Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА
на тему:

Розробка підсистеми захисту веб-додатків від атак CSRF I XSS та атак типу SQL- та PHP -ін'єкцій»

Виконав: студент 4 курсу, гр.2КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Марцун Н. М. _____
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. МБІС
Грицак А. В. _____
(прізвище та ініціали)
«6» червня 2024 р.

Рецензент: к.т.н., доцент каф. ОТ
Городецька О. С. _____
(прізвище та ініціали)
«6» червня 2024 р.

Допущено до захисту
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю. Є. _____
“6” червня 2024 р.

Вінниця ВНТУ – 2024

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ кафедри МБІС

_____ **д.т.н., проф. Яремчук Ю. Є.**

«22» березня 2024р.

З А В Д А Н Н Я
на бакалаврську дипломну роботу студенту
Марцуну Назару Миколайовичу
(прізвище та ініціали)

1. Тема роботи: «Розробка підсистеми захисту веб-додатків від атак CSRF І XSS та атак типу SQL- та PHP-ін'єкцій»
Керівник роботи: к.т.н., доцент. каф. МБІС Грицак А. В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ректора закладу вищої освіти від «22» березня 2024 року № 80
2. Строк подання студентом роботи за тиждень до захисту
3. Вихідні дані до роботи:
Метою роботи є розробка підсистеми захисту веб-додатків від поширених типів атак, таких як CSRF (міжсайтова підробка запитів), XSS (міжсайтовий скриптинг), SQL-ін'єкції та PHP-ін'єкції, для забезпечення безпеки та цілісності даних користувачів та запобігання несанкціонованому доступу до веб-додатків.
4. Зміст текстової частини:
У першому розділі дипломної роботи проводиться аналіз вразливостей веб-додатків до атак CSRF, XSS, SQL- та PHP-ін'єкцій, досліджуються існуючі методи та підходи до захисту від цих типів атак, їхні переваги та недоліки. У другому розділі розробляється архітектура та функціональні вимоги до підсистеми захисту веб-додатків від зазначених типів атак, обґрунтовуються обрані технології та підходи до реалізації захисту. У третьому розділі дипломної роботи здійснюється програмна реалізація підсистеми захисту з використанням запропонованих технологій та методів. У висновках підсумовуються результати виконаної роботи.
5. Перелік графічного матеріалу:

У першому розділі бакалаврської дипломної роботи наявно 6 рисунків, у другом розділі – 4 рисунків, у третьому розділі – 11 рисунків.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Грицак А. В., к.т.н., доц. каф. МБІС		
Розділ II	Грицак А. В., к.т.н., доц. каф. МБІС		
Розділ III	Грицак А. В., к.т.н., доц. каф. МБІС		

7. Дата видачі завдання «22» березня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	22.02.2024	25.02.2024	
2	Аналіз предметної області обраної теми	26.02.2024	09.03.2024	
3	Розробка алгоритму роботи	09.03.2024	04.04.2024	
4	Написання бакалаврської роботи на основі розробленої теми	05.04.2024	06.05.2024	
5	Передзахист бакалаврської дипломної роботи	06.05.2024	24.05.2024	
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	27.05.2024	03.06.2024	
7	Захист бакалаврської дипломної роботи	12.06.2024	13.06.2024	

Студент _____ Марцун Н. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Грицак А. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається із 81 сторінок тексту формату А4, включаючи 21 рисунок, 1 таблицю, посилання на 39 літературних джерел та 4 додатки.

Метою роботи є розробка підсистеми захисту веб-додатків від атак CSRF і XSS та атак типу SQL- та PHP-ін'єкцій з використанням сучасних методів та технологій забезпечення безпеки веб-додатків.

У теоретичній частині роботи проведено аналіз основних типів атак на вебдодатки, їх особливостей та наслідків. Досліджено існуючі методи та механізми захисту від атак CSRF і XSS, SQL- та PHP-ін'єкцій. Також розглянуто сучасні стандарти та рекомендації з безпеки веб-додатків. У другому розділі обґрунтовано необхідність розробки підсистеми захисту веб-додатків, спроектовано архітектуру підсистеми захисту від атак CSRF і XSS та атак типу SQL- і PHP-ін'єкцій, а також розроблено методи захисту від зазначених атак.

Третій розділ присвячений практичній реалізації підсистеми захисту вебдодатків. Обґрунтовано вибір інструментів та технологій для розробки підсистеми, розроблено компоненти захисту від CSRF і XSS атак, SQL- та PHP-

ін'єкцій. Проведено тестування розробленої підсистеми захисту на прикладах веб-додатків та підготовлено інструкції для користувача щодо використання підсистеми.

Результати роботи показали, що розроблена підсистема захисту веб-додатків забезпечує ефективний захист від атак CSRF і XSS, SQL- та PHP-ін'єкцій. Підсистема має модульну архітектуру, низький вплив на продуктивність вебдодатків та надає детальні звіти для моніторингу безпеки. Розроблена підсистема може бути інтегрована в різноманітні веб-додатки для підвищення їх безпеки та захисту конфіденційних даних користувачів.

Ключові слова: веб-додатки, безпека, CSRF, XSS, SQL-ін'єкції, PHP-ін'єкції, підсистема захисту, методи захисту, архітектура, розробка.

ANNOTATION

The bachelor's thesis consists of 81 pages of A4 text, including 21 figures, 1 table, references to 39 literary sources and 4 appendices.

The aim of the study is to develop a subsystem for protecting web applications from CSRF and XSS attacks and SQL and PHP injection attacks using modern methods and technologies for ensuring the security of web applications.

The theoretical part of the paper analyzes the main types of attacks on web applications, their features and consequences. Existing methods and mechanisms for protecting against CSRF and XSS attacks, SQL and PHP injections are investigated. Modern standards and recommendations for web application security are also considered. The second section substantiates the need to develop a web application security subsystem, designs the architecture of the subsystem for protection against CSRF and XSS attacks, SQL and PHP injection attacks, and develops methods of protection against these attacks.

The third section is devoted to the practical implementation of the web application security subsystem. The choice of tools and technologies for the development of the subsystem is substantiated, and the components of protection against CSRF and XSS attacks, SQL and PHP injections are developed. The developed security subsystem was tested on examples of web applications and user instructions for using the subsystem were prepared.

The results have shown that the developed web application security subsystem provides effective protection against CSRF and XSS attacks, SQL and PHP injections. The subsystem has a modular architecture, low impact on web application performance, and provides detailed reports for security monitoring. The developed subsystem can be integrated into various web applications to improve their security and protect confidential user data.

Keywords: web applications, security, CSRF, XSS, SQL injections, PHP injections, security subsystem, security methods, architecture, development.

ЗМІСТ

ВСТУП

Актуальність теми. В умовах стрімкого розвитку інформаційних технологій та зростання популярності веб-додатків проблема забезпечення їх безпеки набуває особливої актуальності. Веб-додатки стають все більш складними та функціональними, що, в свою чергу, призводить до збільшення ризиків виникнення вразливостей та можливих векторів атак. Однією з найбільших загроз для вебдодатків є атаки, спрямовані на порушення конфіденційності, цілісності та доступності даних, такі як міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF), SQL-ін'єкції та PHP-ін'єкції.

XSS атаки дозволяють зловмисникам вводити шкідливий код на веб-сайти, що може призвести до крадіжки облікових даних користувачів, поширення шкідливого програмного забезпечення та інших небезпечних наслідків. CSRF атаки змушують жертву випадково виконувати небажані дії на уразливому веб-сайті, на якому вони автентифіковані. SQL-ін'єкції використовуються для отримання несанкціонованого доступу до баз даних веб-додатків, що може призвести до витоку конфіденційної інформації або навіть повного компрометування системи. PHP-ін'єкції є схожими на SQL-ін'єкції, але експлуатують вразливості в обробці введених даних в PHPкоді.

Захист від цих атак є надзвичайно важливим для забезпечення безпеки вебдодатків та конфіденційності даних користувачів. Ефективний захист потребує комплексного підходу, який включає використання належних методів та механізмів захисту, дотримання сучасних стандартів безпеки, а також розробку спеціалізованих підсистем захисту.

Метою даної роботи є розробка підсистеми захисту веб-додатків від атак CSRF і XSS та атак типу SQL- та PHP-ін'єкцій з використанням сучасних методів та технологій забезпечення безпеки веб-додатків.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

− провести аналіз основних типів атак на веб-додатки, їх особливостей та наслідків;

− дослідити існуючі методи та механізми захисту від атак CSRF і XSS, SQL- та PHP-ін'єкцій;

− спроектувати архітектуру підсистеми захисту веб-додатків від атак CSRF і XSS та атак типу SQL- і PHP-ін'єкцій;

− розробити методи захисту від зазначених атак у складі підсистеми;

− провести тестування розробленої підсистеми захисту на прикладах вебдодатків;

− підготувати інструкції для користувача щодо використання підсистеми захисту.

Об'єктом дослідження є процеси розробки та впровадження підсистеми захисту веб-додатків від атак на веб-додатки.

Предметом дослідження є методи, механізми та архітектура підсистеми захисту веб-додатків від атак CSRF і XSS, SQL- та PHP-ін'єкцій.

Наукова новизна одержаних результатів полягає у розробці комплексної підсистеми захисту веб-додатків, яка забезпечує ефективний захист від атак різних типів та може бути інтегрована у різноманітні веб-додатки.

Практичне значення отриманих результатів полягає у можливості застосування розробленої підсистеми захисту для підвищення безпеки різноманітних веб-додатків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ЗАХИСТУ ВЕБ-ДОДАТКІВ

1.1. Аналіз основних типів атак на веб-додатки

Веб-додатки, у своїй сутності, є інтегрованими системами, які відкриті для взаємодії з користувачем. Ця особливість робить їх особливо уразливими перед різноманітними атаками. Отже, важливим є розуміння різновидів потенційних загроз, як перший крок до належного захисту. Атаки на веб-додатки, при успішному їх здійсненні, можуть призвести до негативних наслідків, таких як витік конфіденційної інформації, фінансові втрати та шкода репутації компанії.
Розуміння механізмів цих атак та ефективних методів їх захисту є ключовим аспектом в сфері сучасної кібербезпеки [1].

Однією з найпоширеніших та найнебезпечніших атак є Cross-Site Scripting (XSS). Ці атаки полягають у вбудовуванні шкідливих скриптів у веб-сторінки, які потім виконуються в браузері жертви. Шкідливі скрипти, в свою чергу, можуть виконувати різноманітні дії, починаючи від крадіжки кукісів і до виконання дій від імені користувача (рис.1.1) [2].

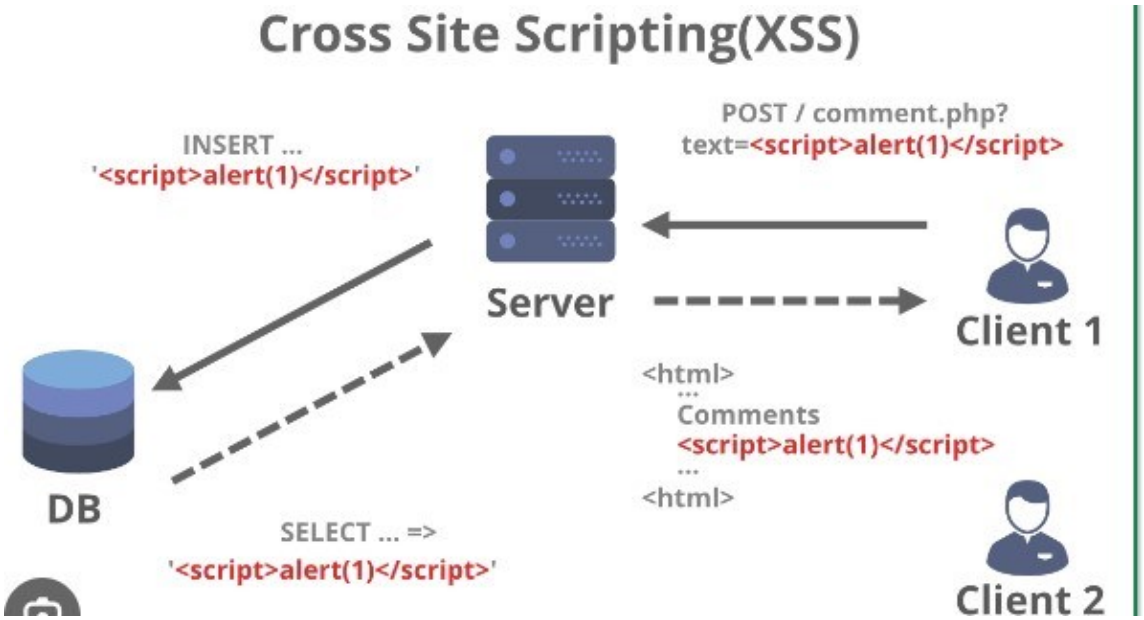


Рисунок 1.1 – Атака Cross-Site Scripting (XSS)

Наприклад, користувач може активувати шкідливий скрипт, ненавмисно відвідавши веб-сторінку, яка була маскована під надійний ресурс, або натиснувши на маніпульоване посилання. Іншим прикладом може бути вбудований скрипт безпосередньо в коментарі або повідомлення на форумах, де контент не проходить фільтрацію на предмет наявності потенційно шкідливого коду.

Ще однією з атак, що представляє серйозну загрозу, є Cross-Site Request Forgery (CSRF). Ці атаки змушують браузер вже аутентифікованого користувача виконати небажані дії на сайті без його відома. Ці дії можуть включати в себе, наприклад, зміну адреси електронної пошти або навіть надсилання запитів на переказ коштів [3].

Механізми проведення атак CSRF можуть включати відправку спеціально сформованих посилань або електронних листів, що містять запити на виконання певних дій на веб-сайті. Важливо зазначити, що часто веб-додатки не вимагають додаткового підтвердження намірів користувача для виконання важливих дій, що робить атаки CSRF особливо небезпечними (рис. 1.2) [4].

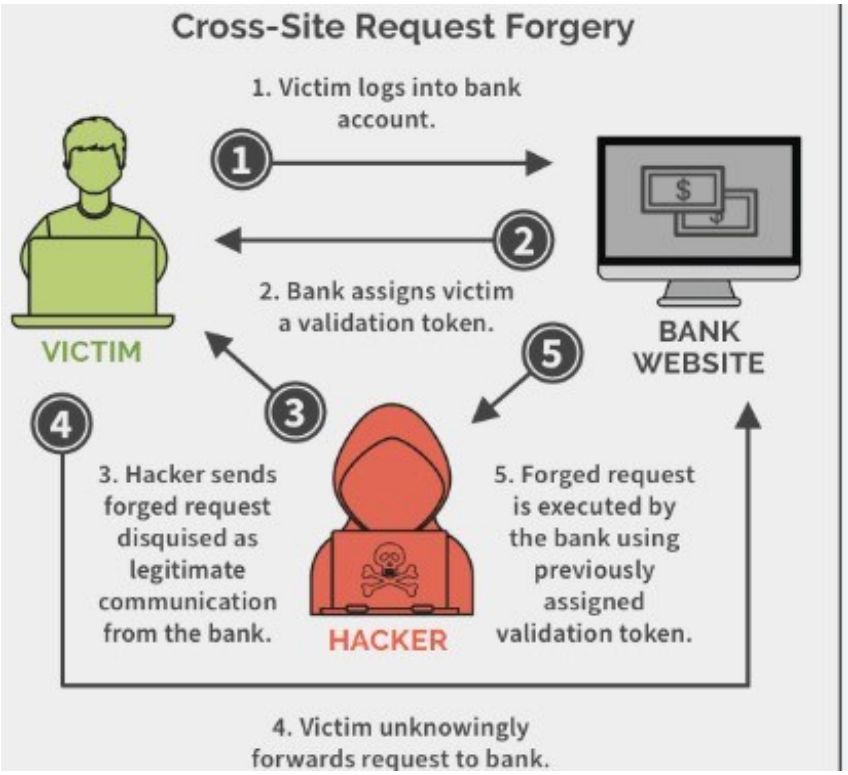


Рисунок 1.2 – Механізми проведення атак CSRF

CSRF атаки змушують веб-браузер користувача виконати небажані дії на сайті, на якому він аутентифікований. Ці дії можуть варіюватися від зміни email адреси до надсилання запитів на переказ коштів.

Механізми проведення атак CSRF:

• атакувальник може надіслати посилання або email, який містить запит на викоання дії на веб-сайті. Якщо користувач у цей момент автентифікований на сайті, його браузер автоматично додасть до запиту свої автентифікаційні cookies, що дасть змогу запиту бути виконаним;

• часто веб-додатки не вимагають додаткового підтвердження намірів користувача для виконання важливих дій, що робить атаки CSRF особливо небезпечними.

SQL Injection є ще одним серйозним видом атак, що становить значну загрозу безпеці веб-додатків. Під час цієї атаки, зловмисники зловживають можливістю маніпулювати SQL-запитами, які надсилаються до бази даних через веб-форми. Це

відкриває для них можливість не лише читати або модифікувати дані в базі даних, але й виконувати адміністративні операції на сервері (рис. 1.3) [5].

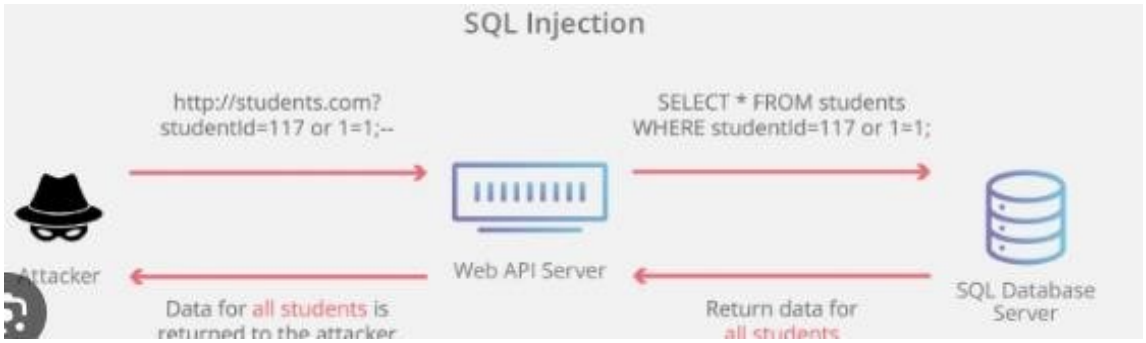


Рисунок 1.3 – SQL Injection

Наприклад, неправильно фільтровані або неекрановані дані, які користувач вводить у форму пошуку або в поля введення, можуть бути використані для впровадження шкідливого SQL коду. Цей код може змінювати структуру виконуваного запиту, що відкриває шлях для зловмисників до отримання неправомірного доступу до бази даних.

Також важливо відзначити, що атака SQL Injection може призвести до розкриття всієї бази даних, включно з особистими даними користувачів, фінансовою інформацією та іншими конфіденційними даними. Це може мати серйозні наслідки як для користувачів, так і для компанії, що володіє веб-додатком, і може призвести до значних втрат даних та порушення конфіденційності.

1.2. Аналіз механізмів захисту від CSRF і XSS атак

Захист від XSS (Cross-Site Scripting) атак є невід'ємною складовою безпеки веб-додатків, оскільки ці атаки використовують вразливості, що дозволяють вбудовувати шкідливі скрипти в контент, що інтерпретується браузером користувача. Шкідливі скрипти можуть здійснювати різноманітні дії, включаючи крадіжку кукісів, сесійних токенів або навіть переписування вмісту веб-сторінок на стороні клієнта [6].

Методи захисту від XSS включають застосування різноманітних технік та механізмів, спрямованих на запобігання виконанню шкідливих скриптів у вебдодатках. Однією з таких технік є використання Content Security Policy (CSP) [7]. CSP дозволяє веб-додаткам встановлювати політику безпеки, в якій вони можуть вказати, з яких джерел контент є надійним, тим самим запобігаючи виконанню зловмисних скриптів. Наприклад, за допомогою директив CSP можна обмежити завантаження скриптів з будь-яких джерел, крім основного сервера.

Ще однією ефективною технікою є автоматичне екранування введення, коли дані, введені користувачем, автоматично екрануються для запобігання їх інтерпретації як частини HTML або JavaScript. Це дозволяє уникнути вразливостей, пов'язаних зі вставкою шкідливих скриптів через форми введення.

Крім того, посилення валідації на стороні сервера є ключовим аспектом захисту від XSS атак. Сервери повинні застосовувати строгі правила валідації, перевіряючи всі вхідні дані на предмет відповідності очікуваному формату і відкидаючи будьякі запити, що містять потенційно небезпечні дані. Це допомагає запобігти виконанню шкідливих дій, які можуть бути виконані зловмисниками через вразливість XSS у веб-додатках.

CSRF атаки використовують автентифіковані сесії користувачів для виконання несанкціонованих дій на веб-сайтах без відома користувача. Такі атаки можуть змусити користувача невідомо виконати дії, такі як зміна пароля або передача коштів (рис. 1.5) [9].

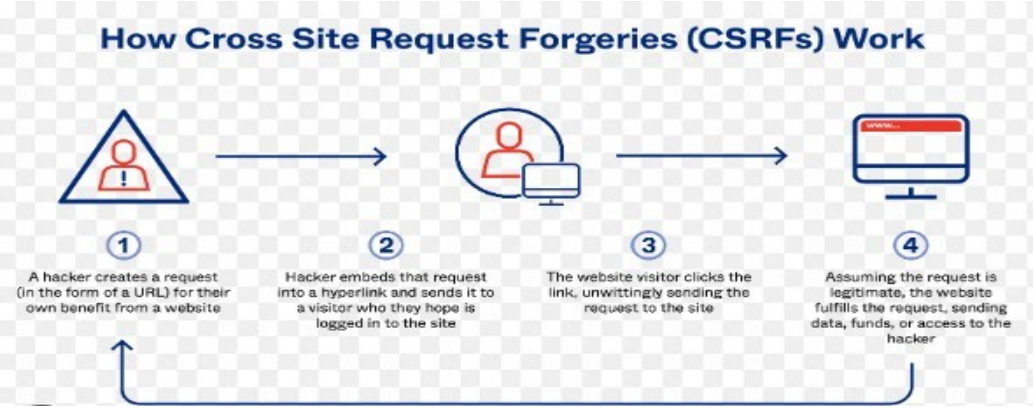


Рисунок 1.5 – CSRF атаки

CSRF атаки використовують автентифіковані сесії користувачів для виконання несанкціонованих дій на веб-сайтах без їхнього відома. Це може призвести до виконання дій, таких як зміна пароля або передача коштів, без уповноваження користувача. Важливою складовою захисту від подібних атак є впровадження відповідних методів та механізмів.

Один із методів захисту від CSRF - це використання токенів CSRF. Кожна форма або запит, який призначений для внесення змін у систему, повинен містити унікальний токен. Цей токен не може бути згенерований або вгаданий зловмисником, що забезпечує, що лише легітимні запити, санкціоновані користувачем, будуть оброблені системою.

Ще одним методом є перевірка походження запиту [10]. Це означає перевірку заголовків HTTP Origin або Referer, що дозволяє перевірити, чи був запит надісланий з дозволеного джерела, а не зі зловмисного сайту. Це допомагає упевнитися, що запити, що надійшли, є відповідними та безпечними.

Також варто використовувати атрибути SameSite для Cookies. Налаштування цих атрибутів для кукісів може автоматично блокувати відправку cookies на міжсайтові запити, ускладнюючи проведення CSRF атак [11].

Впровадження цих заходів утворює міцну базу для захисту веб-додатків від CSRF атак. Ці заходи разом із захистом від XSS створюють комплексний підхід до

безпеки веб-додатків. їх впровадження є необхідним для будь-якої системи, що прагне забезпечити безпеку даних користувачів та систем.

1.3. Аналіз методів захисту від SQL- та PHP-ін’єкцій

SQL-ін'єкції лишаються однією з найнебезпечніших загроз для веб-додатків, оскільки вони можуть надати зловмисникам можливість маніпулювати базою даних, витягувати конфіденційну інформацію, змінювати або видаляти дані. Для запобігання таким атакам застосовуються різноманітні методи захисту.

Один з ефективних методів захисту від SQL-ін'єкцій – це використання параметризованих запитів. Цей підхід є золотим стандартом у запобіганні SQLін'єкціям. Він передбачає використання спеціальних API, які розділяють дані та код, тим самим не дозволяючи переданим даним впливати на структуру SQLзапиту. Під час використання параметризованих запитів, дані користувача обробляються окремо та безпечно в межах запиту (рис. 1.5) [12].

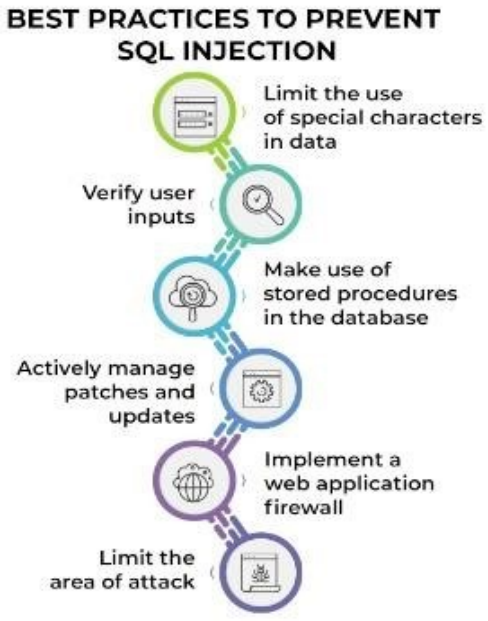


Рисунок 1.5 – Найкращі практики захисту від SQL

Додатково до використання параметризованих запитів, важливим є також належна валідація введених даних. Валідація допомагає перевірити коректність та

відповідність введених даних вимогам до формату. Це дозволяє відсіяти потенційно шкідливі дані перед їх передачею до бази даних.

Ще одним ефективним заходом є обмеження прав доступу до бази даних. Застосування принципів найменших можливих прав доступу (least privilege) дозволяє обмежити зловмисникам можливість виконання небажаних операцій навіть у випадку успішної атаки.

SQL-ін'єкції є однією з найбільш небезпечних загроз для веб-додатків, оскільки вони дозволяють зловмисникам виконувати маніпуляції з базою даних, витягуючи конфіденційну інформацію, модифікуючи або видаляючи дані. Для захисту від таких атак застосовуються різноманітні методи.

Один із стандартних методів запобігання SQL-ін'єкціям полягає у використанні параметризованих запитів. Цей підхід використовує спеціальні API для розділення даних та коду, забезпечуючи недоступність переданих даних для впливу на структуру SQL-запиту. Шляхом використання змінних у запитах, дані користувача обробляються окремо та безпечно [16].

ORM (Object-Relational Mapping) є ще одним методом, який використовується для запобігання SQL-ін'єкціям. Цей підхід дозволяє розробникам працювати з базами даних за допомогою об'єктно-орієнтованого підходу, автоматично запобігаючи багатьом видам ін'єкцій. Популярні ORM бібліотеки, такі як Entity Framework для .NET, Hibernate для Java і ActiveRecord для Ruby on Rails, є прикладами такого підходу.

Крім того, керування правами доступу в базах даних за принципом найменших привілеїв є важливим елементом захисту від SQL-ін'єкцій. Кожен компонент системи повинен отримувати лише ті права доступу, які абсолютно необхідні для його нормального функціонування. Це допомагає мінімізувати можливий збиток у разі успішної атаки.

У контексті PHP-ін'єкцій, які також є серйозною загрозою безпеці, використовуються наступні підходи. Вимкнення небезпечних PHP-функцій, таких як eval(), exec(), system(), є ключовим елементом забезпечення безпеки, оскільки вони дозволяють виконання довільного коду [17]. Для зменшення ризиків, пов'язаних із прямим виконанням PHP коду, рекомендується використовувати зовнішні шаблонізатори та обробники даних. Це дозволяє вбудовувати динамічний вміст на веб-сторінки без виконання PHP коду, зменшуючи потенційні вразливості.

Нарешті, постійне оновлення та підтримка всіх компонентів системи, включно з PHP та пов'язаними бібліотеками, є важливими аспектами забезпечення безпеки. Оновлення регулярно включають патчі для усунення вразливостей, що допомагає зменшити ризик потенційних атак.

1.4. Аналіз сучасних стандартів та рекомендації з безпеки веб-додатків

У контексті сфери кібербезпеки, що постійно розвивається, використання встановлених стандартів і дотримання рекомендацій є ключем до ефективного захисту веб-додатків. Стандарти та рекомендації допомагають організаціям розробляти, тестувати та підтримувати безпечні системи. Одним із ключових стандартів є OWASP Top Ten, який представляє собою список з десяти найкритичніших загроз безпеці веб-додатків і регулярно оновлюється експертами в галузі безпеки. Цей документ слугує основою для розуміння ключових ризиків, з якими стикаються веб-додатки, і пропонує керівництва щодо їх усунення. Наприклад, загрози, які включають ін'єкції та недоліки в аутентифікації та управлінні сесіями, становлять потенційні вразливості, які вимагають уваги розробників для запобігання можливим атакам [20].

Ще одним важливим міжнародним стандартом є ISO/IEC 27001, який описує, як побудувати і підтримувати систему управління інформаційною безпекою

(ISMS). Цей стандарт допомагає організаціям захистити інформацію на всіх рівнях, застосовуючи систематичний підхід до управління конфіденційною інформацією. Він охоплює такі аспекти, як оцінка ризиків, розроблення політик безпеки та контрольних заходів [21].

Ще одним важливим стандартом є PCI DSS, спрямований на організації, які обробляють дані кредитних карт. Цей стандарт забезпечує захист платіжних даних, мінімізуючи можливості їх крадіжки або неправомірного використання. Основні вимоги PCI DSS включають захист даних картодержателя, розроблення та підтримання безпечної мережевої інфраструктури, а також регулярне тестування та моніторинг мережевих ресурсів (рис. 1.6) [22].

Рисунок 1.6 – Найкращі практики веб захисту

Для ефективного управління даними важливо дотримуватися рекомендацій щодо секретності даних, які включають регулярні аудити, шифрування даних та використання авторизованих сервісів для обробки даних. При цьому принципи "Secure by Design", "Secure by Default" та "Secure in Deployment" підкреслюють важливість інтеграції заходів безпеки на всіх етапах розробки та експлуатації програмного забезпечення.

Регулярне тестування та оцінювання вразливостей веб-додатків через пентестинг та сканування на вразливості є ключовими процесами для виявлення та усунення потенційних загроз в IT-інфраструктурі. Навчання розробників методам безпечного програмування також є важливим аспектом забезпечення загальної безпеки розробки. Таким чином, кожен із зазначених стандартів та рекомендацій

відіграє важливу роль у створенні та підтримці захищеного операційного середовища для веб-застосунків, надаючи організаціям необхідні інструменти для проактивного управління ризиками та захисту цінних інформаційних активів.

Додатково до згаданих стандартів і рекомендацій, існують інші важливі аспекти безпеки веб-додатків, які варто враховувати. Наприклад, стандарт безпеки зв'язку TLS (Transport Layer Security) відіграє важливу роль у захисті конфіденційності та цілісності даних, переданих через мережу. Використання TLS дозволяє захистити веб-додатки від атак типу "прослуховування" та "впровадження даних", що можуть призвести до компрометації конфіденційної інформації [23].

Ще однією ключовою складовою безпеки є заходи захисту від DDoS-атак (розподілені атаки з обмеженим доступом). Ці атаки можуть призвести до недоступності веб-додатку для законних користувачів, що може серйозно підірвати довіру до сервісу. Встановлення захисту, такого як виявлення аномального трафіку та використання CDN (Content Delivery Network), допомагає зменшити вплив таких атак і забезпечити безперебійну роботу веб-додатку навіть під час атак [24].

Нарешті, для виявлення та реагування на потенційні загрози в реальному часі використовуються системи моніторингу та реагування на інциденти (SIEM). Ці системи дозволяють аналізувати журнали подій, виявляти аномалії та сповіщати про можливі інциденти безпеки, що дозволяє оперативно реагувати на потенційні загрози та зменшує час відновлення після інцидентів.

1.5 Висновки та постановка задач

У результаті проведеного аналізу теоретичних аспектів захисту веб-додатків можна зробити висновки.

По-перше, веб-додатки є вразливими до різноманітних атак, таких як XSS, CSRF, SQL-ін'єкції та PHP-ін'єкції. Ці атаки можуть призвести до серйозних наслідків, включаючи витік конфіденційних даних, фінансові втрати та репутаційні

збитки. Розуміння механізмів цих атак та ефективних методів їх захисту є ключовим аспектом у сфері сучасної кібербезпеки.

По-друге, існують ефективні методи та механізми захисту від XSS та CSRF атак, такі як використання Content Security Policy (CSP), автоматичне екранування введення, валідація на стороні сервера, використання CSRF токенів, перевірка походження запиту та атрибутів SameSite для Cookies. Захист від SQL-ін'єкцій забезпечується шляхом використання параметризованих запитів, належної валідації введених даних та обмеження прав доступу до бази даних. Для захисту від PHP-ін'єкцій рекомендується вимкнення небезпечних PHP-функцій, використання зовнішніх шаблонізаторів та регулярне оновлення компонентів системи.

По-третє, сучасні стандарти та рекомендації, такі як OWASP Top Ten, ISO/IEC 27001, PCI DSS, а також принципи Secure by Design, Secure by Default та Secure in Deployment, відіграють ключову роль у забезпеченні безпеки веб-додатків.

На основі проведеного аналізу можна сформулювати наступні основні задачі для подальшого дослідження та розробки підсистеми захисту веб-додатків:

− спроектувати архітектуру підсистеми захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій з урахуванням сучасних стандартів та рекомендацій.

− розробити методи захисту від CSRF, XSS, SQL- та PHP-ін'єкцій, включаючи реалізацію CSRF токенів, перевірку походження запиту, використання CSP, автоматичне екранування введення, параметризовані запити, валідацію введених даних та обмеження прав доступу до бази даних.

− провести тестування розробленої підсистеми захисту на прикладах вебдодатків та проаналізувати ефективність запропонованих рішень.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПІДСИСТЕМИ ЗАХИСТУ ВЕБ-ДОДАТКІВ ВІД АТАК CSRF І XSS ТА АТАК ТИПУ SQL- ТА PHP -ІН'ЄКЦІЙ

2.1. Обґрунтування розроблення підсистеми захисту веб-додатків від атак

CSRF і XSS та атак типу SQL- та PHP-ін'єкцій

Метою розроблення підсистеми захисту веб-додатків від атак CSRF і XSS та атак типу SQL- та PHP-ін'єкцій є мінімізація ризику несанкціонованого доступу до даних, їх зміни або видалення за допомогою основних видів атак в мережі Інтернет.

Вимоги до підсистеми захисту повинні враховувати забезпечення безпеки на декількох рівнях додатка, включно з клієнтською частиною, серверною частиною і базою даних. Метою цих вимог є мінімізація ризику несанкціонованого доступу до даних, їх зміни або видалення за допомогою основних видів атак, таких як CSRF, XSS, SQL-ін'єкції та PHP-ін'єкції.

Cross-Site Scripting є типом атак, під час яких зловмисник може впровадити шкідливий скрипт у контент, який потім буде виконано в браузері жертви. Це може призвести до крадіжки облікових даних, сесій або інших чутливих даних. Захист від XSS вимагає комплексного підходу, починаючи з фільтрації введення і закінчуючи кодуванням виведення та суворою політикою безпеки вмісту, щоб обмежити джерела, звідки можуть завантажуватися скрипти [25].

Усі вхідні дані, одержувані від користувачів, повинні проходити через процес очищення, щоб усунути шкідливі скрипти. Кодування виведення: Під час виведення даних на сторінки веб-додатка всі спеціальні символи мають бути закодовані, щоб запобігти їхньому виконанню як коду. Використання Content Security Policy CSP. Налаштування CSP має суворо контролювати джерела ресурсів і скриптів, дозволяючи завантаження тільки з довірених джерел.

CSRF атаки дають зловмиснику змогу зловмиснику змусити користувача виконати небажані дії на вебсайті, на якому він аутентифікований. Ці атаки використовують довіру веб-сайту до браузера користувача. Механізми захисту включають використання токенів CSRF, які забезпечують, що кожен запит на зміну

стану на сервері походить від дійсного користувача. Перевірка заголовків Referer і використання атрибутів кукі також сприяють мінімізації ризиків CSRF.

Кожна форма або запит, який може змінювати стан сервера, має містити унікальний токен, який сервер перевірятиме під час отримання запиту. Перевірка заголовка Referer: Налаштування сервера для перевірки заголовків Referer у запитах HTTP для запобігання запитам зі сторонніх сайтів. SameSite Cookie: Використання атрибута SameSite для куки може допомогти запобігти атакам, обмежуючи надсилання куки в міжсайтових запитах.

SQL-ін'єкції дають змогу зловмисникам вставляти або змінювати SQL-запити, які передаються до бази даних, що може призвести до несанкціонованого доступу до даних або їх знищення. Захист від таких атак здійснюється через використання параметризованих запитів і суворої валідації вхідних даних, що запобігає інтерпретації введених даних як частини SQL-коду [26].

Розроблювана підсистема захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій базується на комплексному підході до забезпечення безпеки на різних рівнях додатку. Основною метою є мінімізація ризиків несанкціонованого доступу, модифікації або видалення даних шляхом впровадження ефективних механізмів захисту від поширених типів атак.

Алгоритм роботи підсистеми захисту починається з отримання вхідного запиту від користувача. Далі виконується перевірка наявності та валідності CSRF токена, який є одним із ключових елементів захисту від CSRF атак. Наступним кроком є валідація та екранування вхідних даних запиту, що дозволяє запобігти потенційним атакам через введення шкідливих даних.

Після цього відбувається перевірка на наявність XSS-атак у вхідних даних. Цей крок є важливим для запобігання виконанню зловмисного коду на стороні клієнта. Далі виконується перевірка на наявність SQL-ін'єкцій, яка дозволяє виявити спроби несанкціонованого доступу до бази даних через модифікацію SQL-запитів. Крім

того, здійснюється перевірка на наявність PHP-ін'єкцій, щоб запобігти виконанню небажаного PHP-коду на сервері.

Якщо запит успішно проходить усі перевірки і визнається безпечним, він передається на виконання. У разі виявлення потенційних загроз під час будь-якої з перевірок, запит відхиляється, щоб запобігти можливим атакам.

Загальна схема алгоритму роботи захисту веб-додатків від атак, яка буде покладена в основу розроблюваної підсистеми, представлена на рисунку 2.1.

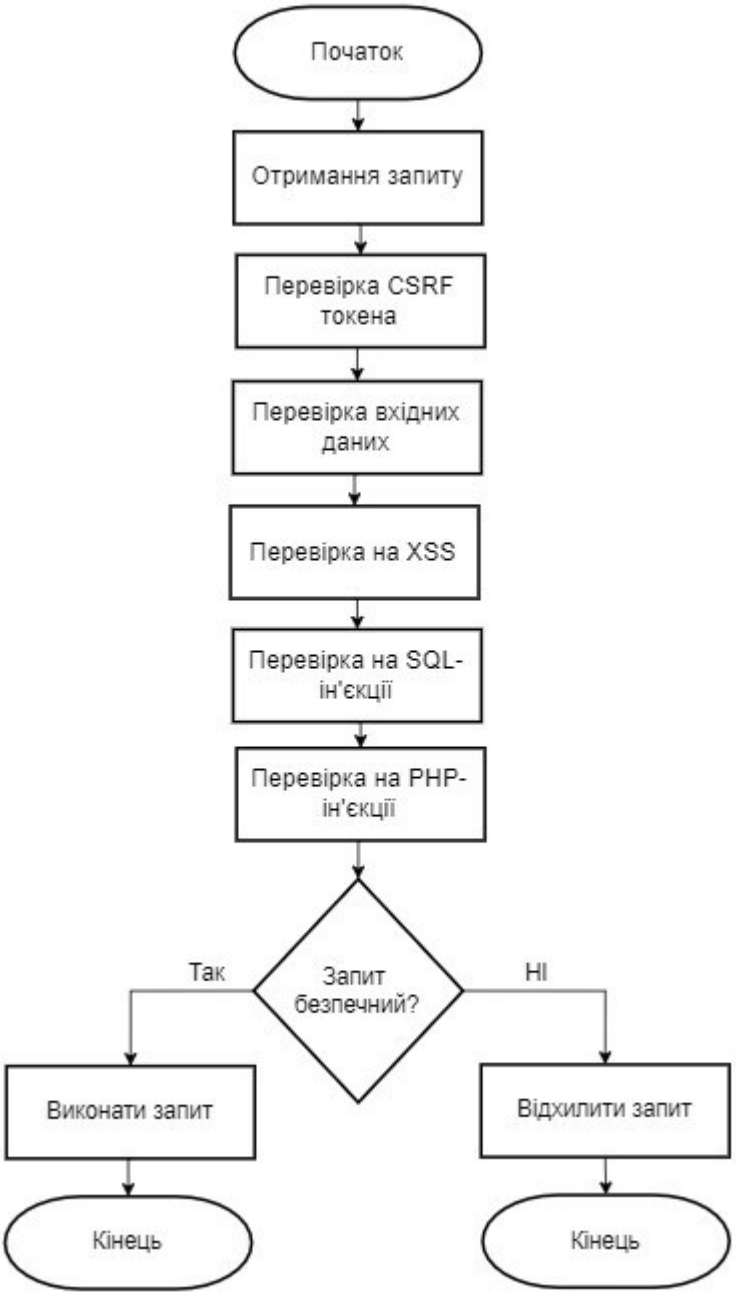


Рисунок 2.1 – Загальна схема алгоритму захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій

Крок 1. Початок.

Крок 2. Отримання запиту. Підсистема отримує вхідний запит від користувача.

Крок 3. Перевірка CSRF токена. Виконується перевірка наявності та валідності CSRF токена у запиті.

Крок 4. Перевірка вхідних даних. Здійснюється валідація та екранування вхідних даних запиту.

Крок 5. Перевірка на XSS. Перевіряється наявність потенційних XSS-атак у вхідних даних.

Крок 6. Перевірка на SQL-ін'єкції. Виконується перевірка вхідних даних на наявність SQL-ін'єкцій.

Крок 7. Перевірка на PHP-ін'єкції. Перевіряється наявність потенційних PHPін'єкцій у вхідних даних.

Крок 8. Запит безпечний? Якщо запит пройшов усі перевірки і визнаний безпечним, переходимо до кроку 9. Інакше, переходимо до кроку 10.

Крок 9. Виконати запит. Запит визнано безпечним і його можна виконати.

Крок 10. Відхилити запит. Запит визнано потенційно небезпечним і його виконання відхиляється.

Крок 11. Кінець.

Таким чином, розроблювана підсистема захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій забезпечує комплексний підхід до безпеки, враховуючи різні типи загроз та реалізуючи ефективні механізми їх виявлення та запобігання.

2.2. Проектування підсистеми захисту веб-додатківвід атак CSRF і XSS

Підсистема захисту веб-додатків від атак CSRF і XSS є важливим компонентом забезпечення безпеки користувачів та збереження цілісності даних. Ці типи атак

можуть призвести до серйозних наслідків, таких як крадіжка конфіденційної інформації, несанкціоноване виконання дій від імені користувача або порушення коректної роботи веб-додатку.

Алгоритм роботи підсистеми захисту від атак CSRF і XSS починається з отримання вхідного запиту від користувача. Далі виконується перевірка наявності CSRF токена у запиті. Якщо токен відсутній або невалідний, запит відхиляється як потенційно небезпечний. У разі наявності та валідності CSRF токена, відбувається фільтрація та екранування вхідних даних запиту з метою запобігання потенційним XSS-атакам.

Якщо вхідні дані визнаються безпечними після фільтрації та екранування, запит передається на подальшу обробку та формування відповіді. У разі виявлення потенційної XSS-атаки під час перевірки, запит відхиляється, щоб запобігти виконанню зловмисного коду на стороні клієнта.

Сформована відповідь відправляється клієнту лише у випадку успішного проходження всіх перевірок безпеки. Таким чином, підсистема захисту від атак CSRF і XSS забезпечує надійний захист веб-додатку та користувачів від цих типів атак.

Алгоритм роботи підсистеми із захисту веб-додатків від атак CSRF і XSS представлений на рисунку 2.2 у вигляді блок-схеми.

Крок 1. Початок.

Крок 2. Отримання запиту. Підсистема отримує вхідний запит від користувача.

Крок 3. Перевірка наявності CSRF токену. Перевіряється наявність CSRF токена у запиті.

Крок 4. CSRF токен валідний? Якщо CSRF токен присутній і валідний, переходимо до кроку 5. Інакше, переходимо до кроку 8.

Крок 5. Фільтрація та екранування вхідних даних. Виконується фільтрація та екранування вхідних даних запиту.

Крок 6. Дані безпечні (XSS)? Якщо вхідні дані визнано безпечними (відсутні XSS-атаки), переходимо до кроку 7. Інакше, переходимо до кроку 8.

Крок 7. Виконання запиту та формування відповіді. Запит визнано безпечним, виконується його обробка та формування відповіді.

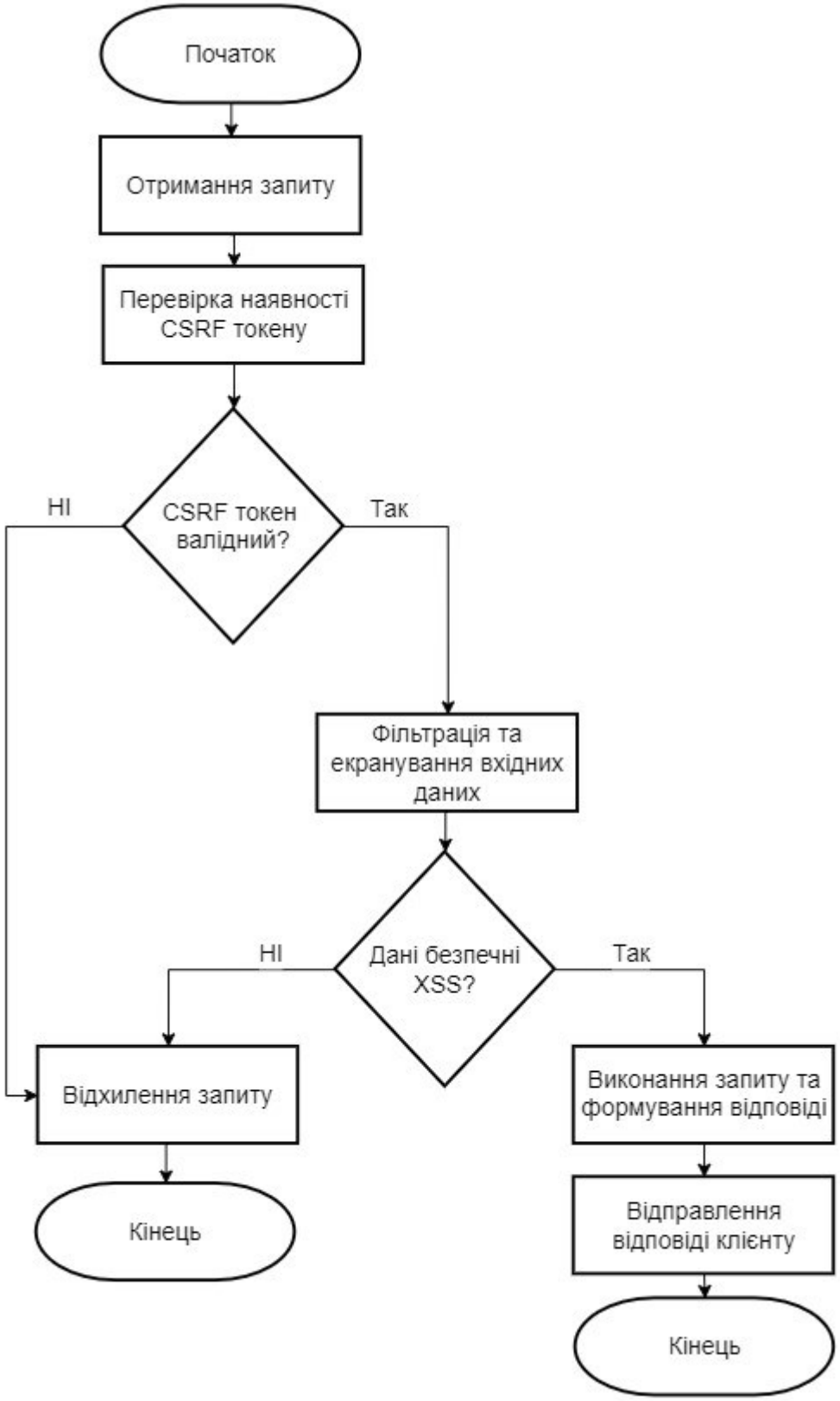


Рисунок 2.2 – Блок-схема алгоритму роботи підсистеми із захисту веб-додатків від атак CSRF і XSS

Крок 8. Відхилення запиту. Запит визнано потенційно небезпечним і його виконання відхиляється.

Крок 9. Відправлення відповіді клієнту. Сформована відповідь відправляється клієнту.

Крок. 10. Кінець

Отже, спроектована підсистема захисту веб-додатків від атак CSRF і XSS реалізує ефективні механізми перевірки наявності та валідності CSRF токенів, фільтрації та екранування вхідних даних, а також відхилення потенційно небезпечних запитів. Це дозволяє значно знизити ризики успішного проведення атак CSRF і XSS та підвищити загальну безпеку веб-додатку.

2.3. Проектування підсистеми захисту веб-додатків від від атак типу SQL- та PHP-ін'єкцій

Захист веб-додатків від атак типу SQL- та PHP-ін'єкцій є критично важливим для забезпечення безпеки та цілісності даних. Ці типи атак можуть призвести до несанкціонованого доступу, модифікації або знищення даних у базі даних, а також до виконання зловмисного коду на сервері.

У контексті запобігання SQL-ін'єкціям, важливими заходами є використання параметризованих запитів і підготовлених виразів. Цей підхід полягає у тому, щоб вхідні дані оброблювалися як параметри запиту, а не як частина самого SQLвиразу. Це ефективно усуває можливість інтерпретації введених даних як коду, що мінімізує ризик виконання шкідливого SQL.

Другим важливим заходом є обмеження прав доступу до бази даних. Це означає, що користувацькі облікові записи, які використовуються веб-додатками для доступу до баз даних, повинні мати лише мінімально необхідні права. Принцип "найменших привілеїв" допомагає зменшити потенційний збиток від успішної

SQL-ін'єкції. Наприклад, якщо додатку потрібно лише читати дані, обліковий запис бази даних не повинен мати прав на створення, оновлення або видалення даних.

Третім заходом є перевірка і санітація даних, що вводяться. Перед використанням введених даних в запитах, вони повинні пройти процедуру перевірки на відповідність очікуваному формату. Наприклад, числа мають бути перевірені на числовий тип, рядки - на відсутність спеціальних символів. Це знижує ризик того, що шкідливі дані будуть оброблені базою даних.

Алгоритм роботи підсистеми захисту від атак типу SQL- та PHP-ін'єкцій починається з отримання вхідного запиту від користувача. Далі виконується валідація та екранування вхідних даних запиту, щоб запобігти потенційним атакам через введення шкідливих даних.

Якщо підсистема налаштована на використання параметризованих запитів, формується параметризований запит до бази даних. В іншому випадку формується звичайний запит. Параметризовані запити дозволяють відокремити вхідні дані від структури запиту, що запобігає можливості виконання зловмисного SQL-коду.

Після виконання запиту до бази даних відбувається перевірка результатів на наявність підозрілих даних. Якщо результати запиту визнаються безпечними, відбувається їх подальша обробка. В іншому випадку здійснюється логування потенційної загрози та сповіщення адміністратора системи.

Якщо підсистема налаштована на використання підготовлених виразів, формується підготовлений вираз для виконання PHP-коду. В іншому випадку формується звичайний вираз. Підготовлені вирази дозволяють відокремити вхідні дані від структури PHP-коду, що запобігає можливості виконання зловмисного PHP-коду.

Після виконання PHP-коду відбувається перевірка на наявність підозрілих операцій. Якщо PHP-код визнається безпечним, формується відповідь на основі

результатів його виконання. В іншому випадку здійснюється логування потенційної загрози та сповіщення адміністратора системи.

Сформована відповідь відправляється клієнту лише у випадку успішного проходження всіх перевірок безпеки. Таким чином, підсистема захисту від атак типу SQL- та PHP-ін'єкцій забезпечує надійний захист веб-додатку та даних від несанкціонованого доступу та виконання зловмисного коду. Алгоритм роботи підсистеми із захисту веб-додатків від атак типу SQL- та PHP-ін'єкцій представлений на рисунку 2.3 у вигляді блок-схеми.

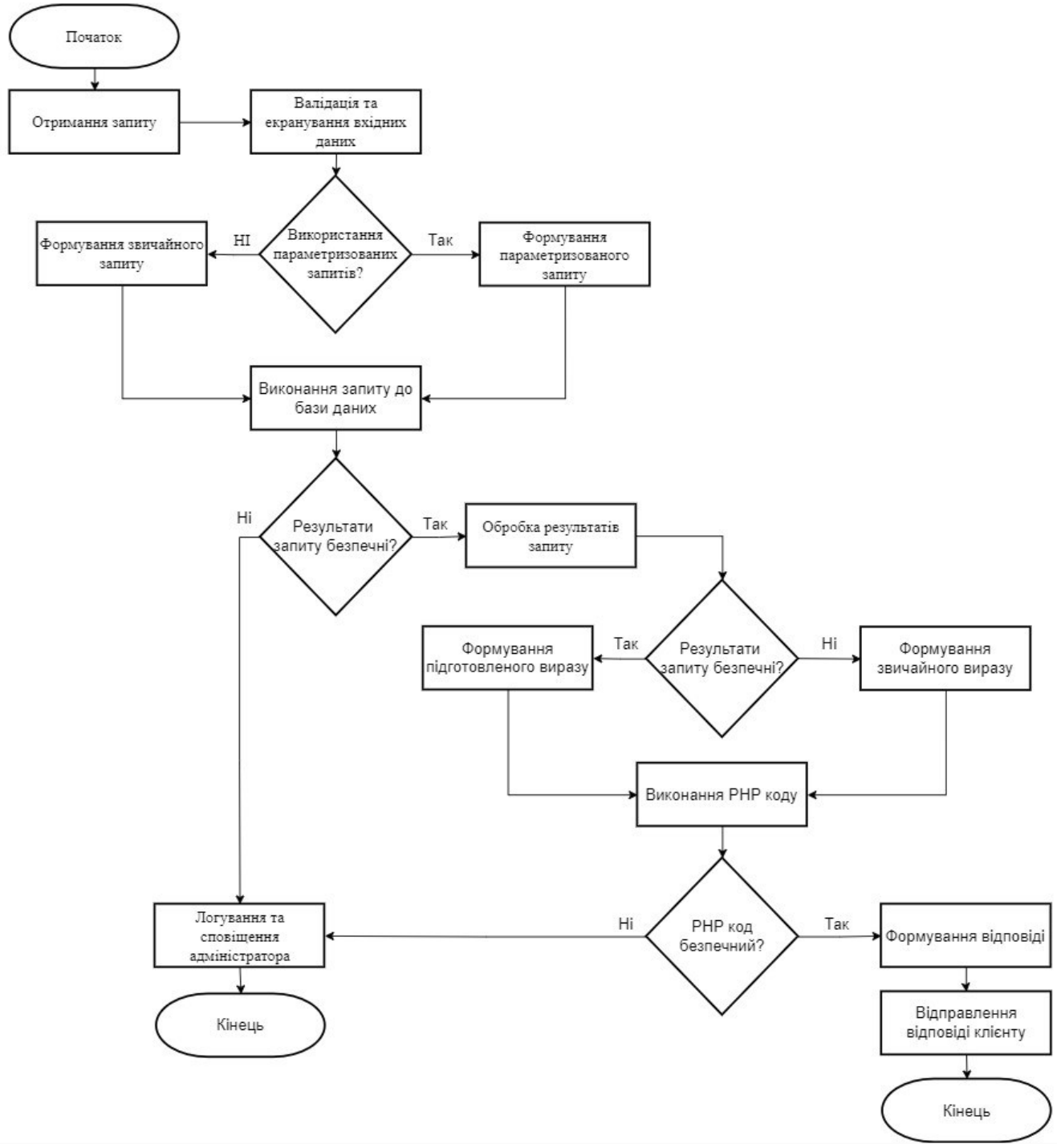


Рисунок 2.3 – Блок-схема алгоритму роботи підсистеми із захисту веб-додатків від атак типу SQL- та PHP-ін'єкцій

Крок 1. Початок.

Крок 2. Отримання запиту. Підсистема отримує вхідний запит від користувача.

Крок 3. Валідація та екранування вхідних даних. Виконується валідація та екранування вхідних даних запиту.

Крок 4. Використання параметризованих запитів? Якщо використовуються параметризовані запити, переходимо до кроку 5. Інакше, переходимо до кроку 6.

Крок 5. Формування параметризованого запиту. Формується параметризований запит до бази даних.

Крок 6. Формування звичайного запиту. Формується звичайний запит до бази даних.

Крок 7. Виконання запиту до бази даних. Виконується запит до бази даних (параметризований або звичайний).

Крок 8. Результати запиту безпечні? Якщо результати запиту визнано безпечними, переходимо до кроку 9. Інакше, переходимо до кроку 13.

Крок 9. Обробка результатів запиту. Виконується обробка отриманих результатів запиту.

Крок 10. Використання підготовлених виразів? Якщо використовуються підготовлені вирази, переходимо до кроку 11. Інакше, переходимо до кроку 13.

Крок 11. Формування підготовленого виразу. Формується підготовлений вираз для виконання PHP-коду.

Крок 12. Логування та сповіщення адміністратора. Виконується логування потенційної загрози та сповіщення адміністратора системи.

Крок 13. Формування звичайного виразу. Формується звичайний вираз для виконання PHP-коду.

Крок 14. Виконання PHP-коду. Виконується PHP-код (підготовлений або звичайний вираз).

Крок 15. PHP-код безпечний? Якщо PHP-код визнано безпечним, переходимо до кроку 17. Інакше, переходимо до кроку 13.

Крок 16. Формування відповіді. Формується відповідь на основі результатів виконання PHP-коду.

Крок 17. Відправлення відповіді клієнту. Сформована відповідь відправляється клієнту.

Крок 18. Кінець.

Підсумовуючи, спроектована підсистема захисту веб-додатків від атак типу SQL- та PHP-ін'єкцій використовує ефективні підходи, такі як параметризовані запити, підготовлені вирази, валідацію та екранування вхідних даних, а також механізми логування та сповіщення про потенційні загрози. Це дозволяє суттєво знизити ризики успішного проведення SQL- та PHP-ін'єкцій та забезпечити високий рівень безпеки веб-додатку та даних.

2.4. Проектування архітектури підсистеми веб-додатків від атак CSRF I XSS та атак типу SQL- та PHP -ін'єкцій

Проектування архітектури підсистеми захисту веб-додатків є критично важливим етапом у процесі розробки безпечного та надійного програмного забезпечення. Ретельний підхід до проектування архітектури дозволяє врахувати всі аспекти безпеки та забезпечити ефективний захист від різноманітних типів атак, таких як CSRF, XSS, SQL та PHP-ін'єкції.

Архітектура підсистеми захисту веб-додатків від атак CSRF, XSS, SQL та PHPін'єкцій складається з наступних ключових компонентів:

− модуль захисту від CSRF атак – цей модуль відповідає за перевірку наявності та валідності CSRF токенів у запитах, що надходять до веб-додатку. Він забезпечує захист від несанкціонованих дій, які можуть бути ініційовані зловмисниками через підроблені запити.

− модуль захисту від XSS атак – цей модуль виконує фільтрацію та екранування вхідних даних користувача, щоб запобігти виконанню зловмисного коду на стороні

клієнта. Він застосовує різні техніки, такі як кодування спеціальних символів та валідація введених даних, щоб мінімізувати ризики XSS атак.

− модуль захисту від SQL-ін'єкцій – цей модуль відповідає за захист від атак типу SQL-ін'єкцій, які можуть призвести до несанкціонованого доступу або модифікації даних у базі даних. Він використовує такі підходи, як параметризовані запити та екранування вхідних даних, щоб запобігти виконанню зловмисного SQLкоду;

− модуль захисту від PHP-ін'єкцій – цей модуль забезпечує захист від атак типу PHP-ін'єкцій, які можуть дозволити зловмисникам виконувати небажаний PHP-код на сервері. Він застосовує механізми, такі як фільтрація вхідних даних та використання підготовлених виразів, щоб запобігти виконанню зловмисного PHPкоду;

− модуль журналювання та сповіщення – цей модуль відповідає за реєстрацію та журналювання подій, пов'язаних з безпекою, таких як спроби атак або підозріла активність. Він також забезпечує механізми сповіщення адміністраторів системи про потенційні загрози, що дозволяє оперативно реагувати на інциденти безпеки;

− модуль конфігурації та налаштувань – цей модуль дозволяє гнучко налаштовувати параметри безпеки підсистеми захисту, такі як рівні фільтрації, політики безпеки та інші конфігураційні опції. Він забезпечує можливість адаптації підсистеми до специфічних вимог веб-додатку та середовища розгортання.

Окрім зазначених модулів, невід'ємною частиною підсистеми захисту є модуль інтерфейсу користувача. Цей модуль надає зручний та інтуїтивно зрозумілий інтерфейс для користувачів, які бажають протестувати свій веб-додаток на наявність вразливостей до атак CSRF, XSS, SQL та PHP-ін'єкцій.

Інтерфейс надає можливість налаштовувати різні параметри тестування, такі як рівень агресивності перевірок, вибір типів атак для тестування та інші опції.

Користувачі можуть вводити різноманітні дані, такі як URL веб-додатку, параметри запитів, дані форм тощо, для перевірки наявності вразливостей.

Користувачі можуть ініціювати процес тестування веб-додатку на наявність вразливостей. Після завершення тестування інтерфейс відображає детальні результати, включаючи знайдені вразливості, рекомендації щодо їх усунення та інші релевантні дані.

Модуль інтерфейсу користувача дозволяє генерувати звіти про результати тестування, які містять детальну інформацію про виявлені вразливості, їх опис та рекомендації щодо їх виправлення.

Правильно спроектована архітектура підсистеми захисту значно знижує ризики успішного проведення атак та підвищує загальну безпеку веб-додатків. (рис. 2.4).

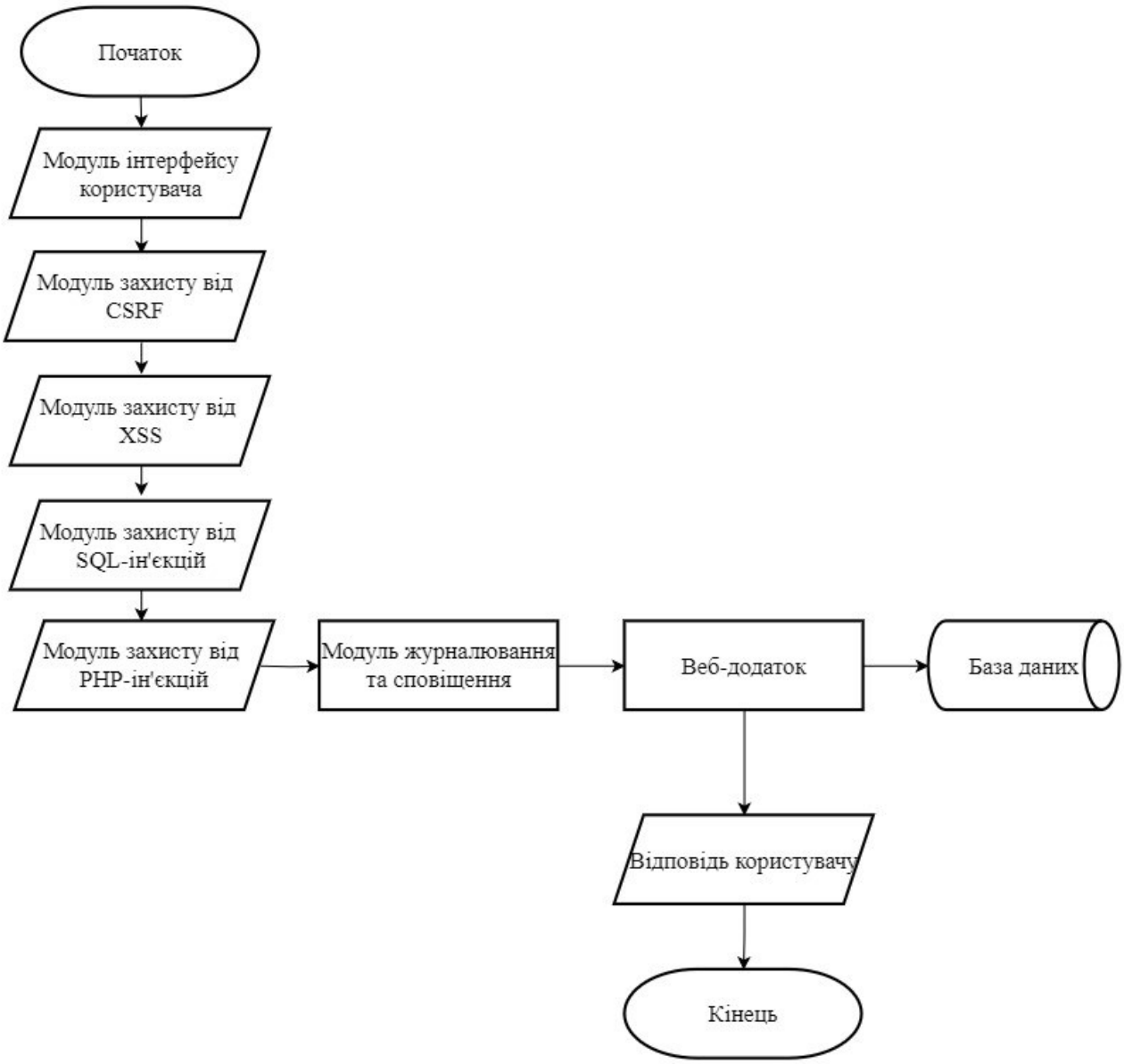


Рисунок 2.4 – Схема взаємодії компонентів підсистеми захисту від атак CSRF, XSS, SQL- та PHP-ін'єкцій

Крок 1. Початок.

Користувач взаємодіє з модулем інтерфейсу користувача підсистеми захисту веб-додатків.

Крок 2. Модуль інтерфейсу користувача дозволяє користувачу вводити дані для тестування, налаштовувати параметри тестування та ініціювати процес перевірки веб-додатку на наявність вразливостей.

Крок 3. Введені користувачем дані передаються до модуля захисту від CSRF. Цей модуль перевіряє наявність та валідність CSRF токенів у запитах, щоб запобігти несанкціонованим діям.

Крок 4. Після проходження перевірки CSRF, дані передаються до модуля захисту від XSS. Цей модуль виконує фільтрацію та екранування вхідних даних, щоб запобігти виконанню зловмисного коду на стороні клієнта.

Крок 5. Дані, що пройшли перевірку на XSS, передаються до модуля захисту від SQL-ін'єкцій. Цей модуль застосовує механізми, такі як параметризовані запити та екранування даних, для запобігання несанкціонованому доступу або модифікації даних у базі даних.

Крок 6. Після проходження перевірки на SQL-ін'єкції, дані передаються до модуля захисту від PHP-ін'єкцій. Цей модуль виконує фільтрацію та перевірку даних, щоб запобігти виконанню зловмисного PHP-коду на сервері.

Крок 7. Дані, що пройшли всі перевірки безпеки, передаються до модуля журналювання та сповіщення. Цей модуль реєструє події, пов'язані з безпекою, та за необхідності надсилає сповіщення адміністраторам системи.

Крок 8. Після проходження всіх перевірок безпеки, дані надходять до самого веб-додатку для подальшої обробки.

Крок 9. Веб-додаток взаємодіє з базою даних для виконання необхідних операцій та отримання даних.

Крок 10. Веб-додаток формує відповідь на основі отриманих даних та результатів обробки.

Крок 11. Сформована відповідь повертається користувачу через модуль інтерфейсу користувача.

Крок 12. Користувач отримує відповідь з результатами тестування, які можуть включати інформацію про виявлені вразливості, рекомендації щодо їх усунення та інші релевантні дані.

Крок 13. Кінець.

Ця покрокова блок-схема ілюструє послідовність взаємодії між різними компонентами архітектури підсистеми захисту веб-додатків. Кожен модуль відповідає за певний аспект безпеки і передає дані далі по ланцюжку після успішного проходження перевірок. Такий підхід забезпечує комплексний захист веб-додатку від різних типів атак та дозволяє ефективно виявляти та запобігати потенційним вразливостям.

Клієнтська частина складається з декількох ключових елементів. Головним файлом є компонент, який слугує точкою входу для клієнтської частини та містить основну структуру веб-сторінки. Для реалізації інтерактивності та динамічної поведінки ми використовуємо компоненти React, які розміщені в директорії components. Кожен компонент відповідає за певний функціонал та взаємодію з користувачем (рис. 2.5).

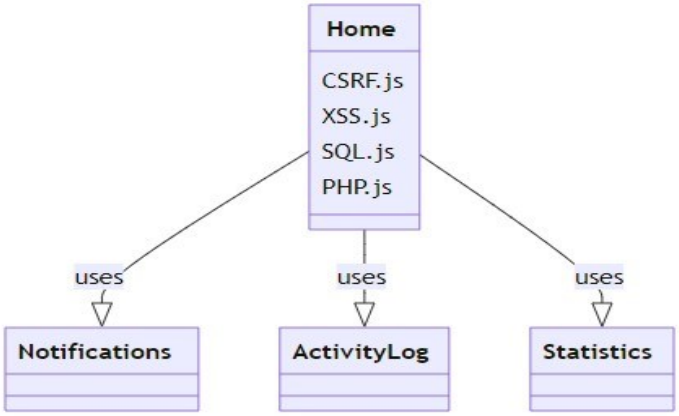


Рис 2.5 – Архітектури клієнтської частини підсистеми захисту веб-додатків

Крок 1. Home – головна сторінка – центральний компонент клієнтської частини, який забезпечує навігацію до модулів тестування безпеки та взаємодіє з допоміжними компонентами для відображення сповіщень, ведення журналу активності та надання статистичних даних.

Крок 2. CSRF.js це – модуль тестування CSRF вразливостей – компонент, який відповідає за перевірку наявності вразливостей до атак Cross-Site Request Forgery шляхом імітації шкідливих запитів та аналізу відповідей сервера.

Крок 3. XSS.js – модуль тестування XSS вразливостей – компонент, який виконує тестування наявності вразливостей до атак Cross-Site Scripting шляхом введення потенційно небезпечних скриптів та перевірки їх виконання на сторінках веб-додатку.

Крок 4. SQL.js – модуль тестування SQL-ін'єкцій – компонент, який перевіряє стійкість веб-додатку до атак типу SQL-ін'єкцій шляхом введення спеціально сформованих SQL-запитів та аналізу відповідей бази даних.

Крок 5. PHP.js – модуль тестування PHP-ін'єкцій – компонент, який тестує наявність вразливостей до атак типу PHP-ін'єкцій шляхом введення шкідливого PHP-коду та перевірки його виконання на серверній частині веб-додатку.

Крок 6. Notifications – модуль сповіщень – допоміжний компонент, який відображає сповіщення та повідомлення користувачеві про результати тестування, виявлені вразливості та інші важливі події.

Крок 7. ActivityLog (Модуль журналу активності) - допоміжний компонент, який веде журнал активності користувача, фіксуючи виконані тести, їх результати та інші дії користувача в системі.

Крок 8. Statistics (Модуль статистики) - допоміжний компонент, який надає статистичні дані та аналітику щодо результатів тестування, кількості виявлених вразливостей, прогресу тестування тощо.

Важливим модулем підсистеми є серверна частина, що виконує захист вебдодатків (рис. 2.6).

Крок 1. Server (Сервер) – центральний компонент серверної частини, який ініціалізує веб-сервер, налаштовує середовище виконання та підключає необхідні модулі та маршрути для обробки вхідних запитів.

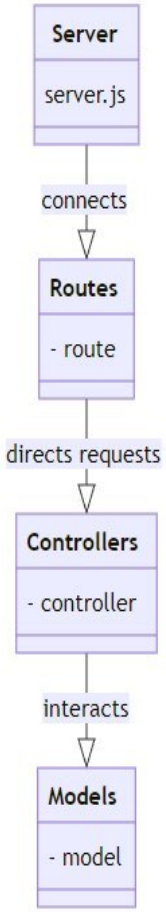


Рис 2.6 – Архітектура компонентів серверної частини підсистеми захисту вебдодатків

Крок 2. Routes (Маршрутизація) – компонент, який відповідає за визначення маршрутів URL та зіставлення їх з відповідними контролерами. Він направляє вхідні запити до відповідних обробників на основі визначених шляхів та методів HTTP.

Крок 3. Controllers (Контролери) – компонент, який містить логіку обробки вхідних запитів та формування відповідей. Контролери отримують дані від клієнтської частини, виконують необхідні операції, взаємодіють з моделями даних та повертають результати обробки у вигляді відповідей.

Крок 4. Models (Моделі даних) – компонент, який представляє структуру та поведінку об'єктів даних у системі. Моделі визначають схему даних, методи

доступу та маніпулювання даними, а також забезпечують взаємодію з базою даних для збереження та отримання інформації.

Крок 5. Взаємодія між компонентами:

- сервер ініціалізує веб-сервер та підключає маршрути;
- маршрутизація отримує вхідні запити та направляє їх до відповідних контролерів;
- контролери обробляють запити, взаємодіють з моделями даних для отримання або збереження інформації та формують відповіді;
- моделі даних забезпечують доступ до даних та їх структуру, взаємодіючи з базою даних для виконання операцій читання та запису.

Архітектура компонентів серверної частини демонструє взаємодію між різними модулями та компонентами, забезпечуючи чітке розділення обов'язків та можливість масштабування та розширення функціональності системи. Вона відображає архітектурний підхід, який дозволяє ефективно обробляти вхідні запити, взаємодіяти з базою даних та формувати відповіді для забезпечення надійного та безпечного функціонування веб-додатку.

Таким чином, ретельне проектування архітектури підсистеми захисту вебдодатків від атак CSRF, XSS, SQL та PHP-ін'єкцій є ключовим фактором у забезпеченні ефективного та надійного захисту. Модульний підхід, що включає окремі компоненти для кожного типу атак, а також модуль інтерфейсу користувача для зручного тестування, дозволяє створити комплексну та гнучку систему безпеки.

2.5. Висновки до розділу 2

У даному розділі розглянуто ключові аспекти проектування та реалізації підсистеми захисту веб-додатків від атак типу SQL- та PHP-ін'єкцій. Шляхом аналізу і обговорення різних заходів запобігання цим видам атак, стало очевидним,

що ефективний захист вимагає комплексного підходу, який охоплює різноманітні аспекти розробки програмного забезпечення.

У контексті запобігання SQL-ін'єкціям було продемонстровано, що використання параметризованих запитів, обмеження прав доступу до бази даних та перевірка введених даних є ефективними стратегіями. Стратегії, такі як обмеження прав доступу, допомагають уникнути можливих наслідків успішної атаки, тоді як використання параметризованих запитів знижує ризик вразливості додатку до SQL-ін'єкцій шляхом уникнення виконання зловмисного SQL-коду.

У випадку захисту від PHP-ін'єкцій, вимкнення небезпечних функцій та використання вбудованих засобів безпеки PHP можуть допомогти запобігти можливій експлуатації вразливостей.

Загальною висновок з розділу є те, що ефективний захист від атак типу SQL- та PHP-ін'єкцій вимагає комплексного підходу, який включає як технічні заходи (такі як використання параметризованих запитів), так і процесуальні політики (такі як регулярний аудит коду та навчання розробників). Такий підхід дозволяє створити міцні та надійні веб-додатки, які захищені від найбільш поширених видів атак.

3. РОЗРОБКА ПІДСИСТЕМИ ЗАХИСТУ ВЕБ-ДОДАТКІВ

3.1. Обгрунтування вибору інструментів та технологій для розробки підсистеми захисту

Вибір інструментів та технологій для розробки підсистеми захисту вебдодатків є результатом уважного розгляду кількох ключових аспектів. Одним із найважливіших факторів у виборі є їхні можливості забезпечувати надійний захист від різноманітних загроз. Крім того, гнучкість в інтеграції та підтримка спільноти також мають велике значення. На останньому етапі, вибір технологій повинен

відповідати вимогам стандартів безпеки та враховувати найкращі практики у відповідній галузі (табл. 3.1).

Таблиця 3.1 – Обґрунтування вибору інструментів та технологій для розробки

Інструмент / Технологія	Переваги	Недоліки
Node.js та Express	- висока продуктивність та асинхронність	- потребує додаткових зусиль для забезпечення безпеки
	- зручні інструменти для маршрутизації та проміжної обробки	
Django (Python)	- вбудовані засоби безпеки	- монолітна структура може ускладнювати масштабування
	- широка спільнота та екосистема	
Ruby on Rails	- конвенції та вбудовані практики безпеки	- швидкість роботи може бути нижчою порівняно з Node.js
	- швидка розробка через конвенції	
Helmet, csurf, XSS-clean	- забезпечення базової безпеки на рівні HTTP-заголовків	- потребують додаткової конфігурації та налаштування
	- менша схильність до SQL-ін'єкцій	
Content Security Policy	- ефективний механізм запобігання XSS-атакам	- потребує ретельної конфігурації та тестування

Отже, поєднання Node.js та Express для серверної частини, бібліотек Helmet, csurf та XSS-clean для базової безпеки, MongoDB та Mongoose для бази даних, а також використання Content Security Policy є оптимальним вибором для розробки підсистеми захисту веб-додатків. Ці інструменти та технології забезпечують високу продуктивність, гнучкість, надійний захист від поширених загроз та відповідність найкращим практикам безпеки у веб-розробці. Node.js та Express обрано через їхню високу продуктивність, зручність маршрутизації та проміжної обробки. Helmet, csurf та XSS-clean використовуються для стандартизації захисних заходів та

зниження ймовірності помилок. MongoDB та Mongoose забезпечують гнучкість, масштабованість та додатковий рівень безпеки через валідацію даних та параметризовані запити. Content Security Policy є однією з найкращих практик у сфері веб-безпеки для запобігання XSS-атакам.

Ці інструменти та технології обрані на основі їхньої здатності задовольняти високі вимоги до безпеки веб-додатків, підтримки активної спільноти та легкості інтеграції з наявною архітектурою застосунку. Їх використання надає надійні засоби для створення безпечних та ефективних веб-додатків, забезпечуючи відповідність до сучасних стандартів безпеки

Отже, вибір зазначених технологій та інструментів для розробки підсистеми захисту веб-додатків є наслідком уважного огляду їхніх характеристик і можливостей, спрямованих на забезпечення максимального рівня безпеки та ефективності веб-додатків.

3.2. Розробка компонентів захисту від CSRF і XSS атак

Цей код представляє компонент, який реалізує функціональність для тестування SQL-ін'єкцій. Компонент містить форму з текстовим полем введення та кнопкою "Test". Користувач може ввести потенційно небезпечний SQL-запит у текстове поле. Після натискання кнопки "Test" введені дані відправляються на сервер через POST-запит на маршрут '/api/sql-test'. Сервер обробляє запит, виконує тестування на наявність SQL-ін'єкцій та повертає результат. Результат відображається користувачеві за допомогою функції alert.

```
// ...

 return (
<div>
      <h2>SQL Injection Simulator</h2>
      <form onSubmit={handleSubmit}>
```

```
        <input
type="text"
value={input}
            onChange={(e) => setInput(e.target.value)}
placeholder="Enter your input"
        />
        <button type="submit">Test</button>
      </form>
   </div>
);
```

Цей фрагмент коду визначає компонент SQL, який є частиною інтерфейсу користувача для тестування SQL-ін'єкцій. Він використовує хук useState для створення стану input, який зберігає введені користувачем дані.

Компонент відображає заголовок "SQL Injection Simulator" та форму з текстовим полем введення і кнопкою "Test". Коли користувач вводить дані в текстове поле, функція onChange викликається та оновлює стан input за допомогою функції setInput. Це дозволяє зберігати введені користувачем дані в стані компонента.

Коли користувач натискає кнопку "Test", викликається функція handleSubmit, яка відправляє введені дані на сервер для тестування на наявність SQL-ін'єкцій.

```
const handleSubmit = (event) => {
event.preventDefault();
fetch('/api/sql-test', {
method: 'POST',
headers: {
'Content-Type': 'application/json'
},
body: JSON.stringify({input})
}).then(response => response.text())
.then(data => {
alert(SQL test result: ${data});
});
};
```

Функція handleSubmit викликається при відправленні форми. Вона запобігає стандартній поведінці відправки форми за допомогою event.preventDefault().

Потім функція відправляє POST-запит на маршрут '/api/sql-test' за допомогою функції fetch. Введені користувачем дані передаються в тілі запиту у форматі JSON.

Після отримання відповіді від сервера, функція then викликається для обробки відповіді. Вона перетворює відповідь у текстовий формат за допомогою response.text(), а потім відображає результат тестування за допомогою функції alert.

Цей компонент забезпечує інтерфейс користувача для введення потенційно небезпечних SQL-запитів та відправки їх на сервер для тестування. Результат тестування відображається користувачеві, що дозволяє перевірити ефективність реалізованих методів захисту від SQL-ін'єкцій.

3.3. Розробка компонентів захисту від SQL- та PHP-ін’єкцій

SQL-і PHP-ін'єкції є одними з найнебезпечніших видів атак на веб-додатки. Вони дозволяють зловмисникам виконувати довільні SQL-запити та PHP-код, що може призвести до витоку даних, видалення даних або захоплення контролю над сервером. У цьому розділі ми розглянемо методи захисту, які використовуються в нашому проекті для запобігання SQL-і PHP-ін'єкцій.

```
const express = require('express'); const
bodyParser = require('body-parser');
const { Client } = require('pg');
const app = express();
app.use(bodyParser.json());
// Конфігурація підключення до бази даних
PostgreSQL const client = new Client({ user:
'your_username', host: 'localhost', database:
'your_database', password: 'your_password',
port: 5432,
});
client.connect();
```

Маршрут для тестування SQL-ін'єкцій та // Екранування введених даних для запобігання SQL-ін'єкціям.

```
app.post('/api/sql-test', (req, res) => {
const userInput = req.body.input;
const sanitizedInput = userInput.replace(/'/g, "''");
```

```
const query = SELECT * FROM users WHERE username =
'${sanitizedInput}'; client.query(query, (err, result) => { if (err) {
console.error('Error executing query', err);
res.status(500).send('Internal Server Error');
} else {
if (result.rows.length > 0) {
res.send('SQL Injection Vulnerability Detected');
} else {
res.send('No SQL Injection Detected');
}
}
});
});
// Запуск сервера
app.listen(3000, () => {
console.log('Server is running on port 3000');
});
```

Цей код демонструє базовий підхід до екранування введених даних та виконання запитів до бази даних. У реальному додатку потрібно також реалізувати додаткові заходи безпеки, такі як використання параметризованих запитів та обмеження прав доступу до бази даних.

```
const {Client} = require('pg');
const client = new Client();

async function getUserById(userId) {
   const query = 'SELECT * FROM users WHERE id = $1';
const values = [userId];
    try
{
      await client.connect();
      const res = await client.query(query, values);
console.log(res.rows[0]);
   } catch (err) {
      console.error(err.stack);
   } finally {
      await client.end();
   }
}
```

PHP-ін'єкції виникають, коли введення користувача включається в PHP-код і виконується без перевірки і екранування. Для захисту від таких атак ми використовуємо такі методи:

Функція eval PHP дозволяє виконувати довільний код, що може бути дуже небезпечно. У нашому проекті ми уникаємо використання eval та замінюємо його безпечними альтернативами.

```
$input = $_POST['user_input'];
$safe_input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

Цей код отримує введені користувачем дані з масиву $_POST['user_input'] і зберігає їх у змінній $input. Потім він використовує функцію htmlspecialchars() для екранування спеціальних символів HTML у введених даних. Параметр ENT_QUOTES вказує на необхідність екранування як одинарних, так і подвійних лапок, а параметр 'UTF-8' задає кодування символів. Екрановане введення зберігається у змінній $safe_input.

```
$ mysqli = new mysqli ( " localhost " , " user " , " password " , " database " );
$safe_input = $mysqli->real_escape_string($safe_input);

$query = "SELECT * FROM users WHERE input='$safe_input'";
$result = $mysqli->query($query);
import React, { useState } from 'react';

function SQL() {     const [input,
setInput] = useState('');

   const handleSubmit = (event) =>
{        event.preventDefault();
fetch('/api/sql-test', {
method: 'POST',            headers: {
             'Content-Type': 'application/json'
          },
          body: JSON.stringify({input})
}).then(response => response.text())
       .then(data => {
          alert(`SQL test result: ${data}`);
       });
   };
```

Цей код визначає компонент React з назвою SQL. Він використовує хук useState для створення стану input, який зберігає введені користувачем дані.

```
    return
(
<div>
        <h2>SQL Injection Simulator</h2>
        <form onSubmit={handleSubmit}>
          <input
type="text"
value={input}
              onChange={(e) => setInput(e.target.value)}
placeholder="Enter your input"
          />
          <button type="submit">Test</button>
        </form>
      </div>
  );
}
```

Компонент відображає форму з текстовим полем введення та кнопкою "Test". Коли користувач вводить дані, функція onChange викликається та оновлює стан input за допомогою функції setInput.

При відправленні форми викликається функція handleSubmit. Вона відправляє POST-запит на маршрут '/api/sql-test' з введеними даними у форматі JSON. Після отримання відповіді від сервера, результат тестування відображається за допомогою функції alert.

```
 export default

SQL;


import React, { useState } from 'react';

function SQL() {     const [input,
setInput] = useState('');


   const handleSubmit = (event) =>
{        event.preventDefault();
fetch('/api/sql-test', {
method: 'POST',            headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({input})
}).then(response => response.text())
      .then(data => {
        alert(`SQL test result: ${data}`);
      });
  };
```

Цей фрагмент забезпечує частину інтерфейсу користувача для введення потенційно небезпечних SQL-запитів та відправки їх на сервер для тестування.

```
    return
(
<div>
        <h2>SQL Injection Simulator</h2>
        <form onSubmit={handleSubmit}>
          <input
type="text"
value={input}
              onChange={(e) => setInput(e.target.value)}
placeholder="Enter your input"
          />
          <button type="submit">Test</button>
        </form>
      </div>
   );
}

export default SQL;
```

Використання підготовлених запитів, екранування введення користувача та обмеження використання функції eval допомагають значно підвищити безпеку вебпрограми.

Ці частини коду демонструють різні аспекти захисту від SQL-ін'єкцій, включаючи екранування введених даних на стороні сервера (PHP) та використання параметризованих запитів з екануванням в SQL-запитах (MySQLi). Також наведено приклад клієнтського компонента React для тестування SQL-ін'єкцій шляхом відправки введених даних на сервер.

3.4. Тестування розробленої підсистеми захисту веб-додатків

Головна сторінка розробленої підсистеми, що зображена на рисунку 3.1 є відправною точкою для користувачів, які бажають протестувати розроблену підсистему захисту веб-додатків.

На сторінці представлено навігаційне меню, що містить посилання на різні модулі для тестування атак, такі як CSRF, XSS, SQL–ін’єкцій та PHP–ін’єкцій. Окрім того доступні кнопки «Журнал активност», «Статистика та аналітика», «Інформація про вразливості». Користувач може обрати потрібний тип атаки чи іншу функцію ПЗ натиснувши на відповідну кнопку.

Рисунок 3.1 – Вигляд головної сторінки розробленої підсистеми

Ці розділи дозволяють користувачеві відстежувати результати тестування та отримувати корисні відомості про виявлені вразливості.

Сторінка "Симулятор XSS–атак" (рис. 3.2) призначена для тестування вразливостей веб-додатку до XSS–атак.

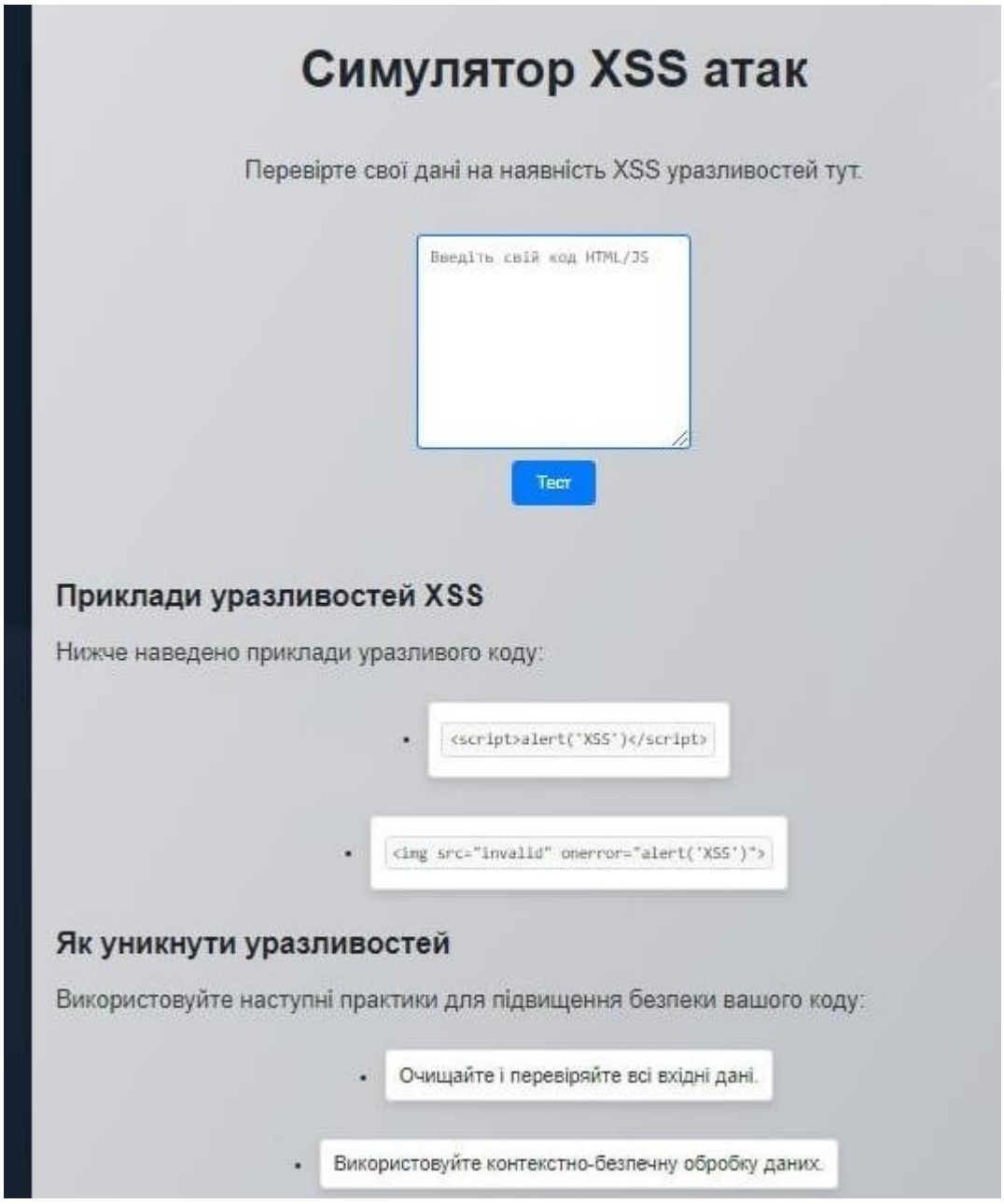


Рисунок 3.2 – Сторінка тестування XSS-атак

Сторінка "Симулятор атаки CSRF" (рис. 3.3) призначена для тестування вразливостей веб-додатку до CSRF-атак. Користувач може ввести свій запит у

текстове поле та натиснути кнопку "Test" для перевірки наявності CSRFвразливостей.

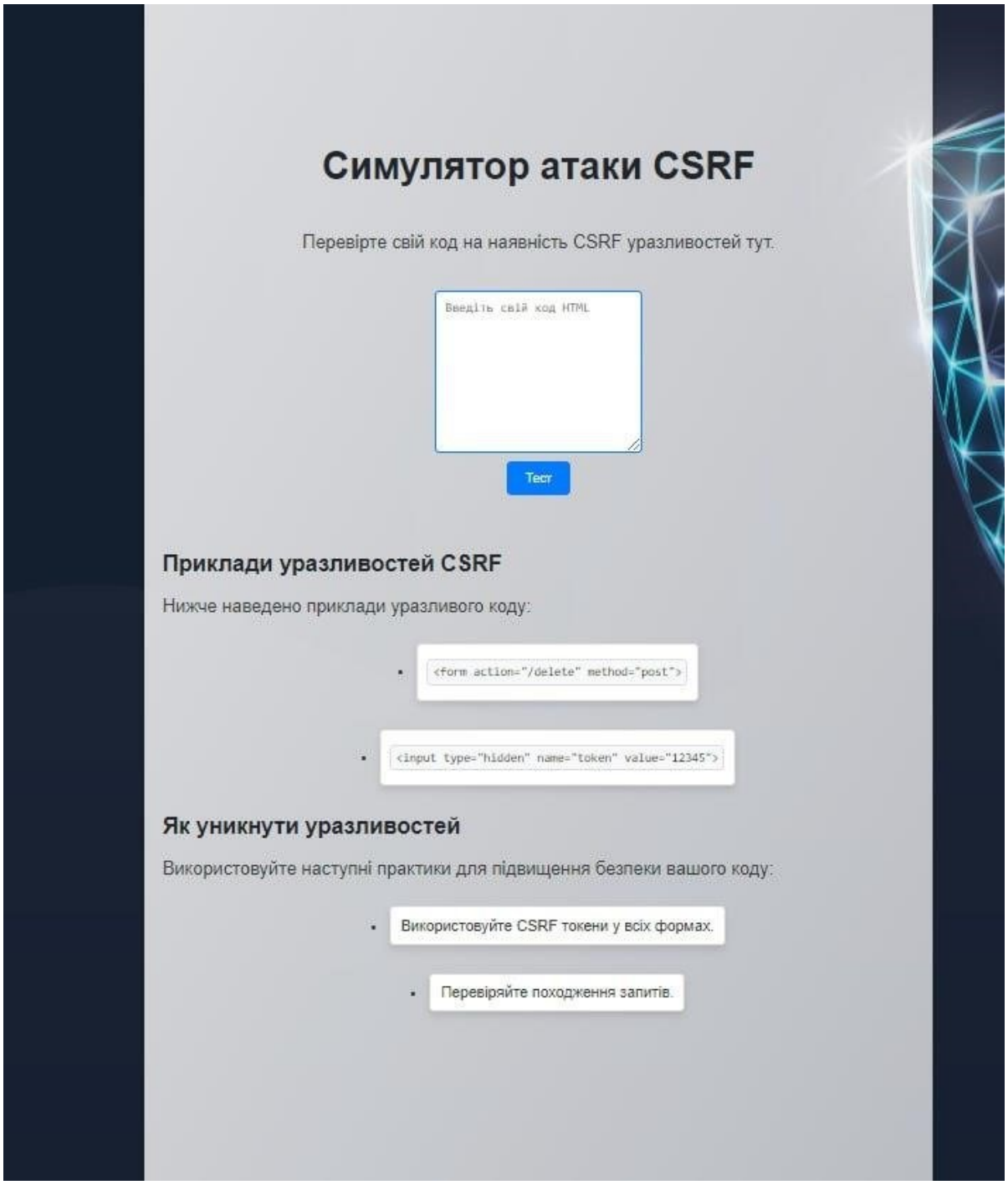


Рисунок 3.3 – Сторінка тестування CSRF-атак

Підсистема захисту веб-додатку оброблює введений запит і відображає результат тестування. Якщо введений запит не містить CSRF-вразливостей, користувач побачить повідомлення про успішне проходження тесту (рис. 3.4).

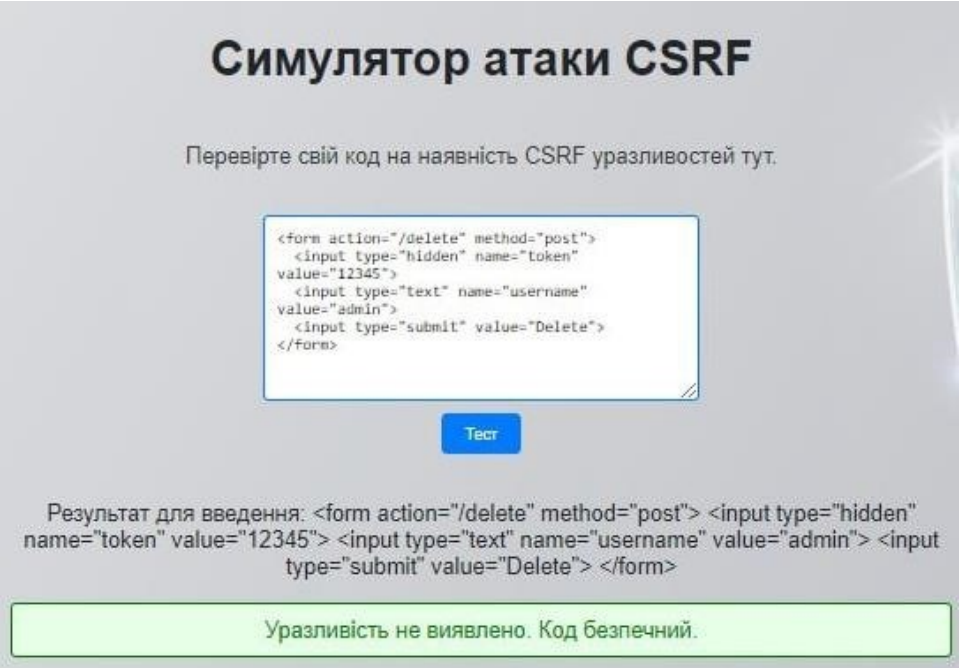


Рисунок 3.4 – Результат проведеного тестування вразливостей веб-додатку на CSRF-атак

У разі виявлення вразливості, користувач отримує відповідне повідомлення та рекомендації щодо її усунення. (рис. 3.5).

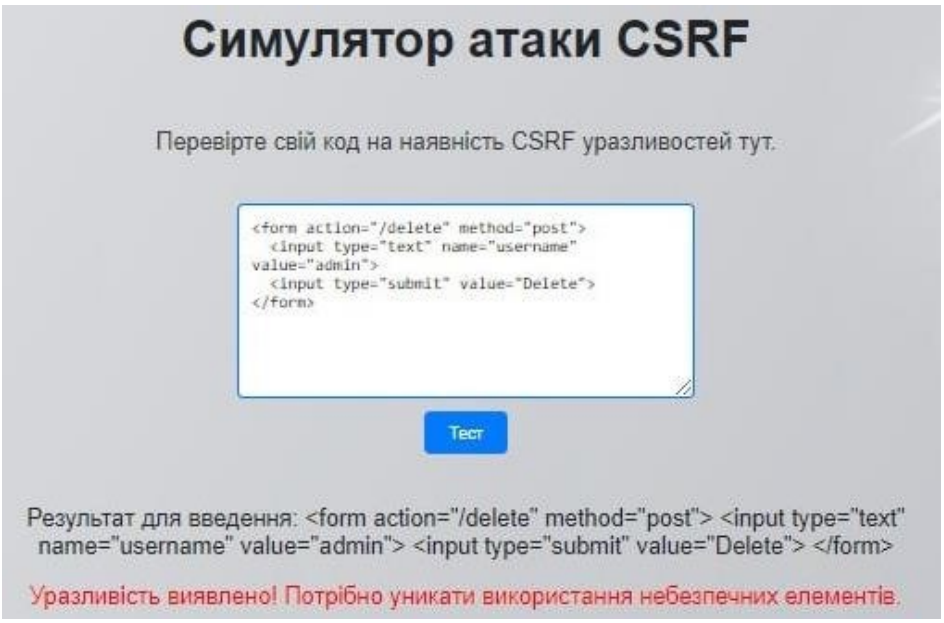


Рисунок 3.5 – Результат тестування вразливостей веб-додатку на CSRF-атак

Наступна сторінка дозволяє перевірити стійкість веб-додатку до атак типу SQLін'єкцій. Користувач може ввести потенційно небезпечний SQL-запит у

текстове поле та натиснути кнопку "Test" для виконання тесту. Підсистема захисту вебдодатку аналізує введений запит на наявність SQL-ін'єкцій та відображає результат перевірки (рис. 3.6).

Якщо запит містить потенційну вразливість, користувач отримає повідомлення про можливу SQL-ін'єкцію та рекомендації щодо її запобігання. У разі відсутності вразливостей, користувач побачить повідомлення про успішне проходження тесту.

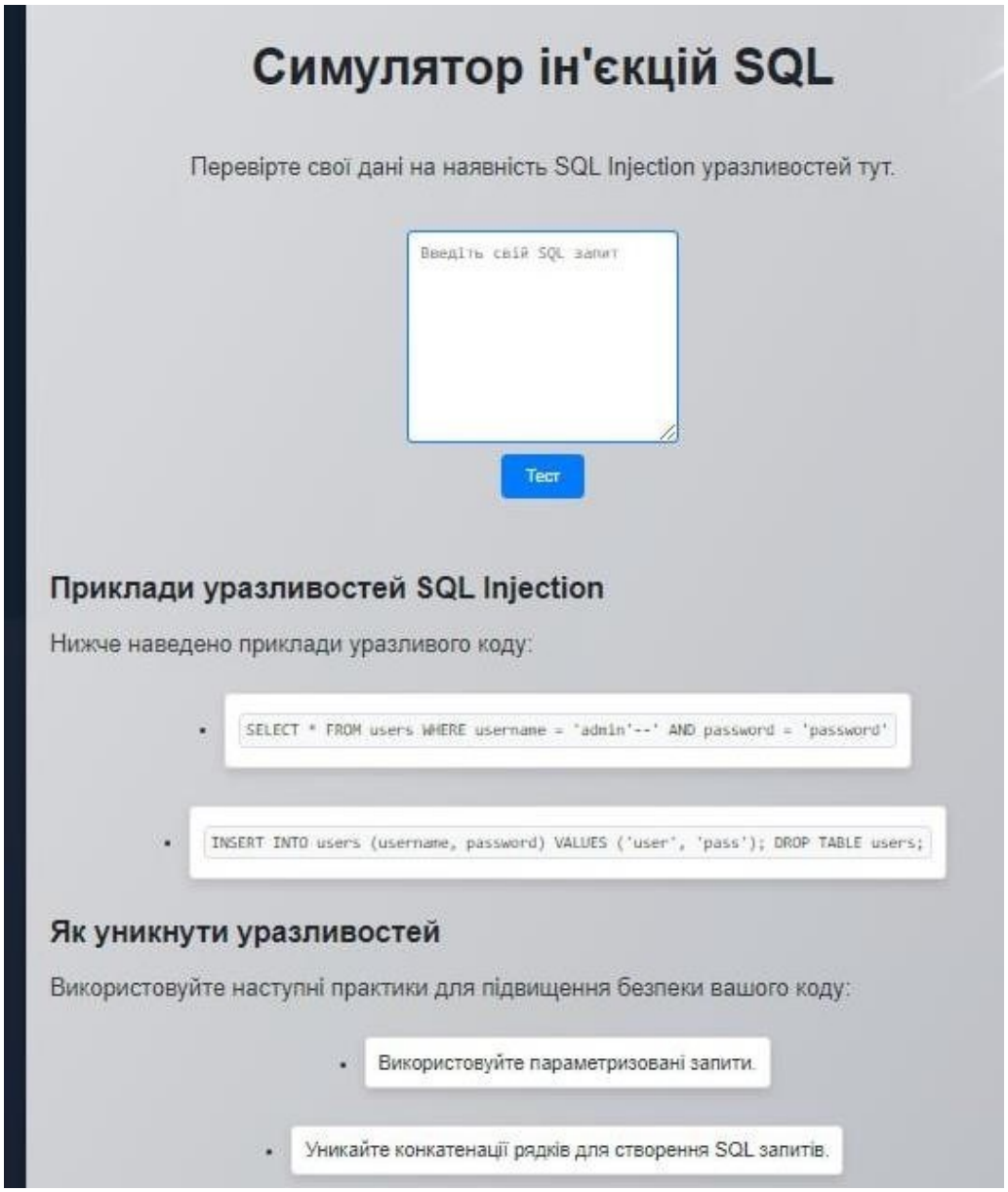


Рисунок 3.6 – Сторінка тестування SQL-ін'єкцій

Підсистема захисту веб-додатку аналізує введений запит на наявність SQLін'єкцій та відображає результат перевірки. Якщо запит містить потенційну вразливість, користувач отримає повідомлення про можливу SQL-ін'єкцію та рекомендації щодо її запобігання. У разі відсутності вразливостей, користувач побачить повідомлення про успішне проходження тесту.

Підсистема захисту веб-додатку аналізує введений запит на наявність PHP ін'єкцій та відображає результат перевірки. Якщо запит містить потенційну вразливість, користувач отримає повідомлення про можливу PHP -ін'єкцію та рекомендації щодо її запобігання (рис. 3.7).

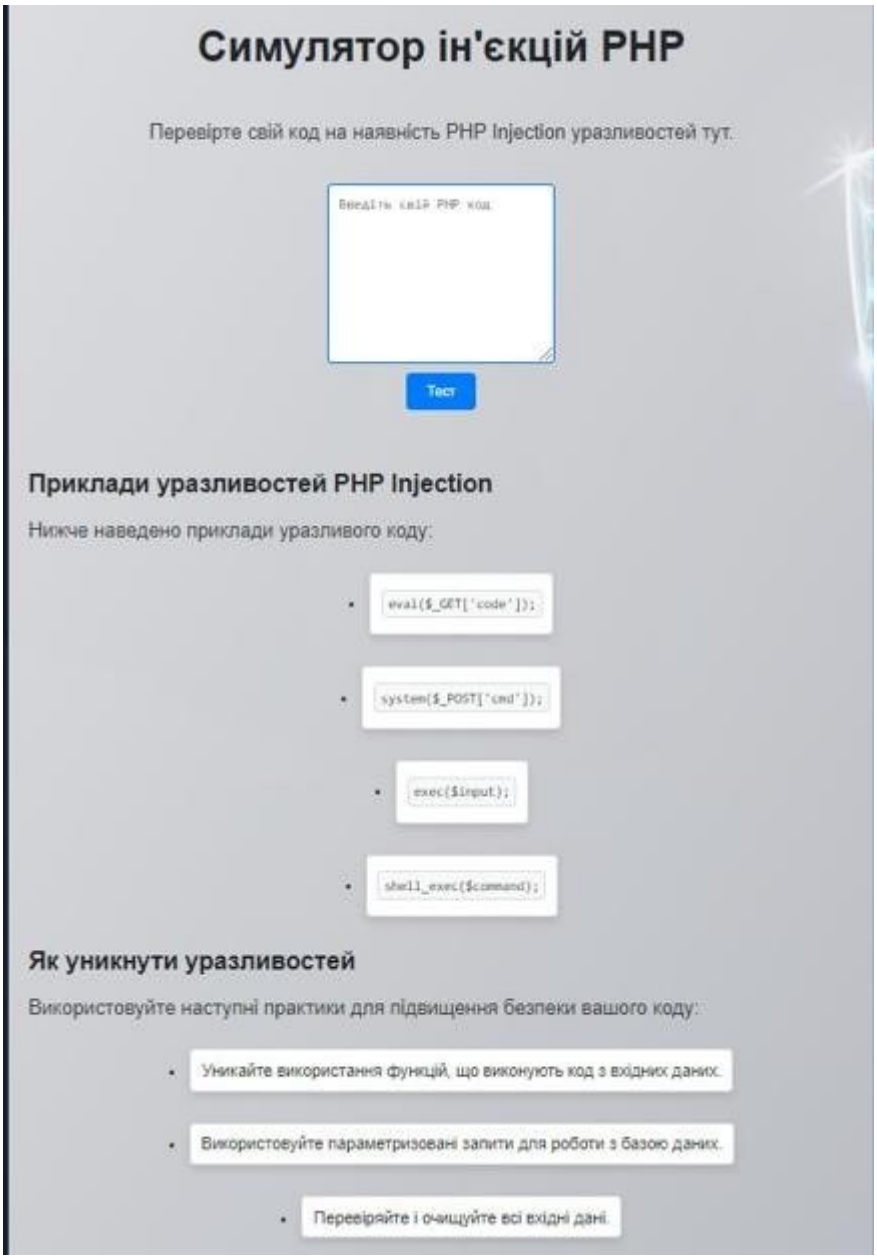


Рисунок 3.7 – Сторінка демонстрації вікна захисту від PHP ін’єкцій

Підсистема захисту веб-додатку проаналізує введені дані та визначить, чи містять вони потенційну вразливість до PHP-ін'єкцій, у разі відсутності вразливостей, користувач побачить повідомлення про успішне проходження тесту.

Наступна сторінка відображає сповіщення щодо результатів проведених тестів. На зображенні показано, що в результаті тестування виявлено вразливість типу

CSRF, але не знайдено вразливості XSS. Крім того, наведено повідомлення про те, що в даний момент немає доступних оновлень для додатку (3.8).

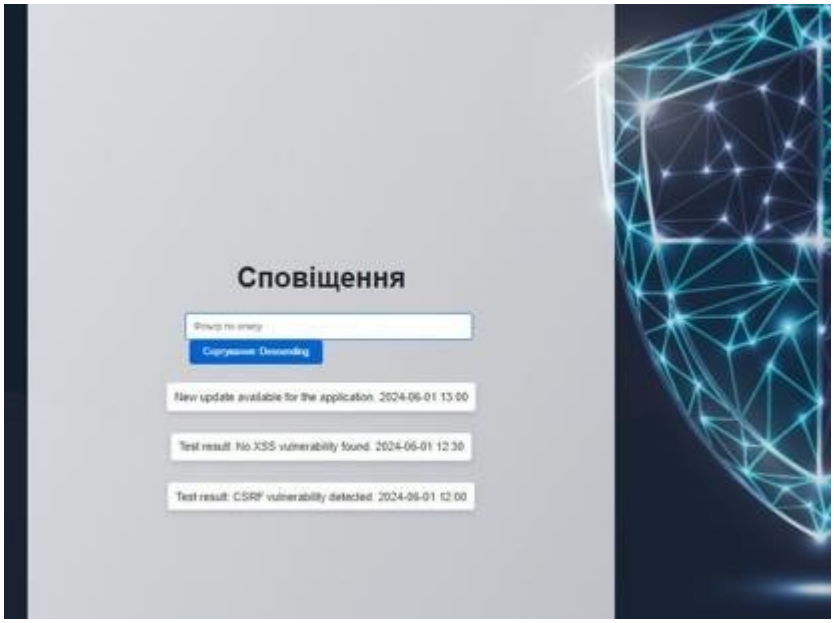


Рисунок 3.8 – Сторінка Сповіщень підсистеми

Система повідомлень коректно відображає усі результати тестів та важливі події, допомагаючи відстежувати стан підсистеми захисту.

На рисунку 3.9 представлено журнал активності, в якому фіксуються дії користувача під час тестування веб-додатку.

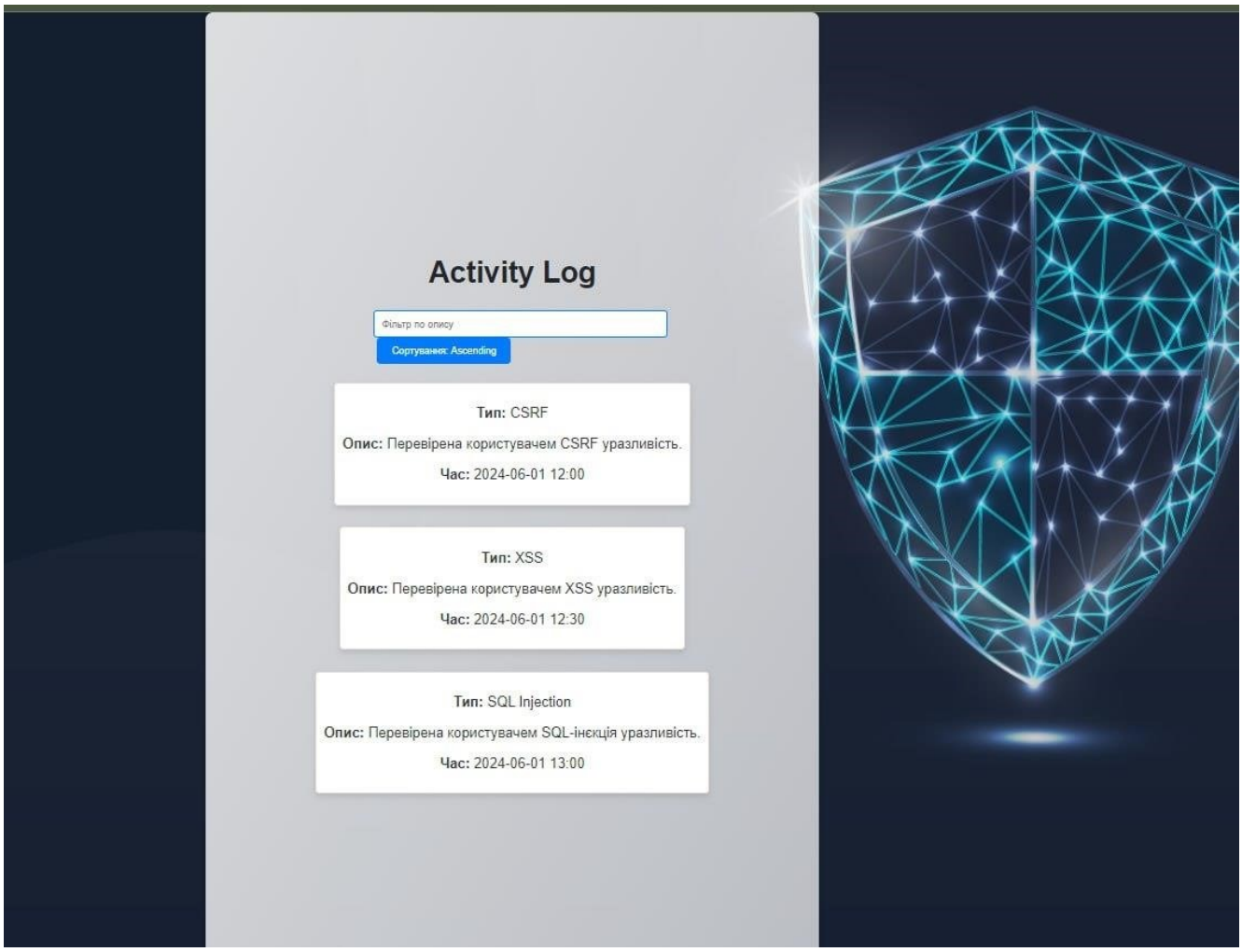


Рисунок 3.9 – Сторінка демонстрації Activity Log

Наведено інформацію про кількість проведених тестів на CSRF, XSS, SQL Injection та PHP Injection вразливості. Ці дані є корисними для відстеження прогресу тестування та визначення пріоритетів у вдосконаленні захисту вебдодатку.

Журнал активності надійно фіксує усі дії користувачів, пов'язані з тестуванням вразливостей, забезпечуючи прозорість та підвищуючи загальну безпеку системи.

На рисунках 3.10 зображено звітність розробленої підсистеми, вона містить статистичні дані та аналітику щодо кількості проведених тестів різних типів виявлених вразливостей.

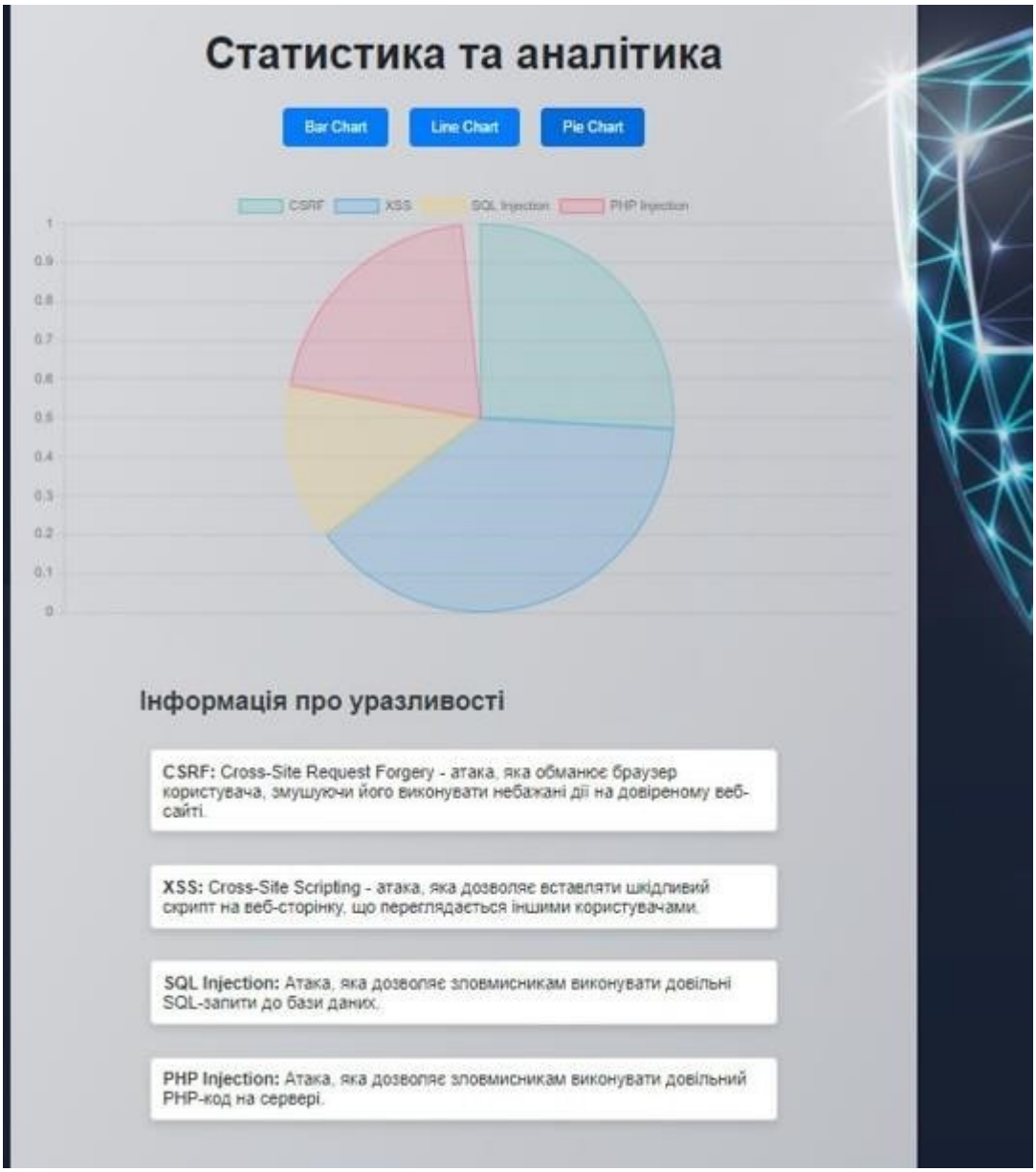


Рисунок 3.10 – Сторінка демонстрації вікна статистики та аналітики

Також, користувачі можуть перевірити ефективність розробленої підсистеми захисту веб-додатків у захисті від CSRF-атак, SQL-ін'єкцій та інших типів атак. Підсистема надає зручний інтерфейс для введення потенційно шкідливих запитів та відображення результатів тестування. Результати тестування допоможуть виявити потенційні недоліки в захисті та вжити відповідних заходів для їх усунення, що в свою чергу підвищить загальну безпеку веб-додатку (рис. 3.11).

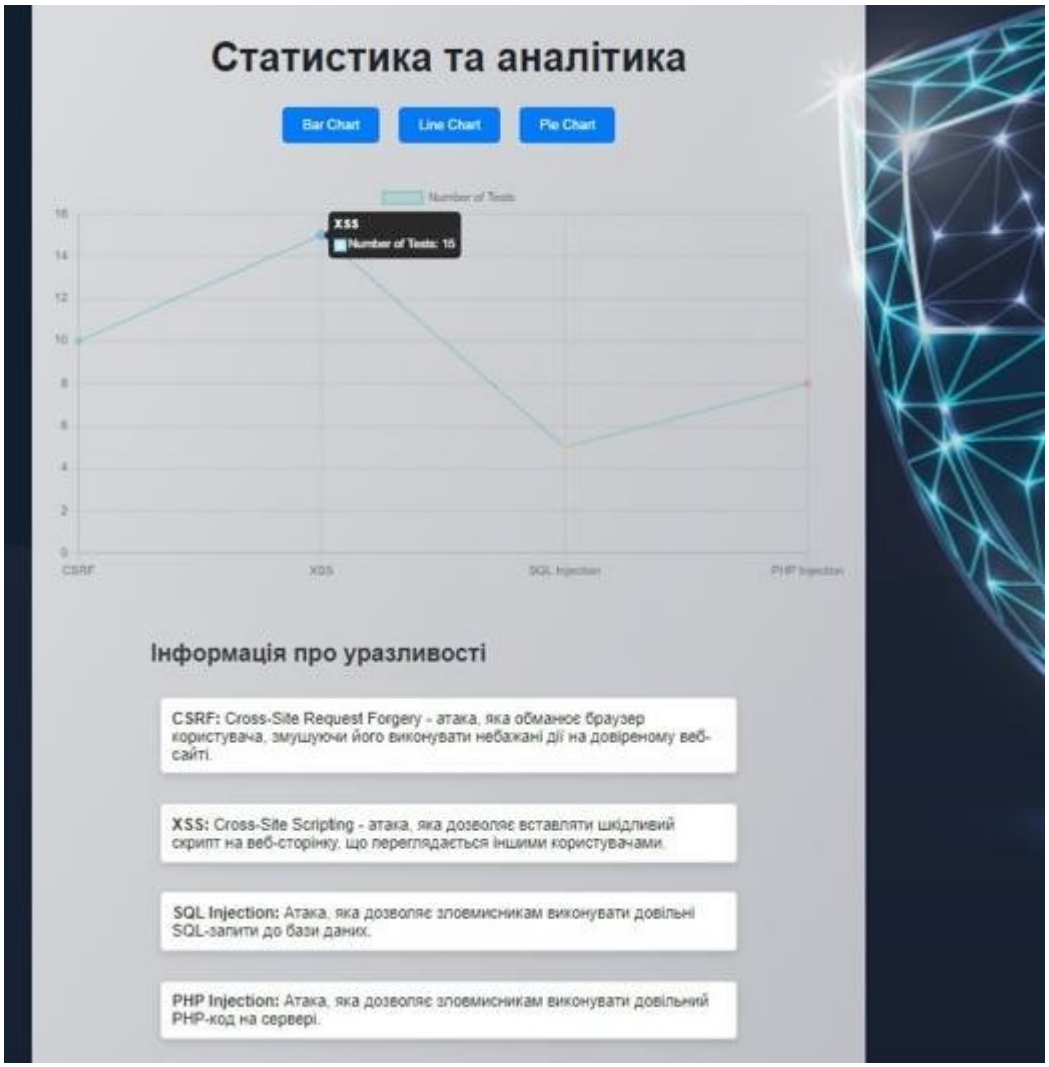


Рисунок 3.11 – Сторінка демонстрації наступного вікна статистики та аналітики

Система статистики та аналітики надає цінні дані про проведені тести, дозволяючи аналізувати частоту та типи атак.

Загалом, розроблена підсистема захисту веб-додатків продемонструвала високу ефективність у тестових умовах, успішно блокуючи всі імітовані атаки. Допоміжні компоненти системи також показали високу надійність та корисність для моніторингу безпеки.

ВИСНОВКИ

У даній бакалаврській дипломній роботі було розглянуто актуальну проблему захисту веб-додатків від різноманітних типів атак, зокрема CSRF, XSS, SQL- та PHP-ін'єкцій. Враховуючи стрімкий розвиток інформаційних технологій та зростаючу популярність веб-додатків, питання забезпечення їх безпеки стає все більш нагальним. Складність та функціональність сучасних веб-додатків збільшують ризики виникнення вразливостей та розширюють вектори можливих атак. Особливу загрозу становлять атаки, спрямовані на порушення конфіденційності, цілісності та доступності даних користувачів.

У ході дослідження було проаналізовано основні типи атак на веб-додатки, їх особливості та потенційні наслідки. Зокрема, розглянуто атаки міжсайтового скриптингу (XSS), які дозволяють зловмисникам впроваджувати шкідливий код на веб-сайти, що може призвести до крадіжки облікових даних користувачів, поширення шкідливого програмного забезпечення тощо. Також досліджено атаки підробки міжсайтових запитів (CSRF), які змушують жертву випадково виконувати небажані дії на вразливому веб-сайті, де вони автентифіковані. SQL-ін'єкції та PHPін'єкції, які експлуатують вразливості в обробці введених даних, можуть призвести до несанкціонованого доступу до баз даних та витоку конфіденційної інформації.

Для ефективного захисту від зазначених атак необхідно застосовувати комплексний підхід, який включає використання належних методів та механізмів захисту, дотримання сучасних стандартів безпеки, а також розробку спеціалізованих підсистем захисту. У роботі було досліджено існуючі методи та механізми захисту від атак CSRF, XSS, SQL- та PHP-ін'єкцій, а також проаналізовано сучасні стандарти та рекомендації з безпеки веб-додатків.

На основі проведеного аналізу було обґрунтовано необхідність розробки підсистеми захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій. Розроблена підсистема має модульну архітектуру, що забезпечує гнучкість та можливість інтеграції з різноманітними веб-додатками. Архітектура підсистеми включає компоненти захисту від кожного типу атак, а також централізований модуль управління та моніторингу безпеки.

Для захисту від CSRF атак були реалізовані механізми генерації та перевірки унікальних токенів, які додаються до кожного запиту, що змінює стан веб-додатку. Це унеможливлює виконання небажаних дій від імені автентифікованого користувача без його відома. Для захисту від XSS атак застосовано методи фільтрації та екранування користувацького введення, а також впроваджено політику безпеки вмісту (CSP), яка обмежує джерела завантаження скриптів та інших ресурсів.

Захист від SQL-ін'єкцій забезпечується шляхом використання параметризованих запитів та підготовлених виразів, які запобігають впровадженню шкідливого коду в SQL-запити. Для захисту від PHP-ін'єкцій застосовано методи валідації та фільтрації користувацького введення, а також реалізовано механізми контролю доступу та обмеження привілеїв виконання PHP-коду.

Розроблена підсистема захисту веб-додатків була протестована на прикладах реальних веб-додатків. Результати тестування показали, що підсистема ефективно виявляє та блокує атаки CSRF, XSS, SQL- та PHP-ін'єкцій, забезпечуючи надійний захист конфіденційних даних користувачів. Підсистема має низький вплив на продуктивність веб-додатків та надає детальні звіти для моніторингу безпеки.

Для зручності використання підсистеми захисту були підготовлені інструкції для користувачів, які описують процес інтеграції підсистеми з веб-додатками, налаштування параметрів безпеки та моніторинг подій безпеки. Інструкції також

містять рекомендації щодо належних практик розробки безпечних веб-додатків та регулярного оновлення компонентів підсистеми захисту.

Отримані результати демонструють, що розроблена підсистема захисту вебдодатків є ефективним інструментом для забезпечення безпеки веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій. Підсистема може бути інтегрована в різноманітні веб-додатки, незалежно від їх масштабу та сфери застосування, що робить її універсальним рішенням для захисту веб-додатків.

Подальші напрямки досліджень можуть включати розширення функціональності підсистеми захисту шляхом додавання нових механізмів захисту від інших типів атак, таких як атаки на відмову в обслуговуванні (DoS), атаки на перелік директорій тощо. Також доцільно дослідити можливості інтеграції підсистеми з існуючими системами моніторингу безпеки та системами виявлення вторгнень (IDS) для забезпечення ще більш комплексного захисту веб-додатків.

Результати даної роботи мають практичне значення для розробників та адміністраторів веб-додатків, які прагнуть підвищити рівень безпеки своїх систем та захистити конфіденційні дані користувачів. Розроблена підсистема захисту може бути використана як готове рішення або як основа для створення спеціалізованих систем захисту, адаптованих до конкретних потреб та вимог веб-додатків.

Підсумовуючи, можна стверджувати, що розробка ефективних підсистем захисту веб-додатків від різноманітних типів атак є важливим завданням в умовах зростаючих загроз інформаційній безпеці. Запропонована в даній роботі підсистема захисту від атак CSRF, XSS, SQL- та PHP-ін'єкцій є вагомим внеском у вирішення цієї проблеми та може служити надійним інструментом для забезпечення безпеки веб-додатків та захисту конфіденційних даних користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Боскін, О. О., et al. "БЕЗПЕКА ВЕБ-ДОДАТКІВ ТА ХАКЕРСЬКІ АТАКИ."

Вестник Херсонского национального технического университета 3 (86) (2023): 83-92.

2. Лаврєнов, В. А., and О. І. Сіренко. "Класифікація загроз веб-застосунків." (2021).

3. Khari, Manju, and Parikshit Sangwan. "Web-application attacks: A survey." 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016.

4. Khari, Manju, and Manoj Kumar. "Comprehensive study of web application attacks and classification." 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016.

5. Ludinard, Romaric, et al. "Detecting attacks against data in web applications." 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS). IEEE, 2012.

6. Венедіктова, Юлія Віталіївна. "Програмний модуль виявлення атак на вебдодатки." (2021).

7. Василенко, І. В. "Універсальній метод захисту веб-додатків." *Системи обробки інформації* 1 (2016): 122-124.

8. Watson, David. "Web application attacks." Network Security 2007.10 (2007): 10-14.

9. Коваль, Дмитро Володимирович. "Програмний модуль захисту інформації від мережевих атак на веб-додатки." (2020).

10. Кільдішев, В. Й., and І. С. Тюпо. "ЗАХИСТ ВЕБ-ДОДАТКІВ: АКТУАЛЬНІ ТЕНДЕНЦІЇ." *Молодий вчений* (2022).

11. Куперштейн, Л. М., et al. *Захисту веб-додатку від xss-атак на основі нейронних мереж*. Diss. ВНТУ, 2020.

12. Oliinyk, Y. O., and P. I. Melnychenko. "Огляд методів захисту Webзастосунків від вразливостей типу CSRF (міжсайтова підробка запитів)." *Problems of Informatization and Management* 3.71: 28-34.

13. Матвеєв, Максим Вячеславович. "Система захисту від Injection і XSS атак веб-застосунків." (2023).

14. Kombade, Rupali D., and B. B. Meshram. "CSRF vulnerabilities and defensive techniques." *International Journal of Computer Network and Information Security* 4.1 (2012): 31.

15. Доліновський, Роман Мирославович. *Методи захисту веб-застосунку від XSS і CSRF вразливостей*. Diss. Тернопіль, ЗУНУ, 2023.

16. Lakhno, Valerii, et al. "WAF ЗАХИСТУ ВНУТРІШНІХ СЕРВІСІВ У СТРУКТУРІ ZERO TRUST." *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»* 1.13 (2021): 81-91.

17. De Ryck, Philippe, et al. "Automatic and precise client-side protection against CSRF attacks." *Computer Security–ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16*. Springer Berlin Heidelberg, 2011.

18. Shrivastava, Ankit, Santosh Choudhary, and Ashish Kumar. "XSS vulnerability assessment and prevention in web application." *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. IEEE, 2016.

19. Собко, Тетяна Анатоліївна. "Аналіз загроз та методів захисту мереж від XSS атак та SQL ін'єкцій." (2021).

20. Попов, Юрій, Сабіна Рузудженк, and Карина Погоріла. "SQL-ін'єкції: огляд потенційних способів захисту." *Комп'ютерні науки та кібербезпека* 3 (2019): 22-26.

21. Праховнік, Н., and Н. Літвінова. "СПОСОБИ ЗАХИСТУ БАЗ ДАНИХ ВІД SQL ІН’ЄКЦІЙ." *Проблеми охорони праці, промислової та цивільної безпеки* (2019): 354-357.

22. Саєнко, Г., and Т. О. Гріненко. "Захист веб-додатків від SQL-ін’єкцій." (2019).

23. Fonseca, Jose, Marco Vieira, and Henrique Madeira. "Evaluation of web security mechanisms using vulnerability & attack injection." IEEE Transactions on dependable and secure computing 11.5 (2013): 440-453.

24. Merlo, Ettore, Dominic Letarte, and Giuliano Antoniol. "Automated protection of php applications against SQL-injection attacks." 11th European Conference on Software Maintenance and Reengineering (CSMR'07). IEEE, 2007.

25. Трифонов, Іван. "Проксуючий фаєрвол для захисту від SQL ін’єкцій." (2019).

26. Валуйський, Станіслав Вікторович, and Тетяна Анатоліївна Собко. "АНАЛІЗ ЗАГРОЗ ТА МЕТОДІВ ЗАХИСТУ МЕРЕЖ ВІД XSS АТАК ТА SQL ІН'ЄКЦІЙ." Збірник матеріалів Міжнародної науково-технічної конференції «ПЕРСПЕКТИВИ ТЕЛЕКОМУНІКАЦІЙ» (2021): 274-276.

27. Kostiuchko, S., A. Sahniuk, and K. Melnyk. "Обхід захисту сайтів за допомогою SQL-ін’єкцій та захист від них." COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION 39 (2020): 136-140.

28. Dahse, Johannes, and Thorsten Holz. "Experience report: An empirical study of PHP security mechanism usage." Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015.

29. Kareem, Fairoz Q., et al. "SQL injection attacks prevention system technology." *Asian Journal of Research in Computer Science* 6.15 (2021): 13-32.

30. Петрів, Петро, and Іван Опірський. "АНАЛІЗ ПРОБЛЕМАТИКИ ВИКОРИСТАННЯ ІСНУЮЧИХ СТАНДАРТІВ З ВЕБ-ВРАЗЛИВОСТЕЙ."

Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка» 2.22 (2023): 96-112.

31. Тригубець, Мирослав Іванович. Порівняльний аналіз сканерів

вразливостей і брандмауера веб-додатків для бізнесу. BS thesis. ТНТУ, 2023.

32. Зеггум, Суфіан. "Дослідження тестувань на проникнення та безпеки веб-додатків." (2023).

33. Твердохліб, Іван, and Ігор Сасовець. "Методики тестування безпеки веб-додатку на основі інформації з відкритих джерел." 4th International Scientific Conference" Problems of Information Economy Formation in Ukraine. 2019.

34. Ясінський, Олександр Юрійович. "Дослідження підходів до безпеки та захисту веб–додатків на основі JS."

35. Javed, Ashar. "Csp aider: An automated recommendation of content security policy for web applications." IEEE Oakland Web 2 (2011): 1-2.

36. Ушенко, М. С. "Аналіз захищеності веб-додатків від атак на прикладному рівні мережевої моделі OSI." (2018).

37. Damanik, Venia Noella Nanisura, and Septia Ulfa Sunaringtyas. "Secure code recommendation based on code review result using owasp code review guide." 2020 International Workshop on Big Data and Information Security (IWBIS). IEEE, 2020.

38. Піддубняк, Поліна Володимирівна. Дослідження систем автоматизованого тестування безпеки веб-додатків. MS thesis. Український державний університет науки і технологій, Дніпро, 2021.

39. Gupta, Shashank, and Brij Bhooshan Gupta. "Detection, avoidance, and attack pattern mechanisms in modern web application vulnerabilities: present and future challenges." International Journal of Cloud Applications and Computing (IJCAC) 7.3 (2017): 1-43.

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
до бакалаврської дипломної роботи на тему:

Розробка підсистеми захисту веб-додатків від атак CSRF I XSS та атак типу SQL- та PHP -ін'єкцій»

08–72.БДР.013.00.081.ТЗ

Керівник бакалаврської дипломної роботи
к.т.н., доц. каф. МБІС
_____ Грицак А. В.
"____" _____

Вінниця – 2024 р.

1. Найменування та область застосування

Підсистема захисту веб-додатків від атак CSRF і XSS та атак типу SQL- та PHPін'єкцій. Область застосування: забезпечення безпеки веб-додатків від поширених типів атак та вразливостей.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробити підсистему захисту веб-додатків від атак CSRF і XSS та атак типу SQL- та PHP-ін'єкцій для підвищення рівня безпеки веб-додатків та захисту конфіденційних даних користувачів.

3.2 Призначення: розроблена підсистема захисту забезпечує ефективний захист веб-додатків від атак CSRF і XSS, а також атак типу SQL- та PHP-ін'єкцій, запобігаючи несанкціонованому доступу, крадіжці даних та іншим загрозам безпеки.

4. Джерела розробки

4.1 Василенко, І. В. "Універсальній метод захисту веб-додатків." Системи обробки інформації 1 (2016): 122-124.

4.2 Кільдішев, В. Й., and І. С. Тюпо. "ЗАХИСТ ВЕБ-ДОДАТКІВ: АКТУАЛЬНІ ТЕНДЕНЦІЇ." Молодий вчений (2022)

4.3 Khari, Manju, and Parikshit Sangwan. "Web-application attacks: A survey." 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016.

4.4 Khari, Manju, and Manoj Kumar. "Comprehensive study of web application attacks and classification." 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016.

4.5 Доліновський, Роман Мирославович. Методи захисту веб-застосунку від XSS і CSRF вразливостей. Diss. Тернопіль, ЗУНУ, 2023.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Core i3–6400U 2.40GHz і подібні до них;
– оперативна пам’ять – не менше 4Gb;
– середовище функціонування – операційна система сімейство Windows;
– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	22.02.2024	25.02.2024
2.	Аналіз предметної області обраної теми	26.02.2024	09.03.2024
3.	Розробка алгоритму роботи	09.03.2024	04.04.2024
4.	Написання бакалаврської роботи на основі розробленої теми	05.04.2024	06.05.2024
5.	Передзахист бакалаврської дипломної роботи	06.05.2024	24.05.2024
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	27.05.2024	03.06.2024
7.	Захист бакалаврської дипломної роботи	12.06.2024	13.06.2024

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

– ПЗ до бакалаврської дипломної роботи;

- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____Марцун Н. М.

Додаток Б. Лістинг коду розробленої підсистеми захисту веб-додатків

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Home from './components/Home'; import CSRF from
'./components/CSRF'; import XSS from './components/XSS'; import SQL
from './components/SQL'; import PHP from './components/PHP'; import
Notifications from './components/Notifications'; import ActivityLog from
'./components/ActivityLog'; import Statistics from
'./components/Statistics'; import Info from './components/Info';
import './styles/main.css';

function App() {
return (
<Router>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/csrf" element={<CSRF />} />
      <Route path="/xss" element={<XSS />} />
      <Route path="/sql" element={<SQL />} />
      <Route path="/php" element={<PHP />} />
      <Route path="/notifications" element={<Notifications />} />
      <Route path="/activity-log" element={<ActivityLog />} />
      <Route path="/statistics" element={<Statistics />} />
      <Route path="/info" element={<Info />} /> {/* */}
    </Routes>
  </Router>
 );
}

export default App; import React
from 'react'; import ReactDOM
from 'react-dom';
import './styles/main.css'; import
App from './App';

ReactDOM.render(
 <React.StrictMode>
   <App />
 </React.StrictMode>,
 document.getElementById('root')
);
{
   "name": "client",
   "version": "0.1.0",
   "private": true,
   "dependencies": {
      "chart.js": "^4.4.3",
      "react": "^18.2.0",
      "react-chartjs-2": "^5.2.0",
      "react-dom": "^18.2.0",
      "react-router-dom": "^6.3.0",
```

```
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
const express = require('express'); const
cors = require('cors'); const bodyParser =
require('body-parser'); const app =
express();

app.use(cors()); app.use(bodyParser.json());



const csrfRoutes = require('./routes/csrfRoutes');
const xssRoutes = require('./routes/xssRoutes'); const
sqlRoutes = require('./routes/sqlRoutes');
const phpRoutes = require('./routes/phpRoutes');

app.use('/api/csrf', csrfRoutes);
app.use('/api/xss', xssRoutes); app.use('/api/sql',
sqlRoutes);
app.use('/api/php', phpRoutes);

module.exports = app; const app =
require('./app'); const mongoose =
require('mongoose');

const PORT = process.env.PORT || 5000; const MONGO_URI =
'mongodb://localhost:27017/web-security-trainer';
```

```
mongoose.connect(MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => {
    console.log('Connected to MongoDB');
app.listen(PORT, () => {          console.log(`Server is running
on port ${PORT}`);
    });
  })
  .catch(err => {
    console.error('Failed to connect to MongoDB', err);
  });
import React, { useState } from 'react'; import
'../styles/main.css';

const ActivityLog = () => {
 const [activities, setActivities] = useState([
  { id: 1, type: 'CSRF', description: 'Перевірка на уразливість CSRF.', timestamp: '2024-06-01 12:00', status: 'Успішно' },
{ id: 2, type: 'XSS', description: 'Перевірка на уразливість XSS.', timestamp: '2024-06-01 12:30', status: 'Неуспішно' },    {
id: 3, type: 'SQL Injection', description: 'Перевірка на уразливість SQL-інєкцій.', timestamp: '2024-06-01 13:00', status:
'Успішно' },
 ]);
 const [filter, setFilter] = useState('');
 const [sortOrder, setSortOrder] = useState('asc');

 const handleFilterChange = (e) => {
  setFilter(e.target.value);
 };

 const handleSortChange = () => {
  setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
 };

 const filteredActivities = activities
  .filter(activity => activity.description.toLowerCase().includes(filter.toLowerCase()))
  .sort((a, b) => {      if
(sortOrder === 'asc') {
    return new Date(a.timestamp) - new Date(b.timestamp);
   } else {
    return new Date(b.timestamp) - new Date(a.timestamp);
   }
  });

 return (
  <div className="container">
   <h1>Журнал активності</h1>
   <div className="controls">
    <input
type="text"
      value={filter}
      onChange={handleFilterChange}
placeholder="Фільтр по опису"          className="input"
    />
    <button onClick={handleSortChange} className="button">
```

```
        Сортування: {sortOrder === 'asc' ? 'За зростанням' : 'За спаданням'}
</button>
    </div>
    <ul className="activity-list">
     {filteredActivities.map((activity) => (
       <li key={activity.id} className={`activity-item ${activity.status === 'Успішно' ? 'success' : 'failure'}`}>
<p><strong>Тип:</strong> {activity.type}</p>
         <p><strong>Опис:</strong> {activity.description}</p>
         <p><strong>Час:</strong> {activity.timestamp}</p>
         <p><strong>Статус:</strong> {activity.status}</p>
       </li>
     ))}
    </ul>
  </div>
 );
};

export default ActivityLog; import
React, { useState } from 'react';
import '../styles/main.css';

const CSRF = () => {   const [input,
setInput] = useState('');   const [result,
setResult] = useState('');
 const [vulnerable, setVulnerable] = useState(null);

 const handleSubmit = (e) => {     e.preventDefault();     const
vulnerabilityDetected = checkForVulnerability(input);
setResult(`Результат для введення: ${input}`);
setVulnerable(vulnerabilityDetected);
 };

 const checkForVulnerability = (code) => {     const
formPattern = /<form.*method=["']post["'].*>/i;     const
tokenPattern = /<input.*name=["']token["'].*>/i;
const hasForm = formPattern.test(code);     const
hasToken = tokenPattern.test(code);     return hasForm
&& !hasToken;
 };

 return (
  <div className="container">
   <h1>Аналіз уразливостей CSRF</h1>
   <p>Перевірте свій код на наявність CSRF уразливостей тут.</p>
   <form onSubmit={handleSubmit} className="form">
    <textarea
value={input}
       onChange={(e) => setInput(e.target.value)}
placeholder="Введіть свій код HTML"          className="input
textarea"
     />
```

```
      <button type="submit" className="button">Перевірити</button>
</form>
    {result && (
      <div className="result">
        <p>{result}</p>
        {vulnerable !== null && (
          <p className={`vulnerability ${vulnerable ? 'vulnerable' : 'safe'}`}>
            {vulnerable ? 'Уразливість виявлено! Потрібно уникати використання небезпечних елементів.' : 'Уразливість
не виявлено. Код безпечний.'}
          </p>
        )}
      </div>
    )}
    <div className="examples">
      <h2>Приклади уразливостей CSRF</h2>
      <p>Нижче наведено приклади уразливого коду:</p>
      <ul>
        <li><code>&lt;form action="/delete" method="post"&gt;</code></li>
        <li><code>&lt;input type="hidden" name="token" value="12345"&gt;</code></li>
      </ul>
      <h2>Як уникнути уразливостей</h2>
      <p>Використовуйте наступні практики для підвищення безпеки вашого коду:</p>
      <ul>
        <li>Використовуйте CSRF токени у всіх формах.</li>
        <li>Перевіряйте походження запитів.</li>
      </ul>
    </div>
  </div>
 );
};

export default CSRF;
import React, { useState } from 'react'; import
'../styles/main.css';

const PHP = () => {   const [input, setInput] =
useState('');   const [result, setResult] =
useState('');   const [vulnerable, setVulnerable] =
useState(null);   const [details, setDetails] =
useState('');

 const handleSubmit = (e) => {     e.preventDefault();
const { detected, reason } = checkForVulnerability(input);
setResult(`Результат для введення: ${input}`);
setVulnerable(detected);
   setDetails(reason);
 };

 const checkForVulnerability = (code) => {
   const unsafePatterns = ["eval", "system", "exec", "shell_exec"];     const foundPattern
= unsafePatterns.find(pattern => code.includes(pattern));     if (foundPattern) {
return { detected: true, reason: `Небезпечна функція знайдена: ${foundPattern}` };
   }
```

```
   return { detected: false, reason: 'Небезпечні функції відсутні' };
  };

  return (
   <div className="container">
    <h1>Аналіз уразливостей PHP</h1>
    <p>Перевірте свій код на наявність PHP Injection уразливостей тут.</p>
    <form onSubmit={handleSubmit} className="form">
     <textarea
value={input}
      onChange={(e) => setInput(e.target.value)}
placeholder="Введіть свій PHP код"
      className="input textarea"
     />
     <button type="submit" className="button">Перевірити</button>
    </form>
    {result && (
     <div className="result">
      <p>{result}</p>
      {vulnerable !== null && (
       <p className={`vulnerability ${vulnerable ? 'vulnerable' : 'safe'}`}>
        {vulnerable ? 'Уразливість виявлено! Потрібно уникати використання небезпечних функцій.' : 'Уразливість не
виявлено. Код безпечний.'}
       </p>
      )}
      <p>{details}</p>
     </div>
    )}
    <div className="examples">
     <h2>Приклади уразливостей PHP Injection</h2>
     <p>Нижче наведено приклади уразливого коду:</p>
     <ul>
      <li><code>eval($_GET['code']);</code></li>
      <li><code>system($_POST['cmd']);</code></li>
<li><code>exec($input);</code></li>
      <li><code>shell_exec($command);</code></li>
     </ul>
     <h2>Як уникнути уразливостей</h2>
     <p>Використовуйте наступні практики для підвищення безпеки вашого коду:</p>
     <ul>
      <li>Уникайте використання функцій, що виконують код з вхідних даних.</li>
      <li>Використовуйте параметризовані запити для роботи з базою даних.</li>
      <li>Перевіряйте і очищуйте всі вхідні дані.</li>
     </ul>
    </div>
   </div>
  );
};

export default PHP; const XSS = () => {   const
[input, setInput] = useState('');   const [result,
setResult] = useState('');   const [vulnerable,
```

```
setVulnerable] = useState(null);   const [details,
setDetails] = useState('');


 const handleSubmit = (e) => {     e.preventDefault();
const { detected, reason } = checkForVulnerability(input);
setResult(`Результат для введення: ${input}`);
setVulnerable(detected);
   setDetails(reason);
 };


 const checkForVulnerability = (code) => {
   const unsafePatterns = ["<script>", "alert(", "onload=", "onerror="];     const
foundPattern = unsafePatterns.find(pattern => code.includes(pattern));     if
(foundPattern) {       return { detected: true, reason: `Небезпечний елемент знайдено:
${foundPattern}` };
   }
   return { detected: false, reason: 'Небезпечні елементи відсутні' };
 };


 return (
   <div className="container">
     <h1>Аналіз уразливостей XSS</h1>
     <p>Перевірте свої дані на наявність XSS уразливостей тут.</p>
     <form onSubmit={handleSubmit} className="form">
       <textarea
value={input}
         onChange={(e) => setInput(e.target.value)}
placeholder="Введіть свій код HTML/JS"           className="input
textarea"
       />
       <button type="submit" className="button">Перевірити</button>
</form>
     {result && (
       <div className="result">
         <p>{result}</p>
         {vulnerable !== null && (
           <p className={`vulnerability ${vulnerable ? 'vulnerable' : 'safe'}`}>
             {vulnerable ? 'Уразливість виявлено! Потрібно уникати використання небезпечних елементів.' : 'Уразливість
не виявлено. Код безпечний.'}
           </p>
         )}
         <p>{details}</p>
       </div>
     )}
     <div className="examples">
       <h2>Приклади уразливостей XSS</h2>
       <p>Нижче наведено приклади уразливого коду:</p>
       <ul>
         <li><code>&lt;script&gt;alert('XSS')&lt;/script&gt;</code></li>
         <li><code>&lt;img src="invalid" onerror="alert('XSS')"&gt;</code></li>
       </ul>
       <h2>Як уникнути уразливостей</h2>
       <p>Використовуйте наступні практики для підвищення безпеки вашого коду:</p>
```

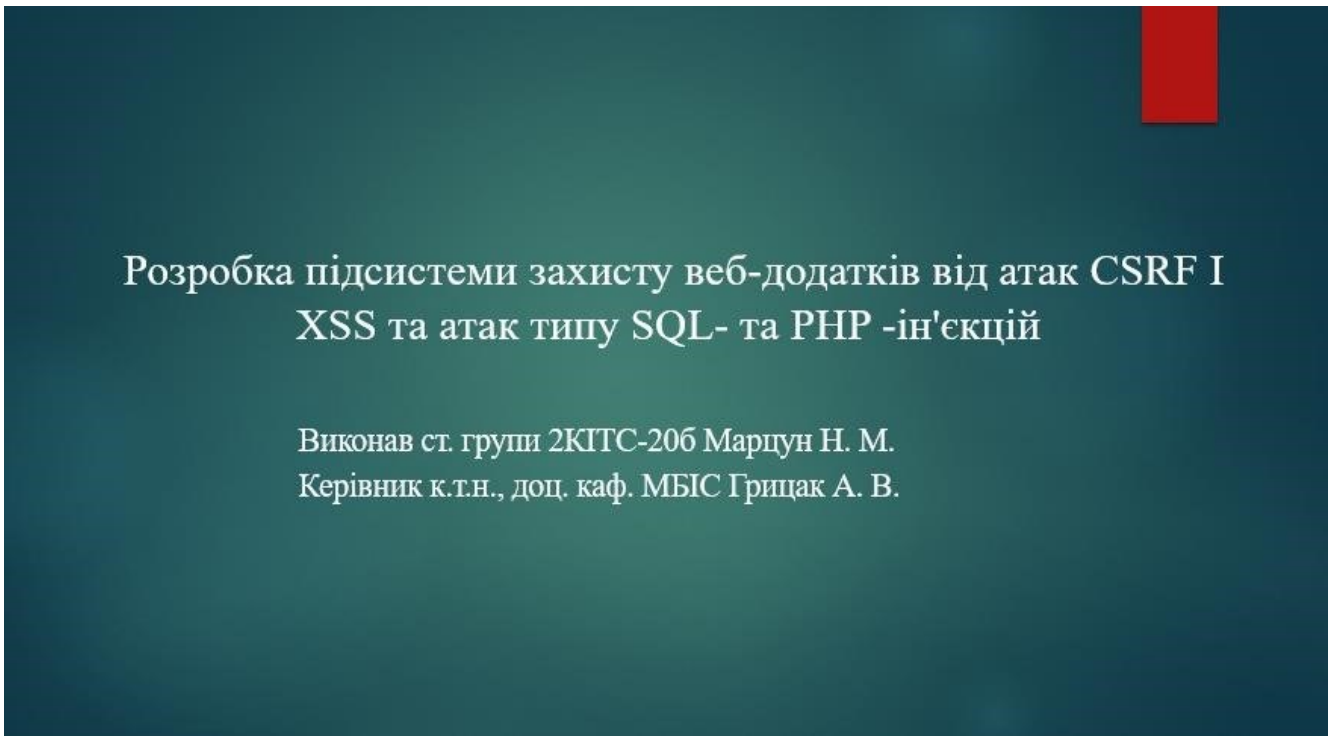
```
      <ul>
        <li>Очищайте і перевіряйте всі вхідні дані.</li>
        <li>Використовуйте контекстно-безпечну обробку даних.</li>
      </ul>
    </div>
  </div>
 );
};

export default XSS;
import React, { useState } from 'react';
import { Bar, Line, Pie } from 'react-chartjs-2';
import 'chart.js/auto'; import
'../styles/main.css';

const Statistics = () => {   const [chartType,
setChartType] = useState('Bar');


 const stats = {
csrfTests: 10,
xssTests: 15,    sqlTests:
5,    phpTests: 8,
successfulTests: 20,
  failedTests: 18,
 };


 const data = {
  labels: ['CSRF', 'XSS', 'SQL Injection', 'PHP Injection'],
datasets: [
    {
      label: 'Number of Analyses',
      data: [stats.csrfTests, stats.xssTests, stats.sqlTests, stats.phpTests],
backgroundColor: [          'rgba(75, 192, 192, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)',
        'rgba(255, 99, 132, 0.2)',
      ],
      borderColor: [
'rgba(75, 192, 192, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(255, 99, 132, 1)',
      ],
      borderWidth: 1,
    },
  ],
 };


 const successData = {
  labels: ['Successful Analyses', 'Failed Analyses'],
  datasets: [
    {
```

```
      data: [stats.successfulTests, stats.failedTests],
backgroundColor: [          'rgba(75, 192, 192, 0.2)',
        'rgba(255, 99, 132, 0.2)',
      ],
      borderColor: [
'rgba(75, 192, 192, 1)',
        'rgba(255, 99, 132, 1)',
      ],
      borderWidth: 1,
    },
  ],
 };

 const options = {
responsive: true,
  maintainAspectRatio: false,
scales: {
    y: {
      beginAtZero: true,
    },
  },
 };

 const renderChart = () => {
  switch (chartType) {      case 'Line':        return
<Line data={data} options={options} />;      case
'Pie':        return <Pie data={data}
options={options} />;      default:
      return <Bar data={data} options={options} />;
  }
 };

 return (
  <div className="container">
    <h1>Статистика та аналітика</h1>
    <div className="chart-controls">
      <button className="button" onClick={() => setChartType('Bar')}>Bar Chart</button>
      <button className="button" onClick={() => setChartType('Line')}>Line Chart</button>
<button className="button" onClick={() => setChartType('Pie')}>Pie Chart</button>
    </div>
    <div className="chart-container">
      {renderChart()}
    </div>
    <div className="chart-container">
      <h2>Успішні та неуспішні аналізи</h2>
      <Pie data={successData} options={options} />
    </div>
    <div className="export-buttons">
      <button className="button">Export to CSV</button>
      <button className="button">Export to PDF</button>
    </div>
    <div className="latest-analyses">
      <h2>Останні аналізи</h2>
```

```
      <ul>
        <li>CSRF - успішно</li>
        <li>XSS - неуспішно</li>
        <li>SQL Injection - успішно</li>
        <li>PHP Injection - успішно</li>
      </ul>
    </div>
  </div>
 );
};
const Notifications = () => {
 const [notifications, setNotifications] = useState([
  { id: 1, type: 'CSRF', message: 'Test result: CSRF vulnerability detected.', timestamp: '2024-06-01 12:00' },
  { id: 2, type: 'XSS', message: 'Test result: No XSS vulnerability found.', timestamp: '2024-06-01 12:30' },
  { id: 3, type: 'Update', message: 'New update available for the application.', timestamp: '2024-06-01 13:00' },
 ]);
 const [filter, setFilter] = useState('');
 const [sortOrder, setSortOrder] = useState('asc');

 const handleFilterChange = (e) => {
  setFilter(e.target.value);
 };

 const handleSortChange = () => {
  setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc');
 };

 const filteredNotifications = notifications
  .filter(notification => notification.message.toLowerCase().includes(filter.toLowerCase()))
  .sort((a, b) => {      if
(sortOrder === 'asc') {
      return new Date(a.timestamp) - new Date(b.timestamp);
    } else {
      return new Date(b.timestamp) - new Date(a.timestamp);
    }
  });

 return (
  <div className="container">
    <h1>Сповіщення</h1>
    <div className="controls">
      <input
type="text"
        value={filter}
        onChange={handleFilterChange}
placeholder="Фільтр по опису"           className="input"
      />
      <button onClick={handleSortChange} className="button">
        Сортування: {sortOrder === 'asc' ? 'Ascending' : 'Descending'}
</button>
    </div>
    <ul className="notification-list">
```

```
      {filteredNotifications.map((notification) => (
        <li key={notification.id} className="notification-item">
          <i className="fas    fa-info-circle"></i>            {notification.message}       <span
className="timestamp">{notification.timestamp}</span>
        </li>
      ))}
    </ul>
  </div>
 );
};
```

Додаток В. Ілюстративний матеріал



Актуальність:

- Зростання кількості та складності веб-атак, таких як CSRF, XSS, SQL- та PHP-ін'єкції
- Необхідність розробки ефективних засобів захисту веб-додатків від цих типів атак

Мета:

- Розробити підсистему захисту веб-додатків від атак CSRF, XSS та атак типу SQL- та PHP-ін'єкцій

Завдання:

1. Проаналізувати особливості атак CSRF, XSS, SQL- та PHP-ін'єкцій та існуючі методи захисту
2. Розробити архітектуру та функціональні вимоги до підсистеми захисту веб-додатків
3. Реалізувати підсистему захисту з використанням сучасних технологій та підходів
4. Провести тестування та оцінку ефективності розробленої підсистеми захисту

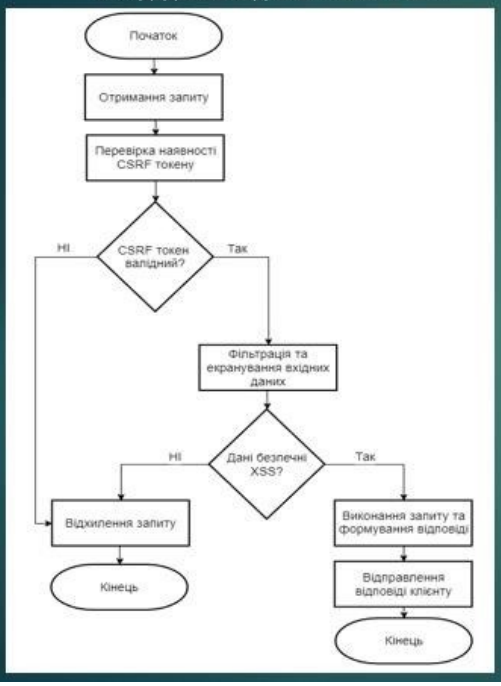
Атаки на веб-додатки:

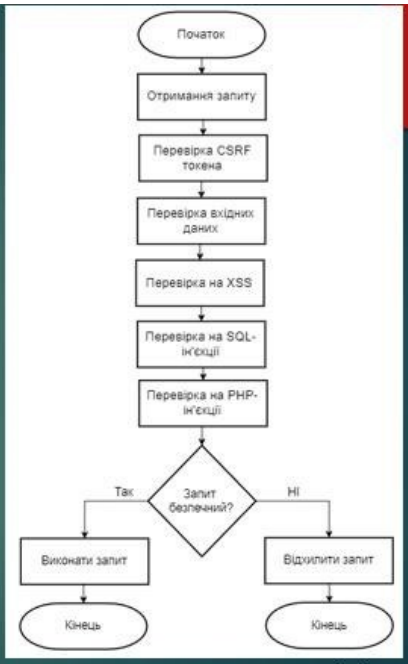
- **CSRF** (Cross-Site Request Forgery): примушує користувача виконувати небажані дії на веб-сайті, на якому він автентифікований
- **XSS** (Cross-Site Scripting): дозволяє зловмисникам виконувати шкідливий код на стороні клієнта через вразливості веб-додатку
- **SQL-ін'єкції:** впровадження шкідливого SQL-коду через вразливості в обробці користувацького вводу для отримання несанкціонованого доступу до бази даних
- **PHP-ін'єкції**: впровадження шкідливого PHP-коду через вразливості в обробці користувацького вводу для виконання небажаних дій на сервері

Методи захисту:

- Валідація та фільтрація користувацького вводу
- Використання параметризованих запитів та підготовлених виразів для запобігання SQL-ін'єкціям
- Екранування спеціальних символів та обмеження виконання скриптів для захисту від XSS
- Перевірка автентичності запитів та використання токенів для захисту від CSRF

Блок-схема алгоритму, що лежить в основі підсистеми із захисту веб-додатків від атак CSRF і XSS





Загальна схема алгоритму, що лежить в основі підсистеми захисту веб-додатків від атак CSRF, XSS, SQL- та PHP-ін'єкцій

Блок-схема алгоритму роботи підсистеми із захисту веб-додатків від атак типу SQL- та PHP-ін'єкцій

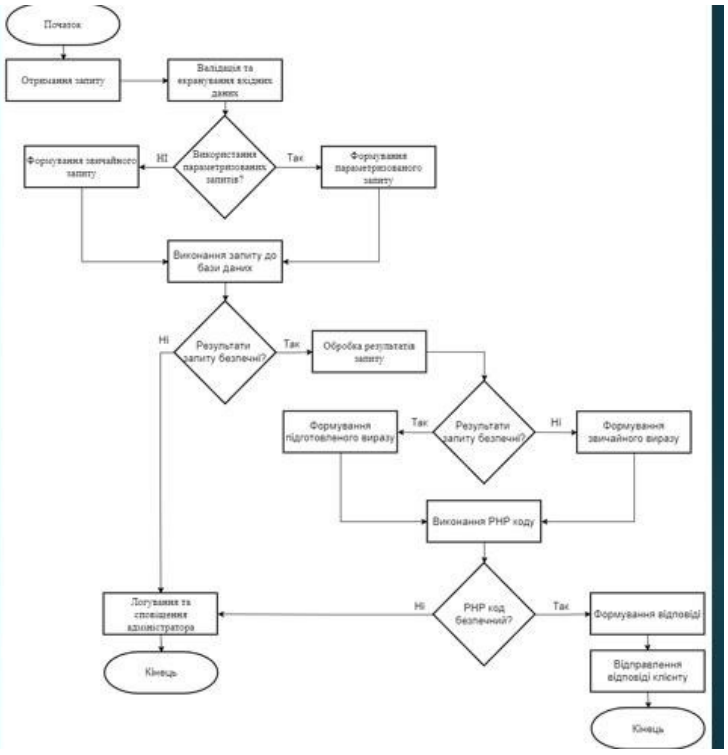
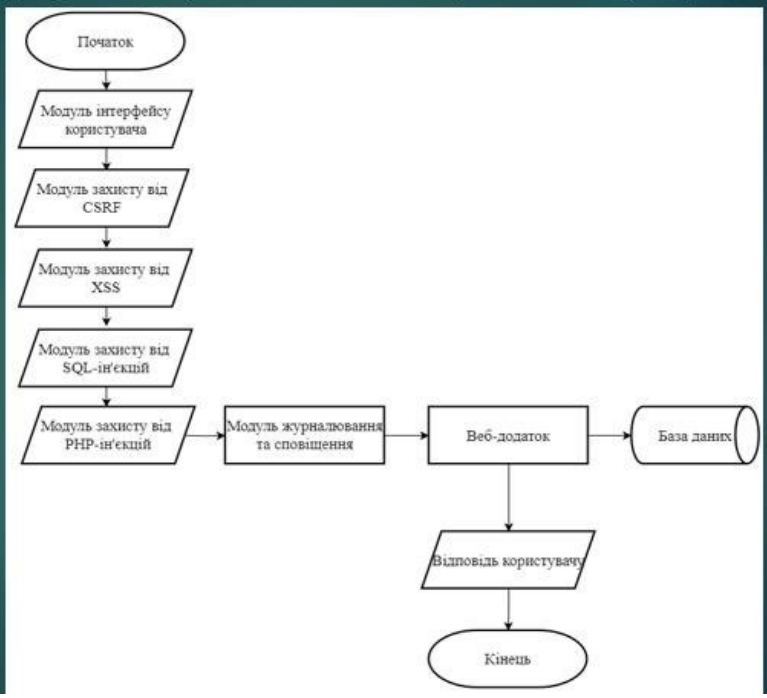
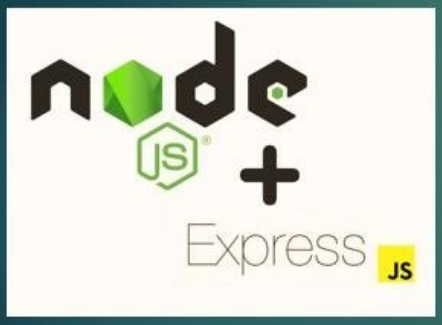


Схема взаємодії користувача із модулями підсистеми захисту від атак CSRF, XSS, SQL- та PHP-ін'єкцій

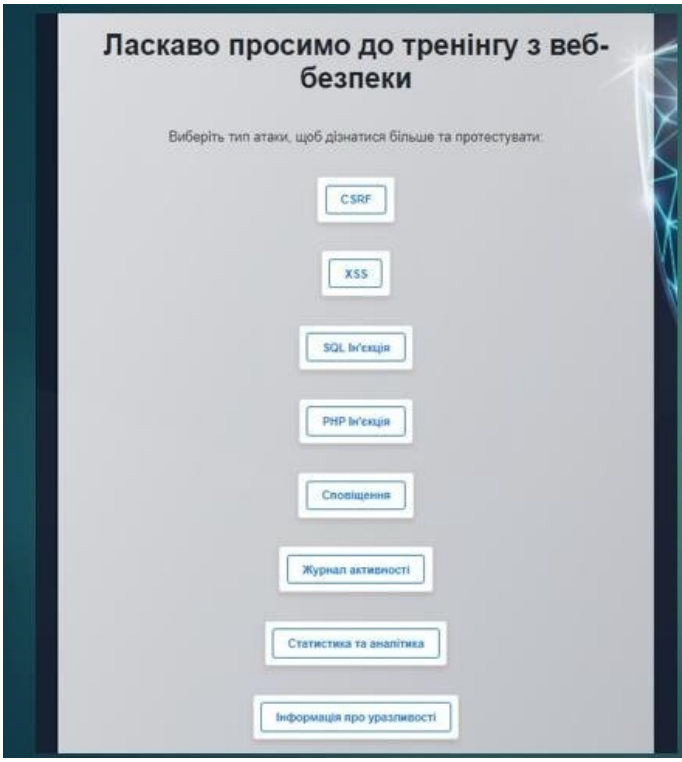


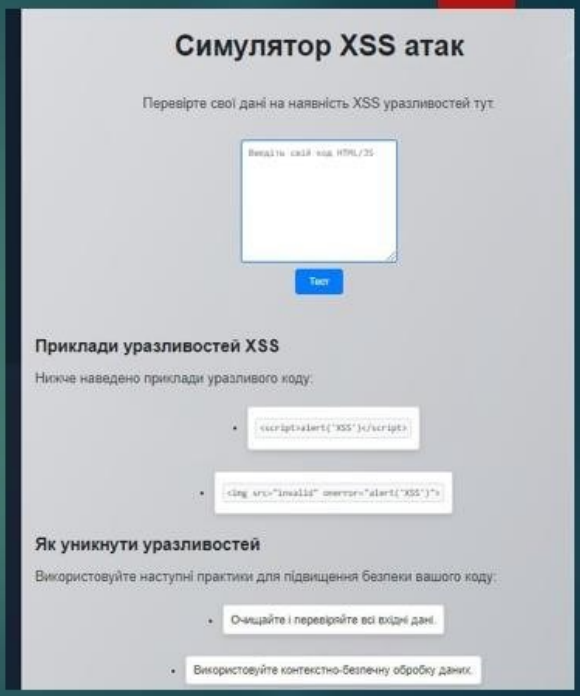
Для розробки підсистеми захисту веб-додатків було обрано наступні технології та інструменти:

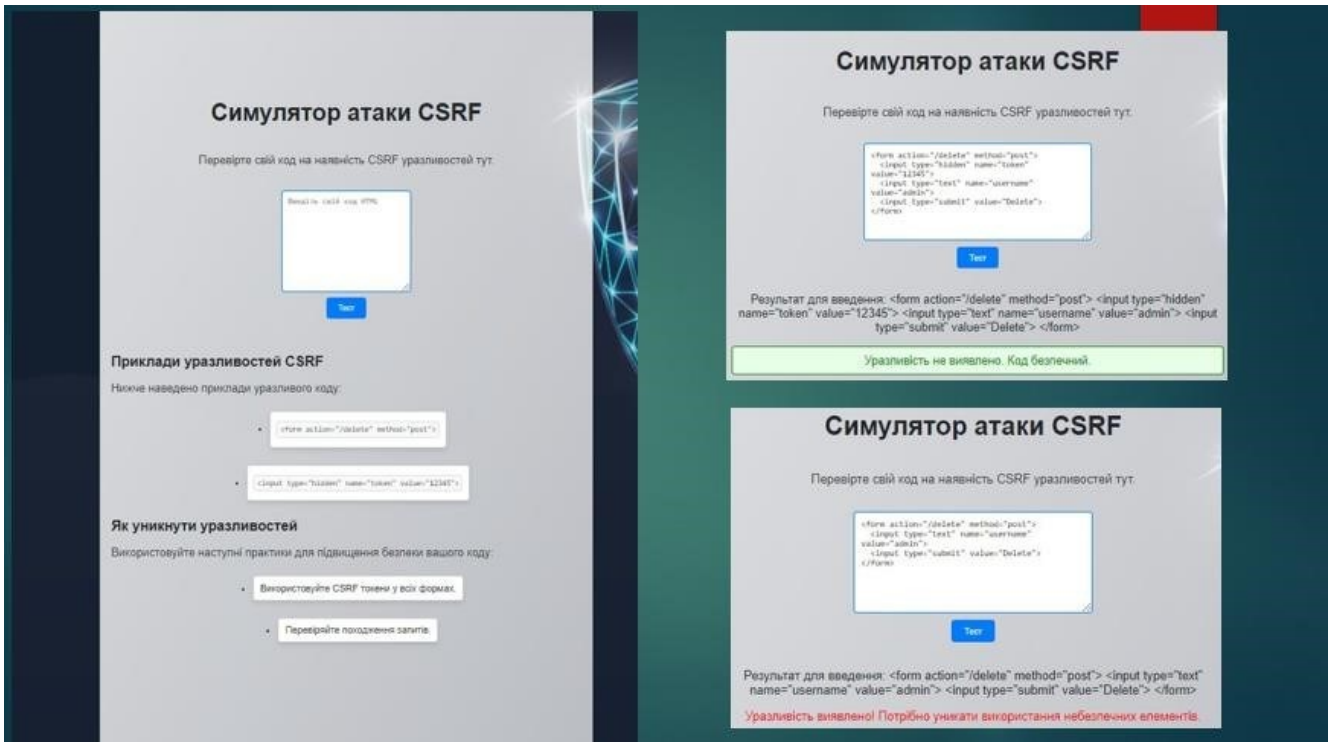
- Node.js та Express для серверної частини
- MongoDB та Mongoose для бази даних
- Content Security Policy для запобігання XSS-атакам

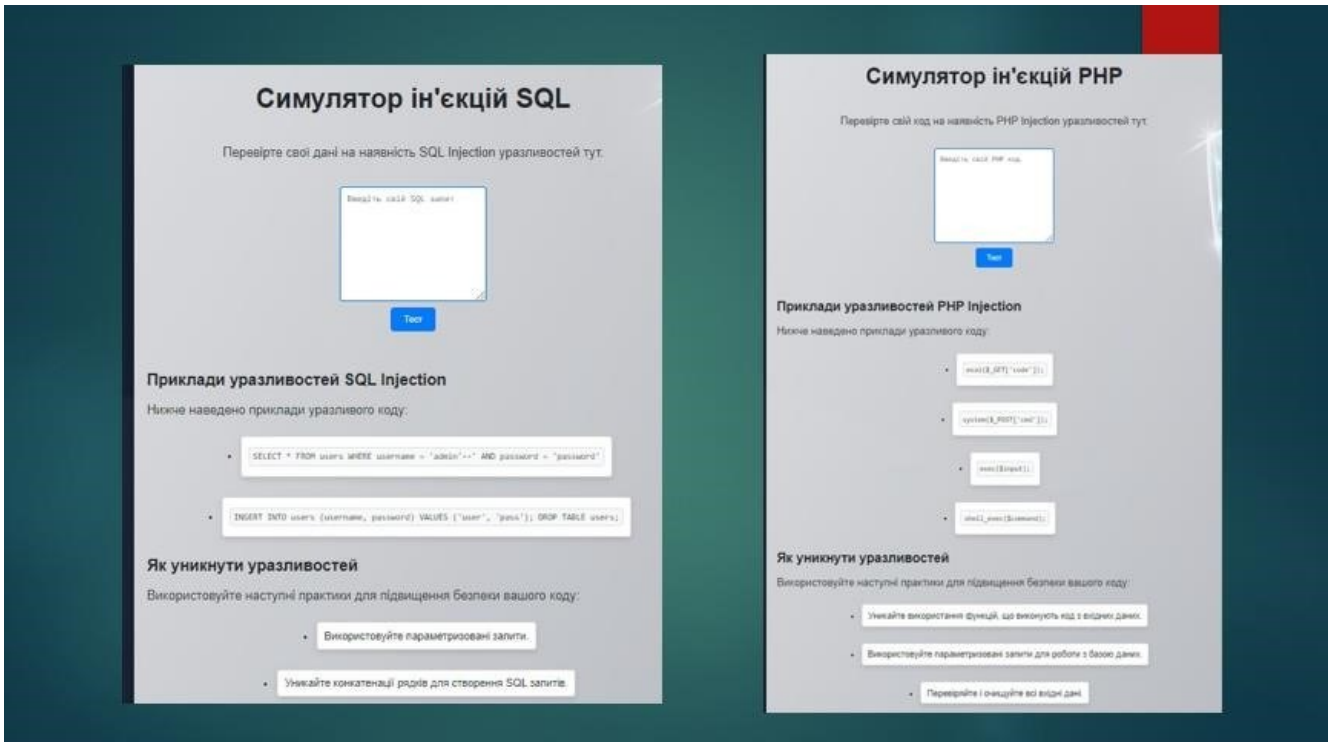


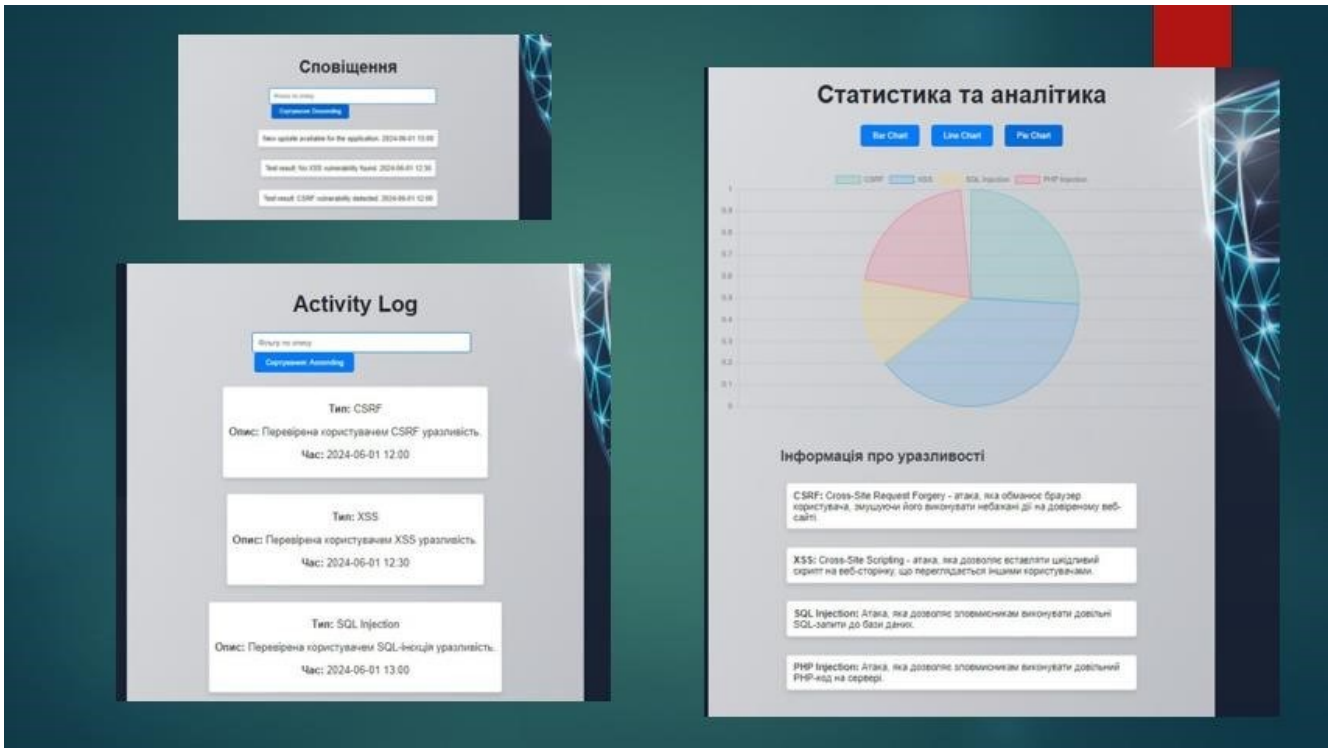
Ці інструменти та технології забезпечують високу продуктивність, гнучкість, надійний захист від поширених загроз та відповідність найкращим практикам безпеки у веб-розробці.

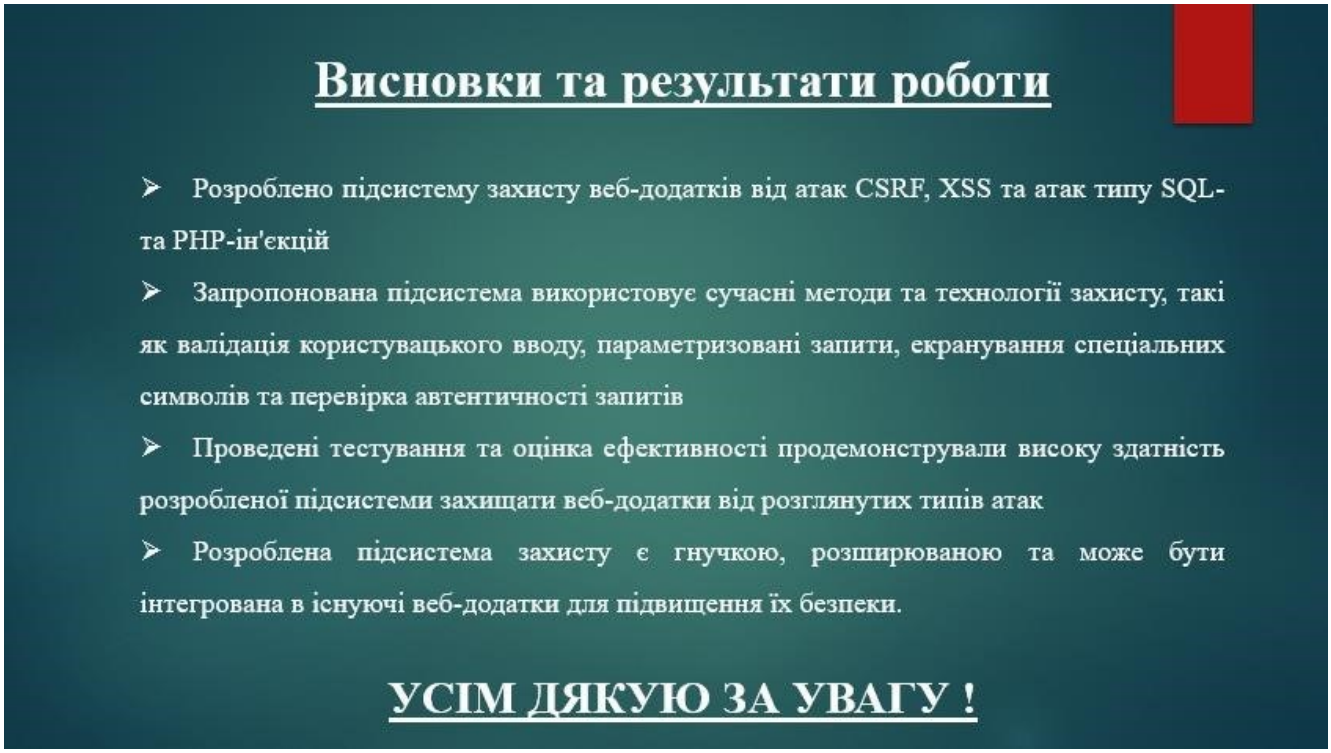





Симулятор атаки CSRF
Перевірте свій код на наявність CSRF уразливостей тут.
Тест
Приклади уразливостей CSRF
Нижче наведено приклади уразливого коду:
Як уникнути уразливостей
Використовуйте наступні практики для підвищення безпеки вашого коду:
Використовуйте CSRF токени у всіх формах.
Перевіряйте походження запитів.
Симулятор атаки CSRF
Перевірте свій код на наявність CSRF уразливостей тут.
Тест
Результат для введення: <form action="/delete" method="post"> <input type="hidden" name="token" value="12345"> <input type="text" name="username" value="admin"> <input type="submit" value="Delete"> </form>
Уразливість не виявлено. Код безпечний.
Симулятор атаки CSRF
Перевірте свій код на наявність CSRF уразливостей тут.
Тест
Результат для введення: <form action="/delete" method="post"> <input type="text" name="username" value="admin"> <input type="submit" value="Delete"> </form>
Уразливість виявлено! Потрібно уникати використання небезпечних елементів.


Симулятор ін'єкцій SQL
Перевірте свої дані на наявність SQL Injection уразливостей тут.
Тест
Приклади уразливостей SQL Injection
Нижче наведено приклади уразливого коду:
Як уникнути уразливостей
Використовуйте наступні практики для підвищення безпеки вашого коду:
Використовуйте параметризовані запити.
Уникайте конкатенації рядків для створення SQL запитів.
Симулятор ін'єкцій PHP
Перевірте свій код на наявність PHP Injection уразливостей тут.
Тест
Приклади уразливостей PHP Injection
Нижче наведено приклади уразливого коду:
Як уникнути уразливостей
Використовуйте наступні практики для підвищення безпеки вашого коду:
Уникайте використання функцій, що виконують код з вхідних даних.
Використовуйте параметризовані запити для роботи з базою даних.
Перевіряйте і очищуйте всі вхідні дані.





Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Додаток Г. Звіт перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка підсистеми захисту веб-додатків від атак CRSF і XSS та атак типу SQL- та PHP- ін'єкцій

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97% Схожість 3%

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Марцун Н.М.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Грицак А.В.
(підпис) (прізвище, ініціали)