МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ДОЛІНОВСЬКИЙ Роман Мирославович

Методи захисту веб-застосунку від XSS і CSRF вразливостей / Methods of protecting a web application from XSS and CSRF vulnerabilities

спеціальність: 125 – Кібербезпека

освітньо-професійна програма –Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
Р.М. Доліновський

Науковий керівник
к.т.н., доцент С.В.Івасьєв

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2023 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ – 2023

АНОТАЦІЯ

Магістерська робота на тему “ Методи захисту веб-застосунку від XSS і CSRF вразливостей” зі спеціальності 125 –кібербезпека написана обсягом 74 сторінки і містить 16 ілюстрацій, 2 додатки та 34 джерела за переліком посилань.

Метою роботи дослідження та вивчення методів захисту веб-додатків від поширених загроз безпеки, таких як міжсайтовий скриптинг (XSS) та міжсайтове підроблення запитів (CSRF); оцінка та описання ефективних підходів для зменшення ризиків, пов'язаних з XSS та CSRF.

Методи досліджень базуються на використанні способів виявлення і виправленню XSS та CSRF вразливостей; викоритання автоматизованих інструментів для пошуку вразливостей.

Досліджено можливості проведення попереднього аналізу на предмет XSS та CSRF вразливостей. Проаналізовано та порівняно існуючі стратегії захисту від цих вразливостей та запропоновано вдосконалення методологій захисту

Розроблено локальний продукт, який не тільки демонструє актуальні вразливості, але й висвітлює методи захисту від них. Такий практичний підхід дає реальне уявлення про те, як вразливості можуть проявлятися в реальних веб-додатках і як їх можна ефективно усунути або запобігти їм.

Результати роботи можуть бути застосовані в сфері веб-розробки та безпеки для удосконалення заходів захисту від атак XSS і CSRF у веб-додатках. Організації, що розробляють та підтримують веб-системи, можуть використовувати рекомендації та методології цієї роботи для забезпечення вищого рівня безпеки своїх продуктів та захисту користувачів від потенційних кіберзагроз.

КЛЮЧОВІ СЛОВА: XSS, CSRF, МІЖСАЙТОВИЙ СКРИПТИНГ, ПІДРОБКА МІЖСАЙТОВИХ ЗАПИТІВ,ВЕБ БЕЗПЕКА, ВЕБ ЗАСТОСУНОК.

ABSTRACT

The master's thesis on "Methods of protecting a web application from XSS and CSRF vulnerabilities" in the speciality 125 - Cybersecurity is 74 pages long and contains 16 illustrations, 2 appendices and 34 references.

The master's thesis on "Methods of protecting a web application from XSS and CSRF vulnerabilities" in the speciality 125 - Cybersecurity is written in 89 pages and contains 16 illustrations, 2 appendices and 34 sources in the list of references.

The purpose of the work is to research and study methods of protecting web applications from common security threats such as cross-site scripting (XSS) and cross-site request forgery (CSRF); to evaluate and describe effective approaches to mitigate the risks associated with XSS and CSRF.

The research methods are based on the use of methods for detecting and fixing XSS and CSRF vulnerabilities; the use of automated tools for finding vulnerabilities.

The possibilities of conducting a preliminary analysis for XSS and CSRF vulnerabilities are investigated. Analysed and compared the existing strategies for protecting against these vulnerabilities and proposed improvements to protection methodologies

Developed a local product that not only demonstrates current vulnerabilities but also highlights methods of protecting against them. This practical approach gives a realistic idea of how vulnerabilities can manifest themselves in real web applications and how they can be effectively addressed or prevented.

The results of this work can be applied in the field of web development and security to improve the protection against XSS and CSRF attacks in web applications. Organisations that develop and maintain web-based systems can use the recommendations and methodologies of this work to ensure a higher level of security for their products and protect users from potential cyber threats.

KEYWORDS: XSS, CSRF, CROSS-SITE SCRIPTING, CROSS-SITE REQUEST FORGERY, WEB SECURITY, WEB APPLICATION.

ЗМІСТ

ВСТУП

Актуальність роботи. Веб-додатки відіграють ключову роль у багатьох секторах, включаючи електронну комерцію, фінанси та охорону здоров'я. Зі збільшенням залежності від веб-додатків зростає і потенційний ризик вразливостей безпеки. XSS та CSRF є одними з найпоширеніших та найбільших загроз безпеці, з якими стикаються веб-додатки. XSS дозволяє зловмисникам вставляти шкідливі скрипти у веб-сторінки, що може призвести до несанкціонованого доступу, крадіжки даних або перехоплення ключа сеансу. CSRF, з іншого боку, використовує довіру, яку веб-додаток надає користувацькому браузеру, і виконує небажані дії від імені користувача.

Постійний розвиток кіберзагроз ще більше підкреслює необхідність усунення XSS та CSRF вразливостей. Оскільки загрози постійно розвиваються і з'являються нові вектори та методи атак. Зосередившись на методах захисту, організації можуть дослідити ці загрози та запропонувати адаптивні стратегії для боротьби з ними.

Мета роботи полягає в дослідженні та вивченні методів захисту веб-додатків від поширених загроз безпеки, таких як міжсайтовий скриптинг (XSS) та міжсайтове підроблення запитів (CSRF). Крім того робота має на меті оцінити та запропонувати ефективні підходи для зменшення ризиків, пов'язаних з XSS та CSRF.

Для досягнення цієї мети було визначено наступні **завдання**:

- проаналізувати атаки XSS і CSRF, та визначити збитки від них;
- дослідити причини виникнення вразливостей XSS і CSRF;
- дослідити можливості попереднього аналізу продукту на вразливості XSS і CSRF;
- дослідити існуючі стратегії захисту від вразливостей CSRF і XSS, та запропонувати удосконалення методологій захисту;

- покроково розробити локальний продукт і показати вразливості, та методи захисту від них;

Об’єкт дослідження - вразливості веб-застосунків.

Предмет дослідження – моделі вразливостей XSS і CSRF веб-застосунків.

Наукова новизна одержаних результатів: комплексно досліджено моделі загроз з використанням атак XSS і CSRF та розроблено рекомендації щодо забезпечення захисту від цього виду атак.

Практична цінність: розроблено продукт який відображає наслідки вразливостей XSS і CSRF, та практичні заходи захисту від них.

Публікації та апробація до кваліфікаційної роботи.

1. Доліновський Р.М., Вразливості XSS: валідація введених даних. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп’ютерно – інтегровані технології» (АКІТ -2023), Тернопіль, 2023. 127 -129 с.

2. Доліновський Р.М., Вразливості CSRF: види та методи захисту. Збірник матеріалів науково-практичної конференції «Кібербезпека та комп’ютерно – інтегровані технології» (КБКІТ -2023), Тернопіль, 2023. 68-70 с.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз стеку веб-технологій

З початком розвитку веб-технологій можна було побачити стрімкий прогрес у цьому напрямку і спроби створити всеосяжний та інтерактивний онлайн світ. Все почалося в 1969 році зі створення Інтернету, який вважається найбільш значущим технологічним винаходом попереднього століття. Ця ключова подія стала відправною точкою для подальших перетворень у сфері комунікації та обміну інформацією.

У 1989 році Тім Бернерс-Лі представив два ключові терміни: HTTP та HTML. HTTP, також відомий як протокол передачі гіпертексту, став основою для обміну даними в Інтернеті, тоді як HTML, або мова розмітки гіпертексту, дозволила створювати статичні веб-сторінки. Ці два елементи визначили еру інформації доступної для всіх з будь-якої точки світу та поклали початок веб-технологій.

Наступний вирішальний етап припав на середину та кінець 1990-х років, який зазвичай називають війною браузерів. Конкуренція між Netscape Navigator та Internet Explorer призвела до появи нових технологій та стандартизації веб-технологій, які були стандартизовані зусиллями W3C у 1994 році. Це забезпечило стабільність і сумісність веб-технологій, що в кінцевому підсумку сприяло швидкому зростанню онлайн-світу.

Поява JavaScript та серверних мов програмування в середині 1990-х років розширила горизонти веб-розробки. Спочатку JavaScript використовувався для створення динамічних функцій на стороні клієнта, але зараз він є ключовим елементом у створенні інтерактивних інтерфейсів для веб-додатків. Серверні мови програмування, такі як PHP, також допомогли розробникам створювати більш складні та функціональні веб-додатки.

Поява AJAX у 2000 році ознаменувала новий етап в еволюції веб-технологій. Ця інновація зробила можливим асинхронне оновлення вмісту сторінки без необхідності її перезавантаження, змінивши спосіб розробки

веб-додатків. Концепція Web 2.0 наголошувала на переході від статичних до динамічних та інтерактивних сайтів, де користувачі брали активну участь у створенні контенту.

За останні роки мобільні технології значно прогресували. Зростання популярності мобільних веб-додатків підкреслило важливість адаптивного дизайну, який оптимізує веб-сайти під різні пристрої та розміри екранів.

Використання сучасних фреймворків і технологій, зокрема React, Angular і Vue.js, дозволяє створювати складні та ефективні веб-додатки. Node.js дозволяє використовувати JavaScript як для клієнтської, так і для серверної розробки, відкриваючи нові можливості для розробки.

Кожен еволюційний крок у розвитку веб-технологій розширював можливості та покращував користувацький досвід. Історія розвитку веб-технологій, від базових статичних сторінок до високопродуктивних веб-додатків, свідчить про інновації та залишається визначальною характеристикою цієї цифрової ери.

1.2 Веб-вразливості та загрози кібербезпеки

Вразливості тісно пов’язані з розвитком технологій, і для кожної нової технології було справжнім викликом протистояти десяткам тисяч хакерів які вишукували потенційну ваду в кожному кутку її коду.

Зростання Інтернету та розвиток веб-технологій призвели до появи нових веб-вразливостей та значних кібератак. На ранніх стадіях веб-розробки, особливо в 1990-х роках, спостерігалося поєднання ентузіазму та обмежень у сфері кібербезпеки. Веб-сайти створювалися здебільшого з використанням простих HTML-сторінок, які надавали мало можливостей для атак. Зі зростанням складності веб-додатків виникла потреба в нових технологіях, які також створили нові можливості для потенційних загроз. Спочатку серйозні вразливості виникали через недоліки в реалізації серверних і клієнтських компонентів веб-додатків.

Вразливості веб-технологій були усунені завдяки створенню консорціуму W3C у 1994 році. Однак стандартизація призвела до більш досконалих і цілеспрямованих атак. Дві основні загрози, що з'явилися, - це атаки на впровадження HTML-коду (XSS) та підробка міжсайтових запитів (CSRF). XSS-атаки полягають у впровадженні шкідливих скриптів у веб-сторінки для крадіжки конфіденційних даних, тоді як CSRF-атаки маніпулюють авторизованими користувачами, змушуючи їх виконувати небажані дії. Ці вразливості викликали занепокоєння розробників, що призвело до використання заходів безпеки, включаючи фільтрацію вхідних даних, токени валідації для запобігання CSRF та інші практики безпеки. Крім того, були введені нові стандарти, такі як Content Security Policy (CSP), щоб зменшити ризик XSS-атак і обмежити виконання шкідливих скриптів.

- Code Red.

Однією з перших серйозних хакерських атак, яка перевернула погляд багатьох компаній на безпекові якості їхніх продуктів став Code Red. Це була атака що використовувала вразливість в Microsoft IIS для атаки на веб-сервери. Вона скомпрометувала багато веб-сайтів і є прикладом того, як використання вразливостей може самостійно поширюватися і завдавати шкоди.

Code Red став відомим як комп'ютерний черв'як, націлений на програмне забезпечення веб-сервера Internet Information Services (IIS) компанії Microsoft. Ця шкідлива програма з'явилася в липні 2001 року і швидко поширилася по всьому Інтернету, спричинивши значні перебої в роботі. Використовуючи відому вразливість у компоненті служби індексування IIS, Code Red виконував довільний код на серверах.

Хробак працював, надсилаючи ретельно розроблений HTTP-запит на вразливі сервери. Якщо на сервері не було виправлення вразливості, Code Red міг успішно скомпрометувати його. Потім хробак поширювався на інші вразливі сервери, генеруючи випадкові IP-адреси та передаючи корисне навантаження експлойту.

Однією з особливостей Code Red була його здатність пошкоджувати веб-сайти, розміщені на скомпрометованих серверах. За замовчуванням веб-сторінка замінювалася повідомленням: "Welcome to http://www.worm.com! Hacked by Chinese!". Незважаючи на те, що таке пошкодження є видимою ознакою скомпрометованого сервера, швидке поширення Code Red змусило Microsoft випустити патчі безпеки та попередження, закликаючи користувачів оновити свої системи. На жаль, черв'як всеодно продовжував заражати сервери, які не встановлювали патчів. Пізніше з'явився Code Red II, наступний варіант вірусу, який демонстрував ще більш руйнівну поведінку.

Вплив Code Red був значним, тож він підкреслив важливість своєчасного встановлення патчів безпеки та висвітлив потенційні ризики, пов'язані з незахищеними веб-серверами. Він також акцентував увагу на необхідність підвищення обізнаності щодо питань кібербезпеки та важливість вжиття проактивних заходів для захисту від відомих вразливостей. Цей інцидент справив серйозний вплив на те, як організації підходять до забезпечення безпеки своїх веб-серверів, і відіграв ключову роль у розвитку практики кібербезпеки.

- Samy Worm.

Якщо ж ми говоримо про вразливості XSS і CSRF, то найбільш відомою атакою такого типу є Samy Worm, також відома як хробак MySpace. У 2005 році хакер Samy Kamkar здійснив цю атаку на MySpace, відому на той час соціальну мережу. Метод роботи хробака полягав у використанні постійної (stored XSS) XSS-уразливості в інфраструктурі MySpace. Ця вразливість надавала Kamkar можливість впроваджувати шкідливий JavaScript-код в особистий профіль користувача.

Особливістю Samy Worm був механізм саморозповсюдження. Впроваджений скрипт виконувався автоматично, коли інші користувачі переглядали заражений профіль Samy. Крім того, він розумно модифікував профіль користувача, додаючи Samy в друзі і приховано вбудовуючи скрипт в їхні профілі. Саме в цьому полягала і вразливість CSRF, яка дозволяла

виконувати запити від імені авторизованих користувачів. В результаті можливості самовідтворення хробак швидко поширився, заразивши тисячі профілів MySpace всього за кілька годин. Хробак Samy швидко здобув славу однієї з найшвидше поширюваних шкідливих програм свого часу. Хоча корисне навантаження хробака було здебільшого нешкідливим, спрямованим на збільшення кількості друзів Samy, воно продемонструвало потенціал зловмисників використовувати XSS та CSRF вразливості для більш шкідливих цілей. MySpace довелося тимчасово закрити, щоб локалізувати хробака та усунути вразливість. Цей випадок став гучним нагадуванням про важливість захисту веб-додатків від XSS та CSRF атак.

Хробак Samy є яскравим прикладом у сфері веб-безпеки, який підкреслює необхідність надійного захисту від таких вразливостей і спонукає до вдосконалення процедур безпеки веб-розробників і платформ. Він слугує прикладом потенційних ризиків, пов'язаних з користувацьким контентом, та важливістю комплексної перевірки вхідних даних і вихідного кодування для запобігання таким атакам.

- Heartland Payment Systems.

Кібератака на Heartland Payment Systems у 2008 році стала масштабним інцидентом, який підкреслив серйозні наслідки вразливостей SQL-ін'єкцій у веб-додатках. Як великий платіжний процесор, Heartland Payment Systems стала мішенню складної атаки, яка призвела до витоку мільйонів записів про кредитні картки.

Наприкінці 2008 року Heartland Payment Systems зазнала значного порушення безпеки, наслідком якого, було викрадення конфіденційних фінансових даних. Зловмисники використали SQL-ін'єкцію для пошуку вразливостей в системі обробки платежів компанії.

SQL-ін'єкція - це метод кібератаки, при якому шкідливий SQL-код вставляється в поля введення або команди. У контексті веб-додатків це зазвичай передбачає маніпуляції з даними, введеними користувачем, для виконання несанкціонованих SQL-запитів до бази даних. У випадку з

Heartland зловмисники використали цю вразливість для отримання несанкціонованого доступу до бази даних платіжної системи.

Хакерам вдалося отримати величезну кількість конфіденційних даних, включаючи номери кредитних карток, імена власників карток та інші деталі транзакцій. Цей масштабний витік даних став одним з найбільш значущих порушень безпеки в історії. Компанія Heartland Payment Systems виявила порушення в січні 2009 року після того, як помітила несанкціоновані транзакції в своїх системах. Компанія негайно звернулася за допомогою до експертів, щоб визначити масштаби атаки. Аналіз виявив наявність шкідливого програмного забезпечення, яке було встановлено для збору та передачі конфіденційних даних.

Порушення безпеки Heartland мало серйозні наслідки як для компанії, так і для всієї платіжної індустрії. Heartland зіткнулася з юридичними проблемами, фінансовими втратами та репутаційними збитками. Крім того, інцидент спричинив ретельне розслідування та регуляторні дії в платіжному секторі.

Цей випадок змушує зрозуміти надзвичайну важливість надійних практик кібербезпеки, підкреслює необхідність ретельних заходів безпеки, включаючи безпечне кодування, регулярний аудит безпеки та поглиблене тестування вразливостей, включно з такими як виявлення та усунення SQL-ін'єкцій. Вкрай важливо впроваджувати комплексні заходи безпеки для забезпечення повного захисту від кіберзагроз.

Крім того, він став поштовхом для того, щоб організації надавали пріоритет кібербезпеці та впроваджували суворі заходи для захисту від нових кіберзагроз, особливо тих, що спрямовані на серцевину систем обробки платежів. Досвід, отриманий в результаті витоку даних Heartland, продовжує впливати на стратегії кібербезпеки і допомагає просувати зусилля, спрямовані на зміцнення стану безпеки організацій, що обробляють фінансову інформацію.

- Likejacking.

Атака Likejacking на Facebook - це приклад оманливої природи схем клікджекінгу, які використовують довіру користувачів до звичних онлайн-інтерфейсів. Ці атаки є особливо хитрими, оскільки вони тонко маніпулюють взаємодією користувачів, що призводить до непередбачуваних результатів. Зловмисники обманюють користувачів, розміщуючи прозорі шари або маніпулюючи макетами веб-сторінок, щоб змусити їх взаємодіяти з прихованими елементами, що може призвести до порушення безпеки в Інтернеті.

Likejacking використав схильність користувачів взаємодіяти із захопливим або інтригуючим контентом. Наприклад, користувач може мимоволі активувати приховану дію "Мені подобається" у Facebook, натиснувши на цікаве, на перший погляд, посилання або відео. Широке використання соціальних мереж, таких як Facebook, може посилити вплив клікджекінг-атак, спричиняючи серйозні наслідки, які впливають на соціальну мережу жертви.

Для запобігання таким атакам було запроваджено різні контрзаходи, такі як скрипти для розриву фреймів, що використовуються Facebook, що зупиняють несанкціоноване вбудовування його сторінок у фрейми. Крім того, платформа використовує механізми, які виявляють і блокують спроби клікджекінгу. Постійна еволюція кіберзагроз вимагає постійної адаптації, і вкрай важливо, щоб користувачі не втрачали пильності.

1.3 Аналіз існуючих методів захисту від веб-вразливостей

Веб-вразливості охоплюють широкий спектр слабких місць у системі безпеки, якими можуть скористатися зловмисники, щоб підірвати цілісність, конфіденційність або доступність веб-додатків. До поширених типів кібератак належать міжсайтовий скриптинг (XSS), коли зловмисники впроваджують шкідливі скрипти у веб-контент; міжсайтова підробка запитів (CSRF), яка передбачає несанкціоновані дії, що виконуються від імені

авторизованих користувачів; SQL-ін'єкція, що дозволяє зловмисникам маніпулювати запитами до баз даних; неправильні конфігурації безпеки, що виникають через неправильно налаштовані параметри; витік конфіденційних даних, що загрожує розголошенням конфіденційної інформації.

Вразливості IDOR можуть призвести до несанкціонованого доступу до чутливих об'єктів. Потенційними вразливостями є заголовки безпеки, завантаження файлів та перехоплення кліків, а також несанкціоновані перенаправлення та переадресації, які можуть створювати ризики перенаправлення користувачів на шкідливі сайти. Аутентифікація та управління сеансами - це інші області, схильні до вразливостей і вразливості до DoS і DDoS-атак, які мають на меті вивести з ладу сервіси. Для ефективного функціонування вкрай важливо забезпечити безпеку API. Забезпечення надійного захисту від вразливостей вимагає комплексної стратегії, яка передбачає безпечне кодування, регулярний аудит, навчання користувачів та впровадження технологій безпеки.

1.3.1 XSS вразливості.

XSS-вразливості часто виникають при розробці веб-додатків через низку поширених помилок. Однією з таких помилок є недостатня перевірка вхідних даних, коли розробники не можуть належним чином перевірити дані, введені користувачем, перед тим, як відображати їх на веб-сторінках. Цей аспект дозволяє зловмисникам впроваджувати шкідливі скрипти, які потім виконуються браузерами користувачів, які нічого не підозрюють. Крім того, недбале кодування вихідних даних сприяє виникненню XSS-уразливостей, коли розробники нехтують адекватним кодуванням динамічного контенту перед його відображенням у браузері.

Ще однією поширеною помилкою є використання небезпечних функцій JavaScript, особливо тих, що пов'язані з маніпуляціями з об'єктною моделлю документа (DOM). Розробники іноді покладаються на незахищені API або ігнорують реалізацію заголовків безпеки, тим самим дозволяючи зловмисникам використовувати доступ браузера до певних функцій. Крім

того, неналежне поводження з користувацьким контентом, наприклад, неналежна фільтрація або перевірка в коментарях користувачів або полях введення, може призвести до XSS-уразливостей.

Недостатня обізнаність та освіченість розробників щодо ризиків, пов'язаних з XSS, також є важливим фактором. Відсутність актуальної інформації про останні передові практики безпеки, нехтування заголовками безпеки, такими як Content Security Policy (CSP), та недооцінка важливості принципів безпечного кодування - все це сприяє поширенню XSS-уразливостей у веб-додатках.

Перший з методів який захищає користувачів від такого типу атак, який виконується на найнижчому рівні, рівні вводу даних - валідація вхідних даних. Послідовність у застосуванні цієї практики може допомогти захистити веб-додатки від потенційних загроз. Процес включає в себе систематичну перевірку і санітарну обробку вхідних даних користувача на стороні сервера, щоб переконатися, що вони відповідають очікуваним форматам і не містять потенційно шкідливого коду. Перевіряючи вхідні дані, розробники можуть запобігти впровадженню шкідливих скриптів у веб-контент. Ця дія захищає від використання вразливостей, які можуть поставити під загрозу безпеку браузерів користувачів. Ця процедура вимагає суворого дотримання заздалегідь визначених критеріїв введення, відкидаючи будь-які дані, що відхиляються від очікуваних шаблонів.

Мінімізація ризику впровадження шкідливих скриптів у веб-додатки передбачає також кодування вихідних даних. Цей процес перетворює створений користувачем вміст у формат, який не дозволяє браузеру інтерпретувати його як виконуваний код. Нейтралізуючи спеціальні символи, які можуть бути частиною сценарію, кодування виводу гарантує, що дані, введені користувачем, розглядаються як звичайний текст, а не як частина коду, що потребує виконання. Цей запобіжний крок є життєво важливим для запобігання XSS-атакам і захисту користувачів від можливої шкоди. Він співпрацює з іншими заходами безпеки, включаючи перевірку даних, щоб

посилити захист веб-додатків від постійно мінливого середовища кіберзагроз.

Наступний важливий захід безпеки, який ефективно знижує ризик міжсайтових скриптових атак - це політика безпеки вмісту (Content Security Policy, CSP). CSP дозволяє веб-розробникам контролювати виконання скриптів на веб-сторінках за допомогою комплексного набору директив. Це досягається за допомогою HTTP-заголовків, які дозволяють визначати надійні джерела контенту, скриптів, таблиць стилів та інших ресурсів. Спеціально вказуючи дозволені домени і типи контенту, CSP запобігає виконанню браузером скриптів з несанкціонованих джерел, ефективно зупиняючи впровадження шкідливих скриптів.

Ця потужна стратегія захисту слідує принципу найменших привілеїв, ефективно зменшуючи потенційний вплив XSS-уразливостей. Заголовки CSP надають розробникам точний контроль над політиками завантаження контенту завдяки можливості вказувати різні директиви. Ці директиви включають 'default-src', 'script-src' і 'style-src'. Крім того, CSP підтримує атрибути "nonce" і "hash", які дозволяють розробникам створювати динамічні політики, що підвищують безпеку. При правильному налаштуванні заголовки Content Security Policy (CSP) не тільки допомагають запобігти атакам Cross-Site Scripting (XSS), але також дають початок комплексній стратегії захисту, доповнюючи інші заходи безпеки, такі як перевірка вхідних даних і кодування вихідних даних.

Використання зовнішніх файлів скриптів і безпечних API для динамічного контенту, а не вбудованого JavaScript, також має не останнє значення. Часті оновлення та підтримка програмних компонентів, зокрема бібліотек і фреймворків, відіграють важливу роль у зменшенні кількості вразливостей. Розробники повинні бути в курсі новітніх практик безпеки, постійно вчитися і виконувати комплексні перевірки безпеки, які включають в себе такі методи, як тестування на проникнення. Впровадження безпечного керування сеансами, використання прапорців "Тільки HTTP" і "Безпечний"

для файлів cookie, а також суворий моніторинг і ведення журналів можуть посилити захист програми від XSS-загроз.

CSRF вразливості.

Помилки при розробці веб-додатків можуть призвести до міжсайтових підробок запитів (CSRF), які ненавмисно ставлять під загрозу безпеку користувачів. Однією з таких помилок є неадекватна реалізація заходів захисту від CSRF, наприклад, не створення та не перевірка унікальних маркерів захисту від CSRF для кожного сеансу користувача або відсутність цих маркерів у критичних формах та запитах, що потенційно створює вразливості.

Розробники можуть не помітити важливість впровадження політики однакового походження, яка дозволяє зловмисникам генерувати запити, що маніпулюють довірою між браузером користувача та певним сайтом. Крім того, нерегулярна або недостатня перевірка даних, введених користувачем, особливо в ситуаціях, що стосуються чутливих операцій, таких як переказ коштів або оновлення паролів, є ще однією частою вразливістю.

Покладання виключно на механізми безпеки на стороні клієнта без надійної перевірки на стороні сервера може зробити системи вразливими до CSRF-атак. Нехтування безпечними методами кодування, включаючи неналежну перевірку особи користувача, може мимоволі надати зловмисникам можливість здійснювати дії від імені авторизованих користувачів. Недостатня поінформованість та обізнаність користувачів про ризики CSRF може призвести до ненавмисного ініціювання дій CSRF за допомогою соціальної інженерії.

Щоб посилити захист від CSRF-атак, розробникам потрібно зробити більше, ніж просто використовувати токени Anti-CSRF. Необхідно створити комплексну систему безпеки, яка передбачає постійний моніторинг та адаптивні механізми реагування. Як приклад, візьмемо випадок, коли банківський додаток включає в себе надійну систему відстеження взаємодії з користувачем. Система реєструє стандартні транзакції та використовує

алгоритми виявлення аномалій для виявлення незвичайних моделей поведінки користувачів. Це дозволяє системі виявляти потенційні атаки CSRF в режимі реального часу. У разі підозри на інцидент, пов'язаний з CSRF, оперативно впроваджуються спеціальні процедури реагування на інциденти, щоб зменшити загрозу, звести до мінімуму будь-які перешкоди для користувачів і захистити конфіденційність чутливої фінансової інформації.

Активна участь у спільноті кібербезпеки є цінним джерелом знань та ідей. Розробники, які відвідують конференції з безпеки, беруть участь у форумах та роблять внесок у платформи розвідки загроз, отримують колективне розуміння нових векторів атак CSRF. Наприклад, обговорення в спільноті можуть виявити нові тактики, які застосовують зловмисники, такі як використання методів соціальної інженерії для маніпулювання користувачами, щоб змусити їх несвідомо виконувати дії CSRF. Розробники можуть використовувати колективну обізнаність для адаптації своїх стратегій захисту, закриваючи потенційні вразливості ще до того, як вони будуть використані.

Під час навчання користувачів дуже важливо використовувати релевантні приклади для пояснення концепції CSRF-атак. Розглянемо сценарій, коли користувач заходить на відому платформу електронної комерції і несвідомо запускає фінансову транзакцію на іншому веб-сайті за маніпулятивним посиланням. Завдяки підвищенню обізнаності про такі ризики та наголошенню на важливості бути пильними в Інтернеті, користувачі стають більш досвідченими у виявленні підозрілої діяльності. Цей метод, орієнтований на клієнтів, разом із захисними заходами, такими як токени Anti-CSRF, створює потужну дворівневу систему захисту від ризиків CSRF.

Отже, використання токенів Anti-CSRF є важливим елементом онлайн-безпеки для досягнення ефективних результатів, але ще більш ефективним є їх інтеграція в комплексну інфраструктуру безпеки. Розробники повинні

використовувати переваги постійного моніторингу, взаємодіяти зі спільнотою безпеки та проінструктувати користувачів щодо встановлення декількох заходів захисту від порушень CSRF. Такий підхід не лише захищає від поточних загроз, але й дозволяє додаткам адаптуватися до нових викликів кібербезпеки, забезпечуючи користувачам безпечне онлайн-середовище.

SQL-ін’єкції

Вразливості пов’язані з SQL залишаються значною і дуже поширеною загрозою для веб-додатків. Ці вразливості часто виникають через повторювані помилки, допущені як на етапі розробки, так і на етапі підтримки продукту. Однією з найпоширеніших помилок є неналежна перевірка та санітарна обробка даних, що вводяться користувачами. Недостатня перевірка даних, введених користувачем, створює можливість для зловмисників вставляти в додаток шкідливі SQL-запити. Впроваджені запити можуть маніпулювати базою даних, що потенційно може призвести до несанкціонованого доступу та витоку чи втрати конфіденційної інформації.

Динамічні SQL-запити без належної параметризації також є поширеною помилкою. Конкатенація вхідних даних користувача в операторах SQL дозволяє зловмисникам маніпулювати структурою запиту, що полегшує несанкціонований доступ до даних або маніпуляції з ними.

Неправильна обробка помилок - ще один вразливий фактор що провокує вразливості в SQL. Розкриття складних повідомлень про помилки користувачам у виробничих умовах ненавмисно розкриває структуру бази даних, надаючи зловмисникам цінну інформацію для створення цілеспрямованих атак на SQL-ін'єкції. Крім того, слабкий контроль доступу та неправильне управління привілеями може призвести до надання надмірних дозволів обліковим записам додатків. Це дає зловмисникам можливість використовувати такі права для отримання несанкціонованого

доступу або маніпулювання даними, тим самим ставлячи під загрозу загальний рівень безпеки.

Для ефективного вирішення проблем безпеки SQL необхідний ретельний підхід, який передбачає впровадження безпечних методів кодування, проведення періодичних оцінок безпеки, створення протоколів обробки помилок, забезпечення контролю доступу та дотримання послідовності у використанні специфічних термінів, абревіатур і символів. Важливо використовувати точну лексику, коли цього вимагає ситуація, зменшувати складність і уникати надмірності або фраз-заповнювачів, підтримувати нейтральний тон і дотримуватися стандартних мовних конвенцій, щоб досягти ясності, зв'язності і граматичної правильності. Впровадження цих заходів може підвищити стійкість веб-додатків до постійно мінливих загроз вразливостей SQL, захистити конфіденційні дані та зберегти цілісність додатків і баз даних.

Параметризовані запити або підготовлені оператори також використовуються для захисту від атак SQL-ін'єкцій. Ці методи дозволяють розробникам відокремити введення користувача від SQL-запитів, створюючи захисний бар'єр, який запобігає впровадженню шкідливого коду. Послідовність і дотримання стандартизованої мови та точний вибір слів мають вирішальне значення для реалізації цих заходів безпеки. Крім того, офіційний тон з акцентом на однозначну мову і логічну, послідовну структуру допоможуть забезпечити ясність і узгодженість документації. Завдяки використанню параметрів як заповнювачів для користувацького вводу, вхідні дані розглядаються виключно як дані, а не як виконуваний код. Такий підхід підвищує безпеку програми, запобігаючи маніпулюванню операціями з базою даних за допомогою ін'єкційних SQL-запитів.

Надійна реалізація валідації вхідних даних є важливим механізмом захисту, який гарантує, що вхідні дані відповідають очікуваним форматам, включаючи перевірку довжини, формату та діапазону. Базової перевірки типів даних недостатньо. Ретельна перевірка користувацьких даних на

відповідність попередньо визначеним критеріям значно мінімізує ризики SQL-ін'єкцій для розробників. Така практика підвищує безпеку, відмовостійкість і стійкість додатків до помилок.

Дотримання принципу найменших привілеїв є потужним захисним методом для зменшення потенційного впливу атак SQL-ін'єкцій. Цей принцип вимагає надання обліковим записам бази даних, пов'язаним з додатком, лише мінімального рівня доступу, необхідного для виконання покладених на них завдань. Обмеження привілеїв таких облікових записів зменшує потенційну шкоду, яку може завдати успішна SQL-ін'єкція, і, як наслідок, робить середовище бази даних безпечнішим.

Регулярне проведення аудиту безпеки та оцінки вразливостей є важливим для виявлення та усунення потенційних вразливостей SQL-ін'єкцій. Поєднання автоматизованих інструментів і ручного тестування має вирішальне значення для виявлення слабких місць у коді програми та її взаємодії з базою даних. Такі перевірки не тільки слугують профілактичним заходом, але й допомагають у постійному вдосконаленні протоколів і практик безпеки.

Встановивши брандмауер веб-додатків (WAF), можна створити додатковий рівень захисту від атак SQL-ін'єкцій. WAF ретельно перевіряє вхідний трафік і виявляє шаблони, які відповідають розпізнаним векторам атак SQL-ін'єкцій. Активно виявляючи та запобігаючи зловмисним спробам, WAF працює як динамічний щит, значно підвищуючи загальний рівень безпеки веб-додатків.

Ще один важлививй аспект безпеки - безперервне навчання методам кодування що активно розвиває культуру безпеки. Висвітлюючи ризики SQL-ін'єкцій та ознайомлюючи розробників з найновішими заходами безпеки, ними засвоюються необхідні знання та навички, тож вони можуть ефективно підтримувати надійні протоколи безпеки. Навчання підвищує обізнаність про безпеку і дає розробникам можливість протистояти загрозам

SQL-ін'єкцій, забезпечуючи надійні заходи безпеки для додатків.

ClickJacking

Вразливості клікджекінгу, поширеної загрози веб-безпеки, коли зловмисник вводить користувача в оману, змушуючи його натиснути на замаскований або невидимий елемент, що призводить до небажаних дій, можуть посилюватися різними помилками при веб-розробці. Однією з поширених помилок є недостатня реалізація технік обходу фреймворків.

Розробники можуть використовувати методи для запобігання фреймування їхніх веб-сторінок іншими сайтами, але такі заходи часто є неповними або неправильно налаштованими, що призводить до появи вразливостей. Іншою поширеною помилкою є відсутність відповідних заголовків безпеки, таких як X-Frame-Options, які пояснюють, як веб-сторінка повинна бути вбудована у фрейми.

Нехтування цими заголовками або їх неправильна реалізація наражає веб-сторінки на ризик атак типу клікджекінг. Покладатися виключно на засоби захисту на стороні клієнта, такі як рішення на основі JavaScript, не підкріплюючи їх засобами захисту на стороні сервера, може бути критичною помилкою. Розробники можуть недооцінювати важливість перевірки на стороні сервера і замість цього зосереджуватися на клієнтському скрипті, залишаючи свої додатки вразливими до спроб перехоплення кліків, які обходять клієнтські засоби контролю.

Не варто недооцінювати важливість регулярного аудиту та оновлення системи безпеки. З розвитком веб-технологій можуть з'являтися вразливості в системі безпеки, а нехтування оновленням програмних компонентів, таких як фреймворки та бібліотеки, може призвести до того, що веб-додатки стануть вразливими до ризиків клікджекінгу.

Впровадження політики безпеки контенту (CSP) використовується для здійснення контролю над джерелами, з яких контент може завантажуватися на веб-сайт. Добре розроблена CSP визначає авторизовані домени, які

можуть вбудовувати веб-додатки, тим самим знижуючи ризик клікджекінгу, обмежуючи місця, де контент може бути відображений в фреймі. Ця політика діє як надійний превентивний захід проти різних типів веб-атак, тим самим підвищуючи загальний рівень безпеки додатку.

Вбудовування скриптів, що унеможливлюють відображення продукту в фреймі, у веб-сторінки є ефективною технікою для запобігання спробам клікджекінгу. Зазвичай написані на JavaScript, ці скрипти активно виявляють, чи не знаходиться сторінка у фреймі на іншому сайті, і вживають коригувальних заходів, щоб вийти з фрейму. Забезпечуючи відображення контенту виключно в призначеному для нього контексті, скрипти для розриву фрейму надають додатковий рівень захисту від клікджекінгу, тим самим посилюючи безпеку веб-додатку.

Візуальні індикатори відіграють не менш важливу роль у підвищенні обізнаності користувачів і полегшенні ідентифікації легального контенту. Завдяки помітному відображенню водяних знаків або логотипів, які зловмисникам важко відтворити, користувачі можуть перевірити автентичність веб-сторінки. Ці візуальні підказки дозволяють користувачам приймати обґрунтовані рішення щодо надійності контенту, з яким вони взаємодіють, що сприяє підвищенню рівня безпеки користувачів.

Інформування користувачів про ризики, пов'язані з клікджекінгом, є фундаментальним аспектом комплексної стратегії безпеки. Підвищуючи обізнаність про потенційні загрози і підкреслюючи важливість перевірки легітимності веб-сайтів, користувачі стають активними учасниками захисту від клікджекінгу. Проінформовані користувачі з більшою ймовірністю проявлятимуть обережність, розпізнаватимуть підозрілі дії та повідомлятимуть про потенційні інциденти безпеки, тим самим сприяючи покращенню веб-безпеки.

Інший метод, який в комплексі повинен працювати з попередніми це впровадження реферерів. Він застосовується для регулювання розкриття інформації в заголовку HTTP-реферaлу. Такий підхід допомагає запобігти

потраплянню конфіденційних даних на потенційно шкідливі сайти, які можуть здійснювати клікджекінг-атаки. Ретельно налаштована політика реферерів включає додатковий захист, який мінімізує можливість ненавмисного розкриття інформації під час взаємодії користувачів.

Важливе значення має посилення інфраструктури безпеки за допомогою брандмауера веб-додатків (WAF), який має спеціалізовані можливості для виявлення та запобігання клікджекінгу. WAF діє як проактивний захист, аналізуючи вхідний трафік, виявляючи шаблони, що вказують на спроби клікджекінгу, і відфільтровуючи потенційно шкідливі запити. Інтеграція WAF в архітектуру безпеки забезпечує додаткову лінію захисту, особливо в динамічних онлайн-середовищах.

1.4 Програмне забезепечення для пошуку вразливостей

Вибираючи інструмент для сканування веб-вразливостей, дуже важливо ретельно оцінити ваші конкретні потреби, враховуючи такі фактори як складність веб-додатків, бажана глибина сканування та бюджет.

Різні інструменти служать різним цілям, тому обирати варто дуже ретельно. Деякі інструменти спеціалізуються на автоматизованому скануванні на наявність поширених вразливостей, тоді як інші пропонують більш комплексні рішення, що включають можливості ручного тестування. Краще вибрати інструмент, який підтримує і точно виявляє вразливості у веб-технологіях, з які застосовуються в конкретному продукті. Крім того, дуже важливими є можливості звітності інструменту. Чіткі та дієві звіти є вкрай корисними для ефективного виправлення ситуації. Тому дуже важливо вибрати інструмент, який надає чіткі та вичерпні звіти, беручи до уваги бюджетні обмеження.

Існують безкоштовні програми з відкритим вихідним кодом, такі як OWASP ZAP та OpenVAS, які можуть бути особливо вигідними для

організацій, що заощаджують, оскільки вони пропонують потужні можливості сканування, не вимагаючи при цьому ліцензійних платежів. Важливо бути в курсі останніх подій у сфері кібербезпеки. Варто шукати інструмент, який регулярно отримує оновлення від постачальника або спільноти для розширення бази даних вразливостей. Це допоможе гарантувати, що сканер зможе виявляти найновіші загрози та вразливості.

Інтеграція з іншими інструментами безпеки, такими як системи управління інформацією та подіями безпеки (SIEM) або платформи відстеження проблем, може значно спростити загальний процес забезпечення безпеки. Бажано обирати інструменти, які виявляють вразливості, а також допомагають визначати пріоритети та керувати зусиллями з їх усунення.

Регулярне оновлення та вдосконалення інструментів сканування має суттєве значення для того, щоб йти в ногу зі змінами у веб-додатках. Адаптація конфігурацій до мінливого ландшафту загроз і конкретних потреб тієї чи іншої організації має далеко не останній пріоритет. Регулярний перегляд стратегії сканування гарантує ефективність інструментів у виявленні та усуненні потенційних вразливостей.

Одним з таких інструментів може бути BurpSuite. Burp Suite, створений PortSwigger, - це набір інструментів, спеціально розроблених для тестування безпеки веб-додатків. Його особливістю є потужний сканер, який автоматизує процес виявлення та оцінки вразливостей безпеки у веб-додатках. Його універсальність робить його популярним серед фахівців з кібербезпеки, оскільки він підходить як для ручного, так і для автоматизованого тестування.

Burp Suite має потужний автоматичний сканер, який знаходить найпоширеніші вразливості веб-додатків, включаючи SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші поширені вразливості. Це дозволяє швидко та ефективно виявляти можливі ризики для безпеки. Тим не менш, автоматизовані сканери мають обмеження, що відрізняє Burp Suite від інших продуктів завдяки можливостям ручного тестування.

Аналітики безпеки можуть взаємодіяти з веб-додатками, досліджувати їхні тонкощі та виявляти непомітні вразливості, які можуть вислизнути від автоматизованих інструментів. Це дуже важливо для виявлення вразливостей, які потребують глибокого розуміння контексту веб-додатку. Крім того, Burp Suite дозволяє фахівцям з безпеки відтворювати різні сценарії атак і оцінювати реакцію веб-додатків на потенційні загрози. Це допомагає розпізнавати вразливості та допомагає розробникам зменшити потенційні ризики до того, як ними скористаються зловмисники.

Однією з основних переваг Burp Suite є його простий у використанні інтерфейс, який підходить як для початківців, так і для досвідчених фахівців з безпеки. Крім того, інструмент забезпечує безперешкодну співпрацю в командах безпеки, сприяючи обміну ідеями під час тестування.

Веб-сайт PortSwigger слугує офіційною платформою Burp Suite, надаючи користувачам необхідну документацію, оновлення та ресурси. Оскільки ризики кібербезпеки продовжують розвиватися, PortSwigger постійно оновлює і вдосконалює Burp Suite, щоб відповідати новим викликам безпеки і підтримувати новітні веб-технології.

Іншим хорошим сканером для веб-додатків може бути OWASP ZAP, також відомий як Open Web Application Security Project ZAP.

OWASP ZAP призначений для виявлення вразливостей безпеки у веб-додатках і пропонує широкий спектр функцій, які задовольняють користувачів з різним рівнем підготовки. Як інструмент з відкритим вихідним кодом, ZAP отримує постійні внески та вдосконалення від глобальної спільноти фахівців з безпеки, пов'язаних з OWASP. Це гарантує, що інструмент залишається актуальним у середовищі кібербезпеки, яке постійно змінюється.

Автоматичні сканери ZAP є однією з його ключових переваг, оскільки вони можуть виявляти такі поширені вразливості, як SQL-ін'єкції та міжсайтовий скриптинг (XSS). Цей інструмент суттєво допомагає

користувачам на ранніх стадіях тестування безпеки, дозволяючи їм ефективно розпізнавати та ранжувати потенційні загрози.

Адаптивність ZAP є ще однією важливою характеристикою, оскільки він пропонує ресурси, придатні як для новачків, так і для експертів. Його інтуїтивно зрозумілий інтерфейс полегшує тестування безпеки веб-додатків для початківців, тоді як розширені функціональні можливості дають змогу досвідченим експертам з легкістю проводити комплексні оцінки.

ZAP вирізняється своїм інтерактивним методом тестування безпеки, який дозволяє користувачам активно взаємодіяти з веб-додатками та імітувати реальні сценарії атак. Це дозволяє їм виявляти тонкі вразливості, які автоматизовані сканери можуть не помітити, забезпечуючи ретельне покриття безпеки.

Користувачі можуть легко отримати доступ до найновіших версій OWASP ZAP через платформу, що надає їм найрелевантніші поради та способи вирішення проблем. Крім того, доступні на сайті освітні ресурси дають можливість користувачам поглибити свої знання про безпеку веб-додатків і отримати цінну інформацію про ефективне використання OWASP ZAP в реальних ситуаціях. Спільнота OWASP ZAP сприяє створенню інклюзивного середовища, де користувачі можуть обмінюватися досвідом, шукати рекомендації та вирішувати проблеми. Мережа експертів співпрацює через форуми та інші канали зв'язку, прагнучи вдосконалити та просувати OWASP ZAP вперед.

2 МЕТОДИ ЗАХИСТУ ВІД ВЕБ ВРАЗЛИВОСТЕЙ

2.1 Методи захисту від XSS вразливостей

Як було описано раніше, існує вичерпний перелік стратегій та методів, які дозволять уникнути виникнення вразливостей такого роду в продукті. До них належать: валідація вхідних даних, екранування даних, використання CSP заголовку в запитах, використання HTTPOnly куків та оновлення бібліотек що використовуються.

2.1.1 Валідація вхідних даних

Валідація є процесом перевірки введених користувачем даних на відповідність певним критеріям. Вона допомагає переконатися, що введені дані не містять шкідливого коду, який може бути виконаний у браузері користувача.

Ескейпінг спеціальних символів є одним з методів валідації введених даних для запобігання XSS-атак. Його основна ідея полягає в тому, що спеціальні символи, які можуть використовуватись для ін'єкції шкідливого коду, заміняються спеціальними послідовностями, які не інтерпретуються браузером як код.

Нижче наведені приклади деяких спеціальних символів та їх ескейпінгу:

- < (менше) і > (більше):

Оригінальний ввід: <script>alert('XSS')</script>

Ескейпований ввід: <script>alert('XSS')</script>

- & (амперсанд):

Оригінальний ввід: user&admin

Ескейпований ввід: user&admin

- " (подвійні лапки):

Оригінальний ввід: <img src="xss.jpg">

Ескейпований ввід: <img src="xss.jpg">

- ' (одинарні лапки):

Оригінальний ввід: alert('XSS')

Ескейпований ввід: alert('XSS')Приклад коду на мові PHP для

екранування спеціальних символів:

$userInput = "<script>alert('XSS');</script>";

$escapedInput = htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');

echo $escapedInput; // Виведе <script>alert('XSS');</script>

Заміна спеціальних символів на ескейп-послідовності гарантує, що браузер буде сприймати їх як звичайний текст, а не як код, що має бути виконаний. Це дозволяє запобігти XSS-атакам, оскільки шкідливий код буде відображатись як текстова інформація на сторінці, а не виконуватись.

Варто врахувати, що ескейпінг спеціальних символів не розв'язує всіх проблем, пов'язаних з XSS-атаками. Він може бути використаний як один з етапів захисту, тож варто поєднувати його з іншими методами, такими як валідація формату, білий список тегів і атрибутів, а також фільтрація небезпечних кодів, для досягнення комплексного захисту від XSS-атак.

Інший метод виконує валідацію формату саме вхідних даних. Цей підхід полягає у перевірці введених користувачем даних на відповідність певному формату, що дозволяє уникнути введення потенційно шкідливого коду.

- Перевірка наявності HTML-тегів:

Важливо перевірити, чи містяться введені дані HTML-теги. Для цього можна використовувати регулярні вирази або спеціальні функції для перевірки наявності < та >, які вказують на можливість введення HTML-коду.

Приклад коду на мові JavaScript для перевірки наявності HTML-тегів:

function isHtmlTagsPresent(input) {

var htmlTagsRegex = /<[^>]+>/g;

```
return htmlTagsRegex.test(input);
}

// Приклад використання
var userInput = "<script>alert('XSS');</script>";
var containsHtmlTags = isHtmlTagsPresent(userInput);
console.log(containsHtmlTags); // Виведе true
```

- Перевірка формату даних:

Деякі типи даних мають визначений формат, який можна перевірити для запобігання введенню шкідливого коду. Наприклад, для електронних адрес можна використовувати регулярні вирази, щоб перевірити, чи має введена адреса правильний формат.

Приклад коду на мові JavaScript для перевірки формату електронної адреси:

```
// Оголошення функції
function isValidEmail(email) {
// Оголошення змінної, значенням якої є регулярний вираз, який буде перевіряти формат введених даних
 var emailRegex = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
// Повернення результатів перевірки вхідних даних
return emailRegex.test(email);
}

// Приклад використання
// Оголошення змінної, значення якої потрібно перевірити
var userEmail = "example@example.com";
// Записування в змінну значення true або false в залежності від результатів функції
var isValid = isValidEmail(userEmail);
```

```
// Вивід результатів виконання функції
console.log(isValid); // Виведе true
```

Важливо враховувати особливості конкретної мови програмування або фреймворка, з яким ви працюєте, та використовувати належні методи валідації для конкретних типів даних, що вводяться користувачами.

Настуний метод валідації даних який ми розглянемо називається “білий список” (“Whitelist”). Цей підхід полягає в обмеженні типів HTML-тегів і атрибутів, які можуть бути використані у введених даних, дозволяючи лише допустимі елементи та їх атрибути. Використання білого списку забезпечує контроль над тим, які елементи і атрибути можуть бути відображені на сторінці, запобігаючи виконанню шкідливого скрипту.

У процесі валідації можна створити список допустимих HTML-тегів, які можуть бути використані в текстових полях або коментарях. Наприклад, при обробці коментарів можуть бути дозволені теги <p>, <a>, , , а інші теги, такі як <script> або <iframe>, будуть відкинуті.

Крім HTML-тегів, можна встановити білий список для атрибутів і їх значень. Наприклад, при обробці введених URL-адрес можна дозволити лише http:// або https:// як схему, виключаючи інші протоколи, які можуть бути потенційно шкідливими.

В залежності від потреб, замість відкидання недопустимих елементів можна також використовувати метод обрізання. У цьому випадку, якщо зустрічається недопустимий елемент або атрибут, він буде видалений або замінений на безпечний еквівалент. Наприклад, якщо введений текст містить <script>, його можна просто видалити або замінити на безпечний текст.

Використання білого списку дозволяє точно контролювати те, що може бути відображено на сторінці, і уникнути потенційно небезпечного виконання шкідливого коду. Цей метод вимагає заздалегідь визначити дозволені HTML-теги, атрибути та їх значення, що може зайняти деяний час, але забезпечує більш високий рівень безпеки для веб-застосунків.

2.1.2 Екранування даних

Цей підхід передбачає використання методів, які обробляють вивід таким чином, щоб запобігти виконанню шкідливого коду в браузері користувача. Ілюстрацією функції безпечного виводу є `textContent` в контексті рендерингу DOM-елементів, яка пропонує безпечний вивід текстового вмісту без обробки його як HTML, що дозволяє уникнути ризиків XSS-уразливостей.

```
// Приклад використання textContent для безпечного виводу
const userInput = "<script>alert('XSS Attack!');</script>";
const outputElement = document.getElementById("output");
// Використання textContent для безпечного виводу
outputElement.textContent = userInput;
// Результат: &lt;script&gt;alert('XSS Attack!');&lt;/script&gt;
```

Крім того, функції безпечного виводу виконують екранування спеціальних символів, які в іншому випадку можуть спровокувати XSS-атаки. Наприклад, використовуючи функцію `innerText`, ви можете уникнути неправильної інтерпретації символів. Це особливо важливо, коли вихідні дані містять спеціальні HTML-символи, які хакери можуть використовувати для проведення XSS-атак.

```
// Приклад використання innerText для безпечного виводу
const userInput = "<script>alert('XSS Attack!');</script>";
const outputElement = document.getElementById("output");
// Використання innerText для безпечного виводу
outputElement.innerText = userInput;
// Результат: &lt;script&gt;alert('XSS Attack!');&lt;/script&gt;
```

Додатковою перевагою безпечних функцій є те, що вони зручно поєднуються із захистом і перевіркою даних, утворюючи всеосяжний механізм безпеки для веб-додатків. Розробники програмного забезпечення можуть створювати веб-додатки, стійкі до XSS, використовуючи функції безпечного виводу.

Приклад використання безпечної функції виводу в поєднанні з іншими заходами безпеки виглядатиме наступним чином:

```
const userInput = "<script>alert('XSS Attack!');</script>";
const sanitizedOutput = sanitizeAndOutput(userInput);
```

Для початку потрібно оголосити функцію, в параметрах передати дані, які потрібно екранувати. Після цього за допомогою JavaScript функції getElementById вбудованої в об’єкт document, оголосити об’єкт елементу на сторінці, куди плануємо вивести вихідне значення. Зрештою, змінюємо значення змінної textContent в об’єкті елементу на заескейплені дані.

```
function sanitizeAndOutput(input) {
  // Логіка екранування та інших заходів безпеки
  const sanitizedInput = performSanitization(input);
  const outputElement = document.getElementById("output");
  outputElement.textContent = sanitizedInput;
  return sanitizedInput;
}
```

Метод, який екранує дані, використовує метод replace прототипу String на даних, які потрібно заескейпити, і замінює спеціальні символи на коди HTML розмітки.

```
function performSanitization(input) {
```

```
  // Логіка екранування та інших заходів безпеки
  // Приклад: заміна символів '<' та '>' на '&lt;' та '&gt;'
  return input.replace(/</g, "&lt;").replace(/>/g, "&gt;");
}
```

2.1.3 Використання CSP заголовку

HTTP був розроблений для обміну базовим текстом і зображеннями. Проте, оскільки веб-додатки ставали дедалі складнішими, виникла потреба в удосконаленні протоколу. Включення заголовків HTTP в HTTP/1.0 стало значним покращенням, оскільки дозволило передавати додаткову інформацію разом із запитами та відповідями.

HTTP-заголовки є ключовою частиною протоколу, яка ідентифікує клієнтів і сервери, визначає мову і тип контенту, керує сеансами і впроваджує правила кешування. Вони також роблять значний внесок у безпеку та автентифікацію веб-додатків, встановлюючи правила обміну конфіденційними даними.

Заголовки також відповідають за оптимізацію завантаження веб-сторінок і зменшення навантаження на сервер за рахунок керування кешуванням ресурсів. Крім того, вони допомагають оптимізувати передачу даних, визначаючи кодування і розмір пакетів. Крім того, HTTP-заголовки встановлюють правила та дозволи для міждоменних запитів, що особливо важливо для сучасних веб-додатків та API, які потребують безпечного обміну даними між різними джерелами.

Для захисту від XSS потрібно використовувати заголовок CSP. Замість того, щоб покладатися виключно на сервер для санітарної обробки та перевірки користувацького контенту, CSP зміщує акцент на контроль джерел, з яких контент може бути завантажений на веб-сайт, таким чином підвищуючи безпеку.

CSP дозволяє адміністраторам веб-сайтів встановлювати правила для браузера щодо завантаження вмісту. Директива script-src відіграє важливу

роль у цьому процесі, вказуючи авторизовані джерела для виконання скриптів. Явно визначаючи довірені джерела, CSP ефективно зупиняє виконання скриптів з несанкціонованих або шкідливих джерел.

Зрештою, CSP надає атрибут "nonce", що дозволяє розробникам створювати окремі криптографічні nonce для кожного скрипта. Ці нонси можуть бути включені безпосередньо в теги скриптів на стороні сервера. Після цього браузер підтверджує, чи відповідає nonce в тезі скрипта nonce, зазначеному в заголовку CSP, перед тим, як дозволити виконання скрипта. Такий незалежний від nonce підхід додає додатковий рівень безпеки, гарантуючи, що виконуються лише скрипти з дійсними nonce.

Директива CSP "strict-dynamic" дозволяє включати скрипти з довірених джерел, навіть якщо вони не вказані в заголовку CSP. Це необхідно для того, щоб дозволити завантажувати динамічний вміст, зберігаючи при цьому безпеку. Однак вбудовані скрипти всеодно вимагатимуть відповідного nonce або хешу для перевірки.

Варто зазначити, що CSP допомагає знизити ризик XSS-атак, обмежуючи використання вбудованих скриптів, обробників подій і стилів. Термін "unsafe-inline" використовується для блокування вбудованих скриптів і стилів, що спонукає розробників переміщати свої скрипти і стилі в зовнішні файли. Цей метод обмежує потенційні шляхи для зловмисників, що планують впровадити шкідливий код, таким чином зменшуючи ризик атак.

Також, CSP забезпечує надійний механізм звітності за допомогою директиви "report-uri" у разі порушення політики. Коли виникає порушення, браузер надсилає звіт на вказаний URI, що дозволяє адміністраторам контролювати і швидко вирішувати можливі проблеми безпеки.

2.1.4 Використання HTTPOnly cookie

Концепція файлів cookie виникла через необхідність зберігати невеликі біти даних на пристрої користувача. Ці файли, які зазвичай називають "cookies", були створені для того, щоб покращити досвід перегляду веб-

сторінок і дозволити веб-сайтам ідентифікувати та нагадувати користувачам про себе. Спочатку файли cookie були створені компанією Netscape у 1994 році і призначалися для збереження інформації про стан користувача під час перегляду ним різних сторінок веб-сайту.

Файли cookie слугують декільком цілям і були розроблені для вирішення певних проблем, пов'язаних з веб-взаємодією. Однією з головних причин їх створення було полегшення управління сеансами, що дозволило б веб-сайтам розпізнавати користувачів і підтримувати їхній статус входу в систему. Це було особливо важливо для онлайн-сервісів, які потребують автентифікації користувачів, таких як платформи електронної пошти та ранні веб-сайти електронної комерції.

З розвитком онлайн-середовища файли cookie знайшли додаткові можливості використання. Вони відіграли важливу роль у персоналізації користувацького досвіду завдяки збереженню уподобань та налаштувань. Наприклад, веб-сайти почали використовувати файли cookie для збереження мовних уподобань, вибору тем та інших користувацьких конфігурацій, що призвело до створення більш персоналізованого та зручного інтерфейсу.

Створення файлів cookie мало основну мету - полегшити таргетовану рекламу, відстежуючи поведінку користувачів в Інтернеті. Такий моніторинг дозволив би рекламодавцям показувати персоналізовану та релевантну рекламу. Однак ця практика викликала занепокоєння щодо конфіденційності, що призвело до створення нормативних актів та ініціатив, спрямованих на захист даних користувачів.

З часом файли cookie стали більш обширними і почали включати в себе різні типи, кожен з яких виконує певну функцію і робить свій внесок у загальну функціональність інтернету. Файли cookie стали вирішальним аспектом оптимізації користувацького досвіду. Тим не менш, виникли дебати про конфіденційність, що призвели до таких досягнень, як атрибути файлів cookie SameSite і поява альтернатив, таких як "відбитки пальців" браузерів.

Атрибут HTTPOnly використовується для HTTP-файлів cookie, щоб обмежити доступ до файлів cookie з JavaScript-скриптів, як захисний захід від XSS (міжсайтового скриптингу). Коли сервер встановлює HTTPOnly для файлу cookie, це означає, що до нього не можна отримати доступ через об'єкт document.cookie в JavaScript. HTTPOnly файли cookie використовуються виключно для взаємодії з сервером при доступі до ресурсів. Встановлення HTTPOnly для файлів cookie має на меті запобігти доступу зловмисників до них у випадку, якщо XSS-уразливість вже дозволила впровадити шкідливий код на сторінку. Це мінімізує ризик втрати конфіденційної інформації, яка може зберігатися в файлах cookie.

Важливо визнати, що HTTPOnly не є комплексним рішенням і не може замінити інші заходи безпеки, включаючи точну обробку вхідних даних та інші засоби контролю безпеки. Крім того, слід вжити негайних заходів для усунення XSS-уразливості, оскільки HTTPOnly лише пом'якшує вплив такого недоліку, а не усуває його повністю.

2.2 Методи захисту від CSRF вразливостей

Для комплексного захисту від CSRF вразливостей, необхідно створити комплексну систему безпеки, яка передбачає постійний моніторинг та адаптивні механізми реагування. Ця система безпеки повинна включати в себе: використання CSRF токенів, SameSite Cookies, Double Submit Cookies та HTTP Referer Header.

2.2.1 CSRF токени

Унікальний токен CSRF створюється кожного разу, коли користувач входить в систему або взаємодіє з нею. Цей маркер, який може бути випадковим числом або складним хешем, тісно пов'язаний з конкретним сеансом користувача. Він включається в запит щоразу, коли користувач взаємодіє з сервером, наприклад, надсилає форму або виконує AJAX-запит.

При використанні техніки AJAX унікальні токени CSRF включаються в HTML-форми, можуть бути частиною прихованого поля форми або міститися в заголовку запиту. Зловмисники не можуть просто надіслати фальшивий запит своїм потенційним жертвам, оскільки їм потрібен доступ до справжнього токену, який є унікальним для кожного користувача.

Коли сервер отримує запит, він перевіряє, чи легітимний токен CSRF і чи належить він конкретному сеансу користувача. Якщо токен недійсний або не відповідає сесії, сервер відхиляє запит, підозрюючи CSRF-атаку.

Дійсний токен для конкретного користувача повинен бути доступний зловмисникам, які намагаються здійснити CSRF-атаку. Однак, оскільки токени є унікальними і залежать від сеансу, вони не можуть з легкістю заволодіти відповідним токеном, що ускладнює проведення атаки.

Крім того токен можна вбудувати в метатеги, які будуть автоматично додавати токен при запиті до сервера, але викрасти його за допомогою XSS, наприклад, вже не вдасться, оскільки токен не буде вбудований в DOM. Зробити це можна наступним чином:

```
<head>
   <meta name="csrf-token" content="your_csrf_token_here">
</head>
```

2.2.2 Використання атрибуту SameSite cookies

Встановлення атрибуту SameSite в значення "Strict" або "Lax" підвищує рівень безпеки файлів cookie, зменшуючи їх вразливість до атак. Атрибут SameSite має всього два основних значення: "Strict" і "Lax".

Коли атрибут SameSite встановлено в значення "Strict", файл cookie буде включений тільки в запити, відправлені зі сторінки, яку користувач відкрив у поточному вікні браузера. Цей підхід застосовується для того, щоб ускладнити проведення CSRF-атак, оскільки файл cookie не буде додаватися до запитів, що надсилаються з інших джерел.

У режимі Lax файл cookie буде додано до зовнішніх запитів, що надсилаються через зображення або скрипти. Однак, він не буде включений в запити, зроблені при переході користувача з інших джерел. Таким чином досягається баланс між безпекою і зручністю, дозволяючи використовувати файли cookie в певних сценаріях і обмежуючи їх використання в інших.

Використовуючи атрибут SameSite, розробники матимуть точний контроль над тим, як файли cookie включаються в різні типи запитів, і можуть вибирати рівень захисту CSRF, який найкраще відповідає конкретним потребам того чи іншого веб-додатку.

2.2.3 Метод Double Submit Cookies

Цей метод заснований на використанні унікального маркера CSRF, який одночасно зберігається в файлі cookie і включається в кожен запит, що надсилається або вбудовується в форму. Мета цього методу - забезпечити механізм подвійного представлення, за допомогою якого сервер може порівнювати значення маркера CSRF, що зберігається в файлі cookie, зі значенням, що міститься в полі форми або додається до заголовка запиту.

При реалізації цього методу для кожної сторінки або дії, що виконується, генерується унікальний маркер CSRF. Токен зберігається в файлі cookie в браузері клієнта і вставляється в форму або додається до інших параметрів запиту. Коли запит надсилається на сервер, значення токена порівнюється зі значенням, що зберігається в файлі cookie. На основі результатів порівняння сервер визначає, чи є запит легальним.

Механізм подвійного представлення ускладнює проведення успішної CSRF-атаки, оскільки для її проведення зловмисник повинен не тільки знати значення CSRF-токена, але й успішно взаємодіяти з файлом cookie на стороні потенційної жертви. Це значно підвищує рівень безпеки веб-додатків і запобігає несанкціонованим операціям з боку автентифікованих користувачів.

2.2.4 Використання POST запитів

Ефективним заходом для захисту від уразливостей CSRF є обов’язкове використання методу POST замість GET для чутливих дій, таких як зміна статусу облікового запису, видалення ресурсів або проведення фінансових операцій. Цей підхід використовує властиві характеристики браузерів, які зазвичай не дозволяють веб-сторінкам автоматично генерувати запити POST за допомогою вбудованих тегів, зокрема тегів <img> і <script>. Ці теги зазвичай використовуються в атаках CSRF шляхом ініціювання запитів GET.

Примусове використання методу POST для важливих дій створює додатковий рівень безпеки. Оскільки браузери не обробляють запити POST так само, як теги <img> і <script>, цей запобіжний захід має вирішальне значення для запобігання автоматичним атакам, які використовують альтернативні теги для запуску небажаних дій на вразливих сторінках. Ця стратегія ефективно зменшує ризик CSRF-атак, які провокують теги, що імітують запити GET.

2.2.3 Автоматизоване тестування, як метод протидії XSS і CSRF

Загальний підхід такого методу полягає в тому, щоб використовувати автоматизовані тести як частину повного стратегічного плану безпеки, який також може включати ручне тестування, аудит коду та навчання команди. Щоб детальніше дослідити такий спосіб захисту, потрібно спочатку зрозуміти, які можливості автоматизації доступні нам для вирішення поставлених завдань.

2.2.4 Автоматизоване тестування CSRF вразливостей

Першим кроком для автоматизації тестування CSRF вразливостей повинна бути розробка тестових сценаріїв. Наприклад зміна паролю, видалення облікового запису і т.д. При розробці тестових сценаріїв необхідно переконатися, що вони точно відтворюють реальну взаємодію користувача з

додатком. Важливо включити широкий спектр сценаріїв, щоб охопити різні аспекти функціональності додатку, де можуть виникнути вразливості CSRF.

Після цього потрібно обрати інструмент, яким буде проводитися тестування. Для перевірки на вразливості CSRF добре підходить BurpSuite, який розглянутий в пункті 1.4. Для початку потрібно придбати версію Professional, і налаштувати доступ до тестового середовища.

Перейшовши в інформаційну панель потрібно вибрати "Нове сканування", щоб відкрити діалогове вікно "Запуск сканування", в якому буде можливість налаштувати параметри Burp Scanner. У відповідному полі "URL-адреси для сканування" потрібно ввести URL-адресу веб-сайту, який потрібно протестувати. Для ознайомлення з базовим функціоналом, можна залишити значення за замовчуванням для всіх інших параметрів. Використовуючи Burp Scanner, необхідно пам'ятати про потенційний негативний вплив на певні програми. Щоб уникнути проблем, варто обмежити використання Burp Scanner невиробничими системами, не ознайомившись з його налаштуваннями та функціоналом. За відсутності дозволу від власника, рекомендовано утримайтися від сканування сторонніх веб-сайтів.

Щоб точно налаштувати різні аспекти поведінки Burp Scanner для різних сценаріїв і цільових сайтів, потрібно вибрати опцію "Налаштування сканування". Після цього обрати "Використовувати попередньо встановлений режим сканування", а потім натиснути "Полегшений". Цей режим забезпечує швидкий, але всебічний огляд об'єкта. Тривалість сканування в цьому режимі не може перевищувати 15 хвилин.

При натисканні "ОК", Burp Scanner почне сканування з URL-адреси, введеної на попередньому кроці. Варто звернути увагу, що на інформаційній панелі з'явиться нове завдання, яке вказуватиме на це сканування. Це завдання буде містити ключову інформацію, включаючи поточну фазу сканування і кількість відправлених запитів. Щоб переглянути новий запис для вашої URL-адреси, потрібно перейти на вкладку "Ціль" > "Карта сайту" і

розгорнути цей розділ, щоб отримати доступ до всього виявленого вмісту. Після нетривалого очікування, можна буде зауважити, що карта оновлюється в режимі реального часу, дозволяючи відстежувати хід сканування на інформаційній панелі.

Після завершення процесу сканування Burp Scanner почне аудит на наявність вразливостей. Будь-які виявлені проблеми будуть відображені на Панелі активності проблем, розташованій на вкладці Панелі моніторингу. Вибравши проблему, можна отримати доступ до вкладки Консультації, яка містить важливу інформацію, що стосується типу проблеми, з вичерпним описом і рекомендованими заходами щодо її усунення.

Звісно, у цей список потраплять CSRF вразливості, проте для більш поглибленого тестування таких проблем варто обдумати використання додаткових розширень для Burp Suite.

2.3 Автоматизоване тестування XSS вразливостей

Механізм автоматизованого тестування XSS вразливостей можна реалізувати багатьма способами, наприклад за допомогою PHPUnit і Selenoid, який є зручним докер-контейнером для Selenium. Для початку, варто розглянути ці два інструменти.

PHPUnit - це фреймворк для тестування, який широко використовується і має високу надійність для мови програмування PHP. Він спеціально розроблений для спрощення процесу розробки модульних тестів, які є ключовими елементами в розробці програмного забезпечення. Створений Себастьяном Бергманом, PHPUnit дозволяє розробникам створювати автоматизовані тести, які забезпечують точність окремих блоків або компонентів у кодовій базі PHP. Ці тести необхідні для підтримки цілісності коду, виявлення помилок на ранніх стадіях циклу розробки та запобігання ненавмисному регресу при внесенні змін до коду.

PHPUnit пропонує підтримку класу PHPUnit_Framework_TestCase, який забезпечує структурований підхід до визначення тестових кейсів, а також надає повний набір методів тверджень для перевірки очікуваних результатів. На додаток до базового модульного тестування, PHPUnit також підтримує інші типи тестування, включаючи інтеграційне тестування і функціональне тестування, які сприяють ретельному і всебічному тестуванню. Як невід'ємна частина сучасної PHP-розробки, PHPUnit легко інтегрується з інструментами безперервної інтеграції, системами контролю версій та іншими компонентами робочого процесу розробки.

Для інсталяції цього фреймворку можна використати пакетний менеджер Composer. Composer - це надійний менеджер залежностей для PHP, який має на меті спростити керування бібліотеками та пакунками у PHP-проектах. Будучи важливим інструментом у сучасній розробці PHP, Composer дозволяє розробникам декларувати бібліотеки, на які покладаються їхні проекти, а також без особливих зусиль керувати встановленням та оновленням цих залежностей. Використовуючи простий конфігураційний файл JSON під назвою "composer.json", розробники можуть вказати вимоги до проекту, такі як версія PHP, розширення та зовнішні пакунки. Composer використовує централізований репозиторій, відомий як Packagist, який містить велику колекцію бібліотек PHP, що можуть бути використані в різних проектах.

Ключова перевага Composer полягає в його здатності ефективно вирішувати і отримувати залежності, забезпечуючи точне встановлення необхідних версій. Крім того, він спрощує автозавантаження, дозволяючи безперешкодно включати класи і функції з встановлених бібліотек. Composer став незамінним інструментом в екосистемі PHP, просуваючи стандартизований та ефективний підхід до управління залежностями, тим самим покращуючи модульність коду, співпрацю та загальний робочий процес розробки. Його широке розповсюдження підкреслює його значення як фундаментального компонента у сучасному середовищі розробки PHP.

Щоб інтегрувати Selenium, спочатку потрібно встановити сервер Selenium і відповідний WebDriver браузера. Можна використовувати окремий сервер Selenium або такі інструменти, як ChromeDriver чи GeckoDriver. Після завантаження та налаштування потрібно переконатися, що сервер Selenium або WebDriver доступні з тестового середовища PHPUnit.

Тестовий файл потрібно розширити класом PHPUnit_Extensions_Selenium2TestCase у тестовому класі PHPUnit, щоб використовувати функції Selenium. Сконфігурувати ініціалізацію і закриття сесії Selenium, можна визначивши необхідні методи, такі як setUp() і tearDown(). Використання анотацій, такі як @group для категоризації тестів і @dataProvider для параметризованого тестування.

У методі setUp() налаштовуються браузер, можливості браузера, хост і порт для Selenium. Задається базова URL-адреса для веб-додатку і використовуються команди Selenium для взаємодії з браузером під час тестування.

Для написання методів тестування для виконання дій у веб-додатку, таких як натискання кнопок, заповнення форм і перевірки очікуваних результатів використовуються команди Selenium WebDriver, такі як clickOnElement(), type() та твердження для перевірки результатів, тобто асерти.

WebDriver служить мостом між мовою програмування і веб-браузерами, дозволяючи розробникам і тестувальникам створювати сценарії, які автоматизують тестування веб-додатків на різних браузерах і платформах. Використовуючи специфічний для браузера драйвер, Selenium WebDriver дозволяє емуляцію взаємодії користувача з веб-елементами, такими як натискання кнопок, заповнення форм, навігація сторінками та перевірка очікуваної поведінки.

Його сумісність з різними браузерами гарантує, що один і той самий набір тестів може бути безперешкодно виконаний, тим самим покращуючи покриття та надійність тестів. Крім того, Selenium WebDriver відіграє

важливу роль у підтримці безперервної інтеграції та безперервної доставки, тим самим підвищуючи ефективність та якість процесів розробки програмного забезпечення.

Завдяки своїй універсальності, розширюваності та потужній підтримці спільноти, Selenium WebDriver став незамінним інструментом для професіоналів, які займаються тестуванням веб-додатків та автоматизацією тестування.

Для зручності тестування можна використовувати Selenoid. Selenoid - це високоефективна контейнерна селенова сітка з відкритим вихідним кодом. Її мета - спростити та покращити процес тестування веб-додатків.

На відміну від традиційних Selenium-сіток, Selenoid використовує контейнери Docker, щоб запропонувати легке і масштабоване рішення для запуску браузерних тестів в ізольованих середовищах. Цей унікальний підхід гарантує, що кожен тест має свій власний виділений браузер і залежності, що допомагає уникнути проблем з розподілом ресурсів і гарантує послідовне і відтворюване середовище тестування. Selenoid сумісний з різними версіями браузерів і може бути легко інтегрований з популярними тестовими фреймворками, такими як TestNG і JUnit. Його модульна архітектура в поєднанні з можливістю динамічного розподілу ресурсів на основі попиту роблять Selenoid потужним інструментом для організацій, які прагнуть оптимізувати свою інфраструктуру веб-тестування. Крім того, Selenoid надає такі функції, як відеозапис, попередній перегляд у браузері в реальному часі та докладні журнали, які значно полегшують всебічний аналіз результатів тестування.

Щоб створити надійну та ефективну мережу Selenium, установка Selenoid за допомогою AeroCube CM включає в себе ряд комплексних кроків. AeroCube CM, інструмент управління конфігурацією, спрощує розгортання та управління Selenoid, реалізацією Selenium grid з відкритим вихідним кодом.

Перед початком процесу інсталяції важливо переконатися, що AeroCube CM належним чином встановлений у системі. Якщо це так, можна переходити до створення конфігураційного файлу, в якому будуть вказані бажані налаштування для Selenoid, такі як версії браузерів, роздільна здатність екрану і мережеві параметри. Після цього, варто перейти до розгортання проєкту за допомогою інтуїтивно зрозумілого інтерфейсу AeroCube CM, інтегрувавши Selenoid в існуючу інфраструктуру. Крім того, слід використовувати можливості моніторингу та масштабування AeroCube CM для ефективного управління ресурсами і адаптації до різних вимог тестування. Після успішного завершення інсталяції рекомендовано провести ретельне тестування, щоб перевірити функціональність і стабільність мережі Selenium на базі Selenoid. Ця комплексна установка, що поєднує сильні сторони AeroCube CM і Selenoid, створює відмовостійке і масштабоване середовище для тестування веб-додатків, підвищуючи загальну ефективність тестування і забезпечуючи безперебійний досвід тестування.

Додатковою корисною опцією є підтримка VNC імеджів браузерів в селеноїді. Щоб увімкнути підтримку VNC, необхідно включити необхідні параметри VNC у конфігурацію, а також завантажити необхідні образи VNC для браузера.

Перевагу надають образам Chrome і Firefox, які попередньо налаштовані з підтримкою VNC. Потрібно використати команди Docker, щоб витягнути потрібні образи з Docker Hub на локальний комп'ютер і переконатися, що версії браузерів відповідають версіям, зазначеним у конфігурації Selenoid.

Після успішного налаштування тестового середовища, можна переходити до написання автоматизовних тестів для перевірки XSS. Для тестування XSS-уразливостей можна створити PHPUnit-тест за допомогою Selenium, розробивши тестовий сценарій, який має на меті впровадити шкідливий вміст у веб-додаток. Важливо відзначити, що проведення таких

тестів повинно здійснюватися відповідально і виключно на системах, де було надано явний дозвіл на тестування.

Припустимо, що є поле форми, яке може бути вразливим до XSS. Тест вводить ретельно розроблене корисне навантаження, щоб перевірити, чи ефективно додаток обробляє та запобігає XSS-атакам. Сам тест виглядає наступним чином:

Для початку потрібно визначити простір імен і включають необхідні класи з PHPUnit і бібліотеки Facebook WebDriver.

```
use PHPUnit\Framework\TestCase;
use Facebook\WebDriver\Remote\RemoteWebDriver;
use Facebook\WebDriver\WebDriverBy;
```

Після цього оголосити клас з іменем XssTest. Цей клас розширює клас TestCase, що надається PHPUnit. Його призначенням є визначення та виконання тестів відповідно до вимог PHPUnit.

```
class XssTest extends TestCase
```

Оголосити змінну в класі у якій будемо зберігати об’єкт веб-драйвера.

```
private $webDriver;
```

Описати метод setUp, що викликається перед кожним тестом. Він ініціалізує WebDriver, створюючи віддалене з'єднання з сервером Selenium, що працює за адресою http://localhost:4444/wd/hub, за допомогою браузера Chrome.

```
protected function setUp(): void
{
   $this->webDriver = RemoteWebDriver::create('http://localhost:4444/wd/hub', ['platform' => 'LINUX', 'browserName' => 'chrome']);
```

```
}
```

І описати сам тесткейс.

```
public function testXssVulnerability()
{
    // Завантажити веб-сторінку
    $this->webDriver->get('http://your-app-url');

    //Визначити форму використовуючи її атрибут name
    $textField = $this->webDriver->findElement(WebDriverBy::name('input_field_name'));
    // Ініціалізуємо скрипт з вразливістю
    $xssPayload = '<script>alert("XSS");</script>';
    // Вводимо підготовлений скрипт в форму
    $textField->sendKeys($xssPayload);
    // Відправляємо ворму натисканням кнопки submit
    $this->webDriver->findElement(WebDriverBy::name('submit_button'))->click();
    // Очікуємо 2 секунди на відповідь сторінка
    sleep(2);
    // Перевіряємо чи скрипт виконався
    $this->assertTrue($this->webDriver->switchTo()->alert()->getText() === 'XSS');
}
```

Після цього описуємо метод tearDown що завершує сесію веб-драйвера.

```
protected function tearDown(): void
{
    $this->webDriver->quit();
}
```

3 РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ ВІД XSS І SCRF ВРАЗЛИВОСТЕЙ

3.1 Налаштування тестового середовища

Тестове середовище буде побудоване на базі фреймворку Express.js, оскільки він дозволяє швидко і гнучко налаштувати веб-сервер. Express.js - це широко використовуваний фреймворк веб-додатків для Node.js, який спрощує процес створення надійних і масштабованих веб-додатків. Він забезпечує мінімалістичну та гнучку структуру, яка дозволяє розробникам легко створювати багатофункціональні API та динамічні веб-додатки.

Виконуючи роль проміжного програмного забезпечення, вона спрощує робочий процес розробки, надаючи основні функції, такі як маршрутизація, підтримка проміжного програмного забезпечення та механізми шаблонів для обробки HTTP-запитів і відповідей. Express.js відомий своїм ненав'язливим дизайном, який заохочує використання сторонніх модулів, сприяючи модульному та настроюваному підходу до створення додатків. Завдяки великій та активній спільноті, обширній документації та великій кількості доступних плагінів, Express.js став наріжним каменем в екосистемі Node.js, надаючи розробникам можливість створювати ефективні та масштабовані серверні додатки для різноманітних випадків використання.

Незалежно від того, чи використовується Express.js у поєднанні з сучасними фронтенд-фреймворками, чи як частина архітектури мікросервісів, він залишається популярним вибором для розробників, які шукають гнучке та ефективне рішення для створення веб-додатків на платформі Node.js.

Для початку процесу створення нового проекту потрібно встановити Node.js та NPM (Node Package Manager). Після того, потрібно перейти до каталогу проекту, відкривши термінал або командний рядок. Потім ініціювати новий проект Node.js, запустивши npm init і слідуючи підказкам, створити файл package.json. Після ініціалізації проекту можна перейти до

встановлення Express.js, виконавши команду npm install express. Ця команда завантажить останню версію Express.js і його залежностей, додавши їх до проекту.

Після завершення встановлення потрібно створити файл точки входу (наприклад, app.js) і почати конфігурувати Express-додаток.

Якщо все зроблено правильно проект буде мати вигляд як на рисунку 3.1.

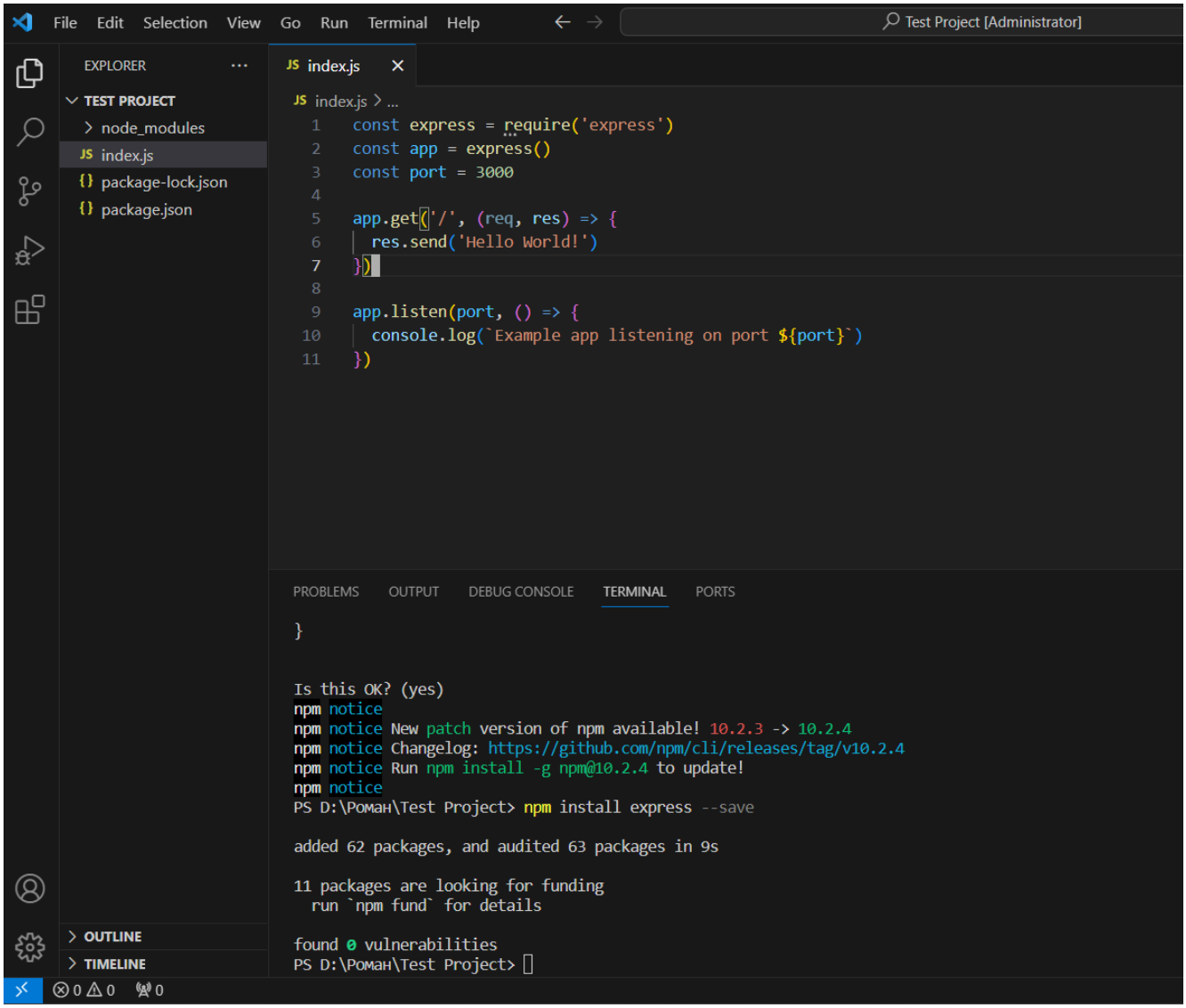

Рисунок 3.1 - Початковий вигляд проекту

У файлі, який є точою входу в наш застосунок, пропишемо стандартний код що ініціалізує і запускає веб-сервер.

3.2 Реалізація методів захисту від XSS вразливостей

Для початку, реалізуємо просту вразливу HTML сторінку з формою, що відправлятиме запит на сервер, який, в свою чергу, буде повертати нам введену стрічку на веб сторінку. Для початку створимо HTML темплейт нашого застосунку:

```
<div>
<input id="XSS_input_field" type="text">
<button id="submit">Send</button>
</div>
<div id="messages"></div>
<script>
</script>
```

Цей код реалізовує лише вигляд тестового додатку в браузері, як зображено на рисунку 3.2. Як можна побачити, реалізовано поле для введення повідомлення та кнопку для його відправки на сервер. Також додано контейнер з ідентифікатором messages, у який будуть виводитися надіслані повідомлення.

Рисунок 3.2 - Вигляд тестової веб-сторінки

Після цього, всередину тегу script, потрібно додати JS код, який буде взаємодіяти з сервером.

Для початку оголосимо змінні, типу const, у яких будемо зберігати сервісні дані, такі як url-адреса веб сервера, шляхи, які доступні нам для взаємодії з ним, та об’єкти дерева DOM.

```
const apiUrl = "http://127.0.0.1:3000";
const XssPost = apiUrl + "/XSS-post";
const XssGet = apiUrl + "/XSS-get";Для початку оголосимо змінні
const messagesView = document.getElementById('messages');
const inputField = document.getElementById('XSS_input_field');
const submitButton = document.getElementById('submit');
```

Після цього, опишемо метод, який при завантаженні сторінки робитиме аснихронний запит до сервера, і відображатиме нам існуючі повідомлення.

```
window.onload = async () => {
    await fetch(XssGet, {
        method: "GET"
    })
    .then( res => res.json())
    .then( res => {
    res.messages.map(message => {
        messagesView.innerHTML += "<p>" + message + "</p>"
    });
});
};
```

Наступним кроком, додамо обробник подій на кнопку надсилання. При кліку будемо додавати до неї атрибут disabled зі значенням true, який зробить кнопку неактивною, щоб унеможливити спам повідомленнями. Після цього згенеруємо параметри url посилання, а саме додамо до них message. У випадку, коли повідомлення пусте, надсилатимемо стандартний текст test. Після отримання позитивної відповіді від сервера, додаємо повідомлення в список відображення на сторінці. Наприкінці, знову робимо кнопку активною, щоб дати змогу надіслати наступне повідомлення.

```
submitButton.addEventListener('click', async () => {
submitButton.setAttribute('disable', true);

let urlParams = new URLSearchParams({
message: inputField.nodeValue || 'test'
});

await fetch(XssPost + "?" + urlParams.toString(), {
method: "GET"
})
.then( res => res.json())
.then( res => messagesView.innerHTML += "<p>" + res.message + "</p>");
submitButton.setAttribute('disable', false);
});
```

На серверній частині, в свою чергу, також, для початку, оголосимо серівсні змінні, у яких будуть знаходитися об’єкти потрібних для роботи додатку бібліотек, порт та шлях розміщення файлу, який слугуватиме імітацією дази даних. Крім того, дозволимо CORS, для локальної розробки.

```
const express = require('express')
```

```
const cors = require('cors')
const fs = require('fs');
const path = require("path");
const app = express()
const port = 3000
const fakeDbFile = path.resolve('./store/XSS-messsages.txt');
app.use(cors())
```

Наступним кроком, ініціалізуємо шлях, за яким буде можливість звернутися до сервера для отримання збережених повідомлень. Для цього ми будемо діставати вміст файлу і розділяти контент по символу \n.

```
app.get('/XSS-get', (req, res) => {
  let buffer = fs.readFileSync(fakeDbFile, (err) => {});
  res.json({messages: buffer.toString().split("\n")})
})
```

Також додамо шлях, що дозволить додати нове повідомлення. При отриманні такого запиту з тіла запиту буде діставатися повідомлення, записуватися в файл, і повертатися в відповіді від сервера.

```
app.get('/XSS-post', (req, res) => {
  let buffer = fs.readFileSync(fakeDbFile, (err) => {});
  fs.writeFileSync(fakeDbFile, buffer.toString() + req.query.message + '\n', (err) => {});
  res.json({message: req.query.message})
})
```

Вкінці файлу ініціалізуємо колбек, який викликатиметься при старті сервера.

```
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Комунікація веб сторінки і сервера налаштована таким чином, що при ініціалізації сторінки на веб-сервер відпраляється запит, який дістає всі збережені повідомлення, які були раніше надіслані, з текстового файлу.

Спробуємо впровадити настуний скрипт на вразливу веб сторінку:

```
<img src="/" onerror="alert(1)">
```

Результат впровадження такого коду можна побачити на рисунку 3.3.

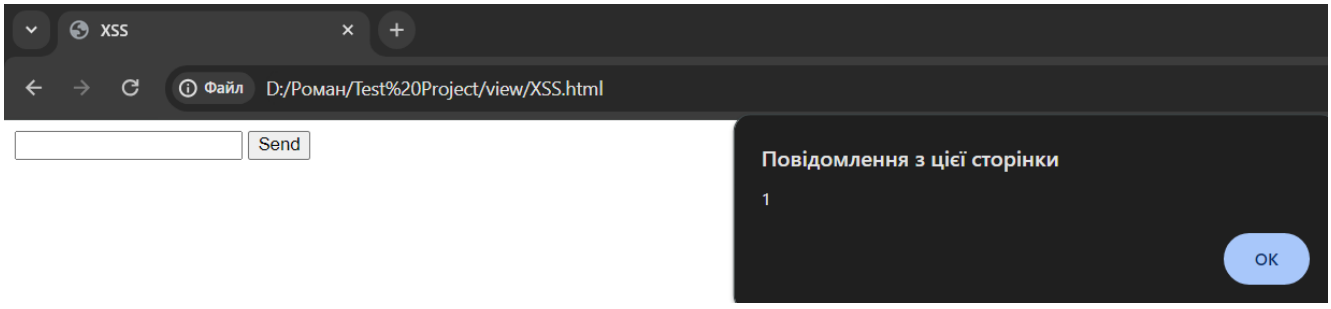

Рисунок 3.3 - Відображення XSS вразливості

Тепер, коли розгорнуте тестове середовище, і підтверджено наявність вразливості в цієї веб-сторінки, можна починати застосовувати методи щодо її запобігання. Перший метод який розглянемо - валідація вхідних даних. Валідація є процесом перевірки введених користувачем даних на відповідність певним критеріям. Вона допомагає переконатися, що введені дані не містять шкідливого коду, який може бути виконаний у браузері користувача.

Зробивши правку в коді, на стороні сервера таким чином, щоб у випадку, коли повідомлення, яке ми надсилаємо, має один із знаків <[^>], повертати користувачу помилку. Правка в коді виглядатиме настуним чином:

```
app.get('/XSS-post', (req, res) => {
  let htmlTagsRegex = /<[^>]+>/g;
  if (htmlTagsRegex.test(req.query.message)) {
    res.statusCode = 400;
    res.send();
  }

  let content = fs.readFileSync(fakeDbFile, {encoding: 'utf8'}, (err) => {});
  fs.writeFileSync(fakeDbFile, content + req.query.message + '\n', (err) =>
{});
  res.json({message: req.query.message})
})
```

Як можна побачити на рисунку 3.4, тепер при запиті до сервера з використанням повідомлення, яке містить заборонені символи повертається 400 (Bad Request) респонс код.

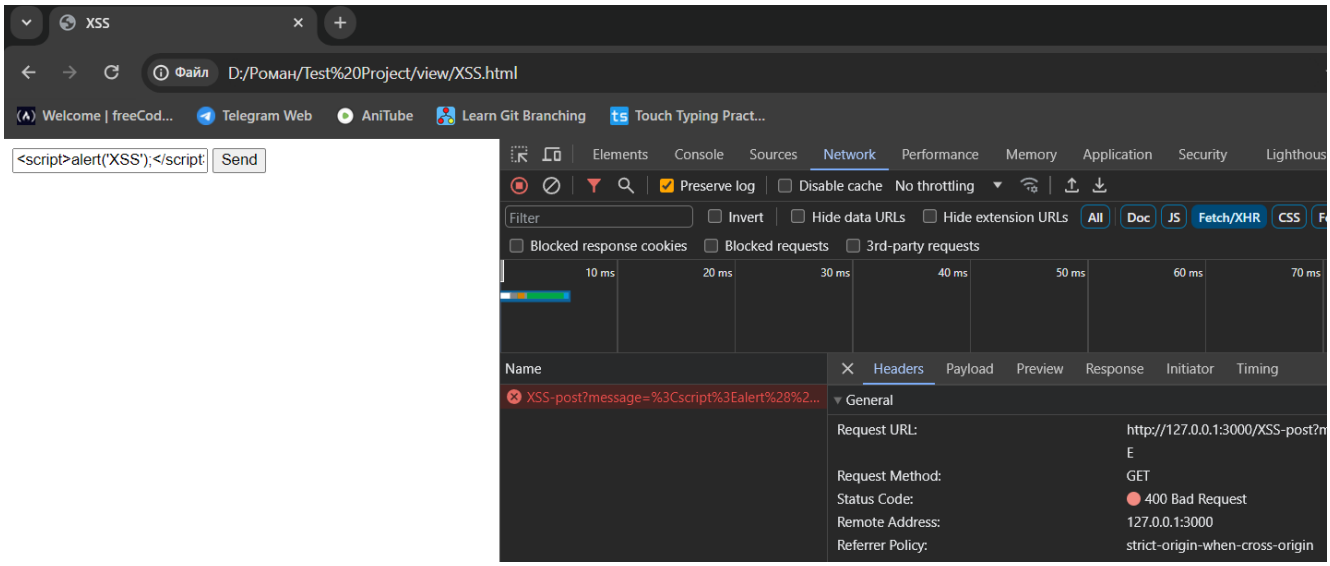

Рисунок 3.4 - Відхилення запиту при неправильних параметрах

Проте такий підхід не є зовсім доречним для листування. У такому разі краще використовувати ескейпінг спеціальних символів. Ескейпінг спеціальних символів є одним з методів валідації введених даних для запобігання XSS-атак. Його основна ідея полягає в тому, що спеціальні символи, які можуть використовуватись для ін'єкції шкідливого коду, заміняються спеціальними послідовностями, які не інтерпретуються браузером як код.

Розглянемо приклад коду який реалізовує цей метод:

```
app.get('/XSS-post', (req, res) => {
  let escapedMessage = req.query.message
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;")
    .replace(/'/g, "&#039;");
```

Дістаючи повідомлення з тіла запиту одразу замінюємо всі спеціальні символи спеціальними HTML кодами. Залишаємо решту коду, який відповідає за запис повідомлень до файлу, та повернення заескейпленого повідомлення в тілі запиту.

```
  let content = fs.readFileSync(fakeDbFile, {encoding: 'utf8'});
  fs.writeFileSync(fakeDbFile, content + escapedMessage + '\n');
  res.json({message: escapedMessage})
})
```

Як можна побачити на рисунку 3.5 у відповіді від сервера, спеціальні символи замінились на сервісні коди HTML, які відображаються як спеціальні символи на сторінці без виконання вразливого коду.

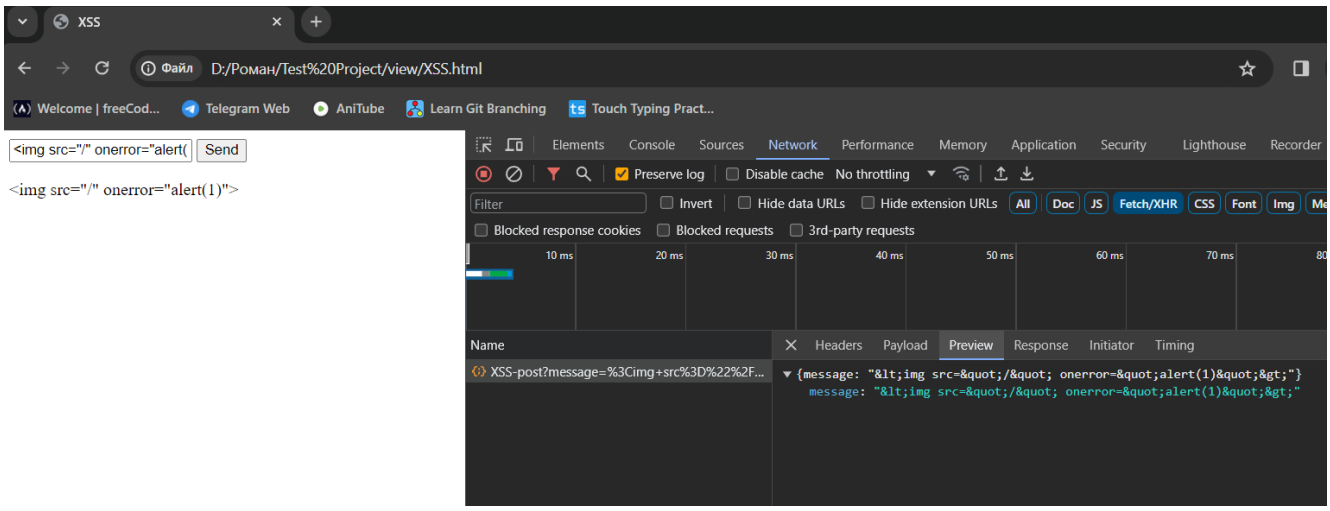

Рисунок 3.5 - Відформатована відповідь від сервера

Важливо зазначити, що у випадку коли дана вразливість була виявлена, не достатньо буде виправити лише її причину. Потрібно боротися також з наслідками. Для цього ескейпінг спеціальних символів потрібно буде використовувати і для читання даних, бо дані які збережені в базі даних, всеодно несуть загрозу.

Щоб відтворити таку поведінку, додамо вразливе повідомлення в базу, не заміняючи спеціальні символи на сервісні коди HTML, та перезавантажимо сторінку. Як можна побачити на рисунку 3.6, одне з повідомлень відобразилось на сторінці як img тег, що спричинило виконання вразливого скрипта.

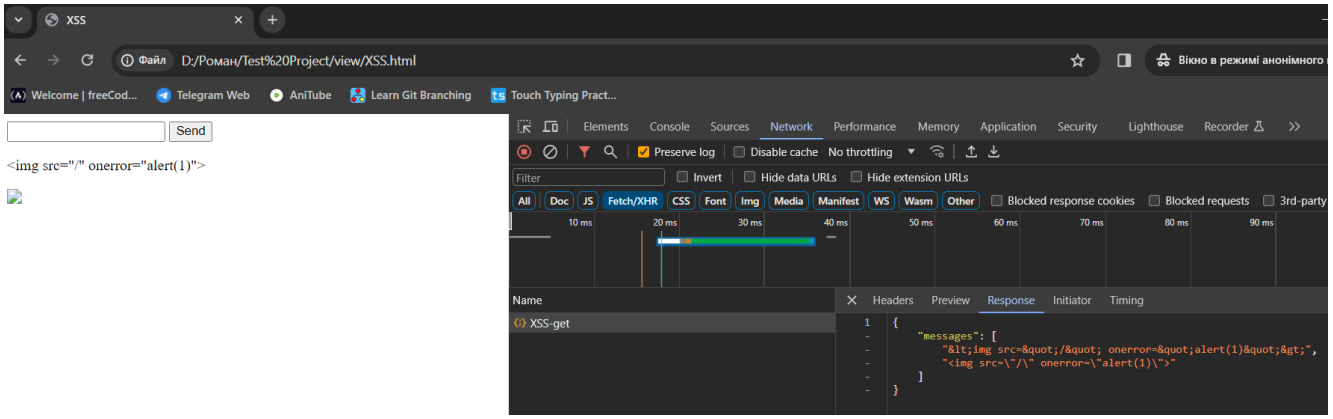

Рисунок 3.6 - Відповідь від сервера з вразливістю

Щоб виправити це, потрібно додати ескейпінг символів і на метод, який надсилає клієнту збережені дані.

Для початку, як і раніше діставатимемо контент файлу, і одразу вкажемо кодування, щоб латинські символи відображалися коректно.

```
let content = fs.readFileSync(fakeDbFile, {encoding: 'utf8'});
```

Після цього стрічку, що міститься в змінній content розділяємо по знаку \n за допомогою методу split. Цей метод поверне нам масив стрічок, на якому в свою чергу, потрібно викликати метод map, результатом якого є виконання колбеку для кожного елементу масиву. Колбек буде виконувати аналогічні операції над повідомленнями до тих, що виконуються перед записом.

```
let messages = content.split("\n").map(message =>
message
.replace(/</g, "&lt;")
.replace(/>/g, "&gt;")
.replace(/"/g, "&quot;")
.replace(/'/g, "&#039;")
);
```

Наприкінці повертаємо масив повідомлень на веб-сторінку, де пройшовши обробку, вони додадуться в DOM.

```
res.json({messages: messages})
```

Тепер маючи в базі безпечні і небезпечні повідомлення, при відправці клієнту, можна звести їх до одного формату (рисунок 3.7). Важливо ескейпити дані як на вході, так і на виході, оскільки ніхто не застрахований

від людського фактору, і не може точно сказати, що така вразливість точно не з’явиться в іншому місці де забули використати цей метод.

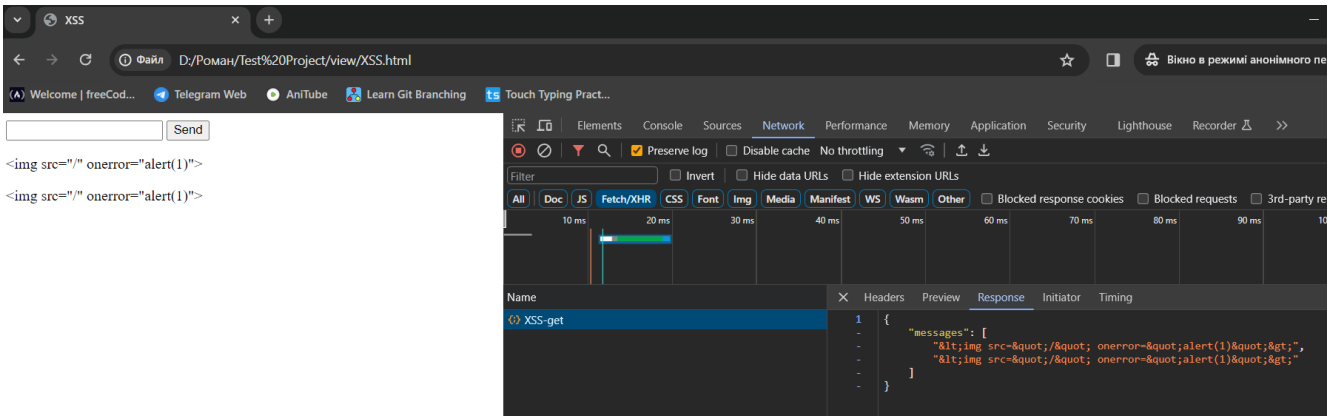

Рисунок 3.7 - Відформатована відповідь від сервера, для обох повідомлень

Проте, бувають випадки, коли користувачу потрібно якимось чином візуально оформити повідомлення. Особливо це релевантно, якщо це повідомлення пізніше надішлеться на електронну пошту. У такому випадку потрібно використовувати Whitelist. Цей підхід полягає в обмеженні типів HTML-тегів і атрибутів, які можуть бути використані у введених даних, дозволяючи лише допустимі елементи та їх атрибути. Використання білого списку забезпечує контроль над тим, які елементи і атрибути можуть бути відображені на сторінці, запобігаючи виконанню шкідливого скрипту.

Щоб реалізувати таку логіку, для початку потрібно створити функцію, яка зможе конвертувати спеціальні HTML символи в звичайні, для тих тегів, які будуть потрібні. Метод вмітиме працювати з тегами, які не містять атрибутів. Код реалізації такої функції може виглядати наступним чином:

```
function allowTags(message, tags) {
tags.map(tag => {
let regExpOpenTag = new RegExp(`&lt;${tag}&gt;`, 'g');
let regExpCloseTag = new RegExp(`&lt;\/${tag}&gt;`, 'g');
```

```
        message = message.replace(regExpOpenTag, `<${tag}>`).replace(regExpCloseTag, `</${tag}>`)
        });

        return message;
        }
```

Для перевірки роботи, додамо два повідомлення в базу: bold text і <i>italic text<i>. Щоб побачити різницю, під час виклику методу allowTags в масив tags передамо лише атрибут “b”, і побачимо, що він показується як елемент DOM, в той час як атрибут “i” відображається як звичайний текст (рисунок 3.8).

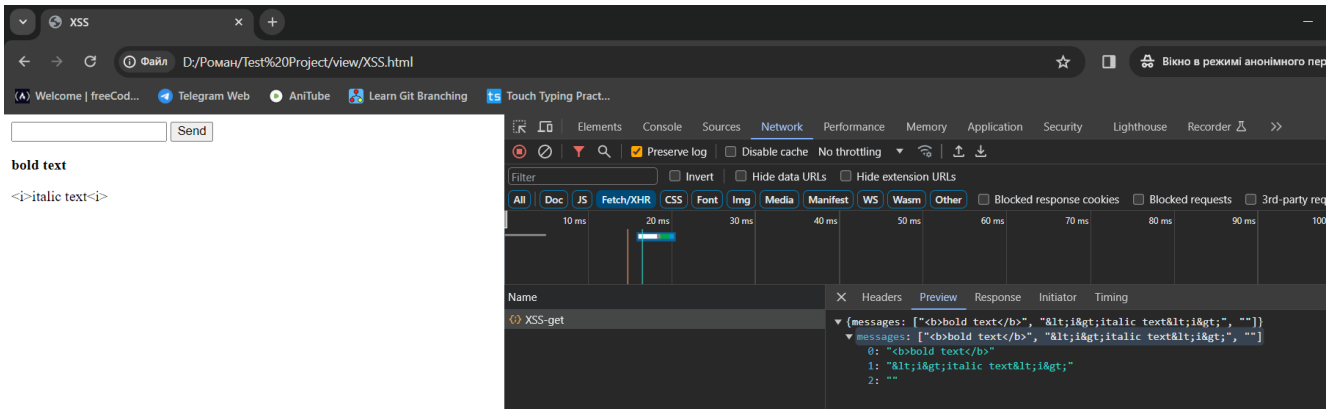

Рисунок 3.8 - Приклад застосування Whitelist

Іншим медотод захисту від XSS атак, який потрібно реалізувати на додатку є використання CSP заголовку. Як я раніше писав, CSP адміністраторам веб-сайтів встановлювати правила для браузера щодо завантаження вмісту з різних джерел. Для цього спочатку підготуємо проект. Щоб продемонструвати приклад роботи цього методу, використаємо зовнішній шрифт від Google під назвою, та застосуємо його як основний шрифт виведення повідомлень.

Для цього додаємо наступний код в тег head веб сторінки:

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Rubik+Bubbles&display=swap" rel="stylesheet">

Наступним кроком в атрибуті style елементу записуємо“font-family: 'Rubik Bubbles', monospace;”. В кінцевому результаті тег повинен виглядати на ступним чином:

Після цього, на нашій веб сторінці ми бачимо що шрифт на веб-сторінці змінився (рисунок 3.9). Можна переходити безпосередньо до додання CSP заголовку до всіх запитів.

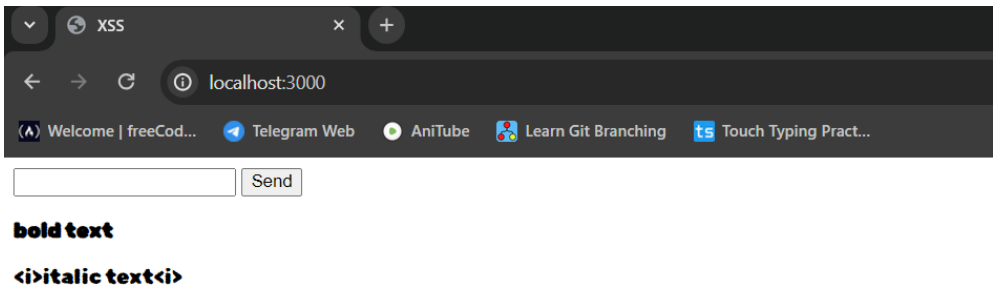

Рисунок 3.9 - Додання зовнішнього шрифту на сторінку

Додання заголовку відбувається на серверній частині додатку, і виглядає наступним чином:

```
app.use(function (req, res, next) {
  res.setHeader(
    'Content-Security-Policy-Report-Only',
```

```
        "default-src 'self'; font-src 'self'; img-src 'self'; script-src 'self'; style-src 'self'; frame-src 'self';"
      );
      next();
    });
```

Після додання такого коду, після перезавантаження веб-сторінки побачимо повідомлення (рисунок 3.10) про те що сторінці відмовлено у застосуванні вбудованого стилю, оскільки він порушує директиву CSP: "style-src 'self'.

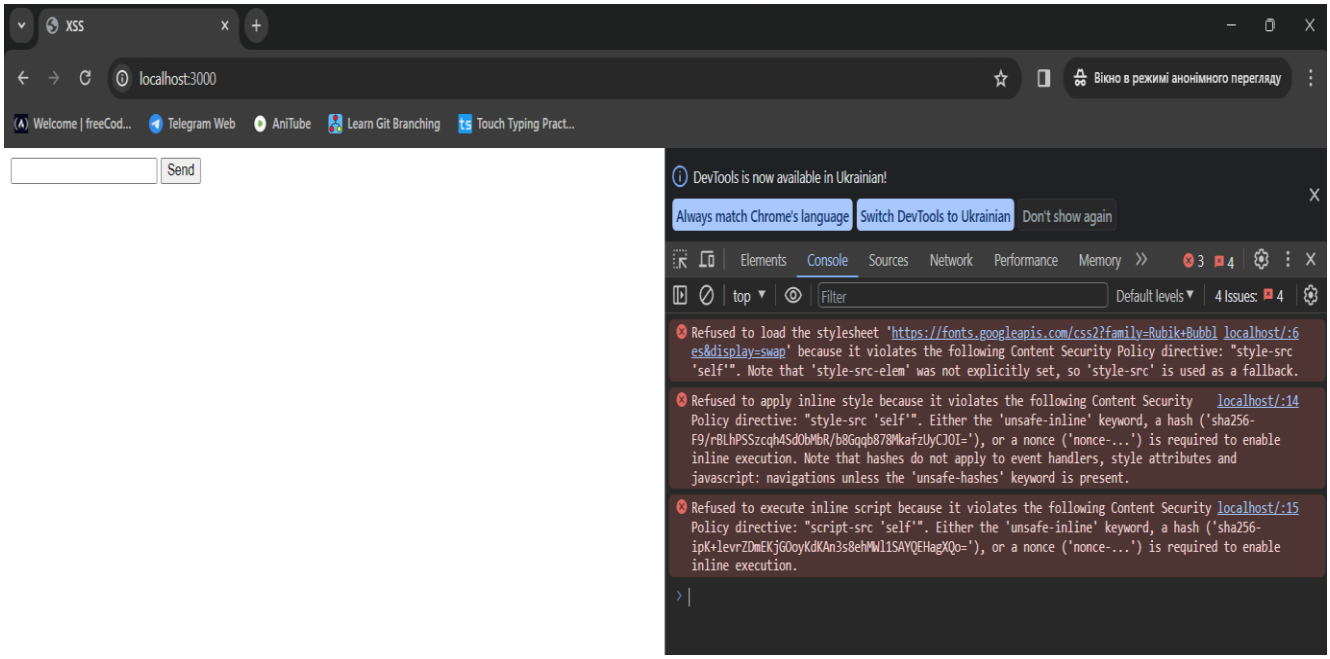

Рисунок 3.10 - Помилка при запиті до зовнішнього сервісу

Оскільки це довірене джерело є документація, про те як додати шрифти з цього ресурсу в список виключень. Після чого вони почнуть відображатися на сторінці.

Проте залишається проблема з виконанням скриптів, які знаходяться всередині HTML файлу (рисунок 3.11). Тобто, це ті скрипти які динамічно додаються на сторінку, бо в основному скрипти виносяться в окремі файли, що підключаються. Такі скрипти без проблем виконуються, а от скрипти що знаходяться в домі HTML сторінки - блокуються.

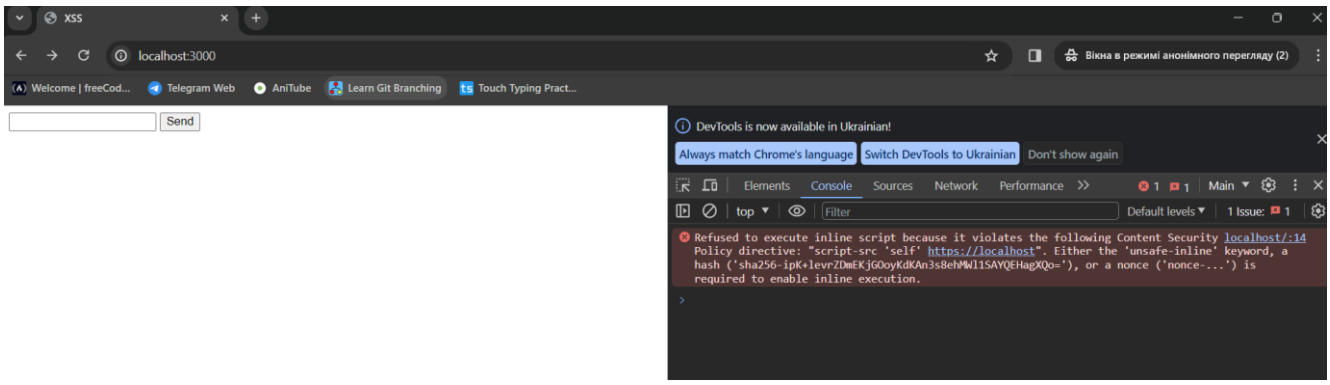

Рисунок 3.11 - Помилка при виконанні скрипту, який знаходиться всередині HTML

Як рішення проблеми для вбудованих скриптів пропонується або повне вимкнення фільтрування за допомогою вказування параметру unsafe-inline, або використання хешу чи nonce. Серед цих двох методів, звісно найкраще використовувати безпечний метод з використанням nonce. Перш за все потрібно згенерувати криптографічний ключ. Для цього добре підійде модуль crypto.

Підключити його до проекту можна додавши до коду const crypto = require('crypto'). Після цього додамо код, який буде повертати нам HTML сторінку, з параметром nonce, який буде додаватись до потрібного нам тегу.

Всередину методу який ініціалізує шлях для отримання веб-сторінки додамо код який генерує nonce за допомогою модуля crypto. Збережемо результат в змінну nonce, після чого додамо заголовок CSP, у якому до значення script-src буде доданий nonce, та який буде включений до відповіді сервера.

```
app.get('/', (req, res) =>88 {
  let nonce = crypto.randomBytes(16).toString('base64');
  res.setHeader(
    'Content-Security-Policy',
    "default-src 'self'; font-src 'self'; img-src 'self'; script-src 'self' 'nonce-" + nonce + "'; style-src 'self' 'unsafe-inline'; frame-src 'self';"
  );
```

Для динамічного nonce на сторінці, ініціалізуємо змінну у якій будемо зберігати шлях до HTML темплейту сторіки. Зчитавши контент за вказаним шляхом, потрібно підмінити ключове слово nonce всередині контенту на згенерований хеш. Після цих дій, зберегти новий контент в файл, і надіслати його у відповідь на запит.

```
const htmlFilePath = path.join(__dirname + '/view/XSS.html');
let htmlContent = fs.readFileSync(htmlFilePath, {encoding: 'utf8'});
htmlContent = htmlContent.replace(/{nonce}/g, nonce);
fs.writeFileSync(htmlFilePath + '.html', htmlContent);
res.sendFile(htmlFilePath + '.html', {nonce: nonce});
});
```

На в веб-сторінці додамо до тегу script атрибут nonce, який повинен бути доданий до всіх скриптів на веб-сторінці, щоб вони мали змогу виконуватися на ній. Код на ній виглядатиме наступним чином:

```
<script nonce="{nonce}">
const apiUrl = "http://localhost:3000";
const XssPost = apiUrl + "/XSS-post";
const XssGet = apiUrl + "/XSS-get";
```

Проведемо перевірку, додавши шкідливий скрипт на сторінку, і зможемо переконатися, що вразливий скрипт тепер не відпрацьовує (рисунок 3.12).

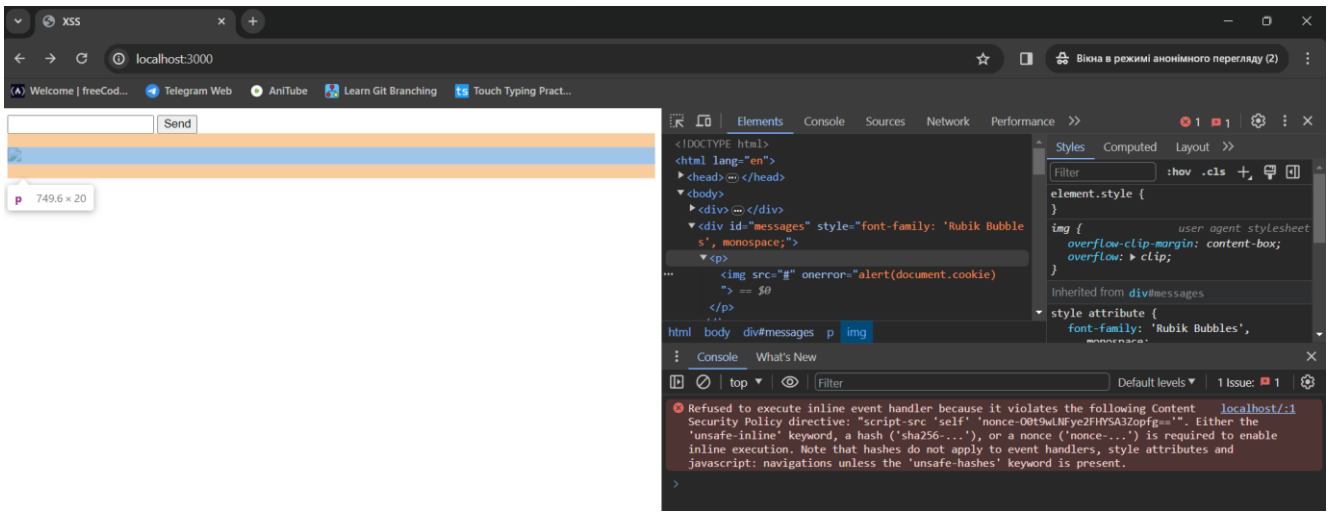

Рисунок 3.12 - Відображення тегу без виконання шкідливого скрипту

Варто зауважити, що переважна кількість XSS спрямована на отримання зловмисником даних сесії, авторизаційних токенів та інших даних, які зберігаються всередині куків. Для захисту куків від такої атаки, сервер встановлює атрибут HTTPOnly для певних куків. Як вказано в пункті 2.1 атрибут HTTPOnly використовується для HTTP-файлів cookie, щоб обмежити доступ до файлів cookie з JavaScript-скриптів, як захисний захід від XSS (міжсайтового скриптингу). Коли сервер встановлює HTTPOnly для файлу cookie, це означає, що до нього не можна отримати доступ через об'єкт document.cookie в JavaScript. HTTPOnly файли cookie використовуються виключно для взаємодії з сервером при доступі до ресурсів.

Для перевірки додамо в куки поле token, задамо йому випадкове значення, впровадимо шкідливий скрипт та спробуємо відобразити це значення через функцію alert. Додамо в метод контролера, який повертає сторінку, код res.cookie('token', nonce), щоб записати в поле token значення nonce, для прикладу (рисунок 3.13).

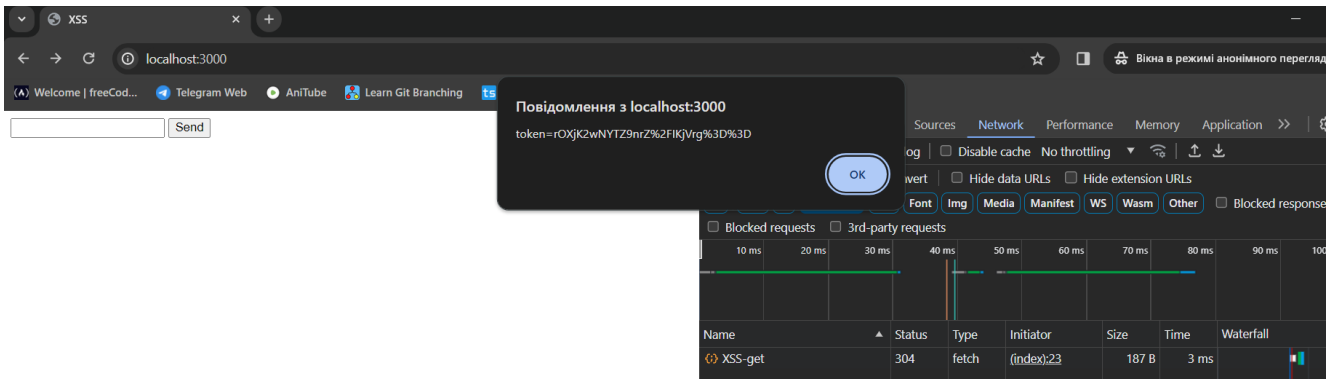

Рисунок 3.13 - Відображення значення куки без атрибута HTTPOnly

Тепер спробуємо додати до куків атрибут, про який згадували раніше. Для цього змінюємо стрічку на серверній стороні, де до того сетилося значення куки за ключом token на res.cookie('token', nonce, {httpOnly: true}). Після перезавантаження сторінки бачимо (рисунок 3.14), що звернення до document.cookies повернуло нам пусте значення, що й очікувалось, хоча в браузері існує кука за ключем token.

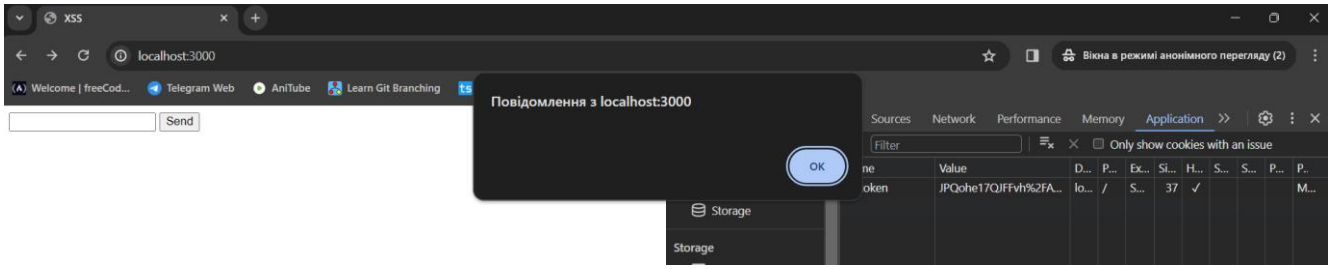

Рисунок 3.14 - Відображення значення куки з атрибутом HTTPOnly

3.3 Реалізація методів захисту від CSRF вразливостей

Перший метод захисту від CSRF вразливостей, який буде розглянуто це використання CSRF токенів. Для цього в тестовий проект додатково потрібно встановити модуль CSRF. Аналогічно до того, як створюється надсилається nonce для скриптів, потрібно надіслати CSRF токен. Після цього, метод контролера для отримання сторінки виглядатиме наступним чином:

app.get('/', (req, res) => {

```
…
  var token = csrf.create(CSRF_SALT)
  htmlContent = htmlContent.replace(/{csrf}/g, token);
  fs.writeFileSync(htmlFilePath + '.html', htmlContent);
  res.sendFile(htmlFilePath + '.html', {nonce: nonce});
});
```

Після цього потрібно додати логіку валідації токену при запиті на додання повідомлення. Код, який реалізує таку логіку, виглядатиме наступним чином:

```
app.get('/XSS-post', (req, res) => {
  if(!csrf.verify(CSRF_SALT, req.query.csrf)) {
    res.sendStatus(401);
    res.send();
  }
```

На даному етапі, при спробі додання повідомлення, веб сервер буде повертати 401 код, і відповідно не буде додавати повідомлення (рисунок 3.15).

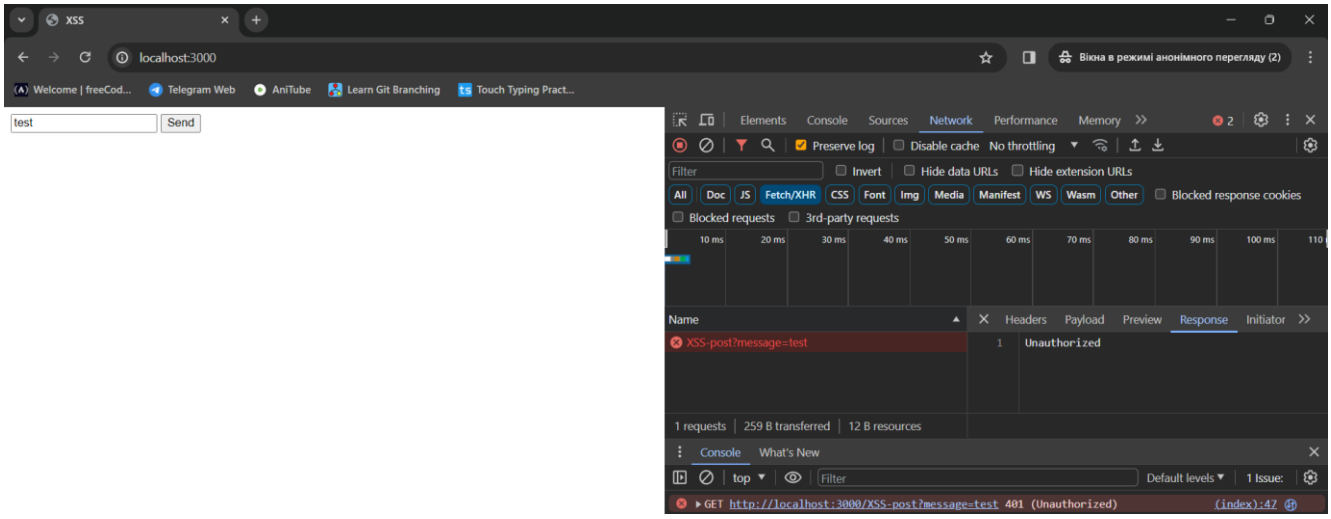

Рисунок 3.15 - Помилка при неавторизованому запиті

Потрібно також безпосередньо в HTML сторінці зміни будуть всередині форми, а саме, потрібно буде додати <input id="csrf" value="{csrf}" hidden> та додати CSRF токен в тіло запиту. Після цього все працює як повинно. При спробі емулювання ситуації, коли csrf токен вказаний невірно, також повертається 401 код.

Використання SameSite cookies є наступним методом захисту, який ми будемо реалізовувати в межах тестового проєкту. Ключ SameSite cookies може приймати всього 2 значення: “Strict” і “Lax”. Щоб відтворити це практично, спробуємо додати повідомлення з іншої сторінки, яку отримали відкривши звичайний html документ. Ось приклад коду, який реалізовує шкідливу дію додання повідомлення:

<img src=”#” onerror=”() => fetch(‘https://locahost:3000/XSS-post?message=hacked’) ”>

У випадку коли користувач відкриває таку сторінку, відправляється запит до нашого сервера, який додає повідомлення hacked (рисунок 3.16). Крім того, всі куки як були підв’язані до домену, на який робився запит, будуть надіслані з запитом (проте це не працює з localhost). Щоб цього уникнути, додаємо на сервер наступний код: res.cookie('token', nonce, {httpOnly: true, sameSite: 'strict'}). Цей код проставить флажок SameSite для куків, і більше не дозволить викликати куки при запитах до цього домену з сторонніх джерел.

Вразливість яку нівелює метод Double Submit Cookies також не вдастся відтворити, з тієї ж причини що і попередню, проте код який реалізовує цей метод виглядатиме наступним чином:

```
app.get('/XSS-post', (req, res) => {
  if(req.cookies?.token !== req.query.csrf || !csrf.verify(CSRF_SALT, req.query.csrf)) {
    res.sendStatus(401).send();
    return;
```

}

…

При отриманні запиту порівнюється значення CSRF токену в куках, і в тілі запиту. Якщо токен відсутній хоча б в одному місці, чи він не ідентичний, запит буде відхилено.

Останній метод захисту, від CSRF атак, який буде розглянуто дуже примітивний, проте вкрай важливий. Це використання POST запитів для чутливих операцій. Цей підхід використовує властиві характеристики браузерів, які зазвичай не дозволяють веб-сторінкам автоматично генерувати запити POST за допомогою вбудованих тегів.

Зараз є можливість запустити сторінку, яка була згенерована для демонстрації вразливості з кодом <img src=”#” onerror=”() => fetch(‘https://locahost:3000/XSS-post?message=hacked’) ”>. Такий запит успішно виконається (рисунок 3.16), і на основній сторінці відобразиться повідомлення з текстом hacked.

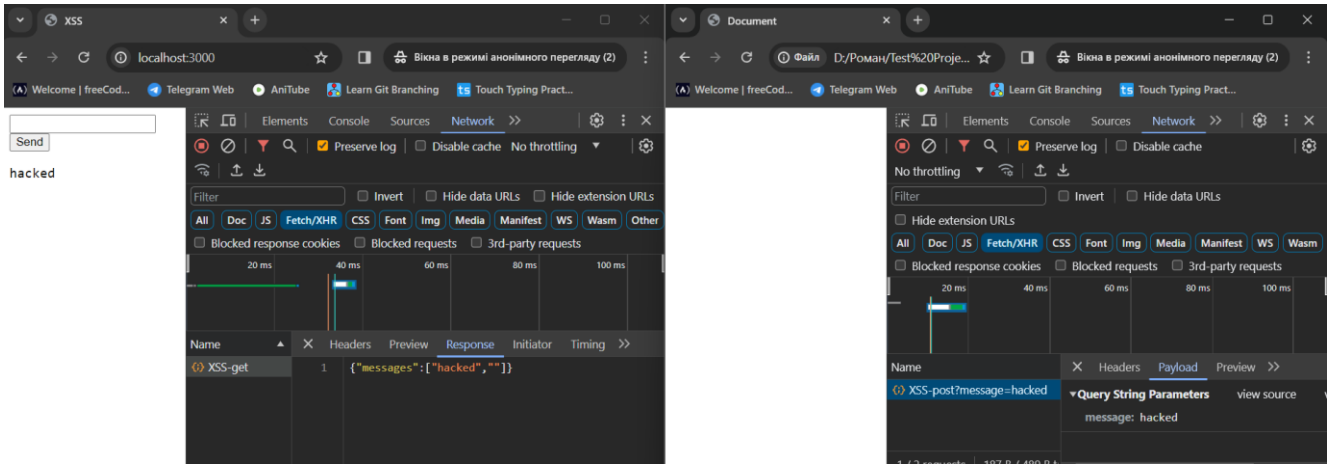

Рисунок 3.16 - Спрацювання CSRF вразливості

Для запобігання такої поведінки варто змінити в контролері веб-сервера тип запиту замість app.get('/XSS-post'... на app.post('/XSS-post'… Крім того, не забути змінити методи запиту на веб сторінці. Після таких правок, подібні вразливості стануть недоступні, а запит буде повертати 404.

ВИСНОВОК

Було досліджено XSS та CSRF атаки та оцінено потенційну шкоду, яку вони можуть завдати. Крім того, в роботі були розглянуті основні причини цих вразливостей, що сприяло кращому розумінню кореневих проблем та запропонувало шляхи їх подолання.

Для забезпечення раннього виявлення та пом'якшення загроз безпеці в роботі було досліджено можливості проведення попереднього аналізу на предмет XSS та CSRF вразливостей. Також було проаналізовано та порівняно існуючі стратегії захисту від цих вразливостей та запропоновано вдосконалення методологій захисту на основі цих висновків. Таким чином, можна впровадити більш ефективні та практичні заходи для захисту веб-додатків від потенційних атак.

Окремий розділ присвячений розробці локального продукту, який не тільки демонструє актуальні вразливості, але й висвітлює методи захисту від них. Такий практичний підхід дає реальне уявлення про те, як вразливості можуть проявлятися в реальних веб-додатках і як їх можна ефективно усунути або запобігти їм.

Проведено комплексний аналіз XSS та CSRF вразливостей, та практичні рішення для покращення захисту веб-додатків від цих атак. Такий підхід до дослідження сприяє розвитку більш безпечного веб-середовища та кращому розумінню сучасних стратегій для захисту веб-додатків від цих атак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дослідження вразливостей Web-сайтів та методів їх усунення [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: http://phone.kpi.ua/wpcontent/uploads/2014/06/4.pdf

2. Article. Cross-Site Scripting (XSS) Makes Nearly 40% of All Cyber Attacksin 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://www.precisesecurity.com/articles/cross-site-scripting-xss-makesnearly-40-of-all-cyber-attacks-in-2019/

3. 2011 Top 25 Most Dangerous Software Errors. CWE/SANS. [Електроннийресурс] - Режим доступу: https://www.sans.org/top25-software-errors -06.05.2019

4. The Web Application Security Consortium (WASC) [Електронний ресурс] – Режим доступу до ресурсу: http://www.webappsec.org/

5. WhiteHat Security’s Approach to Detecting Cross-Site Request Forgery(CSRF). Apr. 2011. [Електронний ресурс] - Режим доступу:https://www.whitehatsec.com/ - 06.05.2019

6. Kolšek, Mitja. (2003). Session fixation vulnerability in web-basedapplications. - Acros Security Режим доступу до ресурсу: http://www.acrossecurity.com/papers/session_fixation.pdf

7. Security in Depth: New Security Features [Електронний ресурс] - Режимдоступу: https://blog.chromium.org/2010/01/security-in-depth-new-securityfeatures.html - 06.05.2019

8. The Web Application Security Consortium (WASC) – Articles [Електронний ресурс] – Режим доступу до ресурсу: http://www.webappsec.org/projects/articles/

9. Boyan Chen [Електронний ресурс] - Режим доступу: https://www.researchgate.net/publication/220875877_A_Study_of_the_Effectiveness_of_CSRF_Guard. - 06.05.2019

10. Security from CSRF. [Електронний ресурс] - Режим доступу: https://www.owasp.org/index.php/CrossSite_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet - 06.05.2019

11. CGISECURITY - The Cross-Site Scripting (XSS) FAQ [Електронний ресурс] – Режим доступу до ресурсу: https://www.cgisecurity.com/xssfaq.html

12. “Why Johnny Can’t Pentest: An Analysis of Black-Box Web Vulnerability Scanners”. [Текст] / Адам Дюпе, Марко Кова, та Джовані Вігна - Springer, 2010, pp. 111– 131c.

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

14. XSS Secured WebApplications by using innerHTML Mutations [Електронний ресурс] - Режим доступу: https://security.stackexchange.com/questions/46836/what-is-mutation-xssmxss - 06.05.2019

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

16. White Hat Security testing. Improper Input Handling. [Електронний ресурс] – Режим доступу: https://www.whitehatsec.com/glossary/content/improperinput-handling - 06.05.2019

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

18. Internet Security Threat Report [Електронний ресурс] - Режим доступу: https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf - 06.05.2019

19. SessionStack Blog - How JavaScript works: 5 types of XSS attacks + tips on preventing them [Електронний ресурс] – Режим доступу до ресурсу: https://blog.sessionstack.com/how-javascript-works-5-types-of-xss-attackstips-on-preventing-them-e6e28327748a

20. Automated Mechanism for Secure Input Handling [Електронний ресурс] -Режим доступу: https://www.researchgate.net/publication/42803679_An_Automated_Mechanism_for_Secure_Input_Handling - 06.05.2019

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

22. OWASP: Vulnerability Scanning Tools. [Електронний ресурс] - Режим доступу: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools - 06.05.2019

23. .Content Security Policy Reference [Електронний ресурс] – Режим доступу до ресурсу: https://content-security-policy.com/

24. Sending form data. [Електронний ресурс] - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data - 06.05.2019

25. 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. - 06.05.2019

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

27. “Cross Site Request Forgery: A common web application weakness”. In: Communication Software and Networks (ICCSN), [Текст] / Mohd. Shadab Siddiqui and Deepanker Verma. 2011 IEEE 3rd International Conference on. IEEE, 2011.

28. DOM based XSS Prevention Cheat Sheet [Електронний ресурс] – Режим доступу до ресурсу:

https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html

29. The Web Application Security Consortium: Web Application Security Scanner List. W3Techs. Usage of server-side programming languages for websites. [Електронний ресурс] - Режим доступу: https://blog.hackmetrix.com/wordpress-vulnerability-scanner/ - 06.05.2019

30. OWASP ZAP [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-zap/

31. Burp Pro [Електронний ресурс] – Режим доступу до ресурсу: https://portswigger.net/burp/pro

32. Web Security testing with Free Web Scanners [Електронний ресурс] - Режим доступу: https://www.antimalware.ru/reviews/free_scanners_security_websites - 06.05.2019

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

34. Cross-Site Request Forgery (CSRF) [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.owasp.org/index.php/CrossSite_Request_Forgery_(CSRF).