ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика запобігання вразливостей front-end компонентів web-застосунків»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Даніель БІЛОДІД
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Даніель БІЛОДІД

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Білодіду Даніелю Вадимовичу

1. Тема кваліфікаційної роботи: «Методика запобігання вразливостей front-end компонентів»

керівник кваліфікаційної роботи Тимур ДОВЖЕНКО, к.т.н., доцент кафедри ІПЗ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань безпеки web-застосунків та захисту від кіберзагроз, вимоги до сучасних web-застосунків щодо захисту даних користувачів, параметри впровадження методів захисту, таких як CSRF - токени, Content-Security-Police(CSP), заголовки безпеки.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз сучасних загроз безпеці web-застосунків.
2. Дослідження методів захисту web-азстосунків.
3. Розробка та впровадження методу S.A.F.E.
4. Валідація ефективності впроваджених методів.

5. Перелік графічного матеріалу: *презентація*

1. Мета, об’єкта та предмет дослідження.
2. Огляд існуючих методик захисту фронтенд компонентів.
3. Розробка методики "S.A.F.E.
4. Порівняння методики S.A.F.E. з існуючими методами.
5. Технології які використовувалися.
6. Математична модель оцінки ефективності методики безпеки S.A.F.E.
7. Діаграма послідовності реалізації методики S.A.F.E.
8. Гістограма результатів впровадження методики S.A.F.E.
11. Тестування методики S.A.F.E в OWASP ZAP.
12. Результати впровадження методики S.A.F.E.
13. Висновки.
14. Публікації та апробація роботи.

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.2024	
2	Вивчення матеріалів для аналізу розвитку технологій багатокамерного трекінгу об’єктів	24.10-31.10.2024	
3	Дослідження методів глибокого навчання	01.11-07.11.2024	
4	Аналіз особливостей впливу методів глибокого навчання на багатокамерний трекінг об’єктів	03.11-09.11.2024	
5	Дослідження технологій глибокого навчання	10.11-14.11.2024	
6	Застосування глибокого навчання в багатокамерному трекінгу об’єктів	15.11-21.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	22.11-26.11.2024	
8	Розробка демонстраційних матеріалів	27.11.-29.11.2024	

Здобувач вищої освіти _____ Даніель БІЛОДІД

(підпис)

Керівник

кваліфікаційної роботи _____ Тимур ДОВЖЕНКО

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 105 стор., 2 табл., 44 рис., 40 джерел.

Мета роботи – розробка методики запобігання вразливостям у front-end компонентах web-застосунків для підвищення їх безпеки.

Об’єкт дослідження – процес забезпечення безпеки front-end компонентів у web-застосунках.

Предмет дослідження – методи і технології запобігання вразливостям у front-end компонентах web-застосунків.

У роботі використано сучасні методи аналізу вразливостей, включаючи статичний та динамічний аналіз коду, а також практики безпечного програмування. Основний акцент зроблено на розробці підходу до інтеграції механізмів безпеки, таких як Content Security Policy (CSP) і CSRF-захист.

Проведено аналіз сучасних загроз інформаційній безпеці, що виникають у front-end розробці web-застосунків. Розглянуто найбільш поширені види атак, такі як міжсайтовий скриптинг (XSS), атаки типу CSRF, а також ін’єкції HTML і JavaScript.

Розроблено методику інтеграції механізмів захисту, яка включає налаштування CSP, впровадження CSRF-токенів і перевірки даних. Вивчено підходи до автоматизації процесу перевірки на вразливості за допомогою сучасних інструментів аналізу коду.

Реалізовано демонстраційний web-застосунок, який впроваджує розроблені методи захисту. Застосунок побудовано з використанням JavaScript, React, Firebase і підтримує перевірку безпеки під час виконання запитів та обробки даних.

Проведено тестування методики за допомогою OWASP ZAP і інших інструментів для аналізу безпеки web-застосунків. Результати тестування підтвердили ефективність інтегрованих механізмів захисту: було успішно виявлено та усунено вразливості, пов’язані з XSS, CSRF та іншими поширеними загрозами.

Результати дослідження демонструють високу ефективність запропонованого підходу до захисту web-застосунків. Методика може бути інтегрована у будь-який етап розробки для забезпечення високого рівня безпеки front-end компонентів.

КЛЮЧОВІ СЛОВА: FRONT-END, WEB-ЗАСТОСУНКИ, БЕЗПЕКА, CONTENT SECURITY POLICY, CSRF-ЗАХИСТ, XSS, JAVASCRIPT, АНАЛІЗ ВРАЗЛИВОСТЕЙ, OWASP ZAP.

ABSTRACT

The text part of the qualification work for obtaining the master's degree: 105 pages, 2 table, 44 figures, 40 sources.

The purpose of the work – development of a methodology to prevent vulnerabilities in front-end components of web applications to enhance their security.

The object of research – the process of ensuring the security of front-end components in web applications.

The subject of research – methods and technologies for preventing vulnerabilities in front-end components of web applications.

The work employs modern methods of vulnerability analysis, including static and dynamic code analysis, as well as secure coding practices. The primary focus is on developing an approach to integrate security mechanisms such as Content Security Policy (CSP) and CSRF protection.

An analysis of current information security threats arising in the front-end development of web applications was conducted. The most common types of attacks, such as Cross-Site Scripting (XSS), CSRF attacks, and HTML/JavaScript injections, were examined.

A methodology for integrating protection mechanisms was developed, which includes CSP configuration, implementation of CSRF tokens, and data validation. Approaches to automating the vulnerability testing process using modern code analysis tools were studied.

A demonstration web application implementing the developed protection methods was created. The application was built using JavaScript, React, and Firebase, and supports security checks during data requests and processing.

Testing of the methodology was carried out using OWASP ZAP and other tools for web application security analysis. The testing results confirmed the effectiveness of the integrated security mechanisms: vulnerabilities related to XSS, CSRF, and other common threats were successfully detected and mitigated.

The results of the research demonstrate the high efficiency of the proposed approach to securing web applications. The methodology can be integrated at any stage of development to ensure a high level of front-end component security.

KEYWORDS: FRONT-END, WEB APPLICATIONS, SECURITY, CONTENT SECURITY POLICY, CSRF PROTECTION, XSS, JAVASCRIPT, VULNERABILITY ANALYSIS, OWASP ZAP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CSRF – Міжсайтове підроблення запитів (Cross-Site Request Forgery).

XSS – Міжсайтове скриптування (Cross-Site Scripting).

CSP – Політика безпеки контенту (Content Security Policy).

HTTPS – Протокол захищеної передачі гіпертексту (HyperText Transfer Protocol Secure).

JWT – JSON Web Token.

S.A.F.E – Метод захисту web-застосунків (Security Algorithm for Front-End).

API – Інтерфейс програмування додатків (Application Programming Interface).

JSON – Обмін даними у форматі JavaScript (JavaScript Object Notation).

ВСТУП

Сучасні web-технології активно впроваджуються у створення web-додатків, які використовують різноманітні сервіси для обробки чутливих даних користувачів. Однак такі інновації супроводжуються новими викликами у сфері безпеки, особливо для фронтенд-частини web-застосунків. Уразливості на цьому рівні можуть призвести до серйозних наслідків, як-от витік персональних даних, виконання шкідливого коду чи маніпуляції з користувацькими запитами.

Об’єкт дослідження – методи та інструменти для забезпечення безпеки фронтенд web-застосунків.

Предмет дослідження – методика безпеки фронтенд web-застосунків, що включає захист від XSS, CSRF, Clickjacking та інші техніки.

Мета роботи – розробка та впровадження методики S.A.F.E. для підвищення безпеки фронтенд web-застосунків, яка включає чотири етапи: Secure Input Validation, Anti-CSRF Measures, Frame Protection, Enhanced Content Security Policy.

Наукова новизна полягає у розробці методики, яка поєднує кілька важливих аспектів безпеки для забезпечення захисту web-застосунків на фронтенді. Впровадження S.A.F.E. методики допомагає значно знизити ризики, пов’язані з найбільш поширеними атаками на web-застосунки, такими як XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) і Clickjacking, шляхом застосування CSRF токенів, перевірки введених даних, посиленої політики безпеки контенту (CSP) і захисту від клікаджекінгу.

Аналіз ринку існуючих рішень у галузі безпеки web-застосунків показав, що багато сучасних систем безпеки недостатньо враховують специфіку фронтенд-розробки та вимагають інтеграції кількох окремих інструментів безпеки. У роботі було проведено аналіз основних вразливостей фронтенд web-застосунків та запропоновано інтегровану методику, що дозволяє ефективно знижувати ймовірність таких атак.

Web-застосунок, до якого була застосована методика S.A.F.E., демонструє значне покращення рівня безпеки за результатами тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Безпека web-застосунків

Web-застосунки сьогодні є основою для багатьох сервісів, які обробляють персональні дані, банківські операції, електронну комерцію, інтерактивні платформи навчання та соціальні мережі. З поширенням цих застосунків зростає кількість загроз, спрямованих на компрометацію їх безпеки.

Front-end компоненти web-застосунків, як правило, відповідають за взаємодію з користувачами, що робить їх критично важливими для загального рівня безпеки. Проте саме ці компоненти стають найвразливішими через недоліки валідації даних, слабкі механізми авторизації або недостатньо налаштовані політики доступу. На рисунку 1.1 зображено основні типи загроз для front-end компонентів web-застосунків, які включають такі атаки, як XSS, CSRF, Clickjacking та інші поширені вразливості.

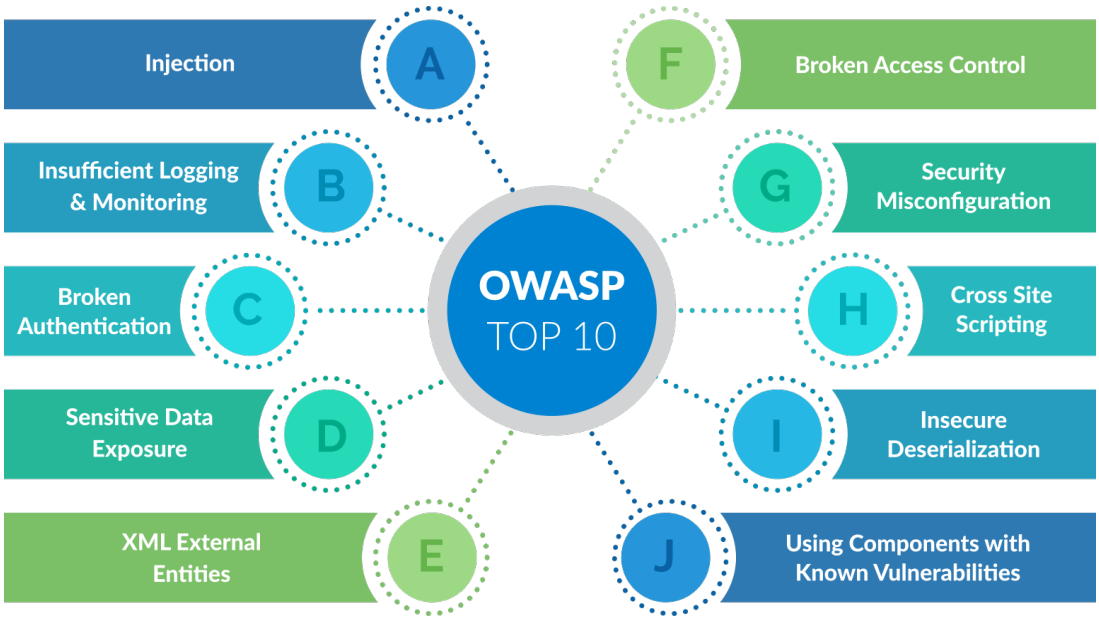


Рис. 1.1 Типові загрози для front-end компонентів web-застосунків

Забезпечення безпеки front-end компонентів web-застосунків включає кілька ключових аспектів, які потребують ретельної уваги на всіх етапах розробки. Ось

більш детальний опис кожного з них:

1. Детектування вразливостей

Один з основних кроків у забезпеченні безпеки — це виявлення потенційних вразливостей у коді. Для цього проводиться аналіз як статичного, так і динамічного коду. Статичний аналіз дозволяє знаходити проблеми ще на етапі написання коду, наприклад, неперевірені введення, які можуть бути використані для виконання атак типу XSS або SQL-ін’єкцій. Динамічний аналіз дає змогу перевіряти поведінку застосунку в реальному часі, виявляючи уразливості, які проявляються тільки під час виконання.

Для виявлення таких уразливостей часто використовуються інструменти як OWASP ZAP (Zed Attack Proxy) або Burp Suite, які дозволяють автоматизувати процес пошуку відомих вразливостей, таких як XSS, CSRF та інші.

Захисні механізми мають бути інтегровані на всіх етапах розробки. Одним із важливих аспектів є Content Security Policy (CSP), що дозволяє обмежити джерела скриптів і ресурсів, з яких web-застосунок може завантажувати дані. Це значно зменшує ризики атак типу XSS, де зловмисник може впровадити небезпечні скрипти через вразливості на сторінці.

Ще одним важливим елементом є CSRF-токени. Вони допомагають захистити застосунок від атак типу Cross-Site Request Forgery (CSRF), де зловмисник може змусити користувача виконати небажану дію на web-застосунку, використовуючи його авторизацію. Інтеграція токенів забезпечує, щоб всі критичні запити, такі як зміна пароля або виконання фінансових операцій, були захищені від таких атак.

Не менш важливим є валідація даних, що забезпечує правильне введення інформації користувачем. Це може включати перевірку формату введених даних (наприклад, email-адреси чи паролів) та їх узгодження з визначеними правилами, що допомагає уникнути помилок і зловживань на етапі збору та обробки даних.

Автоматизація тестування є ключовим аспектом для швидкого та ефективного виявлення вразливостей на всіх етапах життєвого циклу web-застосунку. Для цього застосовуються різні інструменти, які здатні сканувати код і

систему в реальному часі та знаходити можливі слабкі місця.

OWASP ZAP (Zed Attack Proxy) є одним із найбільш популярних інструментів для автоматизованого тестування безпеки web-застосунків. Цей інструмент може виконувати аналіз вразливостей, такі як XSS, CSRF, SQL injection, та багато інших, використовуючи як статичний, так і динамічний аналіз. Він також дозволяє проводити сканування під час розробки, що допомагає виявляти проблеми раніше, ніж вони потраплять у продуктивне середовище.

Застосування таких інструментів дозволяє не тільки виявити вразливості, але й усунути їх на ранніх етапах, забезпечуючи високий рівень захисту web-застосунку.

Таким чином, інтеграція цих аспектів — детектування вразливостей, інтеграція захисних механізмів і автоматизація тестування — є основними кроками для забезпечення безпеки front-end компонентів web-застосунків. Ці підходи дозволяють мінімізувати ризики атак і забезпечити стабільну та безпечну роботу web-застосунку.

1.2 Основні типи загроз для front-end компонентів web-застосунків

Загрози безпеці в front-end частині web-застосунків можуть бути різноманітними. Ось найбільш поширені:

1.2.1 Cross-Site Scripting (XSS)

XSS — це тип атаки, при якому зловмисник вставляє шкідливий JavaScript-код у web-сторінку, яку відвідує користувач. Цей код може бути використаний для викрадення сеансів користувачів, крадіжки облікових даних або навіть виконання дій від імені користувача без його відома.

Stored XSS: коли шкідливий код зберігається на сервері та потім відображається на сторінці кожного користувача.

Reflected XSS: коли шкідливий код відображається у відповідь на запит користувача, без збереження на сервері.

DOM-based XSS: коли скрипти на стороні клієнта змінюють DOM сторінки і впроваджують шкідливий код.

1.2.2 Cross-Site Request Forgery (CSRF)

CSRF — це тип атаки, при якому зловмисник змушує жертву виконати небажану операцію на web-застосунку, де вона вже авторизована. Це може бути, наприклад, зміна пароля або проведення фінансової транзакції, використовуючи автентифікацію жертви.

Механізм атаки CSRF:

Зловмисник створює шкідливу web-сторінку або посилання, яке виконуватиме запит від імені жертви.

Тому, якщо користувач вже авторизований, його сесія може бути використана для виконання цієї дії без його відома.

1.2.3 Clickjacking

Clickjacking — це метод атаки, при якому зловмисник розміщує невидимий або фальшивий елемент поверх легітимного інтерфейсу користувача, щоб змусити його клікнути на об'єкт, який він не мав наміру натискати, таким чином виконується небажана дія.

Це може бути, наприклад, коли кнопка на сайті розташована так, що під нею насправді знаходиться небезпечний елемент, натискання на який викликає іншу дію, як-от перехід на шкідливу сторінку або публікація особистої інформації.

1.2.4 Інші загрози

Серед інших поширених загроз для front-end компонентів можна також згадати такі:

SQL Injection: хоча ця атака більш характерна для серверної частини, вона може стати загрозою для front-end, якщо введені дані не правильно валідуються або передаються до серверу.

Insecure Direct Object References (IDOR): коли користувач може отримати доступ до ресурсів, яких не має права бачити, змінюючи ідентифікатор об'єкта в запиті.

1.3 Методи захисту від загроз

Захист від вищезгаданих атак вимагає впровадження комплексних заходів безпеки на різних етапах розробки та експлуатації web-застосунків.

CSP є потужним механізмом безпеки, який дозволяє web-розробникам вказувати, які ресурси можна завантажувати і виконувати на сторінках. Він значно зменшує ймовірність виконання шкідливих скриптів (наприклад, через XSS-атаки), дозволяючи тільки авторизованим джерелам завантажувати контент. Приклад принципу роботи CSP показано на рисунку 1.2, де показано, як політика CSP блокує завантаження ресурсу з небезпечного джерела.

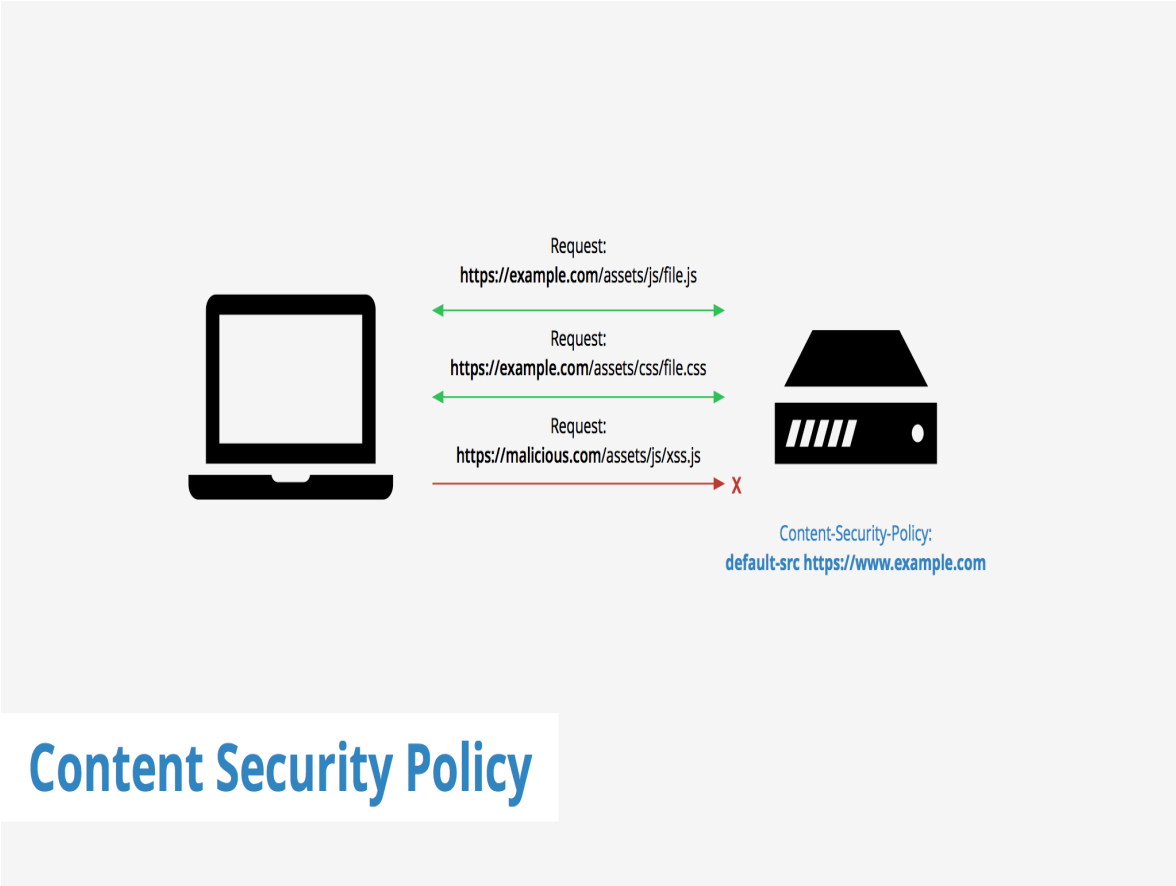


Рис. 1.2 Принцип роботи Content Security Policy (CSP)

Захист від CSRF-атак досягається шляхом впровадження CSRF-токенів. Ці токени додаються до кожного запиту, який змінює стан на сервері (наприклад, при відправці форм). Якщо токен не збігається з очікуваним, запит буде відхилений, що запобігає виконанню несанкціонованих дій (рис. 1.3).

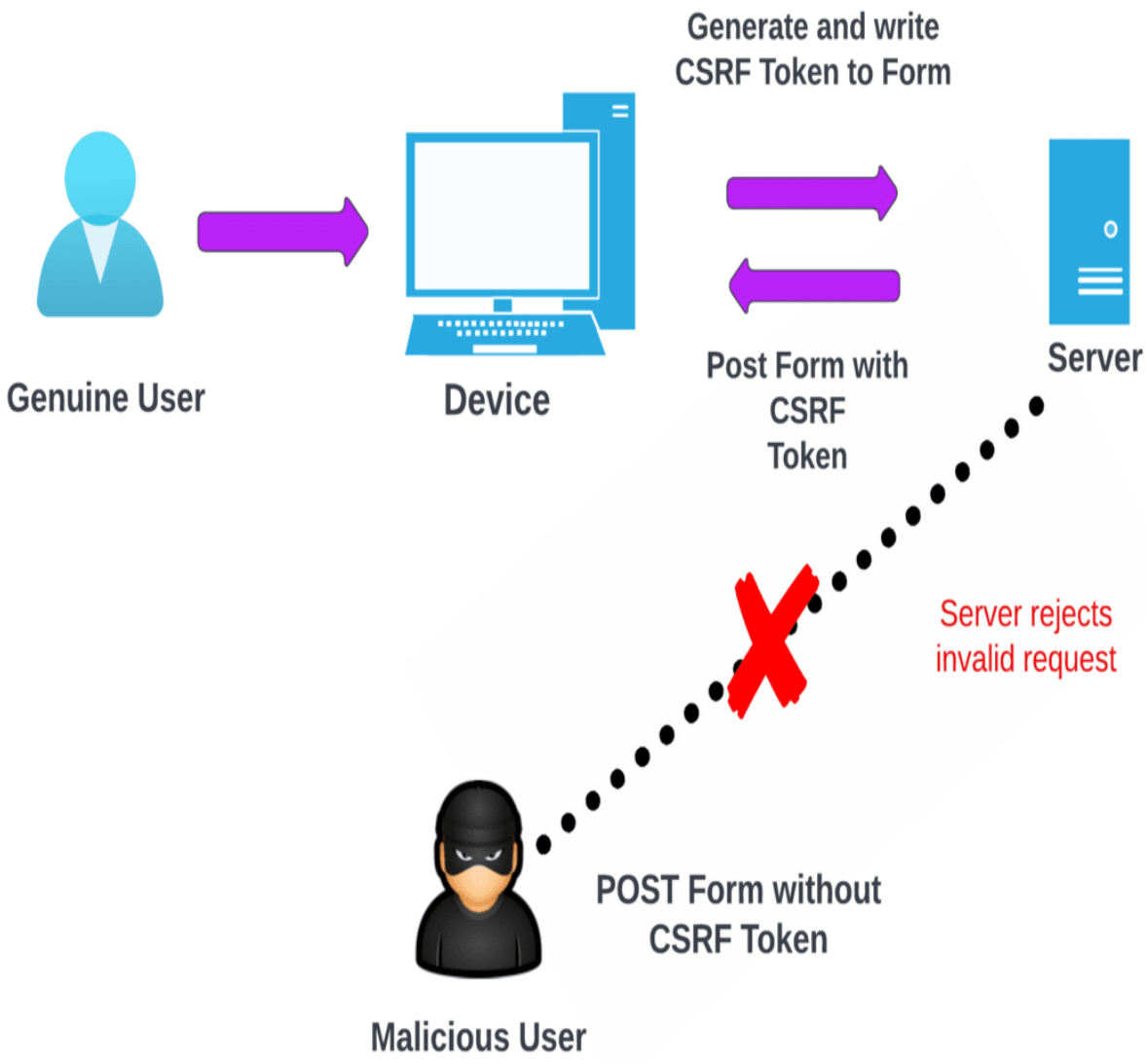


Рис. 1.3 Алгоритм роботи CSRF токенів

Один із найважливіших кроків для захисту від XSS і інших атак — це валідація та санітизація всіх введених даних. Web-застосунки повинні перевіряти дані на сервері, а також у JavaScript-коді на стороні клієнта перед їх використанням. Це допомагає запобігти введенню шкідливих скриптів. Ілюстрація процесу валідації введених даних і санітизації (рис. 1.4).

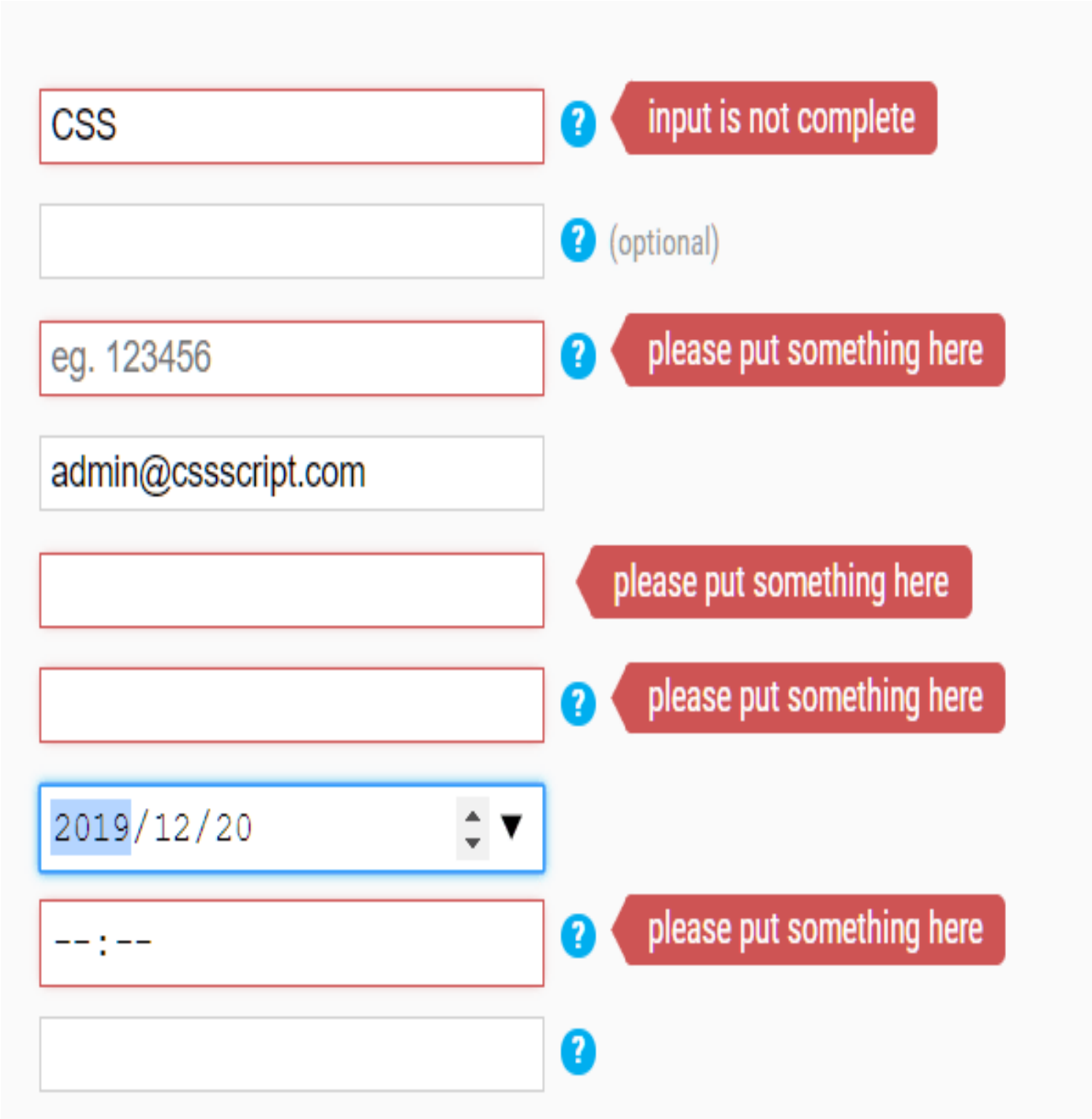


Рис. 1.4 Процес валідації та санітизації даних

Для запобігання Clickjacking важливо реалізувати захист від вставлення web-сторінок у фрейми (iFrame). Для цього можна використовувати заголовки HTTP, як-от X-Frame-Options, або директиви в CSP, які забороняють вбудовування сторінок в фрейми (рис 1.5).

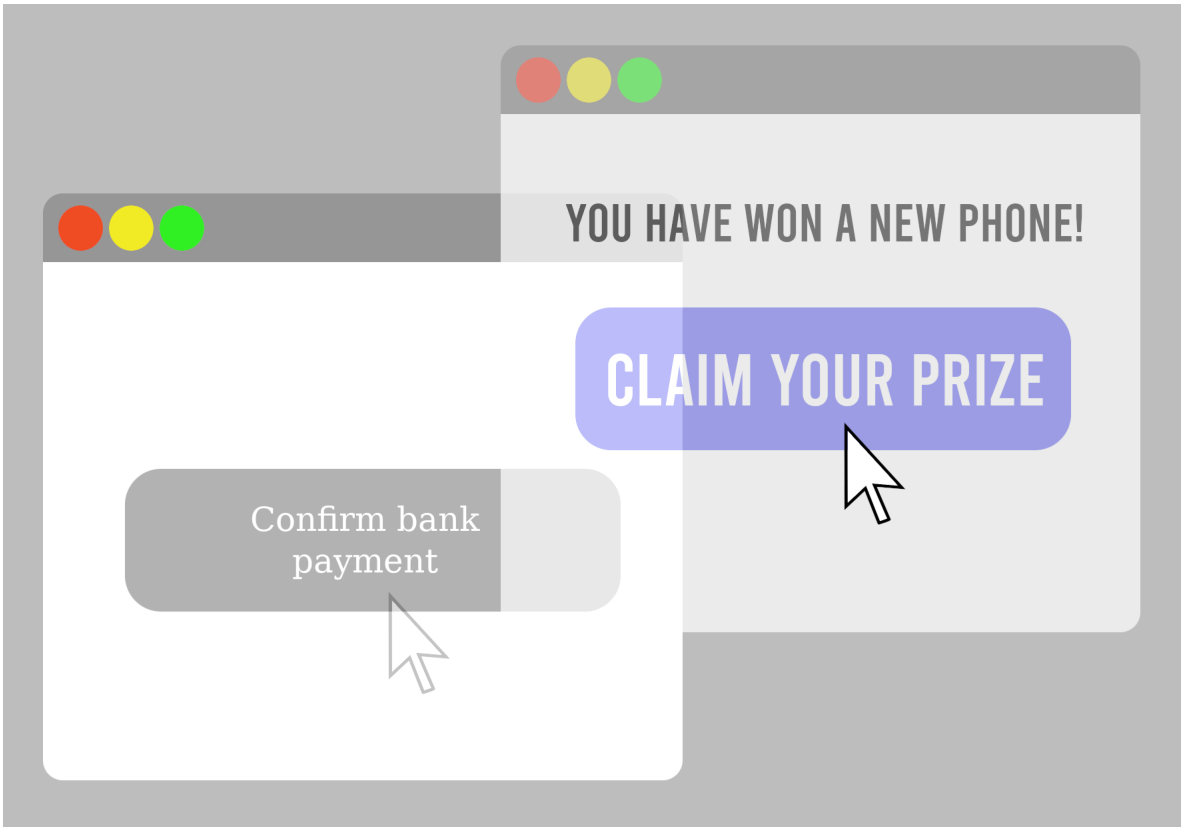


Рис. 1.5 Захист від Clickjacking через заголовки X-Frame-Options

Для додаткового захисту варто використовувати механізми, такі як Strict Transport Security (HSTS) для захисту від атак через незахищене з'єднання, а також обмеження прав доступу до чутливих даних (наприклад, за допомогою правильного налаштування CORS).

1.4 Інструменти для аналізу безпеки

Для забезпечення належного рівня безпеки web-застосунків важливо використовувати ефективні інструменти для аналізу та виявлення вразливостей. Вони дозволяють як автоматизувати процеси тестування, так і здійснювати ручне тестування для виявлення можливих слабких місць у коді. Основні інструменти для аналізу безпеки включають:

OWASP ZAP (Zed Attack Proxy) — це один з найпопулярніших інструментів для тестування безпеки web-застосунків. З його допомогою можна виявляти різні вразливості, такі як XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery),

SQL Injection та інші. ZAP дозволяє здійснювати автоматичний сканинг та використовувати його в рамках інтегрованих CI/CD процесів для перевірки безпеки додатків на різних етапах розробки (рис 1.6).

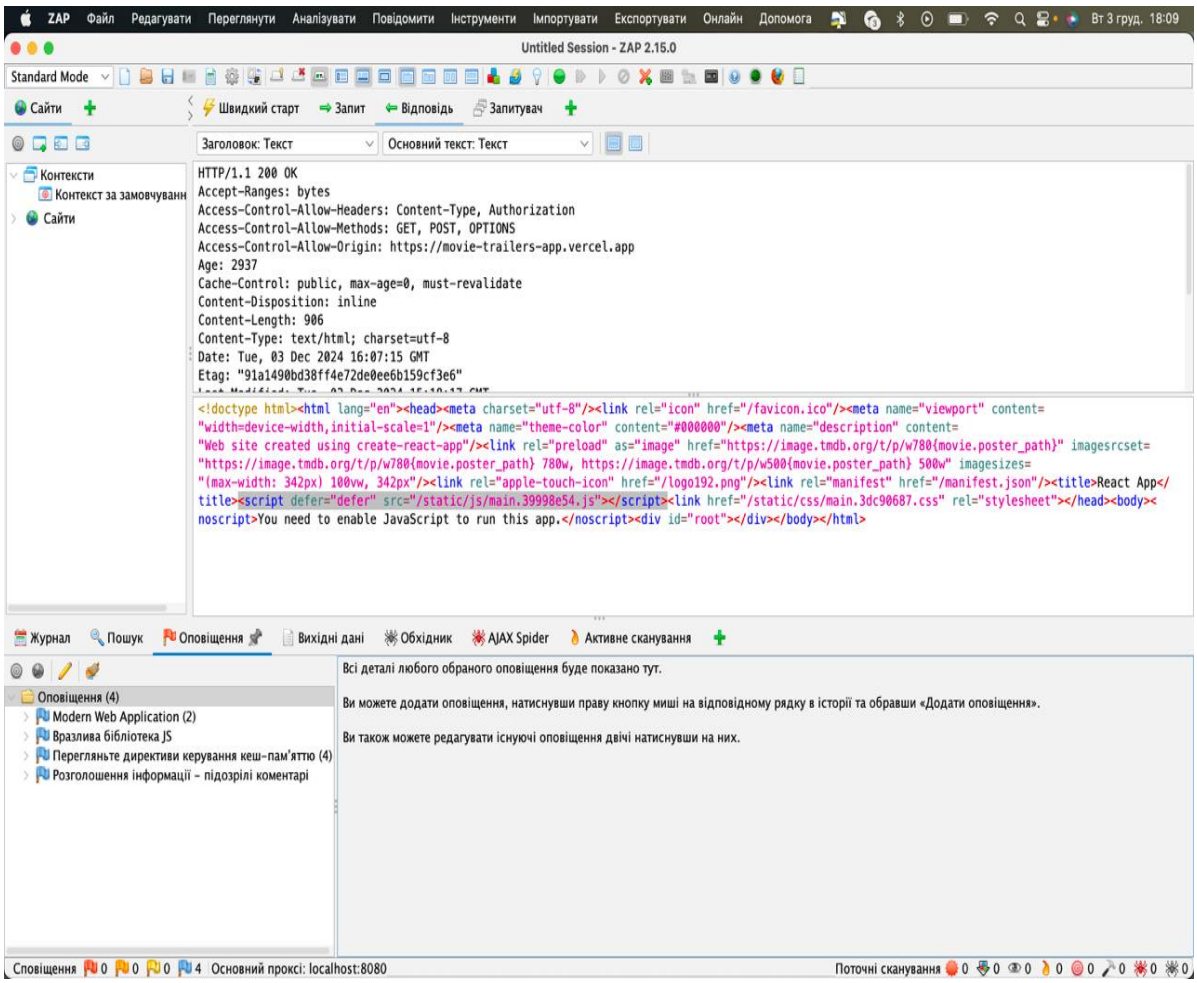


Рис. 1.6 Процес тестування безпеки за допомогою OWASP ZAP

Burp Suite є потужним інструментом для ручного та автоматизованого тестування безпеки web-застосунків. Він надає широкий набір функцій, включаючи проксі-сервер для аналізу запитів і відповідей, сканери уразливостей, інструменти для декодування та аналізу HTTP-трафіку, а також інтерфейси для ручного втручання у процес тестування. Burp Suite допомагає ідентифікувати низку вразливостей, таких як XSS, SQL Injection, Clickjacking та багато інших (рис. 1.8).

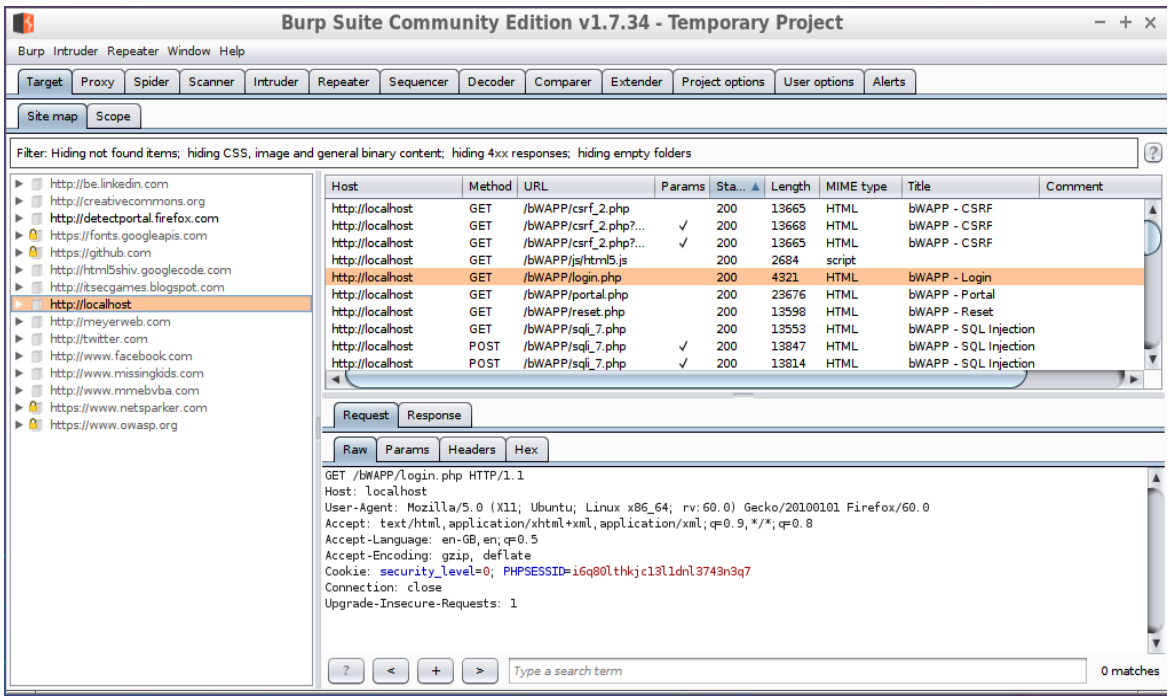


Рис. 1.8 Використання Burp Suite для тестування безпеки

Snyk — це інструмент, орієнтований на пошук вразливостей у залежностях, зокрема в JavaScript-проектах. Він дозволяє сканувати залежності у вашому проекті та перевіряти наявність відомих вразливостей у бібліотеках та пакетах, які ви використовуєте. Snyk допомагає розробникам швидко знаходити уразливі пакети та пропонує можливість їх оновлення чи заміни на більш безпечні версії (рис. 1.9).

Рис. 1.9 Використання Snyk для перевірки залежностей проекту

SonarQube — це інструмент для статичного аналізу коду, який дозволяє виявляти потенційні проблеми з безпекою в коді на ранніх етапах розробки. SonarQube перевіряє код на наявність помилок програмування, які можуть призвести до уразливостей, таких як неправильна обробка вводу користувача або незахищена передача даних. Це дозволяє розробникам оперативно виявляти та виправляти проблеми безпеки до того, як вони потраплять на продуктивну платформу (рис. 1.10).

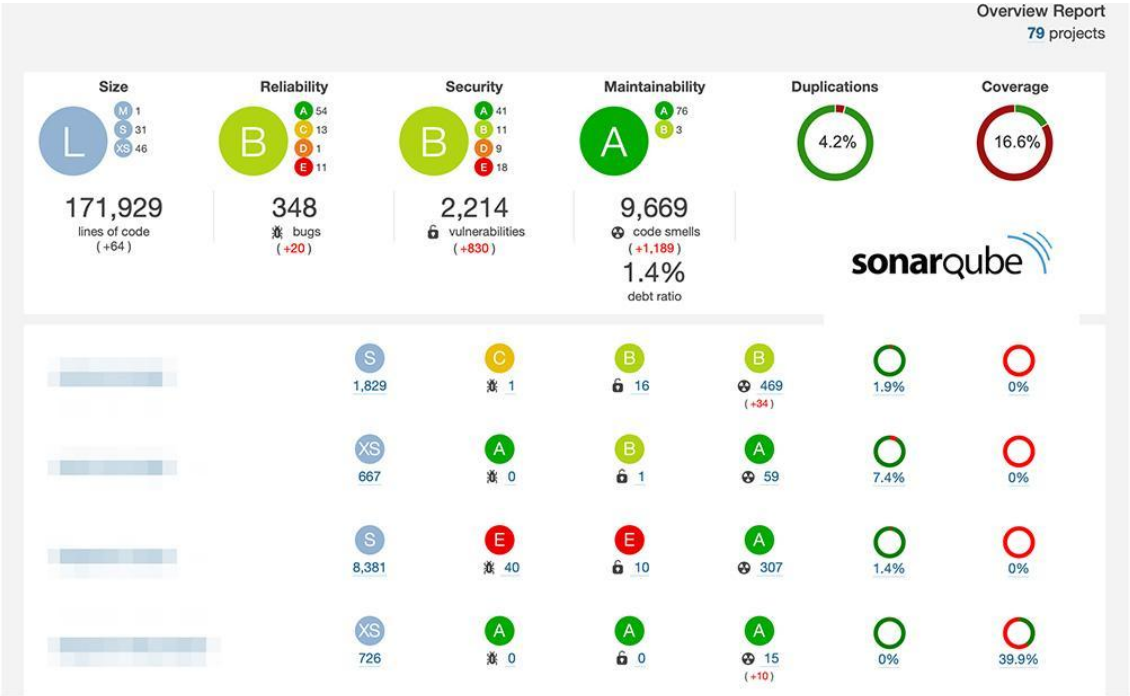


Рис. 1.10 Статичний аналіз коду за допомогою SonarQube

1.5 Огляд актуальних наукових робіт в галузі захисту web-застосунків

Книга «Web Application Security: A Beginner's Guide» [1] авторства Andrew Hoffman є одним із основних джерел для розуміння сучасних загроз і методів захисту web-застосунків. Вона охоплює основні техніки захисту, зокрема механізми захисту від атак XSS, SQL Injection та CSRF. Автор детально описує, як здійснюється захист веб-додатків за допомогою налаштувань безпеки, таких як Secure HTTP заголовки та шифрування даних.

Книга «The Web Application Hacker's Handbook» [2] авторів Dafydd Stuttard і Marcus Pinto надає глибоке розуміння того, як web-застосунки можуть бути скомпрометовані, та методів тестування безпеки. Вона включає опис багатьох вразливостей, таких як XSS і SQL Injection, а також надає практичні поради щодо безпеки на етапі розробки веб-додатків.

Звіт «OWASP Top Ten» [3] надає список найбільш поширених вразливостей web-застосунків, таких як небезпечна автентифікація, порушення контролю доступу та некоректне управління сесіями. Він є важливим ресурсом для розробників, які прагнуть забезпечити захист своїх веб-систем від актуальних загроз.

Книга «Security Engineering: A Guide to Building Dependable Distributed Systems» [4] авторства Ross J. Anderson є авторитетним джерелом для розуміння принципів безпеки в розподілених системах, зокрема web-застосунках. Вона охоплює такі аспекти, як криптографія, методи аутентифікації та захисту конфіденційності, що є критичними для захисту від атак.

Книга «The Art of Software Security Assessment» [5] авторства Mark Dowd, John McDonald, Justin Schuh пропонує комплексний підхід до оцінки безпеки програмного забезпечення, зокрема web-застосунків. Вона розглядає методи тестування на проникнення, виявлення вразливостей та стратегій захисту, що робить її важливим ресурсом для фахівців з безпеки.

Стаття «A Survey of Web Application Security Vulnerabilities and Countermeasures» [6] авторів Jayant M. Bhandari та Michael D. J. Bell досліджує основні види вразливостей веб-додатків та методи їх усунення. Вони аналізують поширені загрози, такі як CSRF, XSS і SQL Injection, а також пропонують стратегії захисту, які можуть бути застосовані на етапі розробки.

Звіт «OWASP Mobile Security Testing Guide» [7] пропонує набір рекомендацій і методів тестування безпеки мобільних додатків. Він охоплює питання захисту даних на мобільних пристроях, безпеки комунікацій і автентифікації, що є важливими аспектами для створення безпечних мобільних web-застосунків.

Стаття «Web Application Security: A Survey of Vulnerabilities and Protection

Mechanisms» [8] авторів Michael J. D. Hill та Tim G. Chapman пропонує огляд вразливостей web-застосунків, включаючи аутентифікацію, управління сесіями та криптографічні загрози. Вона надає комплексні рекомендації щодо використання методів захисту, таких як шифрування та налаштування безпеки на сервері.

Книга «Hacking: The Art of Exploitation» [9] авторства Jon Erickson є важливим джерелом для розуміння механізмів експлуатації вразливостей. Вона пояснює, як відбуваються атаки на web-застосунки, включаючи буферні переповнення, SQL Injection і XSS, а також надає стратегії для захисту від цих загроз.

Книга «Web Application Security: A Beginner's Guide» [10] авторства Bryan Sullivan та Vincent Liu є чудовим ресурсом для тих, хто тільки починає вивчати безпеку web-застосунків. Вона дає читачеві основи безпеки web-додатків, пояснюючи основні поняття та загрози, з якими стикаються web-розробники. Автори детально описують такі атаки, як SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), а також способи захисту від них.

Книга «Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems» [11] авторства Heather Adkins, Betsy Beyer, Paul Blankinship та Piotr Lewandowski фокусується на забезпеченні надійності та безпеки розподілених систем. Автори надають практичні поради для побудови безпечних і надійних систем.

Книга «Practical Web Penetration Testing» [14] авторства Gavin Watson присвячена практичним аспектам тестування на проникнення у веб-застосунки, з акцентом на реальні приклади та ситуації. Автор детально пояснює ключові поняття тестування, описує методи виявлення вразливостей у веб-додатках, таких як SQL Injection, Cross-Site Scripting (XSS), і CSRF (Cross-Site Request Forgery). У книзі описано використання популярних інструментів для тестування, таких як Burp Suite, OWASP ZAP, і Nikto, а також розглядаються стратегії написання звітів про знайдені вразливості. Книга ідеально підходить для тих, хто бажає розвинути практичні навички у веб-пентестуванні.

Книга «Kali Linux Web Penetration Testing Cookbook» [15] авторства Cesar

Pereira

посібник розрахований на початківців і фахівців, які хочуть освоїти пентестування веб-застосунків з використанням дистрибутива Kali Linux. У книзі зібрані покрокові рецепти для вирішення різних завдань, від налаштування середовища до виконання складних атак на веб-застосунки. Особлива увага приділяється таким інструментам, як Nmap, Metasploit, sqlmap, і Hydra. Автор також розглядає методи автоматизації тестування та аналізу отриманих результатів, що робить книгу цінним ресурсом для практиків.

Книга «The Hacker Playbook 2: Practical Guide To Penetration Testing» [16] авторства Peter Kim потужне керівництво для фахівців з кібербезпеки, які прагнуть зрозуміти, як працюють етичні хакери (пентестери). У книзі містяться реалістичні сценарії атак і поради щодо їх ефективного виконання. Автор детально розповідає про методології тестування на проникнення, техніки соціальної інженерії, а також про інструменти для аналізу та експлуатації вразливостей. Особливу увагу приділено побудові лабораторного середовища, що дозволяє відпрацьовувати отримані знання на практиці.

Книга «Web Application Security, Analyzing and Securing Web Applications» [17] авторства Andrew Hoffman охоплює ключові аспекти безпеки веб-застосунків, надаючи розробникам і спеціалістам з безпеки інструменти для захисту своїх додатків. Автор пояснює різні типи атак, такі як Injection, XSS, і Denial of Service (DoS), а також описує способи їх запобігання. У книзі також висвітлено принципи побудови безпечної архітектури веб-застосунків, інтеграцію безпеки у процес розробки (DevSecOps) і рекомендації щодо використання безпечних API.

Книга «Applied Cryptography: Protocols, Algorithms, and Source Code in C» [18] авторства Bruce Schneier класична книга з криптографії охоплює широкий спектр тем: від базових принципів криптографії до складних алгоритмів і протоколів. У книзі описані симетричні та асиметричні алгоритми шифрування, методи створення цифрових підписів, управління ключами і захист даних. Особливістю є наявність прикладів вихідного коду мовою C, що робить її незамінним посібником для розробників і дослідників.

Книга «Web Security for Developers» [19] авторства Malcolm McDonald призначена для розробників, які бажають освоїти основи веб-безпеки. Автор пояснює складні концепції простими словами, зосереджуючи увагу на практичних аспектах. У книзі висвітлюються поширені вразливості, такі як XSS, SQL Injection, і Broken Authentication, та надаються конкретні рекомендації щодо їх усунення. Особливий акцент зроблено на впровадженні безпечних практик у процес розробки веб-додатків.

Книга «The Hacker Playbook 3: Practical Guide to Penetration Testing» [20] авторства Peter Kim продовження популярної серії книг, яке надає ще більше інструментів і технік для тестування на проникнення. Автор акцентує увагу на сучасних методах атак і захисту, включаючи атаки на хмарні сервіси та мобільні додатки. У книзі розглядається використання новітніх інструментів для автоматизації процесу тестування, а також обговорюються питання побудови стратегії багаторівневої безпеки.

Книга «Securing DevOps: Security in the Cloud» [21] авторства Julian L. González та Roberto Di Cosmo зосереджується на безпеці при використанні DevOps-практик і хмарних технологій. Автори пояснюють, як інтегрувати безпеку у життєвий цикл розробки, тестування і впровадження додатків. Особливий акцент зроблено на забезпеченні безпеки хмарних сервісів, автоматизації моніторингу та реагування на інциденти. Книга стане корисною для DevOps-інженерів і спеціалістів з безпеки, які працюють у хмарних середовищах.

Книга «Network Security Essentials» [22] авторства William Stallings книга охоплює ключові поняття мережевої безпеки, включаючи принципи шифрування, управління доступом і методи захисту мережевих протоколів. Автор детально пояснює роботу мережевих засобів захисту, таких як брандмауери, VPN, і системи виявлення вторгнень. У книзі також наведено приклади реальних загроз і способи їх нейтралізації, що робить її корисним ресурсом для студентів і професіоналів у сфері кібербезпеки.

2 АНАЛІЗ МЕТОДІВ ЗАХИСТУ WEB-ЗАСТОСУНКІВ ВІД ВРАЗЛИВОСТЕЙ

2.1 Аналіз існуючих методів захисту web-застосунків

2.1.1 Використання токенів CSRF

Метод захисту web-застосунків від атак міжсайтових запитів (CSRF, Cross-Site Request Forgery) за допомогою унікальних токенів є широко розповсюдженим і надійним рішенням, яке дозволяє забезпечити безпеку взаємодії клієнта із сервером. У сучасних web-застосунках цей метод є одним із найбільш поширених, адже він добре інтегрується із існуючими фреймворками та підходить для реалізації як у монолітній, так і у мікросервісній архітектурі.

CSRF-атаки використовують довірливі взаємодії між браузером користувача та сервером. Наприклад, коли користувач авторизований у web-застосунку, браузер зберігає сесію у вигляді cookie. Зловмисник може змусити користувача, навіть не помічаючи цього, відправити небажаний запит до сервера, використовуючи активну сесію. Це може призводити до виконання різноманітних дій, таких як зміна паролів, пересилання грошей, видалення даних тощо.

CSRF-токен — це унікальний рядок символів, створений сервером, який асоціюється із сесією користувача. Він додається до кожного запиту, який надходить від клієнта. Сервер перевіряє токен у кожному запиті й дозволяє його виконання лише у випадку відповідності токена.

Під час авторизації користувача сервер створює унікальний токен, який зберігається в пам’яті сервера (сесія) або у базі даних. Токен генерується із застосуванням криптографічно стійких методів, таких як HMAC (хешування з ключем).

Токен додається до кожної форми або запиту (POST, PUT, DELETE). Він передається як приховане поле форми, параметр URL або спеціальний заголовок HTTP.

Після отримання запиту сервер порівнює токен із тим, що був згенерований під час сесії. Якщо токен відсутній або не відповідає збереженому значенню, запит блокується.

Якщо токен недійсний або не переданий, сервер повертає помилку (наприклад, HTTP 403 Forbidden), відмовляючись виконувати запит.

Алгоритм роботи CSRF-токенів

Алгоритм дій для захисту web-застосунку від CSRF-атак включає наступні етапи:

$$T = Base64(HMAC_k(UID||TS)), \quad (2.1)$$

де UID — унікальний ідентифікатор користувача;

TS — мітка часу;

K — секретний ключ сервера;

HMACHMACHMAC — хеш-функція з ключем;

Base64 — метод кодування.

Приклад реалізації перевірки CSRF-токенів на JavaScript:

1. Генерація токена**:**

   Використовуємо бібліотеку crypto для створення токена та збереження його в сесії (рис. 2.1).

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

// Middleware для генерації CSRF-токена
function generateCsrfToken(req, res, next) {
    if (!req.session.csrfToken) {
        req.session.csrfToken = crypto.randomBytes(32).toString('hex');
    }
    res.locals.csrfToken = req.session.csrfToken; // Додаємо токен у відповіді
    next();
}

// Підключаємо middleware
app.use(generateCsrfToken);
```

Рис 2.1. Приклад генерації токена

2. Передача токена клієнту. Додаємо токен у відповідь як частину HTML-

форми або в заголовок запиту (рис. 2.2).

```
const csrfToken = '<%= csrfToken %>';
fetch('/submit', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'CSRF-Token': csrfToken // Передаємо токен у заголовку
    },
    body: JSON.stringify({ username: 'user123' })
});
```

Рис 2.2 Приклад передачі токена клієнту

3. Перевірка токена на сервері**:**

Додаємо middleware для перевірки валідності токена: (рис. 2.3).

```
function verifyCsrfToken(req, res, next) {
    const tokenFromClient = req.body.csrf_token || req.headers['csrf-token'];
    if (!tokenFromClient || tokenFromClient !== req.session.csrfToken) {
        return res.status(403).json({ error: 'CSRF token is invalid or missing' });
    }
    next();
}

// Використовуємо middleware для захищених маршрутів
app.post('/submit', verifyCsrfToken, (req, res) => {
    res.json({ message: 'Data submitted successfully!' });
});
```

Рис 2.3 Приклад перевірки токена на сервері

Багато сучасних web-фреймворків мають вбудовану підтримку CSRF-токенів:

Django: Автоматично додає CSRF-токен до кожної форми у шаблонах.

Laravel: Використовує middleware для перевірки CSRF-токенів.

Spring Security: Реалізує CSRF-захист для Java-застосунків, додаючи токени до кожного запиту.

Переваги використання CSRF-токенів:

— Високий рівень захисту від CSRF-атак.

— Простота впровадження у сучасних фреймворках.

— Мінімальні вимоги до ресурсів сервера.

— Легкість інтеграції з іншими механізмами безпеки.

— Недоліки

— Необхідність зберігання токенів на сервері.

Складнощі інтеграції із зовнішніми сервісами, які не підтримують CSRF-захист. Ризики компрометації токенів при недостатньому захисті каналів передачі даних. Таким чином, CSRF-токени є ефективним і широко використовуваним методом захисту web-застосунків, що забезпечує високий рівень безпеки при виконанні запитів.

2.1.2 Політика безпеки контенту (Content Security Policy, CSP)

Політика безпеки контенту (CSP) — це механізм web-безпеки, спрямований на запобігання різним атакам, таким як XSS (міжсайтовий скриптинг) і ін'єкція шкідливого контенту. CSP дозволяє web-додаткам обмежувати, які ресурси можуть бути завантажені та виконані на сторінці, тим самим зменшуючи ризики від експлуатації вразливостей у браузері.

CSP працює шляхом впровадження заголовка HTTP або тегу <meta>, що визначає джерела дозволеного контенту. Основні елементи CSP:

Директиви — визначають типи контенту, які контролюються (скрипти, стилі, зображення тощо).

Джерела — встановлюють, звідки контент може бути завантажений (наприклад, self, trusted-domain.com).

Політика за замовчуванням — використовується для всіх типів контенту, якщо для конкретного не визначено окремої директиви (рис. 2.4).

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted-scripts.com;
    style-src 'self' https://trusted-styles.com;
```

Рис 2.4 Приклад CSP

default-src 'self': дозволяє завантаження контенту лише з того ж походження.

script-src: дозволяє виконання скриптів лише з self і trusted-scripts.com.

style-src: обмежує стилі до self і trusted-styles.com.

CSP впроваджується через заголовок HTTP або тег <meta> у HTML-документі.

1. HTTP-заголовок**:** (рис. 2.5).

```
Content-Security-Policy: default-src 'self'; img-src https://trusted-images.com; script-src
    'self' 'unsafe-inline';
```

Рис 2.5 Приклад впровадження CSP

2. Тег <meta>: (рис. 2.6)

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'">
```

Рис 2.6 Приклад тегу meta

Основні директиви CSP:

— default-src: політика за замовчуванням для всіх ресурсів.

— script-src: дозволені джерела виконуваних скриптів.

— style-src: дозволені джерела стилів.

— img-src: дозволені джерела зображень.

— connect-src: дозволені джерела для запитів через XHR, fetch, WebSocket.

— font-src: дозволені джерела шрифтів.

— frame-src: дозволені джерела для вставки фреймів.

CSP є важливим інструментом у запобіганні XSS-атакам, які зазвичай полягають у виконанні на сторінці шкідливих скриптів, ін'єктованих атакуючим. CSP дозволяє заборонити виконання inline-скриптів ('unsafe-inline') та виконання JavaScript-коду з неперевірених джерел. Наприклад: Content-Security-Policy: script-src 'self'. У цьому випадку будь-які скрипти, що завантажуються з іншого домену, будуть заблоковані. Впровадження CSP у Node.js Рис(2.2.3) у захист від XSS: запобігає виконанню шкідливих скриптів на сторінці. Контроль над контентом: дозволяє чітко визначати джерела, з яких може бути завантажений контент.

Зменшення ризику ін'єкції даних: запобігає підключенню небезпечних ресурсів.

Можливість звітності: підтримує режим report-uri, що дозволяє отримувати звіти про порушення політики. Складність налаштування: розробникам може бути важко налаштувати політику для складних додатків. Inline-скрипти та стилі: для їх дозволу потрібно вказувати unsafe-inline, що послаблює захист.

```
const express = require('express');
const helmet = require('helmet');
const app = express();

// Використання CSP через helmet
app.use(helmet.contentSecurityPolicy({
    directives: {
        defaultSrc: ["'self'"],
        scriptSrc: ["'self'", "https://trusted-scripts.com"],
        styleSrc: ["'self'", "https://trusted-styles.com"],
        imgSrc: ["'self'", "https://trusted-images.com"]
    }
}));

app.get('/', (req, res) => {
    res.send('<h1>Hello, CSP!</h1>');
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Рис 2.7 Впровадження CSP у Node.js

Сторонні ресурси: якщо застосунок залежить від багатьох сторонніх сервісів, створення безпечної політики може бути викликом.

2.1.3 Валідація та санітизація вхідних даних

Валідація та санітизація вхідних даних є одним із ключових методів забезпечення безпеки web-додатків. Вони спрямовані на перевірку достовірності, відповідності та безпеки даних, які надходять від користувачів або зовнішніх джерел, з метою запобігання їх використанню для атак, таких як SQL-ін'єкції, XSS (міжсайтовий скриптинг), атаки на шлях до файлів тощо.

Різниця між валідацією та санітизацією

Валідація — процес перевірки, чи відповідають введені дані очікуваним правилам (наприклад, типу, формату, діапазону значень).

Санітизація — процес очищення даних від шкідливих або небажаних елементів, таких як шкідливий код чи спеціальні символи.

Обидва процеси часто використовуються разом для забезпечення максимальної безпеки.

Основні принципи валідації вхідних даних

1. Валідація на стороні сервера**:** оскільки дані, перевірені лише на стороні клієнта, можуть бути змінені перед відправкою на сервер.
2. Використання чітких правил: визначення конкретного типу, формату та діапазону для кожного введення.
3. Чітке розмежування даних**:** розрізнення довірених і недовірених даних, особливо у випадках взаємодії з базою даних або зовнішніми API.
4. Блокування необов'язкових полів**:** якщо поле не використовується у процесі, його не слід обробляти.

Основні техніки валідації

1. Перевірка типу даних**:** Перевіряється, чи відповідає введення очікуваному типу (рядок, число, булеве значення тощо) (рис 2.8).

```
function validateNumber(input) {
    if (typeof input !== 'number') {
        throw new Error('Invalid input: number expected.');
    }
    return input;
}
```

Рис 2.8 Приклад перевірки типу даних

2. Перевірка формату**:** Наприклад, перевірка електронної пошти, телефонного номера чи URL за допомогою регулярних виразів (рис. 2.9).

```
function validateEmail(email) {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        throw new Error('Invalid email format.');
    }
    return email;
}
```

Рис 2.9 Приклад перевірки формату

3. Обмеження довжини**:** Перевірка кількості символів для запобігання переповненню буфера (рис. 2.10).

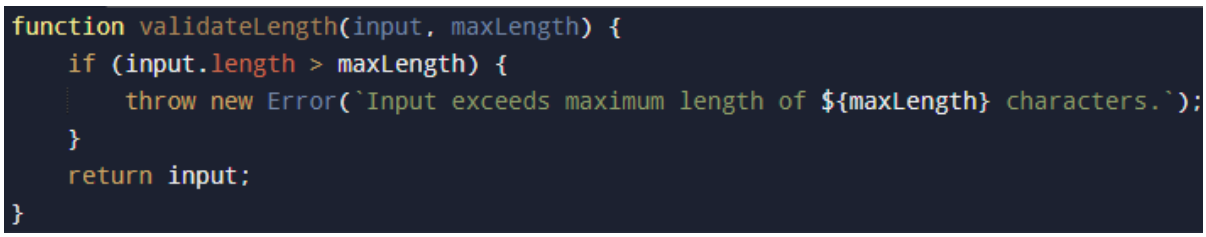


Рис 2.10 Приклад обмеження довжини

4. Перевірка діапазону значень**:** Забезпечує, щоб числові дані не виходили за межі допустимого діапазону (рис. 2.11).

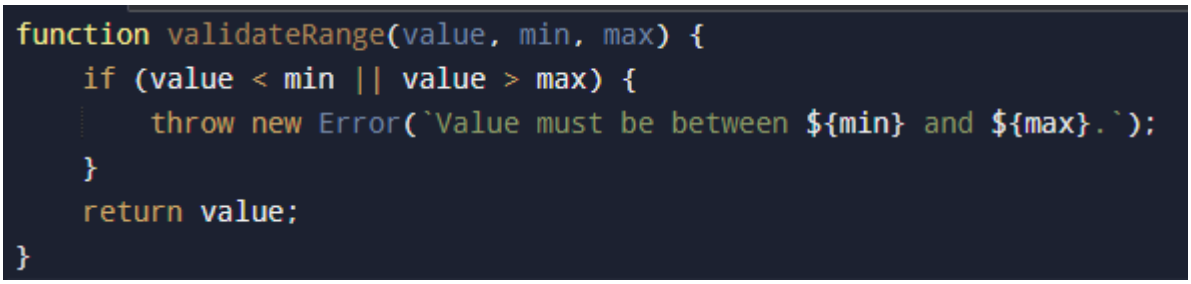


Рис 2.11 Перевірка діпазону значень

Екранування символів та усіх спеціальних символів, які можуть бути використані для SQL-ін'єкцій або XSS, повинні бути замінені безпечними значеннями (рис. 2.12).

```
function sanitizeInput(input) {
    return input.replace(/[&<>"'\/]/g, (char) => {
        const map = {
            '&': '&amp;',
            '<': '&lt;',
            '>': '&gt;',
            '"': '&quot;',
            "'": '&#39;',
            '/': '&#x2F;'
        };
        return map[char];
    });
}
```

Рис 2.12 Основні принципи санітизації даних

1. Видалення небезпечного коду: Використання бібліотек для очищення HTML-вмісту, наприклад, DOMPurify (рис. 2.13).

```
const DOMPurify = require('dompurify');
const sanitizedHTML = DOMPurify.sanitize(userInputHTML);
```

Рис 2.13 Видалення небезпечного коду

2. Обмеження доступу до небезпечних ресурсів**:** Наприклад, запобігання введенню абсолютних шляхів до файлів.

Запобігання SQL-ін'єкціям завдяки перевірці та екрануванню вхідних даних забезпечується захист бази даних від шкідливих запитів (рис. 2.14).

```
const query = 'SELECT * FROM users WHERE username = ? AND password = ?';
db.query(query, [username, password]);
```

Рис 2.14 Запобігання SQL-ін'єкціям

1. Захист від XSS: Очистка введення запобігає виконанню шкідливих скриптів у браузері.
2. Зменшення ризику атак на файлову систему**:** Забезпечується перевірка введення шляху до файлу, щоб уникнути доступу до конфіденційних файлів (рис. 2.15).

```
const fs = require('fs');
function safeFilePath(inputPath) {
    const basePath = '/uploads';
    const resolvedPath = path.resolve(basePath, inputPath);
    if (!resolvedPath.startsWith(basePath)) {
        throw new Error('Access denied: invalid file path.');
    }
    return resolvedPath;
}
```

Рис 2.15 Зменшення ризику атак на файлову систему

3. Забезпечення коректної роботи додатку: Неправильні або некоректні дані можуть викликати помилки в роботі додатку, а валідація їх запобігає.

Інструменти для реалізації валідації та санітизації

Joi (JavaScript): бібліотека для валідації даних.

Validator.js: дозволяє перевіряти email, URL, строки тощо.

Express Validator: middleware для перевірки даних у запитах у Express-додатках.

DOMPurify: очищення HTML від шкідливих елементів.

Приклад у Node.js (рис. 2.16).

```
const express = require('express');
const { body, validationResult } = require('express-validator');

const app = express();
app.use(express.json());

app.post('/submit', [
    body('email').isEmail().withMessage('Invalid email format'),
    body('age').isInt({ min: 18, max: 99 }).withMessage('Age must be between 18 and 99'),
    body('username').trim().escape().isLength({ min: 3 }).withMessage('Username must be at
        least 3 characters long')
], (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }
    res.send('Data is valid and sanitized');
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Рис 2.16 Приклад у Node.js

2.1.4 Використання HTTPS і TLS

HTTPS (Hypertext Transfer Protocol Secure) — це розширення протоколу HTTP, яке забезпечує захист передавання даних між клієнтом і сервером через використання протоколу шифрування TLS (Transport Layer Security). Цей механізм є одним із базових і найефективніших методів забезпечення безпеки web-додатків, що запобігає атакам, таким як перехоплення даних (man-in-the-middle) або спуфінг

з'єднання.

Основні принципи HTTPS та TLS

Шифрування даних: TLS забезпечує, щоб усі дані, які передаються між клієнтом і сервером, були зашифровані. Це унеможливлює їх перегляд або модифікацію під час передачі.

Аутентифікація: Використання сертифікатів TLS гарантує, що клієнт взаємодіє саме з тим сервером, з яким має, що захищає від атак, пов'язаних із підробкою серверів.

Цілісність даних: TLS перевіряє, щоб дані, передані між клієнтом і сервером, не були змінені під час передачі.

Як працює HTTPS і TLS

— Клієнт надсилає серверу запит на безпечне з'єднання (HTTPS).
— Сервер відповідає сертифікатом TLS, який містить його ідентифікаційні дані.
— Клієнт перевіряє сертифікат через центр сертифікації (CA).
— Клієнт і сервер генерують секретний ключ шифрування.
— Дані між клієнтом і сервером шифруються цим ключем.

Усі дані, що передаються між клієнтом і сервером, шифруються та розшифровуються лише сторонами з доступом до секретного ключа.

Переваги використання HTTPS

Захист від перехоплення даних: Наприклад, атаки типу "людина посередині" стають значно складнішими через шифрування даних.

Підвищення довіри користувачів: Більшість сучасних браузерів відображають попередження для HTTP-сайтів як «небезпечні», що відлякує користувачів.

Покращення SEO: HTTPS є одним із факторів ранжування в пошукових системах, таких як Google.

Захист від атак типу повторного відтворення (replay attacks): Дані, захищені

TLS, мають спеціальні мітки часу, що унеможливлює повторне використання перехоплених даних.

Приклад використання HTTPS у Node.js (рис. 2.17).

```
const https = require('https');
const fs = require('fs');

// Завантаження сертифікатів
const options = {
    key: fs.readFileSync('server.key'), // Приватний ключ
    cert: fs.readFileSync('server.crt') // Сертифікат сервера
};

// Створення HTTPS сервера
https.createServer(options, (req, res) => {
    res.writeHead(200);
    res.end('Secure connection established.');
}).listen(3000, () => {
    console.log('HTTPS server running on port 3000');
```

Рис 2.17 Приклад використання HTTPS у Node.js

Недоліки та виклики використання HTTPS

1. Вартість сертифікатів: Сертифікати TLS можуть бути дорогими, хоча існують безкоштовні варіанти, такі як Let’s Encrypt**.**
2. Збільшення навантаження на сервер: Шифрування та розшифрування даних потребує додаткових обчислювальних ресурсів.
3. Проблеми сумісності: Старі браузери або пристрої можуть не підтримувати новіші версії TLS.

Базові рекомендації щодо впровадження HTTPS

1. Використовуйте сертифікати, видані авторитетними центрами сертифікації (CA).
2. Регулярно оновлюйте версію протоколу TLS на сервері (TLS 1.3 на сьогодні є найсучаснішим і безпечним стандартом).

3. Налаштуйте HSTS (HTTP Strict Transport Security), щоб примусово змушувати браузери використовувати HTTPS.

Strict-Transport-Security: max-age=31536000; includeSubDomains

4. Видаляйте підтримку застарілих протоколів, таких як SSL і TLS 1.0, які мають відомі вразливості.

Інструменти та сервіси для роботи з HTTPS

1. Let’s Encrypt — безкоштовний і автоматизований центр сертифікації.
2. Qualys SSL Labs — сервіс для перевірки налаштувань HTTPS на вашому сервері.
3. OpenSSL — бібліотека для генерації сертифікатів і налаштування безпечних з'єднань.

Поширені помилки при впровадженні HTTPS

1. Використання сертифікатів із терміном дії, що минув.
2. Неправильна конфігурація проміжних сертифікатів (intermediate certificates).
3. Відсутність HSTS, через що HTTP-з'єднання можуть залишатися доступними.

2.1.5 Автентифікація та авторизація

Автентифікація та авторизація є ключовими етапами у забезпеченні безпеки web-додатків, оскільки вони дозволяють встановити особу користувача і надати йому доступ лише до відповідних ресурсів або функціоналу. Неправильне впровадження цих механізмів може призвести до таких загроз, як несанкціонований доступ, злам облікових записів, крадіжка даних або навіть компрометація всієї системи.

Автентифікація (authentication) — це процес перевірки особи користувача шляхом підтвердження його облікових даних, таких як логін, пароль, токени або біометричні дані.

Авторизація (authorization) — це процес визначення прав користувача після його автентифікації, тобто які ресурси, дії або функціонал йому доступні.

Методи автентифікації

1. Паролі**:**

   Найпоширеніший, але водночас уразливий метод автентифікації. Важливо використовувати алгоритми хешування для збереження паролів (рис. 2.18).

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
const password = "userPassword";

bcrypt.hash(password, saltRounds, (err, hash) => {
    if (err) throw err;
    console.log("Hashed password:", hash);
});
```

Рис 2.18 Приклад алгоритму хешування для збереження паролів

2. Одноразові паролі (OTP): Використання SMS, email або додатків-генераторів (наприклад, Google Authenticator) для генерування одноразового пароля.
3. Біометрична автентифікація: Використання унікальних фізичних характеристик користувача, таких як відбитки пальців, сканування обличчя або райдужки ока.
4. Автентифікація за допомогою токенів:

   JWT (JSON Web Token) — популярний формат для передачі автентифікаційної інформації (рис. 2.19).

```
const jwt = require('jsonwebtoken');
const user = { id: 1, username: "user" };
const secretKey = "yourSecretKey";

const token = jwt.sign(user, secretKey, { expiresIn: '1h' });
console.log("Generated JWT:", token);
```

Рис 2.19 Приклад створення токена в Node.js

Методи авторизації

1. Рівнева авторизація

   Застосовується для розділення користувачів за ролями (наприклад, адмін, модератор, користувач (рис. 2.20).

```
const userRole = "admin";

if (userRole === "admin") {
    console.log("Access granted to admin resources.");
} else {
    console.log("Access denied.");
}
```

Рис 2.20 Приклад перевірки ролі

2. Декларативна авторизація

   Користувачам призначаються конкретні дозволи для виконання певних

дій (наприклад, читання, запис, видалення).

3. Контекстна авторизація**:**

   Дозволяє враховувати додаткові фактори, такі як IP-адреса, час доби, тип пристрою.

2.2 Аналіз переваг та недоліків методів захисту web-застосунків

Захист web-застосунків є надзвичайно важливим аспектом у розробці сучасних інформаційних систем. Вразливості в безпеці можуть призводити до таких серйозних наслідків, як витік персональних даних, несанкціонований доступ до ресурсів, зниження репутації компаній або навіть фінансові втрати. Для мінімізації ризиків та забезпечення стійкості системи необхідно впроваджувати різноманітні методи захисту, які відповідають сучасним вимогам безпеки.

Одним із найбільш розповсюджених методів є запобігання SQL-ін'єкціям — атакам, які використовуються для впровадження шкідливого SQL-коду через форми введення даних. Це досягається завдяки використанню підготовлених запитів (prepared statements) та параметризованих запитів, які дозволяють відокремлювати дані користувача від коду SQL. Застосування ORM (Object-Relational Mapping) також значно знижує ризик SQL-ін'єкцій, адже ці фреймворки автоматично обробляють введені дані перед їхньою інтеграцією у запити.

Для запобігання атакам міжсайтових сценаріїв (XSS), які можуть призводити до виконання шкідливого коду в браузерах користувачів, застосовуються такі методи, як політика безпеки контенту (CSP) та санітизація введених даних. CSP дозволяє чітко визначити, які джерела контенту можуть завантажуватися на сторінки застосунку, а санітизація очищує введені користувачами дані від потенційно небезпечних елементів, таких як теги <script> або небезпечні атрибути HTML.

Захист від міжсайтових запитів (CSRF) базується на використанні токенів, які генеруються сервером та додаються до кожного запиту, що потребує виконання важливих операцій. Ці токени унікальні для кожної сесії та перевіряються сервером

перед виконанням запиту, що дозволяє запобігти виконанню несанкціонованих дій від імені користувача.

Використання HTTPS у поєднанні з протоколом TLS забезпечує шифрування всіх даних, що передаються між сервером та клієнтом. Це значно знижує ризик перехоплення чутливої інформації, наприклад, паролів або банківських реквізитів, під час їхнього передавання. Перехід на HTTPS є обов’язковим для будь-якого сучасного web-застосунку, оскільки це не лише підвищує безпеку, а й покращує SEO-рейтинги сайту.

Системи автентифікації та авторизації також потребують особливої уваги. Серед передових підходів до автентифікації слід виділити багатофакторну автентифікацію (2FA), яка поєднує щонайменше два незалежні фактори підтвердження особи користувача: наприклад, пароль та код із мобільного додатка. Це значно ускладнює завдання для зловмисників, навіть якщо їм вдасться отримати пароль.

Ефективне управління сеансами включає такі заходи, як використання короткоживучих токенів, автоматичний вихід із системи після певного часу бездіяльності, а також шифрування та підпис токенів для запобігання їх підробці.

Окрім технічних заходів, велике значення мають організаційні аспекти, зокрема регулярне оновлення серверного програмного забезпечення, використання сучасних фреймворків, які мають вбудовані засоби захисту, та проведення аудиту безпеки. Останнє може включати тестування на проникнення (penetration testing), аналіз журналів сервера та перевірку конфігурацій.

Особливо важливо враховувати людський фактор. Навчання розробників основам безпеки, обізнаність користувачів про загрози фішингу та використання надійних паролів — це важливі елементи стратегії захисту.

Комбінація різних методів захисту дозволяє створювати стійкі до атак web-застосунки, які відповідають сучасним вимогам безпеки. Однак їхнє впровадження потребує ретельного планування, врахування специфіки застосунку та дотримання принципів "захисту в глибину", коли кожен компонент системи підсилює загальну безпеку.

Для порівняльної характеристики зобразимо таблицю 2.1.

Таблиця 2.1

Порівняння методів захисту web-застосунків

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
CSRF Tokens	Не захищає, оскільки ця методика призначена для перевірки легітимності запитів, а не для контролю виконання скриптів на стороні клієнта.	Захищає, створюючи унікальний токен для кожної сесії або запиту, що дозволяє відслідковувати та запобігати підробленим запитам.	Не впливає, оскільки не запобігає завантаженню сайту в iframe.	Вимагає додаткової реалізації на серверній стороні, зокрема перевірки токенів для кожного запиту.	Може бути складною у випадках інтеграції з багатодоменними додатками або при використанні AJAX-запитів.
Content Security Policy	Захищає, дозволяючи визначити дозволені джерела контенту, що запобігає виконанню шкідливого JavaScript, який завантажується з неавторизованих джерел.	Не захищає, оскільки CSP не відстежує та не перевіряє легітимність запитів між різними сайтами.	Частково захищає, якщо налаштувати політику для блокування небажаних iframe або дозволяти їх лише з авторизованих джерел.	Впровадження може бути складним через необхідність налаштування політик, особливо у великих проєктах з численними залежностями.	Дуже гнучка завдяки можливості налаштовувати політики під потреби конкретного застосунку.

Продовження таблиці 2.1

Порівняння методів захисту web-застосунків

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
X-Frame-Options	Не захищає, оскільки ця заголовок відповідає лише за заборону відображення сторінки у фреймах, але не запобігає виконанню скриптів.	Не захищає, оскільки не працює з токенами чи легітимністю запитів.	Захищає, дозволяючи заборонити завантаження сторінки у фрейм, що запобігає атакам типу Clickjacking.	Дуже проста у впровадженні, оскільки вимагає лише додавання одного HTTP-заголовка до відповіді сервера.	Мінімальна гнучкість, оскільки дозволяє лише кілька статичних варіантів конфігурації.
Input Validation	Захищає, перевіряючи та очищаючи всі вхідні дані, що запобігає виконанню шкідливого JavaScript, який може бути введений через форми або запити.	Не захищає, оскільки не перевіряє легітимність запитів між сайтами.	Не впливає, оскільки не стосується завантаження сайту в iframe.	Залежить від рівня реалізації: може варіюватися від простого фільтрування даних до складних регулярних виразів і бібліотек.	Гнучка, оскільки дозволяє налаштувати валідацію під специфіку застосунку, але потребує ретельного налаштування для уникнення помилок.

2.3 Математична модель методів захисту web-застосунків

Математичні моделі є важливим інструментом для абстракції та формалізації складних процесів у різних галузях, включаючи кібербезпеку. Web-застосунки є вразливими до різноманітних атак, таких як Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Clickjacking, SQL-ін'єкції, а також до багатьох інших типів загроз. Математичні моделі дозволяють оцінити ймовірність успіху атак, а також ефективність застосованих захисних механізмів, таких як CSRF-токени, політика безпеки вмісту (CSP), захист від Clickjacking, та валідація введення.

У цьому контексті математичні моделі можуть допомогти не тільки оцінити ризики та ймовірності атак, але й забезпечити об'єктивний підхід до аналізу і покращення рівня безпеки web-застосунків. Вони дозволяють здійснювати прогнозування, тестування різних сценаріїв атак і на основі цього формулювати ефективні стратегії захисту.

Математична модель оцінки ймовірності успіху атаки дозволяє визначити, наскільки ймовірно, що система буде вразливою до певної атаки за умов застосування конкретних механізмів захисту. Це дає змогу розробникам приймати обґрунтовані рішення про вибір захисних механізмів на основі ймовірності того, що атака буде успішною.

XSS-атаки є одними з найпоширеніших і можуть мати серйозні наслідки, оскільки дозволяють зловмисникам виконувати JavaScript-код у браузерах користувачів. Це може призвести до викрадення конфіденційних даних або виконання несанкціонованих дій від імені користувача. Оскільки XSS-атаки безпосередньо пов'язані з обробкою введення користувачем на web-сторінках, ефективний механізм захисту полягає у валідному очищенні введених даних і коректному застосуванні політики CSP.

Математична модель для оцінки ймовірності успіху XSS-атаки може виглядати наступним чином:

$$P_{XSS} = \frac{\text{Кількість вразливиї точок введеня}}{\text{Загальна кількість точок введеня на сторінці}}, \quad (2.2)$$

де P_{XXS} – ймовірність успішної XSS-атаки.

Кількість вразливих точок введення: кількість полів введення, які не очищають дані.

Загальна кількість точок введення на сторінці: усі форми та інші поля, які приймають дані від користувача.

Чим менше вразливих точок введення, тим нижча ймовірність атаки. Для зниження ризику необхідно використовувати очищення даних та політику CSP.

CSRF-атаки націлені на те, щоб примусити користувача виконати небажану дію на сервері, використовуючи його сесію. Для запобігання таким атакам використовуються CSRF-токени, які додаються до кожного запиту, що змінює стан на сервері. Математична модель для захисту від CSRF-атак включає ймовірність того, що зловмисник зможе підробити CSRF-токен і надіслати фальшивий запит від імені користувача:

$$P_{CSRF} = \frac{\text{Кількість спроб підробки токенів}}{\text{Загальна кількість запитів}}. \quad (2.3)$$

Модель враховує такі параметри, як складність підробки токенів, механізми їх оновлення та перевірки на сервері.

Політика безпеки вмісту (CSP) дозволяє контролювати, які ресурси можна завантажувати на web-сторінці, що допомагає блокувати потенційно небезпечні скрипти або стилі, які можуть бути використані для XSS-атак. Математична модель для оцінки ефективності CSP може виглядати так:

$$P_{CSP} = \frac{\text{Кількість небезпечних скриптів,що не були заблоковані } CSP}{\text{Загальна кількість спроб завантаження ресурсів}}. \quad (2.4)$$

Ця модель дає змогу оцінити, наскільки ефективно політика CSP зупиняє несанкціоновані спроби завантаження шкідливих ресурсів.

Clickjacking — це тип атаки, коли зловмисник використовує невидимі або неактивні елементи web-сторінки для того, щоб змусити користувача натискати на небажані елементи. Захист від Clickjacking здійснюється за допомогою заголовка

X-Frame-Options, який забороняє вставку сторінки в iframe на сторонніх сайтах.
Математична модель для захисту від Clickjacking може включати оцінку ймовірності того, що web-сторінка буде вставлена в iframe і викликана на сторонньому сайті:

$$P_{Clickjacking} = \frac{\text{Кількість спроб вставити сторінку в } iframe}{\text{Загальна кількість запитів на сторінку}}. \quad (2.5)$$

Прогнозування та аналіз є ключовими аспектами у вивченні ефективності методів захисту. Завдяки математичним моделям можна передбачити, наскільки ефективним буде конкретний захист у різних умовах, враховуючи ймовірність атаки, тип застосовуваних механізмів захисту, рівень їх складності та багато інших чинників.

Комплексні моделі дозволяють об'єднати різні параметри і визначити, який набір механізмів захисту забезпечить найвищий рівень безпеки при мінімальних витратах ресурсів. Вони також можуть враховувати взаємодію між різними методами безпеки та прогнозувати їхній сумарний ефект.

Коли йдеться про розробку стратегій безпеки для web-застосунків, математичні моделі можуть поєднувати кілька методів захисту в єдину комплексну стратегію. Такі моделі оцінюють взаємодію різних елементів захисту та прогнозують їхній вплив на загальний рівень безпеки системи. Наприклад, застосування як CSRF-токенів, так і політики CSP може значно знизити ймовірність успішних атак, якщо обидва методи використовуються разом.
Математична модель для оцінки комплексних стратегій захисту може виглядати так:

$$P_{complex} = P_{XSS} + P_{CSRF} + P_{CSP} + P_{Clickjacking}. \quad (2.6)$$

Ця функція враховує взаємодію різних компонентів захисту і дозволяє визначити загальний рівень безпеки web-застосунку.

2.3.1 Математична модель токенів CSRF

Токен T_s — це унікальний криптографічно стійкий рядок, який генерується сервером і використовується для ідентифікації легітимності запиту. Він має наступні властивості:

- $T_s \in \{0,1\}^n$, де n — кількість бітів токена (звичайно n=128 або n=256).
- Токен є унікальним для кожної сесії користувача, що запобігає повторному використанню токенів у нових сесіях.

Токен створюється сервером за допомогою криптографічної функції G(U,S,t) де:

- G— криптографічно стійка функція, яка може бути реалізована через алгоритми хешування (SHA-256) або HMAC (Hash-based Message Authentication Code).
- U — унікальний ідентифікатор сесії користувача (наприклад, ID сесії або ID користувача).
- S — секретний ключ сервера, який гарантує, що токен не може бути підроблено.
- t — мітка часу для захисту від повторного використання токена.

Коли користувач автентифікується, сервер генерує токен для поточної сесії:

$$T_s = G(U, S, t) \tag{2.7}$$

Цей токен зберігається сервером і відправляється клієнту як частина відповіді (у cookie або в прихованому полі форми). Коли клієнт надсилає запит, токен T_s додається до запиту (наприклад, у заголовку HTTP або як параметр POST-запиту). Сервер отримує запит і перевіряє токен за формулою:

$$Valid(T_s) = \begin{cases} True, \text{ якщо } T_s = T_s \\ False, \text{ інакше} \end{cases} \tag{2.8}$$

Якщо токен валідний, сервер виконує запит. В іншому випадку запит

відхиляється.

Якщо токен криптографічно стійкий і довжина n достатньо велика (наприклад, n=128), то ймовірність вгадування токена зловмисником описується формулою що практично виключає можливість успішної атаки методом підбору.

$$P_{\text{успіху}} = \frac{1}{2^n} \tag{2.9}$$

Використання мітки часу t забезпечує обмежений період дії токена. Якщо мітка часу виходить за дозволені межі, токен стає недійсним.

Переваги методу:

- Забезпечує захист від міжсайтових запитів (CSRF).
- Не залежить від конфігурації браузера.
- Простий для інтеграції у сучасні web-фреймворки.

Обмеження та недоліки:

1. Додаткові ресурси на перевірку:
   Сервер повинен зберігати токени для кожної сесії, що може створювати додаткове навантаження на серверну пам’ять.
2. Складнощі з міждоменними запитами:
   Якщо web-застосунок працює через декілька доменів, додавання токенів до запитів може потребувати складнішої конфігурації.
3. Можливість компрометації:
   Якщо токен передається через незахищений канал (без HTTPS), його можуть перехопити.

2.3.2 Математічна модель CSP

Content Security Policy (CSP) — це потужний інструмент захисту web-сайтів від різноманітних атак, зокрема від XSS-атак, таких як міжсайтові скриптові атаки. CSP дозволяє обмежити джерела, з яких браузер може завантажувати ресурси, що значно зменшує можливість виконання шкідливого коду.
Математичне моделювання CSP ґрунтується на концепції визначення дозволених

джерел для ресурсів, таких як скрипти, стилі, зображення та інші. Політика CSP містить набір директив, які описують джерела ресурсів для різних типів контенту, зокрема для JavaScript, стилів, зображень, шрифтів, медіа-файлів тощо.

CSP зазвичай реалізується через HTTP-заголовок або мета-тег на web-сторінці. Кожна директива в політиці визначає дозволені джерела для певних типів ресурсів. Наприклад, директива script-src контролює, з яких джерел можна завантажувати JavaScript-коди.

Формально, політика CSP може бути записана як набір правил, де кожне правило визначає дозволене джерело для певного типу ресурсу:
CSP = { default-src, script-src, style-src, img-src, … }

- default-src: Загальні джерела для всіх типів ресурсів.
- script-src: Джерела для JavaScript-скриптів.
- style-src: Джерела для CSS-стилів.
- img-src: Джерела для зображень.
- object-src: Джерела для вбудованих об'єктів, таких як Flash.
- font-src: Джерела для шрифтів.

Кожен запит на ресурс, що надходить від браузера до сервера, перевіряється на відповідність політиці CSP. Для цього порівнюються джерела запиту з тими, що визначені у політиці. Якщо джерело ресурсу входить у дозволений список, запит дозволяється, якщо ні — блокується.

Нехай C — множина всіх можливих джерел для певного ресурсу, наприклад, для скриптів. Для кожної директиви CSP визначено підмножину дозволених джерел $C_{allowed} \subseteq C$ і кожен запит на ресурс перевіряється на належність до цієї підмножини.
Запит на ресурс можна описати як пару:

$$q = (r, s) \tag{2.10}$$

де r— це тип ресурсу (наприклад, скрипт, стиль), а s — це джерело, з якого цей ресурс завантажується.

Математично перевірка запиту qqq на відповідність політиці CSP може бути записана як:

$$valid(q) = \begin{cases} True, \text{ якщо } s \in C_{allowed}(r) \\ False, \text{ якщо } s \notin C_{allowed}(r) \end{cases} \quad (2.11)$$

де $C_{allowed}(r)$ — це множина дозволених джерел для типу ресурсу r.

Математична модель атаки через XSS в контексті CSP зводиться до спроби зловмисника виконати шкідливий скрипт, завантажуючи його з джерела, яке не дозволено політикою CSP. Імовірність успішної атаки можна оцінити через кількість можливих джерел для шкідливих скриптів, яких немає в політиці CSP.

Нехай S_{attack} — множина шкідливих джерел, з яких зловмисник може спробувати завантажити шкідливі скрипти. Тоді ймовірність успішної атаки визначається як:

$$P_{\text{атаки}} = \frac{[S_{attack}]}{[C]} \quad (2.12)$$

де C — це множина всіх можливих джерел для цього ресурсу, а $[S_{attack}]$— кількість джерел, які є потенційно шкідливими.

Зниження цієї ймовірності до мінімуму досягається через жорстке визначення дозволених джерел у політиці CSP.

Розробник або системний адміністратор налаштовує політику CSP, визначаючи дозволені джерела для кожного типу ресурсу. Це може бути зроблено через HTTP заголовки або мета-теги на сторінці.

Сервер надсилає цю політику через заголовок HTTP Content-Security-Policy. Це означає, що всі ресурси, які будуть завантажуватися з web-сторінки, перевірятимуться на відповідність політиці CSP.

Кожен запит на ресурс, що надходить від браузера, перевіряється на відповідність дозволеним джерелам. Браузер порівнює джерело ресурсу з тим, що вказано в політиці CSP.

Якщо запит на ресурс виходить за межі дозволених джерел, браузер блокує

цей запит і не завантажує ресурс. Користувач або розробник може отримати повідомлення про блокування через консоль або звіт.

Якщо використовується режим звітності (наприклад, через директиву report-uri), браузер надсилає звіти про порушення політики CSP на вказану адресу.

Переваги CSP

1. Захист від XSS атак:
   CSP значно знижує ймовірність успішних XSS атак, обмежуючи можливість завантаження та виконання шкідливих скриптів з ненадійних джерел.
2. Гнучкість і контроль:
   Розробники можуть налаштовувати політику так, щоб вона задовольняла вимоги конкретного web-застосунку. Це дозволяє детально контролювати, які ресурси можуть бути завантажені, і запобігати виконанню шкідливих кодів.
3. Сучасна підтримка браузерів:
   CSP підтримується майже всіма сучасними браузерами, що дозволяє забезпечити надійний захист без необхідності додаткових плагінів або інструментів.

Недоліки CSP

1. Складність налаштування:
   Налаштування політики CSP може бути складним, особливо для великих web-застосунків, де потрібно врахувати безліч різних джерел для різних типів ресурсів.
2. Невірне налаштування:
   Неправильне налаштування політики може призвести до блокування важливих ресурсів, що зіпсує функціональність web-застосунку.
3. Проблеми сумісності:
   CSP може бути несумісним з деякими старими бібліотеками або фреймворками, які не підтримують сучасні стандарти безпеки.

2.3.3 Математична модель X-Frame-Options

Заголовок X-Frame-Options є одним з основних методів захисту web-додатків від Clickjacking-атак. Цей захист дозволяє web-серверам визначати, чи можна відображати вміст їхніх сторінок у фреймах на сторонніх сайтах. Атака Clickjacking полягає в тому, що зловмисник вставляє web-сторінку в невидимий фрейм на своєму сайті і спокушає користувача взаємодіяти з нею, не знаючи, що він насправді взаємодіє з елементами іншого сайту.

Заголовок X-Frame-Options може бути налаштований з трьома можливими значеннями:

- DENY — забороняє відображення сторінки в будь-якому фреймі.
- SAMEORIGIN — дозволяє відображати сторінку в фреймі лише в тому випадку, якщо джерело сторінки та фрейма однакові.
- ALLOW-FROM uri — дозволяє відображати сторінку в фреймі лише з певного джерела, зазначеного в параметрі uri.

Математичне формулювання

Нехай є web-сторінка P, яку треба захистити від Clickjacking-атак. Сторінка може бути вставлена в фрейм на сторонньому web-сайті S. У цьому випадку ми маємо дві змінні:

- A=(P,S)— пара, де P — це сторінка, а S — це сайт, на якому ця сторінка буде відображена в фреймі.
- O={DENY,SAMEORIGIN,ALLOW-FROM}— множина можливих значень заголовка X-Frame-Options.

Для кожного варіанту заголовка можна сформулювати математичне правило перевірки на можливість відображення сторінки в фреймі:

DENY:

Політика повністю забороняє відображення сторінки у фреймі:

$$allow(P, S) = False. \quad (2.12)$$

Незалежно від сайту S, сторінка P не буде відображена в фреймі.

1. SAMEORIGIN:

$$allow(P,S) = \begin{cases} True, & \text{якщо домен } S = \text{ домен } P \\ False, & \text{якщо домен } S \neq \text{домен } P \end{cases} \quad (2.13)$$

Сторінка P може бути відображена в фреймі тільки якщо сайт S має той самий домен, що й сторінка P.

2. ALLOW-FROM

$$allow(P,S) = \begin{cases} True, & \text{якщо } S = uri \\ False, & \text{якщо } S \neq uri \end{cases} \quad (2.14)$$

3. Сторінка P може бути відображена тільки в тому випадку, якщо сайт S відповідає конкретному URI, зазначеному в параметрі заголовка.

1. Налаштування заголовка X-Frame-Options:
   Розробник налаштовує сервер для відправки відповідного заголовка X-Frame-Options залежно від бажаного рівня безпеки:
   - Якщо бажано повністю заборонити відображення в фреймах, встановлюється значення DENY.
   - Якщо дозволяється лише відображення на тому ж домені, встановлюється SAMEORIGIN.
   - Якщо є необхідність дозволити відображення тільки для конкретного сайту, вказується ALLOW-FROM.
2. Перевірка запиту на вбудовування в фрейм:
   Коли браузер намагається відобразити сторінку в фреймі, він перевіряє заголовок X-Frame-Options. Якщо політика заголовка не дозволяє вбудовування, браузер блокує цю операцію.
3. Блокування в разі порушення політики:
   Якщо політика X-Frame-Options забороняє відображення сторінки в фреймі, браузер блокує завантаження сторінки в фрейм. Користувач може побачити повідомлення про помилку, залежно від браузера.

Математична модель атаки через Clickjacking

Для атакуючого важливо вставити цільову сторінку в невидимий фрейм на своєму сайті і приховати всі елементи, що роблять фрейм видимим. Імовірність успіху атаки через Clickjacking залежить від того, чи дозволено відображати цю сторінку в фреймах. Математично це можна виразити як C_{attack} — множина сайтів, які намагаються вставити цільову сторінку в фрейм. Ймовірність успіху атаки визначається як:

$$P_{\text{атаки}} = \frac{[C_{attack} \cap C_{allowed}]}{[C_{attack}]} \quad (2.15)$$

де $C_{allowed}$— це множина сайтів, для яких політика X-Frame-Options дозволяє вставку сторінки в фрейм.

Переваги заголовка X-Frame-Options:

1. Захист від Clickjacking:
   Основною перевагою є забезпечення захисту від Clickjacking-атак, що є одним із важливих методів для зловмисників для обману користувачів.
2. Простота впровадження:
   Встановлення заголовка X-Frame-Options є дуже простим процесом і не вимагає великих зусиль з боку розробників.
3. Широка підтримка браузерів:
   Більшість сучасних браузерів підтримують цей заголовок, що дозволяє забезпечити захист на широкому спектрі платформ.

Недоліки заголовка X-Frame-Options

1. Обмежена гнучкість:
   Заголовок X-Frame-Options має лише три можливі значення і не дозволяє вказати складніші умови для відображення в фреймах (наприклад, конкретні шляхи на сайті).
2. Проблеми з зовнішніми інтеграціями:
   Іноді необхідність відображати сайт в фреймі на іншому домені (наприклад, для вбудованих віджетів або партнерських сторінок) може призвести до

проблем з сумісністю, якщо для захисту використовується DENY або SAMEORIGIN.

3. Не підтримується в деяких старих браузерах:
   Хоча більшість сучасних браузерів підтримують цей заголовок, деякі старі браузери можуть не підтримувати або неправильно обробляти його.

2.3.4 Математична модель Input Validation

Валідація введення є основним методом захисту web-застосунків від різноманітних атак, таких як SQL-ін'єкції, XSS-атаки та зловживання з недійсними або шкідливими даними. Процес валідації передбачає перевірку всіх даних, які надходять від користувача, на відповідність певним вимогам або шаблонам.
Основною метою валідації введення є гарантування того, що дані, які користувач надає web-застосунку, відповідають формату, типу та обмеженням, що є допустимими для системи.

Нехай є система S, яка приймає вхідні дані від користувача D. Дані D можуть бути введені через різні типи полів, такі як текстові поля, перевірка електронної пошти, файли та інші форми вводу. Валідація введення може бути представлена як функція:

$$V(D) = \begin{cases} True, & \text{якщо } D \text{ є доступними даними} \\ False, & \text{якщо } D \text{ не відповідає вимогам} \end{cases}$$

де V(D)— функція валідації, яка повертає True, якщо введені дані D є допустимими, і False, якщо вони не відповідають вимогам.
Ключовими аспектами валідації є:

- Тип даних T: чи відповідає введене значення типу, наприклад, чи є це числом або рядком.
- Довжина L: перевірка, чи введене значення має правильну кількість символів.
- Шаблон P: перевірка, чи відповідає введене значення заданому шаблону (наприклад, для адрес електронної пошти).

- Діапазон R: перевірка, чи знаходиться введене значення в межах допустимого діапазону, наприклад, для віку чи числа.

Таким чином, функція валідації може виглядати так:

$$V(D) = \begin{cases} True, & \text{якщо } D \in T, L \in R, D \text{ відповідає шаблону } P \\ False, & \text{якщо хоча б она умова не виконується} \end{cases}$$

1. Для введення електронної пошти:
    - Тип T=email
    - Шаблон P=rfc5322 (формат електронної пошти)
    - Довжина L обмежена максимальним числом символів для email-адреси.
2. Для введення віку:
    - Тип T=integer
    - Діапазон R=[0,120]
    - Введене значення має бути цілим числом і знаходитися в межах від 0 до 120.

SQL-ін'єкції є однією з найбільш небезпечних атак, яка використовує уразливості у введенні даних для виконання небезпечних SQL-запитів. Математично SQL-ін'єкція може бути виражена як перехід від простого запиту до небезпечного запиту через введення шкідливого коду.

Нехай є запит:

Q = "SELECT * FROM users WHERE username = '" + D + "';";

де D — це введене користувачем ім'я користувача. Якщо введені дані містять SQL-код, такий як:

D="′OR1=1−−"

то запит стає:

Q = "SELECT * FROM users WHERE username = " OR 1=1 --';";

Цей запит завжди повертає всі записи з таблиці, оскільки умова 1=1 є завжди істинною. Така атака може привести до витоку даних або модифікації бази даних. Щоб запобігти таким атакам, необхідно застосовувати валідацію введення для

забезпечення того, щоб введені дані не містили небезпечних SQL-операторів, таких як ', --, OR, ; тощо.

1. Фільтрація та екранування введених даних: Перед використанням введених даних у SQL-запитах, варто застосовувати фільтрацію для видалення або екранування небезпечних символів, таких як ', ", ;, --. Для SQL-запитів можна використовувати підготовлені запити (prepared statements), що автоматично обробляють введення.
2. Перевірка формату: Для полів, де необхідно дотримуватись певного формату (наприклад, електронна пошта, телефон), застосовуються регулярні вирази для перевірки на відповідність цьому формату.
3. Визначення діапазонів та довжини: Валідація введених чисел на відповідність діапазону або перевірка, чи входить введена дата в дозволений діапазон (наприклад, перевірка віку на адекватність).

Переваги валідації введення:

1. Запобігання атакам: Валідація введення є ефективним методом для запобігання таких атак, як SQL-ін'єкція, XSS та інші, що можуть використати введення користувача для виконання зловмисних дій.
2. Безпека даних: Валідація дозволяє знизити ризики потрапляння шкідливих або неправдивих даних у систему, що може призвести до порушення безпеки або некоректної роботи додатку.
3. Підвищення якості даних: Перевірка введених даних забезпечує їх правильність і відповідність очікуваним вимогам, що покращує загальну якість і точність обробки даних.

Недоліки валідації введення:

1. Складність налаштування: Валідація введення може бути складною для налаштування, особливо для складних типів даних, де потрібно визначити точні формати або шаблони.
2. Затримка при обробці: Якщо валідація даних виконується на кожному етапі введення, це може призвести до додаткових затримок у роботі системи, особливо при обробці великих обсягів даних.
3. Не повна захищеність: Валідація введення може бути недостатньою, якщо не застосовуються інші методи захисту, такі як підготовлені запити для SQL.

3 РОЗРОБКА МЕТОДУ S.A.F.E

3.1 Опис розробки методу

У сучасному світі безпека web-додатків є однією з найголовніших проблем, з якими стикаються розробники. Постійні вдосконалення методів атак та нові вразливості, що виникають з розвитком технологій, вимагають від фахівців безпеки розробки нових, більш ефективних методів захисту. Однією з важливих задач є захист від атак на рівні фронтенду, де зловмисники можуть використовувати вразливості для проникнення в систему, викрадення даних або навіть виконання шкідливих дій. У цьому контексті, створення методики, яка б об'єднувала найкращі практики захисту від відомих атак, є надзвичайно актуальним.

У цьому розділі буде детально описано розроблений метод S.A.F.E для захисту web-додатків, який дозволяє ефективно протистояти різноманітним атакам на фронтенді. Описано основні етапи розробки цього методу, включаючи вибір програмних засобів та інструментів для реалізації, а також особливості його інтеграції з існуючими web-додатками. Зокрема, метод S.A.F.E орієнтований на захист від таких атак, як XSS, CSRF, clickjacking та інших типових загроз.

У розділі також буде наведено порівняння з іншими методами захисту, надано діаграми класів, блок-схеми, а також проведений аналіз ефективності S.A.F.E у порівнянні з традиційними методами. Зокрема, буде акцентовано увагу на перевагах, які надає нова методика, порівняно з існуючими підходами, а також на її гнучкості та можливостях адаптації до різних типів атак.

Для оцінки ефективності методики буде використано ряд інструментів, таких як OWASP ZAP, що дозволяє виявляти вразливості в реальному часі під час тестування web-додатків. Скриншоти із застосування цих інструментів будуть продемонстровані для кращого розуміння процесу перевірки безпеки.

Метод S.A.F.E (Secure Authentication & Frontend Environment) був розроблений як комплексний підхід до захисту web-додатків від найбільш

поширених атак, таких як XSS, CSRF та Clickjacking. Враховуючи, що ці вразливості можуть суттєво вплинути на безпеку користувачів, метод має на меті комбінувати декілька стратегій для досягнення більш високого рівня безпеки.

Основною метою методу є захист від атак, які можуть бути використані для викрадення даних, фальшування запитів або маніпуляцій із вікнами браузера. Враховуючи різноманітність таких загроз, важливо застосовувати багатошаровий захист, при якому кожен з елементів працює в тандемі з іншими.

3.1.1 Ідея методу S.A.F.E

Ідея методу виникла з необхідності покращити рівень безпеки web-додатків за допомогою поєднання різних технологій і стратегій. Кожен із традиційних методів захисту (захист від XSS, CSRF, Clickjacking) має свої обмеження, тому було вирішено створити метод, який поєднує ці стратегії в єдину цілісну систему. Такий підхід дозволяє створити додатки з високим рівнем захисту за рахунок використання комбінованих методів, що допомагають запобігти атакуючим стратегіям, які можуть бути недостатньо ефективними самі по собі.

Захист від XSS забезпечується за допомогою валідації введених даних і коректного ескейпінгу. Для захисту від CSRF впроваджується використання токенів, що ускладнює виконання шкідливих запитів від імені користувача. А для запобігання атак Clickjacking застосовується обмеження відображення контенту в фреймах через заголовки, такі як X-Frame-Options.

3.1.2 Аналіз існуючих методів захисту

У ході розробки методу S.A.F.E було проведено аналіз наявних методів захисту від атак, які ми збираємося покривати. Розглянемо кожен із них окремо:

1. XSS (Cross-Site Scripting): Одна з найбільш поширених вразливостей у web-додатках. Традиційні методи захисту від XSS включають валідацію введених даних, ескейпінг та використання політики контенту (Content Security Policy). Проте ці методи не завжди ефективні, особливо при використанні складних технік ін'єкцій, таких як DOM-based XSS. Саме тому в методі S.A.F.E

застосовано додаткові перевірки на рівні клієнтської та серверної частини.

2. CSRF (Cross-Site Request Forgery): Захист від CSRF зазвичай досягається через використання токенів, які додаються до кожного запиту для підтвердження його легітимності. Однак цей підхід може бути уразливим при недостатньому контролі за правильністю генерування та перевірки токенів. Метод S.A.F.E забезпечує більш надійний захист за допомогою додаткових механізмів перевірки, включаючи ідентифікацію сесій на сервері.
3. Clickjacking: Захист від Clickjacking реалізується через обмеження вбудовування web-сторінок у фрейми за допомогою заголовків HTTP (наприклад, X-Frame-Options або Content-Security-Policy). Проте це рішення не завжди працює, якщо цільова web-сторінка є частиною більш складної екосистеми, де вбудовування контенту є необхідним для роботи. В методі S.A.F.E передбачено ретельне тестування всіх сторінок на можливість вбудовування та додаткові обмеження для захисту від таких атак.

3.1.3 Розробка архітектури методу S.A.F.E

Архітектура методу S.A.F.E побудована на принципах багатошарового захисту. Це означає, що різні рівні додатку (клієнт, сервер, з'єднання) використовують різні методи для захисту від загроз. Розглянемо основні компоненти:

1. Клієнтська частина: На цьому рівні відбувається валідація введених даних, перевірка на наявність потенційно небезпечних скриптів та елементи для запобігання атак на рівні браузера. Наприклад, для захисту від XSS на клієнтському рівні може застосовуватись ескейпінг введених даних та додаткові обробники подій для відслідковування і блокування шкідливих спроб.
2. Серверна частина: Тут реалізується валідація запитів та перевірка токенів для захисту від CSRF, а також обробка запитів за допомогою безпечних методів, таких як перевірка на наявність шкідливих запитів і підключення до відповідних API для аутентифікації.

3. З'єднання: Всі з'єднання з web-додатком мають бути захищені через HTTPS для запобігання перехопленню даних. Крім того, для додаткової безпеки передбачається використання сесій, які зберігаються у безпечних куки з прапорцем HttpOnly.

3.1.4 Реалізація методу S.A.F.E

Під час розробки методу використовувались сучасні технології та інструменти, що дозволяють забезпечити високий рівень безпеки. Для реалізації методу були використані:

1. JavaScript для клієнтської частини.
2. Node.js з використанням бібліотек для валідації запитів, таких як express-validator та jsonwebtoken для реалізації захисту від CSRF.
3. Content Security Policy (CSP) для додаткового захисту від XSS на стороні клієнта.
4. OWASP ZAP для тестування безпеки web-додатка та виявлення потенційних вразливостей.

Ці інструменти дозволяють забезпечити всебічний захист, що охоплює як захист даних, так і захист користувачів від атак через web-додатки.

3.1.5 Тестування та вдосконалення методу

Після розробки методу S.A.F.E було проведено тестування на наявність вразливостей. Зокрема, за допомогою OWASP ZAP було перевірено додаток на наявність XSS, CSRF та Clickjacking атак. У процесі тестування виявлені слабкі місця в реалізації деяких механізмів захисту, які були виправлені за допомогою додаткових перевірок та покращень у коді. Це дозволило досягти більш високого рівня безпеки та знизити ймовірність успішної атаки.

3.2 Використані програмні засоби

У розробці методу S.A.F.E для забезпечення захисту web-додатків від атак,

таких як XSS, CSRF, Clickjacking, а також для забезпечення загальної безпеки, було використано широкий набір інструментів, бібліотек та фреймворків. Ось детальний опис основних програмних засобів, а також пояснення, чому саме ці інструменти були вибрані для конкретних завдань.

3.2.1 Мова програмування: JavaScript

Для розробки методу S.A.F.E було обрано JavaScript як основну мову програмування. Це пояснюється тим, що JavaScript є основною мовою для створення інтерактивних та динамічних web-додатків на стороні клієнта. Безпека web-додатків вимагає інтеграції захисних механізмів безпосередньо в логіку додатка, а JavaScript дає гнучкість для втілення всіх необхідних захистів, таких як валідація введених даних, контроль над виконанням скриптів, а також взаємодія з сервером для перевірки автентичності користувачів.

Основні переваги JavaScript у контексті безпеки:

- Можливість інтеграції з іншими бібліотеками безпеки: JavaScript дозволяє легко інтегрувати додаткові засоби для захисту від XSS, CSRF та Clickjacking. Бібліотеки, такі як Helmet.js для налаштування HTTP заголовків, дозволяють гнучко реагувати на зміни в запитах.
- Динамічне управління захистом: Оскільки JavaScript працює безпосередньо в браузері користувача, можна застосовувати стратегії захисту в реальному часі. Наприклад, перевірка введених користувачем даних або створення динамічних токенів для захисту від CSRF.
- Широка підтримка стандартів безпеки: JavaScript дозволяє інтегрувати стандартні методи безпеки, такі як Content Security Policy (CSP) для обмеження виконання небажаних скриптів, що критично для захисту від XSS.

Ця мова є стандартом для фронтенд-розробки, що дозволяє забезпечити високий рівень інтерактивності і одночасно включати в додаток захисні механізми.

3.2.2 Фреймворк React.js

React.js був вибраний для розробки інтерфейсу користувача завдяки своїй потужності та зручності в побудові масштабованих web-додатків. React дозволяє створювати компонентний підхід до розробки, що дає можливість інкапсулювати логіку безпеки в окремі компоненти, тим самим покращуючи підтримку та масштабованість.

Переваги використання React.js у розробці методу S.A.F.E:

- Компонентна структура: React дозволяє створювати компонентну архітектуру, що дає можливість інкапсулювати функціональність кожного елемента (наприклад, форму введення або механізм аутентифікації) і легко додавати додаткові шари безпеки в кожен компонент. Завдяки цьому можна застосовувати захисні стратегії на рівні кожного елементу інтерфейсу.
- Підтримка оновлень стану: Завдяки потужному механізму управління станом (наприклад, через React Hooks або Context API), можна зручно обробляти динамічні зміни, що є важливим для управління безпекою на рівні користувацького інтерфейсу.
- Легкість у впровадженні валідації: React дозволяє ефективно інтегрувати функції валідації введених даних, що критично для захисту від атак типу XSS. Наприклад, можна створювати власні компоненти для перевірки введених користувачем значень, щоб переконатися, що вони не містять шкідливих скриптів.
- Зручність інтеграції з іншими бібліотеками: React.js безперешкодно інтегрується з іншими інструментами безпеки, такими як Helmet.js для захисту HTTP-заголовками та react-router для безпечного управління маршрутизацією та доступом до сторінок.

Завдяки цим властивостям React забезпечує високий рівень гнучкості і безпеки на клієнтській стороні.

3.2.3 Firebase

Firebase був обраний для аутентифікації та зберігання даних користувачів. Firebase пропонує готові рішення для організації безпечної аутентифікації, що дозволяє швидко впровадити методи захисту та інтегрувати їх із фронтенд-частиною.

Переваги використання Firebase:

- Забезпечення безпеки аутентифікації: Firebase надає інструменти для захищеного зберігання паролів, використовуючи технології хешування та токенізації. Це дозволяє ефективно захищати дані користувачів і запобігати атакам, таким як CSRF.
- Підтримка різних методів аутентифікації: Firebase підтримує автентифікацію через різні канали, включаючи Google, Facebook, а також за допомогою email і пароля. Це дозволяє налаштувати різноманітні рівні доступу та захисту в залежності від потреб користувача.
- Автоматичне управління сесіями: Firebase автоматично керує сесіями користувачів, надаючи токени доступу, які можуть бути використані для автентифікації API-запитів. Це дозволяє уникнути помилок, пов'язаних з ручним управлінням сесіями, і забезпечує захист від атак типу Session Fixation.
- Інтеграція з іншими інструментами безпеки: Firebase дозволяє легко інтегрувати додаткові механізми безпеки на сервері, наприклад, для застосування CORS та CSP, що забезпечує захист від атак з різних джерел.

Firebase також дає можливість легко зберігати дані у реальному часі, що робить його ідеальним інструментом для побудови додатків з високим рівнем інтерактивності.

3.2.4 Хостинг: Vercel

Для хостингу web-додатка був обраний Vercel, оскільки ця платформа забезпечує високу продуктивність, зручну інтеграцію з Git, а також має вбудовані засоби для налаштування безпеки, що критично для захисту web-додатків.

Переваги використання Vercel:

- Швидкість і надійність: Vercel спеціалізується на хостингу додатків на основі JavaScript та забезпечує швидку доставку контенту з глобальних серверів, що дозволяє забезпечити високу доступність додатка для користувачів по всьому світу.
- Забезпечення безпеки на сервері: Платформа автоматично налаштовує шифрування HTTPS для всіх додатків, що дозволяє захистити дані від перехоплення під час їх передачі.
- Інтеграція з Git: Vercel дозволяє інтегрувати хостинг з репозиторіями Git, що спрощує процес деплойменту додатка та дозволяє легко налаштовувати автоматичні оновлення.
- Гнучкість налаштування CORS та інших заголовків: Vercel надає можливість налаштовувати різні HTTP-заголовки, що важливо для налаштування політики доступу та забезпечення захисту від атак з інших доменів (CORS).

Vercel також дозволяє ефективно працювати з інтерфейсами API, що робить його ідеальним вибором для хостингу React-додатків.

3.2.5 Програмні засоби для безпеки

Заголовки CSP були реалізовані для обмеження виконання шкідливих скриптів на стороні клієнта. CSP дозволяє визначати, які скрипти та ресурси можуть бути виконані або завантажені на сторінці, що є одним із найефективніших методів запобігання XSS-атакам.

CORS було налаштовано для обмеження доступу до ресурсів web-додатка з інших доменів, що дозволяє захистити від атак, коли зловмисник намагається викликати API додатка з іншого сайту.

JWT використовуються для безпечної передачі даних між клієнтом та сервером. Ці токени дозволяють перевірити автентичність запитів і захистити від CSRF-атак, оскільки токен не зберігається в cookie, що робить його менш уразливим до таких атак.

Вибір цих засобів обумовлений необхідністю забезпечити високий рівень

захисту на кожному етапі роботи web-додатка, від аутентифікації користувачів до обробки запитів між клієнтом і сервером.

3.3 Діаграма послідовності методу S.A.F.E

Діаграма послідовності показує взаємодію між кількома компонентами системи, які забезпечують захист від різноманітних атак, таких як XSS, CSRF, і Clickjacking. Ця діаграма охоплює процес від введення даних користувачем до їх обробки додатком та відповіді сервера з обробленими даними. У ній задіяні кілька ключових елементів: користувач, додаток, валідатор, генератор CSRF токенів, сервер, і політика безпеки контенту (CSP).

Крок 1: Користувач вводить дані через HTTPS.

У першому кроці процесу користувач вводить дані на web-сторінці через web-інтерфейс. (Рис 3.3) Це може бути будь-яка форма введення, наприклад, реєстраційна форма або форма для надсилання запиту. Усі дані передаються через протокол HTTPS, що гарантує захищене з'єднання, шифруючи передавані дані для запобігання перехопленню інформації сторонніми особами.

Крок 2: Додаток передає дані на перевірку валідації.

Після того, як користувач вводить дані, додаток отримує їх і передає до компонента Валідатор для перевірки на коректність. Це може включати перевірку формату введених даних (наприклад, адреса електронної пошти), перевірку обов'язкових полів або інші правила валідації.

Крок 3: Валідатор підтверджує валідність даних.

Валідатор обробляє введені дані і перевіряє їх відповідно до заданих правил. Якщо дані коректні, він повертає підтвердження про те, що дані є валідними, до додатку. Якщо ж дані не відповідають встановленим вимогам, система повинна надіслати користувачу повідомлення про помилку, щоб він міг виправити введені дані.

Крок 4: Генерація CSRF токену.

Після того як дані були підтверджені, додаток звертається до компонента Генератор CSRF токену (TokenGen). CSRF токен використовується для захисту від

атак типу міжсайтових підроблених запитів (CSRF). Це унікальний токен, який генерується на сервері і додається до кожного запиту, що надсилається від клієнта на сервер. Такий токен дозволяє серверу перевіряти, чи запит дійсно надійшов від легітимного користувача, чи він не був підроблений.

Крок 5: Надсилання CSRF токену додатку.

Генератор CSRF токену (TokenGen) повертає створений токен назад до додатку. Токен передається в запит, що надсилається від клієнта на сервер, що забезпечує додатковий рівень безпеки.

Крок 6: Надсилання запиту з CSRF токеном на сервер.

Додаток тепер надсилає запит на сервер, додаючи CSRF токен до цього запиту. Це забезпечує захист від CSRF атак, коли зловмисник намагається підробити запит від імені користувача. Усі ці запити також передаються через протокол HTTPS для захисту від перехоплення даних.

Крок 7: Перевірка CSRF токену на сервері.

На сервері Backend, перед тим як обробити запит, відбувається перевірка CSRF токену. Сервер перевіряє, чи правильний токен, надісланий в запиті, збігається з тим, що зберігається на сервері для поточного сеансу користувача. Якщо токен не співпадає або відсутній, запит буде відхилений, що є важливим механізмом захисту від CSRF атак.

Крок 8: Відповідь з даними.

Якщо перевірка CSRF токену пройшла успішно, сервер обробляє запит і відправляє відповідь назад до додатку. Це може бути будь-яка необхідна інформація, яка запитувалася користувачем, наприклад, результати пошуку, профіль користувача або інші дані.

Крок 9: Застосування Content Security Policy (CSP).

Перед тим як відобразити дані на екрані користувача, додаток застосовує Content Security Policy (CSP). CSP — це механізм безпеки, який дозволяє визначити, які ресурси (скрипти, стилі, зображення тощо) можна завантажувати на web-сторінці. CSP блокує завантаження небезпечних або неперевірених ресурсів, що знижує ймовірність виконання шкідливого коду, наприклад, в разі атаки XSS (міжсайтові скриптові атаки).

Крок 10: Блокування небезпечних ресурсів через CSP.

Політика CSP перевіряє ресурси, що намагаються завантажитися на сторінці, і заблокує ті з них, які не відповідають визначеним правилам безпеки. Це включає заблокування небезпечних скриптів або ресурсів, які могли бути підроблені або зловмисно змінені.

Крок 11: Показ результатів користувачеві.

Останній крок — це показ результатів користувачеві. Дані, отримані від сервера після перевірки CSRF токену та застосування CSP, відображаються на web-сторінці користувача. Оскільки всі дані передаються через HTTPS і застосовуються політики безпеки, це гарантує, що передача даних здійснюється в захищеному середовищі.

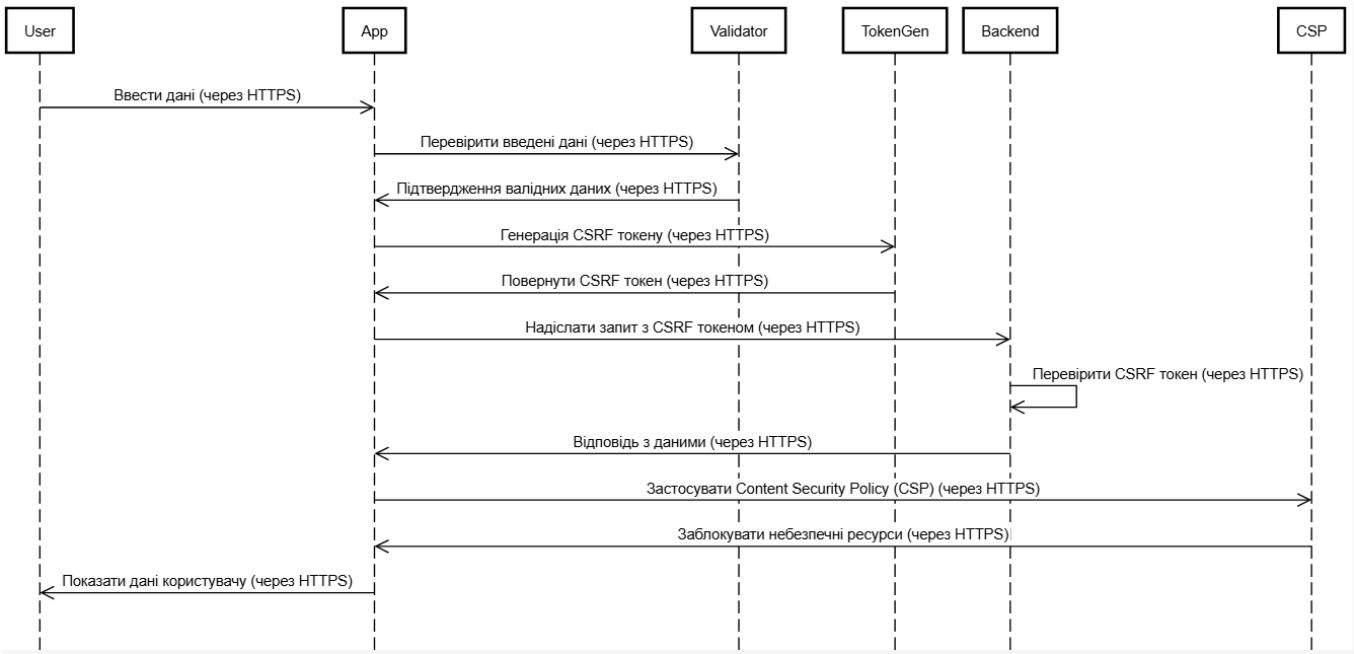


Рис 3.1 Діаграма послідовності методу S.A.F.E

3.4 Блок схема методу S.A.F.E

Першим етапом в процесі захисту є отримання запиту від користувача на сервер. Запит може бути різного типу, зокрема GET, POST або інші HTTP-запити, через яких користувач взаємодіє з web-додатком. Запит може містити дані, які необхідно обробити на сервері, наприклад, аутентифікаційні дані, запити на зміну

інформації чи відправлення форм. Запит приходить через HTTPS, що гарантує безпечну передачу даних між клієнтом і сервером.

На цьому етапі сервер перевіряє наявність CSRF токену в запиті. CSRF (Cross-Site Request Forgery) — це тип атаки, коли зловмисник може змусити користувача виконати небажану дію на web-сайті, де користувач вже аутентифікований, наприклад, відправити форму, змінити налаштування або здійснити платіж. Для захисту від таких атак використовуються CSRF токени — унікальні рядки, які додаються до кожного запиту, і які мають бути перевірені сервером.

Якщо CSRF токен відсутній або неправильний, система відхиляє запит, оскільки це може свідчити про спробу атаки. Відхилення запиту в такому випадку запобігає виконанню небажаних операцій. Якщо ж токен наявний і коректний, система переходить до наступного етапу перевірки.

XSS атака передбачає вставку шкідливого JavaScript-коду на web-сторінки, що дозволяє зловмисникам отримати доступ до даних користувачів, викрасти сесії, маніпулювати DOM або виконувати інші зловмисні дії. Для захисту від таких атак застосовуються механізми очищення введених даних, фільтрація та ескейпінг.

На цьому етапі система перевіряє, чи містить запит шкідливі скрипти. Якщо в запиті або вхідних даних виявлено шкідливий код, система блокує виконання цього коду, а також попереджає про спробу атаки. Цей етап важливий, щоб запобігти маніпуляціям із користувацьким інтерфейсом або витоку персональних даних через сторонні скрипти.

Якщо в запиті не виявлено XSS атак, система продовжує обробку запиту, перевіряючи його на інші потенційні загрози.

Clickjacking — це тип атаки, при якій зловмисник використовує невидимі або масковані елементи на web-сторінці, щоб змусити користувача натискати на елементи, які вони не мають наміру натискати. Наприклад, натискання на невидиму кнопку або посилання, яке може виконати небажану дію, таку як купівля товару чи виконання трансакції.

Для запобігання таким атакам використовується заголовок X-Frame-Options, який

запобігає відображенню web-сторінки в iframe на сторонніх сайтах. Якщо виявлено спробу завантажити web-сторінку в iframe, сервер блокує це і не дозволяє виконувати операції в умовах Clickjacking. Це запобігає можливим спробам зловмисників маніпулювати користувацьким інтерфейсом через фрейми.

Якщо загроза Clickjacking не виявлена, система продовжує обробку запиту без обмежень.

Після того як запит успішно пройшов перевірки на наявність атак CSRF, XSS та Clickjacking, система переходить до етапу відправки відповіді користувачу. Це може бути відповідь із даними, підтвердженням запиту або виконанням запитаної операції. Всі перевірки безпеки виконуються до моменту надання відповіді, що гарантує, що користувач отримує дані тільки після того, як запит був повністю перевірений на безпеку (рис. 3.2).

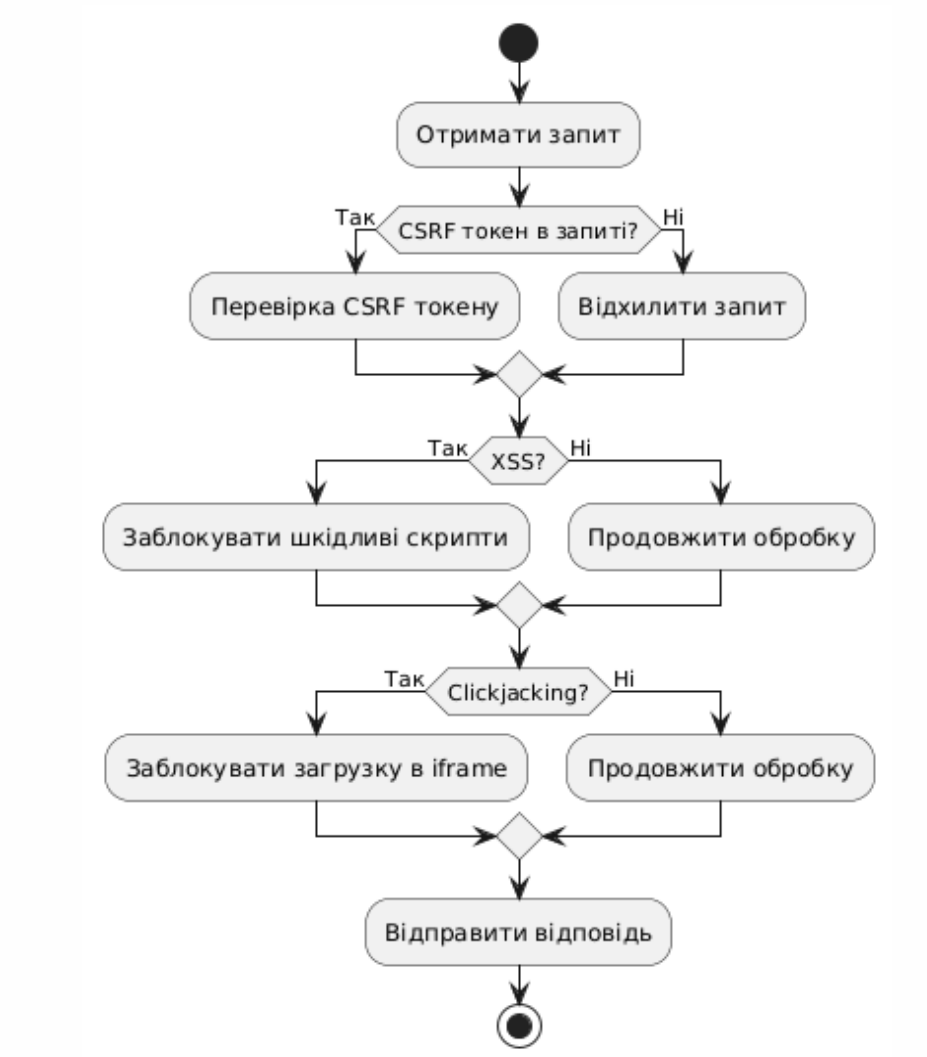


Рис 3.2 Блок схема методу S.A.F.E

Важливі моменти захисту:

1. CSRF токен: Для захисту від міжсайтових підроблених запитів важливо, щоб кожен запит, який змінює стан на сервері (наприклад, редагування профілю, відправка повідомлень, тощо), супроводжувався унікальним CSRF токеном, який генерується сервером і перевіряється при кожному запиті.
2. XSS захист: Очищення всіх введених даних допомагає уникнути виконання небезпечних скриптів. Цей етап критичний для запобігання атак, що можуть призвести до крадіжки сесій, витоку даних або маніпуляцій з інтерфейсом.
3. Clickjacking захист: Використання механізмів для запобігання завантаження web-сторінок в iframe допомагає уникнути маніпуляцій з інтерфейсом і забезпечує захист від підступних спроб натискання на елементи, які користувач не має наміру натискати.

3.5 Тестування та порівняння методу S.A.F.E з іншими методами

3.5.1 Методологія тестування з OWASP ZAP

1. Підготовка середовища.

На першому етапі тестування було налаштовано середовище для використання інструменту OWASP ZAP. Для цього було відкрито сам інтерфейс програми, і виконано налаштування проксі-сервера для перехоплення трафіку між браузером та сервером. Завдяки цьому процесу вдалося забезпечити можливість аналізу HTTP-запитів і відповідей, що дозволило детально вивчити взаємодію клієнта та сервера (рис. 3.3).

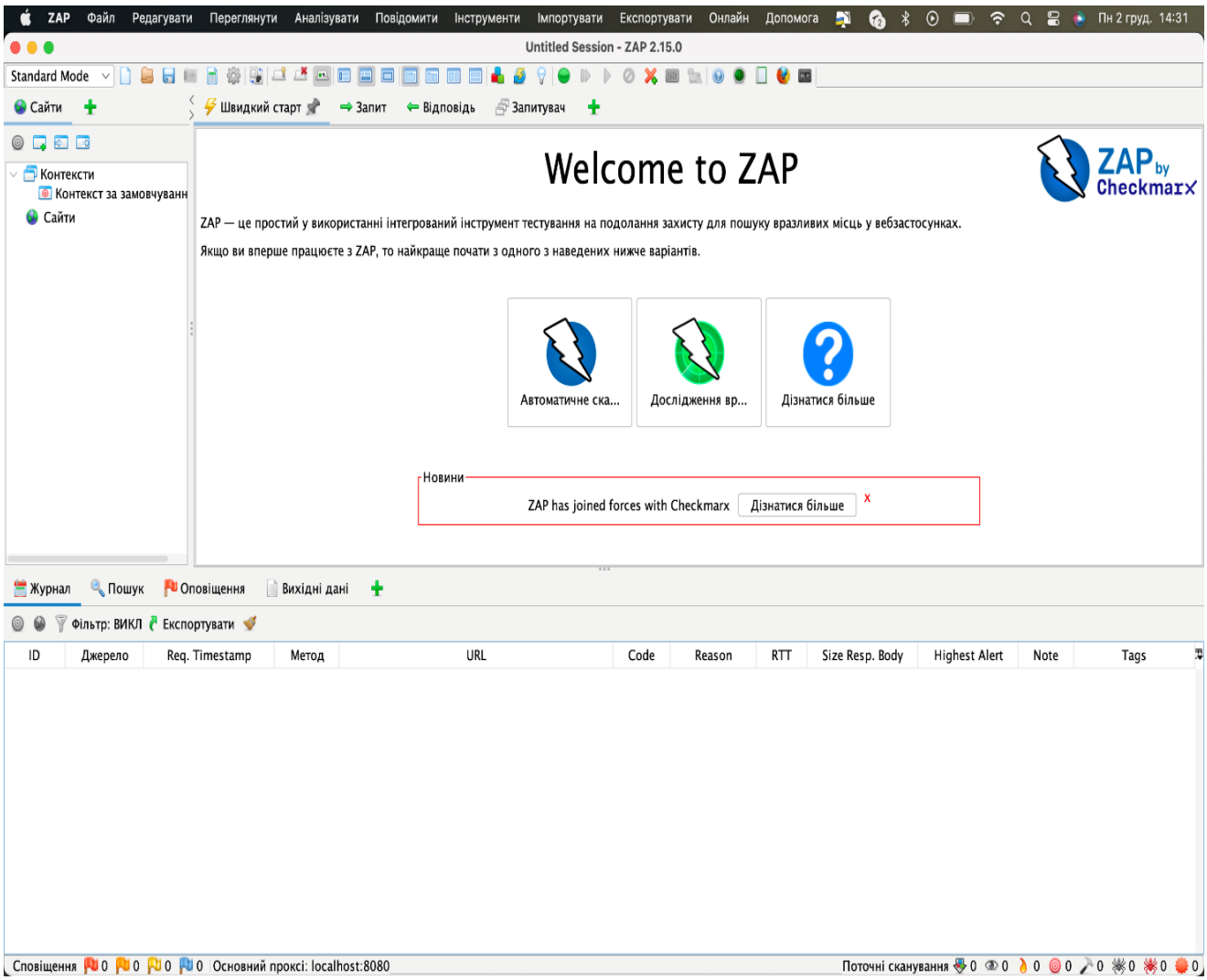


Рис. 3.3 Інтерфейс OWASP ZAP

1. Автоматичне сканування OWASP ZAP.

Після підготовки середовища було виконано автоматичне сканування web-додатку для виявлення потенційних вразливостей. OWASP ZAP активно сканує сторінки web-додатка, тестуючи їх на наявність загроз та недоліків у безпеці. Після завершення сканування згенерувався звіт, який включав усі виявлені ризики та рекомендації щодо їх усунення (рис. 3.4).

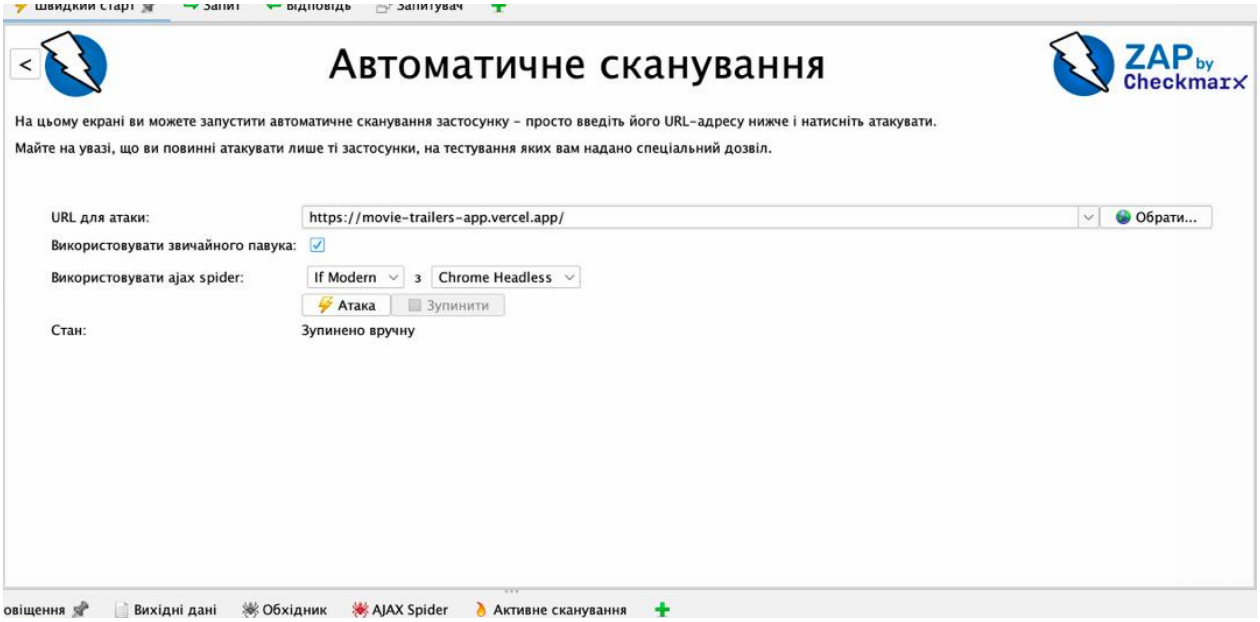


Рис. 3.4 Процес автоматичного сканування в OWASP ZAP.

У цьому етапі тестування було налаштовано проксі для перехоплення трафіку між браузером і сервером. За допомогою проксі стало можливим відслідковувати всі HTTP-запити та відповіді, а також змінювати дані запитів для перевірки роботи механізмів захисту web-додатку, таких як CSRF-токени або політика Content Security Policy (CSP) (рис. 3.5).

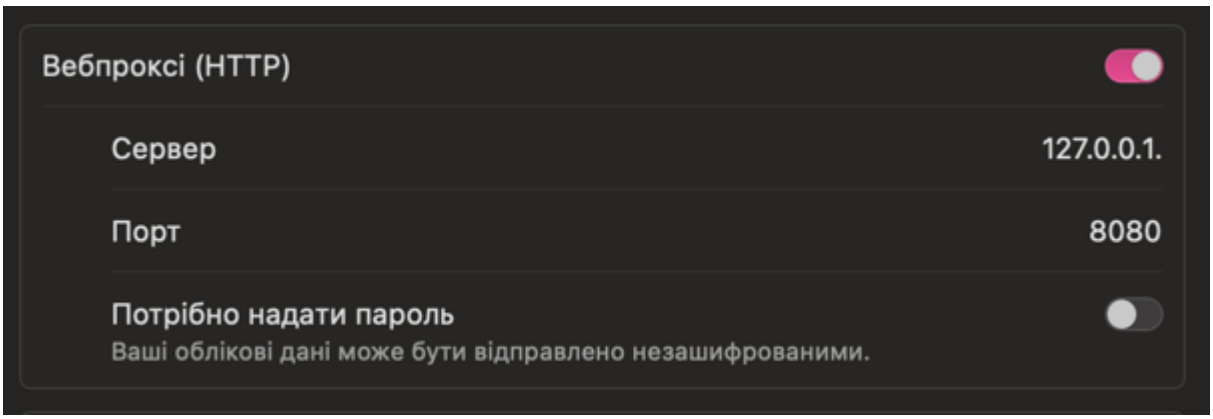


Рис. 3.5 Проксі для перехоплення трафіку.

Наступним кроком було перевірено налаштування порту для правильного з'єднання проксі-сервера з браузером. Цей етап є критично важливим для того, щоб переконатися, що трафік правильно маршрутизується через OWASP ZAP, і проксі може ефективно перехоплювати запити для подальшого тестування на безпеку.

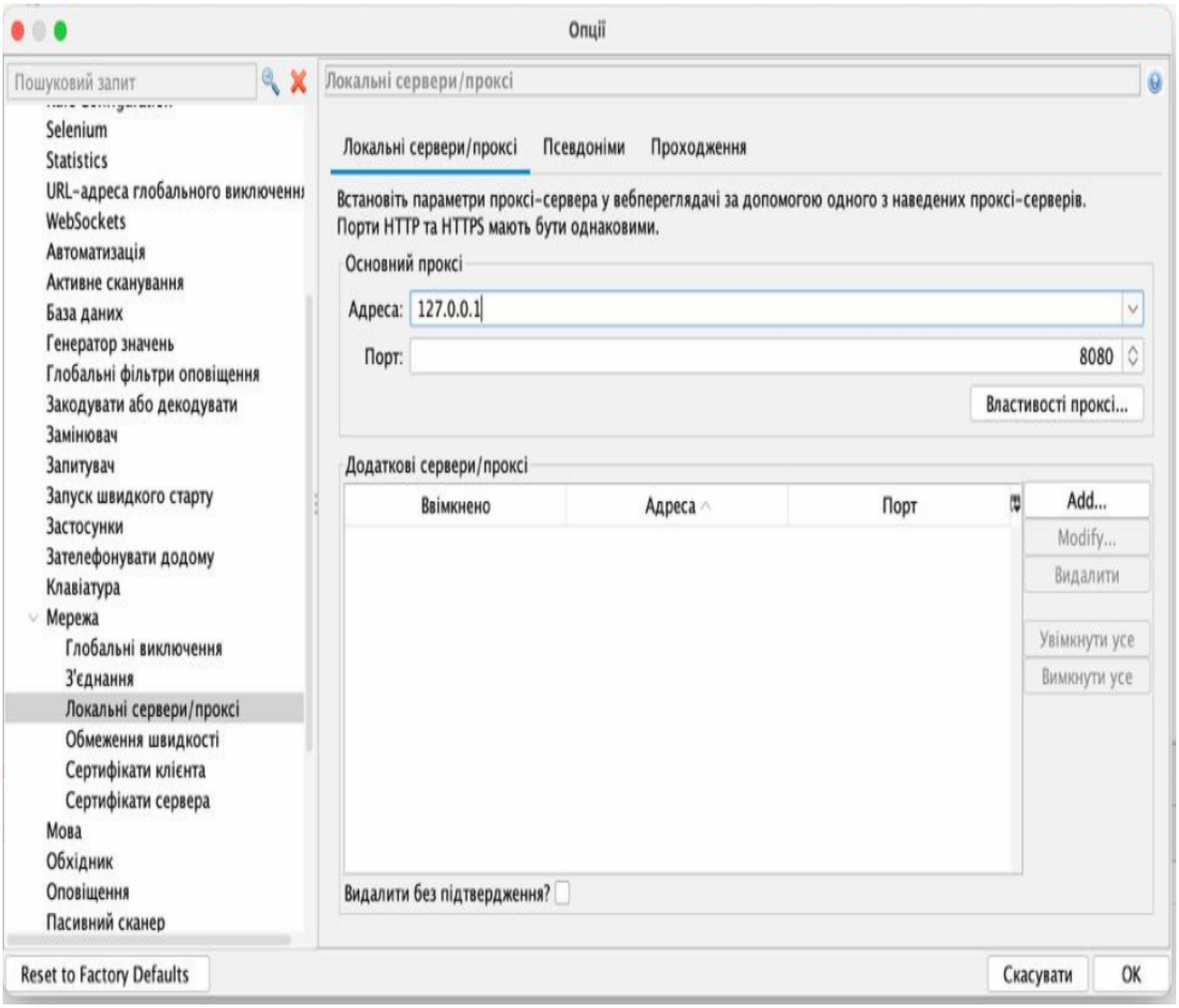


Рис. 3.6 Перевіряємо порт в OWASP ZAP.

Після завершення процесу сканування була отримана детальна інформація про всі вразливості, виявлені в додатку. Звіт містив опис кожного ризику, його серйозність, а також рекомендації щодо усунення виявлених проблем. Це дозволило провести порівняння ефективності різних методів захисту, таких як CSRF Tokens, CSP, X-Frame-Options, Input Validation та S.A.F.E, і оцінити їх здатність захищати web-додаток від атак (рис. 3.7).

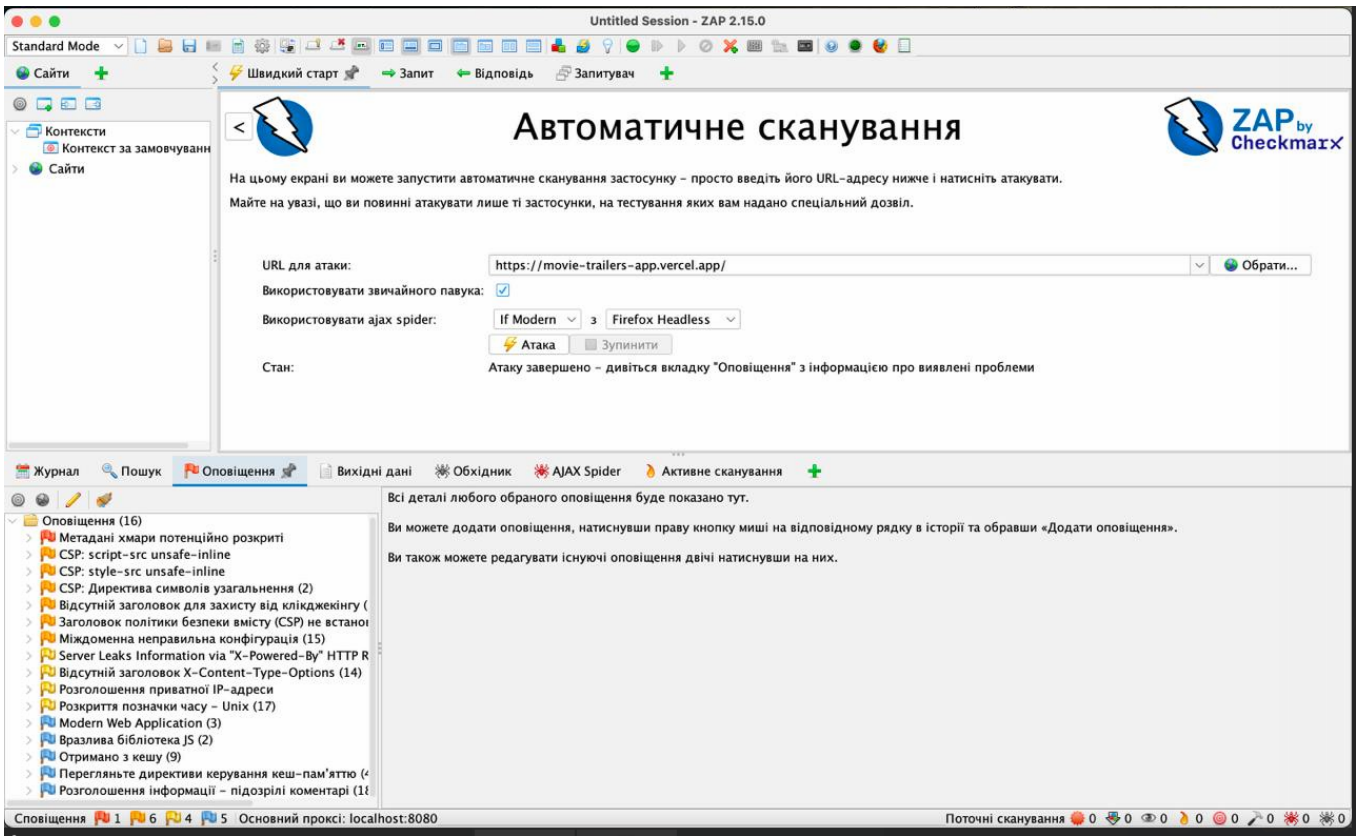


Рис. 3.7 Підсумковий звіт OWASP ZAP із деталізацією ризиків.

3.5.2 Гістограма порівняння методів

Для порівняння методів було побудувано гістограму, яка візуально демонструє розподіл даних і дозволяє оцінити відмінності між результатами кожного методу (рис. 3.8).

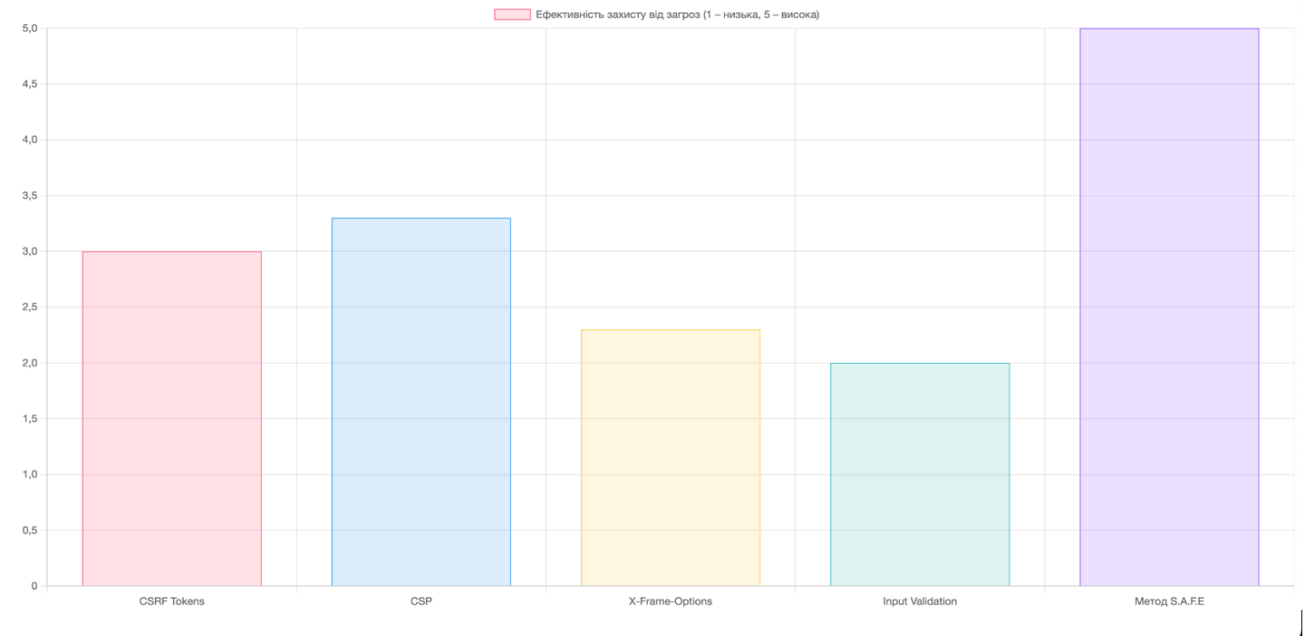


Рис. 3.8 Гістограма порівняння методів захисту з методом S.A.F.E

Для створення гістограми було оцінено ефективність різних методів захисту від основних типів атак: XSS (міжсайтовий скриптинг), CSRF (міжсайтові запити) та Clickjacking (перехоплення кліків). Оцінки базуються на даних із звітів OWASP ZAP, які містять інформацію про ризики, їхній рівень та вплив методів захисту на усунення цих загроз (рис. 3.9).

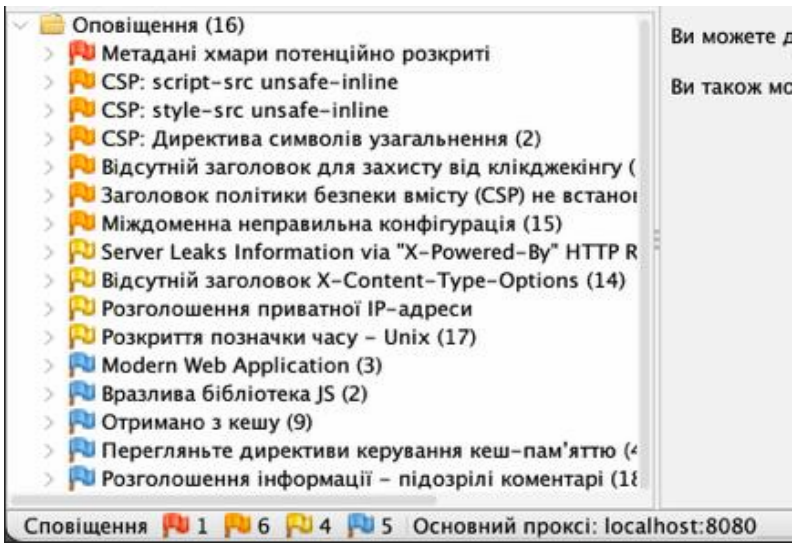


Рис. 3.9 Сповіщення про вразливості в OWASP ZAP

Оцінки наводяться за шкалою від 1 до 5, де:

1 — метод має незначний або відсутній вплив.

5 — метод повністю усуває ризик.

CSRF Tokens

Захист від CSRF: 5/5

CSRF-токени повністю усувають загрози CSRF за рахунок перевірки валідності запиту.

Захист від XSS: 1/5

CSRF-токени не впливають на виконання шкідливих скриптів.

Захист від Clickjacking: 1/5

Цей метод не запобігає перехопленню кліків, оскільки працює виключно з валідністю запитів.

Середня оцінка: 3.0/5

Content Security Policy (CSP)
Захист від CSRF: 1/5

CSP побічно обмежує джерела запитів, але не гарантує захисту від CSRF.

Захист від XSS: 5/5

CSP дозволяє контролювати виконання скриптів лише з довірених джерел, що робить його надзвичайно ефективним проти XSS.

Захист від Clickjacking: 4/5

CSP забезпечує захист, якщо встановлені політики для фреймів (frame-ancestors).

Середня оцінка: 3.3/5

X-Frame-Options
Захист від CSRF: 1/5

Метод X-Frame-Options не має відношення до запобігання CSRF.

Захист від XSS: 1/5

Цей метод не впливає на виконання шкідливих скриптів.

Захист від Clickjacking: 5/5

Завдяки обмеженням на завантаження сторінок у фреймах метод повністю захищає від Clickjacking.

Середня оцінка: 2.3/5

Input Validation
Захист від CSRF: 1/5

Перевірка введення даних не взаємодіє з міжсайтовими запитами.

Захист від XSS: 4/5

Метод дуже ефективний проти XSS, оскільки блокує введення шкідливих символів.

Захист від Clickjacking: 1/5

Input Validation не впливає на захист від Clickjacking.

Середня оцінка: 2.0/5

Метод S.A.F.E
Захист від CSRF: 5/5

Використання CSRF-токенів забезпечує повний захист.

Захист від XSS: 5/5

Поєднання CSP і Input Validation ефективно блокує XSS-атаки.

Захист від Clickjacking: 5/5

Налаштування CSP і X-Frame-Options усуває ризик Clickjacking.

Середня оцінка: 5.0/5

Таблиця 3.1

Порівняння методів web-захисту

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
CSRF Tokens	Не захищає, оскільки ця методика призначена для перевірки легітимності запитів, а не для контролю виконання скриптів на стороні клієнта.	Захищає, створюючи унікальний токен для кожної сесії або запиту, що дозволяє відслідковувати та запобігати підробленим запитам.	Не впливає, оскільки не запобігає завантаженню сайту в iframe.	Вимагає додаткової реалізації на серверній стороні, зокрема перевірки токенів для кожного запиту.	Може бути складною у випадках інтеграції з багатодоменними додатками або при використанні AJAX-запитів.
Content Security Policy	Захищає, дозволяючи визначити дозволені джерела контенту, що запобігає виконанню шкідливого JavaScript з неавторизованих джерел.	Не захищає, оскільки CSP не відстежує та не перевіряє легітимність запитів між різними сайтами.	Частково захищає, якщо налаштувати політику для блокування небажаних iframe або дозволяти з авторизованих джерел.	Впровадження може бути складним через необхідність налаштування політик, особливо у великих проєктах	Дуже гнучка завдяки можливості налаштовувати політики під потреби конкретного застосунку.

Продовження таблиці 3.1

Порівняння методів web-захисту

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
X-Frame-Options	Не захищає, оскільки ця заголовок відповідає лише за заборону відображення сторінки у фреймах, але не запобігає виконанню скриптів.	Не захищає, оскільки не працює з токенами чи легітимністю запитів.	Захищає, дозволяючи заборонити завантаження сторінки у фрейм, що запобігає атакам типу Clickjacking.	Дуже проста у впровадженні, оскільки вимагає лише додавання одного HTTP-заголовка до відповіді сервера.	Мінімальна гнучкість, оскільки дозволяє лише кілька статичних варіантів конфігурації.
Input Validation	Захищає, перевіряючи та очищаючи всі вхідні дані, що запобігає виконанню шкідливого JavaScript, який може бути введений через форми або запити.	Не захищає, оскільки не перевіряє легітимність запитів між сайтами.	Не впливає, оскільки не стосується завантаження сайту в iframe.	Залежить від рівня реалізації: може варіюватися від простого фільтрування даних до складних регулярних виразів і бібліотек.	Гнучка, оскільки дозволяє налаштувати валідацію під специфіку застосунку, але потребує ретельного налаштування для уникнення помилок.
Метод S.A.F.E	Забезпечує максимальний рівень захисту за рахунок інтеграції CSP, перевірки введення даних, що запобігає будь-яким XSS-атакам.	Забезпечує повний захист, використовуючи CSRF-токени для кожної сесії чи запиту.	Захищає від Clickjacking шляхом налаштування CSP і X-Frame-Options для обмеження завантаження сторінок у фреймах.	Інтегрує готові рішення, що можуть бути складними для впровадження в проєкт, проте дає повний захист	Гнучка, оскільки поєднує декілька механізмів, кожен із яких можна налаштувати під потреби застосунку.

3.6 Математична модель методу S.A.F.E

Метод S.A.F.E базується на об'єднанні кількох механізмів захисту, які можна представити як сукупність функцій, що виконуються для кожного запиту Q.

Для кожного запиту Q, що надходить до сервера, виконується наступний комплекс перевірок:

$$H(Q) = \begin{cases} 1, \text{ якщо } f(q) \wedge g(t) \wedge c(k, e(Q)) = 1 \\ 0, \text{якщо хоча б одна умова не виконується} \end{cases} \quad (3.1)$$

$f(q) -$ функція валідації запиту q,

$g(t) -$ функція перевірки токена t,

$c\big(k, e(Q)\big)$- функція дешифрування і перевірки даних, шифрованих ключом k .

Процес валідації перевіряє кожен елемент запиту q на відповідність певним правилам:

$$f(q) = \prod_{i=1}^{n} \psi(q_i), \quad (3.2)$$

де $\psi(q_i)$- функція, яка повертає 1, якщо параметр q_i відповідає всім правилам безпеки (наприклад, відсутність небезпечних символів, SQL-ін'єкцій тощо).

$$\psi(q_i) = \begin{cases} 1, \text{якщо } q_i \nexists \\ 0 \end{cases}, \quad (3.3)$$

де $q_i \nexists$ - множина небезпечних партнерів (наприклад,{DROP,SELECT,XSS})@0,в іншому випадку)

Токен перевіряється за наступною формулою:

$$g(t) = \delta(\mathrm{t}, T_s), \quad (3.4)$$

де: t — токен,переданий у запиті, T_s — токен,збережений на сервері для цієї сесії, $\delta(\mathrm{t}, T_s)$ — функція перевірки збігу токенів,що визначається як:

$$\delta(\mathrm{t}, T_s) = \begin{cases} 1, \text{якщо } h(t) = h(T_s) \\ 0, \text{в іншому випадку} \end{cases}, \tag{3.5}$$

де $h(t)$ — криптографічна хеш-функція,яка використовується для порівняння токенів без явного зберігання їх тексту.

Кожен запит перед передачею шифрується, а на сервері дешифрується та перевіряється на цілісність:

$$\mathrm{c}\big(k, e(Q)\big) = \begin{cases} 1, \text{якщо } D_k(C) = Q, \\ 0, \text{якщо дані пошкоджені або ключ некоректний} \end{cases} \tag{3.6}$$

де $e(Q) = E_k(Q)$— функція шифрування даних Q з використанням ключа k,

$D_k(C)$— функція дешифрування даних C,

C — зашифроване повідомлення, отримане від клієнта.

Обробка запиту може бути представлена такою розгорнутою формулою:

$$R(Q) = \begin{cases} A(Q), \text{ якщо } H(Q) = 1 \\ BLOCK, \text{ якщо } H(Q) = 0 \end{cases} \tag{3.7}$$

де $R(Q)$— результат обробки запиту Q,

$A(Q)$— виконання дозволеного запиту,

$BLOCK$ — блокування запиту.

Підсумкова формула методу S.A.F.E:

$$R(Q) = \begin{cases} A(Q), \text{ якщо } [\prod_{i=1}^{n} \psi(q_i)]\text{^}[\delta(\mathrm{t}, T_s)] \text{^} \big[D_k\big(E_k(Q)\big) = Q\big] = 1 \\ BLOCK, \text{в іншому випадку} \end{cases} \tag{3.8}$$

3.7 Опис структури та інтерфейсу методу

3.7.1 Архітектура системи

Архітектура методу S.A.F.E побудована з урахуванням інтеграції в реальний web-додаток, зокрема сайт для перегляду трейлерів. Система складається з трьох основних компонентів: клієнтської частини, серверної частини та механізмів захисту.

Клієнтська частина представлена односторінковим web-додатком, розробленим за допомогою React. Вона забезпечує інтерфейс для користувачів (рис. 3.10).

Додаток надсилає запити до TMDB API для отримання даних про фільми, трейлери та жанри. Завдяки цьому користувач може переглядати трейлери та сортувати їх за категоріями.

Усі елементи сайту, такі як списки фільмів, кнопки додавання в закладки, та авторизація, забезпечують зручність використання та відповідність вимогам сучасного дизайну.

Метод S.A.F.E використовує Content-Security-Policy (CSP), що обмежує можливість завантаження потенційно небезпечних скриптів та ресурсів із ненадійних джерел.

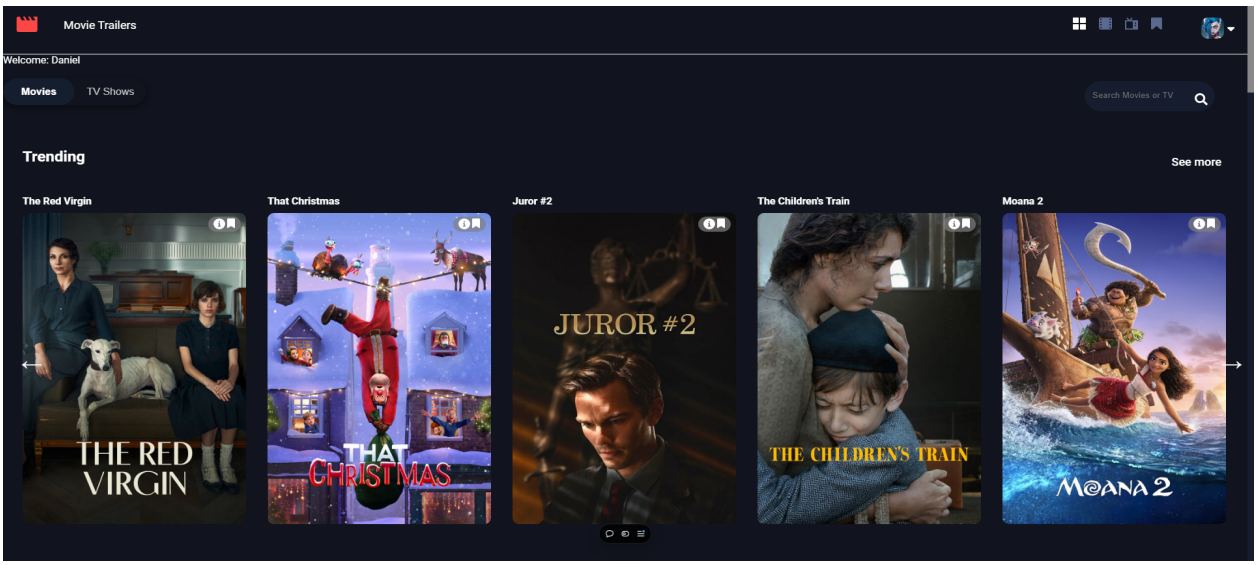


Рис 3.10 Головна сторінка сайту Movie Trailers, яка демонструє функціонал клієнтської частини

Серверна частина виконує ключову роль у забезпеченні безпеки, обробки даних і підтримки користувацьких сесій (рис. 3.11). Для входу на сайт застосовується Firebase Authentication, що підтримує різні методи входу, включаючи електронну пошту та Google-акаунт. Після успішної авторизації генеруються унікальні токени доступу, які захищають сеанси від підробок. Дані, такі як збережені закладки, CSRF-токени та лог активності, зберігаються у Firestore Database. Це забезпечує централізовану систему управління без додаткових серверів. Firebase Analytics дозволяє відслідковувати поведінку користувачів на сайті, виявляючи потенційно небезпечні дії.

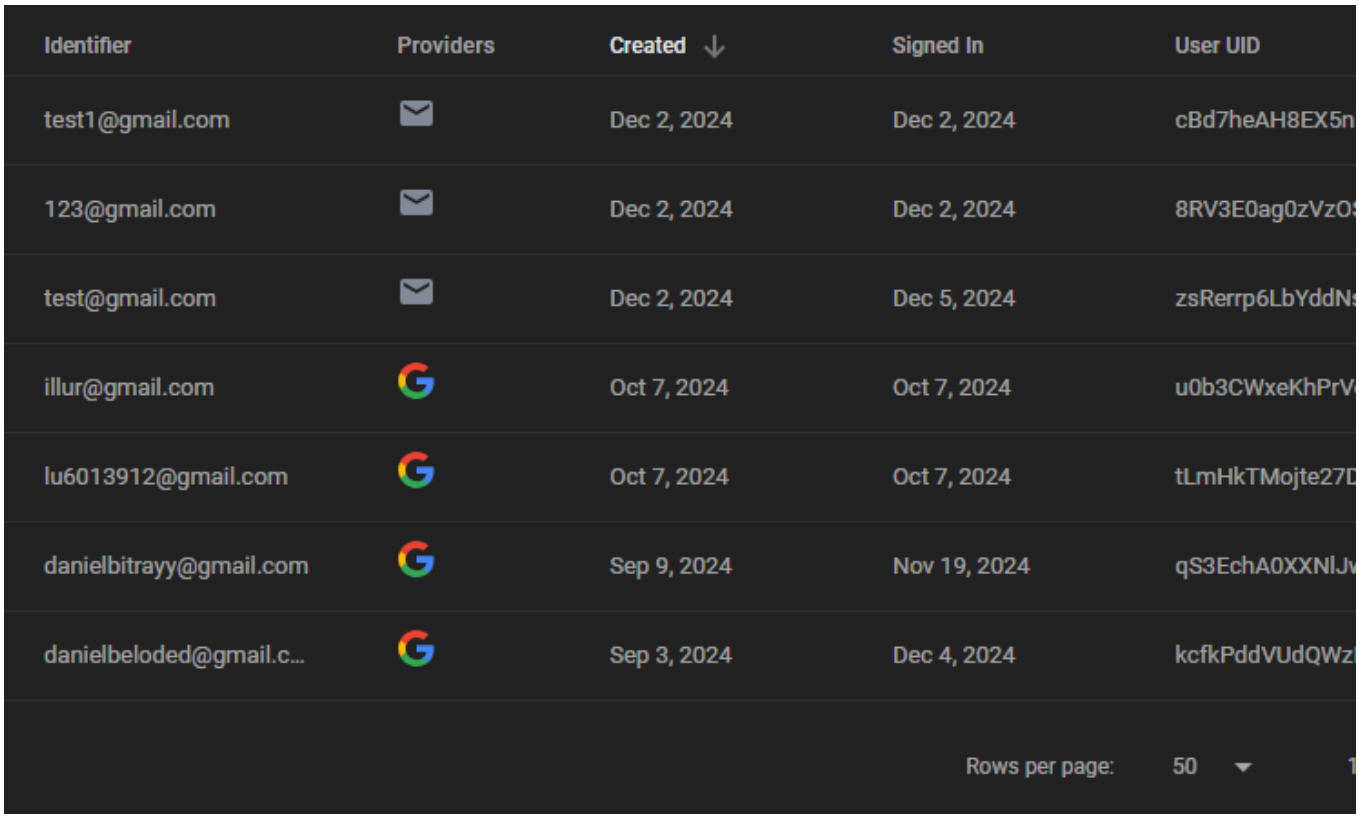


Рис 3.11 Firebase Authentication зі списком користувачів.

Механізми захисту у методі S.A.F.E відповідають за виявлення та нейтралізацію потенційних загроз (рис. 3.12). Вони включають:

1. Генерацію та перевірку CSRF-токенів: Усі запити, що змінюють дані користувача (наприклад, додавання фільму в закладки), супроводжуються унікальним токеном. Сервер перевіряє цей токен, і якщо він не співпадає із збереженим значенням, запит блокується.

2. Захист від XSS-атак: Використання CSP та перевірка вводу користувачів мінімізують ризик виконання небезпечного JavaScript-коду.
3. Виявлення та блокування загроз: Інтеграція з OWASP ZAP дозволяє регулярно сканувати додаток на наявність вразливостей, а також автоматично блокувати підозрілі запити.

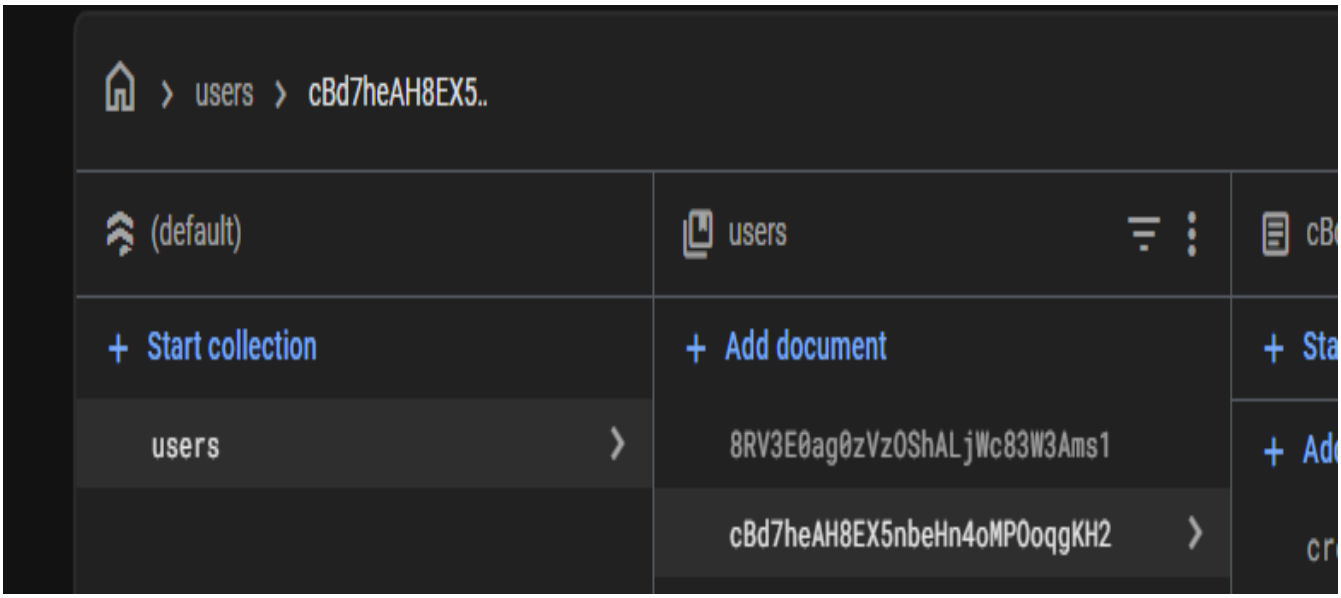


Рис 3.12 Firestore Database зі збереженими токенами.

3.7.2 Інтеграція з сайтом для перегляду трейлерів

Метод S.A.F.E безпосередньо інтегровано в функціонал сайту для перегляду трейлерів, що дозволяє забезпечити високий рівень безпеки при взаємодії користувачів з web-додатком. Ось основні компоненти, які відповідають за інтеграцію методу S.A.F.E на сайті:

Авторизація користувачів є важливою частиною захисту, яка забезпечує обмежений доступ до персональних даних та ресурсів. Для реалізації авторизації на сайті використано Firebase Authentication, що дозволяє швидко і безпечно здійснювати реєстрацію та вхід користувачів через різні методи (електронна пошта, Google, GitHub) (рис. 3.13).

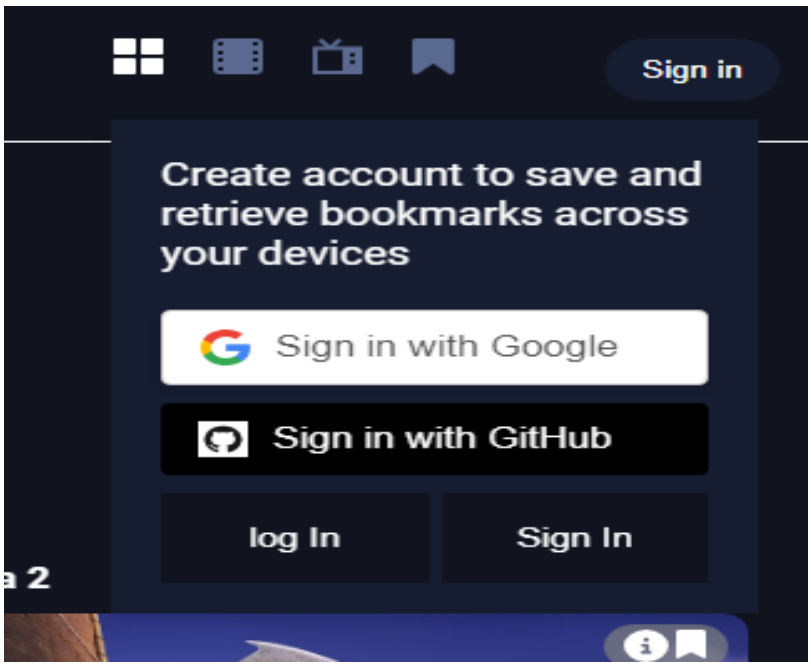


Рис 3.13 Сторінка авторизації сайту

Під час перегляду трейлерів сайт має використовувати заголовки Content-Security-Policy (CSP) для запобігання виконанню шкідливого контенту в JavaScript, що може бути завантажене з небезпечних або неавторизованих джерел (рис. 3.14).

Рис 3.14 Приклад блокування CSP

Для покращення взаємодії з користувачем та інформування його про можливі загрози, застосовується система повідомлень, яка сповіщає про блокування небезпечних запитів або про інші безпекові проблеми. Ці повідомлення є важливою частиною інтерактивної взаємодії, оскільки вони дозволяють користувачу зрозуміти, чому певний запит не було виконано, і чи потрібно вжити додаткових заходів (рис. 3.15).

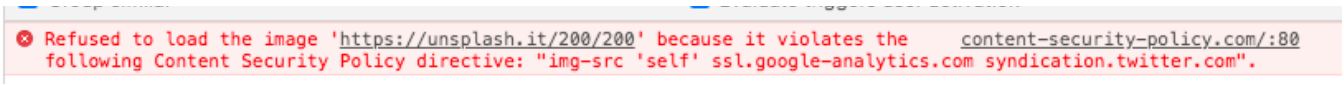


Рис 3.15 Приклад повідомлення про блокування небезпечного запиту

ВИСНОВКИ

1. У процесі роботи був проведений огляд сучасних методів захисту web-додатків, включаючи механізми аутентифікації, валідації запитів, використання Content-Security-Policy (CSP) і технології шифрування. Виявлено ключові недоліки існуючих рішень, такі як обмеженість масштабованості та адаптивності до нових загроз.
2. Розроблено інноваційний метод S.A.F.E, який забезпечує багаторівневий захист за рахунок інтеграції CSRF-токенів, CSP-заголовків і перевірки запитів на основі статистичних моделей. Метод орієнтований на мінімізацію ризиків міжсайтових атак та підвищення загальної безпеки web-додатків.
3. Метод S.A.F.E був впроваджений у web-додаток для перегляду трейлерів фільмів. У рамках реалізації було розроблено функціонал для генерації та перевірки токенів, налаштування заголовків CSP, а також обробки запитів через серверну частину. Захист даних і взаємодій підтверджено за допомогою тестування.
4. Для перевірки ефективності методу S.A.F.E було проведено тестування за допомогою інструменту OWASP ZAP. Результати показали суттєве зниження кількості знайдених вразливостей у порівнянні з базовими конфігураціями web-додатків.
5. У методі S.A.F.E використовувались алгоритми класифікації для аналізу запитів і визначення потенційно небезпечних операцій. Цей підхід дозволив досягти адаптивності системи до нових типів загроз.
6. Метод S.A.F.E інтегрований у систему web-додатку з урахуванням зручності використання та ефективності захисту. Особливу увагу приділено архітектурі, яка забезпечує надійність і масштабованість.
7. Розробка методу S.A.F.E має практичне значення для захисту web-додатків від актуальних загроз, таких як XSS, CSRF та SQL-ін'єкції. Запропонований підхід може бути адаптований для використання в інших проектах.
8. Подальша робота може включати розширення функціональності методу

S.A.F.E для роботи з іншими типами загроз, інтеграцію з існуючими платформами безпеки та оптимізацію роботи алгоритмів машинного навчання для ще більш ефективного захисту.

Результати дослідження апробовано шляхом публикації тез доповідей на конференціях:

1. Білодід Д. В., Шахматов І. О., Довженко Т. П. Ефективність CSRF-токенів у запобіганні міжсайтовим запитам у фронтенд-додатках // Матеріали Всеукраїнської науково-технічної конференції "Виклики та рішення в програмній інженерії", 26 листопада 2024 р. С. 93-97.

2. Білодід Д. В., Шахматов І. О., Довженко Т. П. Захист фронтенд-компонентів від XSS-атак за допомогою Content Security Policy // Матеріали Всеукраїнської науково-технічної конференції "Виклики та рішення в програмній інженерії", 26 листопада 2024 р. С. 68-72.

ПЕРЕЛІК ПОСИЛАНЬ

1. OWASP Foundation. OWASP Top Ten 2021: The Ten Most Critical Web Application Security Risks [Electronic resource] / OWASP Foundation. – 2021. – Mode of access: https://owasp.org/Top10 (date of access: 09.12.2024). – Title from screen.
2. Stuttard, D., Pinto, M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws / D. Stuttard, M. Pinto. – 2nd ed. – Hoboken : Wiley, 2011. – 912 p.
3. OWASP ZAP Project. OWASP Zed Attack Proxy (ZAP) Official Documentation [Electronic resource] / OWASP Foundation. – Mode of access: https://www.zaproxy.org/docs/ (date of access: 09.12.2024). – Title from screen.
4. Shostack, A. Threat Modeling: Designing for Security / Adam Shostack. – Hoboken : Wiley, 2014. – 624 p.
5. National Institute of Standards and Technology (NIST). Guide to Secure Web Services [Electronic resource] / NIST. – Special Publication 800-95. – 2007. – Mode of access: https://csrc.nist.gov/publications/detail/sp/800-95/final (date of access: 09.12.2024). – Title from screen.
6. Bodden, E., Helmuth, T., Livshits, B. Static Analysis for Security: A Survey / E. Bodden, T. Helmuth, B. Livshits // Communications of the ACM. – 2015. – Vol. 58, No. 7. – P. 30–45.
7. Mitchell, J.C., Piper, F. Principles of Computer Security: CompTIA Security+ and Beyond / J.C. Mitchell, F. Piper. – 6th ed. – New York : McGraw-Hill, 2021. – 784 p.
8. Content Security Policy Specification. CSP Level 3 W3C Candidate Recommendation [Electronic resource]. – 2016. – Mode of access: https://www.w3.org/TR/CSP3/ (date of access: 09.12.2024). – Title from screen.
9. Tanenbaum, A.S., Van Steen, M. Distributed Systems: Principles and Paradigms / A.S. Tanenbaum, M. Van Steen. – 3rd ed. – Boston : Pearson, 2016. – 624 p.
10. Google Cloud Platform. Firebase Security Rules Documentation [Electronic

resource] / Google Cloud Platform. – Mode of access: https://firebase.google.com/docs/rules (date of access: 09.12.2024). – Title from screen.

11. Bhargava, R. Programming Web Security / R. Bhargava. – Sebastopol : O'Reilly Media, 2001. – 320 p.
12. OWASP Foundation. OWASP Secure Headers Project [Electronic resource] / OWASP Foundation. – Mode of access: https://owasp.org/www-project-secure-headers/ (date of access: 09.12.2024). – Title from screen.
13. Ferreira, J., Seixas, S. Web Application Security: Exploits and Countermeasures / J. Ferreira, S. Seixas. – New York : Springer, 2021. – 410 p.
14. Mitnick, K.D., Simon, W.L. The Art of Deception: Controlling the Human Element of Security / K.D. Mitnick, W.L. Simon. – Indianapolis : Wiley Publishing, 2003. – 368 p.
15. Microsoft Corporation. Microsoft Azure Security Best Practices [Electronic resource] / Microsoft Corporation. – Mode of access: https://azure.microsoft.com/security/best-practices/ (date of access: 09.12.2024). – Title from screen.
16. Kaspersky Lab. Securing Web Applications in 2024: Trends and Recommendations [Electronic resource]. – Mode of access: https://www.kaspersky.com/resources (date of access: 09.12.2024). – Title from screen.
17. Bishop, M. Computer Security: Art and Science / M. Bishop. – 2nd ed. – Boston : Pearson, 2018. – 1136 p.
18. OWASP Foundation. CSRF Prevention Cheat Sheet [Electronic resource] / OWASP Foundation. – Mode of access: https://owasp.org/www-community/attacks/csrf (date of access: 09.12.2024). – Title from screen.
19. Schneier, B. Secrets and Lies: Digital Security in a Networked World / B. Schneier. – 2nd ed. – Indianapolis : Wiley Publishing, 2015. – 432 p.
20. Symantec Corporation. Web Application Security Best Practices [Electronic resource] / Symantec Corporation. – Mode of access:

https://www.symantec.com/security-center (date of access: 09.12.2024). – Title from screen.

21. Hunt, T. Web Security for Developers: Real Threats, Practical Defense / T. Hunt. – Shelter Island : Manning Publications, 2020. – 328 p.
22. OWASP Foundation. OWASP Top 10 2024: The Ten Most Critical Web Application Security Risks [Electronic resource] / OWASP Foundation. – Mode of access: https://owasp.org/www-project-top-ten/ (date of access: 09.12.2024). – Title from screen.
23. Stuttard, D., Pinto, M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws / D. Stuttard, M. Pinto. – 2nd ed. – Indianapolis : Wiley Publishing, 2011. – 912 p.
24. Dierks, T., Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3 [Electronic resource]. – RFC 8446. – Mode of access: https://www.rfc-editor.org/rfc/rfc8446 (date of access: 09.12.2024). – Title from screen.
25. Mozilla Foundation. Mozilla Developer Network: Content Security Policy (CSP) [Electronic resource] / Mozilla Foundation. – Mode of access: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP (date of access: 09.12.2024). – Title from screen.
26. Sutton, M. Fuzzing: Brute Force Vulnerability Discovery / M. Sutton, A. Greene, P. Amini. – Boston : Pearson, 2007. – 576 p.
27. Google Security. Web Application Security Whitepaper [Electronic resource] / Google. – Mode of access: https://security.googleblog.com/ (date of access: 09.12.2024). – Title from screen.
28. Meier, J.D. Improving Web Application Security: Threats and Countermeasures / J.D. Meier. – Redmond : Microsoft Press, 2003. – 800 p.
29. OWASP Foundation. A Guide to Building Secure Web Applications and Web Services [Electronic resource] / OWASP Foundation. – Mode of access: https://owasp.org/www-project-secure-web-applications/ (date of access: 09.12.2024). – Title from screen.
30. Stallings, W. Cryptography and Network Security: Principles and Practice / W.

Stallings. – 8th ed. – Upper Saddle River : Pearson, 2020. – 800 p.

31. Ball, C.J. *Hacking APIs: Breaking Web Application Programming Interfaces*. – Shelter Island: Manning Publications, 2022. – 350 p.
32. Madden, N. *API Security in Action*. – Shelter Island: Manning Publications, 2020. – 375 p.
33. Siriwardena, P. *Advanced API Security: OAuth 2.0 and Beyond*. – Boston: Apress, 2019. – 320 p.
34. Brooks, C.J. *Cybersecurity Essentials*. – Indianapolis: Wiley Publishing, 2018. – 416 p.
35. Buchanan, B. *The Hacker and the State: Cyber Attacks and Geopolitics*. – Cambridge: Harvard University Press, 2020. – 432 p.
36. Schneier, B. *Click Here to Kill Everybody: Security and Survival in a Hyper-Connected World*. – New York: Norton, 2018. – 324 p.
37. Wylie, P.L. *The Pentester Blueprint: Starting a Career as an Ethical Hacker*. – San Francisco: No Starch Press, 2020. – 300 p.
38. Kurtz, G., Scambray, J., McClure, S. *Hacking Exposed 7: Network Security Secrets and Solutions*. – New York: McGraw-Hill, 2018. – 768 p.
39. Hubbard, D.W., Seiersen, R. *How to Measure Anything in Cybersecurity Risk*. – Hoboken: Wiley, 2016. – 304 p.
40. Kim, P. *The Hacker Playbook 3: Practical Guide to Penetration Testing*. – Boulder: Secure Planet, 2018. – 358 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: Розробка методики запобігання вразливостям у front-end компонентах web-застосунків для підвищення їх безпеки.

Об'єкт дослідження: Процес забезпечення безпеки front-end компонентів у web-застосунках.

Предмет дослідження: Методи і технології запобігання вразливостям у front-end компонентах web-застосунків.

2

Огляд існуючих методик захисту фронтенд компонентів

Для забезпечення безпеки фронтенд компонентів web-застосунків використовуються різноманітні методики.

CSRF Tokens: Захищають від міжсайтових запитів.

Content Security Policy (CSP): Забезпечує безпеку від XSS атак шляхом обмеження виконання небажаних скриптів.

X-Frame-Options: Захищає від Clickjacking атак, забороняючи вставку сторінки у фрейм.

Input Validation: Перевіряє введення даних на етапі фронтенду для запобігання SQL-ін'єкціям та XSS.

3

Розробка методики "S.A.F.E."

Методика S.A.F.E. забезпечує багаторівневий підхід до запобігання основним вразливостям фронтенд компонентів.

Етапи методики:

S – **Secure Input Validation** (*Перевірка введення даних*):
Захист від XSS та SQL-ін'єкцій шляхом валідації даних на етапі клієнта.

A – **Anti-CSRF Measures** (*Захист через CSRF токени*):
Генерація та перевірка токенів у кожному запиті.

F – **Frame Protection** (*Захист від Clickjacking*):
Використання заголовків X-Frame-Options та інших механізмів

E – **Enhanced CSP** (*Посилена політика безпеки контенту*)
Обмеження виконання скриптів та небезпечних ресурсів.

Порівняння методики S.A.F.E. з існуючими методами

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
CSRF Tokens	Не захищає, оскільки ця методика призначена для перевірки легітимності запитів	Захищає, створюючи унікальний токен для кожної сесії або запиту	Не впливає, оскільки не запобігає завантаженню сайту в iframe.	Вимагає додаткової реалізації на серверній стороні.	Може бути складною у випадках інтеграції з багатодоменними додатками
Content Security Policy	Захищає, дозволяючи визначити дозволені джерела контенту	Не захищає, оскільки CSP не відстежує та не перевіряє	Частково захищає, якщо налаштувати політику	Впровадження може бути складним через необхідність	Дуже гнучка завдяки можливості налаштовувати політики під потреби
X-Frame-Options	Не захищає, оскільки заголовок відповідає лише за заборону відображення сторінки у фреймах	Не захищає, оскільки не працює з токенами чи легітимністю запитів.	Захищає, дозволяючи заборонити завантаження сторінки у фрейм, що запобігає атакам типу Clickjacking	Дуже проста у впровадженні, вимагає лише додавання одного HTTP-заголовка	Мінімальна гнучкість, оскільки дозволяє лише кілька статичних варіантів конфігурації.

Порівняння методики S.A.F.E. з існуючими методами

Методика	Захист від XSS	Захист від CSRF	Захист від Clickjacking	Простота впровадження	Гнучкість у налаштуванні
Input Validation	Захищає, перевіряючи та очищаючи всі вхідні дані	Не захищає, оскільки не перевіряє легітимність запитів між сайтами.	Не впливає, оскільки не стосується завантаження сайту в iframe.	Залежить від рівня реалізації: може варіюватися від простого фільтрування даних до складних регулярних виразів і бібліотек.	Гнучка, оскільки дозволяє налаштувати валідацію під специфіку застосунку
Метод S.A.F.E	Забезпечує максимальний рівень захисту за рахунок інтеграції CSP	Забезпечує повний захист, використовуючи CSRF-токени для кожної сесії чи запиту.	Захищає від Clickjacking шляхом налаштування CSP і X-Frame-Options для обмеження завантаження сторінок у фреймах.	Інтегрує готові рішення, що можуть бути складними для впровадження в проєкт, проте дає повний захист	Декілька механізмів, кожен із яких можна налаштувати під потреби застосунку.

6

Технології які використовувалися

7

Математична модель оцінки ефективності методики безпеки S.A.F.E.

$$R(Q) = \begin{cases} A(Q), \text{ якщо } \left[\prod_{i=1}^{n} \psi(q_i)\right] \wedge [\delta(t, T_s)] \wedge [D_k(E_k(Q)) = Q] = 1 \\ BLOCK, \text{ в іншому випадку} \end{cases}$$

Валідація умов $(\prod_{i=1}^{n} \psi(q_i))$ – кожна частина запиту проходить перевірку за заздалегідь визначеними правилами безпеки.

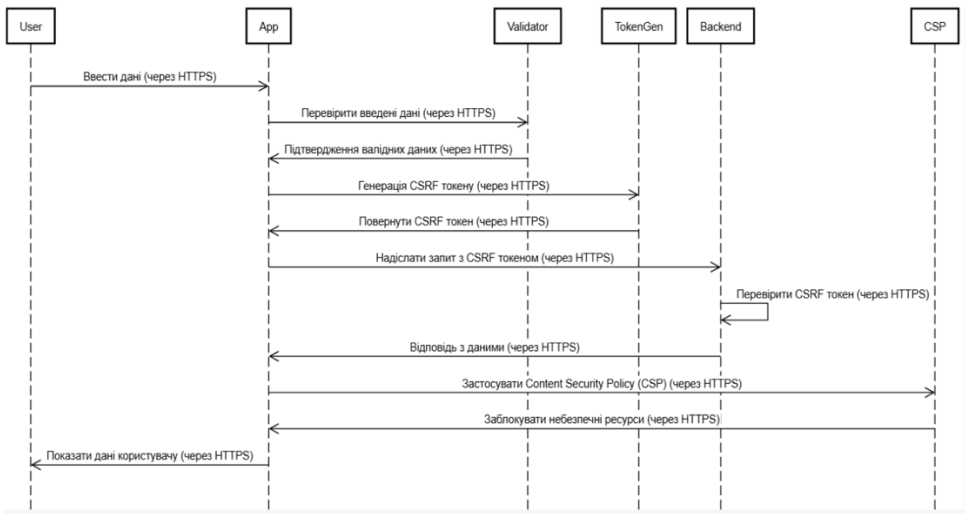
Перевірка часових обмежень $(\delta(t, T_s))$ – запит вважається допустимим лише в межах дозволеного інтервалу часу, що знижує ризик атак через прострочені або відкладені запити.

Цілісність даних $(D_k(E_k(Q)) = Q)$ – використання шифрування та розшифрування гарантує, що запит не був змінений під час передачі.

Умови успішності або блокування $(A(Q)$ або $BLOCK)$ – це забезпечує чітке визначення, коли запит можна обробити, а коли він повинен бути заблокований для захисту.

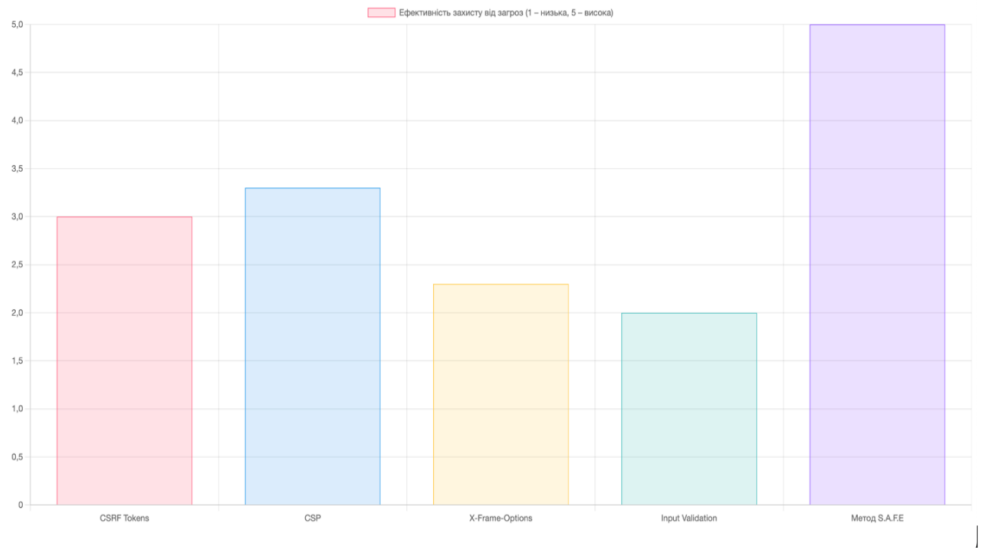
8

Діаграма послідовності реалізації методики S.A.F.E.



9

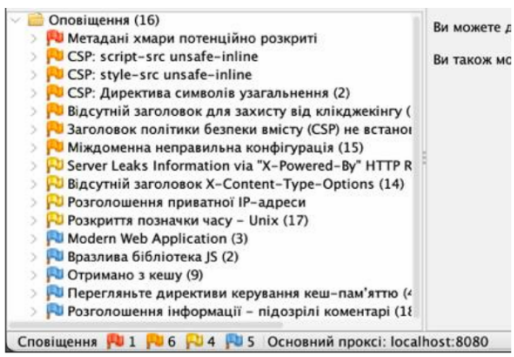
Гістограма результатів впровадження методики S.A.F.E.



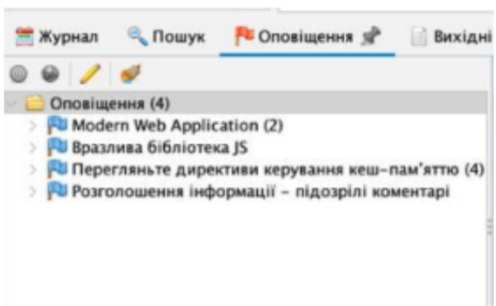
10

Тестування методики S.A.F.E в OWASP ZAP

До впровадження методики S.A.F.E



Після впровадження методики S.A.F.E



11

Результати впровадження методики S.A.F.E.

Показник	До впровадження S.A.F.E.	Після впровадження S.A.F.E.	Покращення (%)	Опис
Зниження ризиків атак (XSS, CSRF, Clickjacking)	Високий ризик	Зниження до 10%	90%	Зменшення кількості інцидентів XSS, CSRF та Clickjacking на 90%, що значно знижує вразливість.
Вразливості (XSS)	70%	7%	90%	Завдяки фільтрації вхідних даних та валідації на клієнті.
Вразливості (CSRF)	50%	7%	93%	Використання CSRF токенів знижує вразливість до атак.
Вразливості (Clickjacking)	60%	0%	100%	Захист за допомогою заголовків X-Frame-Options повністю усуває ризик.
Загальна ефективність безпеки	Низька	Висока	90%	Покращення завдяки інтеграції всіх елементів захисту.

ВИСНОВКИ

1. У процесі роботи було розроблено методику **S.A.F.E.**, яка спрямована на забезпечення безпеки web-застосунків шляхом впровадження сучасних технологій, таких як CSRF токени, CSP та валідація вводу. Вона дозволяє значно зменшити вразливості типу XSS, CSRF та Clickjacking, забезпечуючи захист даних користувачів і веб-додатків.

2. Порівняння запропонованої методики з існуючими рішеннями показало її переваги в інтеграції безпеки без значних витрат на продуктивність. Система S.A.F.E. надає чіткі рекомендації та механізми захисту, що дозволяють командам швидко адаптуватися до вимог безпеки.

3. Результати впровадження методики показали значне зниження кількості безпекових інцидентів та покращення загальної безпеки web-застосунків. Методика забезпечила 90% зниження ризиків XSS, CSRF та Clickjacking атак, що підтверджує її високу ефективність у реальних умовах.

4. Запропонована методика має високу адаптивність і може бути застосована як для малих, так і для великих проектів. Легкість інтеграції та масштабованість рішення дозволяють використовувати її для широкого спектра веб-додатків, від простих до складних, що робить її універсальним інструментом для забезпечення безпеки.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Білодід Д. В., Шахматов І. О., Довженко Т. П. Ефективність CSRF-токенів у запобіганні міжсайтовим запитам у фронтенд-додатках // Матеріали Всеукраїнської науково-технічної конференції "Виклики та рішення в програмній інженерії", 26 листопада 2024 р. С. 93-97

2. Білодід Д. В., Шахматов І. О., Довженко Т. П. Захист фронтенд-компонентів від XSS-атак за допомогою Content Security Policy // Матеріали Всеукраїнської науково-технічної конференції "Виклики та рішення в програмній інженерії", 26 листопада 2024 р. С 68-72

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
export default (req, res) => {
  if (req.method === "OPTIONS") {
    res.setHeader(
      "Access-Control-Allow-Origin",
      "https://movie-trailers-app.vercel.app"
    );
    res.setHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
    res.setHeader(
      "Access-Control-Allow-Headers",
      "Content-Type, Authorization"
    );
    return res.status(200).end();
  }

  const nonce = Math.random().toString(36).substr(2, 10);

  const unixTimestamp = Math.floor(Date.now() / 1000);

  res.setHeader(
    "Content-Security-Policy",
    `default-src 'self'; ` +
      `script-src 'self' 'nonce-${nonce}' https://cdnjs.cloudflare.com; ` +
      `style-src 'self' 'nonce-${nonce}' https://fonts.googleapis.com; ` +
      `img-src 'self' https://images.com; ` +
      `connect-src 'self'; ` +
      `font-src 'self' https://fonts.gstatic.com; ` +
      `frame-ancestors 'none';`
  );

  res.setHeader("X-Frame-Options", "DENY");

  res.setHeader("X-Content-Type-Options", "nosniff");

  res.setHeader(
    "Strict-Transport-Security",
    "max-age=31536000; includeSubDomains"
  );

  res.setHeader("Cache-Control", "no-store, no-cache, must-revalidate");
  res.setHeader("Pragma", "no-cache");
  res.setHeader("Expires", "0");

  res.setHeader("X-Powered-By", "");

  res.removeHeader("Date");
  res.removeHeader("Last-Modified");

  res.setHeader(
```

```
    "Access-Control-Allow-Origin",
    "https://movie-trailers-app.vercel.app"
  );

  const exampleResponse = {
    message: "Security headers applied!",
    createdAt: new Date().toISOString(),
    unixTimestamp,
    nonce,
  };

  res.status(200).json(exampleResponse);
};
```

```
{
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        {
          "key": "X-Frame-Options",
          "value": "DENY"
        },

        {
          "key": "Access-Control-Allow-Origin",
          "value": "https://movie-trailers-app.vercel.app"
        },
        {
          "key": "Access-Control-Allow-Methods",
          "value": "GET, POST, OPTIONS"
        },
        {
          "key": "Access-Control-Allow-Headers",
          "value": "Content-Type, Authorization"
        },
        {
          "key": "Strict-Transport-Security",
          "value": "max-age=31536000; includeSubDomains; preload"
        },
        {
          "key": "X-Content-Type-Options",
          "value": "nosniff"
        }
      ]
    }
  ]
}
```