**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**
**Факультет кібербезпеки та програмної інженерії**
**Кафедра інженерії програмного забезпечення**

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

______Катерина НЕСТЕРЕНКО
“____”___________2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА

## (ПОЯСНЮВАЛЬНА ЗАПИСКА)

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ**
**МАГІСТРА**

**Тема:** “Методика розробки бібліотеки для аналізу та вдосконалення безпеки фронтенд додатків”

**Виконавець:** Литвин Віталій Віталійович

**Керівник:** к.т.н доцент Борковська Любов Олексіївна

**Нормоконтролер:** ассистент Кравченко Ольга Сергіївна

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

**Факультет** кібербезпеки та програмної інженерії
**Кафедра** інженерії програмного забезпечення
**Освітній ступінь** магістр
**Спеціальність** 121 Інженерія програмного забезпечення
**Освітньо-професійна програма** «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _______ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Литвина Віталія Віталійовича

1. Тема кваліфікаційної роботи: «Методика розробки бібліотеки для аналізу та вдосконалення безпеки фронтенд додатків»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання проекту: з 02.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи : програмний продукт розробити мовою програмування JavaScript та NodeJS.

4. Зміст пояснювальної записки:
1. Аналіз безпеки фронт-енд застосунків та можливих атак.
2. Існуючі інструменти та методики для запобігання атак .
3. Аналіз структури та розробка прототипу програмного засобу.

5. Перелік обов'язкових слайдів презентації:
1. Предмет та об’єкт дослідження
2. Введення в безпеку фронт-енд застосунків
3. Cross-Site Scripting
4. DDoS та DoS
5. SQL Ін’єкції
6. Викрадення сеансу
7. Опис рекомендацій щодо усунення виявлених у ході дослідження проблем
8. Опис розробленого застосунку

6. Календарний план-графік

| № пор. | Завдання | Термін виконання | Відмітка про виконання |
|---|---|---|---|
| 1. | Розробка та затвердження графіка роботи | 2.10-8.10 | + |
| 2. | Підготовка та написання 1 розділу. Відсилка керівнику | 9.10-30.10 | + |
| 3. | Підготовка та написання 2 розділу. Відсилка керівнику | | + |
| 4. | Підготовка та написання 3 розділу. Відсилка керівнику | 01.12-10.12 | + |
| 5. | Редагування та друк пояснювальної записки, графічного матеріалу Відсилка ПЗ для перевірки на плагіат одним файлом. | 10.12-15.12 | + |
| 6. | Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника. Підготовка презентації та тексту доповіді. | 10.12-15.12 | + |
| 7. | Передзахист (наявність віддрукованої ПЗ, презентації, положительного відгуку керівника). При наявності цього приймається рішення про допуск к захисту дипломного проекту перед Екзаменаційною комісією. Що оформлюється протоколом засідання кафедри | 11.12-17.12 | + |
| 8. | Отримання рецензії. | | + |
| 9. | Здати секретарю ДЕК: ПЗ, ГМ, CD-R з електронними версіями ПЗ, ГМ, презентацію, відгук керівника, рецензію, довідку про успішність, 2 папки, 2 конверта) | 18.12-24.12 | + |
| 10. | Захист дипломної роботи перед ЕК | 25.12-31.12 | + |

Дата видачі завдання 02.10.2023 р.

Керівник дипломної роботи: к.т.н. доцент Любов БОРКОВСЬКА

Завдання прийняв до виконання: Віталій ЛИТВИН

# РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Методика розробки бібліотеки для аналізу та вдосконалення безпеки фронтенд додатків»: 71 сторінок, 7 рисунків, 1 таблиця, 9 використаних джерел, 1 додаток.

РОЗРОБНИК, ТЕСТУВАННЯ, ВЕБ БЕЗПЕКА, ОЦІНКА ВРАЗЛИВОСТЕЙ, БЕЗПЕКА ЗАСТОСУНКІВ.

**Об’єкт дослідження** - бібліотека для аналізу та вдосконалення безпеки фронт-енд додатків.

**Мета дипломної роботи** – розробка бібліотеки для аналізу та вдосконалення безпеки фронт-енд додатків, яка буде спрямована на виявлення вразливостей, таких як SQL-ін'єкція та XSS, і розробку контрзаходів для захисту фронт-енд додатків від цих атак.

**Метод дослідження** – Аналіз літературних джерел для вивчення існуючих підходів до розробки бібліотек для аналізу та вдосконалення безпеки фронт-енд додатків. Аналіз програмного коду для виявлення вразливостей у фронт-енд додатках. Проектування та розробка бібліотеки для створення бібліотеки для аналізу та вдосконалення безпеки фронт-енд додатків. Експериментальне дослідження для оцінки ефективності розробленої бібліотеки.

**Результати** роботи можуть бути використані при розробці програмних засобів призначених для тестування фронт-енд додатків де є ризик атак на додаток з метою перешкоджання роботи чи отримання персональної чи фінансової інформації.

Розробка та дослідження проводилися під управлінням ОС Windows 11. Розробка програми проводилася у середовищі Visual Studio Code, на мові програмування JavaScript та NodeJS.

## ABSTRACT

Explanatory note for the diploma thesis "Methodology for the development of a library for the analysis and improvement of the security of frontend applications": 71 pages, 7 figures, 1 table, 9 sources used, 1 appendix.

DEVELOPER, TESTING, WEB SECURITY, VULNERABILITY ASSESSMENT, APPLICATION SECURITY.

**Object of study** - a library for the analysis and improvement of the security of frontend applications.

**The purpose of the diploma** thesis is to develop a library for the analysis and improvement of the security of frontend applications, which will be aimed at detecting vulnerabilities such as SQL injection and XSS, and developing countermeasures to protect frontend applications from these attacks.

**Methodology of research** -Analysis of literary sources - to study existing approaches to the development of libraries for the analysis and improvement of the security of frontend applications. Analysis of program code - to identify vulnerabilities in frontend applications. Design and development of the library - to create a library for the analysis and improvement of the security of frontend applications. Experimental research - to evaluate the effectiveness of the developed library.

**Results** of the work can be used in the development of software tools designed for testing frontend applications where there is a risk of attacks on the application with the aim of disrupting the work or obtaining personal or financial information.

The development and research were carried out under the control of the Windows 11 operating system. The program was developed in the Visual Studio Code environment, in the JavaScript programming language.

**ЗМІСТ**

ЗМІСТ ........................................................ 6
ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ ........................................................ 7
ВСТУП ........................................................ 8
РОЗДІЛ 1. АНАЛІЗ БЕЗПЕКИ ФРОНТ-ЕНД ЗАСТОСУНКІВ ТА МОЖЛИВИХ АТАК ........................................................ 9
1.1. Опис та аналіз можливих ризиків та атак на фронт-енд застосунки ..... 14
1.2. Cross-Site Scripting ........................................................ 15
1.3. Міжсайтова підробка запитів (CSRF) ........................................................ 17
1.4. DoS та DDoS ........................................................ 19
1.5. SQL-ін’єкції ........................................................ 22
1.6. Викрадення сеансу (Session Hijacking) ........................................................ 24
1.7. Дослідження актуальності створення та розвитку рішення для захисту фронтенд додатків ........................................................ 27
Висновки ........................................................ 29
РОЗДІЛ 2. ІСНУЮЧІ ІНСТРУМЕНТИ ТА МЕТОДИКИ ДЛЯ ЗАПОБІГАННЯ АТАК ........................................................ 30
2.1. Веб-розвідка ........................................................ 32
2.2. Дизайн ........................................................ 34
2.3.SQL-ін’єкції ........................................................ 36
2.4. XSS ........................................................ 38
2.5. Викрадення сеансу (Session Hijacking) ........................................................ 40
Висновки ........................................................ 43
РОЗДІЛ 3. АНАЛІЗ СТРУКТУРИ ТА РОЗРОБКА ПРОТОТИПУ ПРОГРАМНОГО ЗАСОБУ ........................................................ 44
1.1.Аналіз вразливостей та розробка засобів запобігання атакам ................ 45
Висновки ........................................................ 58
ВИСНОВКИ ........................................................ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ........................................................ 61
Додаток А ........................................................ 59

## ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ОС – Операційна Система.

API - Application Programming Interface, прикладний програмний інтерфейс.

REST - Representational State Transfer, передача репрезентативного стану.

B2C – Business-to-Consumer, бізнес до споживача.

IP - Internet Protocol, інтернет протокол.

SPA – Single Page Application, одно-сторінковий додаток.

SQL - Structured Query Language, структурована мова запитів.

XSS - Cross-Site Scripting, міжсайтове сценарне впровадження.

DDOS – Distributed Denial-of-Service attack, розподілена атака на відмову в обслуговуванні.

DOS - Denial-of-Service attack, атака на відмову в обслуговуванні.

PII – Personal Identifiable Information, особиста ідентифікаційна інформація.

HTML - HyperText Markup Language, гіпертекстова мова розмітки.

HTTP - Hypertext Transfer Protocol, гіпертекстовий протокол передачі.

SQLIA – SQL Injection Attack, атака SQL-ін’єкцією.

# ВСТУП

У сучасному світі, де більшість наших дій виконується за допомогою комп'ютерів, інформаційна безпека є однією з найважливіших проблем. Кібератаки можуть призвести до крадіжки особистої інформації, фінансових збитків і навіть порушення роботи підприємств.

Фронтенд додатки є одним з найпоширеніших типів програмного забезпечення. Вони відповідають за інтерфейс користувача, який ви бачите, коли взаємодієте з веб-сайтом або додатком. Фронтенд додатки часто розробляються з використанням JavaScript, HTML і CSS.

Фронтенд додатки можуть бути вразливими до кількох типів атак, включаючи:

XSS-атаки (Cross-site scripting) дозволяють хакерам вставляти шкідливий код у веб-сторінки або додатки. Цей код може бути використаний для крадіжки особистої інформації, перехоплення сеансів або виконання інших шкідливих дій.

CSRF-атаки (Cross-site request forgery) змушують користувачів виконувати небажані дії, такі як переказ коштів або зміна налаштувань.

SQL-ін'єкції (SQL injection) дозволяють хакерам отримати доступ до даних бази даних, які зберігаються на сервері.

Щоб захистити фронтенд додатки від цих атак, розробники можуть використовувати різні методи, такі як:

Санітарна обробка вмісту, введеного користувачем, щоб видалити потенційно небезпечні символи.

Використання одноразових токенів для запобігання атакам CSRF. Використання безпечних практик розробки, таких як чітке розмежування ролей і обов'язків. У цій роботі буде розроблена бібліотека для аналізу та вдосконалення безпеки фронтенд додатків.

# РОЗДІЛ 1
# АНАЛІЗ БЕЗПЕКИ ФРОНТ-ЕНД ЗАСТОСУНКІВ ТА МОЖЛИВИХ АТАК

Фронтенд додатки є найважливішою частиною сучасних веб-додатків. Вони відповідають за інтерфейс користувача, який дозволяє користувачам взаємодіяти з додатком. Однак фронтенд додатки також є вразливими до атак, які можуть призвести до крадіжки даних, виконання несанкціонованих дій або інших проблем.

Метою цього розділу є ознайомитися з основними поняттями та термінами, пов'язаними з безпекою фронтенд додатків. Також у цьому розділі необхідно розглянути типові вразливості фронтенд додатків, технології та фреймворки для розробки фронтенд додатків, а також методи захисту фронтенд додатків.

Для початку необхідно ознайомитися з основними поняттями та термінами, пов'язаними з безпекою фронтенд додатків. До таких понять належать:

Frontend - це частина веб-додатку, яка відповідає за інтерфейс користувача. Він складається з HTML, CSS і JavaScript.

HTML - це мова розмітки, яка використовується для створення структури веб-сторінок.

CSS - це мова стилізації, яка використовується для форматування веб-сторінок.

JavaScript - це мова програмування, яка використовується для створення інтерактивних веб-сторінок.

Фреймворк - це набори інструментів і бібліотек, які допомагають розробникам створювати веб-додатки.

Веб-додаток - це програмне забезпечення, яке працює в браузері користувача.

Критична вразливість - це вразливість, яка може бути використана для отримання повного контролю над системою або інформацією.

Зловмисний код - це програмний код, який розроблений для шкоди системі або інформації.

Web Application Firewall (WAF) - це пристрій або програмне забезпечення, яке використовується для захисту веб-додатків від атак.

Фронтенд додатки можуть бути вразливими до різних типів атак. Деякі з найпоширеніших вразливостей включають:

XSS (Cross-site scripting) - це атака, в процесі якої шкідливий код впроваджується в веб-сторінку або веб-додаток.

SQL-ін'єкція - це атака, в процесі якої шкідливий код впроваджується в SQL-запит.

CSRF (Cross-site request forgery) - це атака, в процесі якої користувач змушується виконувати несанкціоновані дії на веб-сайті.

Фішинг - це атака, в процесі якої користувачі обманюються, щоб вони надали свої конфіденційні дані, такі як паролі, номери кредитних карток та інша особиста інформація.

DoS (Denial of Service) - це атака, в процесі якої веб-сервер або мережа перевантажуються, щоб блокувати доступ користувачів.

Технології та фреймворки для розробки фронтенд додатків

Zero-day-атака - це атака, яка використовує вразливість, про яку невідомо розробнику програмного забезпечення.

Сучасні веб-додатки часто будуються на основі технологій, яких не існувало 10 років тому. Інструменти, доступні для створення веб-додатків, настільки просунулися за цей час, що іноді здається, що це зовсім інша спеціалізація сьогодні.

Зовсім не рідкість, коли окремий додаток, розгорнутий у веб-браузері, взаємодіє з безліччю серверів. Візьмімо, наприклад, додаток для розміщення

зображень, який дозволяє користувачам входити в систему. Він, ймовірно, матиме спеціалізований сервер розміщення/розповсюдження за однією URL-адресою та окрему URL-адресу для керування базою даних та входом.

Можна стверджувати, що сучасні додатки часто є поєднанням багатьох окремих, але симбіотичних додатків, які працюють разом. Це можна пояснити розвитком чіткіше визначених протоколів мережі та архітектурних моделей API. Середньостатистичний сучасний веб-додаток, ймовірно, використовує кілька з таких технологій:

• REST API: Інтерфейс для спілкування між додатком і зовнішнім світом, заснований на архітектурному стилі REST, який використовує методи HTTP-запитів.

• JSON або XML: Формати даних для обміну між додатком і зовнішнім світом. JSON є більш компактним і легшим для читання, тоді як XML є більш структурованим і стандартизованим.

• JavaScript: Мова програмування, яка використовується для додавання інтерактивності та динамічності веб-сторінкам.

• SPA framework (React, Vue, EmberJS, AngularJS): Каркас для побудови односторінкових додатків (SPA), які динамічно оновлюють вміст без перезавантаження всієї сторінки.

• Система автентифікації та авторизації: Механізм для перевірки користувачів та надання їм доступу до відповідних ресурсів.

• Один або більше веб-серверів (зазвичай на сервері Linux): Комп'ютери, які обслуговують запити до веб-сторінок та додатків.

• Один або більше пакетів програмного забезпечення веб-сервера (ExpressJS, Apache, NginX): Програми, які працюють на веб-серверах для обробки запитів та доставки відповідей.

• Одна або більше баз даних (MySQL, MongoDB тощо): Системи зберігання та управління організованими даними, необхідними додатку.

• Локальне сховище даних на клієнті (cookies, web storage, IndexDB): Механізми для зберігання даних безпосередньо на пристрої користувача (комп'ютері, телефоні тощо).

Рис. 1.1. Архітектура Веб Додатку

Методи захисту фронтенд додатків

Існує ряд методів, які можна використовувати для захисту фронтенд додатків від атак. Деякі з найпоширеніших методів включають:

Використання сучасних технологій та фреймворків - сучасні технології та фреймворки, як правило, мають вбудовані засоби захисту від атак.

Впровадження заходів безпеки - до таких заходів безпеки належать фільтрація вхідних даних, шифрування даних та аутентифікація користувачів.

Регулярне оновлення програмного забезпечення - розробники часто випускають виправлення для відомих вразливостей.

Небезпека веб-застосунків викликана кількома факторами. По-перше, веб перетворився на складну платформу для додатків, на якій розробляються все більш складні програми. На жаль, проблеми з безпекою також зросли і

розглядаються як вторинні. По-друге, хакери все частіше атакують веб-застосунки, оскільки традиційні форми атак, такі як переповнення буфера, стають складнішими для виконання. По-третє, існуючі технології безпеки, такі як мережеві брандмауери та антивірусне програмне забезпечення, забезпечують порівняно надійний захист на рівні хоста та мережі, але не на рівні програми. Загалом, відкриті інтерфейси веб-застосунків стають ціллю атак, коли мережеві та хост-рівневі точки входу стають відносно безпечними.

Вразливості веб-застосунків важко усунути з двох причин. По-перше, більшість веб-застосунків проходять швидку фазу розробки з надзвичайно коротким часом виконання. По-друге, вони розробляються в межах компанії деякими інженерами з інформаційних систем управління (MIS), більшість з яких мають менше підготовки та досвіду в безпечній розробці програмного забезпечення порівняно з інженерами великих програмних компаній, таких як IBM, Sun, Microsoft та Facebook. Згідно з Web Application Security Consortium , найпоширенішими вразливостями є Впровадження сценаріїв між сайтами (Cross-Site Scripting), Витік інформації (Information Leakage), SQL-ін'єкція (SQL Injection), Недостатній захист транспортного рівня (Insufficient Transport Layer Protection) та (Ідентифікація пристрою) Fingerprinting.

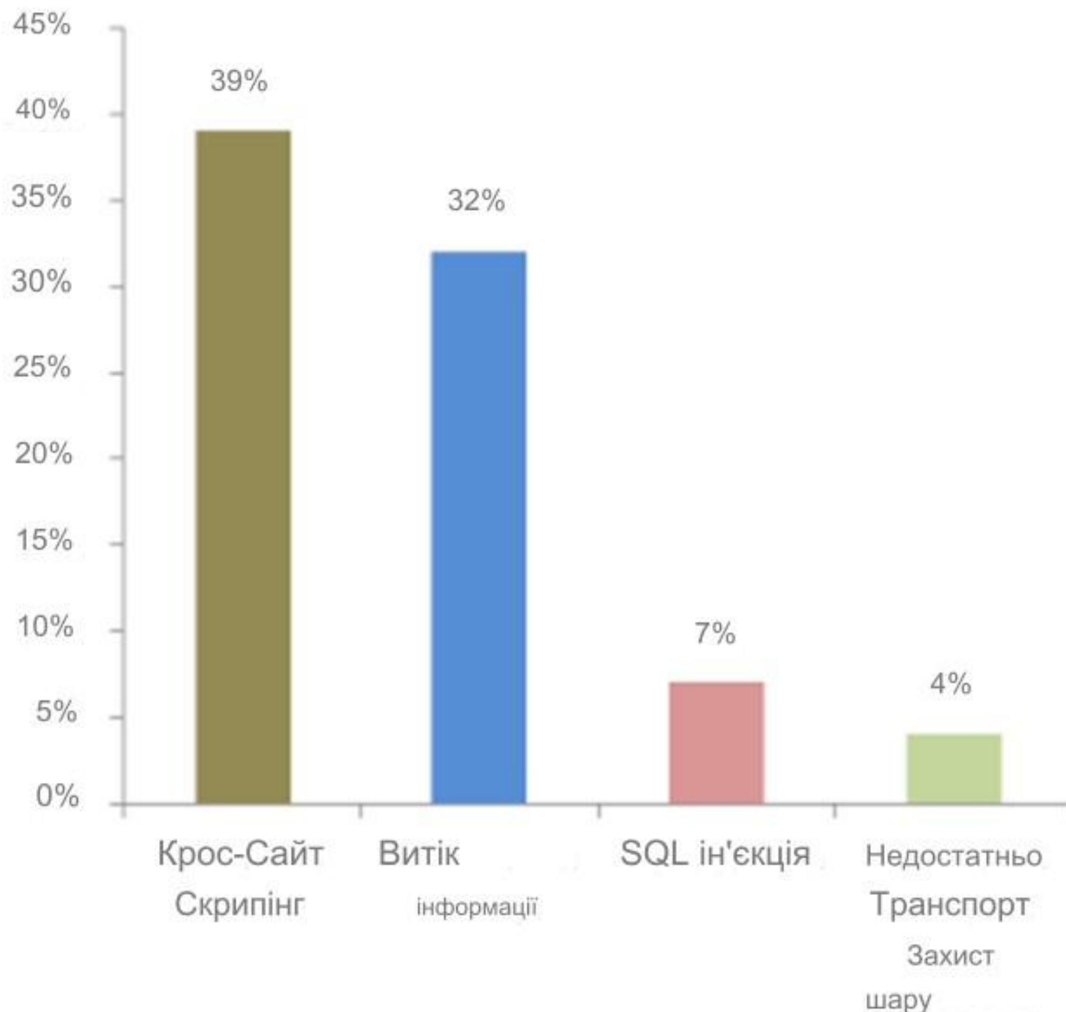


Рис. 1.2. Відсотки найрозповсюджених вразливостей

## 1.1. Опис та аналіз можливих ризиків та атак на фронт-енд застосунки

У кібербезпеці, безпека фронтенду (безпека на стороні клієнта) допомагає запобігати та пом’якшувати інциденти та порушення безпеки, які виникають на фронтенді системи (а не на бекенді або на стороні сервера). Для захисту від атак та порушень даних бізнес повинен захищати як фронтенд, так і бекенд своєї цифрової діяльності. На фронтенді або “на стороні клієнта” це включає все, що бачить клієнт, наприклад, зображення, контент та інші елементи інтерфейсу користувача, а також все, з чим взаємодіє клієнт, наприклад, чат-боти, які

можуть з'являтися в браузері користувача. Оцінюється, що 98% американських сайтів Alexa 1000 не мають належного захисту, який може відбити атаку на фронтенді або на стороні клієнта.

Вразливості фронтенду стають все поширенішими, переважно через зростання кількості бізнесів та кінцевих користувачів, які використовують веб-додатки для обміну конфіденційною та особистою ідентифікаційною інформацією (PII). Оскільки багато веб-сайтів та веб-додатків написані на JavaScript (~98% усіх глобальних веб-сайтів), який також є принципово небезпечною мовою програмування, зловмисники просто експлуатують JavaScript, вбудовуючи шкідливі скрипти в існуючий код. Проблеми безпеки фронтенду також виникають через надмірність скриптів третіх та четвертих сторін, які використовуються для компіляції веб-сайтів.

Одним з небезпечних векторів атаки є XSS. XSS - це зловмисний вектор атаки, який експоненціально зростає за своєю поширеністю. Більше того, ефект від XSS атаки були помічені глобально всім Всесвітнім павутинням. XSS атаки відбуваються майже щодня. Веб-сайти, такі як Twitter, Facebook, Google вже стали жертвами XSS атак.

## 1.2. Cross-Site Scripting

**Cross-Site Scripting** - це атака з введенням коду JavaScript яка дозволяє зловмиснику виконувати введений JavaScript у веб-браузері жертви, щоб отримати доступ до чутливих ресурсів, таких як кукі, паролі, номери кредитних карток тощо. XSS - це атака на веб-браузер на стороні клієнта, але її можливості використовуються на стороні веб-сервера. Для експлуатації XSS-вразливостей на веб-додатках зловмисник створює та вводить шкідливий JavaScript-вантаж на веб-додаток. Цей скрипт вводиться таким чином, що він здається доброзичливим компонентом веб-сайту і, нарешті, цей скрипт виконується в

межах домену довіри до веб-сайту.

Оскільки існує багато технік для введення зловмисний код Java Script у веб-додаток жертви, але найчастіший спосіб, який зловмисник слідує, полягає в тому, що зловмисник вводить скрипт у один з сторінок веб-сайту, щоб жертва завантажила скрипт з веб-сайту. Це можливо тільки за умови, що веб-додаток приймає вхідні дані зі сторони користувача у свої веб-сторінки, оскільки зловмисник може ввести зловмисний рядок JavaScript, який буде відображатися як код у браузері жертви. Оскільки жертва припускає, що повідомлення буде містити тільки текст. Але зловмисник має змогу ввести скрипт: <script>alert(‘‘XSS Exploited’’)<script> у вигляді вхідних даних. Коли браузер жертви натисне на це останнє повідомлення, код JavaScript буде виконуватися автоматично, і в результаті атака XSS буде використана на веб-сторінці жертви.

Для проведення XSS атаки проводяться наступні дії:

По-перше, зловмисник знайшов веб-сайт, який піддається атакам XSS. Зловмисник вводить невідфільтрований зловмисний скрипт на VWA у вигляді останнього повідомлення, метою якого є крадіжка чутливих облікових даних (наприклад, куки) веб-браузера жертви.

Потім веб-браузер жертви переглядає VWA і зловмисний JavaScript надходить у вигляді останнього повідомлення HTTP-відповіді. Зловмисний скрипт тоді виконується в межах поля надійності веб-сайту додатку.

Як тільки зловмисний скрипт буде виконано, облікові дані жертви (наприклад, кукі) можуть бути передані на веб сервер зловмисника.

Згодом, кукі будуть використані зловмисником для перехоплення сесії. Атака XSS зазвичай відбувається в динамічних веб-додатках, які вимагають вводу зі сторони користувача.

Такі атаки мають декілька негативних ефектів які можуть впливати на розробників сайту або безпосередніх користувачів:

Крадіжка cookies - Зловмисник може викрасти cookies проводячи атаки

XSS на вразливий веб додаток для крадіжки ідентифікаторів сесії або сесії перехоплення.

Фішингові атаки - Зловмисник може заставити жертву надати його/її облікові дані, вводячи інформацію в підроблену форму входу, використовуючи можливості DOM маніпулювання, щоб націлитися на веб-сервер.

Кейлогінг - Зловмисник може використовувати можливості відслідковування подій клавіатури, щоб відстежувати всі натискання клавіш жертви та передавати цю інформацію на веб-сервер для доступу до чутливої інформації, такої як пароль, номери кредитних карток тощо.

### 1.3. Міжсайтова підробка запитів (CSRF)

Міжсайтова підробка запитів (CSRF) належить до двадцяти найбільш використовуваних уразливостей, поряд з міжсайтовими сценаріями (XSS) та SQL-ін’єкцією. На відміну від міжсайтових сценаріїв, які привернули велику увагу, та ефективне пом’якшення SQL-ін’єкції за допомогою параметризованих SQL-запитів, міжсайтова підробка запитів отримала порівняно мало уваги. У CSRF-атаці зловмисний сайт інструктує браузер жертви надсилати запит до добросовісного сайту, ніби запит був частиною взаємодії жертви з добросовісним сайтом, використовуючи мережеві з’єднання жертви та стан браузера, наприклад файли cookie, для отримання доступу до сеансу жертви з добросовісним сайтом.

У атаці з підробкою міжсайтового запиту (CSRF) зловмисник порушує цілісність сеансу користувача з веб-сайтом, вводячи мережеві запити через браузер користувача. Політика безпеки браузера дозволяє веб-сайтам надсилати HTTP-запити на будь-яку мережеву адресу. Ця політика дозволяє зловмиснику, який контролює вміст, відображений браузером, використовувати ресурси, які в іншому випадку не знаходяться під його контролем:

Мережеве з'єднання. Наприклад, якщо користувач знаходиться за

брандмауером, зловмисник може використовувати браузер користувача для надсилання мережевих запитів на інші машини за брандмауером, які можуть бути недоступні безпосередньо з машини зловмисника. Навіть якщо користувач не знаходиться за брандмауером, запити містять IP-адресу користувача і можуть сплутати служби, які використовують автентифікацію за IP-адресою.

Читання стану браузера. Запити, надіслані через мережевий стек браузера, зазвичай включають стан браузера, наприклад файли cookie, клієнтські сертифікати або заголовки базової автентифікації. Сайти, які покладаються на цей автентифікаційний стан, можуть бути сплутані цими запитами.

Запис стану браузера. Коли зловмисник змушує браузер створювати мережевий запит, браузер також аналізує та виконує відповідь. Наприклад, якщо відповідь містить заголовок Set-Cookie, браузер змінить своє сховище файлів cookie.

В разі проведення атаки CSRF зловмисники можуть добитися наступних результатів, якщо проведуть атаку CSRF:

2. Переказ коштів з облікового запису жертви. Це можна зробити, наприклад, якщо атакувати банківський веб-сайт і змусити жертву переказати кошти на рахунок зловмисника.

3. Внесення змін до облікового запису жертви. Це можна зробити, наприклад, якщо атакувати соціальну мережу і змінити пароль жертви або її особисту інформацію.

4. Виконання шкідливих дій від імені жертви. Це можна зробити, наприклад, якщо атакувати веб-сайт електронної комерції і змусити жертву зробити покупку у зловмисника.

Ось кілька конкретних прикладів того, чого можуть досягти зловмисники за допомогою атаки CSRF:

Викрадення облікових даних. Зловмисник може створити форму введення облікових даних, яка відправляється на веб-сайт жертви. Коли жертва вводить

свої облікові дані на формі, зловмисник отримує їх.

Передача файлів. Зловмисник може створити посилання на файл, який містить шкідливе програмне забезпечення. Коли жертва натискає на посилання, шкідливе програмне забезпечення завантажується на її пристрій.

Виконання шкідливих дій. Зловмисник може створити форму, яка відправляє запит на веб-сайт жертви, щоб виконати шкідливу дію, наприклад, переказати кошти з облікового запису жертви.

### 1.4. DoS та DDoS

Атака на відмову в обслуговуванні (DoS) – це тип кібератаки, під час якої зловмисник намагається зробити комп’ютерні ресурси недоступними користувачам, для яких комп’ютерні ресурси були призначені. Це можна зробити, надсилаючи надмірну кількість запитів до ресурсу, що перевантажує його і робить його недоступним для законних користувачів.

Розподілена атака на відмову в обслуговуванні (DDoS) – це вид атаки DoS, яка використовує велику кількість комп’ютерів, щоб надсилати запити до ресурсу. Ці комп’ютери можуть бути заражені шкідливим програмним забезпеченням, яке змушує їх надсилати запити, або вони можуть бути підконтрольні зловмиснику.

Атаки DoS і DDoS можуть мати серйозні наслідки для компаній і організацій. Вони можуть призвести до втрати прибутку, пошкодження репутації та навіть порушення закону.

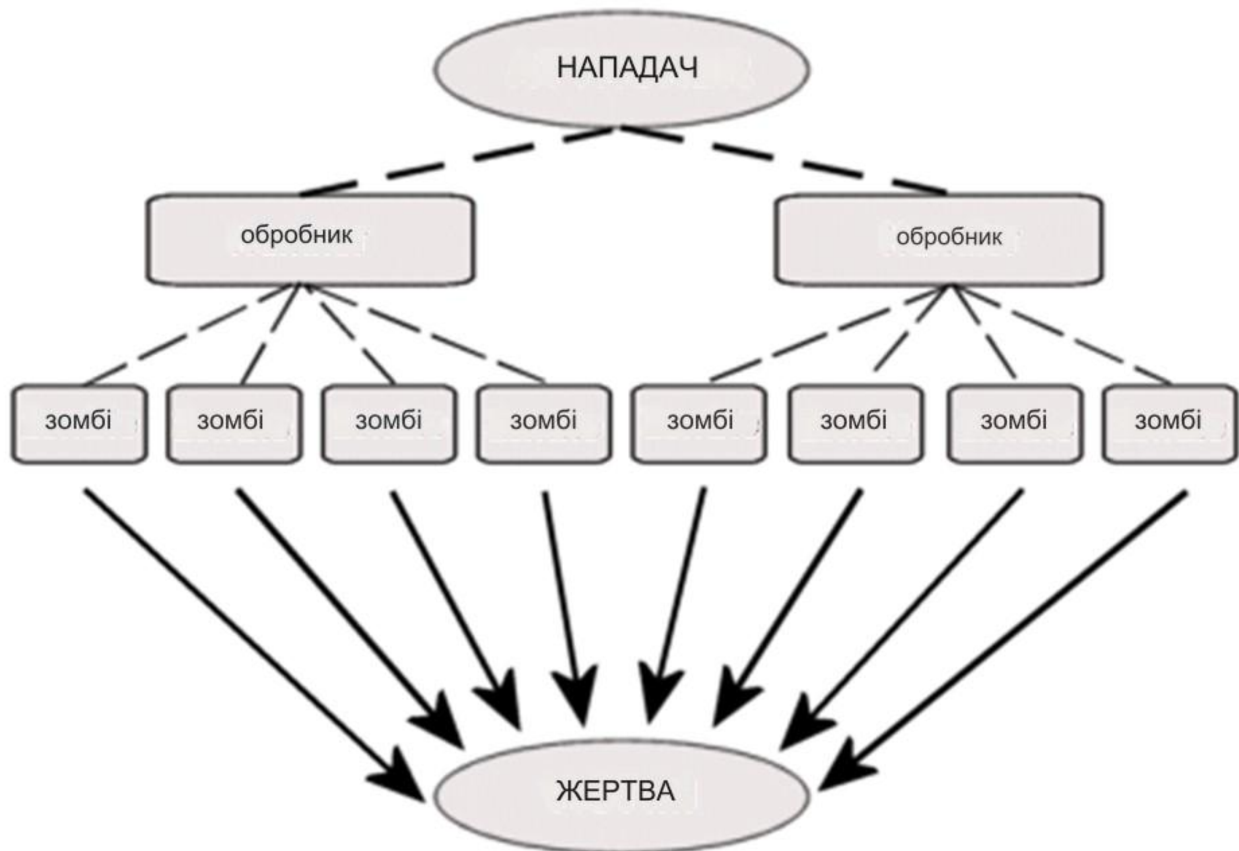


Рис. 1.3. Архітектура DDoS атаки

Можливі цілі атаки можуть включати:

• **Вимагання** Основним мотивом зловмисника може бути фінансовий зиск шляхом погрози будь-яким веб-сайтам бізнесу з можливістю DDoS-атаки на їхній веб-сайт у критичну годину їхнього бізнесу.

• **Шпигунство** Зловмисник може захотіти тримати команду реагування зайнятою обробкою DDoS-атаки, поки зловмисник може виконувати свою справжню роботу над справжньою ціллю. Мотив атаки полягає лише в тому, щоб відвернути увагу сервера шумом DDoS-атаки, поки справжнє крадіжка відбувається з іншого джерела.

• **Протест** Ціллю атаки може бути просто привернути увагу жертви, щоб жертва могла виконати деякі операції, необхідні для конкретної мети. Це спосіб протесту проти якоїсь проблеми.

• **Незручність** Атака може бути виконана для отримання тимчасової переваги шляхом викликання роздратування або незручностей у мережі або в машині. Перевага може змінюватися в залежності від зловмисника до зловмисника. Наприклад, зловмисники можуть атакувати машину противника, щоб виграти в конкретній грі. Роздрібні веб-сайти можуть бути атаковані конкурентами, щоб створити їм погану репутацію.

На сьогоднішній день здійснюється безліч DDoS-атак зловмисниками, які можна класифікувати наступним чином :

1. Атаки на основі обсягу

Найпоширеніший тип DDoS-атаки. Зловмисники намагаються спожити всю пропускну здатність мережі, надсилаючи величезну кількість пакетів на сервер жертви, щоб законним користувачам було дуже важко отримати доступ до сервера через блокування на маршрутизаторах інтерфейсу. Масштаб цих типів атак вимірюється в бітах на секунду. Приклад: UDP Flood, ICMP Flood, інші потоки підроблених пакетів.

2. Атаки протоколу

Атаки протоколу споживають ресурси машини жертви, такі як пам'ять, обчислювальні можливості тощо. Вони можуть створювати довгі черги очікування або інші структури даних на проміжних мережевих пристроях, таких як брандмауери, маршрутизатори або балансувачі навантаження тощо. Масштаб атак протоколу вимірюється в пакетах на секунду. Приклад: SYN Flood, атака фрагментованих пакетів, Ping of Death, Smurf DDoS.

3. Атака на рівні додатків

Головним мотивом зловмисника є збій веб-сервера шляхом надсилання законних повідомлень або запитів на веб-сервер. Масштаб для цієї атаки

вимірюється в запитах на секунду. Приклад: Slowloris, DDoS-атака нульового дня.

## 1.5. SQL-ін’єкції

Уразливості SQL-ін'єкції описуються як одна з найсерйозніших загроз для веб-застосувань. Веб-застосунки, які вразливі до SQL-ін'єкції, можуть дозволити зловмиснику отримати повний доступ до їхніх базових даних.

Оскільки ці бази даних часто містять конфіденційну інформацію про споживачів або користувачів, у результаті порушень безпеки можуть виникати крадіжка особистих даних, втрата конфіденційної інформації та шахрайство.

У деяких випадках зловмисники навіть можуть використовувати уразливість SQL-ін'єкції для захоплення та пошкодження системи, яка розміщує веб-застосунок. Веб-застосунки, які вразливі до атак SQL-ін'єкції (SQLIA), є поширеними - дослідження Gartner Group понад 300 веб-сайтів Інтернету показало, що більшість з них можуть бути вразливими до SQLIA.

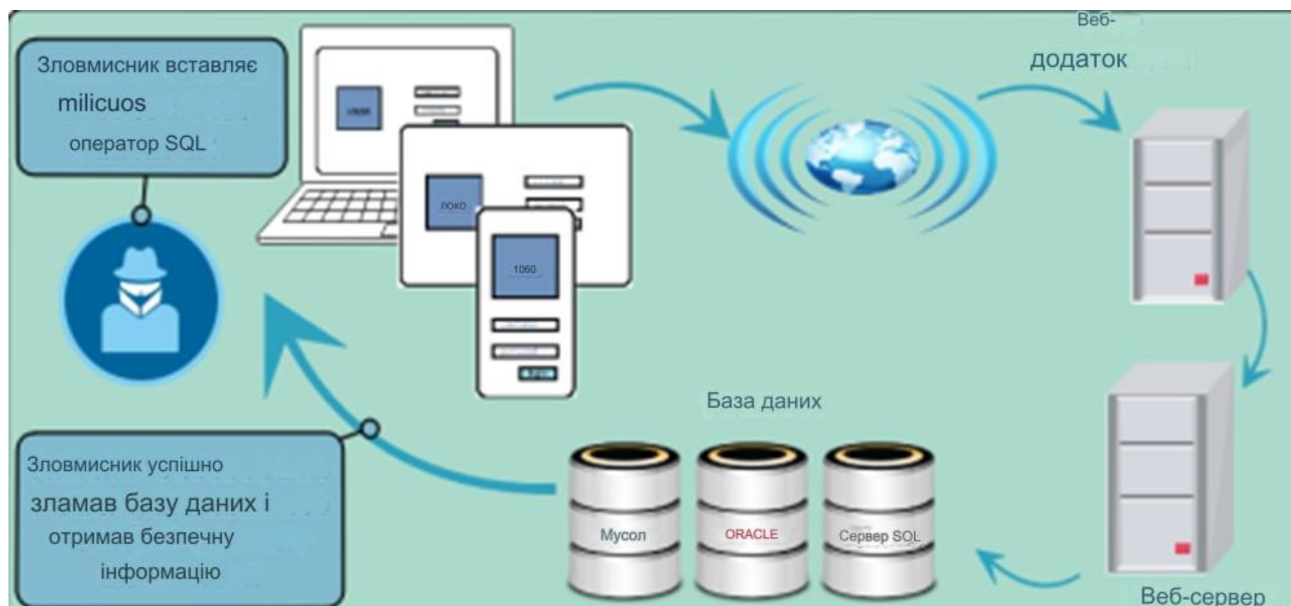


Рис. 1.4. Схема атаки SQL-ін’єкції

Ін’єкції SQL можна вводити у вразливий додаток за допомогою різних механізмів введення. Ось приклади найрозповсюдженіших механізмів:

1.Ін'єкція через введення користувача.

У цьому випадку зловмисники вводять команди SQL, надаючи відповідним чином підготовлений вхід користувача. Веб-застосунок може читати введення користувача різними способами залежно від середовища, в якому розгорнуто додаток. У більшості SQLIA, які спрямовані на веб-застосунки, введення користувача зазвичай надходить із форм, які надсилаються до веб-застосунку через HTTP-запити GET або POST. Веб-застосунки, як правило, можуть отримувати доступ до вхідного тексту користувача, що міститься в цих запитах, як вони б отримували будь-яку іншу змінну в середовищі.

2.Ін'єкція через cookies.

Cookies – це файли, які містять інформацію про стан, згенеровану веб-застосунками та зберігаються на клієнтській машині. Коли клієнт повертається до веб-застосунку, cookies можна використовувати для відновлення інформації про стан клієнта. Оскільки клієнт має контроль над зберіганням cookies, зловмисний клієнт може змінити вміст cookies. Якщо веб-застосунок використовує вміст cookies для створення SQL-запитів, зловмисник може легко подати атаку, вбудувавши її в cookies.

3.Ін'єкція через змінні сервера.

Змінні сервера – це колекція змінних, які містять HTTP, заголовки мережі та змінні середовища. Веб-застосунки використовують ці змінні сервера різними способами, наприклад, для ведення журналу статистики використання та визначення тенденцій перегляду. Якщо ці змінні реєструються в базі даних без дезінфекції, це може призвести до уразливості SQL-ін'єкції. Оскільки зловмисники можуть підробити значення, які розміщуються в заголовках HTTP та мережі, вони можуть використовувати цю уразливість, розміщуючи SQLIA

прямо в заголовках. Коли запит на реєстрацію змінної сервера надсилається до бази даних, тоді активується атака в підробленому заголовку.

4.Ін'єкція другого порядку.

При ін'єкціях другого порядку зловмисники вводять зловмисні вхідні дані в систему або базу даних, щоб опосередковано запустити SQLIA, коли цей вхід використовується пізніше. Мета цього типу атаки значно відрізняється від звичайної (тобто першого порядку) атаки ін'єкції. Ін'єкції другого порядку не намагаються виконати атаку, коли зловмисний вхід спочатку надходить до бази даних. Натомість зловмисники покладаються на знання про те, де вхід буде використовуватися надалі, і створюють свою атаку таким чином, щоб вона відбувалася під час цього використання.

### 1.6. Викрадення сеансу (Session Hijacking)

Викрадення сеансу - це атака, яка в основному використовується для отримання несанкціонованого доступу між підключеннями авторизованого сеансу. Зазвичай це робиться для атаки на веб-сайт соціальної мережі та банківські веб-сайти, щоб отримати доступ до дійсного сеансу та веб-сайту також. Ці атаки є одними з найпоширеніших кіберзагроз у сучасних мережах. Більшість веб-сайтів і мереж уразливі до такого типу атак. Іноді цю атаку викрадення сеансу також називають атакою типу "людина посередині" (MIMA).

Зазвичай розділяють три типи викрадення сеансу:

1. Активне викрадення сеансу

Активне викрадення сеансу - це тип атаки, при якому зловмисник безпосередньо втручається в сеанс між користувачем і сервером. Зловмисник атакує активний сеансі та виключає з нього дійсного користувача, замінюючи його собою за допомогою атаки відмови в обслуговуванні (DOS). Перед здійсненням DOS-атаки зловмисник перехоплює з'єднання та захоплює всі

пакети даних між користувачем і сервером за допомогою інструменту перехоплення пакетів, наприклад Wireshark.

DOS-атака, під час якої зловмисник заповнює цільовий пристрій трафіком, надсилає велику кількість запитів або інформації до цільової мережі та робить її недоступною для сервера, оскільки цільова система не може використовувати послуги, що надсилаються сервером. Після деякого часу цільова машина вимикається або виходить з ладу, щоб не справлятися з потоком трафіку.

Сервер чекає деякий час і повторно надсилає запит до машини користувача для встановлення з'єднання, і в цей момент зловмисник маскується під дійсного користувача та надсилає підтвердження на сервер. Таким чином, зловмисник встановлює з'єднання з сервером замість дійсного користувача.

2. Пасивне викрадення сеансу

Пасивне викрадення сеансу - це тип атаки, при якому зловмисник не втручається в сеанс, але спостерігає за ним.

При пасивному викраденні сеансу зловмисник розміщує себе між дійсним користувачем і сервером і надсилає дійсні пакети користувачеві, маскуючись під сервер, а також отримує пакети від користувача та надсилає на сервер, маскуючись під дійсного користувача. У цьому пасивному викраденні сеансу зловмисник може передавати всі дані через свою систему, і навіть може вносити деякі зміни до пакетів даних, і ні користувач, ні сервер не зможуть виявити цих змін. Таким чином зловмисник може отримати всі необхідні деталі для своїх зловмисних дій.

Але для зловмисника є один недолік. Він/вона може отримувати доступ лише до даних між користувачем і сервером, поки сеанс активний, і якщо користувач виходить із системи або сервер скидає з'єднання з будь-якої причини, зловмисник не зможе отримати доступ до пакетів даних, і сеанс буде остаточно завершено.

3. Гібридне викрадення сеансу

Гібридне викрадення сеансу - це комбінація активного та пасивного викрадення сеансу. У цьому випадку зловмисники використовують обидва типи технік викрадення сеансу для досягнення своєї мети, яка може бути активною та пасивною. Гібридне викрадення сеансу можна розділити на два типи, наведені нижче:

1.1. Атака сліпого підроблення

Атака сліпого підроблення - це вид атаки, в якому зловмисник атакує цільову систему без будь-яких змін у з'єднанні між сервером і машиною жертви. Зловмисник просто захоплює весь пакет із мережі між користувачем і сервером, щоб знайти номер послідовності TCP для отримання автентифікації з сервером і повного контролю над сеансом. Однак у цій атаці є проблема: дуже важко знайти або вгадати номер послідовності TCP через захоплені пакети, оскільки номер послідовності TCP є випадковим числом щоразу, коли він генерує новий і випадковий номер послідовності TCP, що ускладнює пошук правильного номера послідовності. Для цього потрібно багато часу, і для цього зловмиснику потрібно продовжувати захоплювати пакети для аналізу послідовності TCP.

1.2. Атака несліпого підроблення

Атака несліпого підроблення - це тип атаки, в якому зловмисник повинен перебувати в тій самій мережі, а також під тією самою підмережею, де зловмисник може контролювати трафік між жертвою та сервером.

Нападнику легко контролювати трафік з тієї самої мережі, оскільки він може бачити пакети, що передаються через ту саму мережу. Зловмисники продовжують контролювати з'єднання і намагаються вгадати номер послідовності TCP наступних пакетів, щоб отримати автентифікацію через з'єднання за допомогою номера послідовності TCP. Зловмисники знаходять правильний номер послідовності та відновлюють з'єднання на основі правильного номера послідовності з сервером.

Але великою проблемою в цій атаці є те, що сучасні маршрутизатори не дозволяють транслювати пакети в мережі, вони зберігають їх вимкненими для захисту пакетів. Щоб вирішити цю проблему, зловмисник скидає з'єднання, щоб поставити себе між маршрутизаторами для захоплення першого широкомовного пакета.

## 1.7. Дослідження актуальності створення та розвитку рішення для захисту фронтенд додатків

Поширеність всесвітньої павутини робить веб-сайти та їх відвідувачів привабливими цілями для різних видів кіберзлочинності, включаючи порушення даних, фішингові кампанії, програм-вимагачів та шахрайські схеми технічної підтримки. Згідно з останнім звітом Symantec про загрози інтернет-безпеці, щодня відбувається понад 229 000 атак на веб-сайти, а понад 76% веб-сайтів містять невиправлені вразливості. Набори інструментів для веб-атак, такі як RIG та Sundown, роблять ще простішим для кіберзлочинців використання таких вразливостей.

Одними з найкритичніших загроз безпеці сучасних веб-сайтів, які є динамічними, інтерактивними та спільними, є атаки ін'єкцій та міжсайтового скриптування (XSS).. Основною причиною несанкціонованого доступу до даних є SQL-ін'єкція, яка передбачає надсилання шкідливих даних для зміни SQL-запитів, що виконуються базами даних веб-застосунків. У третьому кварталі 2016 року SQL-ін'єкції та атаки XSS становили 55% всіх атак на веб-застосунки. Тому розуміння цих атак та розробка можливих контрзаходів є життєво важливими.

Актуальність розробки бібліотеки, яка об'єднує різні засоби захисту веб-додатків, обумовлена кількома факторами:

**Зростання кількості веб-додатків**

З розвитком технологій все більше і більше компаній створюють веб-

додатки для своїх бізнес-процесів. Це пов'язано з тим, що веб-додатки дозволяють компаніям економити витрати, підвищувати ефективність і розширювати доступ до своїх продуктів і послуг.

Зростання кількості веб-додатків також робить їх більш привабливими цілями для хакерів. Хакери можуть використовувати веб-додатки для крадіжки даних користувачів, зараження комп'ютерів шкідливим програмним забезпеченням або дестабілізації бізнес-процесів.

**Поява нових типів атак**

Хакери постійно розробляють нові методи для атаки веб-додатків. Це ускладнює розробникам веб-додатків забезпечення їх безпеки.

Вразливості фронтенду стають дедалі поширенішими, в першу чергу через зростання кількості підприємств і кінцевих користувачів, які використовують веб-застосунки для обміну конфіденційною та особистою інформацією. Оскільки більшість веб-сайтів і веб-застосунків написані на JavaScript (~98% усіх глобальних веб-сайтів), який також вважається небезпечною мовою програмування, зловмисники просто використовують JavaScript, вставляючи шкідливі сценарії в існуючий код. Проблеми безпеки фронтенду також виникають через велику кількість сторонніх скриптів та бібліотеке, що використовуються для компіляції веб-сайтів.

**Необхідність дотримання нормативних вимог**

Багато компаній зобов'язані дотримуватися нормативних вимог щодо безпеки веб-додатків. Ці вимоги часто передбачають використання різних засобів захисту.

Виконання нормативних вимог може бути складним і витратним. Розробка бібліотеки, яка об'єднує різні засоби захисту веб-додатків, може допомогти компаніям виконати нормативні вимоги.

Ось деякі конкретні переваги розробки такої бібліотеки:

Зручність використання. Бібліотека може полегшити розробникам веб-

додатків забезпечення їх безпеки, забезпечивши їх набором інструментів, які можна використовувати для захисту від різних типів атак.

Ефективність. Бібліотека може допомогти розробникам веб-додатків забезпечити більш ефективний захист, об'єднавши різні засоби захисту в єдиному рішенні.

Зменшення витрат. Бібліотека може допомогти компаніям зменшити витрати на забезпечення безпеки веб-додатків, забезпечивши їх рішенням, яке не вимагає додаткових розробок або впровадження.

В цілому, розробка бібліотеки, яка об'єднує різні засоби захисту веб-додатків, може бути корисною для розробників веб-додатків і компаній, які хочуть підвищити безпеку своїх веб-додатках.

**Висновки**

Було розглянуто та проаналізовано можливі вразливості фронтенд застосунків та основні можливі методи якими можуть скористуватися зловмисники для отримання доступу до системи чи чутливих даних користувачів, розробників чи власників застосункую

# РОЗДІЛ 2
# ІСНУЮЧІ ІНСТРУМЕНТИ ТА МЕТОДИКИ ДЛЯ ЗАПОБІГАННЯ АТАК

Фронтенд додатки є важливою частиною сучасних веб-додатків. Вони відповідають за інтерфейс користувача, який дозволяє користувачам взаємодіяти з додатком. Однак фронтенд додатки також є вразливими до атак, які можуть призвести до крадіжки даних, виконання несанкціонованих дій або інших проблем.

Метою цього розділу є проаналізувати роботи, присвячені безпеці фронтенд додатків, а також бібліотеки та інструменти для аналізу та захисту фронтенд додатків. У цьому розділі необхідно виділити недоліки існуючих аналогів, які можна усунути в розроблюваній бібліотеці.

Багато робіт було присвячено вивченню безпеки фронтенд додатків. Ці роботи розглядають різні аспекти безпеки фронтенд додатків, такі як типи атак, методи захисту та інструменти для аудиту безпеки.

Деякі з найважливіших робіт у цій галузі включають:

"The Web Application Security Consortium (WASC) Threat Classification" - це документ, який класифікує вразливості веб-додатків.

"OWASP Top 10 Web Application Security Risks" - це документ, який визначає 10 найпоширеніших вразливостей веб-додатків.

"Snyk Security Report 2023" - це звіт, який аналізує стан безпеки фронтенд додатків.

У цьому розділі також розглянемо кілька контрзаходів для попереденьо описаних вразливостей та їх недоліки.

Таблиця 1.1.

Перелік властивостей та аналіз їх загрози до складності попередження

| **Вразливість** | **Спосіб атаки** | **Рівень загрози** | **Складність усунення** |
|---|---|---|---|
| XSS | Введення шкідливого коду в форму або інший інтерфейс введення даних | Висока | Середня |
| SQL-ін'єкція | Введення шкідливого запиту в форму або інший інтерфейс введення даних | Висока | Середня |
| CSRF (Cross-site request forgery) | Використання облікових даних користувача для виконання несанкціонованих дій | Висока | Середня |
| Фішинг | Підробка веб-сайту або електронного листа з метою отримання конфіденційної інформації | Висока | Середня |
| DoS (Denial of Service) | Завдання перевантаження веб-сервера або мережі для блокування доступу користувачів | Середня | Низька |
| Brute force | Пробний доступ до облікових даних користувача | Середня | Середня |
| Session hijacking | Перехоплення сеансу користувача для отримання його доступу | Висока | Середня |
| Data theft | Крадіжка даних користувачів, таких як паролі, номери кредитних карток та інша конфіденційна інформація | Висока | Середня |

Для атаки або захисту веб-додатку необхідно мати велику кількість знань

та навичок, таких як знання протоколів мережі, особливостей розробки програмного забезпечення та основні вразливості цих програм. Але для більш успішного використання цих навичок та знань необхідна інформація про додаток який захищається або атакується. Яка архітектура додатку, які технології цей додаток використовує, хто користувачі цього додатку та як вони взаємодіють з ним. Щоб отримати цю інформацію використовується веб-розвідка. Зрештою, веб-розвідка полягає у зборі даних та побудові моделі, яка поєднує технічні та функціональні деталі веб-додатку таким чином, щоб повністю зрозуміти його призначення та використання. Без одного чи іншого хакер не зможе належним чином націлити свої атаки.

## 2.1. Веб-розвідка

Веб-розвідка - це дослідницький етап збору даних, який зазвичай відбувається перед зломом веб-додатку. Її зазвичай проводять хакери, пен-тестери або мисливці за помилками, але також це може бути ефективним засобом для інженерів безпеки, щоб знайти слабко захищені механізми у веб-додатку та виправити їх до того, як їх виявив зловмисник. Самі по собі навички розвідки не мають значної цінності, але стають дедалі ціннішими, коли їх об'єднують з знаннями про наступальний хакінг та досвідом оборонної інженерії безпеки.

Веб-розвідка може включати в себе наступні завдання:

Збір інформації про веб-додаток, наприклад, його адресу, технології, які він використовує, і його вразливості.

Виявлення потенційних точок входу в систему, які можна використовувати для атаки на веб-додаток.

Збір інформації про користувачів веб-додатку, наприклад, їх імена користувачів, паролі та електронні адреси.

Розуміння того, як працює веб-додаток, може бути отримане різними

способами. Іноді достатньо просто пройтися по ньому і звернути увагу на запити до мережі, щоб детально ознайомитися з його внутрішньою роботою. Однак, важливо зазначити, що не всі веб-додатки матимуть інтерфейс користувача, який дозволяє візуально досліджувати додаток і відстежувати його функціональність. Більшість відкритих додатків (часто B2C-додатків, як-от соціальні мережі) матимуть громадський інтерфейс. Проте, не варто вважати, що навіть у цьому випадку доступ до всього інтерфейсу гарантований. Натомість, поки не проведено глибше дослідження, слід припускати, що доступний лише підмножина інтерфейсу.

Під час звичайного аналізу інтерфейсів веб-додатків мало що можна дізнатись про кінцеві точки API. Але завдяки розвідці веб додатків, можливо знайти ці API та створити карту, яка буде детально описувати дозволи адміністраторів та модераторів, які можна порівняти з дозволами простого користувача.

Навички розвідки також можуть використовуватися для збору інформації про додатки, до яких вони буквально не мають доступу. Це може бути внутрішня мережа школи або доступний по мережі файловий сервер компанії. Їм не потрібен інтерфейс користувача, щоб зрозуміти, як працює додаток, якщо вони володіють навичками, щоб реверсувати структуру його API та даних, які вони приймають. Іноді під час розвідки вони навіть знаходять сервери або API, які взагалі не захищені.

Багато компаній покладаються на кілька серверів, як внутрішніх, так і зовнішніх. Просте забуття одного рядка конфігурації мережі або брандмауера може призвести до того, що HTTP-сервер буде відкритий для публічної мережі, а не обмежений внутрішньою.

## 2.2. Дизайн

Безпечний дизайн допоможе встановити безпечні значення за замовчуванням, мінімізувати поверхню атаки та безпечно перейти до чітко визначених та зрозумілих значень за замовчуванням. Він також буде враховувати та дотримуватися різних принципів, таких як:

- Найменший привілей та розподіл обов'язків
- Багаторівневий захист
- Нульова довіра
- Безпека у відкритому доступі

Безпечний життєвий цикл розробки (SDLC) допомагає гарантувати, що всі рішення щодо безпеки, прийняті для продукту, що розробляється, є явним вибором і призводять до правильного рівня безпеки для дизайну продукту. Можна використовувати різні безпечні життєві цикли розробки, і вони зазвичай включають моделювання загроз у процесі проектування.

**Принцип найменшого привілею**

Наприклад, якщо звичайному користувачеві потрібен доступ до певного файлу для виконання своєї роботи, йому слід надати доступ лише до цього файлу, а не до всієї системи файлів. Це робить менш імовірним те, що зловмисник, який отримає контроль над обліковим записом користувача, зможе завдати шкоди всій системі.

**Принцип нульової довіри**

Принцип нульової довіри передбачає, що жодній особі чи системі не слід довіряти автоматично, навіть якщо вона перебуває всередині мережі організації. Це означає, що кожен раз, коли користувач або пристрій намагається отримати доступ до ресурсу, слід перевіряти його автентичність та авторизацію.

**Принцип безпеки у відкритому доступі**

Принцип безпеки у відкритому доступі передбачає, що всі аспекти

безпеки продукту повинні бути відкриті для громадського контролю. Це допомагає виявити та виправити вразливості безпеки швидше та ефективніше.

SDLC є важливою частиною будь-якої організації, яка розробляє програмне забезпечення. Це допомагає гарантувати, що безпека враховується на всіх етапах процесу розробки, від початкового проектування до випуску та техобслуговування продукту.

Моделювання загроз – це структурований підхід до виявлення та пріоритезації потенційних загроз для системи. Процес моделювання загроз включає визначення цінності, яку потенційні засоби пом'якшення мали б для зменшення або нейтралізації цих загроз. Оцінка потенційних загроз на етапі проектування вашого проекту може значно заощадити ресурси, які можуть знадобитися для переробки проекту для включення засобів пом'якшення ризиків на більш пізньому етапі проекту.

Оскільки фронт-енд додатки зберігають облікові дані та пропонують різний користувацький досвід для гостей та зареєстрованих користувачів, необхідно знати, що маємо як систему автентифікації, так і систему авторизації. Це означає, що необхідно дозволити користувачам входити в систему, а також мати можливість розрізняти різні рівні користувачів, визначаючи, які дії їм дозволено виконувати.

Крім того, оскільки зберігаються облікові дані та підтримується процес входу, ми знаємо, що облікові дані будуть надсилатися через мережу. Ці облікові дані також потрібно зберігати в базі даних, інакше процес автентифікації буде порушено. Це означає, що розробникам потрібно враховувати такі ризики:

• Як ми обробляємо дані під час передачі?

• Як ми обробляємо зберігання облікових даних?

• Як ми обробляємо різні рівні авторизації користувачів?

**Ризики щодо даних в транзиті:**

Перехоплення: Зловмисник може перехопити дані, що передаються між користувачем і сервером. Щоб запобігти цьому, потрібно використовувати шифрування, наприклад, HTTPS.

Підміна: Зловмисник може змінити дані під час передачі. Щоб запобігти цьому, потрібно використовувати підпис цифрового сертифіката.

**Ризики щодо зберігання облікових даних:**

Витік: Зловмисник може отримати доступ до бази даних та викрасти облікові дані. Щоб запобігти цьому, потрібно використовувати надійні паролі, хешувати та солити їх, а також регулярно оновлювати програмне забезпечення.

Несанкціоноване використання: Зловмисник може використати вкрадені облікові дані для входу в систему як інший користувач. Щоб запобігти цьому, потрібно використовувати двофакторну автентифікацію.

**Ризики щодо авторизації користувачів:**

Підвищення привілеїв: Зловмисник може отримати доступ до функцій, які йому не дозволено використовувати. Щоб запобігти цьому, потрібно чітко визначити дозволи для кожного рівня користувачів і ретельно їх контролювати.

Горизонтальне переміщення: Зловмисник може отримати доступ до даних користувачів, до яких він не має доступу. Щоб запобігти цьому, потрібно використовувати контроль доступу на основі ролей (RBAC) та інші механізми розмежування доступу.

### 2.3. SQL-ін’єкції

Основною причиною вразливостей ін'єкції SQL є недостатня перевірка вхідних даних. Тому простим рішенням для усунення цих вразливостей є застосування відповідних захисних практик кодування. Тут ми наведено огляд деяких найкращих практик, запропонованих у літературі для запобігання вразливостям ін'єкції SQL, а також інструментів, які можна використовувати для виявлення уразливостей SQL-ін'єкцій, таких як SQL Inject-Me, SQLninja і

Havij.

**SQL Inject-Me** - це інструмент, який можна використовувати для пошуку уразливостей SQL-ін'єкцій в веб-додатках. Він працює, відправляючи рядки екранування бази даних через поля форм в веб-додатку, а потім шукає помилкові повідомлення, які виникають як результат.

**SQLninja** - це інструмент, призначений для виявлення уразливостей SQL-ін'єкцій в веб-додатках, які використовують Microsoft SQL Server як свою базу даних. Це дуже потужний інструмент, який може працювати навіть в дуже ворожому середовищі.

**Havij** - це автоматизований інструмент, який допомагає тестерам на проникнення виявляти уразливості SQL-ін'єкцій в веб-додатках. Це один з найпотужніших інструментів для виявлення уразливостей SQL-ін'єкцій завдяки своїм унікальним методам.

**Перевірка типу вхідних даних**: Ін'єкцію SQL можна виконати, ввівши команди в строковий або цифровий параметр. Навіть проста перевірка таких вхідних даних може запобігти численним атакам. Наприклад, у випадку цифрових вхідних даних розробник може просто відхилити будь-які вхідні дані, що містять символи, відмінні від цифр. Багато розробників випадково пропускають цей вид перевірки, оскільки вхідні дані користувача майже завжди представлені у вигляді рядка, незалежно від їхнього вмісту або призначення.

**Кодування вхідних даних**: Ін'єкція в строковий параметр часто виконується за допомогою метасимволів, які обманюють парсер SQL, щоб він інтерпретував вхідні дані користувача як токени SQL. Хоча можливо заборонити будь-яке використання цих метасимволів, це обмежило б можливість не зловмисного користувача вказувати законні вхідні дані, що містять такі символи. Кращим рішенням є використання функцій, які кодують рядок таким чином, щоб усі метасимволи були спеціально закодовані та інтерпретовані базою даних як звичайні символи.

**Позитивне співставлення шаблонів**: Розробники повинні створити процедури перевірки вхідних даних, які виявляють правильні вхідні дані на відміну від неправильних. Цей підхід зазвичай називається позитивною перевіркою, на відміну від негативної перевірки, яка шукає вхідні дані на наявність заборонених шаблонів або токенів SQL. Оскільки розробники можуть не передбачити всі типи атак, які можуть бути здійснені проти їхнього додатку, але повинні мати можливість визначити всі форми законних вхідних даних, позитивна перевірка є безпечнішим способом перевірки вхідних даних.

**Виявлення всіх джерел вхідних даних**: Розробники повинні перевіряти всі вхідні дані свого додатку. Існує багато можливих джерел вхідних даних для додатку. Якщо ці джерела вхідних даних використовуються для створення запиту, вони можуть стати способом для зловмисника ввести SQLIA. Простіше кажучи, потрібно перевіряти всі джерела вхідних даних.

## 2.4. XSS

Хоча зараз вони зустрічаються рідше, ніж у минулому, вразливості XSS все ще широко поширені в сучасному веб-середовищі. Через постійно зростаючу кількість взаємодій з користувачами та збереження даних у веб-застосунках, можливості виникнення вразливостей XSS у додатках є більшими, ніж будь-коли раніше.

На відміну від інших поширених архетипів вразливостей, XSS можна експлуатувати з кількох ракурсів - деякі з них зберігаються між сеансами (збережені), а інші (відображені) не зберігаються. Крім того, оскільки вразливості XSS залежать від пошуку місць виконання сценаріїв у клієнті, існує ймовірність, що помилки у складних специфікаціях браузера також можуть призвести до непередбаченого виконання сценаріїв (засновані на DOM XSS). Збережені XSS можна виявити шляхом аналізу сховища бази даних, що робить їх легко виявними. Але відображені таDOM вразливості XSS часто важко

виявити та локалізувати - це означає, що ці вразливості можуть існувати на великій кількості веб-застосунків, але ще не були виявлені.

Загрози в XSS можуть бути ін'єкцією клієнтського коду, крадіжкою файлів cookie, тунелюванням XSS, DoS, поширенням зловмисного програмного забезпечення тощо.

Існує багато інструментів, які використовуються для тестування динамічних веб-сторінок, таких як XSS Server, XSSer та OWASP Xenotix. Однак ці інструменти можуть бути використані зловмисниками для пошуку вразливих веб-сторінок.

XSS Server: Це здобич на стороні сервера, яка використовується для експлуатації вразливостей XSS. Призначення цього інструменту полягає в тому, щоб збирати конфіденційні дані від користувачів, коли вони виконують код XSS або коли вони отримують доступ до введеної веб-сторінки, яка має вбудований код XSS. Конфіденційні дані цієї жертви можуть бути файлами cookie, IP-адресою жертви, вмістом веб-сторінки, іменем користувача та паролями тощо.

Cross Site Scripter “XSSer”: це автоматичний фреймворк для експлуатації та виявлення вразливостей CSS у веб-додатках. Цей інструмент є безкоштовним інструментом для тестування на проникнення, який містить багато методів, які допомагають обійти певні фільтри. Він також має кілька різних варіантів введення коду. XSSer вимагає Python і працює на багатьох платформах.

OWASP Xenotix XSS: це розширений фреймворк, який використовується для виявлення та експлуатації вразливостей XSS. "Цей інструмент підтримує як ручний режим, так і автоматизовані режими тестування на основі розподілу часу. Він включає в себе кодувальник XSS, реєстратор натискань клавіш на стороні жертви та завантажувач виконуваних файлів з автоматичним запуском".

Ось деякі з його функцій:

- Payloads: включаючи HTML5-сумісні пейлоади для введення XSS, цей

інструмент має більше 380 пейлоадів.

- XSS Key logger - це один із способів отримати інформацію користувачів, яка вводиться на веб-сторінках. Дія реєстрації натискань клавіш на клавіатурі називається кейлоггером, який можна використовувати для шпигунства за кимось і отримання доступу.

- Кодувальник XSS: Ця функція дозволяє кодувати в різних формах, таких як URL-кодування, HTML, Base64 і HEX-кодування, щоб обійти веб-застосункові брандмауери та інші фільтри. Крім того, існує багато доступних веб-сайтів, які дозволяють користувачам генерувати код XSS, щоб перевірити свої фільтри перевірки вхідних даних на предмет XSS, наприклад XSS String Encoder.

- Тестування XSS: Xenotix має режим автоматичного тестування, який тестує кожен пейлоад на основі періоду часу, який користувачі повинні вказати відповідно до необхідного часу для завантаження веб-сторінки, який залежить від їх пропускної здатності.

Shahriar and Zulkernine розробили інструмент S2XS2 для автоматичного виявлення атак XSS на стороні сервера за допомогою концепції "генерації політик" для перевірки вхідних даних користувача та "введення меж" для інкапсуляції динамічно створеного вмісту на основі тегів HTML та введення коду JavaScript. Цей інструмент написаний на Java та використовує парсер Jericho HTML та парсер Rhino для аналізу вихідного коду веб-сторінок та визначення особливостей коду JavaScript. Автори оцінили цей підхід на чотирьох реальних програмах JSP.

### 2.5. Викрадення сеансу (Session Hijacking)

Існує багато способів захисту від викрадення сеансів. Але ці способи залежать від того, наскільки серйозно користувачі ставляться до своєї безпеки. Кажуть, що зловмисник використовує неусвідомленість користувача щодо

безпеки, щоб викрасти його важливу інформацію, а іноді зловмисник обдурює користувача і викрадає інформацію. Відповідно до методик CEHv8 Ethical Hacking and counter measure's techniques, можна розділити контрзаходи проти викрадення сеансів на два рівні з моделі OSI (Open System Interconnection), як показано нижче:

• Мережевий рівень

• Рівень додатків

**Мережевий рівень**

**1.Захищений сокетний шар**

Завжди використовуйте захищений сокетний шар, який забезпечує кінцеве шифрування даних. Таким чином, будь-які дані, які проходять через цей мережевий захищений сокетний шар, зберігаються в зашифрованому вигляді, що дуже ускладнює для зловмисника перегляд точних даних, які передаються через мережу. Навіть якщо зловмисник отримає дані, йому дуже важко знайти реальні дані в пакетах. Канали SSL використовують відкритий ключ 28 бітів і симетричний ключ 256 бітів, що робить метод шифрування більш складним, сильним і більш захищеним.

**Використання захищенної оболонки (SSH)**

Захищений оболонка також відомий як Secure Socket shell, це один із мережевих протоколів, який надає користувачам безпечний спосіб доступу до віддаленої системи або віддаленого комп'ютера. Ця захищена оболонка забезпечує надійний спосіб автентифікації та більш надійний спосіб шифрування між двома системами в незахищеній мережі, що також допомагає користувачеві захиститися від атак типу "викрадення сеансу".

**Використання HTTPS**

Дуже важливо використовувати з'єднання HTTPS (Hyper Text Transfer Protocol Secure) завжди, коли ми входимо на будь-який веб-сайт або будь-який веб-сервер, або під час онлайн-банкінгу, онлайн-покупок або електронної

комерції. Користувач повинен бути уважним під час усієї онлайн-роботи, оскільки URL завжди повинен бути у формі HTTPS, оскільки це робить з'єднання безпечним і показує, що це безпечне посилання для онлайн-роботи. Якщо посилання не у формі https, воно незахищене, і дані будуть передаватися у звичайному тексті.

**Рівень додатку**

Рівень додатку є кінцевою частиною рівня безпеки, яка займається викраденням ідентифікатора сеансу, існує кілька контрзаходів, які наведені нижче.

**Складний і сильний ідентифікатор сеансу**

Ідентифікатор сеансу надає унікальну ідентичність кожному сеансу, а також користувачеві для відстеження прогресу користувача та стану автентифікації користувачів у веб-застосунку. Кожен додаток надає користувачам ідентифікатор сеансу, який також відомий як ідентифікатор сеансу або токен, який призначається сеансу під час його створення та використовується для обміну з користувачами та сервером додатків для відстеження дій користувачів. Цей ідентифікатор сеансу буде дійсним, поки сеанс дійсний, після того, як сеанс завершиться.

Існує кілька важливих кроків, які необхідно виконати, щоб зробити ідентифікатор сеансу сильним і більш складним.

**Випадковий ідентифікатор сеансу**

Завжди використовуйте генерування випадкового ідентифікатора сеансу, що дуже ускладнює зловмиснику вгадати ідентифікатор сеансу.

Довгий ідентифікатор сеансу: Якщо ідентифікатор сеансу буде достатньо довгим, то він забезпечує хорошу безпеку для захисту від атак методом грубої сили.

**Ідентифікатор сеансу, згенерований сервером**

Завжди використовуйте згенерований сервером ідентифікатор сеансу, що

робить ідентифікатор сеансу більш складним і сильним, оскільки сервери використовують алгоритм для генерування ідентифікатора сеансу, і зловмиснику також дуже важко його зламати.

**Висновки**

В Розділі 2 розглянули основні методики запобігання різноманітним атакам на веб додатки. Серед них є як методики які слід використовувати під час розробки додатку, аналіз можливих вразливостей вже розробленого та введеного в експлуатацію додатку, а також певні інструменти які можуть аналізувати специфічні вразливості або попереджати їм.

# РОЗДІЛ 3
# АНАЛІЗ СТРУКТУРИ ТА РОЗРОБКА ПРОТОТИПУ ПРОГРАМНОГО ЗАСОБУ

Фронтенд додатки є однією з найважливіших складових сучасних веб-додатків. Вони відповідають за взаємодію користувача з додатком, а також за відображення інформації та даних. Безпека фронтенд додатків є важливою темою, оскільки вони можуть бути використані для атак на серверну частину додатка.

У цьому розділі дипломної роботи буде проведено аналіз структури та розроблено прототип програмного засобу для аналізу та вдосконалення безпеки фронтенд додатків. Програмний засіб буде реалізований у вигляді бібліотеки, яка може бути використана для аналізу будь-якого фронтенд додатка, а також засобів які покращать безпеку

Аналіз структури програмного засобу

Програмний засіб буде мати наступну структуру:

**Бібліотека для фронтенду** - бібліотека, яка буде використовуватися для аналізу коду фронтенд додатка.

**Бібліотека для серверної частини** - бібліотека, яка буде використовуватися для аналізу коду серверної частини додатка.

Бібліотека для фронтенду буде реалізована у вигляді JavaScript-модуля. Вона буде використовуватися для аналізу коду фронтенд додатка, виявлення потенційних проблем безпеки та надання рекомендацій щодо їх усунення.

Бібліотека для серверної частини буде реалізована у вигляді Node.js-модуля. Вона буде використовуватися для аналізу коду серверної частини додатка, виявлення потенційних проблем безпеки та надання рекомендацій щодо їх усунення.

Обидві бібліотеки будуть взаємодіяти між собою за допомогою API. API

буде забезпечувати передачу інформації про потенційні проблеми безпеки між бібліотеками.

### 3.1. Аналіз вразливостей та розробка засобів запобігання атакам

Така структура програмного засобу дозволить забезпечити більш ефективний аналіз безпеки веб-додатків. Бібліотека для фронтенду буде аналізувати код фронтенд додатка, а бібліотека для серверної частини буде аналізувати код серверної частини додатка. Це дозволить виявити більш широкий спектр потенційних проблем безпеки.

Крім того, така структура дозволить забезпечити більш високий рівень абстракції. Розробники веб-додатків не будуть повинні знати, як працює кожна бібліотека. Вони будуть взаємодіяти з програмним засобом через API. Це дозволить зробити програмний засіб більш простим у використанні.

Сторонні бібліотеки є важливим компонентом багатьох додатків. Вони надають функції та можливості, які розробники не хочуть або не можуть створювати самостійно. Однак сторонні бібліотеки також можуть бути джерелом вразливостей.

Аналіз підключених до додатку бібліотек на вразливості є важливим кроком у забезпеченні безпеки додатка. Це допоможе вам виявити будь-які вразливості у ваших бібліотеках і вжити заходів щодо їх усунення.

Один з найважливіших моментів, про які слід пам'ятати при розгляді сторонніх залежностей, полягає в тому, що багато з них мають свої власні залежності. Іноді їх називають сторонніми залежностями четвертого рівня. Ручна оцінка окремої сторонньої залежності, яка не має залежностей четвертого рівня, є можливою. Ручна оцінка коду сторонніх залежностей є ідеальною у багатьох випадках. Ручні огляди коду не масштабуються особливо добре, і в багатьох випадках було б неможливо всебічно переглянути сторонню залежність, яка спирається на залежності четвертого рівня. Особливо якщо ці

сторонні залежності четвертого рівня містять власні залежності, і так далі.

Якщо дерево залежностей можна завантажити в пам'ять і змоделювати за допомогою деревоподібної структури даних, перебір дерева залежностей стає досить простим і дивовижно ефективним. При додаванні до основної програми будь-яка залежність і всі її піддерева залежностей повинні бути оцінені. Оцінка цих дерев повинна виконуватися в автоматичному режимі. Найпростіший спосіб почати виявляти вразливості в дереві залежностей - порівняти дерево залежностей вашої програми з відомою базою даних CVE. Ці бази даних містять списки та відтворення вразливостей, виявлених у відомих пакетах OSS і сторонніх пакетах, які часто інтегруються в основні програми.

```
❯ snyk-resolve -f request --dev
npm@2.14.17
├─┬ node-gyp@3.2.1 bundled
│ └── request@2.67.0 bundled
├─┬ npm-registry-client@7.0.9 bundled
│ └── request@2.67.0 bundled
├─┬ npm-registry-couchapp@2.6.12
│ └─┬ couchapp@0.11.0
│   ├─┬ nano@6.2.0
│   │ ├─┬ follow@0.12.1
│   │ │ └── request@2.55.0
│   │ └── request@2.67.0 bundled
│   └── request@2.67.0 bundled
├── request@2.67.0 bundled
└─┬ tap@2.3.4
  ├─┬ codecov.io@0.1.6
  │ └── request@2.42.0
  └─┬ coveralls@2.11.6
    └── request@2.67.0 bundled
```

Рис. 3.1.  Дерево залежностей додатку

Для аналізу підключених бібліотек можемо використовувати декілька API та методів які дозволять нам отримати інформацію про ліцензії підключених бібліотек, а також їх вразливості. Для цього опишемо загальну функцію getLibraryInfo яка б оцінювала безпеку підключених бібліотек npm та CDN. Для

цього можна використовувати наступні методи:

Перевірка за списком відомих уразливостей. Можна використовувати один з таких сервісів, як Snyk: https://snyk.io/, NPM Audit: https://docs.npmjs.com/cli/audit або Dependabot: https://dependabot.com/. Ці сервіси дозволяють перевірити бібліотеку на наявність відомих уразливостей.

Використання статичних аналізаторів коду. Статичні аналізатори коду дозволяють виявити потенційні уразливості в коді бібліотеки. Для цього можна використовувати такі інструменти, як eslint: https://eslint.org/, tslint: https://palantir.github.io/tslint/ або sonarcloud: https://sonarcloud.io/.

Використання статистичної оцінки безпеки. Статистична оцінка безпеки дозволяє оцінити ймовірність наявності уразливостей у бібліотеці. Для цього можна використовувати такі інструменти, як NPM Vulnerability Calculator: https://nvd.nist.gov/vuln-metrics/cvss або Common Vulnerability Scoring System (CVSS): [https://www.first.org/cvss/](https://www.first.org/cvss/).

Для аналізу підключених бібліотек розробимо функцію getLibraryInfo яка буде зв`язувати декілька функції в собі для створення автоматизованого аналізу бібліотек та їх залежностей

Функція getStaticAnalysisResults(libraryPath):

Ця функція призначена для виконання статичного аналізу коду за допомогою інструменту ESLint для конкретного шляху libraryPath.

```
async function getStaticAnalysisResults(libraryPath) {
  const response = await exec(`eslint --config .eslintrc.js ${libraryPath}`);
  const staticAnalysisResults = JSON.parse(response.stdout);
  return staticAnalysisResults;
}
```

Функція npmAudit(libraryPath):

Ця функція виконує аудит проекту за допомогою команди npm audit --path ${libraryPath}, де ${libraryPath} - це шлях до проекту чи бібліотеки.

```
async function npmAudit(libraryPath) {
  const response = await exec(`npm audit --path ${libraryPath}`);
  const auditResults = JSON.parse(response.stdout);
  return auditResults;
}
```

Функція getKnownVulnerabilities(libraryName, libraryVersion):

Ця функція призначена для отримання списку відомих уразливостей для певної бібліотеки Node.js та її версії з використанням сервісу Snyk.

```
async function getKnownVulnerabilities(libraryName, libraryVersion) {
  const response = await
fetch(`https://snyk.io/vuln/npm/${libraryName}@${libraryVersion}`);
  const vulnerabilities = await response.json();
  return vulnerabilities;
}
```

Зібравши цю інформацію ми можемо створити функцію яка буде за результатами попередніх функцій буде робити загальний аналіз безпеки та вразливостей для кожної з бібліотек та створювати JSON об'єкт з усією інформацією:

1. Спочатку функція отримує результати статичного аналізу коду, аудиту npm та список відомих уразливостей.

2. Потім функція рахує кількість помилок статичного аналізу, проблем, виявлених аудитом npm, та відомих уразливостей.

3. Потім функція оцінює небезпеку кожної помилки статичного аналізу, проблеми, виявленої аудитом npm та відомої вразливості.

4. Нарешті, функція обчислює загальну оцінку безпеки бібліотеки.

```
▼ (2) [{…}, {…}]                                        App.js:7
  ▼ 0:
      knownVulnerabilities: 0
      libraryId: "jquery"
      libraryVersion: "3.6.0"
      npmAuditIssues: 0
      securityScore: 0
      staticAnalysisErrors: 0
    ▶ [[Prototype]]: Object
  ▼ 1:
      knownVulnerabilities: 2
      libraryId: "bootstrap"
      libraryVersion: "5.0.0-beta1"
      npmAuditIssues: 1
      securityScore: 16
      staticAnalysisErrors: 1
    ▶ [[Prototype]]: Object
    length: 2
  ▶ [[Prototype]]: Array(0)
```

Рис. 3.2. Приклад загальної оцінки безпеки бібліотеки

Отримавши оцінку можемо написати спеціальну функцію яка буде створювати HTML сторінку з викладенням оцінки безпеки всіх бібліотек, яка буде автоматично зберігатись як файл для перегляду розробником.

---

## Security report

| Library ID | Library version | Static analysis errors | NPM audit issues | Known vulnerabilities | Security score |
|---|---|---|---|---|---|
| jquery | 3.6.0 | 0 | 0 | 0 | 0 |
| bootstrap | 5.0.0-beta1 | 1 | 1 | 2 | 16 |

Рис. 3.3. Звіт з безпеки бібліотек

До захисту від SQL атак краще підходити зі сторони серверу, так як в більшості веб-додатків операції SQL проходять якраз зі сторони серверу, тому клієнт в цій частині ми облишаємо.

Застосунок, що реалізований у Node.js, використовує такі бібліотеки SQL:

SQL Server – через адаптер NodeMSSQL (npm)

MySQL – через адаптер mysql (npm)

У зв’язку з цим необхідно продумати структурування пошуку в кодовій

базі таким чином, щоб можна було знаходити SQL-запити з обох реалізацій SQL.

На щастя, модульна система імпорту, яка входить до складу Node.js, у поєднанні з областю видимості JavaScript, робить це досить простим. Якщо бібліотека SQL імпортується для кожного модуля окремо, пошук запитів стає таким же простим, як пошук імпорту.

Щоб захистити свій веб-додаток від SQL-ін'єкцій, можна використовувати підготовлені SQL-запити. Підготовлені запити дозволяють чітко визначити дані, які вводяться в SQL-запит, що ускладнює для зловмисників впровадження шкідливого коду.

Щоб використовувати підготовлені SQL-запити в Node.js, можна використовувати функцію prepare() модуля mysql. Наприклад, розглянемо наступний код:

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'my_database',
});

connection.connect();

const query = `SELECT * FROM users WHERE username = ?`;
const preparedQuery = connection.prepare(query);

preparedQuery.execute(['user1']);

const results = preparedQuery.fetchAll();
```

Цей код підключається до бази даних MySQL і готує запит SELECT * FROM users WHERE username = ?. Потім він виконує запит, передавши значення user1 як параметр.

У цьому випадку зловмисник не може впровадити шкідливий код у запит,

оскільки він не може контролювати значення, яке вводиться в параметр ?.

Щоб зробити підготовлені SQL-запити ще більш безпечними, можливо використовувати функцію bind() для зв'язування параметрів з конкретними значеннями.

```
const query = `SELECT * FROM users WHERE username = ? AND password = ?`;
const preparedQuery = connection.prepare(query);

preparedQuery.bind('user1', 'password1');

preparedQuery.execute();

const results = preparedQuery.fetchAll();
```

Цей код так само, як і попередній, підключається до бази даних MySQL і готує запит SELECT * FROM users WHERE username = ? AND password = ?. Однак у цьому випадку він використовує функцію bind() для зв'язування параметрів ? з конкретними значеннями user1 і password1. Це робить ваші підготовлені запити ще більш безпечними, оскільки зловмисник не може контролювати значення, які вводяться в параметри.

```
function executeQuery(connection, query, params) {
  connection.connect();
  const preparedQuery = connection.prepare(query);
  preparedQuery.bind(params);
  preparedQuery.execute();
  const rests = preparedQuery.fetchAll();
  return results;
}
```

Така функція буде безпечно готувати та зв’язувати параметри з конкретними значеннями і виконуватиме запит в заданій базі даних.

У першій частині були розглянуті атаки XSS, які використовують можливість браузера виконувати JavaScript-код на пристроях користувачів. Уразливості XSS є поширеними та можуть завдати значних збитків, оскільки вразливості виконання сценаріїв мають широкий спектр потенційних збитків. Однак, хоча XSS досить поширений вид атак, його досить легко пом'якшити або повністю запобігти за допомогою передових практик безпечного кодування та

спеціальних методів пом'якшення XSS.

Найбільшим правилом попереджання XSS атак є попередження використання будь якого вводу користувача в DOM в будь-якому іншому вигляді ніж у типі строка.

```
function sanitizeInput(input) {
  input = String(input);

  input = input.replace(/<[^>]+>/gi, "");

  return input;
}
```

Ця функція переводить будь-який ввід користувача в тип строка, після чого санітизує ввід на предме HTML-тегів, що попередить ввід шкідливого коду, та його інтеграції в DOM дерево в майбутньому. Цю функцію варто використовувати при кожному використанні вводу користувача, наприклад при використанні форми з елементами input.

Використання нативних елементів розробки теж можуть бути менш захищеними, розглянемо наприклад DOMParser.

```
const parser = new DOMParser();
const html = parser.parseFromString('<script>alert("hi");</script>`);
```

API parseFromString завантажує вміст рядка в вузли DOM, які відображають структуру вхідного рядка. Це можна використовувати для заповнення сторінки структурованим DOM із сервера. Наприклад, якщо ви хочете перетворити складний рядок DOM у належно організовані вузли DOM, ви можете використовувати цей API.

Однак ручне створення кожного вузла за допомогою document.createElement() та їх організація за допомогою document.appendChild(child) є безпечнішим варіантом. Це дозволяє вам контролювати структуру та імена тегів DOM, а корисне навантаження лише контролює вміст.

Blob також несе в собі той самий ризик:

```
const blob = new Blob([script], { type: 'text/javascript' });
const url = URL.createObjectURL(blob);
const script = document.createElement('script');
script.src = url;
document.body.appendChild(script);
```

Крім того, blob-об'єкти можуть зберігати дані в багатьох форматах; base64 як blob є просто контейнером для довільних даних. Внаслідок цього найкраще, якщо можливо, не використовувати blob-об'єкти в своєму коді, особливо якщо будь-який етап процесу створення та використання blob-об'єкта включає дані користувача.

Атаки DoS за допомогою регулярних виразів, ймовірно, є найпростішою формою DoS-атак для захисту, але потребують попередніх знань про структуру таких атак (як показано в другій частині цієї книги). За допомогою належного процесу перегляду коду можна запобігти потраплянню "зловісних" або "шкідливих" регулярних виразів у вашу кодову базу.

Потрібно шукати регулярні вирази, які виконують суттєве зворотне відстеження (backtracking) щодо повторюваної групи. Ці регулярні вирази зазвичай мають форму, схожу на (a[ab]*)+, де знак + означає виконання greedy-matching (знайти всі потенційні співпадіння перед поверненням), а знак * означає узгодження підвиразу якомога більше разів.

Проблема з цим регулярним виразом полягає в тому, що він може призвести до експоненціального часу виконання, якщо зловмисник подасть вхідний рядок, який змушує regex виконувати зворотне відстеження багато разів. Наприклад, якщо вхідний рядок містить багато символів "a", то regex буде змушений перевірити всі можливі підрядки, щоб знайти найдовше співпадіння. Це може призвести до того, що програма перестане відповідати на запити або навіть аварійно завершиться. Для цього можемо додати до функції що санітизує ввід користувача перевірку видалення необхідних регулярних виразів:

```
function sanitizeInput(input) {
  input = String(input);
```

```
  input = input.replace(/<[^>]+>/gi, "");
  input = input.replace(/\.{100,}/g, "");
  input = input.replace(/,{100,}/g, "");
  return input;
}
```

Атаки DDoS зазвичай не націлені на логічні помилки, а натомість намагаються перевантажити цільовий об'єкт за допомогою величезного обсягу трафіку, що виглядає легітимним. Таким чином, реальні користувачі не можуть отримати доступ до сервісу, або їх досвід роботи із програмою значно сповільнюється. Хоча DDoS-атаки неможливо повністю запобігти, їх можна пом'якшити кількома способами. Найпростіший спосіб захистити ваш веб-застосунок від DDoS-атаки - це інвестувати в службу керування пропускною здатністю. Ці служби розробляються багатьма постачальниками на ринку, але в кінцевому підсумку вони аналізують кожен пакет даних, який проходить через їх сервери. Служби запускають добре зарекомендовані сканування пакетів, щоб визначити, чи надходять вони в шкідливому порядку. Якщо пакет визначено як шкідливий, він не буде перенаправлено на ваш веб-сервер.

Для зменшення ризику DDoS-атаки у вашій архітектурі веб-застосунку можна також застосувати додаткові заходи. Один із поширених методів відомий як "чорна діра" (blackholing). При цьому ви налаштовуєте кілька серверів на додаток до основного сервера додатку. Підозрілий (або повторюваний) трафік надсилається на сервер-чорну діру, який зовні схожий на ваш сервер додатку, але не виконує жодних операцій. Легітимний трафік направляється на ваш легітимний веб-сервер додатку у звичайному режимі.

```
const checkRequest = (req, res) => {
  const parsedUrl = url.parse(req.url);
  if (parsedUrl.query.count > 100) {

    const blackholeReq = http.request({
      host: blackholeHost,
      port: 8080,
      method: 'POST',
```

```
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      url: parsedUrl.pathname,
    }),
  });

  blackholeReq.on('response', (res) => {
    res.on('data', (data) => {
      console.log(data);
    });
  });

  blackholeReq.end();

  res.statusCode = 429;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify({
    message: 'Too many requests',
  }));
  return;
}

const contentLength = req.headers['content-length'];
if (contentLength && parseInt(contentLength, 10) > 1000000) {
  res.statusCode = 429;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify({
    message: 'Too much data',
  }));
  return;
}


const clientIp = req.headers['x-forwarded-for'] || req.connection.remoteAddress;
if (blockedIps.has(clientIp)) {

  res.statusCode = 429;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify({
    message: 'IP address blocked',
  }));
  return;
}

const next = require('./next');
```

```
  next(req, res);
};
```

Є багато способів дізнатись чи є запит частиною DDoS атаки:

- Кількісний аналіз: Цей метод полягає в тому, щоб перевірити, чи перевищує запит певний пороговий показник. Наприклад, можна перевірити, чи кількість запитів від конкретного IP-адреси перевищує певне значення за певний період часу.
- Якісний аналіз: Цей метод полягає в тому, щоб перевірити запит на наявність певних шаблонів, які є характерними для DDoS-атак. Необхідно перевірити, чи запит містить велику кількість даних або чи використовується він для надсилання спеціальних символів або даних.
- Аналіз поведінки: Цей метод полягає в тому, щоб перевірити, чи є запит частиною більшої атаки. Необхідно перевірити чи запит надсилається з декількох різних IP-адрес або чи використовується він для атаки на певний ресурс.

Також можна перевіряти IP адресу серверу: чи надсилається однаковий запит з декількох IP, чи були заблоковані IP адреси за DDoS-атаки, а також чи має запит велику кількість даних, чи використовуються спеціальні символи, або дані які не є валідними об’єктами JSON або XML.

Також необхідним є захист CSRF атак. Оскільки джерело багатьох CSRF-запитів відрізняється від веб-застосунку який ці запити приймає, можна зменшити ризик CSRF-атак, перевіряючи походження запиту. У HTTP є два заголовки, на які треба звертати увагу, коли перевіряємо походження запиту: referer та origin. Ці заголовки є важливими, оскільки їх не можна змінити програмно за допомогою JavaScript у всіх основних браузерах. Таким чином, правильно реалізований заголовок referer або origin браузера має низьку ймовірність підробки.

```
function validateHeadersAgainstCSRF(headers, allowedOrigins, allowedReferers) {
```

```
  const origin = headers.origin;
  const referer = headers.referer;
  if (!origin || !referer) {
    return false;
  }
  if (!allowedOrigins.includes(origin)) {
    return false;
  }
  if (!allowedReferers.includes(referer)) {
    return false;
  }
  return true;
}

if (!session.isAuthenticated) { return res.sendStatus(401); }
if (!validateHeadersAgainstCSRF(req.headers)) { return res.sendStatus(401); }
```

Більшість сучасних стеків веб-серверів дозволяють створювати посередницьке програмне забезпечення або скрипти, які виконуються для кожного запиту, перш ніж маршрутом буде виконано будь-яка логіка. Таке посередницьке програмне забезпечення можна розробити для реалізації цих методів на всіх ваших маршрутах на стороні сервера.

```
function validateCSRFToken(token, user) {
  const textToken = crypto.decrypt(token);
  const parts = textToken.split(':');

  const userId = parts[0];
  const date = parts[1];
  const nonce = parts[2];

  let validUser = false;
  let validDate = false;
  let validNonce = false;

  if (userId === user.id) {
    validUser = true;
  }

  if (dateTime.lessThan(1, 'week', date)) {
    validDate = true;
  }

  if (crypto.validateNonce(userId, date, nonce)) {
```

```
    validNonce = true;
  }

  return validUser && validDate && validNonce;
}

module.exports = CSRFShield;
```

Такі функції-посередники можна викликати для всіх запитів, що надходять на сервер, або індивідуально визначити для запуску на конкретних запитах. Вони перевіряють, чи заголовки походження та/або referer правильні, а потім переконується, що токен CSRF є дійсним. Якщо будь-яка з цих перевірок не пройдена, повертається помилка ще до того, як буде викликано будь-яку іншу логіку. В іншому випадку, воно переходить до наступного посередника і дозволяє програмі продовжувати виконання без змін.

## Висновки

В ході дослідження безпеки веб-застосунків та аналізу можливих заходів безпеки була створена архітектура застосунку який може аналізувати розроблений застосунок на вразливості до атак, а також допомагати забезпечувати безпеку під час розробки, застосовуючи методи попереджання атак які були досліджені в попередніх розділах.

# ВИСНОВКИ

Дипломний проект містить грунтовну інформацію щодо можливих небезпек фронт-енд застосунків, якими методами користуються зловмисники та які існують розповсюджені типи атак. Також описані сторонні технічні засоби та методики як зможуть знизити ризик атак на веб-застосунок.

У дипломному проекті описане проектування та розробка бібліотеки для вдосконалення та підвищення безпеки фронт-енд застосунків, який розробники можуть використовувати під час розробки для аналізу да загального забезпечення безпеки.

Даний програмний продукт сприятиме підвищенню безпеки фронт-енд застосунків від атак зловмисників які можуть повести за собою втрату особистої чи чутливої інформації, матеріальних збитків чи репутаційних збитків. Також дипломний проект дозволить збільшити обізнаність розробників про необхідність забезпечення безпеки своїх програмних застосунків.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. D. Meier, "Web application security engineering," in IEEE Security & Privacy, vol. 4, no. 4, pp. 16-24, July-Aug. 2006, doi: 10.1109/MSP.2006.109.
2. Gupta, B. B., & Badve, O. P. (2016). Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment. Neural Computing and Applications, 28(12), 3655–3682. doi:10.1007/s00521-016-2317-5
3. Gupta, S., & Gupta, B. B. (2015). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. International Journal of System Assurance Engineering and Management, 8(S1), 512–530. doi:10.1007/s13198-015-0376-0
4. Barth, A., Jackson, C., & Mitchell, J. C. (2008). Robust defenses for cross-site request forgery. Proceedings of the 15th ACM Conference on Computer and Communications Security - CCS ’08. doi:10.1145/1455770.1455782
5. Halfond, William & Viegas, Jeremy & Orso, Alessandro. (2006). A Classification of SQL Injection Attacks and Countermeasures.
6. Hacking Web Apps: Detecting and Preventing Web Application Security Problems. Mike Shema
7. Ginkel, Neline & Groef, Willem & Massacci, Fabio & Piessens, Frank. (2019). A Server-Side JavaScript Security Architecture for Secure Integration of Third-Party Libraries. Security and Communication Networks. 2019. 1-21. 10.1155/2019/9629034.

**Додаток А**

```
async function generateSecurityReport(securityResults, outputPath) {
    const html = `<!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Security report</title>
  </head>
  <body>
    <h1>Security report</h1>
    <table>
      <thead>
        <tr>
          <th>Library ID</th>
          <th>Library version</th>
          <th>Static analysis errors</th>
          <th>NPM audit issues</th>
          <th>Known vulnerabilities</th>
          <th>Security score</th>
        </tr>
      </thead>
      <tbody>`;

    for (const securityResult of securityResults) {
      html += `<tr>
        <td>${securityResult.libraryId}</td>
        <td>${securityResult.libraryVersion}</td>
        <td>${securityResult.staticAnalysisErrors}</td>
        <td>${securityResult.npmAuditIssues}</td>
        <td>${securityResult.knownVulnerabilities}</td>
        <td>${securityResult.securityScore}</td>
      </tr>`;
    }

    html += `</tbody>
    </table>
  </body>
  </html>`;

    await fs.writeFile(outputPath, html);
}

const librariesInfo = await getLibrariesInfo("package.json");
```

```
if (librariesInfo) {
  for (const libraryInfo of librariesInfo) {
    console.log(libraryInfo);
  }
}

async function getLibrariesInfo(packagePath) {
  const libraries = await fetch(packagePath).then(response =>
response.json()).then(packageInfo => packageInfo.dependencies);
  const libraryInfos = await Promise.all(libraries.map(library =>
getLibraryInfo(library)));
  return libraryInfos;
}

async function getLibraryInfo(libraryPath) {

  if (libraryPath.startsWith("https://www.npmjs.com/package/")) {
    const response = await fetch(libraryPath);
    const libraryInfo = await response.json();
    const vulnerabilities = await getKnownVulnerabilities(libraryInfo.name,
libraryInfo.version);
    libraryInfo.knownVulnerabilities = vulnerabilities;
    const auditResults = await npmAudit(libraryPath);
    libraryInfo.auditResults = auditResults;    const staticAnalysisResults = await
getStaticAnalysisResults(libraryPath);
    libraryInfo.staticAnalysisResults = staticAnalysisResults;

    let securityAnalysis = analyzeLibrarySecurity(libraryInfo);
    return securityAnalysis;
  } else if (libraryPath.startsWith("https://cdn.")) {    const libraryName =
libraryPath.split("/").pop();
    const response = await
fetch(`https://cdn.${libraryPath.split("/").slice(1).join("/")}/api/package/${libra
ryName}`);
    const libraryInfo = await response.json();
    const vulnerabilities = await getKnownVulnerabilities(libraryName,
libraryInfo.version);
    libraryInfo.knownVulnerabilities = vulnerabilities;
    const auditResults = await npmAudit(libraryPath);
    libraryInfo.auditResults = auditResults;
    const staticAnalysisResults = await getStaticAnalysisResults(libraryPath);
    libraryInfo.staticAnalysisResults = staticAnalysisResults;

    let securityAnalysis = analyzeLibrarySecurity(libraryInfo);
    return securityAnalysis;
  } else {
```

```
    return null;
  }
}

async function getStaticAnalysisResults(libraryPath) {
  const response = await exec(`eslint --config .eslintrc.js ${libraryPath}`);
  const staticAnalysisResults = JSON.parse(response.stdout);
  return staticAnalysisResults;
}

async function npmAudit(libraryPath) {
  const response = await exec(`npm audit --path ${libraryPath}`);
  const auditResults = JSON.parse(response.stdout);
  return auditResults;
}

async function getKnownVulnerabilities(libraryName, libraryVersion) {
  const response = await
fetch(`https://snyk.io/vuln/npm/${libraryName}@${libraryVersion}`);
  const vulnerabilities = await response.json();
  return vulnerabilities;
}

async function analyzeLibrarySecurity(libraryInfo) {
  const staticAnalysisResults = libraryInfo.staticAnalysisResults;

  const auditResults = libraryInfo.auditResults;

  const vulnerabilities = libraryInfo.knownVulnerabilities;

  const securityResults = {
    libraryId: libraryInfo.name,
    libraryVersion: libraryInfo.version,
    staticAnalysisErrors: staticAnalysisResults.length,
    npmAuditIssues: auditResults.length,
    knownVulnerabilities: vulnerabilities.length,
    securityScore: 0
  };

  for (const error of staticAnalysisResults) {
    if (error.severity === "critical") {
      securityResults.securityScore += 10;
    } else if (error.severity === "high") {
      securityResults.securityScore += 5;
    }
  }
```

```
  for (const issue of auditResults) {
    if (issue.severity === "critical") {
      securityResults.securityScore += 10;
    } else if (issue.severity === "high") {
      securityResults.securityScore += 5;
    }
  }

  for (const vulnerability of vulnerabilities) {
    if (vulnerability.severity === "critical") {
      securityResults.securityScore += 10;
    } else if (vulnerability.severity === "high") {
      securityResults.securityScore += 5;
    }
  }
  return securityResults;
}

const fs = require('fs-extra');
const securityResults = [
  {
    libraryId: 'jquery',
    libraryVersion: '3.6.0',
    staticAnalysisErrors: 0,
    npmAuditIssues: 0,
    knownVulnerabilities: 0,
    securityScore: 0,
  },
  {
    libraryId: 'bootstrap',
    libraryVersion: '5.0.0-beta1',
    staticAnalysisErrors: 1,
    npmAuditIssues: 1,
    knownVulnerabilities: 2,
    securityScore: 16,
  },
];

const outputPath = 'security-report.html';

const html = `<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Security report</title>
```

```
</head>
<body>
  <h1>Security report</h1>
  <table>
    <thead>
      <tr>
        <th>Library ID</th>
        <th>Library version</th>
        <th>Static analysis errors</th>
        <th>NPM audit issues</th>
        <th>Known vulnerabilities</th>
        <th>Security score</th>
      </tr>
    </thead>
    <tbody>`;

for (const securityResult of securityResults) {
  html += `<tr>
      <td>${securityResult.libraryId}</td>
      <td>${securityResult.libraryVersion}</td>
      <td>${securityResult.staticAnalysisErrors}</td>
      <td>${securityResult.npmAuditIssues}</td>
      <td>${securityResult.knownVulnerabilities}</td>
      <td class="security-score">${securityResult.securityScore}</td>
    </tr>`;
}

html += `</tbody>
</table>
<body>
</html>`;

await fs.writeFile(outputPath, html);

const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'my_database',
});

connection.connect();

const query = `SELECT * FROM users WHERE username = ? AND password = ?`;
```

```
const preparedQuery = connection.prepare(query);

preparedQuery.bind('user1', 'password1');

preparedQuery.execute();

const results = preparedQuery.fetchAll();



console.log(results);

function executeQuery(connection, query, params) {
  connection.connect();
  const preparedQuery = connection.prepare(query);
  preparedQuery.bind(params);
  preparedQuery.execute();
  const rests = preparedQuery.fetchAll();
  return results;
}

function sanitizeInput(input) {
  // Переводимо ввід у тип String.
  input = String(input);

  // Санітизуємо ввід на предмет HTML-тегів.
  input = input.replace(/<[^>]+>/gi, "");

  // Перевіряємо ввід на наявність атак DoS.
  input = input.replace(/\.{100,}/g, "");
  input = input.replace(/,{100,}/g, "");

  return input;
}

const checkRequest = (req, res) => {
  const parsedUrl = url.parse(req.url);
  if (parsedUrl.query.count > 100) {

    const blackholeReq = http.request({
      host: blackholeHost,
      port: 8080,
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
```

```
        url: parsedUrl.pathname,
      }),
    });

    blackholeReq.on('response', (res) => {
      res.on('data', (data) => {
        console.log(data);
      });
    });

    blackholeReq.end();

    res.statusCode = 429;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({
      message: 'Too many requests',
    }));
    return;
  }

  const contentLength = req.headers['content-length'];
  if (contentLength && parseInt(contentLength, 10) > 1000000) {
    res.statusCode = 429;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({
      message: 'Too much data',
    }));
    return;
  }


  const clientIp = req.headers['x-forwarded-for'] || req.connection.remoteAddress;
  if (blockedIps.has(clientIp)) {

    res.statusCode = 429;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify({
      message: 'IP address blocked',
    }));
    return;
  }

  const next = require('./next');
  next(req, res);
};

const server = http.createServer((req, res) => {
```

```
  checkRequest(req, res);
});

server.listen(8080);

function validateHeadersAgainstCSRF(headers, allowedOrigins, allowedReferers) {
  const origin = headers.origin;
  const referer = headers.referer;
  if (!origin || !referer) {
    return false;
  }
  if (!allowedOrigins.includes(origin)) {
    return false;
  }
  if (!allowedReferers.includes(referer)) {
    return false;
  }
  return true;
}

if (!session.isAuthenticated) { return res.sendStatus(401); }
if (!validateHeadersAgainstCSRF(req.headers)) { return res.sendStatus(401); }

function CSRFShield(req, res, next) {
  if (!validateHeaders(req.headers, req.method)) {
    logger.log(req);
    return res.sendStatus(401);
  }

  if (!validateCSRFToken(req.csrf, session.currentUser)) {
    logger.log(req);
    return res.sendStatus(401);
  }


  next();
}

function validateHeaders(headers, method) {
  const origin = headers.origin;
  const referer = headers.referer;
  let isValid = false;

  if (method === 'POST') {
    isValid = validLocations.includes(referer) && validLocations.includes(origin);
  } else {
```

```
    isValid = validLocations.includes(referer);
  }

  return isValid;
}

function validateCSRFToken(token, user) {
  const textToken = crypto.decrypt(token);
  const parts = textToken.split(':');

  const userId = parts[0];
  const date = parts[1];
  const nonce = parts[2];

  let validUser = false;
  let validDate = false;
  let validNonce = false;

  if (userId === user.id) {
    validUser = true;
  }

  if (dateTime.lessThan(1, 'week', date)) {
    validDate = true;
  }

  if (crypto.validateNonce(userId, date, nonce)) {
    validNonce = true;
  }

  return validUser && validDate && validNonce;
}

module.exports = CSRFShield;
```