ні або воно було недійсним, то такий запит відхиляється сервером.

Для створення CSRF токену буде використана бібліотека csurf.

На рис. 3.31 був наведений код з створенням CSRF токену на стороні серверу.

```
app.use(csrf({cookie: true}));
app.get('/getCSRFToken', (req, res) => {
    res.json({ CSRFToken: req.CSRFToken() });
});
```

Рисунок 3.31 – Функція для створення CSRF

Як можна побачити з рис. 3.31, тепер клієнт може послати запит для отримання CSRF токену одразу після проходження автентифікації, та після кожного повернення до веб-додатку. Після його отримання, клієнт буде зберігати його на стороні клієнта у пам’яті додатку, та відправляти його з кожним запитом до серверу. Тепер, якщо зловмисник спробує зробити атаку CSRF, його запит буде відкинуто, тому, що йому не буде вистачати CSRF токену. Такий токен буде працювати на всі типи атак CSRF.

Таким чином була усунена вразливість CSRF у розглядаємому додатку.

3.8 Реалізація логування та моніторингу

Ніяка система не може гарантувати абсолютний захист від всього одразу, тому необхідно постійно знати, що коється у системі для швидкого реагування у випадку, якщо атака все ж таки сталась, або зловмисник у процесі її реалізації.

Для реалізації логування яке буде надавати адекватну картину про стан додатку буде використано бібліотеку Winston. Ця бібліотека надає значну свободу щодо вибору налаштувань всього процесу логування.

Перед початковим налаштуванням потрібно вирішити як будуть визначатися рівні логування. Вони дозволяють розрізняти типи подій в системі і додавати контекст до важливості кожної події. При правильному налаштуванні цих рівнів, буде легко відрізнити критичні події, що вимагають негайного реагування, від чисто інформативних подій.

Розглянемо ці рівні.

1) Fatal level - використовується для позначення катастрофічної ситуації. Коли додаток не може відновитися. Реєстрація на цьому рівні зазвичай означає кінець виконання програми.

2) Error level - є станом помилки в системі, яка призводить до зупинки конкретної операції, але не системи в цілому.

3) Warn level - вказує на небажані або незвичайні умови виконання, але не обов'язково на помилку. Прикладом може бути використання резервного джерела даних, коли основне джерело недоступне.

4) Info level - Інформаційні повідомлення носять виключно інформативний характер. На цьому рівні можуть реєструватися події, керовані користувачем або залежні від додатка. Зазвичай цей рівень використовується для реєстрації подій часу виконання, таких як запуск або завершення роботи служби.

5) Debug level - використовується для представлення діагностичної інформації, яка може знадобитися для усунення несправності.

6) Trace level - фіксує усі можливі деталі поведінки додатка під час розробки.

На рис. 3.32 було наведене початкове налаштування для бібліотеки Winston.

```
const logLevels = {
    fatal: 0,
    error: 1,
    warn: 2,
    info: 3,
    debug: 4,
    trace: 5,
};

const logger = createLogger({
    levels: logLevels,
    transports: [new transports.Console()],
});
```

Рисунок 3.32 – Початкове налаштування Winston

Як можна побачити з рис. 3.32, у початкове налаштування Winston було передана репрезентація рівнів логування, які будуть відображені у самих логах.

Наступним кроком буде забезпечення читабельного формату логів. Наявність легких для читання логів значно полегшить процес їхнього аналізу для розробників і системних адміністраторів. Також важливо використати структурований формат, який легко аналізується машинами. Це дозволяє виконувати деяку автоматичну обробку логів.

Для цієї задачі був вибраний формат json, який є універсальним фаворитом для структурованого логування, тому що він може бути легко прочитаним як людьми, так і комп'ютерами . Він також конвертується в інші формати навіть при роботі з іншими мовами програмування. При роботі з json необхідно використати стандартизовану схему створення, щоб чітко визначити кожне поле. Це також спрощує пошук потрібної інформації при аналізі логів.

На рис. 3.33 було включене перетворення логів у json формат.

```
const logger = createLogger({
    levels: logLevels,
    format: format.combine(format.timestamp(), format.json()),
    transports: [new transports.Console()],
});
```

Рисунок 3.33 – Визначення json формату у логгері Winston

Як можна побачити з рис. 3.33, до нового формату також був добавлений timestamp, який буде показувати дату створення запису логу.

Наступним кроком буде автоматизація процесу логування та визначення місця збереження логів. Для реалізації автоматичного відлову помилок, та визначення місця зберігання, потрібно додати нові параметри до налаштування логера.

На рис. 3.34 були зображені ці параметри.

```
const logger = createLogger({
    levels: logLevels,
    format: format.combine(format.timestamp(), format.json()),
    transports: [new transports.Console()],
    exceptionHandlers: [new transports.File({ filename: "exceptions.log" })],
    rejectionHandlers: [new transports.File({ filename: "rejections.log" })],
});
```

Рисунок 3.34 – Визначення місця збереження логів

Як можна побачити з рис. 3.34, до налаштувань додались два нових параметра, exceptionHandler, rejectionHandler. Ці параметри дозволяють логеру відловлювати усі помили походу виконання програми, та зберігати їх у файли, які указані у параметрі filename. Кожен тип помилки піде в свій лог файл.

По стандарту, коли файл з логами буде переповнений, старі записи почнуть видалятися для того, щоб актуальніші записи могли зайняти їх місце.

Таким чином був реалізован процес логування для розглядаємого додатку.

4 ПЕРЕВІРКА СТВОРЕНОЇ СИСТЕМИ БЕЗПЕКИ

4.1 Повторна якісна оцінка додатку

Для визначення якості побудованої системи захисту була проведена повторна якісна оцінка розглядаємого додатку.

Повторні результати були наведені у таблиці 4.1.

Таблиця 4.1 – Якісна оцінка розглядаємого додатку

Назва ризику	Тип вразливості	Збиток	Вірогідність виникнення	Результуючий ризик
1	2	3	4	5
Відправлення шкідливого коду на сервер з клієнта через незахищені поля вводу	SQL injection	Низький	Дуже низька	Дуже низький
Отримання, та обробки сервером шкідливого коду	SQL injection	Низький	Дуже низька	Дуже низький
Злому паролю	Порушення автентифікації	Низький	Дуже низька	Дуже низький
Викрадення токену сесії	Порушення автентифікації	Середній	Середня	Середній
Розкриття паролів при отриманні несанкціонованого доступу до бази даних	Розкриття конфіденційних даних	Низький	Дуже низька	Дуже низький
Викрадення конфіденційної інформації при перехоплені трафіку	Розкриття конфіденційних даних	Дуже Низький	Дуже низька	Дуже низький
Отримання несанкціонованого доступу до облікового запису адміністратора	Порушення контролю доступу	Середній	Дуже низька	Низький
Отримання несанкціонованого доступу до функцій адміністратора	Порушення контролю доступу	Середній	Дуже низька	Низький

Продовження таблиці 4.1

1	2	3	4	5
Введення шкідливого коду у клієнтську частину додатку	XSS	Дуже низький	Дуже низька	Дуже низький
Вразливість відправлення конфіденційних даних через веб-сайт зловмисника	CSRF	Низький	Дуже низька	Дуже низький
Несвоєчасне знаходження вразливостей у системі	Недостатнє логування та моніторингу	Низький	Середня	Середній

Як можна побачити, після впровадження нової системи безпеки, результуючий ризик для всіх типів вразливостей був знижений багаторазово. Два пункти, які все ще мають середній ступінь ризику це несвоєчасне знаходження вразливостей, та викрадення токену сесії.

У випадку з несвоєчасним знаходженням вразливостей багато чого сильно зав’язано на людському факторі. Яка б надійна не була система логування, людина все одно може не побачити загрози, яка вже була записана у лог.

У випадку з токеном сесії, так як тепер роль сесійних токенів грають cookie з JWT, вони все ще можуть бути викрадені через шкідливе програмне забезпечення (ПЗ) “Stealer”.

Шкідливе ПЗ “Stealer” є одним з найбільш поширених типів шкідливих програм, виявлених в даний час. Предметом полювання зловмисників є крадіжка якомога більшої кількості персональних даних, від базової системної інформації до локально збережених імен користувачів й паролів [3].

В даному випадку крадіжка буде проводитися напряму з файлової системи жертви при потраплянні ПЗ “Stealer” до комп’ютеру.

4.2 Перевірка системи безпеки pin-test утилітою

У якості додаткової перевірки, розглядаємий додаток був перевірений pin-test утилітою, на загальну безпеку, та дотримання стандартів безпеки.

На рис. 4.1 були наведені результати роботи pin-test утиліти з результатами сканування розглядаємого додатку.

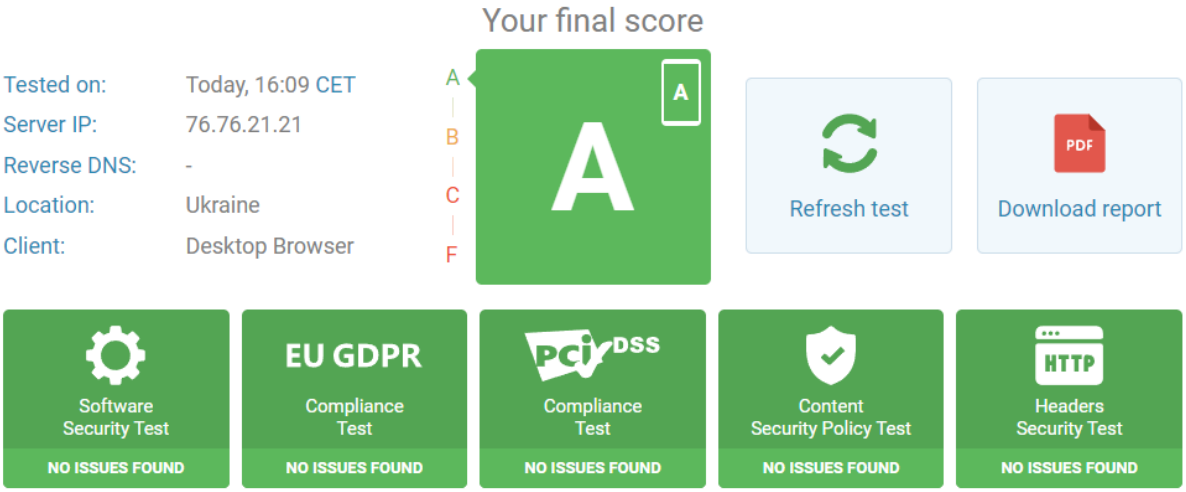

Рисунок 4.1 – Результат сканування безпеки

Як можна побачити з рис. 4.1, розглядаємий додаток отримав максимальну оцінку безпеки по усім параметрам, що доводить ефективність впровадженої системи.

ВИСНОВКИ

В даній кваліфікаційній роботі було вирішено задачу щодо реалізації комплексної системи безпеки для вразливого веб-додатку, використовуючи провідні методи та практики для її побудови.

З цією метою був проведений аналіз поточного стану сучасних веб-додатків. Даний аналіз показав загальну картину, про положення веб-додатків, як у повсякденному житті користувача, так і у екосистемі бізнесу різних розмірів. Були представленні головні достоїнства та недоліки сучасних веб-додатків. Особливу увагу було приділено актуальній проблемі підвищеної вразливості сучасних веб-додатків до зловмисних атак, які можуть скомпрометувати весь додаток, та його інформацію. Також були наведені статистичні графіки причин виникнення цих вразливостей.

З графіків стало зрозуміло, що найпоширенішою причиною виникнення вразливості є помилка при написанні коду програми. Саме тому був проведений аналіз цих вразливостей для повного розуміння, як вони уникають у коді, та яку небезпеку вони можуть принести.

Після аналізу основних вразливостей була зроблена якісна оцінка вразливого веб-додатку, який потребував створення системи безпеки навколо нього. Якісна оцінка включала у себе огляд можливих наслідків при успішній експлуатації вразливостей зловмисником, та загальні ймовірність на експлуатації цих вразливостей.

На основі отриманих результатів було розроблено комплексну систему безпеки навколо вразливого веб-додатку з використанням провідних методів реалізації систем безпеки для веб-додатків. До цієї системи входить виправлення не дуже безпечних рішень при написанні коду веб-додатка на більш безпечні, як на стороні клієнта, так и на стороні серверу. Також було розроблено мобільний додаток, який зв’язаний з сервером, за для забезпечення двуфакторної автентифікації. Нарешті було налагоджено процес логування та моніторингу, що дозволить набагато швидше відстежувати аномальну поведінку додатку.

Після реалізації системи безпеки, було зроблено повторну якісну оцінку вже захищеного веб-додатку. По результатом оцінювання, захищеність значно зросла, а шанси на експлуатацію вразливостей зменшились у рази. Для додаткової перевірки, було використано спеціалізований pin-test додаток, який сканує веб-додаток

на вразливості, та перевіряє його захищеність від них. Результати сканування показали найвищий показник захищеності, що доводить ефективність створеної системи безпеки.

Результати роботи доцільно використовувати для запровадження системи безпеки для веб-додатків від маленького до середнього розмірів, а також в навчальному процесі при виконанні лабораторно-практичних занять по спеціальності 125 «Кібербезпека».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мазепа А. Д. Безпечна автентифікація до веб-додатку з використанням JWT та browser fingerprinting / А. Д. Мазепа, А. С. Тарасов. // Харків, ХНУРЕ, Матеріали XXII Міжнародного молодіжного форуму "Радіоелектроніка та молодь у XXI столітті". Том 4. – 2021. – С. 60–61.

2. Мазепа А. Д. Розуміння та захист від атаки DNS rebinding / А. Д. Мазепа, А. С. Тарасов. // Харків, ХНУРЕ, Матеріали XXII Міжнародного молодіжного форуму "Радіоелектроніка та молодь у XXI столітті". Том 4. – 2021. – С. 62–63.

3. Тарасов А. С. Проблеми захисту персональних даних в комп’ютерних системах від шкідливого програмного забезпечення stealer / А. С. Тарасов, А. Д. Мазепа. // Харків, ХНУРЕ, Матеріали XXII Міжнародного молодіжного форуму "Радіоелектроніка та молодь у XXI столітті". Том 4. – 2021. – С. 64–65.

4. Статистика співвідношення кількості веб-додатків до статичних сайтів [Електронний ресурс] – Режим доступу до ресурсу: https://earthweb.com/how-many-websites-are-there/.

5. Статистика атак на компанії за останні три роки [Електронний ресурс] – Режим доступу до ресурсу: https://www.whitehatsec.com/news/new-report-shows-half-of-websites-were-vulnerable-to-exploitation-throughout-2021/.

6. Аналіз причин виникнення вразливостей у веб-додатках [Електронний ресурс] – Режим доступу до ресурсу: https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/.

7. Визначення поняття зламаного контролю доступу [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

8. Визначення поняття зламаного XSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.whitehatsec.com/glossary/content/cross-site-scripting.

9. Визначення ентропії паролю [Електронний ресурс] – Режим доступу до ресурсу: https://specopssoft.com/blog/password-entropy/.

10. Орієнтир тестування bcrypt [Електронний ресурс] – Режим доступу до ресурсу: https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40.