Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідуюча кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

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

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Програмний модуль захисту складових та транзакцій web-застосунку»

Виконавець: студент IV курсу, групи КБ-41

_____ **Юрій СОКИРАН**
(підпис) (ім'я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник	Іван ПАРХОМЕНКО	

Нормоконтроль	Сергій ДАКОВ	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідуюча кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентові **КБ-41** **Сокирану Юрію Юрійовичу**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Програмний модуль захисту складових та транзакцій web-застосунку

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Вразливості та слабкі місця, структура та архітектура web-додатків, методи та засоби захисту, запобігання витоку даних

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Нормативно-правова база у сфері захисту інформації, структура та складові веб-додатків, найбільш розповсюджені вразливості, ознаки наявності дірок в веб-додатку, захист від загроз порушеного контролю доступом, SQL-ін'єкцій, XSS-ін'єкцій, CSRF атак.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Виявлення ймовірних слабких та вразливих місць

web-додатку та формування метод їх усунення

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 жовтня 2021 року

Завдання видав	_____	Іван ПАРХОМЕНКО
	(підпис)	(ім’я, прізвище)
Завдання прийняв до виконання	_____	Юрій СОКИРАН
	(підпис)	(ім’я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.10.2021 – 26.01.2022	*виконано*
2	Аналіз літератури	02.01.2022 – 23.02.2022	*виконано*
3	Розгляд архітектури та складових веб-додатків	24.02.2022 – 27.03.2022	*виконано*
4	Розгляд статистичних даних щодо поширеності вразливостей	28.03.2022 – 03.04.2022	*виконано*
5	Дослідження основних вразливостей	04.04.2022 – 17.04.2022	*виконано*
6	Дослідження ефективності заходів уникнення вразливостей	18.04.2022 – 27.04.2022	*виконано*
7	Реалізація веб-додатку із вразливостями	28.04.2022 – 05.05.2022	*виконано*
8	Впровадження засобів захисту від ін’єкцій та підробки запитів	06.05.2022 – 20.05.2022	*виконано*
9	Реалізація захисту від помилок контролю доступу	21.05.2022 – 04.06.2022	*виконано*
10	Оформлення пояснювальної записки	05.06.2022 – 05.06.2022	*виконано*
11	Підготовка до захисту	06.06.2022 – 10.06.2022	*виконано*

Завдання видав	_____	Іван ПАРХОМЕНКО
	(підпис)	(ім’я, прізвище)
Завдання прийняв до виконання	_____	Юрій СОКИРАН
	(підпис)	(ім’я, прізвище)

Термін подання дипломної роботи до ЕК 06 червня 2022 року

УДК 004.056.5

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 55 сторінок, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. У пояснювальній записці дипломної роботи міститься 24 рисунки. Список використаних джерел містить 40 найменувань і займає 5 сторінок.

Методи дослідження дипломної роботи:

- аналіз літератури;
- аналіз статей на тему дослідження вразливостей веб-додатків;
- аналіз статистичних даних, представлених компаніями, що спеціалізуються на захисту від атак;

Об'єктом дослідження є процес налаштування програмних модулів, що реалізують захист від розповсюджених загроз.

Предметом дослідження є загрози інформаційній безпеці веб-додатку та способи і методи захисту від них.

Метою роботи є реалізація засобів захисту веб-додатків.

Для досягнення зазначеної мети поставлено **наступні завдання**:

- дослідити структуру веб-додатків;
- провести аналіз найбільш поширених вразливостей, характерних для визначеної структури;
- побудувати веб-додаток та визначити в ньому можливі вектори атаки;
- використати програмні засоби та механізми захисту задля унеможливлення експлуатації визначених вразливостей;

Практична цінність – програмна реалізація веб-додатку із використанням засобів захисту.

Ключові слова: веб-додаток, аналіз вразливостей, експлуатація вразливостей, XSS ін’єкція, CSRF атака.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

HTTP - Hypertext Transfer Protocol

SQL – Structured Query Language

URL – Uniform Resource Locator

OWASP - Open Web Application Security Project

SMTP – Simple Mail Transfer Protocol

FTP – File Transfer Protocol

WAF – Web Application Protocol

ACL – Access control list

XSS - Cross-Site Scripting

CSRF – Cross-Site Request Forgery

DoS – Denial of Service

ЗМІСТ

ВСТУП

Уразливість веб-сайту — це слабкість або неправильна конфігурація веб-сайту чи коду веб-програми, що дозволяє зловмиснику отримати певний рівень контролю над сайтом і, можливо, сервером хостингу. Більшість уразливостей експлуатуються за допомогою автоматизованих засобів, таких як сканери вразливостей і мережі ботів. Кіберзлочинці створюють спеціалізовані інструменти, які аналізують в Інтернеті певні платформи, шукаючи поширені та оприлюднені вразливості. Після виявлення ці вразливості потім експлуатуються для крадіжки даних, розповсюдження шкідливого вмісту або введення зіпсованого вмісту або спаму на уразливий сайт [1].

Середня вартість злому даних розвинутої компанії становить 4,24 мільйона доларів [2]. Ця цифра відображає постійно зростаючий розрив у витратах між організаціями, які впроваджують більш просунуті процеси безпеки, і тими, хто має менший рівень безпеки. Вартість злому даних значно нижча для компаній із впровадженою архітектурою безпеки, адже їх складніше зламати та в них скоріш за все прораховані плани для ліквідації наслідків. Злом даних може бути руйнівним для тих, хто не має цього захисту [2].

Знання про те, що загроз веб-безпеці багато, змушує підприємства посилювати свій захист, забезпечувати безпеку своїх даних і користувачів, а також впроваджувати інструменти та процеси, які можуть пом’якшити шкоду, завдану атакою. Це означає оплату експертизи та технологічних рішень у сфері кібербезпеки та страхових премій [3].

Безпека веб-сайту є однією з основних проблем для великих організацій, а також окремих розробників. Уразливості, що не фіксуються під час розробки, але потрапляють на доступний користувачам сервер стають легкою мішенню для хакерів. Причому страждають не тільки власники ресурсу, а й кінцеві користувачі, які потрапляють на заражений веб-сайт та можуть піддатися атаці, яка може поставити під загрозу їх систему або незахищена конфігурація системи баз даних

може призвести до потенційного витоку даних і пароля кожного зареєстрованого користувача на веб-сайті [4].

У разі успішної атаки організації також можуть понести більше фінансових витрат на:

- Виправлення пошкодження, завдані атакою
- Сплата за викуп, щоб отримати заморожені або вкрадені дані
- Штрафи від контролюючих органів, якщо організації не дотримуються законодавства про конфіденційність та безпеку даних. Коли подібне стається, компанія також повинна заплатити за юридичну допомогу.

Проблеми з веб-безпекою також можуть призвести до того, що організації втрачають дохід, оскільки клієнти переміщують свій бізнес в інше місце [3].

РОЗДІЛ 1
ВРАЗЛИВОСТІ ТА СЛАБКІ МІСЦЯ WEB-ДОДАТКІВ

1.1 Нормативно-правова база відносно захисту інформації

Так як веб-додатки безпосередньо оперують інформацією, доступом до неї, збирають дані про користувачів, то необхідно дотримуватися правопорядків визначених законодавством України. Наприклад, згідно статті 31 Конституції України кожному гарантується таємниця листування, телефонних розмов, телеграфної та іншої кореспонденції. Винятки можуть бути встановлені лише судом у випадках, передбачених законом [5].

Стаття 32 визначає протиправність втручання в особисте і сімейне життя, крім випадків, передбачених Конституцією України, а саме не допускається збирання, зберігання, використання та поширення конфіденційної інформації про особу без її згоди, крім випадків, визначених законом, і лише в інтересах національної безпеки, економічного добробуту та прав людини.

У законі України «Про інформацію» представлено визначення таким поняттям як «інформація», «захист інформації», «інформація з обмеженим доступом». В законі наведено види інформації за змістом та визначено відповідальність за порушення законодавства про інформацію [6].

Закон України «Про захист інформації в інформаційно-комунікаційних системах» регулює відносини у сфері захисту інформації в інформаційних, електронних комунікаційних та інформаційно-комунікаційних системах. Закон визначає, що об’єктом захисту в системі є інформація, що обробляється в ній, та програмне забезпечення, яке призначено для обробки цієї інформації. В ньому зазначено, що порядок доступу до інформації, перелік користувачів та їх повноваження стосовно цієї інформації визначаються володільцем інформації і що Власник системи забезпечує захист інформації в системі в порядку та на умовах,

визначених у договорі, який укладається ним із володільцем інформації, якщо інше не передбачено законом [7].

1.2 Структура веб-додатку

Структура веб-додатку визначає взаємодію між програмними модулями, проміжними системами та базами даних, щоб забезпечити стабільну роботу веб-додатку [8].

Веб-додаток зазвичай складається з трьох основних частин – клієнтської частини (frontend), серверної частини (backend) і бази даних, як представлено на рис 1.1. По суті, є дві програми, що працюють одночасно:

- Код, який живе в браузері і реагує на введення користувача
- Код, який живе на сервері і відповідає на HTTP-запити та звертається до бази даних.

При написанні програми веб-розробник повинен вирішувати, що повинен робити код на сервері відносного дій коду у браузері [4].

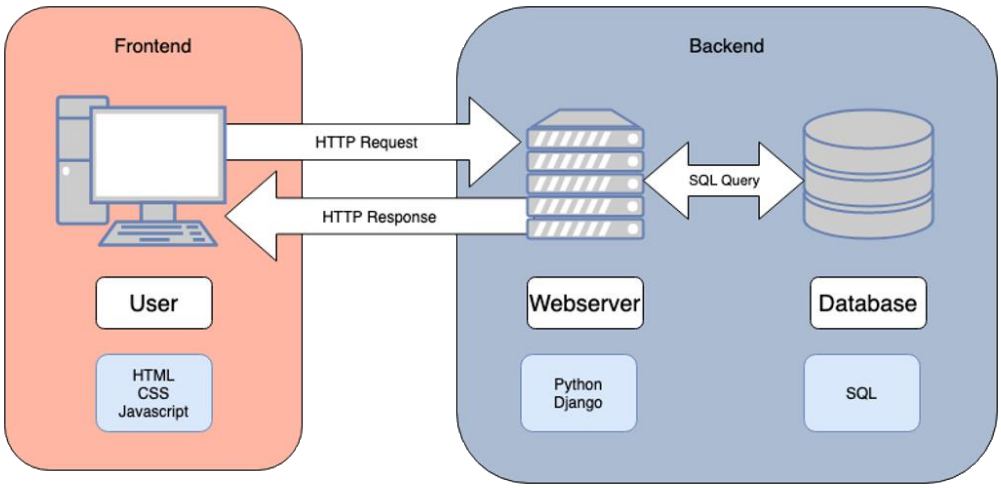

Рисунок 1.1 – Зліва клієнтська частина, що відображається на пристрої користувача, справа серверна частина, яка спілкується HTTP запитами із клієнтом та SQL запитами із базою даних.

1.2.1 Серверна частина веб-додатку

Серверна частина (backend) потрібна для динамічного веб-сайту, зазвичай вона відповідає за зв'язок з базою даних і надання даних на інтерфейс для відображення. Будь-яка конфіденційна інформація зазвичай зберігається в серверній частині і ніколи не надсилається на інтерфейс, оскільки клієнт може бачити дані інтерфейсу, його програмний код, однак користувач не може отримати доступ до даних на сервері та серверний код [8].

Властивості серверного коду [9]:

- Серверний код користувач ніколи не бачить (за винятком рідкісних несправностей)
- Будь-який код, який може виконуватися на комп’ютері та відповідати на HTTP-запити, може запускати сервер. Тому існує загалом багато бібліотек та мов програмування, що здатні допомогти у створені серверної частини веб-додатку: Rails на мові програмування Ruby, Node.js на Javascript, Django на Python, PHP, C# та Java; але список можливостей можна продовжувати довго.
- Звертається до бази даних, що зберігає такі дані, як профілі користувачів, пости блогу, сторінки тощо.
- Може відповідати лише на HTTP-запити для певної URL-адреси, а не на будь-яку дію користувача.
- Сторінку, яку бачить користувач може створюватися і на сервері. Це, як правило, справедливо лише для веб-програм, які створюють свої макети заздалегідь на сервері та потім відправляють користувачу у відповідь на його запит.

1.2.2 Клієнтська частина веб-додатку

Клієнтська частина (frontend) використовується для відображення основного вмісту веб-сторінки, зазвичай це єдине, що бачить клієнт після відвідування сайту. На клієнтській стороні використовується такі технології як Javascript, HTML та CSS. HTML використовується для формування макета всієї сторінки, CSS

використовується для надання стилю сторінки, а Javascript використовується для надання логіки сторінці та динамічної зміни її вмісту. Всі ці складові приймає веб-браузер, аналізує та відображає користувачу [8].

Властивості клієнтського коду [9]:

- Важливим моментом є те, що користувач може переглядати та редагувати код на стороні клієнта, тому важливо тримати це у голові при розробці та не залишати у коді ніяких критичних даних та ключів.
- Реагує на введення користувача та може змінювати зміст сторінки в залежності від введених даних.
- Не зберігає нічого після оновлення сторінки, для збереження даних та змін використовується серверний код.
- Неможливо зчитувати файли з сервера безпосередньо, необхідно спілкуватися через HTTP-запити.

1.2.3 База даних для веб-додатку

Рівень бази даних зберігає та отримує дані. Він також відповідає за керування оновленнями, надання одночасного (одночасного) доступу з веб-серверів, забезпечення безпеки, забезпечення цілісності даних та надання послуг підтримки, таких як резервне копіювання даних. Важливо, що хороший рівень бази даних повинен забезпечувати швидкий і гнучкий доступ до мільйонів рядків [10].

Для керування даними на рівні бази даних потрібне складне програмне забезпечення. На щастя, більшість систем управління базами даних (СУБД) або сервери баз даних розроблені так, що складності програмного забезпечення приховані. Для ефективного використання сервера баз даних необхідні навички проектування бази даних і формулювання запитів за допомогою мови SQL. Розуміння базової архітектури сервера баз даних неважливо для більшості користувачів [10].

В основному існує два типи баз даних – база даних SQL і база даних NoSQL [4].

1) База даних SQL:

Ці типи баз даних використовуються, коли дані, що зберігаються, структуровані, і структура не очікується змін.

Бази даних, які використовують SQL, є реляційними базами даних. Такі бази даних мають чітко визначену схему, яка не може змінюватися для певних записів чи рядків. Бази даних SQL мають величезний недолік – подвійність даних. Це означає, що якщо вам потрібно з'єднати один запис з іншим, знадобиться окремий простір і зусилля. Однак у базах даних SQL з'єднання можна встановлювати за допомогою «з'єднань» і «ключів», що значно полегшує роботу [11].

Бази даних SQL бажано використовувати, коли вам потрібна цілісність даних, довговічність даних, ізоляція даних і узгодженість даних.

2) База даних NoSQL:

Якщо дані не мають структури і можуть відрізнятися від користувача до користувача, використовується база даних NoSQL.

Бази даних NoSQL не мають чітко визначеної схеми, що робить їх гнучкими. Це означає, що якщо у вас є програма, яка потребує бази даних, яка має функціональні можливості для зміни схеми на основі сутності, то перевага надається базам даних NoSQL, таким як MongoDB, DynamoDB тощо [11].

Бази даних NoSQL є кращими, коли швидкість є пріоритетом, вони можуть отримати доступ до великих обсягів даних за короткий проміжок часу.

Вирішуючи, яку базу даних використовувати, зазвичай виявляють один або кілька з наступних факторів, які спонукають до вибору бази даних NoSQL:[12]

- Зберігання структурованих даних і даних без чітко визначеної структури;
- Величезні обсяги даних;
- Потреба в масштабуванні архітектури;
- Сучасні парадигми додатків, як-от мікросервіси та потокова передача даних в реальному часі;

1.3 Вразливості веб-додатків

1.3.1 Список загроз від OWASP

Open Web Application Security Project (далі OWASP), є міжнародною некомерційною організацією, яка займається безпекою веб-додатків. Один із основних принципів OWASP полягає в тому, щоб усі їхні матеріали були вільно доступні та легко доступні на їхньому веб-сайті, що дає можливість будь-кому покращити безпеку власних веб-додатків. Їх найвідомішим проектом є OWASP Top 10 [13].

OWASP Top 10 — це регулярно оновлюваний звіт, у якому викладено проблеми безпеки щодо безпеки веб-додатків, зосереджено на 10 найбільш критичних ризиках. Звіт складається командою експертів з безпеки з усього світу. OWASP відноситься до Топ-10 як «документу про проінформованість», і вони рекомендують всім компаніям включити звіт у свої процеси, щоб мінімізувати та/або пом`якшити ризики безпеки [13].

OWASP Top 10 не містить контрольного списку векторів атаки, але натомість його метою є підвищення обізнаності про розповсюджені вразливості безпеки, які слід враховувати розробникам додатків. Це сприяє усвідомленню практик розвитку захищеності та допомагають прищепити культуру безпечного розвитку.

Щоб розібратися в топ-10 OWASP, потрібно розуміти роль постачальників безпеки і вашої власної організації в захисті ваших веб-додатків. Деякі зони ризику можуть тільки вирішуватимуться самими розробниками додатків. Багато постачальників безпеки можуть це зробити у деяких сферах впливу, але часто не можуть забезпечити повне або найкраще покриття проти вразливостей. Кращі рішення забезпечують поєднання людей, процесів і технологій щоб пом’якшити ризики, пов’язані з ТОП-10 [14].

Топ-10 в 2020 році виглядає наступним чином (рис 1.2):

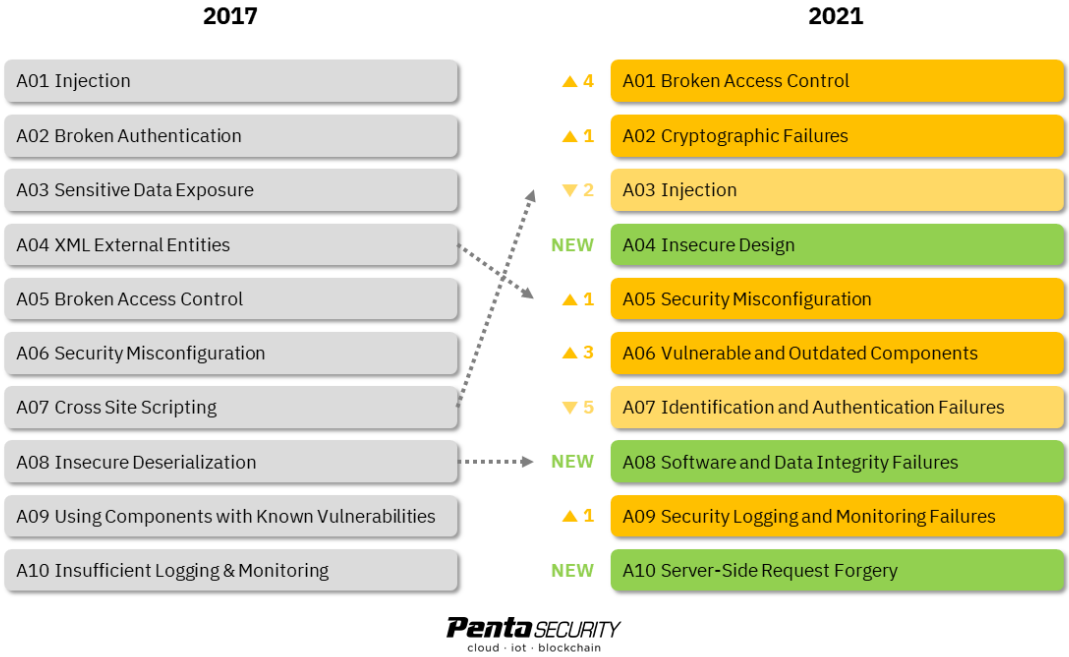

Рисунок 1.2 – Порівняння топ 10 найпоширеніших загроз згідно до OWASP 2017 та 2021 року [15]

1.3.2 Список загроз програмного забезпечення із відкритим кодом від Snyk

Ще одна компанія, що складає звіті щодо поширеності загроз – Snyk. Компанія Snyk, заснована у 2015 році, допомагає визначати вразливості безпеки у відкритому коді, який розгортають клієнти. За словами компанії, її продуктом зараз користуються близько 400 000 розробників програмного забезпечення [16].

Продукти безпеки компанії розроблені, щоб допомогти розробникам програмного забезпечення знаходити слабкі місця, порушення та вразливості у своєму коді. База даних уразливостей компанії фіксує проблеми безпеки, виявлені в бібліотеках програмного забезпечення з відкритим кодом, і виправляє код. Уразливості безпеки визначаються та усуваються під час процесу розробки, перш ніж програмний продукт буде використаний [17].

У 2020 році компанія представила свій аналіз вразливостей, що були знайдені в відкритому програмному забезпечені (рис 1.3). І це дуже важливо, адже як стверджує Gartner, понад 95% IT-підприємств по всьому світу використовують програмне забезпечення з відкритим кодом для своїх важливих IT-навантажень, незалежно від того, знають вони про це чи ні [18].

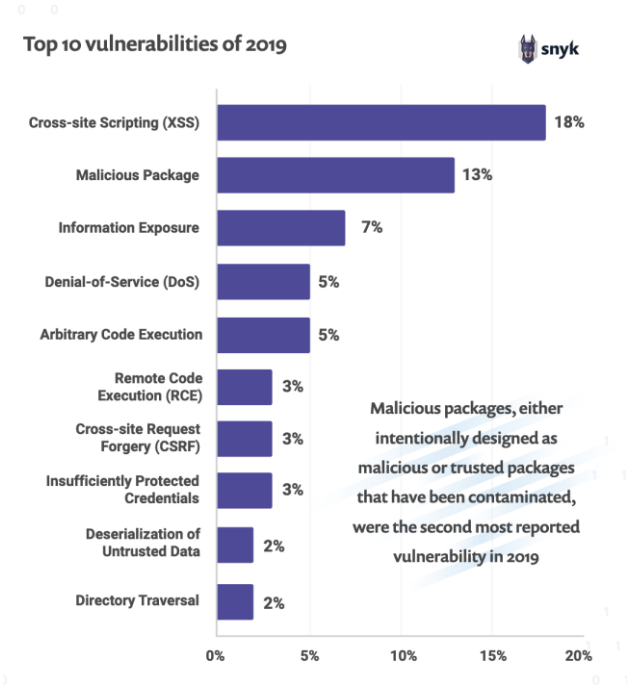

Рисунок 1.3 – Найбільш поширені загрози в відкритому програмному забезпечені на момент 2019 року [19]

1.4 Ознаки наявності дірок в безпеці додатку

1.4.1 Порушений контроль доступу.

Контроль доступу запроваджується таким чином, що користувачі не можуть діяти за межами передбачених їм дозволів. Збої в контролі зазвичай призводять до несанкціонованого розкриття інформації, зміни або знищення всіх даних або виконання бізнес-функції супереч обмежень користувача.

Серед поширених уразливостей контролю доступу [20]:

- Порушення принципу найменших привілеїв або заборони за замовчуванням, коли доступ має надаватися лише для певних можливостей, ролей або користувачів.
- Обхід перевірок контролю доступу шляхом зміни URL-адреси (змінення параметрів або примусового перегляду), внутрішнього стану програми або сторінки HTML, або за допомогою інструменту атаки, який змінює запити з серверу.
- Дозвіл перегляду або редагування чужого облікового запису шляхом надання його унікального ідентифікатора (небезпечні прямі посилання на об’єкт).
- Доступ до інтерфейсу сервера з відсутніми елементами керування доступом для POST, PUT та DELETE.
- Підвищення привілеїв. Дія як користувач без входу в систему або як адміністратор під час входу як користувач.
- Маніпуляції з метаданими, відтворення або втручання в токен доступу, файл cookie або приховане поле, яким маніпулюють для підвищення привілеїв або зловживання перевіркою на достовірність даних користувача.

1.4.2 Криптографічні збої

Паролі, номери кредитних карток, медичні картки, особиста інформація та ділові таємниці потребують додаткового захисту [21].

Для всіх таких даних необхідно визначити:

- Чи передаються будь-які дані відкритим текстом? Це стосується таких протоколів, як HTTP, SMTP, FTP. Зовнішній інтернет-трафік є небезпечним. Важливо перевірити весь внутрішній трафік, наприклад, між балансувальниками навантаження, веб-серверами або внутрішніми системами.
- Чи використовуються якісь старі чи слабкі криптографічні алгоритми чи протоколи за замовчуванням або в старішому коді?
- Чи використовуються ключі чи паролі за замовчуванням, чи генеруються слабкі криптографічні ключі або використовуються повторно, чи відсутнє належне керування ключами чи ротація?

- Чи не застосовується шифрування, наприклад, чи відсутні директиви безпеки заголовків HTTP?
- Чи правильно перевірено отриманий сертифікат сервера?
- Чи використовується випадковість для криптографічних цілей, яка не була розроблена для задоволення криптографічних вимог? Чи має обрана функція достатню непередбачуваність?
- Чи використовуються застарілі хеш-функції, такі як MD5 або SHA1, чи використовуються некриптографічні хеш-функції, коли потрібні криптографічні хеш-функції?

Якщо на більшість з цим запитань відповідь «так», то варто переосмислити використання криптографії на вашому підприємстві.

1.4.3 Ін’єкції

Згідно досліджень Snyk (рис. 1.4) XSS ін’єкції вразливості є найбільш розповсюдженими вразливостями з 2014 року. Програма вразлива для атаки ін’єкцією, коли [22, 23]:

- Дані, надані користувачем, не перевіряються, не фільтруються та не очищаються програмою.
- Динамічні запити або непараметризовані виклики без екранування з урахуванням контексту використовуються безпосередньо в інтерпретаторі програми.
- Ворожі дані використовуються в параметрах пошуку звернень до баз даних для вилучення додаткових конфіденційних записів.
- Ворожі дані використовуються безпосередньо або об’єднуються із текстом запиту.

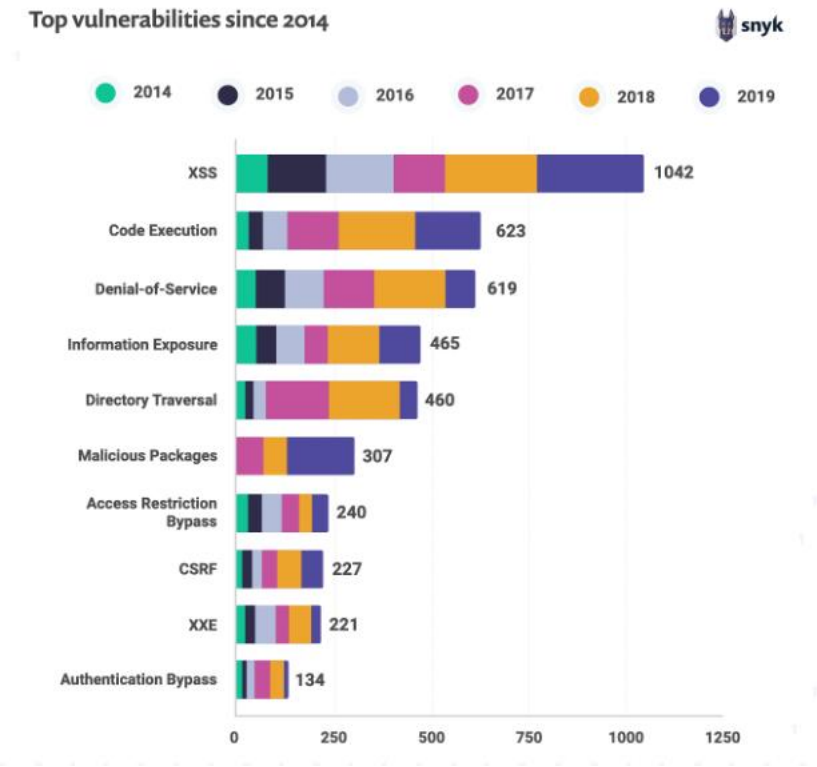

Рисунок 1.4 – Найбільш поширені вразливості в відкритому програмному забезпечені з 2014 року [19]

1.4.4 Неправильна конфігурація безпеки

Програма може бути вразливою, якщо [24]:

- Відсутнє безпечні налаштування в будь-якій частині стеку програми або неправильно налаштовані дозволи.
- Ввімкнено або встановлено непотрібні функції (наприклад, непотрібні порти, служби, сторінки, облікові записи чи привілеї).
- Облікові записи за замовчуванням та паролі залишаються активними та незмінними.
- Обробка помилок виявляє користувачам сліди стека або інші надто інформативні повідомлення про помилки.
- Для оновлених систем найновіші функції безпеки вимкнено або не налаштовано надійно.

- Параметри безпеки на серверах додатків, бібліотеках, базах даних тощо, не налаштовані на безпечні значення.
- Сервер не надсилає заголовки чи директиви безпеки, або вони не налаштовані на захищені значення.
- Програмне забезпечення застаріле або вразливе.

1.4.5 Помилки ідентифікації та автентифікації

Підтвердження ідентичності користувача, аутентифікація та керування сеансами є критичними для захисту від атак, пов'язаних із автентифікацією. Можуть бути недоліки автентифікації, якщо програма [25]:

- Дозволяє автоматизовані атаки або атаку грубою силою.
- Дозволяє стандартні, слабкі або добре відомі паролі, такі як «Пароль1» або «admin/admin».
- Використовує слабкі або неефективні процеси відновлення облікових даних і забуття пароля, такі як «відповіді на основі знань», які неможливо зробити безпечними.
- Використовує звичайний текст або слабо зашифровані сховища паролів.
- Відсутня або неефективна багатофакторна автентифікація.
- Видає ідентифікатор сеансу в URL-адресі.
- Повторне використання ідентифікатора сеансу після успішного входу.
- Неправильно скасовує ідентифікатори сеансів. Сеанси користувача або маркери автентифікації (переважно маркери єдиного входу) належним чином не анулюються під час виходу або періоду бездіяльності.

1.4.6 Відмова в обслуговувані

Відмова в обслуговувані (Denial-of-service, DoS) — це зловмисна спроба вплинути на доступність цільової системи, наприклад веб-сайту чи програми, для законних кінцевих користувачів. Як правило, зловмисники генерують великі обсяги

пакетів або запитів, які в кінцевому підсумку перевантажують цільову систему. У разі розподіленої атаки відмови в обслуговуванні (Distributed DoS) зловмисник використовує кілька скомпрометованих або контрольованих джерел для створення атаки [26].

DDoS-атаки досягають ефективності, використовуючи декілька зламаних комп'ютерних систем як джерела трафіку атаки. Експлуатовані машини можуть включати комп'ютери та інші мережеві ресурси, такі як пристрої Інтернету речей [27].

Програма або додаток є вразливими до атаки відмови в обслуговувані якщо вони:

- Неправильно обробляють внутрішні виключення та можуть припинити свою роботу у ході утворення якоїсь помилки всередині системи;
- Не перевіряють отримані вхідні дані на їх розмірність і через це дуже довго опрацьовують їх;
- Не мають обмеження по часу на роботу із запитами, тобто продовжують очікувати кінця великої задачі навіть якщо вона продовжується занадто довго;

1.4.7 Підробка міжсайтових запитів

Підробка міжсайтових запитів (Cross Site Request Forgery) – також відома як Sea Surf або Session Riding, є вектором атаки, який обманом спонукає веббраузер виконати небажану дію на сайті, в який ввійшов користувач. Залежно від характеру дії зловмисник може отримати повний контроль над обліковим записом користувача. Якщо скомпрометований користувач має привілейовану роль у програмі, то зловмисник може отримати повний контроль над усіма даними та функціями програми [28].

Щоб атака CSRF стала можливою, мають бути дотримані три ключові умови [29]:

- Відповідна дія. У програмі є дія, яку зловмисник має причину спонукати. Це може бути, наприклад, зміна дозволів або будь-яка дія щодо даних користувача (зміна власного пароля користувача).
- Обробка сеансів на основі файлів cookie. Виконання дії включає в себе створення одного або кількох запитів HTTP, і програма покладається виключно на файли cookie сеансу, щоб ідентифікувати користувача, який зробив запити. Немає іншого механізму для відстеження сеансів або перевірки запитів користувачів.
- Жодних непередбачуваних параметрів запиту. Запити, які виконують дію, не містять параметрів, значення яких зловмисник не може визначити або вгадати. Наприклад, коли користувач змушує змінити свій пароль, функція не є вразливою, якщо зловмиснику потрібно знати значення існуючого пароля.

Висновки за розділом 1

У розділі 1 було проаналізовано структуру та складові веб-додатків, їхню архітектуру. Відокремлено три основні частини – клієнтська, серверна та база даних. Проведено аналіз розповсюдженості вразливостей серед веб-додатків та відкритого програмного забезпечення, що безпосередньо використовується в цих додатках, а після цього наведено список найбільш розповсюджених слабких місць та їх причини утворення.

Клієнтська частина використовується для відображення основного вмісту веб-сторінки, зазвичай це єдине, що бачить клієнт після відвідування сайту. На клієнтській стороні використовується такі технології як Javascript, HTML та CSS.

Серверна частина потрібна для динамічного вебсайту, зазвичай вона відповідає за зв'язок з базою даних і надання даних на інтерфейс для відображення. Будь-яка конфіденційна інформація зазвичай зберігається в серверній частині і ніколи не надсилається на інтерфейс, оскільки клієнт може бачити дані інтерфейсу, його програмний код, однак користувач не може отримати доступ до даних на сервері та серверний код.

Рівень бази даних зберігає та отримує дані. Він також відповідає за керування оновленнями, надання одночасного (одночасного) доступу з вебсерверів, забезпечення безпеки, забезпечення цілісності даних та надання послуг підтримки, таких як резервне копіювання даних. Важливо, що хороший рівень бази даних повинен забезпечувати швидкий і гнучкий доступ до мільйонів рядків.

Проаналізувавши дані надані OWASP та Snyk можна зробити висновок, що кожного року не просто з’являються нові вразливості, а й старі нікуди не зникають, незважаючи на те, що вони вже досить досліджені.

Таким чином список найбільш розповсюджених вразливих місць веб-додатків містить наступне:

1. Ін’єкції
2. Порушений контроль доступу
3. Криптографічні збої
4. Неправильна конфігурація безпеки
5. Помилки ідентифікації та автентифікації
6. Відмова в обслуговувані
7. Підробка міжсайтових запитів

РОЗДІЛ 2
МЕТОДИ ТА ЗАСОБИ ПІДВИЩЕННЯ ЗАХИЩЕНОСТІ ВЕБ-ДОДАТКІВ

2.1 Методи захисту від експлуатації додатків

2.1.1 Порушений контроль доступу.

Контроль доступу ефективний лише за умови виконання перевірок на боці серверу або API без сервера, де зловмисник не може змінити перевірку контролю доступу або метадані.

Методи запобігання неефективного контролю доступом:

- Заборона за замовчуванням на усі ресурси, окрім тих, що мають бути загальнодоступними. Хороше практичне правило щодо контролю доступу — почати з мінімальних необхідних привілейованих функцій. Наприклад, за замовчуванням кожному користувачеві програми має бути відмовлено в доступі до ресурсів програми, і лише законний користувач отримує дозвіл на їх перегляд, доступ та зміну [30].
- Впровадження механізму контролю доступу один раз і використання його повторно в усій програмі. Слід довіряти лише автентифікації та авторизації на стороні сервера, оскільки вона застосовує однакові елементи керування до всіх служб, користувачів і програм. Групи безпеки та розробники повинні розробити шаблон для розподілу обов’язків. Наприклад, шаблони ролей допомагають серверу перевіряти запити на основі дозволів, пов’язаних з конкретною роллю, і не надавати доступ неавторизованим користувачам [30].
- Слід вимкнути передачу списку каталогів веб-серверу та переконайтеся, що метадані файлів (наприклад, папка .git, що містить дані про репозиторій) і файли резервної копії відсутні в кореневих веб-сайтах.
- Необхідно реєструвати помилки контролю доступу та додати сповіщення для адміністраторів, коли це необхідно (наприклад, повторні збої) [20].

- Важливо обмежити швидкість доступу до API та контролера, щоб мінімізувати шкоду від автоматизованих інструментів атак.
- Ідентифікатори сеансу, що зберігають стан, повинні бути визнані недійсними на сервері після виходу. Токени чи файли кукі, що містять дані про користувача без стану мають бути короткочасними, щоб звести до мінімуму можливість для зловмисника [20].

2.1.2 Криптографічні збої

Способи уникнення помилок при використанні криптографії [21]:

- Класифікуйте дані, які обробляються, зберігаються або передаються програмою. Визначте, які дані є конфіденційними відповідно до законів про конфіденційність, нормативних вимог або потреб бізнесу.
- Не зберігайте конфіденційні дані без потреби. Дані, які не зберігаються, не можуть бути вкрадені.
- Обов’язково зашифруйте всі конфіденційні дані.
- Забезпечте наявність актуальних і надійних стандартних алгоритмів, протоколів і ключів;
- Використовуйте правильне керування ключами.
- Вимкніть кешування відповідей, які містять конфіденційні дані.
- Не використовуйте застарілі протоколи, такі як FTP і SMTP для транспортування конфіденційних даних.
- Зберігайте паролі, використовуючи потужні функції хешування, такі як Argon2, scrypt або bcrypt.
- Уникайте застарілих криптографічних функцій, таких як MD5, SHA1.

2.1.3 Ін’єкції

Щоб запобігти ін’єкції, потрібно зберігати дані окремо від команд і запитів:[23]

- Кращим варіантом є використання безпечного API, який повністю уникає використання інтерпретатора, забезпечує параметризований інтерфейс або переходить до інструментів об’єктно-реляційного відображення (ORM).
- Використовуйте перевірку введення на стороні сервера. Це не повний захист, оскільки багато програм вимагають спеціальних символів, таких як текстові області або API для мобільних додатків.
- Для будь-яких динамічних запитів екрануйте спеціальні символи.
- Використовуйте LIMIT та інші елементи керування SQL у запитах, щоб запобігти масовому розголошенню записів у разі впровадження SQL.
- Задля запобігання SQL ін’єкцій використовуйте підготовлені вирази SQL. Вони замість того, щоб безпосередньо додавати дані до рядку запиту та виконувати його, зберігають підготовлений вираз, наповнюють його даними, які спершу очищаються та перевіряються на правильність перед виконанням.[31]
- Впровадити брандмауер веб-програм (WAF), який може виявляти та відфільтровувати атаки ін’єкції SQL (разом з іншими вразливими місцями). Такі брандмауери усувають відомі загрози за допомогою списків сигнатур, які слід заблокувати та постійно оновлюються [32].

2.1.4 Неправильна конфігурація безпеки

Необхідно впровадити безпечні процеси встановлення, включаючи [24]:

- Середовища розробки, тестування та інші мають бути налаштовані однаково, з різними обліковими даними, які використовуються в кожному середовищі. Цей процес слід автоматизувати, щоб мінімізувати зусилля, необхідні для створення нового безпечного середовища.
- Додаток має бути без будь-яких непотрібних функцій, компонентів, документації. Видаліть або не встановлюйте невикористані функції та бібліотеки.
- Процес перегляду та оновлення конфігурацій, що відповідають усім нотаткам безпеки, оновленням і виправленням має бути як частина процесу керування виправленнями.

- Архітектура додатків, що поділена на сегменти забезпечує ефективне та безпечне розділення між компонентами або клієнтами за допомогою сегментації, контейнеризації або груп безпеки в хмарі.
- Слід налаштувати автоматизований процес для перевірки ефективності конфігурацій і налаштувань у всіх середовищах.

2.1.5 Помилки в аутентифікації

Як запобігти помилок в ідентифікації та аутентифікації [25]:

- Не створювати облікові записи за замовчуванням, особливо для адміністраторів.
- Запровадити процес перевірки паролів, такі як пошук паролю у списку 10000 найрозповсюдженіших паролів кожного разу як створюється користувач або коли він змінює пароль.
- При створенні користувацького запису узгодити перевірку на складність паролю та рекомендувати змінювати пароль повторно через певний період часу.
- Обмежити невдалі спроби входу, але таким чином, щоб не створити сценарій відмови в обслуговуванні для звичайних користувачів.
- Не використовувати ідентифікатор сеансу в URL-адресі, він має бути надійно збереженим та визнаним недійсним після виходу.

2.1.6 Відмова в обслуговувані

Одним із перших методів пом’якшення DDoS-атак є мінімізація площі поверхні, яка може бути атакована, тим самим обмежуючи можливості зловмисників і дозволяючи створювати захист в одному місці. У деяких випадках ви можете зробити це, розмістивши свої обчислювальні ресурси за мережами розповсюдження вмісту (CDN) або балансувальниками навантаження та обмеживши прямий інтернет-трафік певним частинам вашої інфраструктури, наприклад серверам баз даних. В інших випадках ви можете використовувати брандмауери або

списки контролю доступу (ACL), щоб контролювати, який трафік надходить до ваших програм [26].

Щоразу, коли ми виявляємо підвищений рівень трафіку, що потрапляє на хост, важливим є можливість приймати лише стільки трафіку, скільки може обробити наш хост, не впливаючи на доступність. Це поняття називається обмеженням швидкості. Більш просунуті методи захисту можуть зробити ще один крок вперед і розумно приймати тільки легітимний трафік шляхом аналізу самих окремих пакетів. Для цього вам потрібно зрозуміти характеристики хорошого трафіку, який зазвичай отримує ціль, і вміти порівнювати кожен пакет з цим базовим рівнем [26].

2.2 Веббрандмауер

Брандмауер вебпрограм або Web Application Firewall (далі WAF) допомагає захистити веб-програму від зловмисного HTTP-трафіку. Встановивши бар’єр фільтрації між цільовим сервером і зловмисником, WAF може захистити від атак, що здійснюються на веб-додатки [33].

WAF захищає ваші веб-програми, фільтруючи, відстежуючи та блокуючи будь-який зловмисний HTTP/S-трафік, що надходить до веб-програми, і запобігає виходу будь-яких несанкціонованих даних із програми. Це робиться завдяки набору правил, що допомагають визначити, який трафік є шкідливим, а який безпечним. Подібно до того, як проксі-сервер діє як посередник для захисту особистості клієнта, WAF працює подібним чином, але навпаки — так званий зворотний проксі-сервер — діє як посередник, який захищає сервер веб-програм від потенційно шкідливого клієнта [34].

Залежно від налаштувань, він може захистити як одну машину, так і цілу мережу комп’ютерів.

Зв'язок через Інтернет здійснюється шляхом запиту та передачі даних від відправника до одержувача. Оскільки дані не можуть бути відправлені як єдине ціле, вони розбиваються на керовані пакети даних, які складають початково переданий об’єкт. Що саме перевіряє брандмауер? Кожен пакет даних складається із заголовка

(керуючої інформації) та корисного навантаження (фактичних даних). Заголовок містить інформацію про відправника та одержувача. Перш ніж пакет зможе потрапити у внутрішню мережу через визначений порт, він повинен пройти через брандмауер. Ця передача залежить від інформації, яку вона несе, і від того, наскільки вона відповідає попередньо визначеним правилам [35].

2.3 Класифікація брандмауерів

2.3.1 Негативна модель

Брандмауери з негативними моделями - це ті, які використовують техніку внесення в чорний список. У цьому випадку атакуючий вектор або характеристики зловмисних дій є спеціально поміченими і внесеними до списку. Ці вектори при виявленні брандмауером блокуються. Наприклад, якщо користувач надає вхідні дані як <script>alert(xss)</script>, запит блокується, і користувач не може використовувати веб-сайт для певного сеансу, зареєструвавши IP-адресу зловмисника. Це тому, що запит має конкретне корисне навантаження у своєму чорному списку. Таким чином, ці типи практик брандмауера можна легко використати, спробувавши різні типи корисних навантажень, які не внесені брандмауером у чорний список [36].

2.3.2 Позитивна модель

Брандмауер зображує позитивну модель, коли він не залежить від векторів атаки. Зазвичай він взаємодіє з програмою і розуміє логічний потік веб-додатка. Він підтримує кілька політик безпеки вмісту для захисту програми від можливих атак, і ці політики регулярно оновлюватимуться.

Хороший брандмауер повинен перевіряти всі поширені вразливості та боротися з атаками на стороні клієнта. Він повинен мати хороший механізм перевірки та фільтрації введення. Існують експлойти залежно від типу

використовуваних брандмауерів, які є загальнодоступними в Інтернеті. Тому перед тестуванням веб-додатків на проникнення слід дослідити як саме працює встановлений брандмауер [36].

2.3.3 Змішана модель

Змішані моделі — це комбінація як позитивних, так і негативних моделей (тобто вони мають функції чорного списку та ефективно взаємодіють з користувачем. Більшість комерційних WAF сьогодні в цій моделі. Вони мають позитивні риси і деякі помилкові позитиви. Наприклад, якщо брандмауер має хороший механізм фільтрації, але він не справляється з уразливістю переповнення буфера.

Деякі брандмауери відкриваються під час механізму фільтрації, що полегшує зловмиснику пошук експлойтів, що відповідають компанії, що розробляє брандмауер, і може зосередитися лише на своїй інфраструктурі, щоб знайти лазівки в програмі. Існує багато способів визначити тип брандмауера, який використовується у веб-додатку, який ми називаємо методами виявлення WAF [36].

Висновки за розділом 2

У розділі 2 було наведено основні способи і методи упередження та захисту від потенційно небезпечних вразливостей, що були наведені в минулому розділі. Додатково було розглянуто веббрандмауер, як один із додаткових рішень, що доповнюють комплексний захист системи від перелічених атак та експлоїтів.

Таким чином, задля уникнення порушеного контролю доступом варто хоча б забороняти за замовчуванням доступ на усі ресурси, окрім тих, що мають бути загальнодоступними, довіряти лише автентифікації та авторизації на стороні сервера, оскільки вона застосовує однакові елементи керування до всіх служб, користувачів і програм.

Для запобігання криптографічні збоїв необхідно шифрувати всі конфіденційні дані, використовувати правильне керування ключами, уникати застарілих криптографічних функцій, таких як MD5, SHA1.

Методами захисту від ін’єкцій є використання підготовлених SQL виразів, перевірка введених користувачем даних на сервері, екранування спеціальних символів в динамічних запитах.

Щоб запобігти помилок в ідентифікації та аутентифікації важливо не створювати облікові записи за замовчуванням, особливо для адміністраторів та при створенні користувацького запису узгодити перевірку на складність паролю та рекомендувати змінювати пароль повторно через певний період часу. Додатково необхідно запровадити багатофакторну аутентифікацію, щоб запобігти автоматичному підбору облікових даних.

Брандмауер веб-програм допомагає захистити веб-програму від зловмисного HTTP-трафіку. Встановивши бар’єр фільтрації між цільовим сервером і зловмисником, WAF може захистити від атак, що здійснюються на веб-додатки.

Веб-брандмауер захищає ваші веб-програми, фільтруючи, відстежуючи та блокуючи будь-який зловмисний HTTP/S-трафік, що надходить до веб-програми, і запобігає виходу будь-яких несанкціонованих даних із програми. Це робиться завдяки набору правил, що допомагають визначити, який трафік є шкідливим, а який безпечним.

РОЗДІЛ 3
ПРОГРАМНИЙ ЗАСТОСУНОК ЗАХИСТУ ВЕБ-ДОДАТКУ

У практичній частини розробимо веб-додаток для демонстрації ймовірних вразливостей, а потім використаємо рекомендації стосовно захисту та розробимо програмні модулі, що допоможуть захиститися від знайдених вразливостей.

3.1 Архітектура та технологічний стек створеного додатку

Веб-додаток являє собою блог або просту соціальну мережу, де користувачі можуть публікувати свої пости, а інші коментувати їх. Неавторизовані користувачі не мають доступу до сторінки із постами. У адміністраторів, спеціальних користувачів, створених через базу даних, є доступ таблиці всіх користувачів на окремій сторінці.

На стороні клієнта використовується Javascript без додаткових фреймворків, звичайний HTML та бібліотека Bootstrap [37] задля швидкої стилізації компонентів без необхідності заздалегідь створювати дизайн сторінки.

На стороні сервера використовується Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою Javascript [38]. Важливими особливостями Node.js є швидкість розробки, розвинута екосистема, а саме велика кількість бібліотек від інших розробників,

В ролі бази даних обрано PostgreSQL, адже PostgreSQL постачається з багатьма функціями, які допомагають розробникам створювати програми, адміністраторам захищати цілісність даних і створювати стійкі до відмов середовища, а також допомагають керувати даними незалежно від того, наскільки великий чи малий набір даних [39].

В базі даних існує дві таблиці – користувачі та пости з блогу.

Структура таблиці користувачів має наступний вигляд (рис 3.1):

#	column_name	data_type
1	id	int4
2	name	varchar
3	password	varchar
4	role	varchar
5	csrf_token	varchar

Рисунок 3.1 – Параметри користувачів в таблиці – ідентифікатор, ім’я, пароль, його роль та збережений токен проти міжсайтової підробки запитів.

Таблиця постів блогу має наступну структуру (рис. 3.2):

#	column_name	data_type	is_nullable
1	id	int4	NO
2	text	varchar	YES
3	userId	int4	YES

Рисунок 3.2 – Параметри постів в таблиці – ідентифікатор, вміст посту та ідентифікатор користувача-автора.

Важливим моментом є те як проводиться авторизація користувачів. Для цього використовуються JSON Web Token (далі JWT або токен) - компактний спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити й довіряти їй, оскільки вона має цифровий підпис. Авторизація найпоширеніший сценарій використання JWT. Щойно користувач увійде в систему, кожен наступний запит включатиме JWT, що дозволить користувачеві отримати доступ до сторінок, служб і ресурсів, які дозволені цим токеном [40]. Цей токен буде зберігатися в файлах cookie браузера.

3.2 Встановлення вразливих місць в веб-додатку

3.2.1 Приклад вразливої для SQL ін’єкції логіки додатку

Для створення вразливості SQL ін’єкціями достатньо коду, що не розмежує ввід користувача та запит, що надсилається до бази даних та виконується. Таким чином, зловмисник має змогу доповнювати вже створений запит таким чином, аби додаток виконував іншу команду. Наприклад, маємо наступний запит до бази даних (рис 3.3):

```
const user = await client.query(
  `SELECT * FROM "public"."users" where "name" ='` + name + `';`,
);
```

Рисунок 3.3 – Код для додання нового запису в базу даних

В даному випадку:

- user – результат виконання команди;
- await client.query – функція надсилання запиту до бази даних та очікування відповіді асинхронної функції
- (`SELECT * FROM "public"."users" WHERE “name” = + name + ';'` - запит, де name – це введений користувачем логін.

Проблема даної конструкції в тому, що name вставляється одразу в запит і ніяк не перевіряється чи фільтрується. Це може призвести до того, що користувач введе наступний текст у поле вводу:

someUsername’; DROP TABLE “public”.”users”; --

Тобто, зловмисник самостійно закриває лапки «‘» і додає кому з крапкою «;», що означає кінець запиту. Після чого сам вводить свою команду, якою скидає таблицю. В кінця ставить «--» задля того, аби закоментувати іншу частину запиту.

Таким чином, SQL ін’єкція можлива на сторінці аутентифікації користувача (рис 3.4).

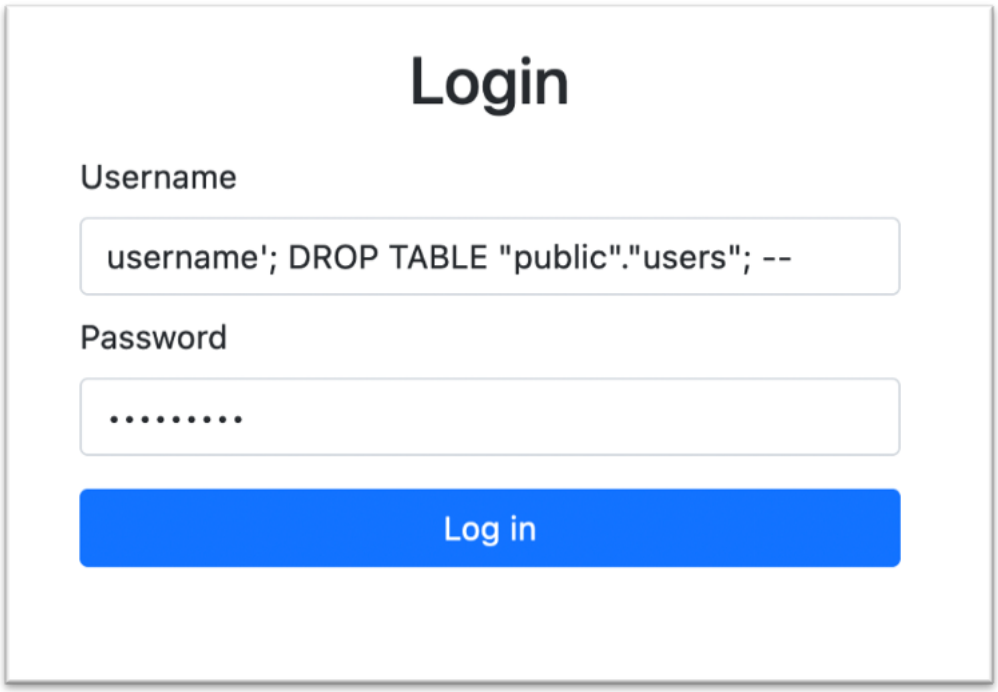

Рисунок 3.4 – Форма даних користувача, де у полі введення логіну користувача можлива SQL-ін’єкція

3.2.2 Приклад вразливої для XSS ін’єкції логіки додатку

Змоделюємо ситуації, в яких зловмисник може скористатися XSS вразливостями. Частіше за все проблема полягає у використані небезпечних функцій чи конструкцій, що записують текст напряму в html-код сторінки. Це, наприклад, параметр innerHTML, що використовується у Javascript чи конструкція {{! !}} із PHP. Одним із прикладів можливого місця для вразливості є пошук на сайті. Якщо необхідно вивести ввід користувача на сторінці, то розробник може скористатися записом в html-код і створити таким чином прогалину в безпеці додатку.

Розглянемо наступну логіку додатку (рис 3.5):

```
const parsed = parseQuery(location.search);
const searchResult = document.querySelector('#search-result');
searchResult.innerHTML = parsed.query;
```

Рисунок 3.5 – Код додання пошукового запису на сторінку

В даному випадку:

- parsed – це об’єкт, що містить всі параметри передані в пошуковій строці, наприклад якщо запит має вигляд “?id=1001”, то об’єкт має вигляд { ‘id’ : ‘1001’ }.
- Далі береться параметр query із цього parsed, та записується всередину елементу searchResult.

Тепер зловмисник має змогу зробити пошуковий запит (рис 3.6) із своїм шкідливим кодом та скинути утворене посилання кому завгодно і виконати цей код на його пристрої (рис 3.7). Це приклад відображеного XSS.

Наприклад, посилання може мати наступний вигляд:

https://vulnerable-site.com/posts/?query=%3Cb%20onmouseover=%22alert(%27This%20is%20XSS!%27)%22%3EHover%20the%20mouse%20for%20XSS%3C/b%3E

- vulnerable-site.com – адреса сайту,
- /posts – розділ сайту,
- далі йде параметр пошуку ?query=<b onmouseover=`alert('This is XSS!')`>Hover the mouse for XSS, але всі спецсимволи були закодовані, наприклад пробіл – це %20, хоча це не заважає строці експлуатувати вразливість.

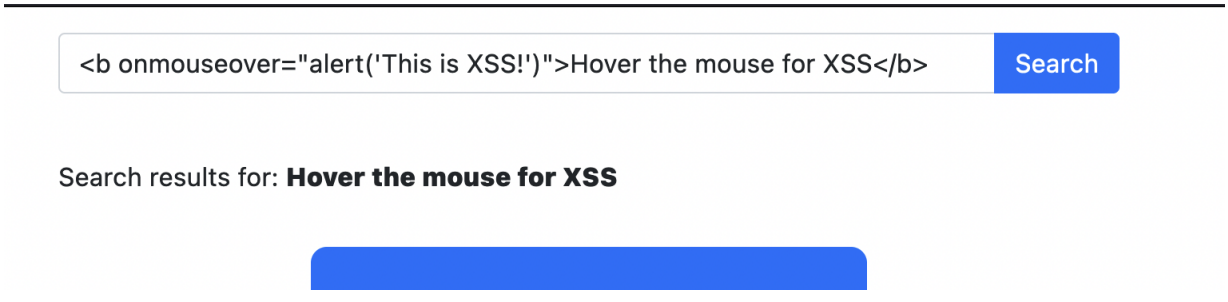

Рисунок 3.6 – Вразлива строка у пошуковій строці, нижче сформована XSS-ін’єкція

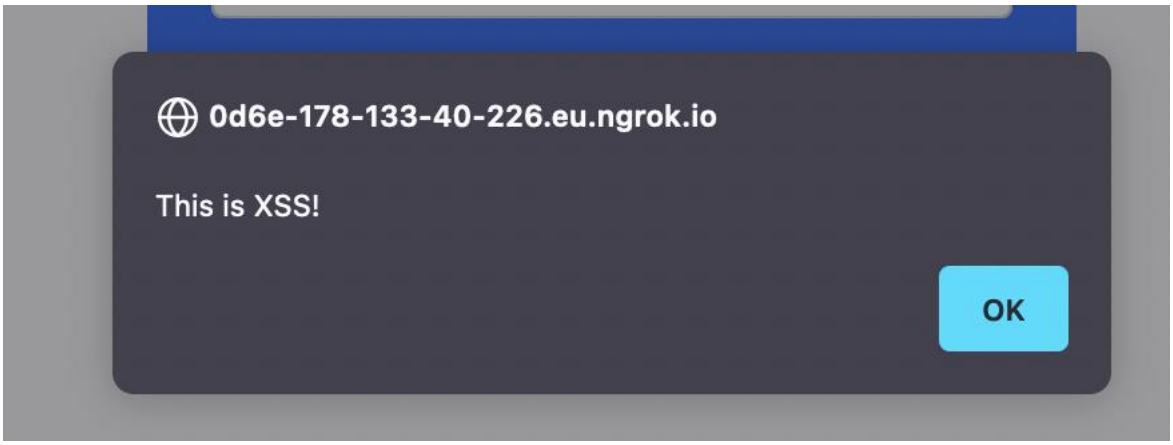

Рисунок 3.7 – Виконання зазначених дій скрипта при наведені на текст

Ще один приклад можливий у текстових полях, які дають користувачу можливість самостійно стилізувати вміст тексту за допомогою html розмітки.

Наприклад, наступний код (рис 3.8):

```
const postElement = document.createElement('div');
const postHTML = `
  <div class="card">
    <div class="card-header">Blog title by <b>${post.name}</b></div>
     <p class="card-text">${post.text}</a>
  </div>`;

postElement.innerHTML = postHTML;
posts.prepend(postElement);
```

Рисунок 3.8 – Код запису посту на сторінку

В даному фрагменті створюється елемент div, описується формат і вигляд розмітки у postHTML, а потім ця розмітка додається до загального списку постів. Це приклад збереженого XSS.

Одним із можливих застосувань XSS вразливостей є крадіжка cookie користувача. У разі виконання якогось скрипту на боці користувача, зловмисник може відправити собі cookie користувача, наприклад, вставивши у сторінку картинку та вказати свій сервер на місце її зберігання. Таким чином можна зробити запит без відома користувача:

- hacker.com – зловмисний веб-сервер
- document.cookie – об’єкт, де міститься строка із усіма cookie користувача.

3.2.3 Приклад вразливої для CSRF Attack логіки додатку

Для атаки підробки запиту між різними сайтами достатньо, щоб користувач був автентифікований на сторінці, тобто мав збережений певний токен в сховищі браузера. Звісно, веб-додаток може не зберігати дані в сховищі браузера користувача, але таки чином значно погіршиться зручність додатку і клієнт може відмовитися від його використання на користь альтернативи.

Для проведення атаки створимо ще одну сторінку, hacker.html (рис 3.9).

```
<html>
<body>
  Click to get CSRF'ed!
  <form name="csrf_form" action="https://vulnerable-site.com/api/posts" method="POST">
    <input type="hidden" name="text" value="I WAS HACKED!">
  </form>

  <button id="csrf">Click me!</button>

  <script type="text/javascript">
    var button = document.querySelector('#csrf');
    button.addEventListener('click', function() {
      document.csrf_form.submit();
    });
  </script>
</body>
</html>
```

Рисунок 3.9 – Код сторінки для проведення атаки CSRF

Сторінка буде мати кнопку, при натисканні якої буде відбуватися запит на сторінку, де користувач вже автентифікований. В нашому випадку, він буде одразу відсилати запит до серверу і цей процес нічим не відрізняється від запиту на сторінку.

Елемент <form> містить посилання за яким відбувається запит - https://vulnerable-site.com/api/posts та метод POST. Всередині <form> елемент міститься елемент <input>, який містить дані для запиту. В даному випадку це text “I WAS HACKED”. Нижче скрипт, що по натисканні кнопки відправляє форму.

В даному випадку зловмисна сторінка просто створить запис від імені користувача із вказаним текстом, що не несе великої шкоди, але знаючи цю вразливість з’являється можливість, наприклад, надсилати фішингові листи власникам акаунтів із великою аудиторією та нашкодити їх репутації, поширивши певну дезінформацію.

3.2.4 Приклад логіки додатку, що має порушений контроль доступу

Порушений контроль доступу в веб-додатку може дати змогу переміщатися між сторінками незважаючи на доступи та ролі користувача, що є дуже небезпечний для системи. Для створення подібної вразливості важливо не наявність якогось коду, що створює вразливість, а відсутність коду, що перевіряє привілеї користувача.

Наприклад, якщо неавторизований користувач знає посилання для панелі адміністратора і там буде відсутня перевірка, то він може просто перейти за посиланням і це призведе до витоку даних.

3.3 Застосування методів захисту від знайдених вразливостей

3.3.1 Методи захисту від SQL ін’єкцій

Так як до цього ми створювати запити до бази даних просто вставляючи дані, введені користувачем, то варто почати використовувати перевірку цих даних.

Першим більш дієвим способом є використання параметризованих запитів. Наприклад, перепишемо запит пошуку користувача (рис 3.10):

```
const user = await client.query(
  `SELECT * FROM "public"."users" where "name" ='` + name + `';`,
);
```

Рисунок 3.10 – Код пошуку користувача в базі даних

Із використанням параметрів запит виглядатиме наступним чином (рис 3.11):

```
const user = await client.query(
  `SELECT * FROM "public"."users" where "name" = $1;`, [name]
);
```

Рисунок 3.11 – Код пошуку користувача із використанням параметрів

Бібліотека node-postgres, яку було використано для роботи із базою даних підтримує параметризовані запити, передаючи текст запиту без змін, а також параметри на сервер PostgreSQL, де параметри безпечно підставляються в запит за допомогою перевірених методів заміни параметрів на самому сервері.

3.3.2 Усунення XSS вразливостей з веб-додатку

Для запобігання XSS вразливостей як і для SQL ін’єкцій слід фільтрувати вміст користувацьких запитів аби вони не містили сценарії, що можуть виконуватися на стороні інших користувачів.

Для цього створимо допоміжну функцію (рис 3.12), що буде переводити всі спеціальні символи у кодів символів для HTML. Для відокремлення спеціальних

символів від звичайних використаємо регулярний вираз «[^\w.]», що означає «не буквені символи, не пробіл і не крапка».

```
// converts all special symbols to HTML Decimal Codes
const htmlSanitize = (str) => {
  return String(str).replace(/[^\w. ]/gi, function(c){
      return '&#'+c.charCodeAt(0)+';';
  });
}
```

Рисунок 3.12 – Код функція фільтрації спеціальних символів

Зі сторони клієнта тепер ми будемо використовувати цю утиліту задля фільтрації небажаного вмісту в місцях, де ми вставляємо новий код у сторінку.

Якщо раніше текст посту із блогу вставлявся таким чином (рис. 3.13):

<p class="card-text">${post.text}

То тепер можемо відфільтрувати його та позбутися від збережених раніше зловмисних строк (рис. 3.14):

<p class="card-text">${htmlSanitize(post.text)}.

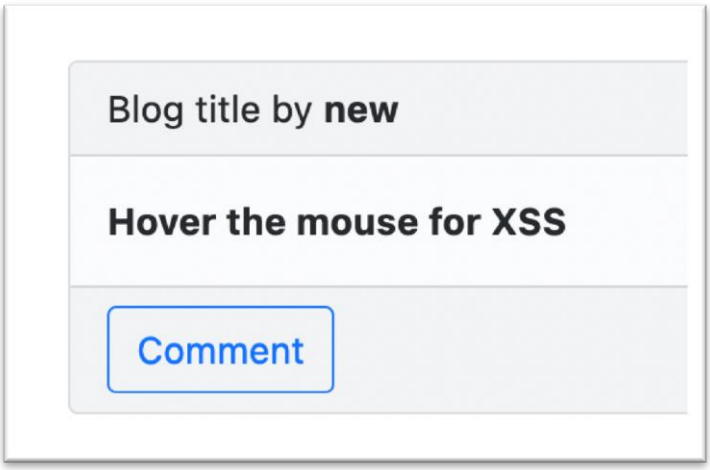

Рисунок 3.13 – Текст відображається на сайті враховуючи html-теги і параметр onmouseover, тобто при наведені мишкою буде виконано скрипт

Рисунок 3.14 – Текст після фільтрації відображається не враховуючи html-теги, скрипт при наведені не спрацьовує

Також необхідно посилити захист cookie користувача. Задля унеможливлення крадіжки cookie необхідно додати захисні параметри при створенні cookie. Тобто, генерації cookie має наступний вигляд (рис 3.15):

```
res.cookie('token', accessToken, {
  maxAge: 900000,
  httpOnly: true,
  secure: true,
  sameSite: true,
});
```

Рисунок 3.15 – Код генерації cookie для користувача із безпечними параметрами

- res.cookie – функція, що встановлює cookie користувачу;
- accessToken – саме значення cookie;
- maxAge – визначає термін придатності cookie;
- secure – передача cookie можлива тільки через безпечний протокол https;
- httpOnly – передача cookie можлива тільки у http запит, відтепер доступ із коду через document.cookie не є можливим;

- sameSite – cookie не буде використовувати, якщо першопочатковий запит був відправлений з іншого сайту.

3.3.3 Використання токену задля усунення вразливості до CSRF атаки

У випадку, коли користувач переходить на сторінку, яка потім виконує дії на іншому сайті від його імені, варто додати токен, який буде у користувача тільки в тому випадку, коли він зайшов на сторінку самостійно. Зловмисник не зможе отримати подібний токен, якщо він не матиме даних самого користувача для авторизації.

Для цього створимо новий шлях для запитів на сервері (рис. 3.16):

```
router.post('/csrf-token', async (req, res) => {
  const { client } = req.context;
  const { token } = req.cookies;

  if (!token) {
    return res.sendStatus(401);
  }

  const user = await getDataFromToken(token);

  if (!user) {
    return res.status(401).send('User was not extracted from token');
  }

  const { userId } = user;
  const csrfToken = genereateCsrfToken(userId);

  await client.query(
    `UPDATE "public"."users" SET "csrf_token" = $1 WHERE "id" = $2;`, [csrfToken, userId]
  );

  return res.send({ csrfToken });
});
```

Рисунок 3.16 – Логіка генерування CSRF-токену на сервері

Тут ми отримаємо client для роботи з базою даних та токен користувача із куків. Якщо нема токену або користувача не знайдено, то відправимо повідомлення

про помилку. Потім ми отримуємо ідентифікатор користувача та генеруємо новий токен саме для запобігання CSRF. Навіть якщо хакер отримуємо цей ідентифікатор, він не зможе створити подібний токен для CSRF, адже йому не вистачатиме секретного значення, що використовується при створені.

В кінці ми записуємо це значення у базу даних, щоб можна було порівняти при відправці форми та відправляємо клієнту токен для CSRF.

Після цього з клієнтської сторони можемо передати цей створений токен для CSRF у заголовку (рис. 3. 17):

```
await axios.post(`${apiUrl}/posts`,
  { text: value }, { headers: { 'X-CSRF-TOKEN': csrfToken }}
);
```

Рисунок 3.17 – Код відправки токену з клієнтської сторони

В даному випадку:

- axios.post – функція відправки запиту;
- ${apiUrl}/post – шлях запиту
- headers: { 'X-CSRF-TOKEN': csrfToken } – спеціальний заголовок у запиті, де csrfToken отриманий раніше токен від CSRF.

Цей функціонал має бути використано у всіх важливих запитах, інакше сервер буде викликати помилку і не опрацьовувати запит.

3.3.4 Використання програмного модулю для запобігання порушеного контролю доступу

Аби не надавати користувачу доступ до тих ресурсів до яких в нього немає доступу будемо використовувати підхід «білого списку», тобто не будемо

забороняти певні ресурси, а навпаки визначимо ті, до яких доступ буде надано. Інші ресурси, що не буде визначено, будуть вважатися недоступними з самого початку.

Спочатку створимо об’єкт, що буде містити налаштування (рис 3.18).

```
const firewallConfig = [
  {
    route: '/',
    isAuthNeeded: false,
    acceptedRoles: ['user', 'admin'],
  }, {
    route: '/login',
    isAuthNeeded: false,
    acceptedRoles: ['user', 'admin'],
  }, {
    route: '/posts',
    isAuthNeeded: true,
    acceptedRoles: ['user', 'admin'],
  }, {
    route: '/dashboard',
    isAuthNeeded: true,
    acceptedRoles: ['admin'],
  }];
```

Рисунок 3.18 – Налаштування програмного модулю для запобігання порушеного контролю доступу

- route – шлях певної сторінки на сайті;
- isAuthNeeded – чи має користувач бути авторизованим на сайті;
- acceptedRoles – ролі користувачів, що мають доступ до сторінки. Існує дві ролі – користувач та адміністратор.

Далі проведемо перевірку всіх необхідних параметрів відвідувача сторінки. Даний програмний модуль – це middleware, тобто такий модуль, що отримує кожний запит на сервері та може корегувати, додавати нові дані в запит або і взагалі скидати виконання запиту.

Спочатку знайдемо конфігурації для певної сторінки (рис 3.19). Якщо її не знайдено – повертаємо сторінку помилку із повідомленням «Page not found».

```
const pathConfig = firewallConfig.find(config ⇒ config.route === path);
if (!pathConfig) {
  return renderError('Page not found');
}
```

Рисунок 3.19 – Пошук необхідного налаштування для певної сторінки

Далі згідно до знайденої конфігурації перевіряємо чи необхідна користувачу авторизація на цій сторінці (рис. 3.20). Якщо в користувача не збережено токену для ідентифікації або якщо токен неправильний чи в нього пройшов термін придатності, повертаємо сторінку помилки.

Далі слідує перевірка чи наявні роль користувача в списку дозволених ролей (рис. 3.20).

```
if (pathConfig.isAuthNeeded) {
  const { token } = req.cookies;

  if (token == null) return renderError('You are not authorized');

  const user = await getDataFromToken(token);

  if (user === null) {
    return renderError('Could not extract user from token');
  }

  if (!pathConfig.acceptedRoles.includes(user.role)) {
    return renderError('You are not allow to access this page');
  }
}
```

Рисунок 3.20 – Код перевірки наявності необхідних доступ до сторінки

Висновки за розділом 3

У розділі 3 було побудовано веб-додаток та виділено його вразливості, а саме можливість SQL та XSS ін’єкцій, CSRF атаки та відсутність належного контролю доступом для користувачів. Технологічний стек додатку складається з Javascript, HTML та CSS, Bootstrap на клієнтському рівні, Node.js на серверному, у ролі бази даних обрано PostgreSQL.

Імітації вразливостей було приділено велику увагу, адже побачивши як саме можна зламати веб-додаток, потім можна і побачити чи правильно його буде захищено від подібних випадків в майбутньому. Для початку було створено SQL ін’єкцію на сторінці авторизації користувача. Подібною атакою можна спробувати видалити цілу таблицю або спотворити вже існуючу. Далі було відтворено можливі місця для XSS ін’єкції, найбільш розповсюдженої вразливості в веб-додатках, а саме створено можливість виконувати введений скрипт в змісті постів та в результаті пошукового запиту. Ще одним слабким місцем веб-додатку була форма, де можливим є CSRF атака. Таким чином можна було від імені користувача публікувати любий пост, якщо вона зайде на зловмисну сторінку, спеціально створену для експлуатації. Також в веб-додатку був порушений контроль доступу, при ньому неавторизований користувач міг заходити на сторінки, які мали б бути недоступні йому.

Після аналізу зазначених слабких місць веб-сторінки, було створено декілька допоміжних функцій, наприклад для екранування спеціальних символів у вводі користувача, програмний модуль контролю доступом, що спирається на токен JWT користувача та на його роль, використано параметризовані запити до бази даних. Загалом усі знайдені раніше вразливості було «знешкоджено».

ВИСНОВКИ

У дипломній роботі було побудовано веб-додаток, проведено аналіз його слабких місць та використано програмні модулі задля захисту від знайдених вразливостей. Таким чином було зменшено ризик SQL та XSS ін’єкцій, атаки CSRF та експлуатації порушеного контролю доступом.

Виходячи із поставленої мети дипломної роботи були виконані наступні завдання:

- досліджено структуру веб-додатків, а саме розглянуто клієнтську, серверну частини та базу даних, виділено їх особливості та характеристики;
- проведено аналіз найбільш поширених вразливостей, характерних для визначеної структури;
- побудовано веб-додаток та визначено в ньому можливі вектори атаки;
- використано програмні засоби та механізми захисту задля унеможливлення експлуатації визначених вразливостей. Таким чином, закрито можливість виконувати власні запити в формі авторизації, вставляти власний скрипт, який може бути виконано на пристрої жертви, в поле пошуку та поле створення нового посту, виконувати запити від імені користувача, якщо він перейде на зловмисну сторінку. Ще було обмежено доступ до зазначених в конфігурації сторінок користувачам, що не авторизовані в системі або не мають певну роль для цієї сторінки;

Всі задачі були виконані в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a Website Vulnerability and How Can it be Exploited? [Електронний ресурс]. – Режим доступу: https://www.sitelock.com/blog/what-is-a-website-vulnerability/

2. 50 Web Security Stats [Електронний ресурс]. – Режим доступу: https://expertinsights.com/insights/50-web-security-stats-you-should-know/

3. 7 Common Web Security Threats for an Enterprise [Електронний ресурс]. – Режим доступу: https://www.fortinet.com/resources/cyberglossary/web-security-threats

4. S. K. Lala, A. Kumar. Secure Web development using OWASP Guidelines / Shubham K. Lala // 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS). - 2021. – P. 323-325

5. Конституція України, редакція від 01.01.2020 [Електронний ресурс]: Закон України № 254к/96-ВР від 01.01.2020. – Режим доступу: https://zakon.rada.gov.ua/laws/show/254%D0%BA/96-%D0%B2%D1%80#Text

6. Про інформацію [Електронний ресурс]: Закон України № 2657-XII від 01.01.2022. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2657-12#Text

7. Про захист інформації в інформаційно-комунікаційних системах [Електронний ресурс]: Закон України № 80/94-ВР від 01.01.2022. – Режим доступу: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text

8. Web Application Architecture [Електронний ресурс]. – Режим доступу: https://stackify.com/web-application-architecture/

9. Web Application Architecture from 10,000 Feet [Електронний ресурс]. – Режим доступу: https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/

10. Database Applications and the Web [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/web-database-applications/0596005431/ch01.html

11. SQL & Databases for Web Development [Електронний ресурс]. – Режим доступу: https://www.codingninjas.com/blog/2020/11/28/sql-databases-for-web-development/

12. What is NoSQL? [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/nosql-explained

13. What is OWASP? What is the OWASP Top 10? [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/security/threats/owasp-top-10/

14. How Akamai Augments Your Security Practice to Mitigate the OWASP Top 10 Risk [Електронний ресурс]. – Режим доступу: https://www.akamai.com/site/en/documents/white-paper/how-akamai-augments-your-security-practice-to-mitigate-the-owasp-top-10-risks.pdf

15. 2021 OWASP Top 10 Update [Електронний ресурс]. – Режим доступу: https://www.pentasecurity.com/blog/2021-owasp-top-10-update/

16. Snyk’s Skyrocketing Valuation [Електронний ресурс]. – Режим доступу: https://www.wsj.com/articles/snyks-skyrocketing-valuation-reflects-growing-interest-in-late-stage-enterprise-businesses-11579650334

17. Cybersecurity co Snyk raises $300m [Електронний ресурс]. – Режим доступу: https://en.globes.co.il/en/article-cybersecurity-co-snyk-raises-300m-at-85b-valuation-1001384232

18. Why Open Source [Електронний ресурс]. – Режим доступу: https://pimcore.com/en/why-open-source

19. State of Open Source Security Report 2020 [Електронний ресурс]. – Режим доступу: https://go.snyk.io/rs/677-THP-415/images/State%20Of%20Open%20Source%20Security%20Report%202020.pdf

20. Broken Access Control [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

21. Cryptographic Failures [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

22. SQL Injection [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/sql-injection

23. Injection [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/A03_2021-Injection/

24. Security Misconfiguration [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

25. Identification and Authentication Failures [Електронний ресурс]. – Режим доступу: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

26. DDoS Attack Protection [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/shield/ddos-attack-protection/

27. What is a DDoS Attack? [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

28. CSRF – Cross Site Request Forgery [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

29. Cross-site request forgery (CSRF) [Електронний ресурс]. – Режим доступу: https://portswigger.net/web-security/csrf

30. Broken Access Control and How to Prevent It [Електронний ресурс]. Режим доступу: https://crashtest-security.com/broken-access-control-prevention/#preventing-broken-access-control-vulnerabilities

31. Using Prepared Statements as SQL Injection Prevention [Електронний ресурс]. – Режим доступу: https://www.invicti.com/blog/web-security/sql-injection-vulnerability/#UsingPreparedStatements

32. SQL Injection (SQLi) Attack [Електронний ресурс]. – Режим доступу: https://snyk.io/learn/sql-injection/

33. What is Web application security [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/security/what-is-web-application-security/

34. What is a Web Application Firewall (WAF)? [Електронний ресурс]. – Режим доступу: https://www.f5.com/services/resources/glossary/web-application-firewall

35. 8 Types of Firewall [Електронний ресурс]. – Режим доступу: https://phoenixnap.com/blog/types-of-firewalls

36. K. Nagendran, S. Balaji, B. A. Raj, P. Chanthrika and R. G. Amirthaa, "Web Application Firewall Evasion Techniques" / K. Nagendran // 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). – 2020. - P. 194-199.

37. Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/

38. Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/uk/

39. Why use PostgreSQL? [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/about/

40. What is JSON Web Token? [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction