Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи магістра

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: «Методи захисту від міжсайтового скриптингу»

Виконавець: студент ІІ курсу, групи КБм-21

Максим СКРИПНИК
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Сергій БУЧИК	
Нормоконтроль	Юрій ЩЕБЛАНІН	

Київ 2023

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
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-21 (група) Скрипника Максима Вікторовича (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Методи захисту від міжсайтового скриптингу

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 3 від 20.10.2022

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес захисту хмарних веб-систем від XSS.
Предмет досліджень	Методи виявлення та запобігання XSS у хмарних системах.
Мета	Розробка рекомендацій щодо виявлення та запобігання міжсайтовому скриптингу у хмарних веб-системах.
Вихідні дані для проведення роботи	Методи захисту від міжсайтового скриптингу.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна Удосконалення рекомендацій для захисту від міжсайтового скриптингу за рахунок аналізу мтодів виявлення та запобігання від XSS у хмарних веб-системах.

Практична цінність Удосконалення та поглиблення рекомендацій щодо захисту від міжсайтового скриптингу у сучасних веб-системах.

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)
Розробка плану для досягнення мети роботи	24.10.2022 – 23.01.2023
Аналіз літературних джерел	24.01.2023 – 14.02.2023
Розробка рекомендацій щодо захисту від міжсайтового скриптингу	15.02.2023 – 24.04.2023
Оформлення і друк пояснювальної записки	25.04.2023 – 19.05.2023

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект Зниження збитків від міжсайтового скриптингу у користувачів та розробників хмарних систем.

Соціальний ефект Покращення захисту інформації у хмарних веб-системах в організаціях різних форм власності.

7. ДОДАТКОВІ ВИМОГИ

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

Завдання прийняв до виконання _____ Максим СКРИПНИК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 24.10.2022 р.
Термін подання кваліфікаційної роботи до ЕК 19.05.2023 р.

УДК. 004.432.16

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Методи захисту від міжсайтового скриптингу»: 72 сторінки основного тексту, 55 рисунків, 2 додатки та 4 таблиці, 20 літературних джерел.

Об’єкт дослідження – процес захисту хмарних веб-систем від XSS.

Мета роботи – розробка рекомендації щодо виявлення та запобігання міжсайтового скриптингу у хмарних системах.

Методи дослідження – аналіз захисту від міжсайтового скриптингу на основі тестового проекту, методи захисту від XSS у хмарних системах.

У роботі досліджено сучасні загрози та методи протидії міжсайтовому скриптингу. Проаналізована сучасна література та ресурси з захисту від XSS загроз. Розроблені налаштування і код додатку написаним на мові програмування PHP у хмарному середовищі Amazon Web Services, на базі якого розроблені рекомендації щодо захисту від XSS у хмарних системах.

Наукова новизна: удосконалено рекомендації для захисту від міжсайтового скритингу за рахунок аналізу методів виявлення та запобігання від XSS у хмарних веб-системах.

Актуальність теми: Захист від міжсайтового скриптингу є надзвичайно важливим аспектом кібербезпеки в сучасних веб-додатках. Цей вид атаки дозволяє зловмисникам впроваджувати шкідливий код на веб-сторінки, що може призвести до викрадення конфіденційної інформації, перенаправлення на інші сайти або навіть до керування веб-додатком від імені його користувачів. Хмарні веб-системи є особливо вразливими до XSS-атак, оскільки вони зазвичай містять велику кількість складних взаємодій між користувачем та сервером.

Ключові слова: міжсайтовий скриптинг, хмарні системи, XSS, захист від міжсайтового скриптингу у хмарних веб-системах.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

XSS	–	Cross Site Scripting
OWASP	–	Open Web Application Security Project
RCE	–	Remote Code Execution
CSRF	–	Cross Site Request Forgery
HTML	–	HyperText Markup Language
DOM	–	Document Object Model
PCI-DSS	–	Payment Card Industry Data Security Standard
HIPAA	–	Health Insurance Portability and Accountability Act
CSP	–	Content Security Policy
SAST	–	Static Application Security Testing
DAST	–	Dynamic Application Security Testing
IAST	–	Interactive Application Security Testing
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
PHP	–	Hypertext Preprocessor
URI	–	Uniform Resource Identifier
AWS	–	Amazon Web Services
WAF	–	Web Application Firewall
SIEM	–	Security information and event management

ЗМІСТ

ВСТУП

Актуальність даної роботи полягає у тому, що міжсайтовий скриптинг є однією з найбільш поширених атак на веб-додатки, і актуальність захисту від цих загроз не зменшується з часом. XSS-атаки використовуються для вставки шкідливого коду на веб-сторінки, який може бути виконаний на комп'ютерах користувачів, що відвідують ці сторінки. Це може призвести до крадіжки даних користувачів, зламу аккаунтів, розповсюдження вірусів і інших шкідливих програм.

Хмарні сервіси зазвичай складні та розподілені, з багатьма компонентами та системами, що взаємодіють між собою. Це робить їх більш вразливими до XSS-атак, оскільки атакувач може спробувати використовувати слабкі місця в одній системі, щоб отримати доступ до іншої.

Популярність хмарних веб-систем зростає з кожним днем, і це призводить до збільшення кількості хмарних додатків, які стають основою роботи для багатьох підприємств та організацій. Проте, зростання кількості хмарних додатків також збільшує вразливість цих систем до атак, включаючи міжсайтовий скриптинг.

Міжсайтовий скриптинг може стати настільки серйозною загрозою, що може призвести до отримання неправомірного доступу до важливих даних, таких як паролі, номери кредитних карток та інша конфіденційна інформація. Це може призвести до великих фінансових втрат та вплинути на репутацію компанії.

Таким чином, захист від XSS-атак у хмарних системах є важливою складовою безпеки, яка повинна бути включена в план безпеки всіх хмарних сервісів. Враховуючи складність хмарних систем та розподілену структуру, важливо підходити до захисту від XSS-атак з високим рівнем уваги та серйозності. Тому тему для магістерської кваліфікаційної роботи було обрано «Методи захисту від міжсайтового скриптингу».

Основною задачею цієї роботи є проаналізувати та дослідити методи виявлення та запобігання міжсайтовому скриптингу у сучасних веб-системах, а також дослідити захист від таких загроз у хмарних веб-системах.

Науковою новизною є удосконалити рекомендації для захисту від міжсайтового скритингу за рахунок аналізу методів виявлення та запобігання від XSS у хмарних веб-системах.

Об'єктом дослідження є процес захисту хмарних веб-систем від XSS.

Метою магістерської кваліфікаційної роботи є розробка рекомендації щодо виявлення та запобігання міжсайтового скриптингу у хмарних системах.

Предметом дослідження є методи виявлення та запобігання міжсайтовому скриптингу у хмарних системах.

Завдання магістерської роботи:

- проаналізувати XSS загрози та класифікацію до них;
- дослідити особливості захисту від міжсайтового скриптингу у хмарних системах;
- розглянути основні методи виявлення XSS загроз;
- дослідити методи запобігання міжсайтовому скриптингу у хмарних системах;
- розробити рекомендації щодо захисту від XSS у хмарних системах.

Апробація результатів роботи:

Скрипник М. Класифікація та методи захисту від міжсайтового скриптингу у хмарних веб-системах / М. Скрипник, С. Бучик // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: Збірник матеріалів доповідей та тез; м. Київ, 27-28 жовтня 2022 року; Київський національний університет імені Тараса Шевченка / Редкол.: В.В. Ільченко, д.ф-м.н., проф., (голова); та ін. – К.: ВПЦ "Київський університет", 2022. – 159 с. (с.50-51)

Скрипник М. Політика безпеки вмісту у хмарних системах / М. Скрипник, С. Бучик // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: Збірник матеріалів доповідей та тез; м. Київ, 27 квітня 2023 року; Київський національний університет імені Тараса Шевченка / Редкол.: В.В. Ільченко, д.ф-м.н., проф., (голова); та ін. – К.: ВПЦ "Київський університет", 2023. – 166 с. (с.7-8)

РОЗДІЛ 1
АНАЛІЗ ТА КЛАСИФІКАЦІЯ МІЖСАЙТОВОГО СКРИПТИНГУ

1.1 Поняття міжсайтового скриптингу або XSS

Міжсайтовий скриптинг (XSS) – це тип ін’єкцій, під час яких шкідливий код вставляється до коду веб-додатків. XSS-атаки відбуваються, коли зловмисник використовує веб-програму для пересилання шкідливого коду, в більшості випадків, у формі скрипту на стороні браузера, іншому юзерові. Вразливості, які дозволяють цим атакам бути успішними, досить широко поширені та виникають у будь-якому місці, де веб-додаток використовує вхідні дані від користувача в межах вихідних даних, які він генерує, без їх перевірки чи кодування [1].

У 2022 році Open Web Application Security Project (OWASP) опублікував список топ-10 уразливостей (рисунок 1.1), які часто зустрічаються в мережі Інтернет. Міжсайтовий сценарій, також відомий як міжсайтовий скриптинг, є сьомим за частотою та найбільш критичним ризиком безпеки веб-додатків, що означає виявлення атак XSS, все ще має враховуватися важливим з точки зору кібербезпеки [2].

No.	Types of Security Threats
1	Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities (XXE)
5	Broken Access Control
6	Security Misconfiguration
7	Cross-Site Scripting (XSS)
8	Insecure Deserialization
9	Using Component with Known Vulnerabilities
10	Insufficient Logging & Monitoring

Рисунок 1.1 – Типи вразливостей за популярністю (за даними OWASP)

Вплив використаної вразливості XSS на веб-програму може сильно відрізнятися залежно від конкретної атаки. Виконуючи код сценарію в поточному контексті користувача, зловмисники можуть викрасти файли cookie сеансу та здійснити викрадення сеансу, щоб видати себе за жертву або захопити її обліковий запис. У поєднанні з соціальною інженерією це може призвести до розкриття конфіденційних даних, CSRF-атак (якщо зловмисник має доступ до анти-CSRF-токенів) або навіть встановлення зловмисного програмного забезпечення.

Якщо жертва має права адміністратора в цільовій програмі, успішну XSS-атаку можна використати для підвищення привілеїв, а потім для виконання коду на сервері (прочитайте наш аналіз інциденту apache.org Jira, щоб дізнатися, як це може статися). Оскільки веб-інтерфейси API HTML5 надають браузеру дедалі більше доступу до локальних даних і апаратного забезпечення, зловмисники потенційно можуть використовувати вразливості XSS для доступу до ваших локальних ресурсів, від даних у сховищі вашого браузера до камери та мікрофона. XSS є великою проблемою безпеки веб-додатків [3].

Зловмисник може використовувати XSS, щоб надіслати шкідливий сценарій нічого не підозрюючому юзеру. Браузер користувача не може дізнатися, що скрипту не можна довіряти, і виконає сценарій. Оскільки він вважає, що сценарій надійшов із надійного джерела, зловмисний код може отримати доступ до будь-яких файлів cookie, маркерів сеансу чи іншої конфіденційної інформації, яка зберігається браузером і використовується на цьому сайті. Ці сценарії можуть навіть переписати вміст сторінки HTML.

Веб-сторінка містить текст і розмітку HTML, які доступні на сервері та читаються клієнтським браузером. Веб-сайти, які створюють лише статичні сторінки, можуть мати повний контроль над тим, як клієнт інтерпретує ці сторінки. Додатки, які створюють динамічні сторінки, не мають повного контролю над тим, як їхні результати інтерпретуються клієнтом. XSS-атаки можуть відбуватися на рівні програми, коли серверна програма (тобто динамічна веб-сторінка) використовує необмежений вхід через HTTP-запит, базу даних або файли без будь-якої перевірки,

що дозволяє ін’єкції зловмисного коду (вказано на рисунку 1.2). Використання таких уразливостей дозволяє хакерам викрадати конфіденційну інформацію та виконувати інші шкідливі дії. Приклади вразливостей XSS включають нездатність закодувати вихідні дані HTML у веб-браузері та нездатність перевірити вхідні дані для веб-додатків, як вказано на рисунку 1.3.

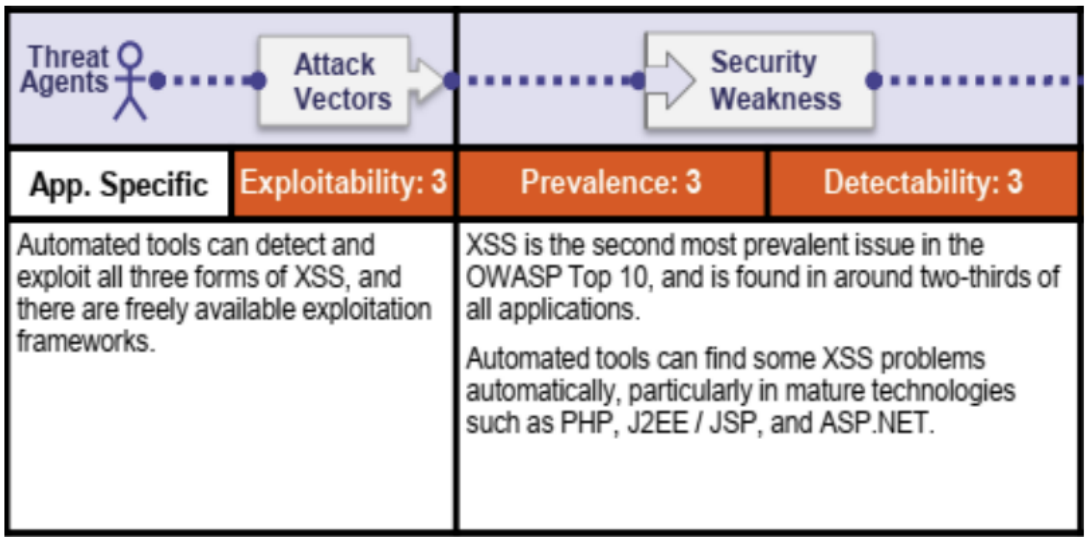

Рисунок 1.2 – Вектор XSS атаки на вразливість

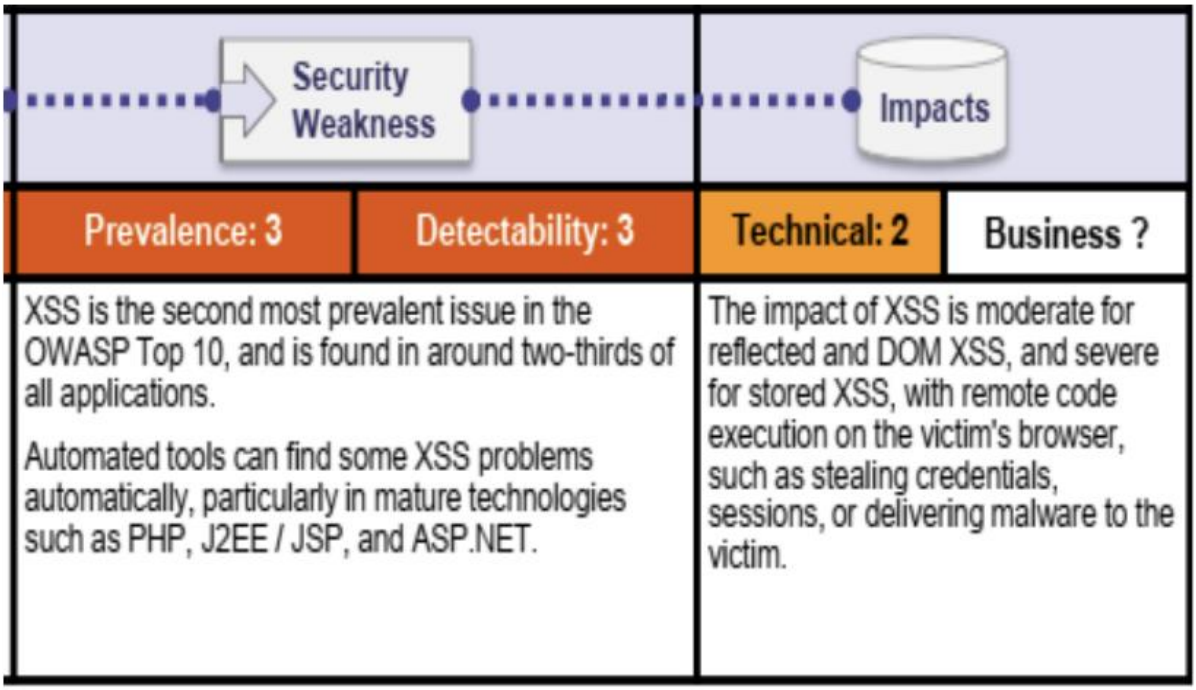

Рисунок 1.3 – Вектор впливу XSS атак

Наслідки атаки XSS зазвичай залежать від типу програми, функціональних можливостей і даних, а також привілеїв користувача. Наслідки міжсайтового скриптингу можуть бути серйозними, включаючи крадіжку особистих даних, отримання конфіденційної інформації, відмову в обслуговуванні, зміну способу роботи веб-браузера та навіть розповсюдження черв'яків, які отримують доступ до комп'ютера користувача та переглядають історію браузера користувача або дистанційно керують ним.

XSS-атаки настільки популярні, тому що їх досить легко запустити та вони не вимагають великих технічних навичок. Деякі атаки XSS можна запустити, маючи лише базові знання JavaScript і HTML. Це дозволяє зловмисникам легко навчитися здійснювати XSS-атаки [4].

Міжсайтові сценарії відбуваються, коли:

1. Дані надходять у веб-програму через ненадійне джерело, найчастіше через веб-запит.
2. Дані включаються в динамічний вміст, який надсилається веб-користувачу без перевірки на шкідливий вміст.

Шкідливий вміст, який надсилається веб-переглядачу, часто має форму сегмента JavaScript, але також може включати HTML, Flash або будь-який інший тип коду, який може виконувати браузер. Різноманітність атак на основі XSS майже безмежна, але зазвичай вони включають передачу зловмиснику особистих даних, як-от файлів cookie або іншої інформації про сеанс, перенаправлення жертви на веб-контент, який контролює зловмисник, або виконання інших зловмисних операцій на комп'ютері користувача. під виглядом вразливого сайту.

Можливості зловмисника, який запускає XSS-атаку, можуть бути досить широкими. Компаніям важко відстежити XSS-атаку, тому що зловмисник може здійснити XSS-атаку та використати вразливість багатьма способами. Зловмисник, який використовує міжсайтові сценарії, зазвичай може:

- видавати себе за жертву/користувача або маскуватися під неї;
- виконати будь-яку дію, яку жертва/користувач здатна виконати;

- зчитувати будь-які дані, до яких жертва/користувач має доступ;
- захопити облікові дані жертви/користувача;
- виконати віртуальну дефейсацію веб-сайту;
- впровадити на веб-сайт функції трояна.

1.2 Класифікація XSS загроз

За загальновідомою класифікацією існує три основних види міжсайтових сценаріїв:

- збережений (stored) XSS;
- відображений (reflected) XSS;
- DOM XSS.

Збережені атаки – це ті, коли сценарій постійно зберігається на цільових серверах, наприклад у базі даних, у форумі повідомлень, у журналі відвідувачів, у полі коментарів тощо. Потім жертва отримує шкідливий сценарій із сервера, коли він запитує збережений інформації (див. рисунок 1.4).

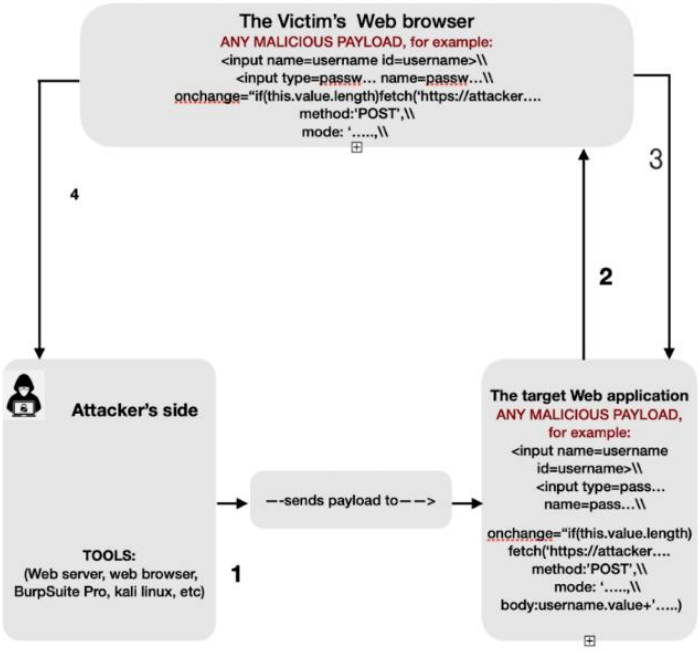

Рисунок 1.4 – Приклад збереженого міжсайтового сценарію

Приклади: Поле «пошуку» (де користувач може ввести код JavaScript/HTML у текстове поле), Вхід, реєстрація, коментарі тощо. Принаймні один із цих випадків має бути присутнім на одній з веб-сторінок будь-якого веб-сайту. Тут маємо сценарій: супротивник зазвичай вводить свій код або просто їх корисне навантаження в полі «Коментар» цільового веб-сайту. Автоматично натиснувши «Enter» або кнопку «Надіслати», їх коментар буде опубліковано та інші користувачі можуть це побачити. Тому їхні повідомлення зберігаються в базі даних цього веб-сайту, як і зловмисні сценарії. Таким чином при кожному перегляді цей сценарій буде запускатися у цільового користувача [5].

Коротке пояснення [5]:

– Зловмисник публікує свої коментарі (зловмисне навантаження JavaScript), наприклад, на сайті блогу. Коли користувач переходить на певний веб-сайт, йому надається фрагмент шкідливого коду зловмисника як частину оригінальної веб-сторінки (цього блогу).

– Таким чином, несвідомо під час відвідування сайту жертвою (нашим користувачем) вона ризикує запустити цей код. Ось тут і вступає в гру «дезінфекція, перевірка введених користувачем даних».

Ще треба виділити “сліпий” міжсайтовий сценарій. Сліпий міжсайтовий сценарій є формою збереженого XSS. Зазвичай це відбувається, коли корисне навантаження зловмисника зберігається на сервері та повертається жертві з серверної програми. Наприклад, у формах зворотного зв’язку зловмисник може надіслати зловмисне корисне навантаження за допомогою форми, і коли серверний користувач/адміністратор програми відкриє надіслану форму зловмисника через серверну програму, корисне навантаження зловмисника буде виконано.

Відображена XSS-атака, яку зазвичай називають «непостійною XSS-атакою», є найбільш розповсюдженою, хоча й не найнебезпечнішою.

Найпростіший різновид міжсайтових сценаріїв, відображених XSS-атак відбувається, коли веб-програма отримує дані з HTTP-запиту, а потім негайно відповідає без перевірки чи кодування даних (див. рисунок 1.5). Оскільки програма

жодним чином не обробляє дані, зловмисник може легко здійснити атаку на інших користувачів на основі сценаріїв.

Під час відображеної атаки впроваджений сценарій представляє себе як повідомлення про помилку, результат пошуку чи подібну дію через шкідливе посилання. Після натискання це посилання виконає сценарій, який дозволяє впровадженому коду перейти на вразливий сайт і «відобразити» назад у браузері користувача. Браузер виконує код, оскільки вважає сайт надійним джерелом. Потім сценарій може виконувати будь-які дії, доступні користувачеві під час сеансу, а також захоплювати будь-які дані, передані користувачем під час сеансу [6].

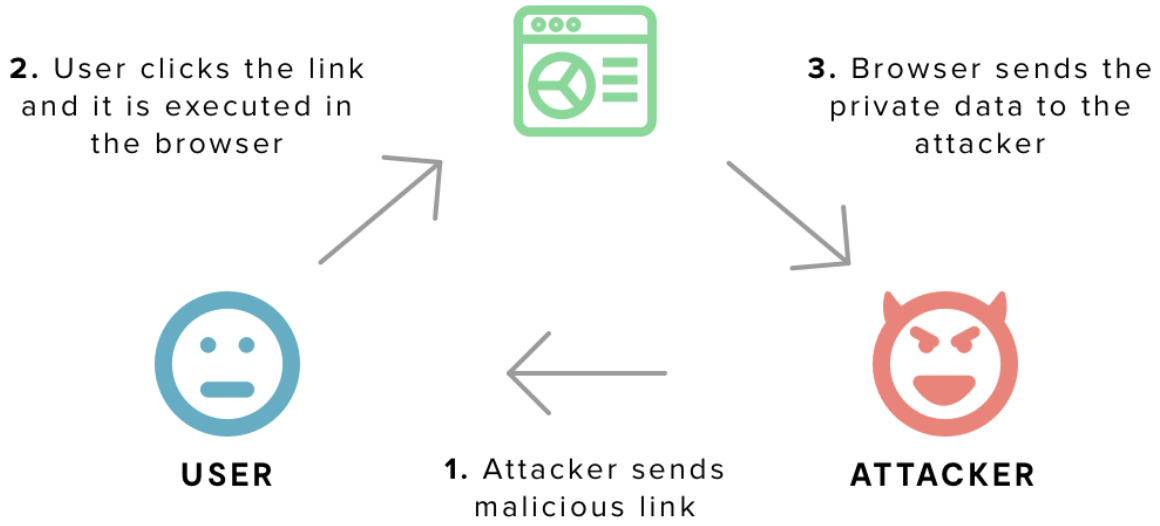

Рисунок 1.5 – Приклад відображеного XSS

Уразливості XSS на основі DOM зазвичай виникають, коли JavaScript бере дані з джерела, яке контролює зловмисник, наприклад URL-адресу, і передає їх до приймача, який підтримує динамічне виконання коду, наприклад eval() або innerHTML. Це дозволяє зловмисникам запускати шкідливий JavaScript, що зазвичай дає їм змогу захоплювати облікові записи інших користувачів [7].

Щоб здійснити XSS-атаку на основі DOM, вам потрібно розмістити дані в джерелі, щоб вони поширювалися до приймача та викликали виконання довільного JavaScript (див. рисунок 1.6).

Найпоширенішим джерелом для DOM XSS є URL-адреса, доступ до якої зазвичай здійснюється за допомогою об’єкта window.location. Зловмисник може створити посилання, щоб відправити жертву на вразливу сторінку з корисним навантаженням у рядку запиту та фрагментувати частини URL-адреси. За певних обставин, наприклад, коли націлено на сторінку 404 або веб-сайт, на якому працює PHP, корисне навантаження також можна розмістити в шляху [7].

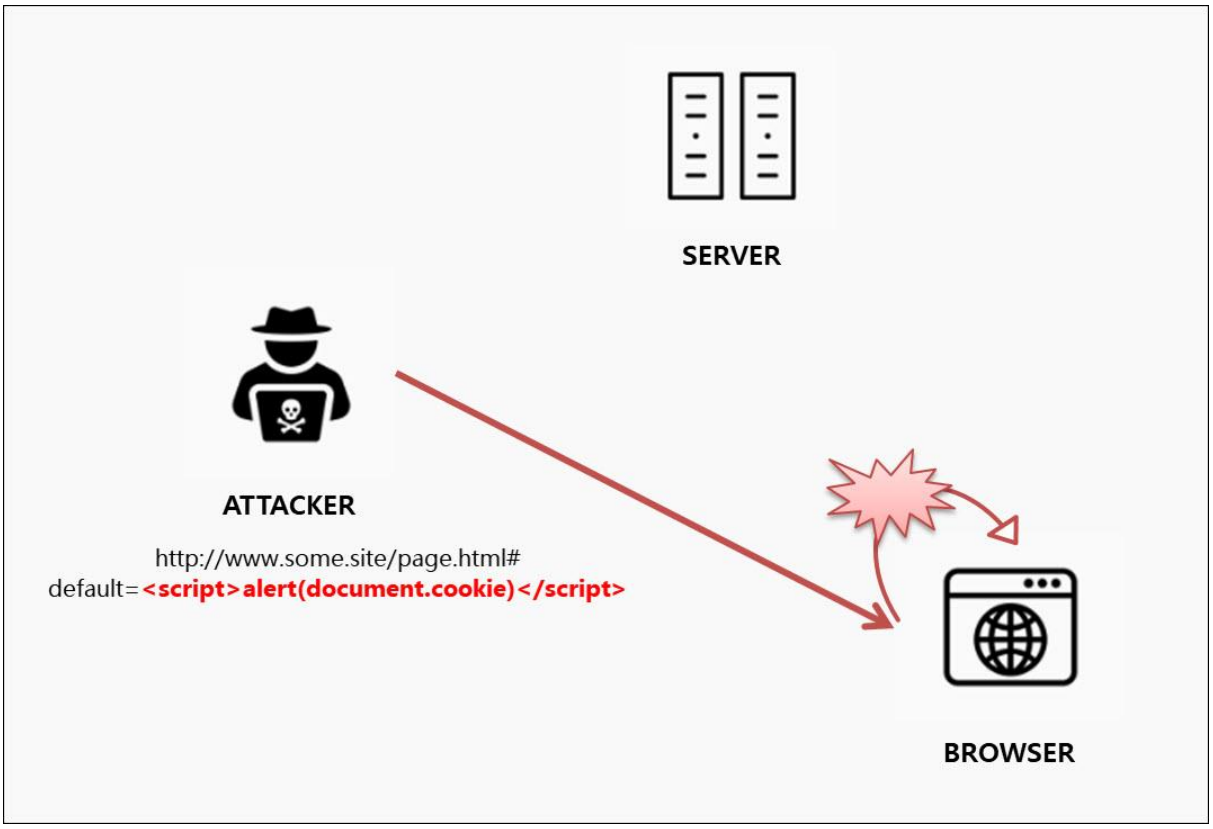

Рисунок 1.6 – Приклад DOM XSS

Протягом багатьох років більшість людей вважали ці типи скриптингу (збережені, відображені, DOM) трьома різними типами XSS, але насправді вони дуже часто збігаються. Ви можете мати як збережений, так і відображений XSS на основі DOM. Ви також можете мати збережений і відображений XSS не на основі DOM, але це заплутано, тому, щоб прояснити речі, починаючи приблизно з середини 2012 року дослідницька спільнота запропонувала та почала використовувати два нові терміни, щоб допомогти організувати типи XSS, які можуть виникнути [8]:

- Серверний XSS;
- Клієнтський XSS.

Серверний XSS виникає, коли ненадійні дані, надані користувачем, включені у відповідь HTTP, згенеровану сервером. Джерелом цих даних може бути запит або збережене розташування. Таким чином, ви можете мати як відображений сервер XSS, так і збережений сервер XSS. У цьому випадку вся вразливість міститься в коді на стороні сервера, а браузер просто рендерить відповідь і виконує будь-який дійсний сценарій, вбудований у нього.

Клієнтський XSS виникає, коли ненадійні дані, надані користувачем, використовуються для оновлення DOM за допомогою небезпечного виклику JavaScript. Виклик JavaScript вважається небезпечним, якщо його можна використовувати для введення дійсного JavaScript у DOM. Це джерело цих даних могло бути з DOM або їх міг надіслати сервер (через виклик AJAX або завантаження сторінки). Остаточним джерелом даних міг бути запит або збережене місце на клієнті чи сервері. Таким чином, ви можете мати як XSS відображеного клієнта, так і XSS збереженого клієнта.

Класифікація залежно від місця знаходження показана на рисунку 1.7.

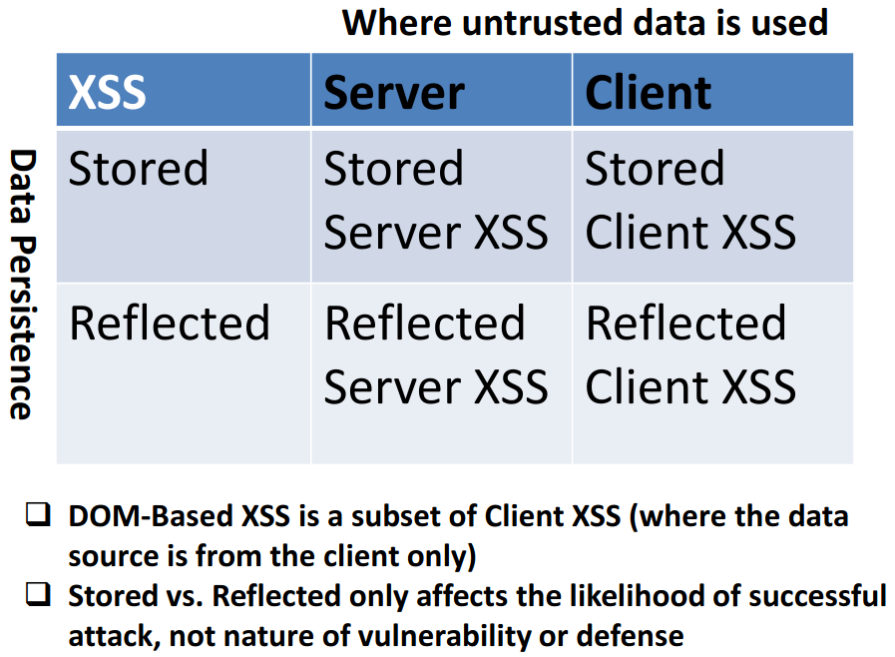

Рисунок 1.7 Класифікація XSS за місцем знаходження

З цими новими визначеннями визначення XSS на основі DOM не змінюється. XSS на основі DOM — це просто підмножина клієнтського XSS, де джерело даних знаходиться десь у DOM, а не на сервері [8].

З огляду на те, що як серверний XSS, так і клієнтський XSS можна зберігати або відображати, ця нова термінологія призводить до простої, чіткої матриці 2 x 2 із клієнтським і серверним XSS на одній осі та збереженим і відображеним XSS на іншій осі, як показано на рисунку 1.7.

1.3 Особливості міжсайтового скриптингу у хмарних системах

Атаки міжсайтових сценаріїв (XSS) у хмарних системах передбачають введення зловмисником шкідливих сценаріїв у законний веб-сайт або веб-програму, яка працює на хмарному сервері. Ці шкідливі сценарії виконуються у веб-переглядачі жертви, яка отримує доступ до скомпрометованого сайту чи програми, дозволяючи зловмиснику викрасти конфіденційну інформацію, таку як облікові дані для входу, номери кредитних карток або іншу особисту інформацію (PII), або захопити сеанс жертви і виконувати несанкціоновані дії [9].

XSS-атаки в хмарних системах можуть мати значний вплив як на постраждалого хмарного постачальника, так і на його клієнтів. Деякі з можливих наслідків атак XSS включають:

1. Витік даних: якщо зловмиснику вдасться викрасти конфіденційні дані, такі як ідентифікаційна чи фінансова інформація, хмарний постачальник і його клієнти можуть зазнати серйозних репутаційних і фінансових втрат.
2. Порушення служби. Якщо атака XSS спричиняє стан відмови в обслуговуванні (DoS) у хмарній інфраструктурі, це може вплинути на доступність і надійність хмарних служб.
3. Невідповідність нормативним вимогам: якщо до постачальника хмарних послуг поширюються такі вимоги, як PCI-DSS або HIPAA, XSS-атака може призвести до невідповідності та нормативних штрафів.

4. Юридична відповідальність: якщо постачальник хмарних послуг не забезпечує належним чином свої послуги та дані своїх клієнтів, він може зіткнутися з судовим позовом і відповідальністю за збитки.

Щоб запобігти XSS-атакам у хмарних системах, хмарні провайдери повинні впроваджувати заходи безпеки, такі як перевірка вхідних даних, вихідне кодування та політики безпеки вмісту (CSP). Клієнти хмари також можуть захистити себе, дотримуючись належної гігієни безпеки, наприклад, використовуючи надійні паролі, оновлюючи програмне забезпечення та обережно натискаючи посилання чи завантажуючи вкладення з невідомих джерел.

У хмарних середовищах атаки XSS може бути складніше виявити та подолати, оскільки хмарні системи зазвичай включають розподілену інфраструктуру та складну взаємодію між кількома компонентами. Крім того, хмарні постачальники повинні керувати декількома орендарями, кожен зі своїми програмами, даними та вимогами до безпеки [10].

Щоб вирішити ці проблеми, хмарні постачальники можуть використовувати різні методи, зокрема [10]:

1. Автоматизоване сканування вразливостей: постачальники хмарних послуг можуть використовувати автоматизовані інструменти для сканування на наявність уразливостей XSS у веб-додатках, які працюють на їхніх серверах.

2. Ізоляція кількох клієнтів: хмарні постачальники можуть ізолювати програми та дані клієнтів, щоб запобігти впливу XSS-атак на інших клієнтів.

3. Виявлення загроз у режимі реального часу. Хмарні постачальники можуть використовувати алгоритми машинного навчання та інші вдосконалені методи для виявлення та реагування на атаки XSS у режимі реального часу.

Висновок за першим розділом

У цьому розділі було розглянуто означення загрози міжсайтовий сценарій, проведено аналіз цього типу загроз та його наслідків. Також, у другому пункті розділу

було розглянуто класифікацію XSS загроз, було описано стандартний поділ на збережені, відображені та DOM XSS загрози, зокрема, було запропоновано підхід для опису класифікації з поділом на серверні та клієнтські загрози.

У третьому пункті було розглянуто вплив міжсайтового скриптингу на хмарні середовища. Висновком є те, що атаки XSS у хмарних системах можуть мати серйозні наслідки, такі як витік даних, збій у роботі служби, недотримання нормативних вимог і юридична відповідальність. Хмарні постачальники та клієнти повинні вживати заходів для запобігання атакам XSS, реалізуючи заходи безпеки, дотримуючись належної гігієни безпеки та використовуючи передові методи безпеки.

РОЗДІЛ 2
ТЕСТУВАННЯ ДОДАТКІВ НА МІЖСАЙТОВИЙ СКРИПТИНГ

2.1 Аналіз методів виявлення XSS загроз

Було проведено багато попередніх досліджень, спрямованих на визначення способів використання міжсайтового скриптингу для контролю та налаштування роботи веб-сторінки. Кілька платформ пропонують способи тестування або використання вразливостей XSS. Кілька таких веб-сайтів: Web Goat, Acunetix, Pentest Tools і Burp. Ці параметри дозволяють користувачам вводити адреси веб-сайтів і перевіряти їх на наявність уразливостей. Оскільки більшість атак XSS включає JavaScript, усі інструменти виявлення повинні мати можливість виявляти шкідливий JavaScript. Однак тестування безпеки, яке вони можуть надати, буде обмежено лише системами, якими володіє користувач або має дозвіл працювати з ними [11].

Щоб мінімізувати атаки XSS, організації повинні оцінити код своїх веб-додатків і усунути будь-які вразливості XSS. Щоб успішно ідентифікувати потенційні міжсайтові сценарії, організації повинні прийняти кілька заходів, зокрема:

- Оцінити будь-який об’єкт, який може відкрити або запустити браузер. Це включає повідомлення електронної пошти, вкладення, завантаження, веб-сторінки та будь-який інший документ, який містить HTML-посилання.
- Виконати швидкий статичний аналіз кожного об’єкта, оцінюючи його на шкідливі можливості та посилання, відомі сигнатури атак, структурні відхилення та інші аномалії.
- Виконати повний аналіз поведінки, повністю виконавши кожен об’єкт і перевіривши його на наявність методів ухилення та зловмисних дій.
- Відстежувати мережу на наявність побічних ефектів, спричинених зловмисним програмним забезпеченням, що працює в мережі, наприклад впровадженням коду, зв’язком зловмисного програмного забезпечення з командними й контрольними серверами та іншими аномальними діями.

Перевірка коду спрямована на виявлення недоліків безпеки в програмах разом із точними основними причинами. Він передбачає аудит вихідного коду програми, щоб переконатися, що вона самозахищається у своєму середовищі. Відповідно до OWASP: «Якщо код не перевірено на наявність дірок у безпеці, ймовірність того, що програма має проблеми, становить майже 100%». Для виконання цього завдання можна використовувати автоматизовані інструменти. Однак, на жаль, вони не розуміють контексту (завжди потрібна перевірка людиною), і тут вступає в дію перевірка коду безпеки вручну [12].

Якщо перевірку коду вручну виконано правильно, наступний тест на проникнення з використанням автоматизованих інструментів має виявити незначну кількість вразливостей або їх відсутність. Цікаво, що OWASP надає детальний посібник із ручного перегляду коду на наявність уразливостей міжсайтових сценаріїв, включаючи повний посібник з ручного тестування для відображених, збережених і вразливостей XSS на основі DOM для вашої довідки. Для виявлення поширених уразливостей XSS можна використати наступні процеси вручну:

Визначте код, який виводить введені користувачем дані: коди, які виводять введені користувачем дані без належної обробки, загрожують уразливості XSS. Натисніть Ctrl + U, щоб переглянути джерело виводу сторінки з браузера та перевірити, чи ваш код розміщено в атрибуті. Якщо так, вставте наступний код і перевірте, щоб переглянути результат: «onmouseover= alert(‘hello’);» Ви можете перевірити, щоб переглянути вихідні дані за допомогою цього сценарію: <script>alert(document.cookie);</script>; Проте, якщо код, який ви переглядаєте, фільтрує символи <i>, спробуйте натомість такий сценарій: &{alert(‘hello’);}

Перевірте, чи вивід закодовано. Переконайтеся, що HtmlEncode використовується для кодування виводу HTML, який містить будь-який тип введення. Також переконайтеся, що UrlEncode використовується для кодування рядків URL. Вхідні дані можуть надходити з рядків запитів, полів форм, файлів cookie, HTTP-заголовків і вхідних даних, зчитаних із бази даних, особливо якщо інші програми спільно використовують базу даних. Якщо ці кодування відсутні, ваша

програма знаходиться під загрозою вразливості XSS. Крім того, шляхом кодування даних ви перешкоджаєте браузеру розглядати HTML як виконуваний сценарій.

Визначте код, який обробляє URL-адреси: код, який керує URL-адресами, може загрожувати XSS та іншими вразливими місцями. Перегляньте свій код і системи таким чином:

- Перевірте, чи оновлено веб-сервер. Якщо на вашому веб-сервері не встановлено найновіші виправлення безпеки, він може бути вразливим до атак обходу каталогу:
- Якщо ваш код фільтрує «/», зловмисник може легко обійти фільтр, використовуючи URL-кодування (відсоткове кодування) для того самого символу. Наприклад, відсоткове кодування для «/» — «%c0f%af», яке можна використовувати для обходу фільтра, як показано в такій URL-адресі: http://www.abc.com/..%c0f%af ../winnt.
- Якщо ваш код обробляє введення рядка запиту, переконайтеся, що він обмежує вхідні дані та виконує перевірку меж. Також переконайтеся, що код не є вразливим, якщо зловмисник вводить величезну кількість даних через параметр рядка запиту, як-от URL-адресу, наведену тут: http://www.abc.com/test.aspx?var=InjectHugeAmountOfDataHere.

Модульне тестування вашого коду: модульне тестування (unit testing) — це метод тестування програмного забезпечення, за допомогою якого окремі одиниці вихідного коду перевіряються, щоб визначити, чи придатні вони для використання. Модульне тестування має на меті виокремити кожну частину програми та показати, що окремі компоненти правильні. Використовуйте модульне тестування, щоб переконатися, що певний біт даних правильно екранований. Модульне тестування допомагає виявити XSS та інші недоліки на ранніх етапах циклу розробки. Якщо можливо, тестуйте кожне місце, де відображаються надані користувачем дані. Після того, як ви знайдете та виправите помилку XSS у своєму коді, розгляньте можливість додавання для неї регресійного тесту.

Виконайте такі основні тести своєї програми: створіть тестовий профіль користувача та використовуйте цей профіль для взаємодії з вашою програмою.

Вставте рядки, які містять метасимволи HTML і JavaScript, у всі вхідні дані програми, такі як форми, параметри URL-адреси, приховані поля або значення файлів cookie.

Якщо ваша програма неправильно екранує цей рядок, ви побачите сповіщення та знатимете, що щось не так. Усюди, де ваша програма обробляє URL-адреси, надані користувачем, введіть такий код JavaScript: alert(0) або data:text/html,<script>alert(0)</script>. Усе це може допомогти виявити збережені помилки XSS.

Для тестування та виявлення вразливостей XSS у веб-додатках можна використовувати стандартні методи та інструменти тестування безпеки. Деякі з популярних методів тестування (вказані на рисунку 2.1), які можна використовувати, включають [13]:

Статичне тестування безпеки додатків (SAST): SAST використовується для захисту додатків шляхом перегляду вихідного коду для виявлення вразливостей або доказів відомих небезпечних методів, коли він не працює. Значною перевагою статичного аналізу коду є те, що вам не потрібно чекати, поки додаток буде розгорнуто в проміжному середовищі з тестовими даними. Замість цього ви можете просто протестувати код. Це забезпечує 100% охоплення коду перевіркою вразливостей і робить пошук вразливостей швидшим і дешевшим. Інструменти SAST використовують стратегію тестування білої скриньки, яка сканує вихідний код програм та їх компонентів для виявлення потенційних недоліків безпеки.

Динамічне тестування безпеки додатків (DAST): інструменти DAST взаємодіють із додатками, щоб виявити потенційні вразливості безпеки через інтерфейс. Інструменти DAST не мають доступу до вихідного коду; замість цього вони здійснюють атаки, використовуючи стратегію чорної скриньки для виявлення вразливостей. За допомогою динамічного аналізу перевірки безпеки виконуються під час запуску або виконання коду чи програми, що перевіряється. Техніка, відома як фаззинг, використовується в динамічних тестах для надсилання випадкових неправильно сформованих даних як вхідних даних до програми, щоб визначити, чи зможе вона виявити недоліки XSS.

Інтерактивне тестування безпеки додатків (IAST): IAST поєднує найкраще з SAST і DAST. Він аналізує код на наявність XSS та інших уразливостей у безпеці, коли програма працює, і взаємодіє з функціями програми.

Рисунок 2.1 – Класифікація автоматизованого тестування додатків

За допомогою методів, описаних вище, ви можете швидко протестувати та виявити вразливості XSS у своїх веб-додатках. Веб-сканери вразливостей, такі як Invicti, Acunetix, Veracode, Checkmarx та інші, є потужними інструментами, які можуть сканувати весь ваш веб-сайт або програму та автоматично перевіряти XSS та інші недоліки безпеки. Хоча вони часто не оптимізовані для конкретної програми, вони дозволяють швидко та легко знаходити більш очевидні вразливості. Крім того, вони реалізують більшість методів тестування веб-програм, розглянутих вище, і дозволять застосувати ці методи для сканування вашої веб-програми на наявність уразливостей. Потім він створить звіт про знайдені вразливості та спробує їх автоматично виправити [14].

2.2 Огляд проекту для тестування на XSS вразливості

Найкращий спосіб знайти недоліки – це перевірити код на безпеку та знайти всі місця, де вхідні дані HTTP-запиту могли б потрапити у вихідний HTML-код. Якщо одна частина веб-сайту є вразливою, існує велика ймовірність того, що є й інші проблеми.

Розглянемо проект на базі якого проведемо тестування нашого додатку. Проект є прикладом блогу який має декілька сторінок:

- сторінка регістрації;
- сторінка входу;
- домашня сторінка;
- сторінка статті;
- сторінка перегляду користувача.

Проект написаний мовою PHP 8.2, використовує Apache Server як сервер та MySql як базу даних. На локальному сервері проект розташований для тестування XSS загроз, у наступному розділі буде розглянуто розташування проекту на хмарній платформі (AWS) для побудови безпеки додатку у хмарному середовищі. Користувач може реєструватися на сайті, залогінитися, переглядати всі статті на домашній сторінці, а також певну одну статтю та залишати на ній коментарі. Увесь код проекту до змін міститься у Додатку А.

Етапи тестування було описано у попередньому пункті, для цього буде зробимо:

1. ручне тестування додатку на клієнтські вразливості.
2. ручне тестування додатку на серверні вразливості.
3. перевірку коду на потенційні XSS вразливості.

Тестування включатиме в себе як Black-box testing (коли тестування додатку не включає розгляд коду) так і White-box testing (коли тестування буде включати в себе аналіз вихідного коду). Після проведення аналізу веб додатку буде таблиці з знайденими вразливостями. Ця таблиця буде використана в наступному розділі при покращенні додатку для запобігання загроз.

2.3 Тестування додатку на клієнтські XSS вразливості

Клієнтський міжсайтовий сценарій (XSS) виникає, коли зловмисник вставляє виконуваний код браузера в одну відповідь HTTP. Введена атака не зберігається в самій програмі; вона є непостійною і впливає лише на користувачів, які відкривають зловмисно створене посилання або веб-сторінку третьої сторони. Рядок атаки включається як частина створеного параметра URI або HTTP, неналежним чином обробляється програмою та повертається жертві [15].

Проаналізуємо сторінки. Серед сторінок додатку, лише декілька приймають параметр з адресної строки. Розглянемо сторінку перегляду статті. Для перегляду статті в адресній строці передається назва статті та юзернейм, після чого за допомогою запиту до бази даних юзер може отримати статтю з певною назвою.

На рисунку 2.2 ми бачимо як виглядає передача параметру через адресну строку. Також ми бачимо, що дані з цього параметру використовуються на самій сторінці. Таким чином ми можемо припустити, що якщо навмисне передати до параметру щкідливий код, то він зможе спрацювати. Спробуємо додати туди простий Javascript код.

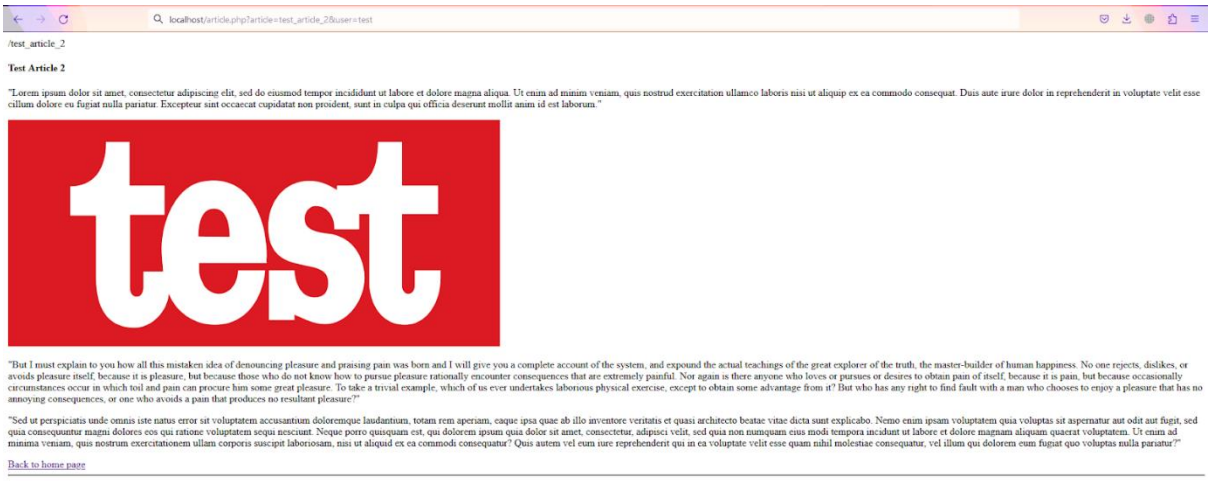

Рисунок 2.2 – Вигляд сторінку перегляду статті

Як ми бачимо на рисунку 2.3 ми додали шкідливий код як параметр і він спрацював. Звісно, в даному прикладі нічого шкідливого не вийшло, ми просто вивели повідомлення з потрібним нам текстом. Але також ми можемо додати більш складний код туди, наприклад, http://localhost/article.php?article=<img src=evil.js></img>. Таким чином зловмисник може передати будь-який шкідливий код і запустити його, а отже він може вкрасти дані про сесії користувача, куки або змінити кеш, що є надзвичайно небезпечним для звичайного користувача.

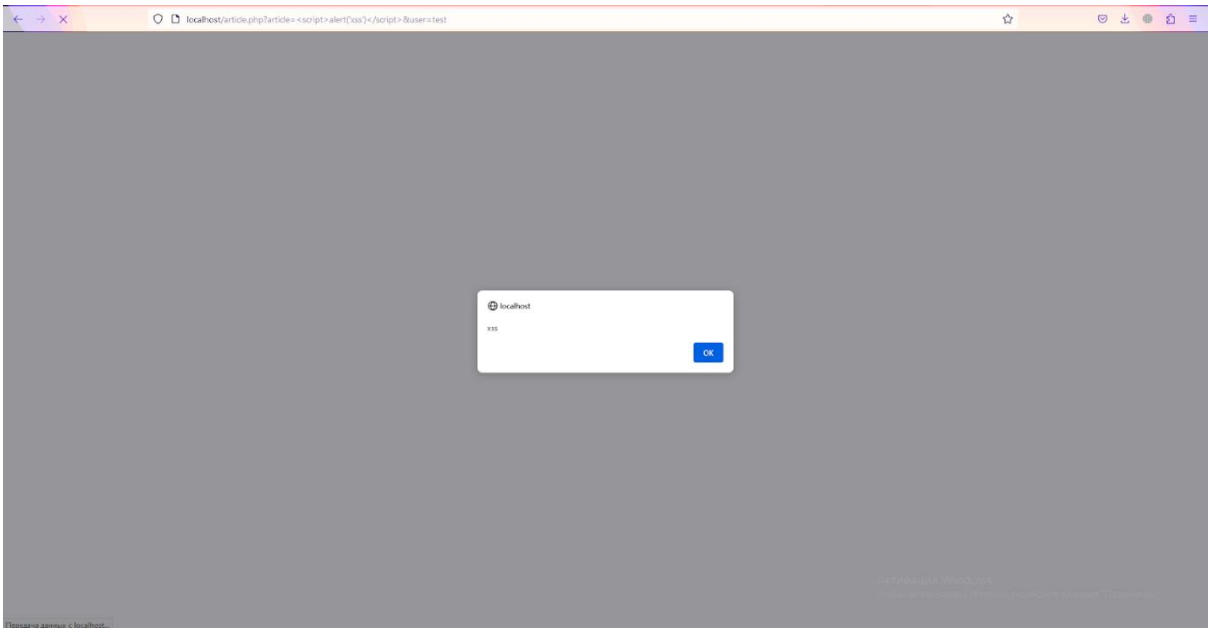

Рисунок 2.3 – Додавання шкідливого коду до адресної строки

Спробуємо змінити другий параметр user. Як видно на рисунку 2.4 ми можемо ввести шкідливий код в адресну строку і запустити його. Знову ж таки ми можемо вставити більш складний код і шкода від нього може збільшитися в рази. Наприклад, спробуємо за допомогою коду змінити усі адреси посилань на сторінці, щоб юзер після натискання на посилання перейшов на той сайт який нам потрібно. Для цього використаємо таке посилання у адресній строці: *http://localhost/article.php?article=test_article_2&user=%3Cscript%3Ewindow.onload%20=%20function()%20{var%20AllLinks=document.getElementsByTagName(%22a%22);AllLinks[0].href%20=%20%22http://badexample.com/malicious.exe%22;%20}%3C/script%3E.*

Після вводу посилання зі шкідливим кодом, ми отримали сторінку на якій був вставлений скрипт (див рисунок 2.4).

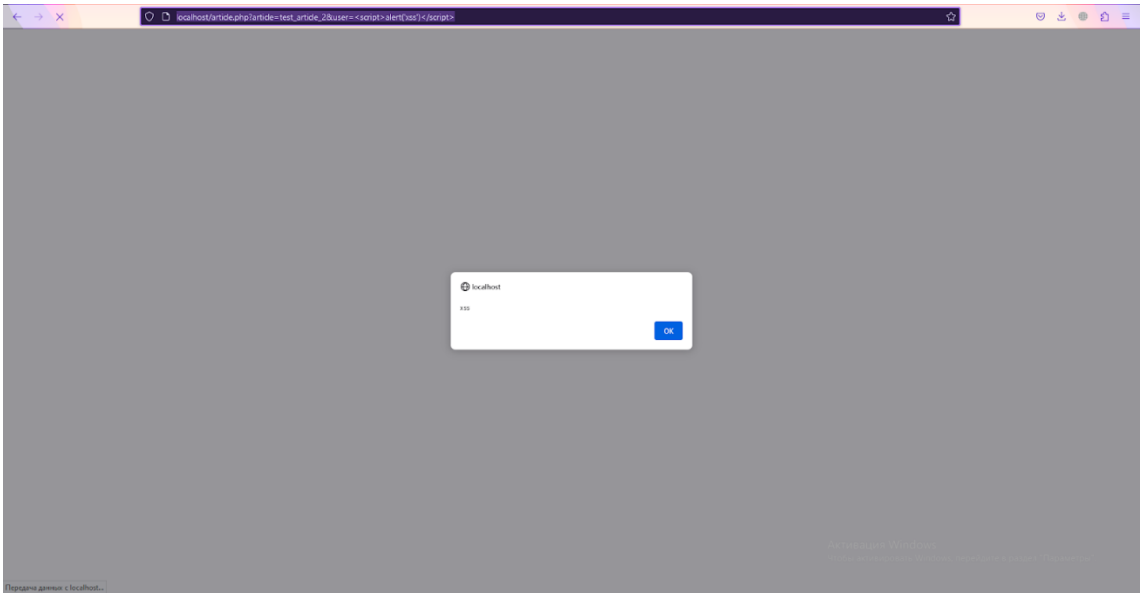

Рисунок 2.4 – Додавання шкідливого коду до адресної строки

Усі ж посилання на сторінці змінили адресу (див. рис. 2.5), тобто при натисканні на будь-яке посилання переведе юзера до *http://badexample.com/malicious.exe*. Після переходу на цей сайт запуститься файл з розширення exe, який може мати шкідливе навантаження, а отже буде надзвичайно небезпечним.

Рисунок 2.5 – Код сторінки після проведення відображеної атаки

Звісно, варіантів таких атак може бути безліч, тому що шкідливий код може відрізнятися. Наприклад, зловмисник може додати обробник події на будь-який елемент сторінки (onhover, onclick тощо) і коли жертва буде взаємодіяти зі сторінкою, то буде вразливою до будь-яких шкідливих дій. Самі посилання можуть розповсюджуватися у вигляді фішингу, тобто жертва може отримати замасковане

посилання, наприклад, на пошту, після переходу за ним вона перейде на сайт і нічого не підозрюючи буде атакована зловмисником.

Звісно, варіантів таких атак може бути безліч, тому що шкідливий код може відрізнятися. Наприклад, зловмисник може додати обробник події на будь-який елемент сторінки (onhover, onclick тощо) і коли жертва буде взаємодіяти зі сторінкою, то буде вразливою до будь-яких шкідливих дій. Самі посилання можуть розповсюджуватися у вигляді фішингу, тобто жертва може отримати замасковане посилання, наприклад, на пошту, після переходу за ним вона перейде на сайт і нічого не підозрюючи буде атакована зловмисником.

Також, крім перевірки сторінки перегляду статті, перевіримо і інші сторінки, які містять у собі параметри в адресній строці. Як ми бачимо з рисунків 2.6 та 2.7 інші сторінки не є вразливими до такого типу атаки.

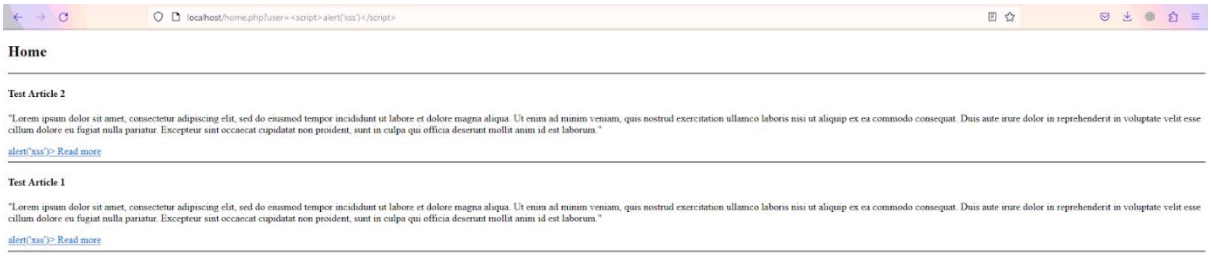

Рисунок 2.6 – Сторінка home.php

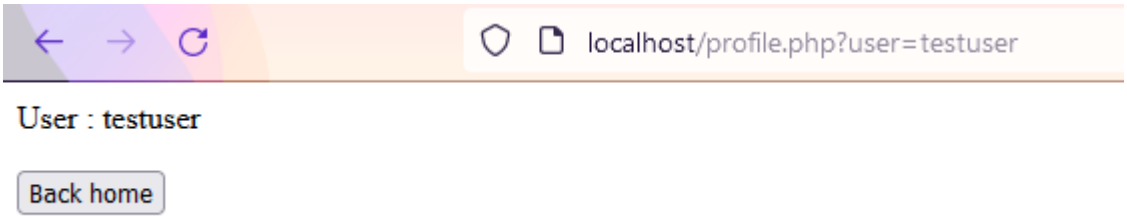

Рисунок 2.7 – Сторінка profile.php

Як видно з рис. 2.6. зміна параметру дещо змінила текст посилань, але код не запустився, тому ніякої шкоди дана сторінка не несе. Щодо сторінки на рис. 2.7, то там зовсім нічого не змінилося. Інші ж сторінки нашого додатку й зовсім не містять параметрів у посиланні, тому до них клієнтська XSS атака не може бути

застосована. Тому після тестування на такий тип атаки ми можемо зробити висновок, що до неї є вразливою тільки сторінка article.php.

2.3 Тестування на серверні XSS вразливості

Серверні міжсайтові сценарії є найнебезпечнішим типом міжсайтового сценарію. Веб-програми, які дозволяють користувачам зберігати дані, потенційно піддаються такому типу атак. Такий XSS виникає, коли веб-програма збирає вхідні дані від користувача, які можуть бути зловмисними, а потім зберігає ці введені дані в сховищі даних для подальшого використання. Введені дані не відфільтровано належним чином. Як наслідок, шкідливі дані виглядатимуть як частина веб-сайту та запускатимуться в браузері користувача з правами веб-програми. Оскільки ця вразливість зазвичай стосується принаймні двох запитів до програми, її також можна назвати XSS другого порядку [16].

Для використання серверного XSS не потрібне шкідливе посилання. Успішне використання відбувається, коли користувач відвідує сторінку зі збереженим XSS. Наступні етапи стосуються типового сценарію атаки XSS, що зберігається [10]:

- зловмисник зберігає шкідливий код на вразливій сторінці;
- користувач проходить автентифікацію в додатку;
- користувач відвідує вразливу сторінку;
- шкідливий код виконується браузером користувача.

Для тестування на такий тип вразливості потребується протестувати всі сторінки, які можуть зберігати інформацію введену користувачем у базі даних.

Спершу розглянемо сторінку входу користувача (див. рис. 2.8). На сторінці знаходиться форма входу, юзер вводить логін і пароль заздалегідь створеного користувача, після чого якщо логін і пароль правильні (співпадають с такими із бази даних) юзера переводить на домашню сторінку. Якщо пароль невірний, то для юзера виводится помилка.

Рисунок 2.8 – Сторінка входу користувача

Можемо протестувати дану сторінку, ввівши певний шкідливий код до полів входу. У нас вже є створений юзер з логіном ‘testuser’, спробуємо додати тег <script>alert(‘test stored xss’)</script>.

Як видно на рис. 2.9, після введення шкідливого коду до поля логін і паролю, спрацювала валідація, після чого користувач просто не зміг увійти до додатку. Спробуємо ввести шкідливий код трохи у іншому вигляді. Результат вводу цього коду можемо побачити на рис. 2.10.

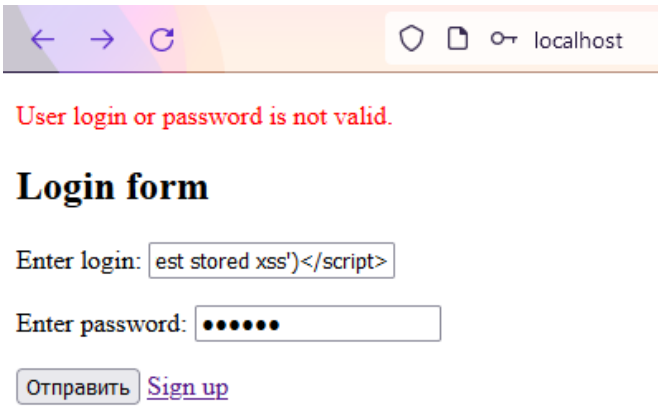

Рисунок 2.9 – Сторінка входу користувача після введення шкідливого коду

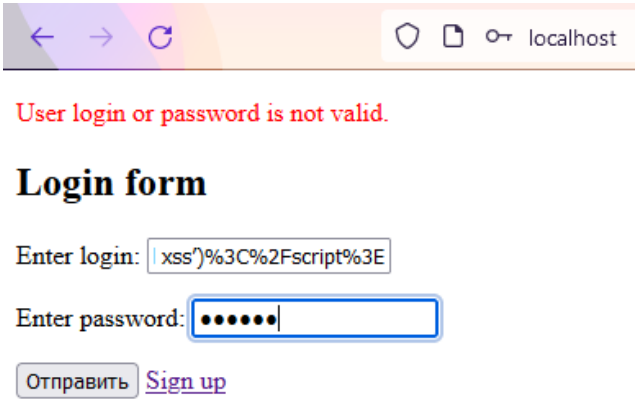

Рисунок 2.10 – Сторінка входу користувача після введення шкідливого коду

Після введення коду вказаного вище, результат не змінився, не змінився він і після вводу іншого коду: *"><script>alert(‘test stored xss’)</script>*. Тобто можемо зробити висновок, що зміна кодування у даному прикладі не грає ролі. Так як нам так і не вдалося зберегти якісь дані на сервері, можемо зробити висновок, що дана сторінка не містить у собі stored XSS вразливостію.

Розглянемо далі сторінку створення користувача (регістрації). Як ми бачимо на рис. 2.11 сторінка складається іх поля логін, паролю та підтвердження паролю. Після введення користувачем валідних даних, користувач створюється у базі даних, після чого користувача переводить на домашню сторінку.

На цій сторінці вже зберігаються дані, тобто потенційно зловмисник може зберегти тут зловмисний код. Отже, нам потрібно протестувати сторінку ввівши до полів форми приклади шкідливого коду. Для прикладу пропишемо у полі логіну простий код який буде виводити повідомлення.

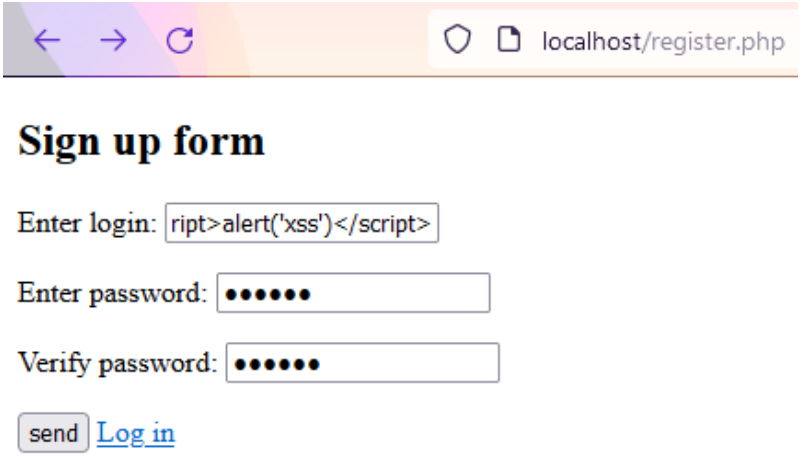

Рисунок 2.11 – Введення шкідливого коду у форму створення користувача

Після введення коду до форми і натискання кнопки підтвердження, у базі даних був створений юзер з ім’ям “*test<script>alert('xss')</script>*”. Тобто користувач був створений з ім’я, яке містить у собі шкідливий код, а отже при відображенні цього ім’я не сторінці цей код буде запускатися. Можемо переглянути базу даних і впевнитися, що користувач дійсно був створений (див. рис. 2.12).

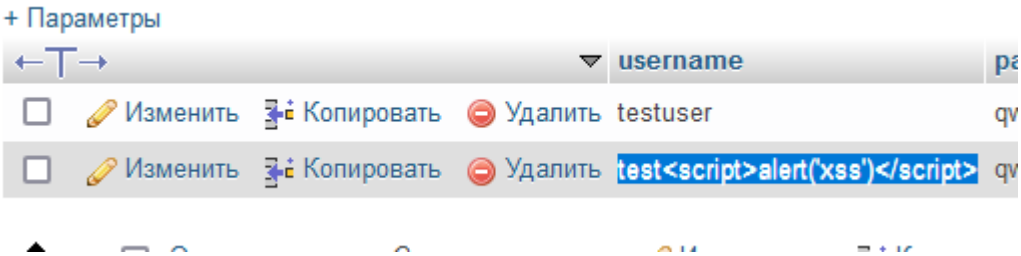

Рисунок 2.12 – Таблиця користувачів у базі даних

Наслідки збереження даного коду є дуже важкими. Наприклад, на сторінці перегляду статті відображається ім’я поточного користувача та коментарі до статті, які теж містять ім’я користувача. Якщо в першому варіанті шкідливе навантаження спрацює лише для поточного користувача, тобто самого зловмисника, то в другому варіанті якщо зловмисник залишить коментар до статті, будь який юзер якій відкриватиме дану статтю і коментар запустить у себе шкідливий код і стане жертвою шахрая. Результат цього можна побачити на рис. 2.13.

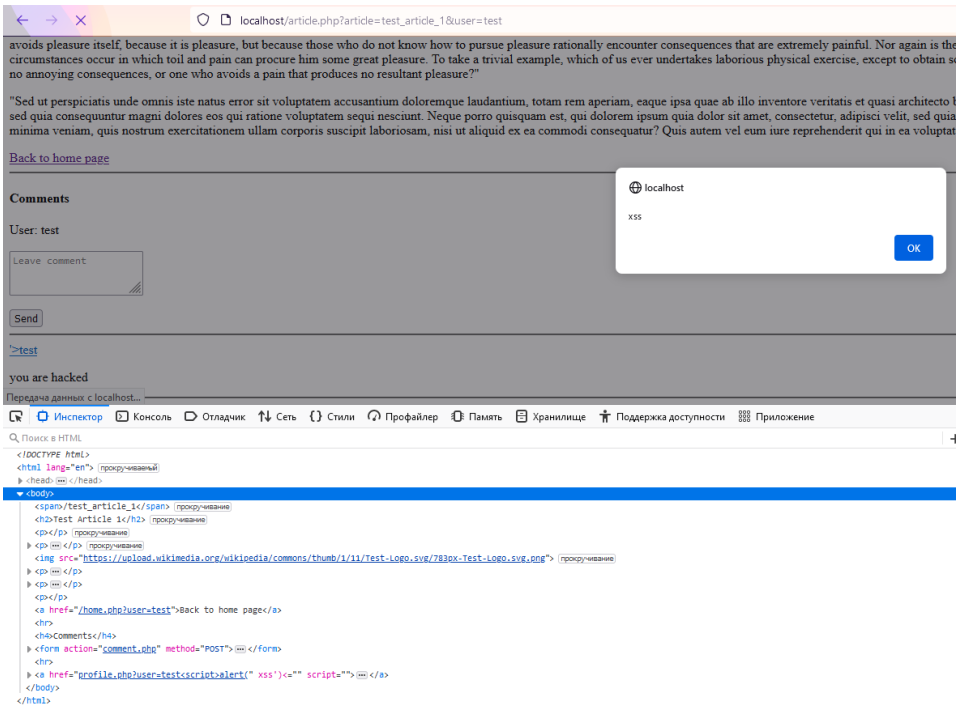

Рисунок 2.13 – Перегляд користувачем коментаря залишеного іншим користувачем, логін якого містить XSS

Також коли юзер перейде на сторінку користувача якого ми створили раніше, то код спрацює теж як показано на рис. 2.14.

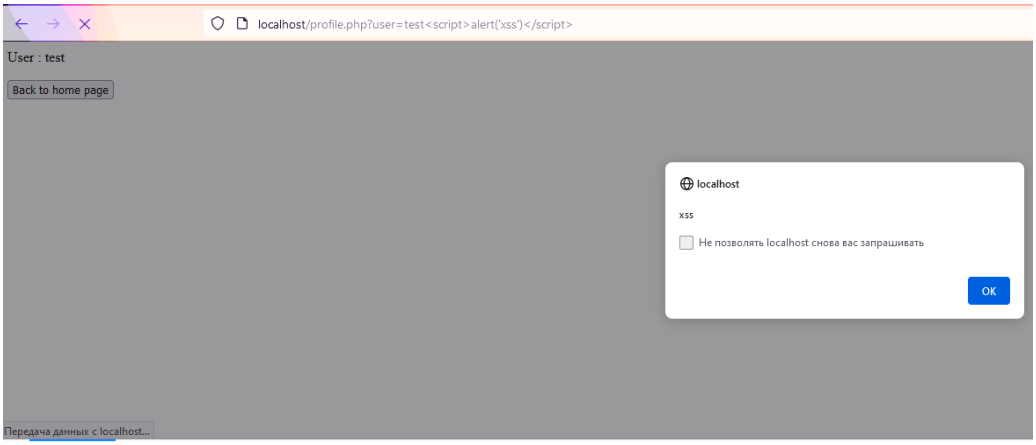

Рисунок 2.14 – Сторінка користувача з міжсайтовим скриптингом

Далі розглянемо сторінки на яких теж можуть вводитися дані. На домашній сторінці (див. рис. 2.15) немає жодних полів для введення даних, сторінка призначена для відображення даних з серверу. Отже, цю сторінку можемо вважати безпечною від збережених XSS вразливостей.

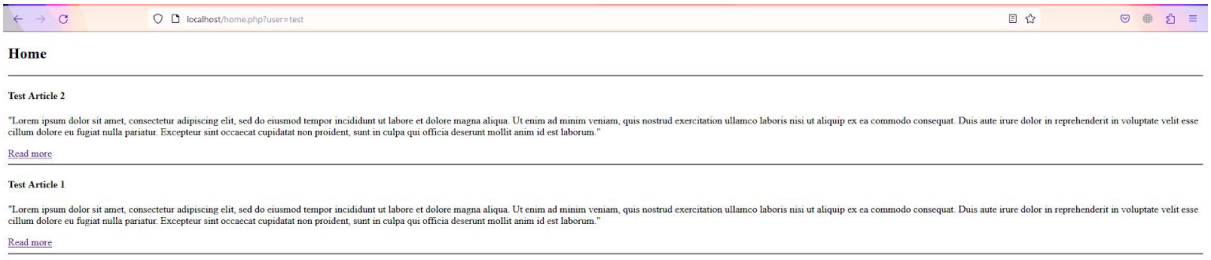

Рисунок 2.15 – Зовнішній вигляд домашньої сторінки

На рисунку видно, що в переліку статей для кожної з неї є посилання на окрему статтю. Перейшовши за цим посиланням користувач може відкрити сторінку перегляду статті, яку було розглянуто раніше.

На рис. 2.16 можемо бачити, що для створення коментаря в нижній частині сторінки наявне поле введення. Так як коментарі зберігаються у базі даних, то можна припустити, що дане поле може бути вразливим до міжсайтового скриптингу.

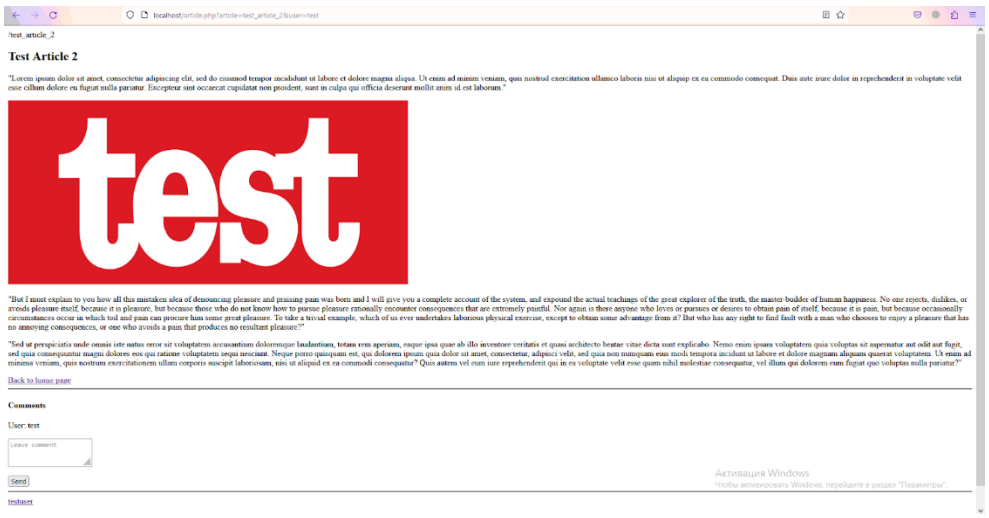

Рисунок 2.16 – Зовнішній вигляд сторінки перегляду статті

Для того щоб протестувати як і раніше використаємо приклад вразливого коду у тексті коментаря. для цього введемо наступний приклад коду: «<script>console.log(document.cookie)</script>» (див рисунок 2.17).

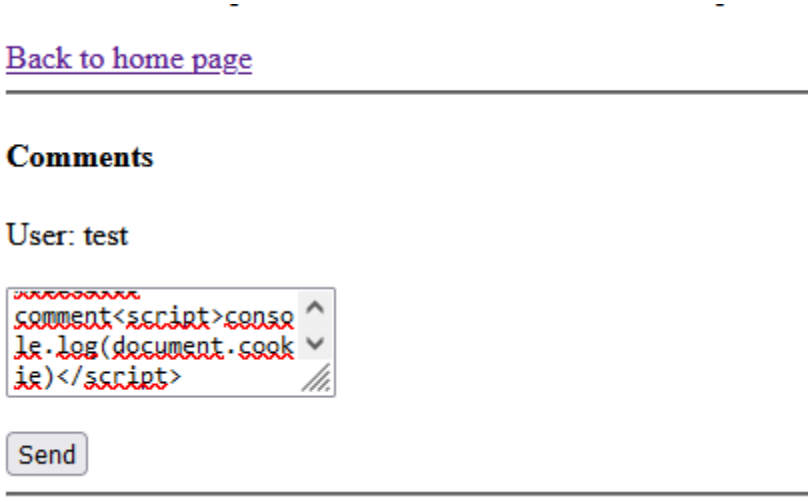

Рисунок 2.17 – Введення шкідливого коду у поле створення коментаря

Після натискання кнопки збереження коментар був успішно створений і записаний у базу даних. Як можна побачити на рис. 2.18, він був збережений разом із шкідливим кодом, тобто після того як цей запис буде отриманий із бази даних, то він буде отриманий теж разом із шкідливим кодом. Тому коли дані, які прийдуть із серверу запишуться у сторінку, цей код запуститься і користувач буде атакований зловмисником.

+ Параметры

userid	text	username	article
1	best article!!!	testuser	test_article_2
2	you are hacked	test<script>alert('xss')</script>	test_article_1
2	malicious comment<script>console.log(document.cook...	testuser	test_article_2

Рисунок 2.18 – Таблиця коментарів у базі даних

В фрагменті шкідливого коду, який ми ввели у полі створення коментаря ми виводимо в консоль браузера дані про куки. Таким чином використавши ‘console.log’ функцію, під час відкриття сторінки нічого візуально не відбудеться, проте якщо відкрити консоль, то можна побачити, що туди вивелись дані про куки на сайті як на рис. 2.19. Звісно, шкідливий код може це і не виводити навіть у консоль, а робити все у фоновому режимі, тому користувач може навіть не дізнатися, що він став жертвою міжсайтового сценарію. Це є прикладом чому такі вразливості є надзвичайно небезпечними.

Рисунок 2.19 – Результат роботи шкідливого коду збереженого у коментарі

Інші сторінки додатку не мають полів для введення даних, тому будемо вважати, що вони не є вразливим до серверних XSS загроз, так як створити якісь дані на цих сторінках неможливо, тому й зловмисний код вставити не можна теж. Отже, результатом тестування є те, що вразливими до цих типів загроз є сторінка перегляду статті та сторінка реєстрації користувача.

2.4 Тестування коду на потенційні вразливості

У попередніх пунктах було розглянуто тестування на серверні та клієнтські XSS вразливості. За допомогою тестування в локальному середовищі через браузер було розглянуті різні випадки введення даних користувачем. Таким чином було виявлено деякі загрози, але тестування у попередніх пунктах будувалося на принципі black box testing, тобто тестування не враховувало реалізацію додатку та його початкового коду. У цьому пункті ми розглянемо сам код, тобто застосуємо white box testing. Повний код можна знайти у Додатку А, в розділі буде ж розглядатися лише деякі його частини.

Розглянемо сторінку index.php (див. рис. 2.20). Ця сторінка відповідає за логін користувача.

```
<form action="home.php" method="POST">
    <p>Enter login: <input type="text" name="login" /></p>
    <p>Enter password: <input type="password" name="password" /></p>
    <input type="submit" value="Send">
</form>
```

Рисунок 2.20 – Фрагмент коду сторінки index.php

У цьому коді можна побачити два поля введення – "login" та "password", і обидва вони піддаються потенційній XSS атаці. Наприклад, якщо зловмисник вставить рядок, який вказано на рисунку 2.21, в поле "login", то коли користувач натисне кнопку "Send", скрипт буде виконано, і на сторінці з'явиться спливаюче вікно з повідомленням "XSS attack!".

```
"><script>alert('XSS attack!');</script>
```

Рисунок 2.21 – Фрагмент шкідливого коду

Аналогічно, якщо зловмисник вставить рядок, який вказано на рисунку 2.22, в поле "password", то коли користувач натисне кнопку "Send", скрипт буде виконано, і всі cookies з поточної сторінки будуть відправлені на сервер зловмисника.

```
"><script>
    document.location='http://attacker.com/steal.php?cookie='+document.cookie;
</script>
```

Рисунок 2.22 – Фрагмент шкідливого коду

Далі розглянемо сторінку home.php. Однією з найбільш серйозних проблем є те, що він використовує GET-запит для отримання імені користувача з URL. У цьому рядку: «$username = $_GET['user'];» , $username отримується з GET-запиту user. Це може бути дуже небезпечно, оскільки зловмисник може зламати сайт, змінивши значення user в URL-адресі, щоб отримати доступ до чутливої інформації або змінити дані користувача. Також, в коді не використовується фільтрація введених даних при виводі на сторінку, що може призвести до XSS-атак, оскільки несправжні дані можуть виконуватися як код.

На сторінці article.php було знайдено багато загроз ще у попередніх пунктах, після аналізу коду цієї сторінки було знайдено такі потенційні вразливості:

1. у рядку 7, відображається заголовок сторінки, який береться з параметра $_GET['article'] без фільтрації. Це означає, що можлива XSS-атака, якщо параметр article містить зловмисний код.
2. у рядках 36-38 та 41, параметри $_GET['user'] та $_GET['article'] використовуються без екранування для створення посилань. Це може призвести до XSS-атак, якщо параметри містять зловмисний код.
3. у рядках 49-51, параметри $_GET['user'] та $_GET['article'] використовуються без екранування для передачі значень в форму коментаря. Це може призвести до XSS-атак, якщо параметри містять зловмисний код.
4. у рядках 63-68, параметр text вводиться без фільтрації в коментарі до статті. Це може призвести до XSS-атак, якщо коментар містить зловмисний код.

Розглянемо сторінку register.php (рисунок 2.23). Він є схожим до коду сторінки index.php, тому вразливості будуть майже ідентичними. Потенційно вразливими будуть поля login, password verifypassword.

```
<form action="login.php" method="POST">
        <p>Enter login: <input type="text" name="login" /></p>
        <p>Enter password: <input type="password" name="password" /></p>
        <p>Verify password: <input type="password" name="verifypassword" /></p>
        <input type="submit" value="send">
</form>
```

Рисунок 2.23 – Фрагмент коду сторінки register.php

Потенційна XSS атак теж буде схожою як і до сторінки index.php. Наприклад, якщо зловмисник вставить в поле код з рисунку 2.21, то коли користувач натисне кнопку "Send", скрипт буде виконано, і на сторінці з'явиться спливаюче вікно з повідомленням "XSS attack!".

Розглянемо сторінку profile.php (див. рисунок 2.24). Ця сторінка виглядає безпечно. Вона просто виводить на екран параметр "user", який отримується з параметрів запиту GET і не обробляється ніяким чином, але в цьому контексті вважається безпечним, оскільки не дозволяє виконання коду в браузері або доступ до будь-яких конфіденційних даних. Посилання "Back to home page" не містить параметрів запиту GET, що також є хорошою практикою.

```
User :
    <?php
        echo $_GET['user'];
    ?>

<a href="/home.php">Back to home page</a>
```

Рисунок 2.24 – Фрагмент коду сторінки profile.php

Останньою сторінкою є comments.php (див. рисунок 2.25). Цей файл не використовується для відображення даних, лише для збереження коментарів. Тобто після натиску кнопки збереження, форма перенаправляється на цю сторінку, після збереження даних користувача повертає на сторінку перегляду статті. Цей код на

перший погляд не містить жодних XSS вразливостей. Але, так як він використовується в контексті веб-сайту, то можливо виникнення XSS вразливості у зв’язку зі способом відображення коментарів. Так як, коментарі виводяться на сторінці, то слід переконатися, що всі введені дані очищаються та екрануються перед виведенням на сторінку.

```php
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $sql = "INSERT INTO comments (text, article, username) VALUES (:text, :article, :username)";
        $stmt = $conn->prepare($sql);
        $text = $_POST["text"];
        $article = $_POST["article"];
        $username = $_POST["username"];
        $stmt->bindValue(":text", $text);
        $stmt->bindValue(":article", $article);
        $stmt->bindValue(":username", $username);
        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
```

Рисунок 2.25 – Фрагмент коду сторінки comments.php

2.5 Результати тестування на XSS вразливості

В цьому розділі було проведено тестування тестового додатку на XSS вразливості. В ході тестування були застосовані Black-box і White-box методи тестування.

Під час Black-box тестування були протестовані усі сторінки додатку і було знайдено вразливості вказані в таблиці 2.1.

Таблиця 2.1

Результати ручного тестування додатку на можливі XSS загрози

Розташування вразливості	Тип вразливості	Опис вразливості
Сторінка перегляду статті	Клієнтська	За допомогою вводу шкідливого коду до параметр сторінки (article), цей код може запуститися на сторінці
Сторінка перегляду статті	Клієнтська	За допомогою вводу шкідливого коду до параметр сторінки (user), цей код може запуститися на сторінці
Сторінка регістрації	Серверна	При введенні даних нового користувача юзер може ввести текст зі шкідливим кодом
Сторінка профілю користувача	Серверна	При серверній атаці виводить дані і запускає шкідливий код, якщо дані його містять
Сторінка перегляду статті	Серверна	При серверній атаці виводить дані і запускає шкідливий код, якщо дані його містять
Сторінка перегляду статті	Серверна	Містить поле вводу коментаря, яке може зберегти текст зі шкідливим навантаженням до бази даних

Під час White-box тестування були протестовані усі сторінки додатку і було знайдено вразливості вказані в таблиці 2.2.

Таблиця 2.2

Результати аналізу коду на вразливості XSS

Розташування вразливості	Опис вразливостей
index.php	У поля введення можна вставити шкідливий код
home.php	сторінка використовує GET-запит для отримання імені користувача з URL
article.php	Параметри використовуються без екранування, дані виводяться без належної фільтрації
register.php	У поля введення можна вставити шкідливий код
comments.php	Дані зберігаються без належної фільтрації

Висновки за другим розділом

У цьому розділі було розглянуто основні методи виявлення XSS загроз, був розглянутий та проаналізований тестовий проект, в результаті чого були знайдені вразливості у вихідному коді. Були задіяні Black-box та White-box тестування та складені дві таблиці знайдених вразливостей. На результаті даних зібраних у цьому розділі буде розглянуто запобігання XSS вразливостям у наступному розділі.

РОЗДІЛ 3
ЗАПОБІГАННЯ МІЖСАЙТОВОМУ СКРИПТИНГУ У ХМАРНИХ СЕРЕДОВИЩАХ

3.1 Аналіз методів запобігання XSS загрозам

Щоб атаки XSS були успішними, зловмисник повинен вставити та запустити шкідливий вміст на веб-сторінці. Кожну змінну у веб-додатку потрібно захистити. Гарантія того, що всі змінні проходять валідацію, а потім відхиляються або дезінфікуються, відома як ідеальна стійкість до ін’єкції. Будь-яка змінна, яка не проходить цей процес, є потенційною слабкістю. Вихідне кодування та очищення HTML допомагають усунути ці прогалини [17].

Вихідне кодування рекомендовано, коли вам потрібно безпечно відобразити дані саме так, як їх ввів користувач. Змінні не слід інтерпретувати як код замість тексту. Почніть із використання захисту вихідного кодування за замовчуванням у вашій структурі, якщо ви бажаєте відображати дані, які вводить користувач. Функції автоматичного кодування та екранування вбудовані в більшість структур [18].

Існує багато різних методів кодування виводу, оскільки браузери по-різному аналізують HTML, JS, URL-адреси та CSS. Використання неправильного методу кодування може створити слабкі місця або зашкодити функціональності вашої програми.

Контекст HTML означає вставлення змінної між двома основними тегами HTML, наприклад або <b> (див. рисунок 3.1).

```
<div> $varUnsafe </div>
```

Рисунок 3.1 – Приклад вразливого HTML коду

Зловмисник може змінити дані, які відображаються як $varUnsafe. Це може призвести до додавання атаки до веб-сторінки. Щоб безпечно додати змінну до

контексту HTML, використовуйте кодування об’єкта HTML для цієї змінної, додаючи її до веб-шаблону. В табл. 3.1 є декілька прикладів закодованих значень для конкретних символів.

Таблиця 3.1

Приклади закодованих значень у HTML

Символ	Закодоване знаення
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&#x27;

Якщо ви використовуєте JavaScript для запису в HTML, подивіться на атрибут .textContent, оскільки він є безпечним приймачем і автоматично кодує сутність HTML

Контексти атрибутів HTML стосуються розміщення змінної у значенні атрибута HTML. Ви можете зробити це, щоб змінити гіперпосилання, приховати елемент, додати альтернативний текст до зображення або змінити вбудовані стилі CSS. Ви повинні застосувати кодування атрибутів HTML до змінних, які розміщуються в більшості атрибутів HTML (див. рисунок 3.2).

```
<div attr="$varUnsafe">
<div attr=”*x” onblur=”alert(1)*”> // Example Attack
```

Рисунок 3.2 – Приклад вразливого використання змінних у атрибутах

Дуже важливо використовувати лапки, як-от " або ', щоб оточувати свої змінні. Лапки ускладнюють зміну контексту, у якому працює змінна, що допомагає запобігти XSS. Лапки також значно зменшують набір символів, який потрібно кодувати, що робить вашу програму надійнішою і кодування легше реалізувати.

Якщо ви використовуєте JavaScript для запису в атрибут HTML, перегляньте методи .setAttribute і [attribute], які автоматично кодуватимуть атрибути HTML. Це

безпечні приймачі, якщо назва атрибута жорстко закодована та нешкідлива, як ідентифікатор або клас. Як правило, атрибути, які підтримують JavaScript, наприклад onClick - небезпечно використовувати з ненадійними значеннями атрибутів.

Контексти JavaScript стосуються розміщення змінних у вбудованому JavaScript, який потім вбудовується в документ HTML. Це зазвичай спостерігається в програмах, які активно використовують спеціальний JavaScript, вбудований у їхні веб-сторінки.

Єдине «безпечне» місце для розміщення змінних у JavaScript – це «значення даних у лапках» (див. рисунок 3.3). Усі інші контексти є небезпечними, і ви не повинні розміщувати в них змінні дані.

```
<script>alert('$varUnsafe')</script>
<script>x='$varUnsafe'</script>
<div onmouseover="'$varUnsafe'"</div>
```

Рисунок 3.3 – Приклади «значення даних у лапках»

«Контексти CSS» стосуються змінних, розміщених у вбудованому CSS. Це зазвичай, коли ви хочете, щоб користувачі могли налаштувати зовнішній вигляд своїх веб-сторінок. CSS напрочуд потужний і використовувався для багатьох типів атак. Змінні слід розміщувати лише у значенні властивості CSS. Інші «контексти CSS» небезпечні, і ви не повинні розміщувати в них змінні дані (див. рисунок 3.4).

```
<style> selector { property : $varUnsafe; } </style>
<style> selector { property : "$varUnsafe"; } </style>
<span style="property : $varUnsafe">Oh no</span>
```

Рисунок 3.4 – Приклад розташування змінних у CSS коді

Якщо ви використовуєте JavaScript для зміни властивості CSS, спробуйте використовувати style.property = x. Це безпечний прийом даних, і дані в ньому автоматично кодуються CSS.

«Контексти URL-адреси» стосуються змінних, розміщених у URL-адресі. Найчастіше розробник додає параметр або фрагмент URL-адреси до бази URL-адрес, яка потім відображається або використовується в певній операції. Використовуйте

URL-кодування для цих сценаріїв. Приклад вразливого посилання показаний на рисунку 3.5.

Кодуйте всі символи у форматі кодування %HH. Переконайтеся, що всі атрибути взято в лапки, так само як в JS і CSS.

```
<a href="http://www.owasp.org?test=$varUnsafe">link</a >
```

Рисунок 3.5 – Приклад вразливого посилання у HTML коді

Буватимуть ситуації, коли ви використовуєте URL-адресу в різних контекстах. Найпоширенішим є додавання його до атрибута href або src тегу <a>. У цих сценаріях слід виконати кодування URL-адреси, а потім кодування атрибутів HTML (див рисунок 3.6).

```
url = "https://site.com?data=" + urlencode(parameter)
<a href='attributeEncode(url)'>link</a>
```

Рисунок 3.6 – Приклад кодування атрибутів посилання HTML

Якщо ви використовуєте JavaScript для створення значення запиту URL-адреси, подивіться на використання window.encodeURIComponent(x). Це безпечний приймач, і дані в ньому автоматично кодуватимуть URL-адресу. Приклад небезпечних контекстів вказано у табл. 3.2.

Таблиця 3.2

Небезпечні контексти

Небезпечні контексти
<script>Directly in a script</script>
<!-- Inside an HTML comment -->
<style>Directly in CSS</style>

```
<ToDefineATag href="/test" />
```

Вихідне кодування не ідеальне. Це не завжди запобігатиме XSS. Ці місця відомі як небезпечні контексти. Небезпечні контексти показані в табл. 3.2. Не розміщуйте змінні в небезпечних контекстах, оскільки навіть кодування виводу не запобіжить атаці XSS повністю.

Інші області, на які слід звернути увагу, включають:

- функції зворотного виклику;
- де URL-адреси обробляються в такому коді, як цей CSS { background-url : “javascript:alert(xss)”; }
- усі обробники подій JavaScript (onclick(), onerror(), onmouseover()).
- небезпечні функції JS, такі як eval(), setInterval(), setTimeout()

Іноді користувачам потрібно створити HTML. Одним зі сценаріїв було б дозволити користувачам змінювати стиль або структуру вмісту в редакторі WYSIWYG. Кодування виводу тут запобігає XSS, але порушує заплановану функціональність програми. Стиль не буде відображено. У цих випадках слід використовувати обробку HTML.

Очищення HTML видаляє небезпечний HTML зі змінної та повертає безпечний рядок HTML. OWASP рекомендує DOMPurify для очищення HTML (див. рисунок 3.6).

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

Рисунок 3.6 – Приклад очищення коду

Є ще деякі речі, які слід враховувати:

- Якщо ви дезінфікуєте вміст, а потім змінюєте його, ви можете легко скасувати свої зусилля щодо безпеки.
- Якщо ви дезінфікуєте вміст, а потім надсилаєте його до бібліотеки для використання, переконайтеся, що він якось не змінює цей рядок. В іншому випадку, знову ж таки, ваші зусилля безпеки будуть недійсними.

- Ви повинні регулярно виправляти DOMPurify або інші бібліотеки HTML Sanitization, які ви використовуєте. Браузери змінюють функціональні можливості, і регулярно виявляються обхідні шляхи

Фахівці з безпеки часто говорять про джерела та поглиначі. Якщо ви забрудните річку, вона потече кудись вниз за течією. Те ж саме з комп'ютерною безпекою. Приймачі XSS – це місця, де змінні розміщуються на вашій веб-сторінці.

На щастя, багато місць, де можна розмістити змінні, є безпечними (див. рисунок 3.7). Це тому, що ці приймачі обробляють змінну як текст і ніколи її не виконають. Спробуйте змінити свій код, щоб видалити посилання на небезпечні приймачі, такі як innerHTML, і замість цього використовуйте textContent або значення.

```
elem.textContent = dangerVariable;
elem.insertAdjacentText(dangerVariable);
elem.className = dangerVariable;
elem.setAttribute(safeName, dangerVariable);
formfield.value = dangerVariable;
document.createTextNode(dangerVariable);
document.createElement(dangerVariable);
elem.innerHTML = DOMPurify.sanitize(dangerVar);
```

Рисунок 3.7 – Приклад безпечного JS коду

Безпечні атрибути HTML включають: align, alink, alt, bgcolor, border, cellpadding, cellspacing, class, color, cols, colspan, coords, dir, face, height, hspace, ismap, lang, marginheight, marginwidth, multiple, nohref, noresize, noshade, nowrap, ref, rel, rev, rows, rowspan, scrolling, shape, span, summary, tabindex, title, usemap, valign, value, vlink, vspace, width.

Захист системи безпеки, кодування виводу та очищення HTML забезпечать найкращий захист вашої програми. OWASP рекомендує їх за будь-яких обставин. Подумайте про застосування наступних засобів керування на додаток до наведених вище.

- Атрибути файлів cookie – вони змінюють спосіб взаємодії JavaScript і браузерів із файлами cookie. Атрибути файлів cookie намагаються обмежити вплив

атаки XSS, але не запобігають виконанню шкідливого вмісту та не усувають основну причину вразливості.

- Політика безпеки вмісту – білий список, який запобігає завантаженню вмісту. З впровадженням легко зробити помилки, тому це не має бути вашим основним захисним механізмом. Використовуйте CSP як додатковий рівень захисту та подивіться на шпаргалку тут.
- Брандмауери веб-програм – вони шукають відомі рядки атак і блокують їх. WAF ненадійні, і нові методи обходу відкриваються регулярно. WAF також не усуває першопричину вразливості XSS. Крім того, WAF також пропускають клас уразливостей XSS, які працюють виключно на стороні клієнта. WAF не рекомендуються для запобігання XSS, особливо XSS на основі DOM.

3.2 Політика безпеки вмісту або CSP

Одним з ефективних інструментів захисту є застосування безпечної політики вмісту (Content Security Policy, CSP), яка дозволяє підвищити безпеку сайту та захистити його від різноманітних атак, таких як Cross-Site Scripting (XSS) та інших.

CSP – це механізм безпеки, який забезпечує контроль над тим, який контент може бути відображений на веб-сайті та дозволяє відокремити код веб-сторінок від даних користувача. Це дозволяє зменшити ризик XSS-атак, які зазвичай використовуються для крадіжки даних користувачів або поширення шкідливих програм. CSP також дозволяє захистити веб-сайт від атак на внесення даних, коли зловмисник вводить шкідливий код на веб-сторінці, щоб отримати доступ до інформації користувача [19].

Щоб увімкнути CSP на веб-сайті, потрібно налаштувати веб-сервер таким чином, щоб він повертав заголовок Content-Security-Policy HTTP. Цей заголовок містить правила безпеки, які браузер повинен застосовувати до контенту на веб-сторінках. Наприклад, за допомогою CSP можна заборонити виконання JavaScript зовнішніх джерел, що знижує ризик XSS-атак. Приклад використання CSP можна побачити на рисунку 3.8.

```
Content-Security-Policy: default-src 'self'; img-src *;
media-src example.org example.net; script-src userscripts.example.com
```

Рисунок 3.8 – Приклад використання CSP на сервері

CSP також дозволяє контролювати використання ресурсів на веб-сторінці, таких як зображення, стилі або скрипти. Наприклад, за допомогою CSP можна дозволити використання зображень лише з домену веб-сайту, що знижує ризик атак на внесення даних. Крім того, CSP дозволяє застосовувати політику для відстеження та повідомлення про помилки. Якщо браузер не може виконати правила CSP, то він відправляє повідомлення на сервер про те, які правила були порушені. Це дозволяє адміністраторам веб-сайту знати, які атаки намагалися здійснити на сайті та вчасно вживати заходів для їх запобігання.

Оскільки CSP є механізмом безпеки, він має деякі обмеження. Наприклад, він не може запобігти всім типам атак, а також може призвести до незручностей у використанні деяких функцій веб-сайту. Крім того, якщо ви не налаштуєте CSP належним чином, то це може спричинити проблеми з функціональністю веб-сайту та знизити його продуктивність [19].

У цілому, CSP є потужним засобом для захисту веб-сайту від атак, які можуть викликати серйозні проблеми з безпекою користувачів та веб-сайту. Щоб включити CSP на веб-сайті, вам потрібно налаштувати веб-сервер таким чином, щоб він повертав заголовок Content-Security-Policy HTTP. Цей заголовок містить правила безпеки, які браузер повинен застосовувати до контенту на веб-сторінках. Налаштування CSP може забрати певний час, але воно забезпечує підвищену безпеку веб-сайту та знижує ризик вразливостей, що можуть бути використані зловмисниками.

3.3 Запобігання XSS вразливостям у коді

У минулому розділі було розглянуто тестування додатку на XSS вразливості. При тестуванні були знайдені декілька вразливостей вручну, а також було розглянуто код на потенційні місця, де могли би бути застосовані міжсайтові сценарії. Прийшов час виправити код для запобігання таким загрозам. Повний виправлений код можна побачити у Додатку Б.

Розглянемо файл index.php. Код містить вразливості XSS, тому що введені користувачем дані виводяться на сторінці без екранування. Щоб виправити ці вразливості, можна використовувати HTML-екранування та валідацію даних. Так як ця сторінка містить у собі форму, яка оброблюється у іншому файлі, то і валідацію, звісно, краще робити на стороні серверу. Але і на стороні клієнту ми можемо додати певну валідацію. Наприклад, ми можемо додати атрибут ‘required’, для того щоб користувач не міг ввести пусту строку до поля введення, також можемо встановити максимальну та мінімальну довжину тексу, а також встановити паттерн символів, щоб користувач міг ввести тільки дозволені символи. Приклад, коду після змін можна побачити на рисунку 3.9.

```
<h2>Login form</h2>
<form action="home.php" method="POST">
<p>Enter login: <input type="text" name="login" pattern="[A-Za-z0-9]+" min="4" max="16"/></p>
<p>Enter password: <input type="password" name="password" pattern="[A-Za-z0-9]+" min="8" max="16"/></p>
<input type="submit" value="Send">
</form>
<a href="register.php">
    Sign up
</a>
```

Рисунок 3.9 – Змінений код сторінки index.php

Далі розглянемо код файлу home.php. Цей код також містить вразливості XSS через те, що дані, які були передані через GET-запит, використовуються без екранування в HTML-виводі. Щоб виправити цей код потрібно зробити:

1. Додати функцію htmlspecialchars() до всіх виводів HTML-коду. Ця функція конвертує спеціальні символи HTML у відповідні HTML-сутності, які не

можуть бути виконані браузером як код. Це захистить від XSS-атак, що можуть спровокувати виконання небезпечного JavaScript коду на стороні клієнта.

2. Додати використання функції htmlspecialchars() для введення username. Таким чином, ми можемо захистити від XSS-атак при введенні імені користувача через GET-запит.

3. Застосувати функцію htmlspecialchars() до всіх інших даних, що виводяться на сторінку, таких як назва статті, текст статті та посилання на статтю.

Після усіх виправлень код буде виглядати як на рисунку 3.10.

```
<h2>Home</h2>

<hr>

<?php

$username = htmlspecialchars($_GET['user']);

$conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");

$sql = "SELECT * FROM article";

$result = $conn->query($sql);

foreach($result as $row) {
    $articleName = htmlspecialchars($row["name"]);
    $articleText = htmlspecialchars($row["text"]);
    $articleLink = htmlspecialchars($row["link"]);

    echo "<h4>".$articleName."</h4>";
    echo "<p>".$articleText."</p>";
    echo "<a href=article.php?article=".$articleLink."&user=".$username."> Read more </a>";
    echo "<hr />";
}

?>
```

Рисунок 3.10 – Код файлу home.php після виправлень

Тепер виправимо код файлу article.php. Щоб виправити потенційні XSS-атаки в даному коді, потрібно здійснити наступні кроки:

1. Замінити всі виклики змінної $_GET на функцію htmlentities з параметром ENT_QUOTES для екранування спеціальних символів HTML-коду відображуваних даних (див рисунок 3.11).

2. Замінити всі виклики змінної $_POST на функцію htmlspecialchars з параметром ENT_QUOTES для екранування спеціальних символів HTML-коду відправлюваних даних (див. рисунок 3.12).

3. Використовувати валідацію даних на стороні сервера, щоб перевірити коректність відправлюваних даних.

```php
if($stmt->rowCount() > 0)
{
    foreach($stmt as $row) {
        $articleName = $row["name"];
        $articleText = $row["fullarticle"];
        $articleLink = $row["link"];

        echo "<h2>".htmlentities($articleName, ENT_QUOTES, 'UTF-8')."</h2>";
        echo "<p>".htmlentities($articleText, ENT_QUOTES, 'UTF-8')."</p>";
    }
}
```

Рисунок 3.11 – Приклад використання htmlentities в файлі article.php

```php
<h4>Comments</h4>

<form action="comment.php" method="POST">
    <p>User: <?php echo htmlentities($_GET['user'], ENT_QUOTES, 'UTF-8'); ?></p>
    <input type="text" value="<?php echo htmlspecialchars($_GET['user'], ENT_QUOTES, 'UTF-8'); ?>" s
    <input type="text" value="<?php echo htmlspecialchars($_GET['article'], ENT_QUOTES, 'UTF-8'); ?>
</form>

<hr>
```

Рисунок 3.12 – Приклад використання htmlspecialchars в файлі article.php

Далі розглянемо файл comment.php. Цей файл не виводить нічого на сторінку, але він працює у контексті додатку та створює рекорди у базі даних і містить певні вразливості. Виправлення полягає у видаленні HTML-тегів з тексту коментаря за допомогою функції strip_tags() та екрануванні спеціальних символів HTML-коду в тексті коментаря за допомогою функції htmlspecialchars(). Ці дії запобігають

виконанню зловмисних скриптів на стороні клієнта, коли коментар відображається на веб-сторінці. Виправлений код можна побачити на рисунку 3.13.

Тепер після виправлення коду, ми можемо бути впевненими, що коментар не зможе містити у собі шкідливий код, а разом з виправленнями у попередніх файлах шкідливий код не зможе ще й відображатися. Таким чином ми зможемо повністю нівелювати XSS вразливості зв'язані з коментарями.

```php
<?php

    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");

        $text = $_POST["text"];
        $article = $_POST["article"];
        $username = $_POST["username"];

        // Видалення HTML-тегів з тексту коментаря
        $text = strip_tags($text);

        // Екранування спеціальних символів HTML-коду в тексті коментаря
        $text = htmlspecialchars($text, ENT_QUOTES | ENT_HTML5, 'UTF-8');

        // Вставка коментаря до бази даних
        $sql = "INSERT INTO comments (text, article, username) VALUES (:text, :article, :username)";
        $stmt = $conn->prepare($sql);

        $stmt->bindValue(":text", $text);
        $stmt->bindValue(":article", $article);
        $stmt->bindValue(":username", $username);

        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
```

Рисунок 3.13 – Виправлений код файлу comment.php

Ще розглянемо сторінку register.php. Ця сторінка містить HTML-форму для реєстрації користувача. Форма містить три поля введення: "login", "password" та "verifypassword", а також кнопку відправки форми на сервер. При відправлені форми, дані будуть відправлені на сторінку "login.php" методом POST. На сторінці "login.php" будуть оброблені дані, що були введені в форму.

Проте, в коді відсутні будь-які перевірки на правильність введених даних, тому ця форма є вразливою до атак типу SQL Injection та Cross-Site Scripting (XSS). Для запобігання цим атакам, слід додати перевірки на правильність введення даних,

наприклад, використовувати функції валідації введених даних та захищати введені дані перед тим, як вони будуть відправлені на сервер. Виправлення будуть такими же як і в файлі index.php.

Щодо файлу login.php (див. рисунок 3.14), то він теж містить потенційні вразливості, оскільки не фільтрує вхідні дані з форми перед вставкою їх в базу даних. Це може призвести до вставки шкідливого коду в базу даних та подальшого відображення його на сторінці.

Щоб усунути XSS вразливості у даному коді, можна виконати наступні дії:

1. Використати функцію htmlspecialchars() для екранування спеціальних символів HTML, які можуть використовуватися для XSS атак.

2. Використати функцію strip_tags() для видалення тегів HTML та PHP з вхідних даних. Це можна зробити таким чином:

Обидва вищезгадані методи можуть бути використані окремо або разом, щоб забезпечити максимальний захист від XSS вразливостей. Змінений код можна побачити на рисунку 3.14.

```php
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");

        $sql = "INSERT INTO Users (username, password) VALUES (:username, :password)";
        $stmt = $conn->prepare($sql);

        $login = htmlspecialchars(strip_tags($_POST["login"]), ENT_QUOTES, 'UTF-8');
        $password = htmlspecialchars(strip_tags($_POST["password"]), ENT_QUOTES, 'UTF-8');

        $stmt->bindValue(":username", $login);
        $stmt->bindValue(":password", $password);

        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
```

Рисунок 3.14 – Змінений код файлу login.php

3.4 Налаштування безпечного хмарного середовища

Amazon Web Services або AWS є дочірньою компанією Amazon.com, що надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки. Існує і безкоштовна підписка, яка доступна протягом перших 12 місяців. Технологія дозволяє абонентам мати у своєму розпорядженні повноцінний віртуальний кластер комп'ютерів, який завжди доступний через Інтернет. Віртуальні комп'ютери AWS мають більшість атрибутів реального комп'ютера, включаючи апаратні пристрої (процесор, відеокарту, локальну та оперативну пам'ять, жорсткий диск або SSD-накопичувач); операційну систему на вибір; мережу; і попередньо встановлені прикладні програми, такі як вебсервер, база даних, CRM і т. д. Кожна система AWS також віртуалізує консольний ввід/вивід (клавіатура, дисплей і миша), що дозволяє користувачам AWS підключитися до своєї системи AWS за допомогою браузера. Браузер виступає як вікно у віртуальний комп'ютер, дозволяючи користувачу входити в систему, налаштовувати та використовувати свої віртуальні системи так само, як справжній, фізичний комп'ютер. Це дозволяє їм налаштувати систему так, щоб надавати інтернет-орієнтовані сервіси та послуги своїм клієнтам [20].

Для використання додатку будуть використані сервіси Амазону, а доступ до коду буде надаватися через GitHub. Вибір впав саме на цю технологію із-за декількох причин:

1. у AWS хмарні користувачі розподілені, тобто не впливають один на одного;
2. AWS за замовчуванням використовує HTTPS протокол;
3. AWS містить багато вбудованих функцій для захисту середовища.

Саме останній пункт найбільш важливий, бо для захисту додатку буде потрібно налаштувати CSP.

Amazon Web Services (AWS) надає можливість використання CSP за допомогою AWS Web Application Firewall (WAF). WAF дозволяє налаштовувати правила безпеки на рівні програм, наприклад, на основі CSP. За допомогою AWS WAF можна

встановити правила фільтрації вхідного трафіку на основі заголовків HTTP, такі як заголовки Content-Security-Policy або X-XSS-Protection.

Для використання CSP у хмарному сховищі AWS S3 необхідно створити нову поведінку CloudFront з налаштуванням роботи CSP та додати її до домену CloudFront. Приклад налаштування файлу політики CSP знаходиться на рис. 3.15.

```
default-src 'self' *.example.com;
script-src 'self' 'unsafe-inline' 'unsafe-eval' *.example.com;
```

Рисунок 3.15 – Приклад налаштування CSP

Для налаштування CSP через CloudFront потрібно зробити наступні дії:

1. Відкрийте консоль керування AWS та виберіть Amazon CloudFront.
2. Натисніть кнопку "Створити розподіл".
3. Виберіть тип розподілу "Веб-сайт", а потім натисніть "Get Started" (Почати).
4. Введіть адресу джерела (Origin Domain Name) та налаштування кешування, потім натисніть "Create Distribution" (Створити розподіл).
5. Після того, як розподіл буде створено, виберіть його зі списку розподілів і перейдіть на вкладку "Behaviors" (Поведінка).
6. Натисніть кнопку "Create Behavior" (Створити поведінку).
7. Вкажіть параметри поведінки, включаючи політику CSP. У настройках поведінки CSP можна вказати в заголовку "Content-Security-Policy".
8. Введіть відповідні директиви політики.
9. Збережіть параметри поведінки, а потім збережіть зміни розподілу.

Приклад вікна налаштування CSP можна побачити на рисунку 3.16.

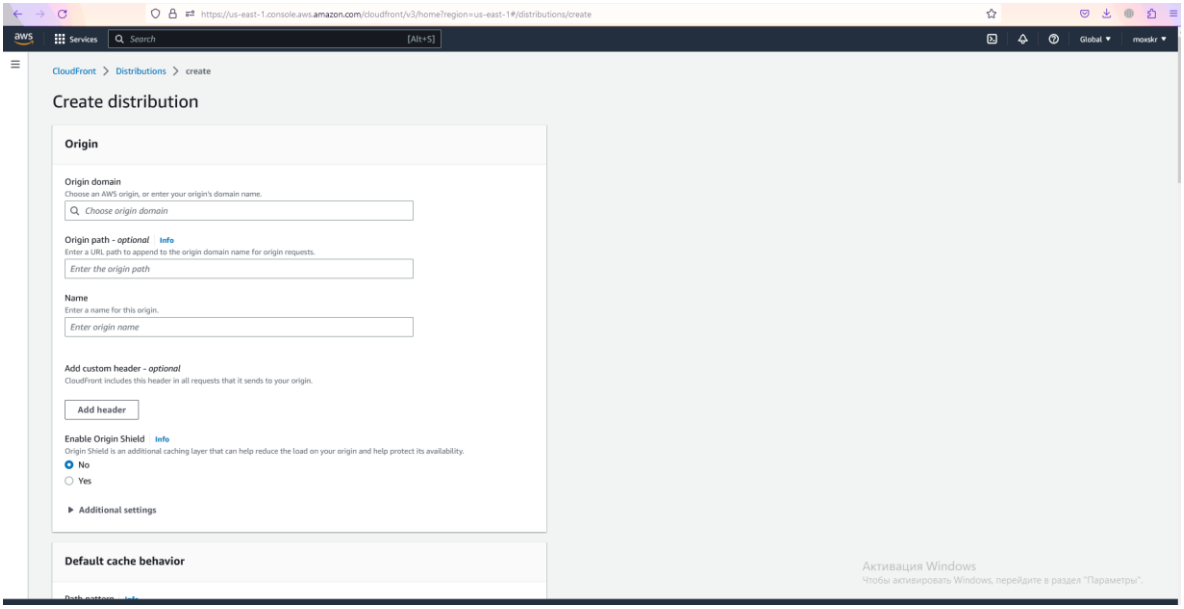

Рисунок 3.16 – Приклад сторінки налаштування CloudFont

Тепер після налаштування CSP слід задеплоїти код до репозиторію GitHub, після чого через AWS консоль ми можемо запустити додаток у хмарі (див. рисунок 3.17).

Рисунок 3.17 – Додаток запущений у хмарі

3.5 Тестування додатку після запобігання XSS загрозам

Після того як було виправлено код, застосовано політику CSP, а також додаток був розміщений у хмарному середовищі, потрібно протестувати заново додаток, щоб впевнитися, що він є захищеним від XSS загроз.

Для перевірки того чи правильно було застосовано політику CSP досить просто переглянути відповіді HTTP запитів, які надходять до браузеру при відкритті додатку. Для цього можемо використати Chrome DevTools утиліту яка вбудована в браузер Google Chrome. Як можна побачити на рисунку 3.18, до HTTP відповіді додався заголовок «Content-Security-Policy» зі значеннями вказаними у конфігурації середовища.

```
HTTP/1.1 200 OK
Date: Tue, 07 Mar 2023 09:53:15 GMT
Server: Apache/2.4.51 (Win64) PHP/7.4.26
X-Powered-By: PHP/7.4.26
Content-Length: 517
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Security-Policy: default-src 'self'; img-src https://*; child-src 'none';
```

Рисунок 3.18 – HTTP відповідь з CSP

Далі розглянемо ті вразливості, які були вказані раніше. Для цього в усіх місцях де були знайдені XSS загрози зробимо теж тестування, яке було зроблено у попередньому розділі. Для початку розглянемо клієнтські вразливості.

Як видно з рис. 3.19 і 3.20 при тестуванні на клієнтський XSS шкідливий код не відпрацював. Отже, можна зробити висновок, що додаток тепер є стійким до цього виду загроз. Далі розглянемо серверний XSS.

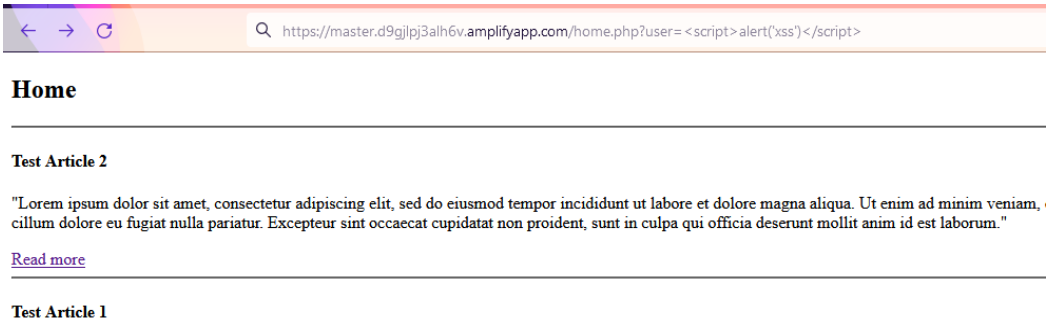

Рисунок 3.19 – Тестування сторінки home.php на відобрежений XSS

Рисунок 3.20 – Тестування сторінки article.php на клієнтський XSS

Як видно на рис. 3.21, на сторінці створення користувача навіть не можна вставити шкідливий код із-за фільтрації символів, тому можна вважати цю сторінку стійкою до загроз. Перейдемо до сторінки article.php. Проблема була у створенні коментарів. Спробуємо ввести шкідливий код та зберегти коментар (див. рисунок 3.22).

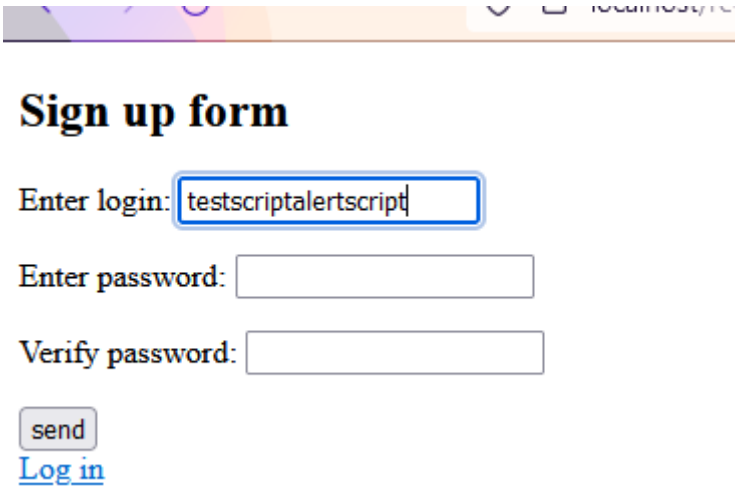

Рисунок 3.21 – Тестування сторінки register.php на серверний XSS

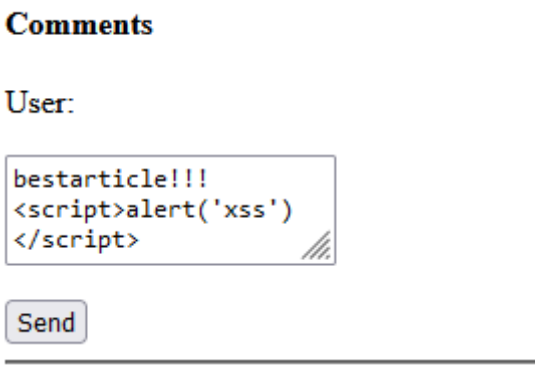

Рисунок 3.22 – Створення шкідливого коментарю

Після збереження коментарю сторінка оновилася, але ніякий шкідливий код не відпрацював. Як видно з рис. 3.23, шкідливий код не був збережений. Отже, можна вважати, що запобігання серверним XSS загрозам спрацювало у даному додатку.

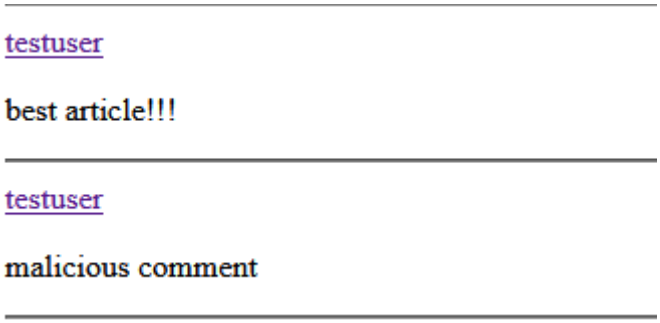

Рисунок 3.23 – Відображення коментарів на сторінці article.php

Висновки за третім розділом

У цьому розділі було розглянуто запобігання XSS вразливостям у хмарних додатках. Зокрема, були виправлені місця потенційних вразливостей в коді, було застосовано політику CSP за допомогою вбудованих інструментів хмарного середовища AWS. Після чого вихідний код застосунку був розміщений у хмарному середовищі.

В результаті тестування додатку в хмарному середовищі, було виявлено, що додаток є стійким до загроз знайдених у попередньому розділі, а також була успішно застосована політика CSP для хмарного застосунку.

РОЗДІЛ 4
ЗАГАЛЬНІ РЕКОМЕНДАЦІЇ ЩОДО ВИЯВЛЕННЯ ТА ЗАПОБІГАННЯ МІЖСАЙТОВОМУ СКРИПТИНГУ У ХМАРНИХ СИСТЕМАХ

4.1 Рекомендації щодо виявлення XSS вразливостей

Після аналізу у попередніх розділах можна визначити що найважливіший підхід до виявлення XSS полягає у використанні статичного аналізу для сканування вихідного коду веб-додатку. Інструменти статичного аналізу можуть ідентифікувати потенційні вразливості в коді, включно з областями, чутливими до атак XSS. Це можна зробити вручну розробниками або за допомогою автоматизованих інструментів, які сканують код на потенційні вразливості.

Ще дуже важливим є тестування на вразливості вручну, яке було продемонстроване в роботі. Таким чином навіть не маючи доступу до коду можна знайти вразливі місця у хмарному додатку.

Інструменти динамічного аналізу також можна використовувати для виявлення атак XSS у хмарних системах. Динамічний аналіз передбачає аналіз поведінки веб-програми під час її роботи та пошук моделей або поведінки, які вказують на атаку. Це можна зробити за допомогою таких інструментів, як сканери веб-програм, які імітують атаки на веб-програму для виявлення потенційних уразливостей.

Іншим важливим аспектом виявлення XSS-атак у хмарних системах є моніторинг і логіювання. Відстежуючи підозрілу активність у системі та реєструючи всю відповідну інформацію, адміністратори можуть швидко виявляти потенційні атаки та реагувати на них. Це можна зробити за допомогою систем управління інформацією про безпеку та подіями (SIEM), які збирають і аналізують дані з різних джерел для виявлення інцидентів безпеки.

Коли виявлено атаку XSS, важливо швидко та належним чином відреагувати, щоб зменшити шкоду. Це може включати блокування зловмисного запиту, сповіщення системних адміністраторів або користувачів або вжиття інших заходів

для виправлення, якщо це необхідно. Також важливо дослідити атаку, щоб виявити першопричину та вжити заходів для запобігання повторенню подібних атак у майбутньому.

Одним із поширених підходів є використання брандмауера веб-додатків (WAF), який може виявляти та блокувати зловмисні запити. WAF можуть аналізувати вхідні запити на наявність підозрілих шаблонів або корисних даних і блокувати запити, ідентифіковані як потенційні атаки XSS. Це можна зробити за допомогою виявлення на основі сигнатур, яке шукає в запиті певні шаблони, які, як відомо, пов’язані з атаками XSS, або за допомогою виявлення на основі поведінки, яке аналізує поведінку запиту для виявлення потенційних атак. Більшість хмарних провайдерів надають вбудовані інструменти WAF, один з них був використаний у кваліфікаційної роботі.

Окрім виявлення та пом’якшення атак XSS, також важливо бути в курсі останніх загроз безпеці та вразливостей. Це включає в себе отримання інформації про нові методи атак і вразливості, а також регулярне оновлення програмного забезпечення та застосування патчів безпеки. Це також може включати проведення регулярних перевірок безпеки та тестування на проникнення для виявлення та усунення будь-яких потенційних недоліків у системі.

Підсумовуючи, виявлення атак XSS у хмарних системах є важливою частиною підтримки безпеки та цілісності системи. Це вимагає поєднання методів виявлення, найкращих практик, моніторингу та логіювання, а також швидкого реагування на інциденти. Застосовуючи проактивний підхід до безпеки та зберігаючи пильність щодо нових загроз, адміністратори та розробники хмарних систем можуть забезпечити безпеку та захист своїх систем від зловмисних атак.

4.2 Рекомендації щодо запобігання XSS вразливостей

В результаті дослідження в третьому розділі можна виділити такі рекомендації щодо запобігання міжсайтовим сценаріям у хмарних системах:

1. Перевірка введених даних: одним із найефективніших способів запобігання XSS-атакам є перевірка введених даних користувачами. Це означає перевірку всіх введених даних на наявність шкідливого вмісту або сценаріїв і відхилення їх у разі виявлення. Перевірка введених користувачем даних може здійснюватися на рівні сервера, клієнта або на обох. Переконавшись, що приймається лише дійсний вхід, система може уникнути виконання шкідливого коду, введеного зловмисниками.

2. Вихідне кодування. Іншим важливим заходом для запобігання атакам XSS є вихідне кодування. Це передбачає перетворення спеціальних символів у вихідних даних на їхні еквівалентні символи HTML або URL-кодування. Вихідне кодування може виконуватися на рівні сервера, клієнта або на обох. За допомогою кодування вихідних даних система може гарантувати, що будь-який вхід від користувачів, який може містити шкідливий код, не буде інтерпретований браузером як такий.

3. Політика безпеки вмісту (CSP): CSP – це функція безпеки, яка дозволяє системі визначати джерела вмісту, які може завантажувати веб-програма. Це може включати лише дозвіл вмісту з надійних джерел, блокування вбудованих сценаріїв і вимкнення використання небезпечних динамічних функцій, таких як eval(). Впроваджуючи CSP, хмарні системи можуть обмежити здатність зловмисників впроваджувати шкідливі сценарії у веб-додаток.

4. HTTPS: важливо використовувати HTTPS для шифрування всього зв’язку між браузером користувача та веб-програмою. Це може допомогти запобігти зловмисникам від перехоплення та зміни мережевого трафіку. HTTPS можна реалізувати за допомогою шифрування SSL або TLS, і для забезпечення найвищого рівня безпеки рекомендується використовувати останню версію протоколу.

5. Навчання користувачів: навчання користувачів ризикам XSS-атак і тому, як їх розпізнавати та уникати, може бути ефективним способом запобігання цим типам атак. Це може включати надання користувачам інформації про те, як ідентифікувати підозрілі посилання, уникати натискання посилань із невідомих джерел і бути обережним під час введення особистої інформації у веб-форми.

6. Оновлення безпеки: регулярне оновлення програмного забезпечення та застосування оновлень безпеки може допомогти запобігти атакам XSS шляхом усунення відомих вразливостей. Це включає оновлення самої веб-програми, а також будь-яких сторонніх бібліотек або плагінів, які використовує програма. Підтримка системи в актуальному стані з останніми виправленнями безпеки і є важливим аспектом підтримки її загальної безпеки.

Підсумовуючи, запобігання XSS-атакам у хмарних системах потребує комплексного підходу, який включає перевірку вхідних даних, вихідне кодування, CSP, HTTPS, навчання користувачів і оновлення безпеки. Впроваджуючи ці найкращі практики, адміністратори та розробники хмарних систем можуть допомогти захистити свої системи від зловмисних атак і забезпечити безпеку та цілісність даних своїх користувачів.

Висновки за четвертим розділом

У цьому розділі було розроблено рекомендації щодо виявлення та запобігання XSS загроз у хмарних системах на базі досліджень, які були проведені у попередніх розділах. Таким чином для виявлення міжсайтового скриптингу у хмарних системах потрібно використовувати ручне тестування додатків, аналіз коду, автоматизований аналіз додатків та брандмауери. Для запобігання таким загрозам потрібно використовувати перевірку вихідних даних, вихідне кодування, CSP, HTTPS, постійно оновлювати безпеку та навчати користувачів.

ВИСНОВКИ

Міжсайтовий скриптинг є дуже серйозною загрозою для сучасних веб-систем і особливо для хмарних веб-систем. Збільшення популярності хмарних систем збільшує не тільки кількість самих веб-додатків у хмарі, а й складніть цих систем і їх взаємодії. Таким чином традиційні засоби захисту потребують удосконалення для використання в хмарних додатках.

Захист від XSS-атак у хмарних системах є важливою складовою безпеки, яка повинна бути включена в план безпеки всіх хмарних сервісів. Враховуючи складність хмарних систем та розподілену структуру, важливо підходити до захисту від XSS-атак з високим рівнем уваги та серйозності.

В першому розділі було проаналізовано доступну літературу пов’язану з захистом від міжсайтового скриптингу та запобіганню загрозам у хмарних системах. На основі проведеного аналізу було визначено класифікацію та особливості захисту хмарних додатків від XSS, що і було викладено у першому розділі роботи.

В другому розділі були більш детально розглянуті методи виявлення міжсайтового скриптингу. Після аналізу методів виявлення було проведено тестування тестового додатку на знаходження можливих XSS загроз. Під час тестування були проведено як ручне тестування додатку, так і огляд вихідного коду додатку. Після проведення тестування були знайдені вразливості та потенційні місця вразливостей, з яких була складена таблиця результатів тестування.

В третьому розділі було проаналізовано методи запобігання XSS загрозам у хмарних додатках. Після розгляду методів, вони були задіяні у виправленні коду тестового додатку для запобігання загрозам виявленим у другому розділі. Ще була розглянута політика безпеки вмісту (CSP), був проведений її детальний аналіз, після чого на базі хмарного провайдера Amazon Web Services було сконфігуровано політику CSP для тестового додатку. Після всіх дій щодо запобігання XSS загрозам у тестовому хмарному додатку, було проведено ще одне тестування на вразливості, яке показало, що заходи з запобігання вразливостям були успішні.

На базі дослідження у перших трьох розділах, у четвертому розділі було викладено загальні рекомендації щодо захисту хмарних систем від міжсайтового скриптингу. Таким чином, було досягнуто головну мету цієї роботи, а саме аналіз методів захисту від XSS і створення рекомендацій, які у подальшому можна буде використати у проектуванні хмарних систем.

Поставлені завдання були виконані у повному обсязі. Застосування створених рекомендації щодо захисту хмарних веб-систем від міжсайтового скриптингу добре себе показали високі на тестовому проекті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OWASP Foundation. Cross Site Scripting (XSS) [Електронний ресурс]. – Режим доступу до ресурсу: https://owasp.org/www-community/attacks/xss/

2. R M Wibowo, Sulaksono. Web Vulnerability Through Cross Site Scripting (XSS) Detection with OWASP Security Shepherd // Indonesian Journal of Information Systems (IJIS) Vol. 3, No. 2. – 2021.

3. Zbigniew Banach. What is a cross-site scripting vulnerability? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.invicti.com/blog/web-security/cross-site-scripting-xss/

4. Jalen Mack, Yen-Hung (Frank) Hu, and Mary Ann Hoppa. A Study of Existing Cross-Site Scripting Detection and Prevention Techniques Using XAMPP and VirtualBox // Virginia Journal of Science Volume 70, Issue 3, doi: 10.25778/bx6k-2285. – 2019.

5. Jean Rosemond Dora, Karol Nemoga. Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity // Cybersecur. Priv. 2021, 1, 319 – 339 https://doi.org/10.3390/jcp1020018. - 2021.

6. Crowdstrike.com. What is Cross Site Scripting (XSS)? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.crowdstrike.com/cybersecurity-101/cross-site-scripting-xss/

7. PortSwigger. DOM-based XSS [Електронний ресурс]. – Режим доступу до ресурсу: https://portswigger.net/web-security/cross-site-scripting/dom-based

8. OWASP Foundation. Types of XSS [Електронний ресурс]. – Режим доступу до ресурсу: https://owasp.org/www-community/Types_of_Cross-Site_Scripting

9. Cisco. XSS Threat Cloud Enviroments [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cisco.com/c/en/us/products/security/xss-threat-cloud-computing-environments.html

10. Rakesh Kumar, Rinkaj Goyal. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey // Comput. Sci. Rev. doi:10.1016/J.COSREV.2019.05.002 – 2019.

11. Describing XSS: The story hidden in time [Електронний ресурс]. – Режим доступу до ресурсу: https://medium.com/@ryoberfelder/describing-xss-the-story-hidden-in-time-80c3600ffe81

12. Amakiri Welekwe. How to Find XSS Vulnerability [Електронний ресурс]. – Режим доступу до ресурсу: https://www.comparitech.com/net-admin/how-to-find-xss-vulnerability

13. OWASP Cheat Sheet Series. Cross Site Scripting Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

14. Ahmed Abdullah Alqarni , Nizar Alsharif, Nayeem Ahmad Khan , Lilia Georgieva, Eric Pardade, Mohammed Y. Alzahrani. MNN-XSS: Modular Neural Network Based Approach for XSS Attack Detection // Computers, Materials & Continua DOI:10.32604/cmc.2022.020389 – 2021.

15. OWASP Foundation. Testing for Stored Cross Site Scripting [Електронний ресурс]. – Режим доступу до ресурсу: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting.html

16. M. Alsaffar, Saud Aljaloud, B. Mohammed, Zeyad Ghaleb Al-Mekhlafi, Tariq S. Almurayziq, G. Alshammari, Abdullah Alshammari. Detection of Web Cross-Site Scripting (XSS) Attacks // Electronics DOI:10.3390/electronics11142212 – 2022.

17. Guangquan Xu, Xiaofei Xie, Shuhan Huang, Jun Zhang, Lei Pan, W. Lou, K. Liang. JSCSP: A Novel Policy-Based XSS Defense Mechanism for Browsers // IEEE Transactions on Dependable and Secure Computing DOI:10.1109/tdsc.2020.3009472 – 2022.

18. PortSwigger. Cross-site scripting [Електронний ресурс]. – Режим доступу до ресурсу: https://portswigger.net/web-security/cross-site-scripting

19. MDN Web Docs. Content Security Policy (CSP) [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

20. Amazon Web Services [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Amazon_Web_Services

ДОДАТОК А

Код файлу article.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><?php echo $_GET['article'] ?></title>
</head>
<body>
    <?php
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $articleLink = $_GET['article'];
        $sql = "SELECT * FROM article WHERE link = :article_link";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(":article_link", $articleLink);
        $stmt->execute();
        echo "<span>/".$articleLink."</span>";
        if($stmt->rowCount() > 0)
        {
            foreach($stmt as $row) {
                $articleName = $row["name"];
                $articleText = $row["fullarticle"];
                $articleLink = $row["link"];

                echo "<h2>".$articleName."</h2>";
                echo "<p>".$articleText."</p>";
            }
        }
    ?>
    <a href="/home.php?user=<?php echo $_GET['user']?>">Back to home page</a>
    <hr />
    <h4>Comments</h4>
    <form action="comment.php" method="POST">
        <p>User: <?php echo $_GET['user']?></p>
        <input type="text" value="<?php echo $_GET['user']?>" style="display: none;"
name="username">
        <input type="text" value="<?php echo $_GET['article']?>" style="display: none;"
name="article">
        <p><textarea name="text" placeholder="Leave comment"></textarea></p>
```

```
        <input type="submit" value="Send">
    </form>
    <hr>
    <?php
        $articleLink = $_GET['article'];
        $sql = "SELECT * FROM comments WHERE article = :article_link";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(":article_link", $articleLink);
        $stmt->execute();
        if($stmt->rowCount() > 0)
        {
            foreach($stmt as $row) {
                $username = $row["username"];
                $text = $row["text"];
                echo "<a href='profile.php?user=".$username."'>".$username."</a>";
                echo "<p>".$text."</p>";
                echo "<hr>";
            }
        }
    ?>
</body>
</html>
```

Код файлу comment.php

```
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $sql = "INSERT INTO comments (text, article, username) VALUES (:text, :article,
:username)";
        $stmt = $conn->prepare($sql);
        $text = $_POST["text"];
        $article = $_POST["article"];
        $username = $_POST["username"];
        $stmt->bindValue(":text", $text);
        $stmt->bindValue(":article", $article);
        $stmt->bindValue(":username", $username);
        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }?>
```

Код файлу home.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h2>Home</h2>
    <hr>
    <?php
    $username = $_GET['user'];
    $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
    $sql = "SELECT * FROM article";
    $result = $conn->query($sql);
    foreach($result as $row) {
        $articleName = $row["name"];
        $articleText = $row["text"];
        $articleLink = $row["link"];
        echo "<h4>".$articleName."</h4>";
        echo "<p>".$articleText."</p>";
        echo "<a href=article.php?article=".$articleLink."&user=".$username."> Read more
</a>";
        echo "<hr />";
    }
    ?>
</body>
</html>
```

Код файлу index.php

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<meta charset="utf-8">
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*;
child-src 'none';">
</head>
<body>
    <h2>Login form</h2>
```

```
    <form action="home.php" method="POST">
    <p>Enter login: <input type="text" name="login" /></p>
    <p>Enter password: <input type="password" name="password" /></p>
    <input type="submit" value="Send">
    </form>
    <a href="register.php">
        Sign up
    </a>
</form>
</body>
</html>
```

Код файлу login.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $sql = "INSERT INTO Users (username, password) VALUES (:username, :password)";
        $stmt = $conn->prepare($sql);
        $login = $_POST["login"];
        $password = $_POST["password"];
        $stmt->bindValue(":username", $login);
        $stmt->bindValue(":password", $password);
        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
</body>
</html>
```

Код файлу register.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign up</title>
</head>
<body>
    <h2>Sign up form</h2>
    <form action="login.php" method="POST">
        <p>Enter login: <input type="text" name="login" /></p>
        <p>Enter password: <input type="password" name="password" /></p>
        <p>Verify password: <input type="password" name="verifypassword" /></p>
        <input type="submit" value="send">
    </form>
    <a href="index.html">
        Log in
    </a>
</body>
</html>
```

Код файлу profile.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    User :
    <?php
        echo $_GET['user'];
    ?>
    <br>
    <br>
    <a href="/home.php">Back to home page</a>
```

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

ДОДАТОК Б

Код файлу article.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><?php echo $_GET['article'] ?></title>
</head>
<body>
    <?php
    $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
    $articleLink = htmlentities($_GET['article'], ENT_QUOTES, 'UTF-8');
    $sql = "SELECT * FROM article WHERE link = :article_link";
    $stmt = $conn->prepare($sql);
    $stmt->bindValue(":article_link", $articleLink);
    $stmt->execute();
    echo "<span>/".htmlentities($articleLink, ENT_QUOTES, 'UTF-8')."</span>";
    if($stmt->rowCount() > 0)
    {
        foreach($stmt as $row) {
            $articleName = $row["name"];
            $articleText = $row["fullarticle"];
            $articleLink = $row["link"];

            echo "<h2>".htmlentities($articleName, ENT_QUOTES, 'UTF-8')."</h2>";
            echo "<p>".htmlentities($articleText, ENT_QUOTES, 'UTF-8')."</p>";
        }
    }

    ?>
    <a href="/home.php?user=<?php echo htmlentities($_GET['user'], ENT_QUOTES, 'UTF-8');
?>">Back to home page</a>
    <hr />
    <h4>Comments</h4>
    <form action="comment.php" method="POST">
        <p>User: <?php echo htmlentities($_GET['user'], ENT_QUOTES, 'UTF-8'); ?></p>
        <input type="text" value="<?php echo htmlspecialchars($_GET['user'], ENT_QUOTES,
'UTF-8'); ?>" style="display: none;" name="username">
```

```
        <input type="text" value="<?php echo htmlspecialchars($_GET['article'], ENT_QUOTES,
'UTF-8'); ?>" style="display: none;" name="article">
    </form>
    <hr>
    <?php
        $articleLink = htmlentities($_GET['article'], ENT_QUOTES, 'UTF-8');
        $sql = "SELECT * FROM comments WHERE article = :article_link";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(":article_link", $articleLink);
        $stmt->execute();
        if($stmt->rowCount() > 0)
        {
            foreach($stmt as $row) {
                $username = $row["username"];
                $text = $row["text"];

                echo "<a href='profile.php?user=".htmlentities($username, ENT_QUOTES, 'UTF-
8')."'>".htmlentities($username, ENT_QUOTES, 'UTF-8')."</a>";
                echo "<p>".htmlentities($username, ENT_QUOTES, 'UTF-8')."</p>";
                echo "<hr>";
            }
        }

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

Код файлу comment.php

```
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $text = $_POST["text"];
        $article = $_POST["article"];
        $username = $_POST["username"];
        // Видалення HTML-тегів з тексту коментаря
        $text = strip_tags($text);

        // Екранування спеціальних символів HTML-коду в тексті коментаря
        $text = htmlspecialchars($text, ENT_QUOTES | ENT_HTML5, 'UTF-8');
        // Вставка коментаря до бази даних
        $sql = "INSERT INTO comments (text, article, username) VALUES (:text, :article,
:username)";
```

```
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(":text", $text);
        $stmt->bindValue(":article", $article);
        $stmt->bindValue(":username", $username);
        $stmt->execute();
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
```

Код файлу home.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h2>Home</h2>
    <hr>
    <?php
    $username = htmlspecialchars($_GET['user']);
    $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
    $sql = "SELECT * FROM article";
    $result = $conn->query($sql);
    foreach($result as $row) {
        $articleName = htmlspecialchars($row["name"]);
        $articleText = htmlspecialchars($row["text"]);
        $articleLink = htmlspecialchars($row["link"]);
        echo "<h4>".$articleName."</h4>";
        echo "<p>".$articleText."</p>";
        echo "<a href=article.php?article=".$articleLink."&user=".$username."> Read more
</a>";
        echo "<hr />";
    }
?>
</body>
</html>
```

Код файлу index.php

```
<!DOCTYPE html>
<html>
```

```
<head>
<title></title>
<meta charset="utf-8">
</head>
<body>
    <p style="color: red;">User login or password is not valid.</p>

    <h2>Login form</h2>
    <form action="home.php" method="POST">
    <p>Enter login: <input type="text" name="login" pattern="[A-Za-z0-9]+" min="4"
max="16"/></p>
    <p>Enter password: <input type="password" name="password" pattern="[A-Za-z0-9]+" min="8"
max="16"/></p>
    <input type="submit" value="Send">
    </form>
    <a href="register.php">
        Sign up
    </a>
</form>
</body>
</html>
```

Код файлу login.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<?php
    try{
        $conn = new PDO("mysql:host=localhost;dbname=diplom", "root", "");
        $sql = "INSERT INTO Users (username, password) VALUES (:username, :password)";
        $stmt = $conn->prepare($sql);
        $login = htmlspecialchars(strip_tags($_POST["login"]), ENT_QUOTES, 'UTF-8');
        $password = htmlspecialchars(strip_tags($_POST["password"]), ENT_QUOTES, 'UTF-8');
        $stmt->bindValue(":username", $login);
        $stmt->bindValue(":password", $password);
        $stmt->execute();
    }
```

```
    catch(PDOException $e) {
        echo $e->getMessage();
    }
?>
</body>
</html>
```

Код файлу register.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign up</title>
</head>
<body>
    <h2>Sign up form</h2>
    <form action="login.php" method="POST">
        <p>Enter login: <input type="text" name="login" /></p>
        <p>Enter password: <input type="password" name="password" /></p>
        <p>Verify password: <input type="password" name="verifypassword" /></p>
        <input type="submit" value="send">
    </form>
    <a href="index.html">
        Log in
    </a>
</body>
</html>
```

Код файлу profile.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
</head>
<body>
    User :
    <?php
        echo $_GET['user'];
    ?>
    <br>
    <br>
    <a href="/home.php">Back to home page</a>
</body>
</html>
```