Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження та удосконалення стандартних засобів захисту web-додатків

Виконав(ла): студент(ка) 6 курсу, групи СБм-61
спеціальності 125 Кібербезпека
(шифр і назва спеціальності)
(підпис) Степанов А. В.
(прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Загородна Н. В.
(підпис) (прізвище та ініціали)
« » 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 кібербезпека
(шифр і назва спеціальності)

студенту Степанов Андрій В'ячеславович
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та удосконалення стандартних засобів захисту web-додатків

Керівник роботи д.т.н., професор Карпінський Микола Петрович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи 14.12.2021
3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Проаналізувати літературні джерела в області дослідження
Здійснити опис вразливостей
Здійснити опис найпоширеніших методів захисту від них
Розробити на їх основі удосконалення та розглянути їх ефективність.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г. М., к.т.н, доцент		20.12
Безпека в надзвичайних ситуаціях	Клепчик В. М., старший вилкдач		15.12.21

7. Дата видачі завдання 25.09.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Підбір наукової літератури	до 20.09.2021	виконав
2.	Написання першого розділу роботи	до 24.09.2021	виконав
3.	Написання другого розділу роботи	до 15.10.2021	виконав
4.	Написання тез до публікації	до 30.10.2021	виконав
5.	Написання третього розділу роботи	до 19.11.2021	виконав
6.	Написання висновків по роботі	до 03.12.2021	виконав
7.	Підготовка демонстраційних матеріалів	до 21.12.2021	виконав
8.	Захист у ДЕК	23.12.2021	виконав

Студент _____ Степанов А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Удосконалення стандартних методів захисту веб-додатків // Дипломна робота ОР «Магістр» // Степанов Андрій В’ячеславович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБм-61 // Тернопіль, 2021 // С. 87 , рис. – 27 , табл. – , слайдів – 12 , додат. – .

Ключові слова: WEB-ДОДАТОК, СТАНДАРТНІ ЗАСОБИ ЗАХИСТУ, ВРАЗЛИВІСТЬ, ПЕРЕБІР, ВАЛІДАЦІЯ, ТЕСТУВАННЯ, WAF, XSS, SQL-ІН’ЄКЦІЯ.

В роботі було проведено огляд літературних джерел в області дослідження. Здійснено огляд загального стану безпеки та вразливостей веб-додатків. Описано процес тестування безпеки веб-додатка. Також, здійснено огляд використання WAF. Описано, найпоширеніші способи захисту веб-додатків від підбору, слабкої валідації відновлення пароля, XSS та SQL-ін’єкцій.

У результаті виконання дипломної роботи, були розроблені удосконалення, які є простими в реалізації, ефективними, надійними та продуктивними, що дозволяє використовувати їх при розробці нових додатків, або інтегрувати їх, як удоконалення, до вже наявних механізмів захисту додатка.

ANNOTATION

Study and improvement of standard methods of web-applications protection // Thesis of the Master degree // Stepanov Andrii Viacheslavovich // Ternopil Ivan Puluj National Technical University, Department of Computer Information Systems and Software Engineering, Department of Cybersecurity // Ternopil, 2021 // P. 87 , Fig. – , Tables – , Annexes – , References. – .

Ключові слова: WEB-APPLICATION, STANDARD METHODS OF WEB-APP PROTECTION, VULNERABILITY, BRUTE FORCE, VALIDATION, TESTING, WAF, XSS, SQL-INJECTION.

The paper reviews literature sources in the field of research. A comparative analysis of the general state of security and vulnerabilities of the web-apps is conducted. The Web-app testing process was described. Also, WAF purposes and goals was overviewed. Described the most common web-app security methods for brute force, weak password recovery validation, XSS and SQL-Injection attacks.

As a result of this work enhancement to the standard web-app security methods was developed. They are easy to implement, effective, reliable and efficient. All these properties allow developers to use them in new web-apps, or integrate them to the existing security systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - База даних

ПЗ - Програмне забезпечення

ЦРПЗ - Цикл розробки програмного забезпечення

СКБД - Система Керування Базами Даних

API - Application Program Interface

CSRF - Cross-site request forgery

DAST - Dynamic Application Security Testing

MPA - Multi Page Application

RASP - Run-time Application Security Protection

SaaS - Software as a service

SAST - Static Application Security Testing

SPA - Single Page Application

WAF - Web-application firewall

XSS - Cross-site scripting

ВСТУП

Безпека веб-додатків (також відома як Web AppSec) — це ідея створення веб-сайтів, щоб вони функціонували належним чином, навіть коли вони піддаються атаці. Концепція передбачає набір засобів керування безпекою, вбудованих у веб-додаток для захисту його активів від потенційно шкідливих агентів.

Веб-додатки, як і будь-яке програмне забезпечення, неминуче містять дефекти. Деякі з цих дефектів є реальними вразливими місцями, якими можна скористатися, створюючи ризики для організацій. Засоби безпеки веб-додатків захищають від таких дефектів як XSS, SQL-injection, CSRF, Directory Listinng і т.д. Це передбачає використання безпечних методів розробки та впровадження заходів безпеки протягом усього життєвого циклу розробки програмного забезпечення (ЦРПЗ), забезпечуючи усунення недоліків на рівні дизайну та помилок на рівні реалізації.

Зловмисники вважають веб-додатки першочерговими цілями через:

- внутрішню складність їх вихідного коду, що підвищує ймовірність непомітних, на перший погляд, вразливостей і маніпуляцій зловмисним кодом;
- високі винагороди, включаючи конфіденційні приватні дані, зібрані в результаті успішного маніпулювання вихідним кодом;
- простоту виконання, оскільки більшість атак можна легко автоматизувати та запускати проти тисяч або навіть десятків чи сотень тисяч цілей одночасно.

Поширеними цілями атак на веб-додатки є системи керування вмістом (наприклад, WordPress), інструменти адміністрування баз даних (наприклад, phpMyAdmin) і додатки SaaS. Вони можуть мати вразливості до: Brute Force, XSS,

SQL-Injection та різного роду вразливості, пов’язані з використанням неякісних тем та плагінів.

Організації, які не можуть захистити свої веб-програми, ризикують зазнати атаки. Це може призвести до крадіжки інформації, псування відносин із клієнтами, відкликання ліцензій та судового розгляду. Таким чином, розробка та удосконалення методів захисту веб-ресурсів від атак залишається актуальною проблемою, особливо з урахуванням постійного вдосконалення методів та інструментів атак, а також зі зростанням економічних, соціальних та політичних наслідків зловмисних дій.

Мета роботи - підвищення ефективності методів захисту веб додатків, шляхом покращення продуктивності, збільшення складності шифрування, додавання нових етапів та обмежень до процесу перевірки і т.д.

З мети мають випливати задачі:

- проаналізувати літературні джерела в області дослідження;
- здійснити опис вразливостей;
- здійснити опис найпоширеніших методів захисту від них;
- розробити на їх основі удосконалення та дослідити їх ефективність.

Об’єкт дослідження -безпека веб-додатків.

Предмет дослідження - методи захисту веб-додатків

Наукова новизна отриманих результатів - удосконалено існуючі стандартні методи захисту веб-додатків, що дозволило підвищити ефективність їх захисту від атак типу SQL-injection, XSS, Brute Force та слабка валідація відновлення пароля.

Практичне значення даної роботи полягає в тому, щоб запропонувати удосконалення до стандартних засобів захисту веб-додатків, які були б ефективними та придатними до використання у вже існуючих, або майбутніх додатках.

Апробація результатів роботи. Окремі результати роботи доповідались на IX науково-технічній конференції «Інформаційні моделі, системи та технології», Тернопіль, ТНТУ, 8 – 9 грудня 2021 р.

РОЗДІЛ 1 ТЕСТУВАННЯ WEB-ДОДАТКІВ НА ВРАЗЛИВОСТІ

1.1 Процес роботи веб-додатку

Щоб зрозуміти принцип атаки на веб-додатки, потрібно спочатку розібратися як вони працюють. Веб-додатки можуть бути SPA, або MPA. Що визначає, як буде оновлюватись контент на сторінці. Зазвичай, у роботі використовується веб-сервер, для обробки запитів з клієнта та база даних для зберігання даних користувачів. Також, при роботі додатка можуть використовуватися додаткові місця для зберігання статичного контенту: картинок, музики, відео і т.д.

Принцип роботи веб-додатка на прикладі SPA:

1. Браузер запитує з сервера HTML файл.
2. Сервер повертає пустий HTML файл з прикріпленими стилями та скриптами.
3. Поки скрипт виконується, клієнт бачить пусту сторінку, або екран загрузки.
4. Додаток отримує необхідні дані, генерує вигляд сторінки і вставляє його в об’єктну модель документа.
5. Додаток готовий до роботи

Принцип роботи веб-додатка на прикладі MPA:

1. Браузер запитує з сервера HTML файл.
2. Сервер здійснює запити до БД для збору необхідної для сторінки інформації
3. Сервер повертає HTML сторінку зі скриптами, стилями та картинками.
4. Додаток готовий до роботи

Принцип роботи типового веб-додатку, типу MPA показаний на Рисунку 1.1

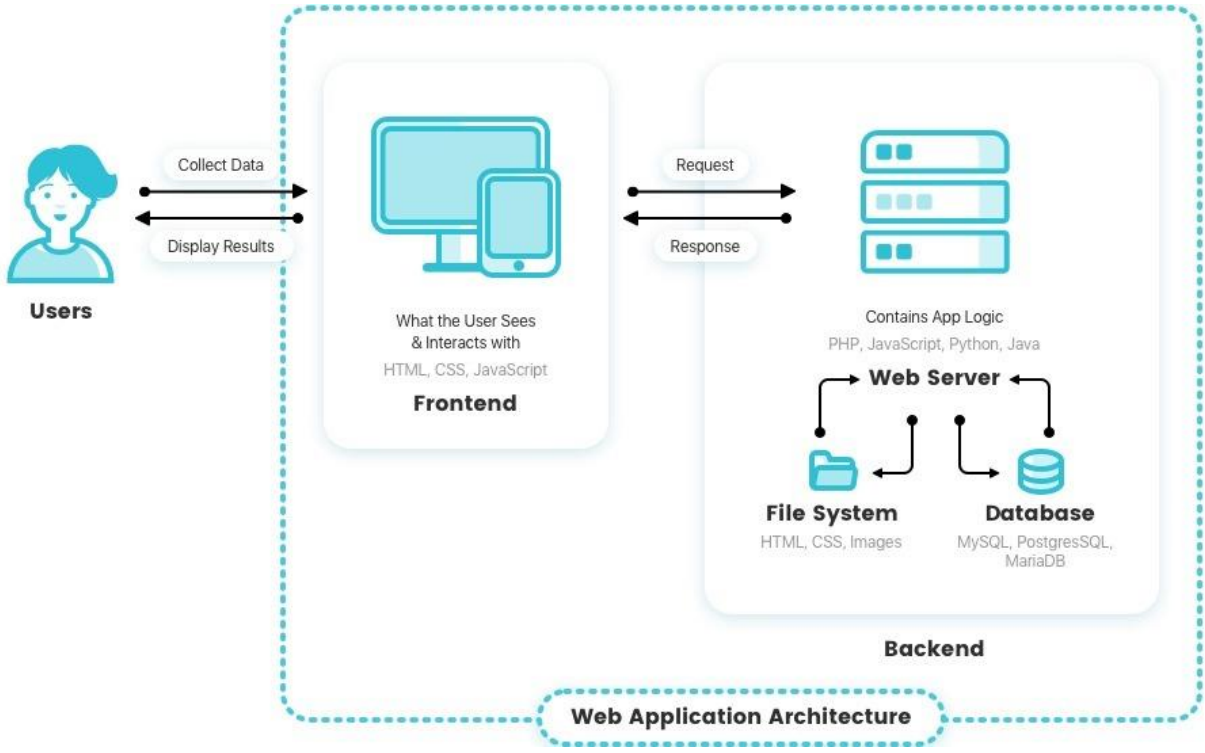

Рисунок 1.1 - Процес роботи веб-додатку типу MPA

Загалом, MPA додатки старші, та більш поширені ніж SPA.

1.2 Вразливості та їх вплив на безпеку бізнесу

Вразливість – це недолік програми або пристрою, який може бути використаний зловмисниками. Зловмисники можуть використовувати вразливість для досягнення такої мети, як крадіжка конфіденційної інформації, скомпрометувати систему, зробивши її недоступною (у сценарії відмови в обслуговуванні), або зіпсувати дані.

Вплив вразливостей залежить від експлойту. Acunetix призначає серйозність в основному залежно від впливу, який експлойт може мати на систему. Серйозність також залежить від того, наскільки важко використати вразливість.

Ваш бізнес може мати багато систем, які працюють одночасно, і деякі з них важливіші за інші. Acunetix дозволяє оцінювати ці системи, використовуючи бізнес-критичність. Основні системи мають вищу критичність, ніж несуттєві.

- Високий рівень серйозності: цей рівень вказує на те, що зловмисник може повністю порушити конфіденційність, цілісність або доступність системи без необхідності спеціального доступу, взаємодії з користувачем або обставин, які не контролюються зловмисником. Цілком імовірно, що зловмиснику вдасться ескалувати атаку на операційну систему та інші системи.
- Середня серйозність: цей рівень вказує на те, що зловмисник може частково порушити конфіденційність, цілісність, або доступність цільової системи. Їм може знадобитися спеціалізований доступ, взаємодія з користувачем або обставини, які не залежать від зловмисника. Такі вразливості можуть використовуватися разом з іншими вразливими місцями для ескалації атаки.
- Низький рівень серйозності: цей рівень вказує на те, що зловмисник може обмеженим чином поставити під загрозу конфіденційність, цілісність або доступність цільової системи. Їм потрібен спеціалізований доступ, взаємодія з користувачем або обставини, які не залежать від зловмисника. Для ескалації атаки такі вразливі місця необхідно використовувати разом з іншими вразливими місцями.

Комбіновані вразливості - у більшості випадків вразливостей середнього та низького ступеня тяжкості атака можлива або більш небезпечна, коли зловмисник поєднує її з іншими вразливими місцями. Такі вразливі місця часто включають соціальну інженерію.

Уразливості високого рівня. У випадку кількох вразливостей високого ступеня тяжкості кількість випадків, виявлених у 2020 році, зросла порівняно з 2019 роком. Ймовірні причини та наслідки будуть розглянуті в наступних розділах, присвячених конкретним уразливостям.

1.3 Стан безпеки веб додатків

Звіт за 2021 рік [1], на жаль, досить песимістичний. Тенденція покращення, що спостерігалася в попередні кілька років, закінчилась. Кілька вразливостей з високим та середнім ступенем небезпеки зараз більш поширені, ніж у 2020 році, включаючи деякі серйозні ризики безпеки, які можуть призвести до втрати конфіденційної інформації.

Таке погіршення викликано пандемією COVID-19. Пандемія змусила більшість компаній перейти на віддалену роботу, і тому багато керівників служб безпеки вирішили зосередитися на безпеці кінцевих точок, безпеки операційних систем і на ПЗ для боротьби з фішингом, шкідливими сайтами і шкідливим кодом. Отже, було недостатньо ресурсів для підвищення рівня безпеки в Інтернеті. Замість того щоб вкладати кошти в ретельні процеси, компанії вибирали швидкі і недосконалі рішення, часто засновані на неправильно налаштованих брандмауерах веб-додатків (WAF).

Фахівці Acunetix вважають, що такі рішення можуть мати серйозні наслідки в майбутньому. В результаті переходу на віддалену роботу важливість веб-додатків зросла. Щоб підвищити ефективність віддаленої роботи, багато компаній зробили свої процеси доступними через веб-браузери, використовуючи веб-додатки і API. Це дозволило зловмисникам отримати доступ до даних компанії через веб-сторінки і, як наслідок, могло привести до серйозних витоків даних.

Проте розробка стійкого до атак веб-програмного забезпечення стикається з великою кількістю різних проблем, а не тільки з браком ресурсів і переходом на віддалену роботу. Ще до епохи COVID розробники часто натрапляли на труднощі при написанні безпечного коду, допускали типові функціональні помилки,

пропускали перевірку, допускали введення користувачем даних з ненадійних джерел, передавали ненадійні дані безпосередньо в запити SQL, використовували незахищені ідентифікатори сеансів користувачів і механізми управління сеансами і так далі (рисунок 1.2).

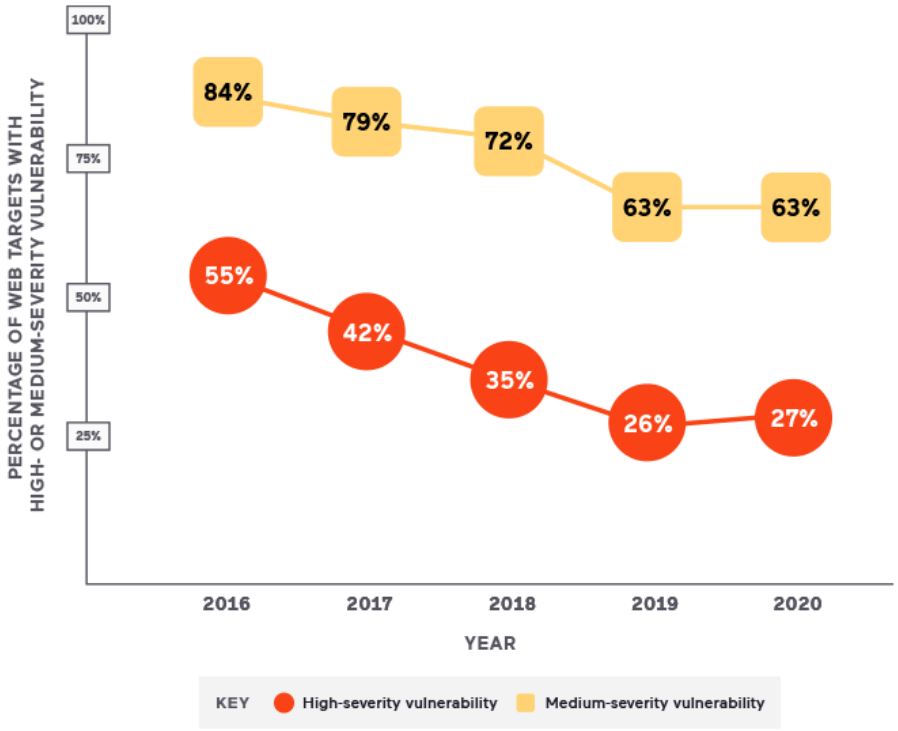

Рисунок 1.2 - Статистика веб-додатків з середньою, та високою рівнями вразливості

Нові середовища віддаленої роботи ще більше ускладнюють розробникам забезпечення безпеки коду додатків через проблеми зі зв'язком. Якщо акцент в області безпеки зміщується з рішень для забезпечення безпеки веб-додатків, розробникам також не вистачає інструментів і навчання для поліпшення своїх навичок, пов'язаних з безпекою. Якби у них був доступ до професійних рішень безпеки веб-додатків, вони б отримували не тільки інформацію про існування проблем, але і гайди, які навчили б їх, як уникнути таких помилок у майбутньому.

Без таких інструментів розробники будуть створювати все більше і більше вразливостей.

1.4 Короткий огляд найпоширеніших вразливостей web-додатків

Щороку Acunetix надає аналіз найбільш поширених вразливостей веб-безпеки і вразливостей мережевого периметра. Щорічний звіт про вразливість веб-додатків (тепер є частиною Invicti AppSec Indicator) заснований на реальних даних, взятих з Acunetix Online. Команда фахівців випадковим чином вибирає веб-сайти і веб-додатки, просканові за допомогою сканера Acunetix, робить їх анонімними і проводить статистичний аналіз. І ось до яких висновків з кібербезпеки вони прийшли.

У звіті компанії Acunetix [1] основна увага приділяється поширеним вразливостям і неправильним налаштуванням безпеки – тим, які також є в списку OWASP. Можна відзначити менше SQL-*ін'єкцій* і проблем з обходом каталогів (обходом шляху). Але багато інших серйозних проблем були настільки ж частими, як і роком раніше. Мова йде про RCE (віддалене виконання коду), проблеми міжсайтового скриптинга (XSS), вразливі бібліотеки JavaScript, вразливості WordPress, підробку серверних запитів (SSRF), атаки шляхом впровадження заголовка хоста і багато іншого.

Рисунок 1.5 - Процент найпоширеніших властивостей веб-додатків

Звіт також містить дані про інші відомі вразливості і проблеми безпеки програмного забезпечення, включаючи переповнення буфера, відмова в обслуговуванні і DDoS-вразливості, проблеми, пов’язані з контролем доступу і порушень аутентифікації, таких як слабкі паролі, неправильна конфігурація веб-сервера і багато іншого. У випадку всіх цих проблем тенденція аналогічна: спостерігається невелике збільшення їх кількості.

1.5 Тестування безпеки веб-додатків

1.5.1 Актуальність тестування веб-додатків

На веб-додаток у сучасному середовищі може вплинути широкий спектр проблем. На діаграмі нижче (рисунок 1.3) показано кілька атак, які використовуються зловмисниками та які можуть призвести до серйозної шкоди окремому додатку або організації в цілому. Знання різних атак, які роблять додаток вразливим, на додаток до потенційних результатів атаки, дає змогу вашій фірмі запобігти усуненню вразливостей та точно перевірити їх.

Виявивши основну причину вразливостей, на ранніх етапах SDLC можна запровадити пом’якшувальні засоби контролю, щоб запобігти будь-яким проблемам. Крім того, знання про те, як працюють ці атаки, можна використовувати для націлювання на відомі об’єкти інтересу під час тестування безпеки веб-додатків.

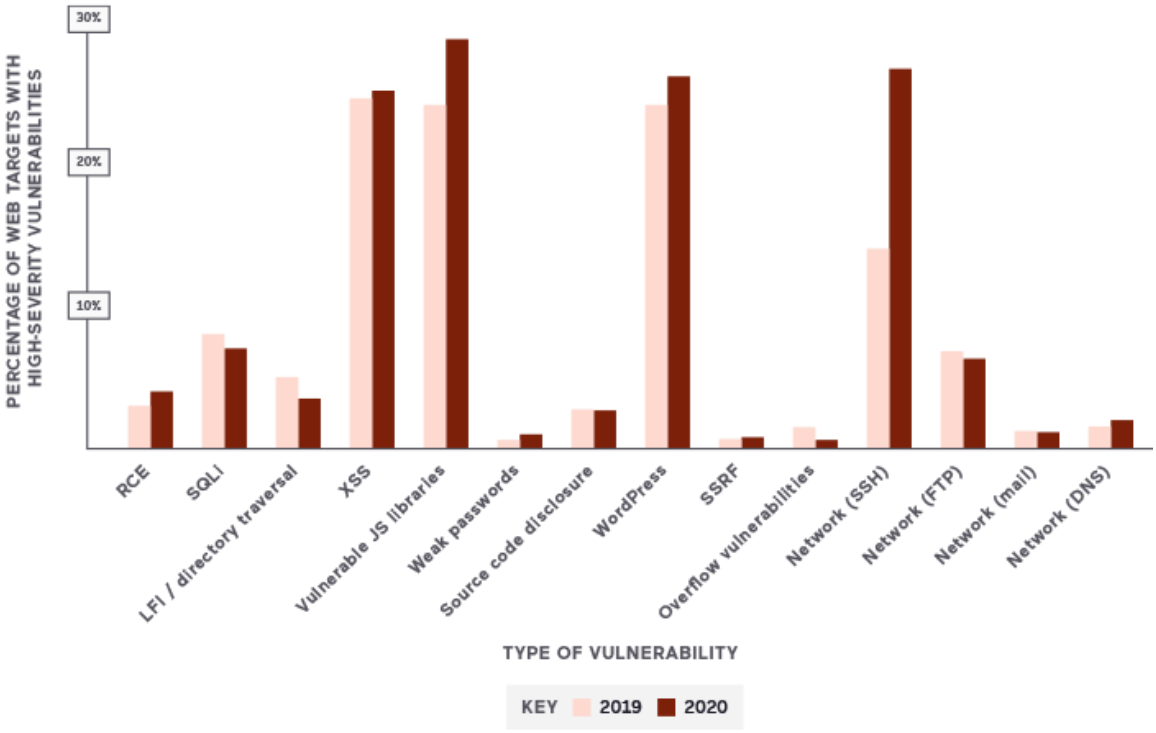

Рисунок 1.3 - Діаграма вразливостей веб-додатків

Розпізнавання впливу атаки також є ключем до управління ризиками вашої фірми, оскільки наслідки успішної атаки можна використовувати для оцінки загальної серйозності вразливості. Якщо під час перевірки безпеки виявлено проблеми, визначення їх серйозності дозволить вашій фірмі ефективно визначити пріоритети зусиль щодо усунення. Почніть з критичних проблем і працюйте над проблемами меншого впливу, щоб мінімізувати ризик для вашої компанії.

Перш ніж виявити проблему, оцінка потенційного впливу на кожну програму в бібліотеці додатків вашої фірми може полегшити визначення пріоритетності тестування безпеки програми. Завдяки встановленому списку високопрофільних додатків тестування безпеки web можна запланувати, щоб спершу націлити на критичні програми вашої фірми з більш цілеспрямованим тестуванням, щоб знизити ризик для бізнесу.

Тестування веб-безпеки спрямоване на виявлення вразливих місць у веб-додатках та їх конфігурації. Основною метою є прикладний рівень (тобто те, що працює за протоколом HTTP). Тестування безпеки веб-додатків часто включає надсилання різних типів введення, щоб спровокувати помилки та змусити систему вести себе несподівано. Ці так звані «негативні тести» перевіряють, чи виконує система те, для чого вона не призначена.

Також важливо розуміти, що тестування веб-безпеки полягає не тільки в тестуванні функцій безпеки (наприклад, аутентифікації та авторизації), які можуть бути реалізовані в програмі. Не менш важливо перевірити, чи інші функції реалізовані безпечним способом (наприклад, бізнес-логіка та використання належної перевірки введення та вихідного кодування). Мета полягає в тому, щоб забезпечити безпеку функцій, доступних у веб-додатку.

1.5.2 Процес тестування веб-додатків

Тестування безпеки веб-додатків — це процес тестування, аналізу та звітування про рівень безпеки та/або положення веб-додатка.

Його використовують веб-розробники та адміністратори безпеки для тестування та оцінки надійності веб-додатків за допомогою ручних та автоматизованих методів тестування безпеки. Ключова мета тестування безпеки веб-додатків полягає в тому, щоб виявити будь-які вразливості або загрози, які можуть поставити під загрозу безпеку або цілісність веб-додатків.

Тестування безпеки веб-додатків — це широкий процес, який включає безліч процесів, які дозволяють тестувати безпеку веб-додатків. Це систематичний процес, який починається з визначення та визначення обсягу всієї програми з подальшим плануванням кількох тестів.

Зазвичай тестування безпеки веб-додатків проводиться після розробки веб-додатка. Веб-додаток проходить суворий процес тестування, який включає серію сфабрикованих шкідливих атак, щоб побачити, наскільки добре веб-додаток працює/відповідає. Загальний процес тестування безпеки зазвичай супроводжується звітом у форматі, який містить виявлені вразливості, можливі загрози та рекомендації щодо подолання недоліків безпеки.

Деякі з процесів тестування включають:

- Тестування атаки грубою силою.
- Правила якості паролів.
- Сесійні файли cookie.
- Процеси авторизації користувачів.
- Процеси аутентифікації користувачів.
- SQL-ін'єкція.

1.5.3 Типи тестів безпеки веб-додатків

Динамічний тест безпеки додатків (DAST).

Цей автоматизований тест безпеки додатків найкраще підходить для внутрішніх програм із низьким рівнем ризику, які мають відповідати нормативним оцінкам безпеки. Для програм із середнім ризиком і критичних програм, які зазнають незначних змін, найкращим рішенням є поєднання DAST з деяким ручним тестуванням веб-безпеки на предмет поширених уразливостей.

Статичний тест безпеки додатків (SAST).

Цей підхід до безпеки програм пропонує автоматизовані та ручні методи тестування. Це найкраще для виявлення помилок без необхідності виконувати програми у виробничому середовищі. Це також дозволяє розробникам сканувати вихідний код і систематично знаходити та усувати вразливості безпеки програмного забезпечення.

Тест на проникнення.

Цей ручний тест безпеки програми найкраще підходить для критичних програм, особливо тих, які зазнають серйозних змін. Оцінка включає бізнес-логіку та тестування на основі супротивників, щоб виявити розширені сценарії атак.

Самозахист додатків під час виконання (RASP).

Цей підхід до безпеки додатків, що розвивається, охоплює ряд технологічних прийомів для *інструментування* програми, щоб можна було відстежувати атаки під час їх виконання та, в ідеалі, блокувати в режимі реального часу.

1.6 Web application firewall (WAF)

Брандмауери веб-додатків (WAF) — це апаратні та програмні рішення, які використовуються для захисту від загроз безпеки додатків. Ці рішення призначені для перевірки вхідного трафіку, щоб блокувати спроби атаки, таким чином компенсуючи будь-які недоліки очищення коду.

Захищаючи дані від крадіжки та маніпуляцій, розгортання WAF відповідає ключовим критеріям сертифікації PCI DSS. Вимога 6.6 стверджує, що всі дані власників кредитних і дебетових карток, які зберігаються в базі даних, мають бути захищені.

WAF зазвичай інтегруються з іншими рішеннями безпеки для формування периметра безпеки. Вони можуть включати послуги захисту від розподіленої відмови в обслуговуванні (DDoS), які забезпечують додаткову масштабованість, необхідну для блокування масових атак. Приинцип роботи WAF показаний на рисунку 1.4.

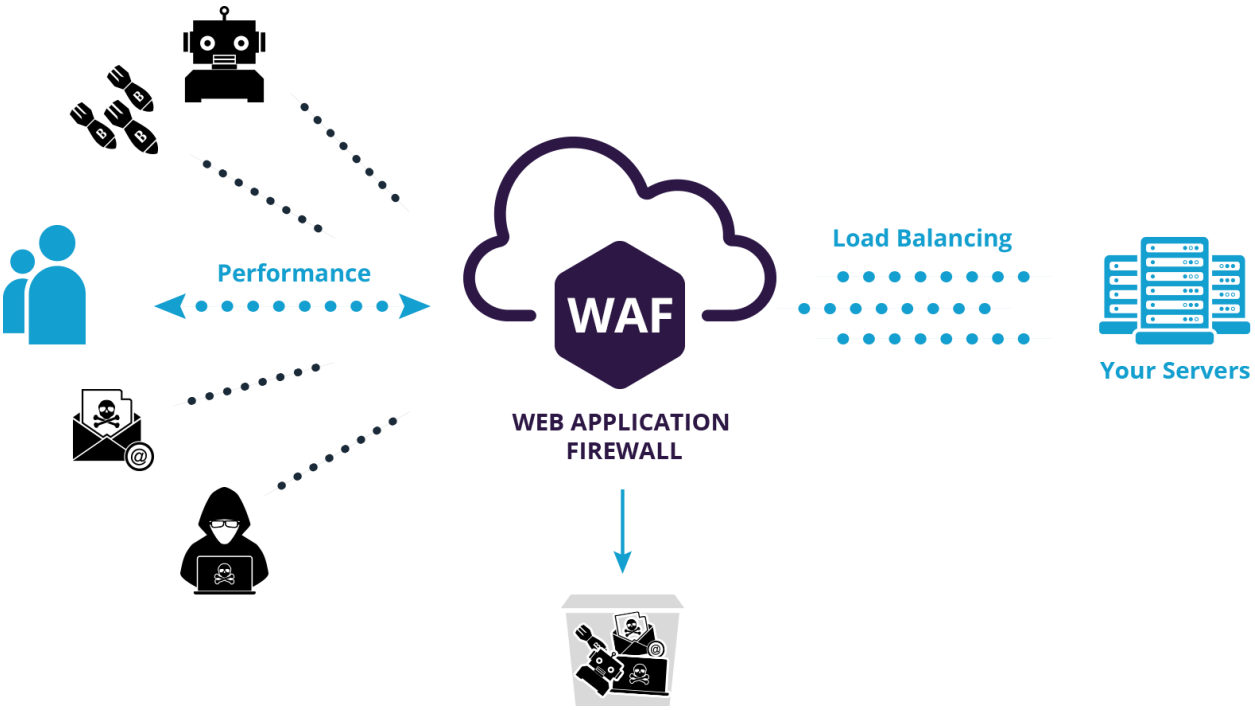

Рисунок 1.4 - Демонстрація роботи фаєрвола веб-додатків

Як правило, для розгортання WAF не потрібно вносити будь-які зміни в програму, оскільки вона розміщується попереду своєї DMZ на краю мережі. Звідти він діє як шлюз для всього вхідного трафіку, блокуючи шкідливі запити, перш ніж вони зможуть взаємодіяти з програмою.

WAF використовують кілька різних евристик, щоб визначити, якому трафіку надається доступ до програми, а який потрібно відсіяти. Постійно оновлюваний пул сигнатур дозволяє їм миттєво ідентифікувати поганих акторів і відомих векторів атак.

Майже всі WAF можна налаштувати спеціально для конкретних випадків використання та політики безпеки, а також для боротьби з виникаючими загрозами (наприклад, нульового дня). Нарешті, більшість сучасних рішень використовують дані про репутацію та поведінку, щоб отримати додаткове уявлення про вхідний трафік.

РОЗДІЛ 2 СТАНДАРТНІ МЕТОДИ ЗАХИСТУ ВІД НАЙПОШИРЕНІШИХ ТИПІВ АТАК

Члени Web Application Security Consortium створили проект для розробки та популяризації стандартної термінології опису цих проблем. Це дасть можливість розробникам додатків, фахівцям в області безпеки, виробникам програмних продуктів і аудиторам використовувати єдину мову для взаємодії.

У багатьох організацій Web-додатки використовуються як критично важливі системи, які повинні щодня обслуговувати багатомільйонні транзакції. Однак справжня цінність Webсайтів повинна оцінюватися на основі потреб кожної організації. Важливість чого-небудь досить важко уявити тільки у вигляді певної суми.

Вразливості в Web-додатках досить давно становили небезпеку для користувачів. Після ідентифікації вразливості для здійснення атаки використовується одна з кількох технік. Зазвичай на ці техніки посилаються як на класи атак (методи використання вразливостей). Багато хто з цих класів мають поширені назви, наприклад «переповнення буферу» (Buffer Overflows), «впровадження коду SQL» (SQL Injection), і «міжсайтового виконання сценаріїв» (Cross-site Scripting). Ці класи атак будуть використані в якості основи для опису та класифікації загроз Web-додатків.

Даний документ містить компіляцію відомих класів атак, які представляли загрозу для Web-додатків в минулому і представляють зараз. Кожному класу атак присвоєно стандартну назву і описані його ключові особливості. Класи організовані в ієрархічну структуру. Створення класифікації загроз безпеки Web-додатків є важливою подією для розробників додатків, фахівців у галузі безпеки, виробників програмних продуктів та інших сторін, які займаються безпекою Web.

На основі класифікації надалі можуть бути створені методики обстеження додатків, рекомендації з розробки додатків з урахуванням безпеки, вимоги до продуктів і служб.

За останні кілька років індустрія безпеки web-додатків адаптувала кілька десятків плутаних і езотеричних термінів, що описують уразливості. Такі назви вразливостей, як «міжсайтового виконання сценаріїв» (Cross-site Scripting), «підробка параметрів» (Parameter Tampering), і «отруєння печива» (Cookie Poisoning) не точно визначають суть проблеми і можливі наслідки атак. Приміром, наявність уразливості типу міжсайтового виконання сценаріїв (Cross-site Scripting) може призвести до викрадення значень cookie користувача. Знання значень cookie дає зловмисникові можливість перехопити сесію користувача і отримати контроль над його обліковим записом. Для експлуатації цієї уразливості використовується метод маніпуляції параметрами вводу та підробка параметрів URL. Наведений сценарій атаки може бути описаний з використанням різних жаргонізмів. Цей складний і мінливий словник часто викликає проблеми і розбіжності у відкритих форумах, навіть якщо сторони згодні з основною ідеєю.

Протягом довгих років не існувало ресурсу, який би описував уразливості в Web-додатках у досить повної і стандартною формою. Коли неофіт безпеки Web-додатків приступає до навчання, його швидко вводить в оману відсутність стандартної мови. Ця ситуація затуманює і без того незміцнілі знання і уповільнює розуміння картини в цілому. Нам необхідний формальний, стандартизований інструмент для обговорення вразливостей, якщо ми збираємося підвищувати рівень захищеності Web-додатків.

2.1 Підбір (brute-force attack)

2.1.1 Опис атаки

Дана атака відбувається на етапі автентифікації. Дана атака спрямована на використовувані Web-додатком методи перевірки ідентифікатора користувача, служби або програми. Аутентифікація використовує як мінімум один з трьох механізмів (факторів): "щось, що ми маємо", "щось, що ми знаємо" або "щось, що ми є". У цьому пункті описується атака, спрямована на обхід та експлуатацію вразливостей в механізмах реалізації аутентифікації Web-серверів.

Атака грубої сили (також відома як злом грубої сили) це кібератака суть якої полягає у підборі логіна та пароля так, щоб врешті-решт знайти потрібний.

Атаки грубою силою прості та надійні. Зловмисники дозволяють комп’ютеру виконувати роботу за них – наприклад, пробувати різні комбінації імен користувачів та паролів – поки не знайдуть той, який працює. Спіймати та нейтралізувати атаку грубої сили, яка триває, є найкращою протидією: як тільки зловмисники мають доступ до мережі, їх набагато важче зловити.

Типи атак:

- Найпростішою атакою грубої сили є *атака за словником*, коли зловмисник працює зі словником можливих паролів і пробує їх усі. Словникові атаки починаються з деяких припущень про поширені паролі, які можна спробувати вгадати зі списку в словнику. Ці атаки, як правило, є дещо застарілими, з огляду на нові й більш ефективні методи. Останні комп’ютери, виготовлені протягом останніх 10 років, можуть грубою силою зламати 8-символьний буквено-цифровий пароль – великі та малі літери, цифри та спеціальні символи – приблизно за дві години. Комп’ютери настільки швидкі, що можуть грубою силою розшифрувати слабкий хеш шифрування за лічені місяці. Ці види атак грубої сили

відомі як вичерпний пошук ключів, коли комп'ютер пробує кожну можливу комбінацію всіх можливих символів, щоб знайти потрібну комбінацію.

- *Повторна переробка облікових даних* — це ще один тип атаки грубої сили, яка повторно використовує імена користувачів і паролі від інших успішно проведених атак, щоб спробувати проникнути в інші системи.

- *Атака зворотного перебору* використовує звичайний пароль, як-от «password», і згодом намагається змусити ім'я користувача йти разом із цим паролем. Оскільки пароль є одним із найпоширеніших паролів у 2017 році, цей метод є більш успішним, ніж можна подумати.

У атаці грубої сили мало тонкощів, тому зловмисники можуть автоматизувати кілька атак для паралельного виконання, щоб розширити свої можливості пошуку позитивного для них результату.

2.2.2 Найпоширеніші методи захисту

Атакам грубої сили потрібен час для виконання. Деякі атаки можуть зайняти тижні або навіть місяці, щоб здобути хоча б якусь корисну інформацію. Більшість засобів захисту від атак грубої сили передбачає збільшення часу, необхідного для успіху, понад те, що технічно можливо, але це не єдиний захист.

Найпоширенішими методами захисту проти перебору є:

- Збільшити довжину пароля: більше символів означає більше часу на злам.

- Збільшення складності пароля: додаткові параметри для кожного символу також збільшують час для зламування методом грубої сили.

- Впровадження CAPTCHA. Вони запобігають використанню інструментів для атаки грубою силою, як-от John the Ripper, залишаючи мережі, системи та веб-сайти доступними для людей.

- Використання багатофакторної автентифікації: багатофакторна автентифікація додає другий рівень безпеки до кожної спроби входу, що вимагає втручання людини та може зупинити успіх грубої атаки.
- Обмежити спроби входу: атаки грубої сили збільшують лічильник невдалих спроб входу в більшість служб каталогів – хороший захист від атак грубої сили полягає в тому, щоб заблокувати користувачів після кількох невдалих спроб, таким чином анулюючи атаку грубої сили, що триває.

2.2 Слабка валідація відновлення пароля

2.2.1 Опис атаки

Дана атака, як і попередня, пов’язана з етапом автентифікації. Відновлення пароля є важливою частиною обслуговування онлайн-користувачів. Слабка перевірка відновлення пароля - це ситуація, коли програма дозволяє зловмиснику незаконно отримати, змінити або відновити пароль іншого користувача. Веб-сайт вважається таким, що не має ненадійну систему відновлення пароля, якщо інформацію, необхідну для підтвердження ідентичності користувача для відновлення пароля, легко вгадати або можна обійти. Якщо системи відновлення паролів слабкі, їх можна зламати за допомогою атак грубої сили, системних недоліків або секретних запитань, які можна легко вгадати (або підшукати).

Приклад слабкої валідації відновлення пароля

Багато веб-сайтів вимагають від користувача лише вказати свою адресу електронної пошти разом із номером телефону, а зачасту і без нього. Цю інформацію можна легко отримати в Інтернеті. В результаті, інформація для перевірки не є дуже секретною. Крім того, інформація може бути скомпрометована іншими методами, такими як міжсайтові сценарії та фішинг.

2.2.2 Найпоширеніші методи захисту

Найпоширенішими методами захисту відновлення паролю є:

- Підказки паролів. Веб-сайт, який використовує підказки, щоб нагадати користувачеві його пароль, може бути атакований, оскільки підказка сприяє атакам Brute Force. Користувач може мати досить хороший пароль «122277King» з відповідним підказкою на пароль «bday+fav author». З цього натяку зловмисник може зрозуміти, що пароль користувача є комбінацією дня народження користувача та улюбленого автора користувача. Це допомагає значно звузити атаки словника грубою силою на пароль. Оскільки люди бувають різні, підказкою до пароля можу слугувати сам пароль.
- Відповідь на питання. Суть цього підходу полягає у додаванні відповіді на питання на етапі реєстрації Часто, воно виглядає як: “В якому місті ви народилися?”, “За яку футбольну команду вболіваєте?”, “Школу під яким номером Ви відвідували?”. На етапі відновлення паролю обране користувачем питання запитується і перевіряється відповідь. Оскільки зловмисник теж може побачити питання, він може досить легко вгадати/підібрати правильну відповідь.
- Надсилання коду на номер телефону, e-mail користувача. Самий сучасний метод ідентифікації користувача, проте не без недоліків.

Важливо відзначити, що саме підхід з надсиланням коду є рекомендованим на сьогоднішній день. Проте, навіть зараз багато сайтів створюються і з іншими способами ідентифікації користувача, з тих чи інших причин.

2.3 Cross-site scripting (XSS)

2.3.1 Опис атаки

Атаки міжсайтових сценаріїв (XSS) — це тип ін’єкції, під час якої шкідливі сценарії впроваджуються на безпечні й надійні веб-сайти. Атаки XSS відбуваються,

коли зловмисник використовує веб-додаток для надсилання шкідливого коду, як правило, у формі сценарію браузера іншому кінцевому користувачеві. Недоліки, які дозволяють цим атакам досягати успіху, є досить поширеними і трапляються скрізь, де веб-додаток використовує вхідні дані від користувача в отриманих результатах, не перевіряючи чи не кодуючи їх.

Зловмисник може використовувати XSS, щоб надіслати шкідливий сценарій нічого непідозрюючому користувачеві. Браузер кінцевого користувача не може дізнатися, що скрипту не можна довіряти, і виконає його. Оскільки він вважає, що сценарій надійшов із надійного джерела, шкідливий сценарій може отримати доступ до будь-яких файлів cookie, токенів сеансу або іншої конфіденційної інформації, яку зберігає браузер і використовується на цьому сайті. Ці сценарії можуть навіть переписувати вміст сторінки HTML.

Зловмисники можуть експлуатувати різні вразливості. Найбільшого поширення набули два типи атаки:

- Відбита (Reflected XSS) – найпоширеніший непостійний тип атаки, що вимагає виконання певної дії з боку користувача.
- Зберігаючи (Persistent XSS) - постійний тип атаки з використанням шкідливого коду на сервер, не вимагає втручання користувача.

2.3.2 Найпоширеніші методи захисту

Не існує єдиної стратегії для усунення міжсайтових сценаріїв, а різні типи веб-програм вимагають різного рівня захисту. Можна вжити ряд захисних заходів, таких як:

- *Якщо можливо, уникайте використання HTML у вводі.* Один дуже ефективний спосіб уникнути постійних міжсайтових скриптових атак – це заборонити користувачам розміщувати HTML у введених формах. Є й інші

варіанти, які дозволяють користувачам створювати багатий вміст без використання HTML, наприклад редактори Markdown і WYSIWYG.

- *Перевірка введених даних* – перевірка означає впровадження правил, які забороняють користувачеві розміщувати дані у формі, яка не відповідає певним критеріям. Наприклад, вхідні дані, які запитують «Прізвище» користувача, повинні мати правила перевірки, які дозволяють користувачеві подавати лише дані, що складаються з буквено-цифрових символів. Правила перевірки також можна налаштувати, щоб відхиляти будь-які теги або символи, які зазвичай використовуються в міжсайтових сценаріях, наприклад теги «<script>».

- *Дезінфекція даних* – очищення даних подібне до перевірки, але це відбувається після того, як дані вже були опубліковані на веб-сервері, але ще до того, як вони будуть показані іншому користувачеві. Існує кілька онлайн-інструментів, які можуть очищати HTML і відфільтровувати будь-які ін’єкції шкідливого коду.

- *Вживання заходів безпеки для файлів cookie.* Веб-програми також можуть встановлювати спеціальні правила для обробки файлів cookie, які можуть пом’якшити крадіжку файлів cookie за допомогою міжсайтових скриптових атак. Файли cookie можуть бути прив’язані до певних IP-адрес, щоб зловмисники, які виконували міжсайтові сценарії, не могли отримати до них доступ. Крім того, можна створити правила для повного блокування JavaScript доступу до файлів cookie.

- *Встановлення правил WAF* - WAF також можна налаштувати для застосування правил, які запобігатимуть відображенню міжсайтових сценаріїв. Ці правила WAF використовують стратегії, які блокують дивні запити до сервера, включаючи міжсайтові атаки сценаріїв. Cloudflare WAF пропонує установку під ключ і захищає веб-додатки від міжсайтових сценаріїв, DDoS-атак, ін'єкції SQL та інших поширених загроз.

2.4 SQL/No-SQL ін’єкції

2.4. 1. Опис атаки

В даній секції описано атаку, спрямовану на виконання коду на Web-сервері. Більшість серверів використовують дані, передані користувачем, при обробці запитів. Часто ці дані використовуються при складанні команд, що застосовуються для генерації динамічного вмісту. Якщо при розробці не враховуються вимоги безпеки до переданих зі сторони користувача даних, зловмисник отримує можливість виконувати непередбачувані розробниками команди.

Атака з ін’єкцією SQL складається зі вставки або «ін’єкції» SQL-запиту через вхідні дані від клієнта до програми. Успішна ін'єкція SQL може зчитувати конфіденційні дані з бази даних, змінювати дані бази даних (вставляти/оновлювати/видаляти), виконувати операції адміністрування бази даних (наприклад, вимкнути СУБД), відновити вміст певного файлу, наявного у файлі СУБД. системи та в деяких випадках видавати команди операційній системі. Атаки з ін’єкцією SQL – це тип атаки з ін’єкцією, при якій команди SQL вводяться у вхідну площину даних, щоб вплинути на виконання попередньо визначених команд SQL.

Факти про SQL-ін’єкції:

- Атаки з ін'єкцією SQL дозволяють зловмисникам підробити особистість, підробити наявні дані, спричинити проблеми з відмовою, такі як анулювання транзакцій або зміна балансу, дозволити повністю розкрити всі дані в системі, знищити дані або зробити їх недоступними, стати адміністраторами серверної бази даних.
- Ін'єкція SQL дуже поширена в додатках PHP і ASP через поширеність старих функціональних інтерфейсів. Через характер доступних програмних

інтерфейсів програми J2EE та ASP.NET мають меншу ймовірність легко експлуатуватися ін’єкціями SQL.

- Серйозність атак із застосуванням SQL обмежується навичками та уявою зловмисника, а також, в меншій мірі, глибокими контрзаходами захисту, такими як з’єднання з низькими привілеями до сервера бази даних тощо. Загалом, ін’єкція SQL є досить серйозною загрозою.

Бази даних NoSQL надають слабші обмеження узгодженості, ніж традиційні бази даних SQL. Вимагаючи менше реляційних обмежень і перевірок узгодженості, бази даних NoSQL часто пропонують переваги продуктивності та масштабування. Проте ці бази даних все ще потенційно вразливі до ін’єкційних атак, навіть якщо вони не використовують традиційний синтаксис SQL. Оскільки ці атаки ін’єкції NoSQL можуть виконуватися в рамках процедурної мови, а не декларативної мови SQL, потенційний вплив більший, ніж традиційна ін’єкція SQL.

Виклики бази даних NoSQL записуються мовою програмування програми, користувацьким викликом API або форматуються відповідно до загальної конвенції (наприклад, XML, JSON, LINQ тощо). Шкідливий вхід, націлений на ці специфікації, може не ініціювати первинну перевірку очищення програми. Наприклад, фільтрація звичайних спеціальних символів HTML, таких як < > & ; не запобігає атакам на JSON API, де спеціальні символи включають / { } :.

Зараз існує понад 150 баз даних NoSQL, доступних для використання в рамках програми, що надають API різними мовами та моделями відносин. Кожен пропонує різні функції та обмеження. Оскільки між ними немає спільної мови, приклад коду ін’єкції застосовуватиметься не до всіх баз даних NoSQL. З цієї причини кожному, хто тестує атаки ін’єкції NoSQL, потрібно буде ознайомитися з синтаксисом, моделлю даних і основною мовою програмування, щоб створити конкретні тести.

Атаки ін'єкції NoSQL можуть виконуватися в різних областях програми, ніж традиційна ін'єкція SQL. Якщо ін’єкція SQL буде виконуватися в движку баз даних, варіанти NoSQL можуть виконуватися на рівні програми або на рівні бази даних, залежно від використовуваного API NoSQL і моделі даних. Зазвичай атаки ін’єкції NoSQL виконуються, коли рядок атаки аналізується, оцінюється або об’єднується у виклик API NoSQL.

Додаткові таймінгові атаки можуть мати відношення до відсутності перевірок паралельності в базі даних NoSQL. Вони не підпадають під ін’єкційне тестування.

2.4.2. Найпоширеніші методи захисту

До стандартних методів захисту можна віднести:

- Фільтрація рядкових параметрів. Щоб впровадження коду було неможливо, для деяких систем управління базами даних, в тому числі, для MySQL, потрібно брати в лапки всі рядкові параметри. У самому параметрі замінюють лапки на \", апостроф на \', зворотну косу риску на \\ (це називається «екранувати спецсимволи»).
- Усікання вхідних параметрів. Для внесення змін в логіку виконання SQL-запиту потрібно впровадження достатньо довгих рядків. Так, мінімальна довжина такого рядка у наведених вище прикладах становить 8 символів («1 OR 1=1»). Якщо максимальна довжина коректного значення параметра невелика, то одним з методів захисту може бути максимальне усікання значень вхідних параметрів. Наприклад, якщо відомо, що поле id у вищенаведених прикладах може приймати значення не більше 9999, можна «відрізати зайві» символи, залишивши не більше чотирьох.
- Використання плейсхолдерів (підготовлених запитів). Вигляд атак типу «SQL-ін'єкція» можливий, тому що значення (дані) для SQL-запиту передаються

разом із самим запитом. Так як дані не відокремлені від SQL коду, вони можуть впливати на логіку всього виразу. На щастя, SQL пропонує спосіб передачі даних окремо від коду. Такий спосіб називається підготовленими запитами. Виконання підготовлених запитів складається з двох етапів: спочатку формується шаблон запиту - звичайний SQL-вираз, але без дійсних значень, а потім, окремо, передаються значення для цього шаблону. Перший етап називається підготовкою, а другий виразом. Підготовлений запит можна виконувати кілька разів, передаючи різні значення.

РОЗДІЛ 3 УДОСКОНАЛЕННЯ СТАНДАРТНИХ МЕТОДІВ ЗАХИСТУ

В даному розділі буде описано та реалізовано імплементацію удосконалень стандартних методів захисту web-додатків, описаних у попередньому розділі. На клієнтській частині буде використовуватися стандартний стек технологій: HTML, CSS, JS. На серверній частині буде використовуватися NodeJS, в якості серверної мови програмування. Та MongoDB/MySQL в якості СКБД. Для спрощення написання логіки на серверній частині буде використовуватися, дефакто, стандартний фреймворк - Express.js.

Проте, варто відмітити, що удосконалень, представлених у цьому розділі, немає цілісних серед стандартних методів захисту, від описаних у попередньому розділі вразливостей. Тому суть даної роботи полягає в тому, щоб доповнити стандартні методи.

3.1 Удосконалення стандартних методів захисту проти атаки підбору (brute force)

Серед удосконалень, які змогли б захистити веб-сайт від атаки методом перебору можна виділити:

- Використання вайтліста, щоб обмежити доступ до певних сторінок: На практиці, не всі сторінки знаходяться в загальному доступі. Інколи, їхнє відвідування дозволене лише певному колу уповноважених осіб. Вайтліст — це чудовий спосіб обмежити доступ до сторінок, веб-програм, електронної пошти, програм та інших систем лише для окремих користувачів, IP-адрес або доменів. Наприклад, використання дозволеного списку дозволяє вказати, які IP-адреси можуть отримати доступ до ваших сторінок входу. Будь-які спроби доступу за IP-

адресами, відмінними від тих, які ви включили в цей список, будуть автоматично заблоковані.

- Обмежити доступ до URL-адрес автентифікації: Вимогою для атак грубої сили є надсилання облікових даних. Якщо змінити URL-адресу сторінки входу — наприклад, перейти з /wp-login.php на /mysite-login — цього може бути достатньо, щоб зупинити більшість автоматизованих і масових інструментів. На жаль, цей спосіб не працюватиме під час розширених атак, якщо посилання можна вгадати або якщо воно видиме на сторінці, але це простий спосіб запобігти автоматичним атакам.
- Використання 256-бітних ключів шифрування. Так само, як і паролі, ключі шифрування можна зламати за допомогою атак грубої сили. але при використанні 256-бітного шифрування замість стандартного 128-бітного, за допомогою сучасних комп'ютерів знадобиться так багато часу для взлому, що вони вважаються незламними.

3.1.1 Використання вайтліста

Як було описано вище, вайтліст — це чудовий спосіб обмежити доступ до сторінок, веб-програм, електронної пошти, програм та інших систем лише для окремих користувачів, IP-адрес або доменів. Наприклад, використання дозволеного списку дозволяє вказати, які IP-адреси можуть отримати доступ до ваших сторінок входу. Будь-які спроби доступу за IP-адресами, відмінними від тих, які ви включили в цей список, будуть автоматично заблоковані.

Наприклад, на web-ресурсі є сторінка адміністратора з адресою **/admin.** Відповідно, на ній знаходиться форма для логіну адміністратора. І ми хочемо, щоб тільки адміністратори могли її відвідувати. В даному випадку є 2 можливих рішення цієї задачі:

- Внесення дозволених IP-адрес до списку в середині вихідного коду серверної частини веб-додатка.
- Внесення дозволених IP-адрес до бази даних.

Обидва способи мають свої переваги та недоліки. Але, найкращим з точки зору масштабованості, буде другий варіант. Оскільки він не потребує редеплою веб-додатка при додаванні нового IP в список та дозволяє уникнути додавання чутливої інформації в вихідний код. Отже, ми будемо реалізовати варіант саме з використанням бази даних.

Отже, для початку нам потрібна колекція, в якій ми будемо зберігати дозволені адреси, назвемо її **whiteLists**. Далі, нам потрібно придумати модель документа. Назручнішим варіантом буде:

```
_id: {
     _id: string,
     address: string,
     allowedList: string[],
}
```

Ми припускаємо, що в додатку вже реалізована система з додавання IP-адрес в список дозволених і приступимо до написання перевірки, яка захистить додаток від зайвого навантаження при атаці підбором.

Оскільки при переході по даній адресі у користувача повинна відображатися сторінка в браузері (повинен повернутися HTML+CSS+JS), то даний запит є типу GET. Тому саме для цього типу запиту ми додаємо логіку:

```
app.get('/admin', async (req, res, next) => {})
```

Далі, там потрібно отримати IP-адресу користувача, який виконав запит (Лістинг 3.1).

Лістинг 3.1 Отримання IP адреси користувача

```
app.get('/admin', async (req, res, next) => {
 /* отримуємо ip-адресу користувача, що здійснив запит */
```

```
 const ipAddress = req.connection.remoteAddress;
})
```

Тепер, нам потрібно здійснити запит до бази даних та дістати список дозволених адрес для даного роуту (Лістинг 3.2).

Лістинг 3.2 Отримання списку дозволених адрес для даної сторінки

```
app.get('/admin', async (req, res, next) => {
     /* отримуємо ip-адресу користувача, що здійснив запит */
     const ipAddress = req.connection.remoteAddress;
/* шукаємо документ для даної сторінки за її адресою */
     const whiteListDoc = await db.whiteLists.findOne({
'address': '/admin' });
})
```

Після цього, нам залишилося здійснити перевірку ip-адреси користувача використовуючи оператор **if**. У випадку, якщо адреси користувача, який здійснює запит, немає у списку дозволених - потрібно його редірект на домашню сторінку **“/”**, якщо є у списку повернути сторінку за адресою **/admin** (Лістинг 3.3).

Лістинг 3.3 Імплементація використання вайтліста

```
app.get('/admin', async (req, res, next) => {
     /* отримуємо ip-адресу користувача, що здійснив запит */
     const ipAddress = req.connection.remoteAddress;
     /* шукаємо документ для даної сторінки за її адресою */
const whiteListDoc = await db.whiteLists.findOne({ 'address':
'/admin' });
     /* виносимо масив з дозволеними адресами в окрему константу для
зручності */
          const allowedList = whiteListDoc.allowedList;
     /* у випадку, якщо адреси користувача НЕМАЄ у списку дозволених
- здійснюємо редірект на домашню сторінку */
          if (!allowedList.includes(ipAddress)) {
          return res.redirect('/');
          }
     /* повертаємо сторінку адміністратора, якщо перевірка пройдена
*/
          return res.render('admin');
}).
```

Повертаючись до питання реалізації, варіант з використанням IP-адрес в середині вихідного коду буде виглядати майже там само. Єдина відмінність - замість запиту до бази, буде використана константа-масив з коду.

3.1.2 Зміна адреси сторінки

Даний спосіб призначений для захисту від автоматизованих атак підбором. Оскільки зачасту розробники використовують прості влучні назви для адрес сторінок, автоматизований софт для атак налаштований так, щоб переходити по цих сторінках, заповнювати текстові поля та надсилати форму. Даний спосіб - це скоріше підхід, аніж якийсь комплекс дій та логіки. Отже, припустимо, що в нас є сторінка за адресою **/admin:**

```
app.get('/admin', async (req, res, next) => {
     // якась логіка тут
})
```

Для того, щоб запобігти автоматизованим атакам достатньо просто змінити назву адреси, яка буде не настільки очевидною. Це може бути щось просте та примітивне. Наприклад my-admin, mysite-admin, admin-admin. Так і щось зовсім неочевидне, по типу some-random-words. Після зміни, код буде виглядати так:

```
app.get('/my-admin', async (req, res, next) => {
     // якась логіка тут
})
```

3.1.3 Використання 256-бітних ключів шифрування

В даному методі все досить просто, чим більше довжина ключа - тим краще. Стандартний 128-бітний ключ шифрування виглядає так: kYp3s6v9y$B&E)H@. Все що потрібно зробити, це замінити його на ключ довжиною 256-біт. Він виглядає так:
Xp2s5v8y/B?E(G+KbPeShVmYq3t6w9z$. Цього буде достатньо, щоб захиститися від атак на ключ шифрування методом підбору.

3.2 Удосконалення стандартних методів захисту проти слабкої валідації відновлення пароля

Спектр удосконалень до захисту від даної вразливості досить невеликий. Оскільки потрібно щоб процес був надійним, відбувався автоматично і не потребував дій зі сторони власників веб-додатку.

Серед удосконалень можна виділити:

- Збільшення кількості питань. Чим більше питань - тим важче та довше буде підібрати відповідь.
- Використання пін-коду замість секретного питання. Оскільки буде запитуватися саме пін-код, у зловмисника не буде від чого відштовхуватись при спробі вгадати/підібрати правильну відповідь.
- Додавання прив’язки до часу та користувача. У випадку, якщо використовується підхід з кодом на телефон, або пошту користувача, вказану при реєстрації, потрібно звернути увагу на саму реалізацію. Якщо код просто генерується і любий користувач може його використати - це надзвичайно небезпечно. В даному випадку рекомендується здійснити прив’язку коду до конкретного користувача, який запросив зміну паролю. Також слід задати час, протягом якого код буде активним. В противному випадку, зловмисник може просто вгадати його за певний проміжок часу.

3.2.1 Збільшення кількості питань

В даному випадку, все просто. Чим більше запитань - тим більше часу піду на пошук відповідей. Зловмиснику недостатньо буде знайти відповідь на якесь одне питання. Зазвичай, використовується відповідь на одне єдине запитання, тому форма реєстрації виглядає приблизно так (Рис. 3.1).

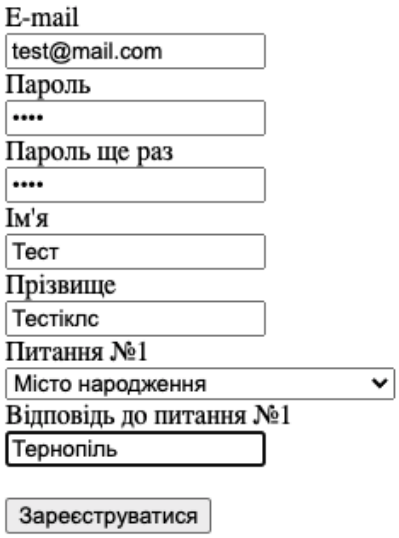

Рисунок 3.1 - Типова форма реєстрації з одним питанням

Для того, щоб застосувати збільшену кількість питань на етапі відновлення пароля, спочатку потрібно добавити їх на формі реєстрації (Рис 3.2). Слід відмітити, що в реалізації важливо уникнути повторення запитань. Щоб обрані запитання були унікальні в межах одного користувача. Для відповіді на питання бажано задати мінімальну довжину 2 символи, адже це саме та кількість символів з якої починаються найкоротші найменування. Наприклад, назва гурту U2.

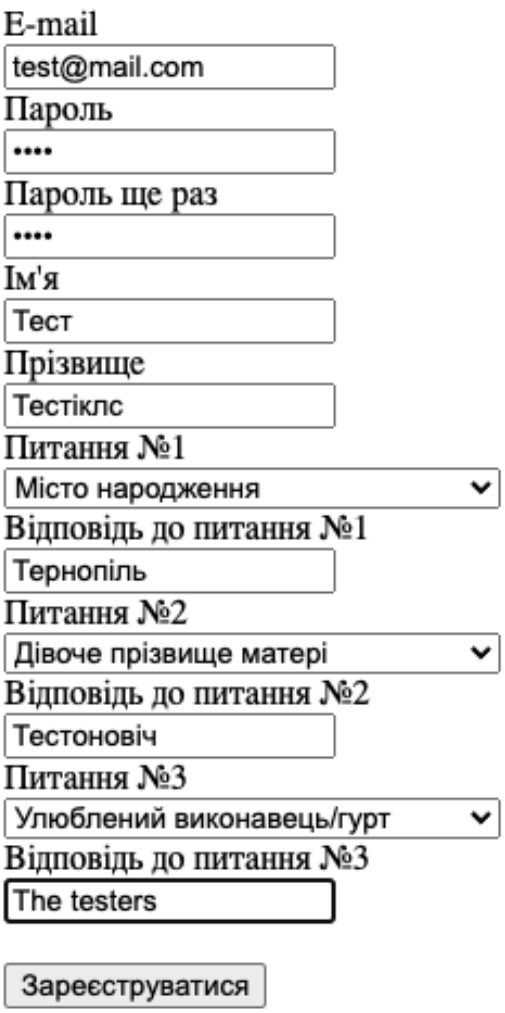

Рисунок 3.2 - Форма реєстрації зі збільшеною кількістю питань

Також, важливим етапом реалізації є винесення питань в окрему колекцію, щоб потім посилатися на неї з документу користувача. Модель даних для питання буде надзвичайно проста:

```
_id: {
     _id: string,
     name: string,
}
```

В такому разі, модель даних користувача буде мати наступний вигляд:

```
_id: {
     _id: string,
     …
```

```
        questions: {
            questionId:                                    string,
            value: string,
        }[]
    }
```

Тепер, коли ми зберігаємо питання та відповіді на них в базі, ми можемо перейти до етапу відновлення пароля (Рис. 3.3).

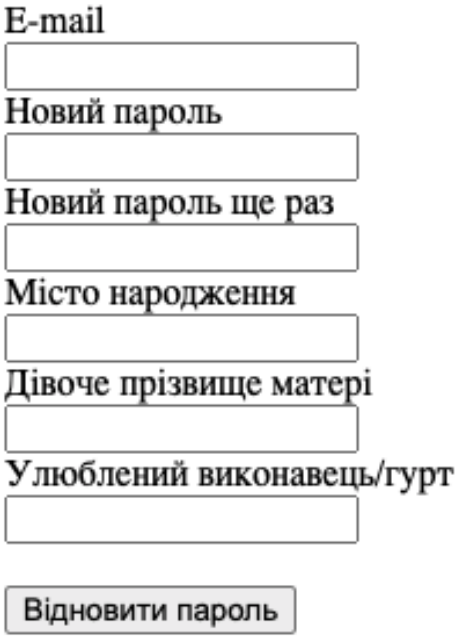

Рисунок 3.3 - Форма відновлення пароля з обраними користувачем питаннями

Далі - все просто. На сервері, в функції яка здійснює перевірку правильності введених даних потрібно порівняти відповіді з запиту, надісланого користувачем, з відповідями, які зберігаються в документі юзера в базі даних (Лістинг 3.4).

Лістинг 3.4 Імплементація валідації правильності введених користувачем відповідей на три питання

```
async function check(req, res, next) {
    /* Дістаємо надіслані користувачем поля з тіла запиту */
    const { email, password, passwordRepeat, answers } = req.body;
    const user = await db.users.findOne({ email: email });
```

```
/* Повертаємо помилку, якщо користувача з такою адресою не
знайдено */
if (!user) {
     return res.status(404).send('Користувача з таким e-mail не
     знайдено')
}

const questions = user.questions;
/* використовуючи метод для роботи з масивами reduce проходимось
по запитанням користувача */
const answersValid = questions.reduce((acc, curr) => {
     /* дістаємо _id запитання та правильну відповідь з масиву
     питань документа користувача */
     const { questionId, value } = curr;
     /* В даному місці ми за поточним, на даній ітерації, _id
     запитання, дістаємо надіслану користувачем відповідь. У
     випадку, якщо хоча б на одній ітерації відповідь на запитання
     буде неправильна - в кінцевому результаті повернеться
     значення false, яка означатиме, що відповідь на якесь
     питання була невірною */
     return acc && answers[questionId] === value;
 }, true);
/* у випадку, якщо якась відповідь на питання була неправильною,
повертаємо помилку .*
 if (!answersValid) {
     res.status(403).send('Дана неправильна відповідь на одне з
     питань')
     }
     /*
     якась логіка далі...
     */
}
```

Спробуємо здійснити відновлення пароля, вказавши одну відповідь не правильно. Наприклад, вкажемо неправильне місто народження (Рис. 3.4).

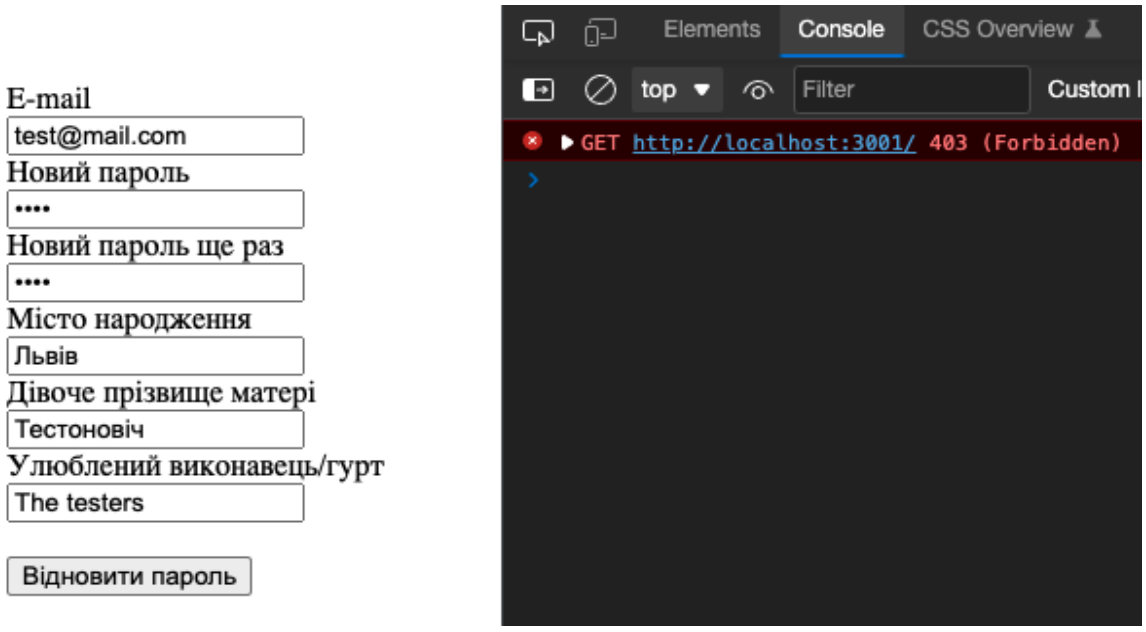

Рисунок 3.4 - Результат відправки форви з неправильною відповіддю на одне з запитань

Якщо ми поглянемо у кладку “Network”, то побачимо, що ми отримали очікувану відповідь, а саме: “Дана неправильна відповідь на одне з питань” (Рис. 3.5).

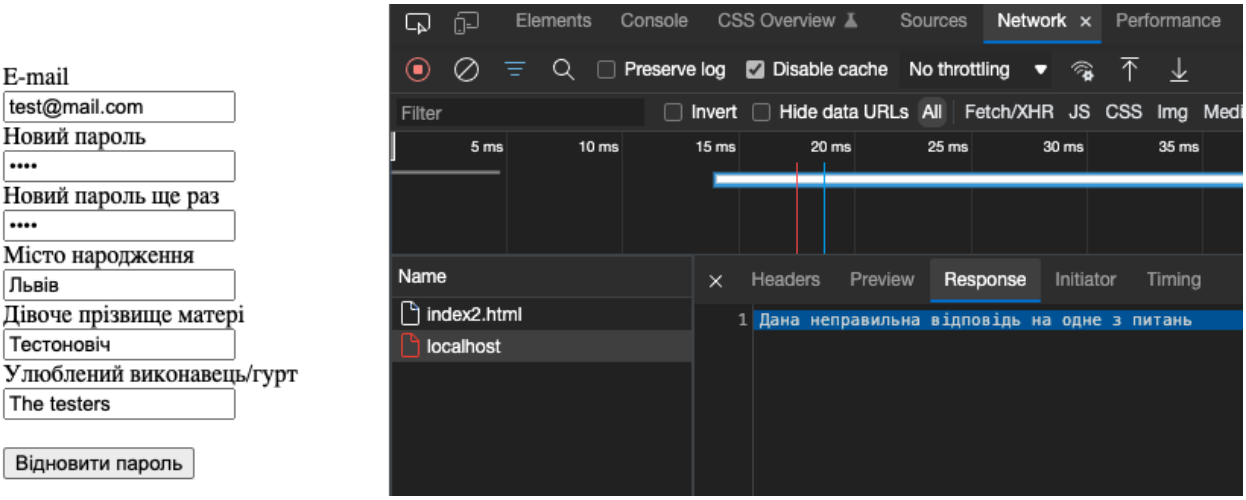

Рисунок 3.5 - Відображення помилки у вкладці “Network”

Отже, все готово. Тепер у користувача запитуються 3 питання. І якщо хоча б одна відповідь неправильна - сервер поверне помилку, а пароль не відновиться.

3.2.2 Використання пін-коду

Суть даного методу полягає у використанні пін-коду з певної кількості цифр. Якщо конкретне питання відразу дає зловмиснику конкретну відповідь, або певний можливий діапазон відповідей, то використання пін-коду не дає жодних конкретних ідей. Проте, слід відмітити, що це також залежить від користувача. Якщо він якимось чином використає цифри з дати свого народження, то дану комбінацію можна дуже легко вгадати. Як альтернатива, можна згенерувати пін-код для користувача, але розглядати даний варіант у цьому розділі ми не будем.

В даному випадку модель користувача буде виглядати так:

```
_id: {
     _id: string,
     …
     pin: number,
}
```

З даною модифікацією форма буде виглядати так (Рис 3.6).

Рисунок 3.6 - Форма реєстрації з полем для

Відповідно, форма відновлення пароля набуде наступного вигляду (Рис. 3.7).

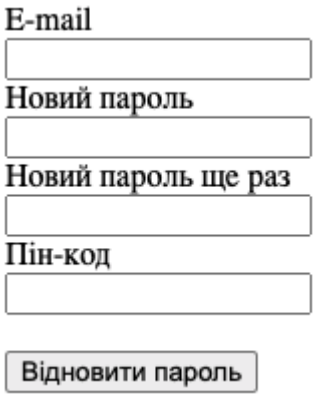

Рисунок 3.7 - Форма відновлення пароля з полем для введення пін-кода

Перевірка пін-коду буде ідентичною з описаною вище перевіркою правильності відповіді на питання (Лістинг 3.5).

Лістинг 3.5 Імплементація використання пін-кода для аутентифікації користувача

```
async function check(req, res, next) {
 /* Дістаємо надіслані користувачем поля з тіла запиту */
```

```
 const { email, password, passwordRepeat, pin } = req.body;
     const user = await db.findOne({ email: email });
/* Повертаємо помилку, якщо користувача з такою адресою не знайдено
*/
     if (!user) {
return res.status(404).send('Користувача з таким e-mail не знайдено')
     }

     const userPin = user.pin;
     /* у випадку, якщо пін-код неправильний - повертаємо помилку */
     if (userPin !== pin) {
          res.status(403).send('Неправильний пін-код');
     }
     /*
     якась логіка далі...
     */
}
```

3.2.3 Додавання прив’язки до часу та користувача

Надсилання коду на телефон чи e-mail та надсилання посилання на відновлення паролю, є найефективнішими засобами для захисту відновлення пароля. Але й ця технологія може бути вразливою до атак. Тому, важливо звернути увагу на деталі реалізації. Для того, щоб максимально удосконалити дану технологію потрібно забезпечити прив’язку до конкретного користувача.

В даному розділі ми будемо розглядати підхід з надсиланням посилання на відновлення пароля користувача.

Для реалізації даного підходу потрібно створити нову колекцію, назвемо її **codes.** Модель документа, зі згаданими вище удосконаленнями, буде виглядати так:

```
_id: {
     userId: string,
```

```
    expirationDate: Date,
    code: number,
}
```

При використанні даного метода, відновлення паролю буде проведене у 2 етапи:

1. Введення e-mail адреси користувача, що бажає відновити пароль
2. Перехід по посиланню, яке веде на форму введення та повторення нового пароля

Форма відновлення пароля буде виглядати наступним чином (Рис 3.8)

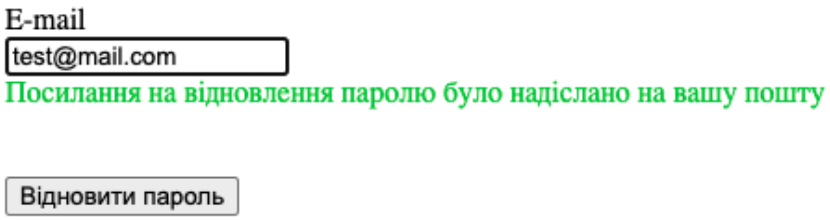

Рисунок 3.8 - Форма з введенням email адреси користувача, який бажає відновити пароль

Далі, на серверній частині, ми повинні обробити цей запит, а саме:

- Перевірити існування користувача з такою email адресою
- Згенерувати документ вищезгаданої колекції codes для даного користувача
- Створити посилання, яке буде вести на форму з введенням нового пароля. Та яке буде містити цей самий код, або в самій url-адресі, або в самій формі. В прикладі ми будемо використовувати перший підхід
- При обробці нового паролю, потрібно перевірити чи не вийшов термін придатності коду та з його допомогою дістати користувача, якому потрібно встановити новий пароль

Отож почнемо. Генерація коду для користувача буде мати наступний вигляд (Лістинг 3.6).

Лістинг 3.6 Імплементація генерації коду підтвердження з прив’язкою до часу та користувача

```
async function generatePasswordRestorationCode(req, res, next) {
     /* Дістаємо надіслані користувачем поля з тіла запиту */
     const { email } = req.body;
     const user = await db.findOne({ email: email });
/* Повертаємо помилку, якщо користувача з такою адресою не знайдено
*/
     if (!user) {
return res.status(404).send('Користувача з таким e-mail не знайдено')
 }
 /* створюємо теперішню дати */
     const expirationDate = new Date();
/* збільшуємо її на 5 хвилин, стільки часу буде діяти наш код
відновлення пароля */
     expirationDate.setMinutes(expirationDate.getMinutes() + 5);
/* створюємо об'єкт, який ми вставимо в базу даних, щоб потім
використати на етапі встановлення нового пароля */
     const codeDoc = {
          userId: user._id,
          expirationDate,
          code: generateRandomCode(),
     }
 /* вставляємо документ в базу */
 await db.codes.insert(codeDoc);
/* Створюємо новий інстанс URL для спрощення роботи з посиланнями */
 const url = new URL('http://localhost:3001/reset-password');
     /* Добавляємо код відновлення пароля в url-адресу */
```

```
    url.searchParams.append('code', codeDoc.code);
    /* надсилаємо лист з посилання на пошту користувачу */
    sendEmail({
          to: email,
        subject: 'Відновлення пароля',
        body: url,
    })
}
```

Форма введення нового пароля, яку користувач попаде після переходу по посиланню буде виглядати так (Рис. 3.9).

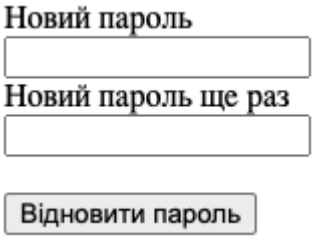

Рисунок 3.9 - Форма введення нового пароля

Тепер нам залишається тільки перевірити валідність коду, який надійде разом з запитом та встановити користувачу новий пароль (Лістинг 3.7).

Лістинг 3.7 - Імпелементація валідації коду прив’язаного до часу та користувача

```
async function handlePasswordReset(req, res, next) {
 /* Дістаємо надіслані користувачем поля з тіла запиту */
 const { code, password, passwordRepeat } = req.body;
 const codeDoc = db.codes.findOne({ code: code });

 if (!codeDoc) {
```

```
   /* якщо такого коду не існує - повертаємо повідомлення про помилку
*/
   return res.status(404).send('Код відновлення не знайдено');
 }
 /* Дістаємо властивості об'єкта для зручності */
 const { userId, expirationDate } = codeDoc;
 const user = await db.users.findOne({ _id: userId });

 if (!user) {
   /* якщо такого користувача не існує - повертаємо повідомлення про
помилку */
   return  res.status(404).send('Користувача  не  знайдено,  або  він
видалений');
 }

 if (expirationDate < new Date) {
   /* Якщо час дії посилання вийшов - повертаємо повідомлення про
помилку */
   return res.status(400).send('Термін дії посилання на відновлення
пароля вичерпано');
 }

 /* Перевіряємо паролі і т.д. */
}
```

Все готово. Тепер відновлення паролю надійне і готово до використання на продакшн рівні.

3.3 Удосконалення стандартних методів захисту від XSS

До удосконалень можна віднести:

- Використання сучасних фреймворків/бібліотек для побудови користувацьких інтерфейсів. Досить несподіване рішення. Проте, не менш ефективне. Наприклад, в бібліотеці React, JSX запобігає вставці зловмисного коду. За замовчуванням, React DOM екранує будь-які значення, що включені в JSX, перед їх рендерингом. Таким чином, це гарантує, що ви ніколи не включите в код те, що явно не написано у вашому додатку. Перед виводом все перетворюється на рядок. Це допомагає запобігти атакам XSS (cross-site-scripting).
- Дезінфекція даних, що будуть додані в CSS в якості URL атрибуту.
- Оскільки скрипт користувач може в якості URL-адреси використати відносний, а не абсолютний шлях, це може стати серйозною вразливістю. Потрібно здійснити дезінфекцію. В першу чергу це стосується атрибутів, значенням яких може бути URL (наприклад backgroundImage).
- Дезінфекція даних при використанні element.innerHTML, element.outerHTML, document.write(...) та їх еквівалентів. Дані властивості та методи є надзвичайно вразливими до різного роду атак, основною з яких є XSS. Оскільки ці методи виконують абсолютно любий переданий код.

3.3.1 Використання сучасних фреймворків та бібліотек для побудови користувацьких інтерфейсів

Отже, як було описано вище, в даному випадку все дуже просто. Використання сучасних бібліотек та фреймворків для побудови користувацьких інтерфейсів дійсно може допомогти в окремих випадках. Візьмемо до прикладу бібліотеку React. В офіційній документації бібліотеки пише наступне: “За замовчуванням, React DOM екранує будь-які значення, що включені в JSX, перед

їх рендерингом. Таким чином, це гарантує, що ви ніколи не включите в код те, що явно не написано у вашому додатку. Перед виводом все перетворюється на рядок. Це допомагає запобігти атакам XSS (cross-site-scripting).”

Спробуємо переконатися в цьому в доволі просто сценарії (Лістинг 3.8)

Лістинг 3.8 Додавання шкідливого коду в користувацький текст

```
/* створюємо константу, яка являє собою скрипт у вигляді стрічки */
const someSuspiciousText = "<script>alert('U R HACKED!')</script>";
/* вставляємо цю константу в JSX */
const App = () => <div>{someSuspiciousText}</div>;
```

Тепер спробуємо відкрити сторінку з даним скриптом. Як результат, скрипт не виконався й відображається на сторінці як простий текст. Як і було описано в документації бібліотеки (Рис. 3.10).

Рисунок 3.10 - Виведення небезпечного скрита в якості просто тексту бібліотекою React

Якщо ми відкриємо інструменти розробника, то ми ще раз можемо переконатися, що скрипт був вставлений в якості звичайно тексту, а не як HTML-елемент (Рис 3.11).

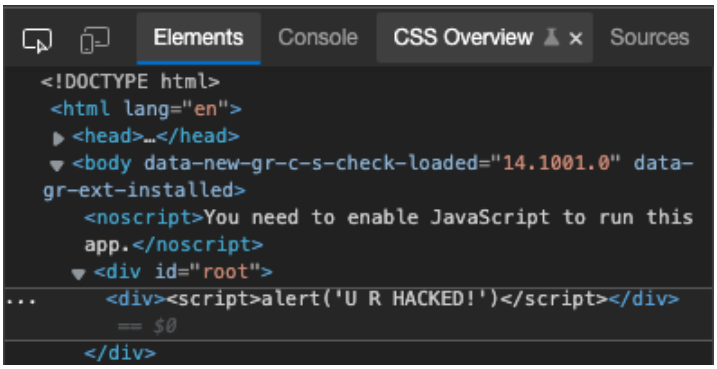

Рисунок 3.11 - Інструменти розробника в яких скрипт вставлений як простий текст

3.3.2 Дезінфекція даних, що будуть додані в CSS в якості URL атрибуту

Зазвичай виконання JavaScript із контексту CSS вимагає або передачі javascript:attackCode() до методу CSS url() або виклику методу CSS expression(), який передає код JavaScript для безпосереднього виконання.

З мого досвіду, виклик функції Expression() з контексту виконання (JavaScript) був вимкнений. Щоб пом’якшити вплив методу CSS url(), нам потрібно переконатися, що URL-адреса кодує дані, передані методу CSS url().

Частою практикою є те, що користувачі можуть вказувати url-адреси на свої аватари в профілях, фонові зображення і тому подібне. Тому потрібно здійснювати дезінфекцію даних перед тим, як встановити значення атрибуту. Наступний код буде використовувати відносну URL-адресу, яка може відобразити конфіденційну інформацію з серверної машини (Лістинг 3.9).

Лістинг 3.9 - Імплементація XSS атаки через css атрибут

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="UTF-8">
```

```
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="#avatar" style="height: 400px; width: 500px;">
 </div>
 <script>
   const maliciousUserUrl = './secret-file.png';
   const avatarElement = document.getElementById('#avatar');

   avatarElement.style.backgroundImage = `url(${maliciousUserUrl})`;
 </script>
</body>
</html>
```

Відповідно, результат виконання буде наступним (Рис. 3.12).

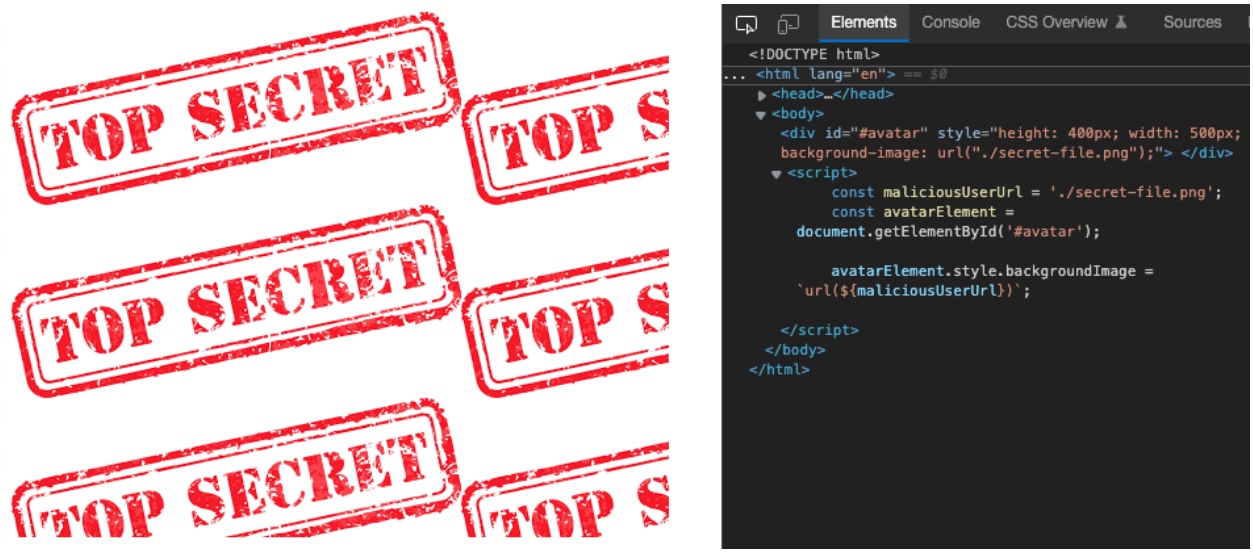

Рисунок 3.12 - Виведення чутливої інформації шляхом використання відносної URL-адреси

Є багато рішень даної проблеми, ми скористаємось найефективнішим з них. Ми здійснюємо перевірку регулярним виразом і у випадку, якщо URL-адреса користувача не пройде перевірку - встановимо дефолтне зображення-плейсхолдер в якості аватара користувача.

З даною модифікацією код буде виглядати так (Лістинг 3.10).

Лістинг 3.10 - Імплементація захисту від XSS атаки шляхом використання регулярних виразів

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport"
       content="width=device-width,    user-scalable=no,    initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
</head>
<body>
 <div id="#avatar" style="height: 400px; width: 500px; background-
position: center; background-size: cover; background-repeat: no-
repeat">
 </div>
 <script>
   const maliciousUserUrl = './secret-file.png';
   const placeholderImageUrl = 'https://monstar-lab.com/global/wp-
content/uploads/sites/11/2019/04/male-placeholder-image.jpeg';
   const avatarElement = document.getElementById('#avatar');
   const expression = /(https?:\/\/(?:www\.|(?!www))[a-zA-Z0-9][a-zA-
Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|www\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
```

```
9]\.[^\s]{2,}|https?:\/\/(?:www\.|(?!www))[a-zA-Z0-
9]+\.[^\s]{2,}|www\.[a-zA-Z0-9]+\.[^\s]{2,})/gi;
   let userAvatar;
   if (maliciousUserUrl.match(expression)) {
     userAvatar = maliciousUserUrl;
   } else {
     userAvatar = placeholderImageUrl;
   }
   avatarElement.style.backgroundImage = `url(${userAvatar})`;
 </script>
</body>
</html>
```

Результат даної модифікації виглядає наступним чином (Рис. 3.13).

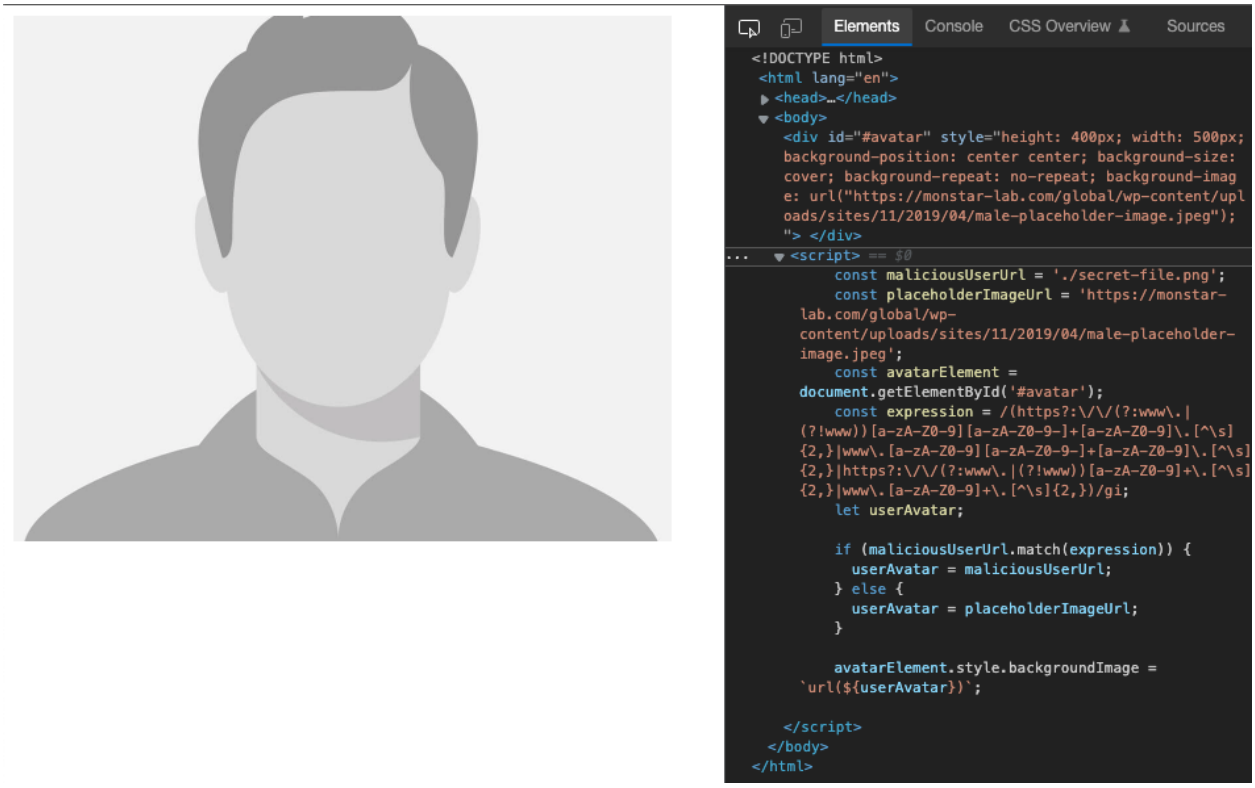

Рисунок 3.13 - Використання картинки-плейсхолдера при невдалій валідації картинки користувача, з відносною URL-адресою

3.3.3 Дезінфекція даних при використанні element.innerHTML, element.outerHTML, document.write(...) та їх еквівалентів.

Вище перечислені методи не рекомендуються до використання в коді, за виключенням декількох певних ситуацій, коли це дійсно потрібно. Одним з прикладів, коли це дійсно потрібно - збереження та використання користувацьких HTML-шаблонів. Наприклад на сайті є можливість створення нотаток з різним форматуванням користувачами. В даному випадку, в базі буде зберігатися саме HTML. Загрозами в даному випадку є тег <script> та <img>, які можуть бути добавлений зловмисником і виконуватися при відкритті даної нотатки іншими користувачами. Щоб переконатися, що вищезгадані методи небезпечні, перевіримо їх на практиці. В прикладі використаємо саме тег <img>, оскільки це досить неочікувана реалізація даної атаки (Лістинг 3.11).

Лістинг 3.11 - Імплементація XSS атаки через метод innerHTML

```
<script>
/* Зберігаємо елемент DOM в константу */
const userHtmlContainer = document.getElementById('#user-html');
/* Створюємо стрічку з потенційно небезпечними тегами */
const customUserHtml = `
     <h1>Привіт!</h1>
     <p>Тебе хакнуто!</p>
     <img src=x onerror="alert('Вас успішно хакнули XSS атакою!')">
   `;
/* Вставляємо користувацькі теги в DOM */
userHtmlContainer.innerHTML = customUserHtml;
</script>
```

Відповідно, результат буде виглядати наступним чином (Рис. 3.14).

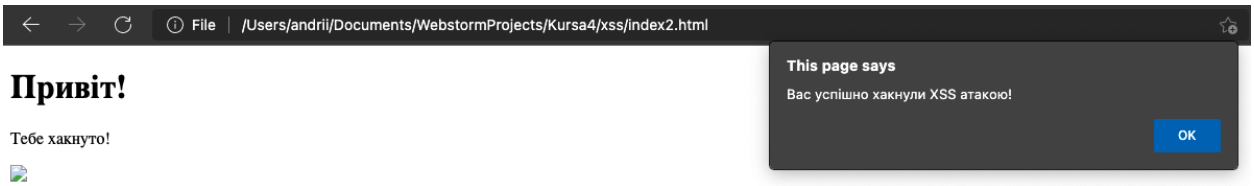

Рисунок 3.14 - Використання тегу <img> для виконання XSS атаки

Також, слід відзначити, що останні версії деяких сучасних браузерів автоматично стерилізують тег <script> при використанні .innerHTML() та інших методів. Проте, все рівно потрібно потурбуватися, щоб вилучити його зі стрічки до того, як добавити його в DOM.

Отже, найпростішим, та найефективнішим рішенням буде використання спеціалізованих бібліотек для вилучення небажаних тегів зі стрічки. В даному прикладі, ми використаємо бібліотеку від ClaudFlare - авторитетної компанії, яка надає послуги з захисту та оптимізації роботи серверів по всьому світу. З використанням даної бібліотеки, код виглядатиме так (Лістинг 3.12).

Лістинг 3.12 - Імплементація захисту від XSS атаки шляхом дезінфекції даних

```
 <!-- Підключаємо бібліотеку -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/sanitize-html/1.27.1/sanitize-html.min.js" integrity="sha512-xNM601PwoBDQJcBQ3ATPbr8X/MS2K81vh/f0/VwYkYA8yofd3Y89E5VBWX+9iT0YfqiwOW+/eWr2JUBmxQGbdQ==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
 <script>
 /* Зберігаємо елемент DOM в константу */
 const userHtmlContainer = document.getElementById('#user-html');
/* Створюємо стрічку з потенційно небезпечними тегами */
const customUserHtml = `
```

```
<h1>Привіт!</h1>
<p>Гарного дня!</p>
<img src=x onerror="alert('Вас успішно хакнули XSS атакою!')">
`;
/* Використовуючи пакет для дезінфекції HTML дезінфікуємо потенційно
небезпечну стрічку */
const sanitizedHtml = window.sanitizeHtml(customUserHtml);
/* Абсолютно безпечно вставляємо користувацькі теги в DOM */
userHtmlContainer.innerHTML = sanitizedHtml;
</script>
```

Тепер перевіряємо результат (Рис. 3.15).

Рисунок 3.15 - Відсутність модального вікна з повідомленням про XSS атаку, при використанні допоміжної бібліотеки для дезінфекції HTML.

Як можна побачити зі скріншота, модальне вікно не з’явилось, а отже бібліотека дійно вилучила потенційно небезпечні теги.

3.4 Удосконалення стандартних засобів захисту від SQL-ін’єкцій

До удосконалень можна віднести:

- Приведення до цілочисельного типу. В SQL-запити часто підставляються цілочислові значення, отримані від користувача. Також, дуже часто використовується ідентифікатор, у вигляді цілого числа. Його можна примусово привести до типу Int. Так ми виключимо появу в ньому небезпечних виразів. Якщо

хакер передасть у цьому параметрі замість числа SQL код, результатом приведення ціле число, або помилка. В разі якщо це нуль, то логіка всього SQL-запиту не зміниться. Якщо результатом буде помилка, то при хорошому UX/API користувач, який здійснив запит, побачить повідомлення про помилку.

- Використання білих списків. Переважна більшість статей, присвячених ін'єкціям, зовсім не беруть до уваги цей момент. Але реальність така, що ми стикаємося з необхідністю підставляти у запит як дані, так й інші елементи — ідентифікатори (імена полів і таблиць) і навіть елементи синтаксису, ключові слова. Нехай навіть такі незначні, як DESC чи AND, але вимоги до безпеки таких підстановок все одно мають бути не менш строгими! Суть методу полягає в тому, що всі можливі варіанти вибору повинні бути жорстко прописані в нашому коді і в запит повинні потрапляти тільки вони, на підставі введення користувача.

3.4.1 Приведення до цілочисельного типу

Як і було описано в другому розділі, не завжди є сенс проводити важкі перевірки, або дезінфекцію для того, щоб уберегтися від SQL-ін’єкції. Особливо у випадку, якщо ми очікуємо отримати просте число. В мовах програмування для написання серверних додатків є безліч різних методів для роботи з числами. В даному розділі ми продовжимо використовувати Node JS і його можливості. Отже, приступимо до реалізації:

Припустимо, у нас є невеличкий роут, в кому параметром передається ID міста (Лістинг 3.13).

Лістинг 3.13 - Роут з параметром ідентифікатора міста в якості параметра

```
app.get('/cities/:cityId', (req, res, next) => {
/* дістаємо ID міста з переданих параметрів */
 const { cityId } = req.params;
```

```
/* виводимо результат в консоль */
 console.log('cityId:', cityId);
});
```

Тепер, за допомогою баузеро перейдомо по цій URL адресі і спробуємо передати в якості cityId SQL-ін’єкцію. Для цього потрібно відкрити сторінку з наступною URL-адресою: localhost:3001/cities/-1 WHERE '1' = '1'. Лог, який ми залишили виведе нам в консоль наступну інформацію (Рис. 3.16).

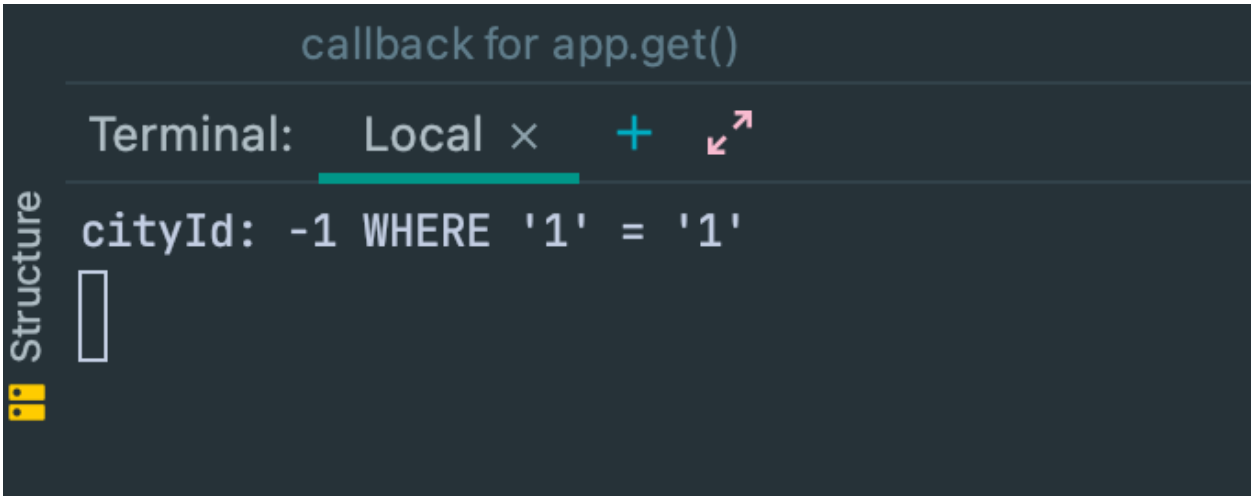

Рисунок 3.16 - Вивід переданої SQL-ін’єкції у консоль

Отже ми отримали SQL-ін’єкцію у вигляді cityId. Тепер, давайте спробуємо скористатися нативними можливостями мови програмування для того, щоб захиститися від даної вразливості, якщо ми очікуємо отримати число в якості параметра. В мові програмування JavaScript це виглядатиме так (Лістинг 3.14).

Лістинг 3.14 - Використання метода parseInt для дезінфекції даних

```
app.get('/cities/:cityId', (req, res, next) => {
 /* дістаємо ID міста з переданих параметрів */
 const { cityId } = req.params;
```

```
 /* використовуючи функцію parseInt дістаємо всі числа зі стрічки до
першого не числа */
 const parsedCityId = parseInt(cityId);
 /* виводимо результат в консоль */
 console.log('parsedCityId:', parsedCityId);
});
```

Дивимось в консоль та отримуємо очікуваний результат (Рис. 3.17).

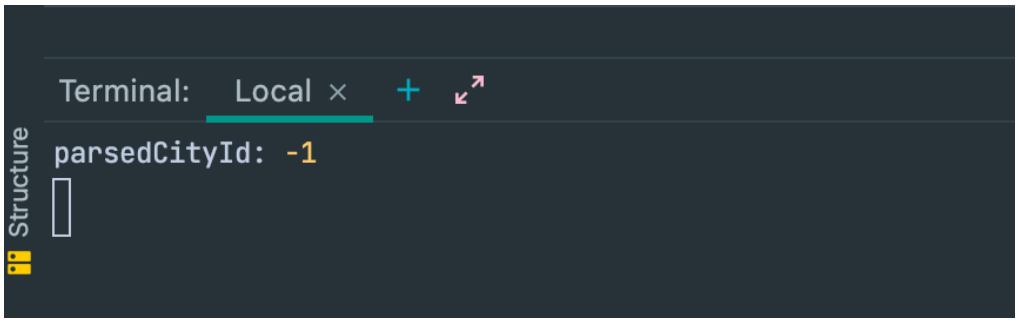

Рисунок 3.17 - Результат дезінфекції SQL-ін’єкції нативними методами JavaScript у консолі

Тепер порівняємо витрачений час на виконання шляхом дезінфекції та приведенням до цілого числа (Лістинг 3.15).

Лістинг 3.15 - Порівняння часу виконання стандартного методу та приведенням до цілочисельного типу

```
app.get('/cities/:cityId', async (req, res, next) => {
 /* дістаємо ID міста з переданих параметрів */
 const { cityId } = req.params;
 console.time('Serialize');
 connection.escapeId(cityId)
 console.timeEnd('Serialize');
 console.time('Parsing');
 parseInt(cityId);
```

```
 console.timeEnd('Parsing');
});
```

Результат з консолі наступний (Рис. 3.18).

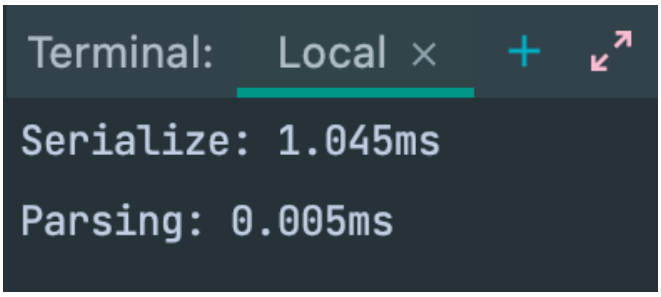

Рисунок 3.18 - Порівняння часу виконання дезінфекції та приведення до цілого числа

Отже, із даного результату можна зробити висновок, що використання приведення до цілого числа більш продуктивніше за стадартну дезінфекцію. Тому нам вдалося захиститися від SQL-ін’єкції використавши легкий нативний метод мови програмування, який не витрачає багато ресурсів сервера, та чудово справляється зі своїм завданням.

3.4.2 Використання білих списків

Отже, як було сказано у другому розділі, переважна більшість статей, присвячених ін'єкціям, зовсім не беруть до уваги цей момент. Але як і описано в минулому пункті, це також досить простий в реалізації та легкий, в плані навантаження сервера, підхід. Отже роут буде виглядати схоже до попереднього (Лістинг 3.16).

Лістинг 3.16 Роут з полем, по якому буде відбуватися упорядкування, в якості параметра

```
app.get('/cities/:orderBy, (req, res, next) => {
 /* витягуємо поле, за яким буде відбуватися упорядкування */
 const { orderBy } = req.params;
 /* виводимо його в консоль */
 console.log('orderBy:', orderBy);
});
```

Відповідно, в консолі ми будемо мати таку ж проблему (Рис. 3.19).

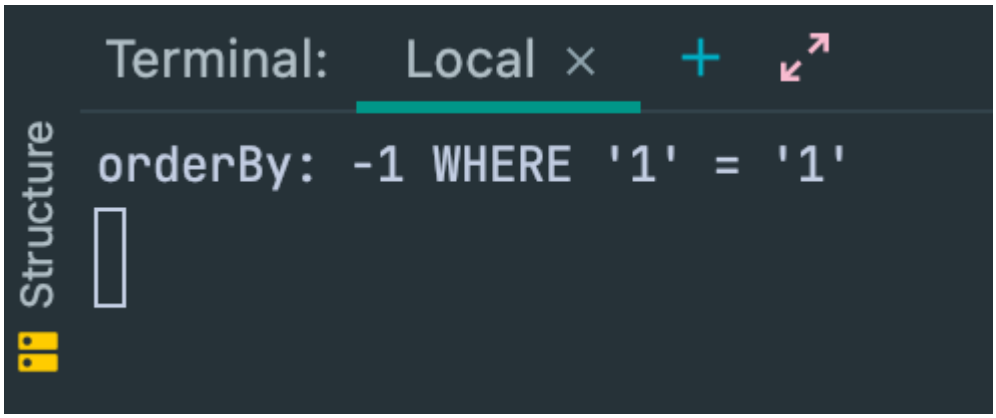

Рисунок 3.19 - Вивід SQL-ін’єкції у консоль

Отже, щоб захиститися від цієї атаки, нам потрібно заздалегідь в коді мати допустимі варіанти параметру. Після цього залишається тільки перевірити, чи підпадає переданий параметр від один із варіантів. Якщо так, то виконуємо відповідне сортування. Якщо ні, то просто повертаємо список яким у довільному порядку. Модифікований код буде виглядати наступним чином (Лістинг 3.17).

Лістинг 3.17 - Використання білих списків для захисту від SQL-ін’єкцій

```
/* Створюємо в коді константу з допустими значеннями */
const allowedFields = ['name', 'country', 'state'];
app.get('/cities/:orderBy', (req, res, next) => {
 /* витягуємо поле, за яким буде відбуватися сортування */
 const { orderBy } = req.params;
```

```
 /* виводимо його в консоль */
 console.log('orderBy:', orderBy);
 /* створюємо змінну, в якій буде міститись запит до бази */
 let query;
 /* перевіряємо, чи належить переданий параметр до доступних варіантів
*/
 if (allowedFields.includes(orderBy)) {
   /* якщо так, то впорядковуємо по ньому */
   query = `SELECT * FROM cities ORDER BY ${orderBy}`;
 } else {
   /* якщо ні, то використовуємо довільний порядок */
   query = 'SELECT * FROM cities';
 }
 /* виводимо запит в консоль */
 console.log('query:', query);
});
```

Отже, спробуємо спочатку атакувати SQL-ін’єкцією і подивитися на результат (Рис. 3.20).

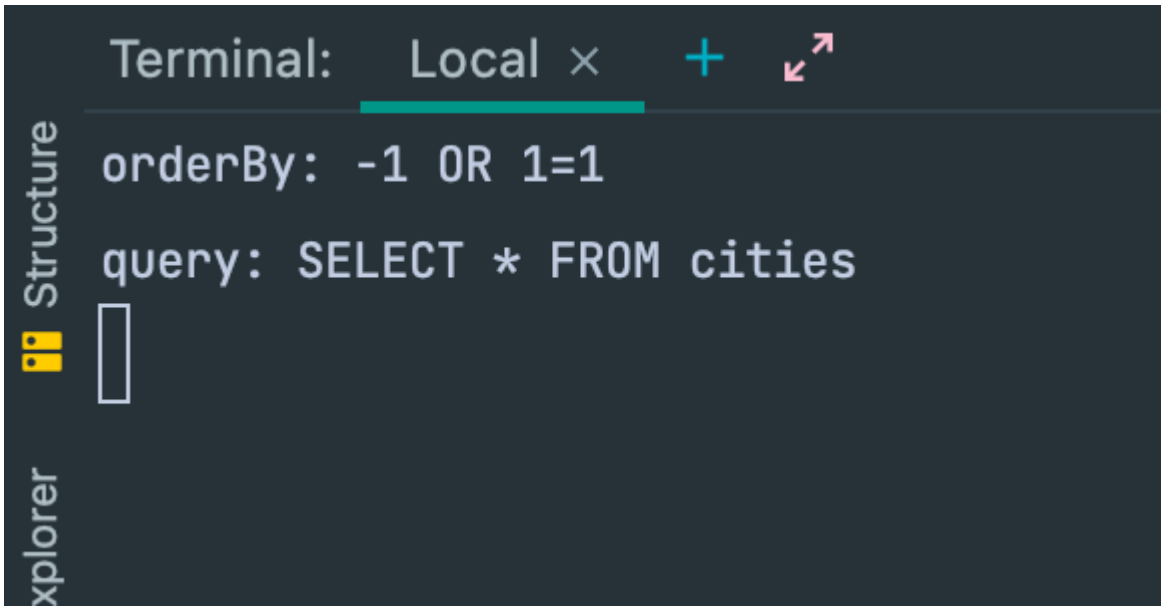

Рисунок 3.20 - Вивід результатів після атаки SQL-ін’єкцією

Як можна побачити, запит до бази поверне запити в довільному порядку і не використає параметр-ін’єкцію. Тепер, спробуємо передати параметр, який знаходиться у списку дозволених (Рис 3.21).

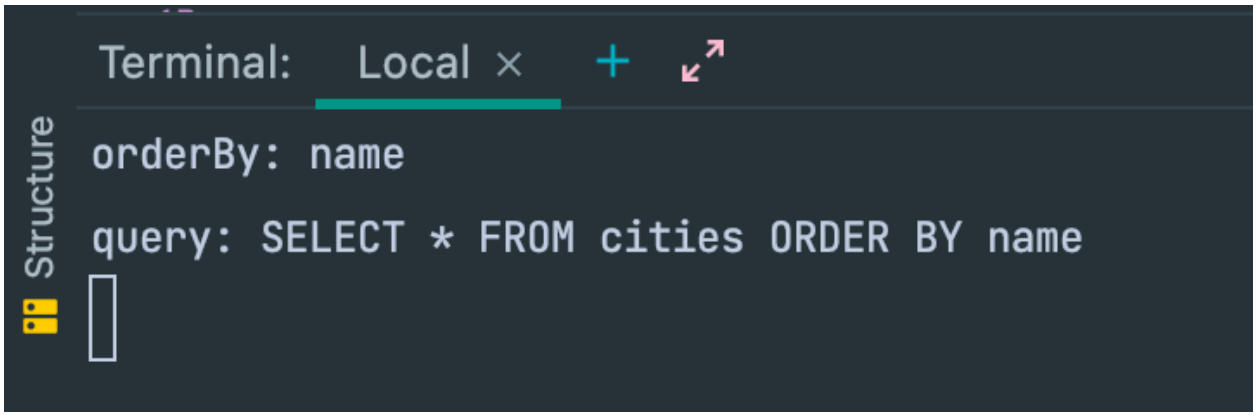

Рисунок 3.21 - Вивід результатів після передачі дозволених параметрів

Тепер здійснимо порівняння затрат часу на виконання (Лістинг 3.18).

Лістинг 3.18 - Порівняння часу виконання стандартного методу та удосконалення перевіркою наявності у білому списку

```
/* Створюємо в коді константу з допустими значеннями */
const allowedFields = ['name', 'country', 'state'];
app.get('/cities/:orderBy', (req, res, next) => {
 /* витягуємо поле, за яким буде відбуватися сортування */
 const { orderBy } = req.params;
   console.time('Serialize');
   connection.escapeId(cityId)
   console.timeEnd('Serialize');
   console.time('White list');
   allowedFields.includes(orderBy)
   console.timeEnd('White list');
```

```
});
```

Результат виконання наступний (Рис 3.22).

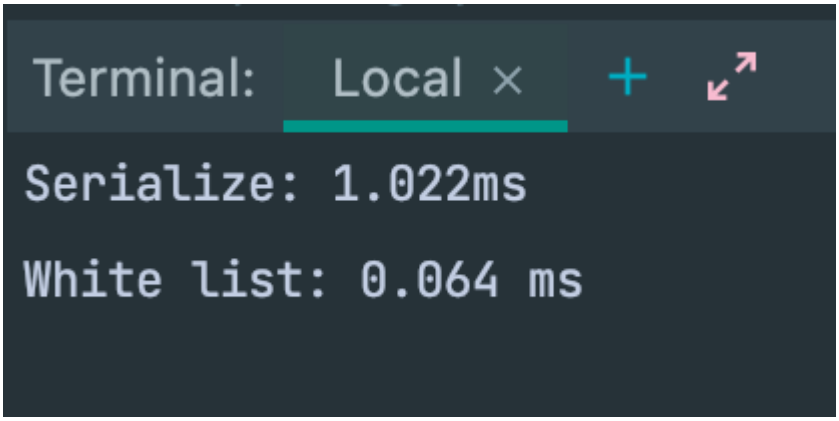

Рисунок 3.22 - Порівняння часу виконання дезінфекції та використання списків дозволеного

Отже, із даного результату можна зробити висновок, що використання білих списків більш продуктивне за стандартну дезінфекцію. Тому нам вдалося захиститися від SQL-ін'єкції шляхом використання дозволених списків. Даний спосіб є простим, малозатратним та ефективним.

Таблиця з порівнянням стандартних методів та удосконалень приведена на таблиці 3.1

Таблиця 3.1 - Порівняння часу виконання удосконалень та стандартних методів

Назва удосконалення	Стандартний час виконання	Час виконання з удосконаленням
Приведення до цілочисельного типу	1.045 мс.	0.005 мс.
Використання білих списків	1.022 мс.	0.064 мс

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

При розробці даних удосконалень робочі місця працівників з екранними пристроями мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів.

Для забезпечення безпеки та захисту здоров'я усе випромінювання від екранних пристроїв було зведене до гранично допустимого рівня (вплив на людину факторів довкілля - шуму, вібрації, забруднювачів, температури тощо, який не спричиняє соматичних або психічних розладів, а також змін стану здоров'я, працездатності, поведінки, що виходять за межі пристосувальних реакцій) з погляду безпеки та охорони здоров'я працівників.

Організація робочого місця забезпечує відповідність усіх елементів робочого місця та їх розташування ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.

Освітлення робочого місця створює відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98.

Мікроклімат приміщення з робочим місцем підтримується на постійному рівні та відповідає вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 42 (далі - ДСН 3.3.6.042-99).

Робочий стіл або робоча поверхня достатнього розміру та мають поверхню з низькою відбивною здатністю, допускають гнучкість під час розміщення екрана, клавіатури, документів і відповідного устаткування.

Робоче крісло стійке і дозволяти легко рухатися та займати зручне положення.

Сидіння має регулювання по висоті, спинка сидіння - як по висоті, так і по нахилу.

Звукоізоляція огороджувальних конструкцій приміщень з ВДТ забезпечує параметри шуму, що відповідають вимогам ДСТУ 2867-94, ГОСТ 12.1.036-81.

Приміщення для роботи обладнані системами опалення та кондиціонування повітря, відповідно до ДБН В.2.5-67:2013.

Нормовані параметри мікроклімату, іонного складу повітря, вмісту шкідливих речовин відповідають вимогам ГОСТ 12.1.005-88.

Віконні прорізи приміщень для роботи обладнані регульованими пристроями (жалюзі).

Штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ ЕОМ та ПЕОМ, здійснюється системою загального рівномірного освітлення. Зазначення освітлення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500 лк.

Як джерела світла в разі штучного освітлення застосовуються переважно люмінісцентні лампи типу ЛБ. У разі влаштування.

Значення напруженості електростатичного поля на робочих місцях з ВДТ (як у зоні екрана дисплея, так і на поверхнях обладнання, клавіатури, друкувального пристрою) не перевищують гранично допустимих за ГОСТ 12.1.045-84.

Значення напруженості електромагнітних полів відповідають нормативним значенням (ГОСТ 12.1.002-84)

Інтенсивність потоків інфрачервоного випромінювання не перевищує допустимих значень відповідно до ГОСТ 12.1.005-88.

Інтенсивність потоків ультрафіолетового випромінювання не перевищує допустимих значень відповідно до ГОСТ 12.1.005-88.

Обладнання і організація робочого місця з ЕОМ забезпечує відповідність конструкції всіх елементів робочого місця та їх взаємного, розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності (ДСТУ 8604:2015).

4.2 Підвищення стійкості роботи огд об’єктів госп діяльності у воєнний час

Ефективність економіки держави залежить від того, наскільки окремі галузі господарства здатні стійко працювати не тільки у звичайних умовах, а й в умовах НС мирного та воєнного часу.

Значні руйнування, пожежі та втрати серед населення, викликані наслідками НС, можуть стати причиною різкого скорочення випуску промислової та сільськогосподарської продукції, а отже і зниження економічного потенціалу держави. Виникає потреба завчасного вживання заходів щодо забезпечення стійкої роботи промислових об’єктів на випадок виникнення НС.

Знання можливих НС, характерних для даної місцевості та виробництва, дозволяє диференційовано і цілеспрямовано розробляти та здійснювати заходи, які можуть запобігти аваріям, катастрофам та стихійним лихам або пом’якшити їх наслідки.

Стійкість роботи об’єкта господарської діяльності – це здатність його в умовах НС випускати продукцію у запланованому обсязі та визначеної номенклатури, а у разі слабких та середніх руйнувань або порушення матеріального постачання - відновлювати виробництво власними силами у короткий термін.

На стійкість роботи об’єкта впливають такі фактори:

- захищеність робітників та службовців від уражальних факторів у НС;

- здатність інженерно-технічного комплексу об’єкта (будівель, споруд, обладнання та комунально-енергетичних мереж) протистояти руйнівній дії уражальних факторів аварій, катастроф, стихійного лиха та сучасної зброї;
- надійність постачання об’єкта електроенергією, водою, паливом, комплектуючими та сировиною;
- підготовленість об’єкта до проведення аварійно-рятувальних та відновлювальних робіт;
- оперативність управління виробництвом та здійсненням заходів ЦЗ у НС;
- підвищення стійкості об’єкта досягають проведенням комплексу інженерно-технічних, технологічних, організаційних заходів.

До інженерно-технічних заходів належать роботи, що забезпечують стійкість виробничих будівель і споруд, обладнання та комунально-енергетичних систем.

Технологічні заходи забезпечують підвищення стійкості об’єкта спрощенням технологічного процесу виробництва кінцевої продукції та виключенням або обмеженням розвитку аварій.

Організаційні заходи передбачають розробку ефективних дій керівного складу, служб та формувань ЦЗ, спрямованих на захист виробничого персоналу, проведення рятувальних та інших невідкладних робіт, а також відновлення виробництва.

Заходи щодо підвищення стійкості об’єктів здійснюють відповідно до вимог Норм проектування інженерно-технічних заходів цивільного захисту. Дані вимоги призначені для того, щоб в умовах НС:

- забезпечити захист населення та знизити масштаби руйнувань (пожеж, затоплень, заражень);
- підвищити стійкість роботи об’єктів і галузей економіки;

- створити умови для успішного проведення робіт з ліквідації наслідків НС.

Вимоги норм проектування реалізують під час проектування та забудови міст, будування нових промислових підприємств, об'єктів енергетики, транспортних систем, систем водо- та газопостачання, а також під час їх реконструкції.

Головним документом, відповідно до якого слід планувати та здійснювати інженерно-технічні заходи цивільного захисту (ІТЗ ЦЗ) є «Будівельні норми і правила» (БН і П 2.00.05-90), а також «Загальні вимоги до розвитку і розміщення потенційно небезпечних виробництв з урахуванням ризику надзвичайних ситуацій техногенного походження» (Київ, НАН України, 1995). Запровадження норм проектування ІТЗ ЦЗ здійснюється диференційовано з урахуванням ролі і важливості міст і об'єктів економіки. Для цього міста поділяють на групи, а об'єкти – на категорії за такою класифікацією: міста: «Особливої групи», I, II та III груп; об'єкти господарювання: «Особливої важливості», I та II категорій. Об'єкти атомної енергетики виділяють в окрему групу.

Для «категорійних» міст і об'єктів з метою реалізації ІТЗ встановлено дві зони: можливих слабких руйнувань, де очікується (за прогнозом) надмірний тиск у фронті повітряної УХ Рф = 10–30 кПа; можливих сильних руйнувань, у межах якої очікується Рф 30 кПа.

Межа зони сильних руйнувань для міст «особливої», I, II, III груп пролягає в межах проектної забудови міста (ПЗМ), а зони слабких руйнувань – на відстані 7 км від межі проектної забудови міста (ПЗМ приймають відповідно до затвердженого генерального плану забудови на розрахунковий період). Для об'єктів «особливої важливості» межа зони сильних руйнувань пролягає на відстані 3 км від межі проектної забудови об'єкта; слабких – 10 км.

Основні вимоги до планування і забудови нових міст, а також реконструкції збудованих міст, такі:

- Забудовувати місто треба окремими житловими масивами, мікрорайонами. Їх межами мають бути парки, смуги зелених насаджень, широкі магістралі, водойми, що створюють протипожежні розриви .
- У містах і мікрорайонах, де немає природних водойм, слід створювати штучні із запасом води для гасіння пожеж, проведення дезактивації території і санітарної обробки населення.
- У кожному секторі (мікрорайоні) має бути не менше однієї евакуаційної магістралі для евакуації населення із ОУ в заміську зону. Ширину магістралі, м, визначають за формулою L = Hmax+ 15, де Hmax – висота найвищої будівлі на магістралі (окрім висотних громадських будівель каркасної конструкції), м.
- Міжміські автомобільні дороги слід прокладати в обхід міста. Це зменшить забруднення повітряного басейну міста від автотранспорту і не порушить транспортних зв’язків у разі повної руйнації міста при НС.
- Створення лісопаркової смуги навколо міста і будівництво в ній туристичних і спортивних баз, пансіонатів тощо. Це має важливе значення для організації відпочинку населення, а у разі НС – для розміщення евакуйованого населення міста.

Нові важливі промислові підприємства слід будувати за межами зони можливих руйнувань (міської забудови). У місті можна будувати лише бази та склади з товарами першої необхідності, підприємства для обслуговування населення.

Вибираючи місце будівництва об’єкта, враховують наявність поблизу підприємств, які можуть бути джерелом небезпеки (гідровузли, хімічні підприємства та ін.), рельєф місцевості, сейсмічність району, панівні вітри тощо.

Групи нових підприємств та окремих категорійних об’єктів слід будувати в економічно перспективних малих і середніх містах, селищах і сільських населених пунктах, розташованих від межі проектної забудови категорійних міст і об’єктів особливої важливості на такій відстані: не менше 60 км від міст «особливої» і I групи, 40 км – від міст II групи, 25 км – від міст III групи та об’єктів «особливої важливості».

Розміщення АЕС повинно забезпечувати радіаційну безпеку населення у разі аварії. Мінімально допустима відстань АЕС від межі проектної забудови міста залежить від чисельності населення міста і потужності АЕС і становить не менше 25 км для міста з населенням 100–500 тис., не менше 100 км для міст з населенням більше 2 млн. осіб.

Підприємства з переробки легкозаймистих і пальних рідин, вибухових речовин і матеріалів, об’єктів, що мають СДОР, а також базові склади зазначених речовин і матеріалів слід розміщувати в заміській зоні на безпечній відстані від населених пунктів і об’єктів, нижче за схилом місцевості щодо житлових масивів, автомобільних доріг і залізниць. Базові склади нафти і нафтопродуктів, які споруджують на берегах річок (на відстані до 200 м від краю води) слід розміщувати нижче (за течією води) і на відстані не менше 100 м від населених пунктів.

Проектування і будівництво нових об’єктів здійснюється відповідно до таких вимог:

- Будівлі і споруди розміщують розосереджено, з протипожежними розривами між ними $L_p=H1+H2+(15…20)$ м, де H1, H2 – висота сусідніх будівель, м.
- Найбільш важливі промислові будівлі та споруди будують з меншою кількістю висотності та з використанням вогнетривких матеріалів.

- Склади для зберігання палива та легкозаймистих матеріалів розміщують біля межі об’єкта або за його межами, в підземних спорудах.
- Дороги на території об’єкта мають бути з твердим покриттям, забезпечувати найкоротше сполучення між виробничими будівлями та мати не менше двох виїздів з різних боків об’єкта.

Для забезпечення надійного постачання об’єкта господарювання електроенергією, водою та газом в комунально-енергетичних системах слід передбачати:

- дублювання джерел постачання;
- кільцювання систем;
- прокладання комунікацій під землею;
- створення резервних джерел постачання або резервних запасів;
- використання пристроїв для автоматичного вимкнення пошкодженої дільниці.

Електропостачання має здійснюватися від енергосистем, до яких входять електростанції на різних видах палива. Електроенергію до дільниць виробництва слід подавати окремими електрокабелями, прокладеними під землею.

Виконання вимог норм проектування сприяє не тільки безпечному та безперебійному функціонуванню промислових об’єктів, але й покращенню умов праці та проживання в певному районі.

ВИСНОВКИ

Отже, у процесі виконання даної роботи було проведено дослідження стану захищеності веб-додатків, за результатами якого можна зробити висновок про негативний вплив переходу на дистанційну роботу на безпеку.

Також було здійснено опис найпоширеніших способів захисту від таких сразливостей як підбір, слабка валідація відновлення пароля, XSS та SQL-інєкції.

Способи захисту веб-додатків від зловмисників залежить від конкретних технологій і компонентів, що використовуються при створенні веб-додатків, їх переваг та недоліків у даній сфері. Існують різні класифікації вразливостей, кожна атака через вразливість має свої особливості, але основними причинами вразливостей в захисті веб-додатків є помилки в розробці, реалізації та застосуванні компонентів веб- додатків.

Метою даної роботи було удосконалення стандартних засобів захисту веб-додатків від вразливостей. Це було досягнуто шляхом збільшення часу, необхідного для взлому, підвищенням продуктивності валідації та використанням неочевидних підходів до захисту від конкретної вразливості.

СПИСОК ЛІТЕРАТУРИ

1. https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2021/?#web-server-vulnerabilities-and-misconfigurations
2. Паперовський Б. О. Аналіз сучасних методів виявлення аномалій та можливих вторгнень в роботі комп'ютерної системи / Б. Паперовський, Н. Загородна // Матеріали VI науково-технічної конференції „Інформаційні моделі, системи та технології", 12-13 грудня 2018 року. — Т. : ТНТУ, 2018. — С. 49. — (Інформаційні системи та технології).
3. Яциковська У. Вплив атак на відмову в обслуговуванні на комп'ютерні мережі / У. Яциковська, Я. Кінах // Матеріали IV науково-технічної конференції „Інформаційні моделі, системи та технології", 15-16 травня 2014 року — Т. : ТНТУ, 2014 — С. 49. — (Безпека інфокомунікацій).
4. Карпінський М. П. Методи безпечної роботи при використанні SQL-сервера ORACLE / Карпінський М.П., Сальніков М. // Вісник Тернопільського державного технічного університету , 2002 — том 7. — с.95-100
5. Методологія та технологія створення складних програмних систем : Навчально-методичний посібник / . — Тернопіль : ТНТУ , 2017 — 40 с.
6. Яциковська У. О. Механізм захисту від атак на відмову / Уляна Олегівна Яциковська, М. Карпінський, Ярослав Ігорович Кінах // Матеріали II науково-технічної конференції „Інформаційні моделі, системи та технології", 25 квітня 2012 року — Т. : ТНТУ, 2012 — С. 38. — (Секція 3. Комп'ютерні системи та мережі).
7. https://beaglesecurity.com/blog/article/how-to-improve-web-application-security.html
8. https://owasp.org/www-community/attacks/xss/
9. https://www.cloudflare.com/learning/security/what-is-web-application-security/

10. https://www.offensive-security.com/metasploit-unleashed/client-side-attacks/
11. https://www.cloudflare.com/learning/bots/brute-force-attack/
12. https://www.imperva.com/learn/application-security/application-security/
13. https://www.synopsys.com/glossary/what-is-web-application-security.html
14. https://www.azoft.ru/blog/spa-mpa-pwa

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

ІХ НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ

2021

УДК 004.056

А. В. Степанов, А. А. Микитишин
(Тернопільський національний технічний університет ім. І. Пулюя)

ДОСЛІДЖЕННЯ ТА УДОСКОНАЛЕННЯ СТАНДАРТНИХ МЕТОДІВ ЗАХИСТУ WEB-ДОДАТКІВ

UDC 004.056
A. V. Stepanov, A. A. Mykytyshyn

STUDY AND IMPROVEMENT OF STANDARD METHODS OF WEB-APPLICATIONS PROTECTION

Веб-додаток - це будь-яка комп'ютерна програма, яка виконує певну функцію, використовуючи веб-браузер в якості свого клієнта. Додаток може бути таким же простим, як дошка оголошень або контактна форма на веб-сайті, або ж настільки складним, як текстовий процесор або багатокористувацький мобільний ігровий додаток, завантажений на телефон[1].

Сайти в інтернеті, як і інше програмне забезпечення (ПЗ), схильні до різних вразливостей, що дозволяє зловмисникам отримувати доступ до секретних і важливих даних або виконувати інші незаконні дії. Деякі вразливості дуже небезпечні і зустрічаються частіше, ніж хотілося б, інші ж менш небезпечні і зустрічаються рідко. Важливо знати, які вразливості бувають, перевіряти свій ресурс на їх присутність і вчасно їх виправляти[2].

До найпопулярніших вразливостей можна віднести: Injection, Injection flaws, SQL, NoSQL, OS, LDAP — ін’єкції коду, Broken Authentication, Broken Access Control, Security Misconfiguration, Cross-Site Scripting XSS, Brute Force, Vulnerable JavaScript libraries[1].

Грамотний захист завжди будується на розумінні слабкостей ПЗ, яке необхідно захистити. Це дозволяє відсіяти неактуальні спроби атак і виділити тільки ті, які використовують реальні вразливості, наявні в системі.

Стандартні методи захисту веб-додатків (такі як: використання токена, серіалізація даних, CAPTCHA, збільшення складності пароля і т.д.) є мінімальним задовільним рівнем захисту, наявність якого не гарантує цілковиту захищеність ресурсу від різного виду атак. Деякі з методів існують лише в одному варіанті, який перевірений часом та дає повний захист від конкретної вразливості, в той час як інші можуть бути вдосконалені та давати набагато більший захист.

Удосконалення стандартних методів захисту веб-ресурсів від атак залишається актуальною проблемою, особливо з урахуванням постійного вдосконалення методів та інструментів атак, а також зі зростанням економічних, соціальних та політичних наслідків зловмисних дій. Його можна здійснити у декілька різних способів, в залежності від методу, який використовується. Це може бути банальне покращення продуктивності, збільшення складності шифрування, додавання нових етапів та обмежень до процесу перевірки і т.д. В подальшому дослідженні буде проаналізовано вплив різного плану вдосконалень методів забезпечення безпеки веб додатків на їх стійкість.

Література.

1. Andrew Hoffman. Web Application Security / O'Reilly Media, Inc. (USA), 2020. – 217 c.
2. Handbook on Ontologies / eds. S. Staab and R. Studer. — International Handbooks on Information Systems. — Berlin: Springer, 2009. — 832 p.