Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Савченка Сергія Володимировича

на здобуття ступеня вищої освіти Бакалавра

Комплексна систему захисту REST API додатків
від атак на основі підбору сесій

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ. 2101129.21.01.12 ПЗ

Виконав: студент 4 курсу, група КБ-21-1 Сергій САВЧЕНКО
Керівник: д-р філософії, ст. викладач, Микола СТЕЦЮК
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

2 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Савченко Сергій Володимирович

(Прізвище,ім'я, по батькові студента)

1 Тема роботи Комплексна систему захисту REST API додатків від атак на основі підбору сесій

Керівник роботи Микола СТЕЦЮК

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи лістинги створених шкідливих скриптів, які використовувалися для викрадення токенів сесій користувачів, результати проведених тестувань та покращень

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області, Історія розвитку REST API, Огляд можливих методів захисту REST API, Порівняння та вибір інструментів для проведення тестування, Розробка веб-сервера, вибір потенційних додаткових інструментів, Початок тестування із викрадення сесій, Аналіз проведених випробувань та можливі покращення захисту

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Основні алгоритми виконаного тестування, отримані результати після запровадження захисту

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис,дата	
		Завдання видав	Завдання прийняв

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконав
Ознайомлення з предметною областю	Лютий	виконав
Дослідження існуючих рішень	Лютий	виконав
Постановка задачі	Березень	виконав
Визначення загальних принципів рішення задачі	Березень	виконав
Деталізація принципів рішення задачі	Квітень	виконав
Розробка проєктних рішень	Квітень	виконав
Апробація проєктних рішень	Травень	виконав
Оформлення пояснювальної записки згідно вимог	Травень	виконав
Оформлення графічної частини	Червень	виконав
Захист КР	Червень	виконав

Студент _____ Сергій САВЧЕНКО

Керівник кваліфікаційної роботи _____ Микола СТЕЦЮК

АНОТАЦІЯ

Тема кваліфікаційної роботи: Комплексна систему захисту REST API додатків від атак на основі підбору сесій

Автор роботи: Савченко Сергій Володимирович.

Керівник роботи: Микола Стецюк.

Пояснювальна записка: 69 с., 2 додатки, 17 рисунків, 40 джерел.

Графічна частина: 3 плакат, 13 презентаційних слайдів.

REST API, АРХІТЕКТУРА, ТОКЕНИ СЕСІЇ, ПЕРСОНАЛЬНІ ДАНІ, ВЕБ-ДОДАТОК.

Кваліфікаційна робота бакалавра присвячена викраденню і захисту сесійних токенів користувача на створеному веб-додатку.

У роботі детально проаналізовано предметну область та історію розвитку REST API, що дозволило визначити ключові особливості та виклики у забезпеченні їх безпеки. Здійснено всебічний огляд існуючих методів захисту REST API. На основі порівняльного аналізу різних інструментів для тестування безпеки обрано найбільш ефективні засоби, які відповідають специфіці досліджуваного веб-сервера. Розроблено веб-сервер, а також визначено потенційні додаткові інструменти для посилення захисту, такі як системи моніторингу та виявлення аномалій.

На основі результатів випробувань здійснено глибокий аналіз виявлених проблем та розроблено рекомендації щодо покращення захисту REST API, включаючи впровадження додаткових заходів безпеки, оптимізацію існуючих механізмів та підвищення загальної надійності системи. Таким чином, робота забезпечує комплексний підхід до створення та тестування безпечного REST API, що відповідає сучасним стандартам інформаційної безпеки

.22.05.2025

ABSTRACT

Subject of qualification work: A comprehensive system for protecting REST API applications from attacks based on session matching

Author of the work: Savchenko Serhii Volodymyrovych.

Head of work: Mykola STETSYUK.

Explanatory note: 69 p., 2 appendices, 17 pictures, 40 sources.

Graphic part: 3 poster, 13 presentation slides.

REST API, ARCHITECTURE, SESSION TOKENS, PERSONAL DATA, WEB APPLICATION.

The bachelor's qualification work is devoted to the theft and protection of user session tokens on the created web application.

The work analyzes in detail the subject area and history of REST API development, which allowed to identify key features and challenges in ensuring their security. A comprehensive review of existing methods of protecting REST APIs is carried out. Based on a comparative analysis of various security testing tools, the most effective tools that meet the specifics of the studied web server were selected. The web server was developed, and potential additional tools to enhance security, such as monitoring and anomaly detection systems, were identified.

Based on the test results, an in-depth analysis of the identified problems was carried out and recommendations were developed to improve the protection of the REST API, including the implementation of additional security measures, optimization of existing mechanisms, and improvement of the overall reliability of the system. Thus, the work provides a comprehensive approach to creating and testing a secure REST API that meets modern information security standards.

29.05.2025

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій, де веб-додатки та сервіси стають невід'ємною частиною бізнес-процесів і повсякденного життя, безпека даних і комунікацій набуває особливої ваги. Зі зростанням цифрової трансформації, інтеграції різних систем і розширення кількості користувачів, питання захисту інформації стає пріоритетом для організацій будь-якого масштабу. Одним із ключових елементів, що забезпечують взаємодію між різними програмними системами, є REST API (Representational State Transfer Application Programming Interface). REST API дозволяє розробникам створювати гнучкі, масштабовані та ефективні рішення, що забезпечують обмін даними між серверами та клієнтами, а також інтеграцію різноманітних сервісів і платформ. Водночас, зростання популярності REST API супроводжується збільшенням потенційних ризиків, пов'язаних із безпекою, що вимагає постійної уваги і впровадження сучасних методів захисту.

REST API, як архітектурний стиль, базується на принципах простоти, масштабованості та уніфікованого інтерфейсу, що робить його ідеальним для побудови сучасних веб-сервісів. Вони широко застосовуються у мобільних додатках, хмарних платформах, мікросервісній архітектурі, IoT (Інтернеті речей) та багатьох інших сферах. Завдяки своїй гнучкості, REST API дозволяють розробникам швидко створювати сервіси, які можуть взаємодіяти незалежно від платформи чи мови програмування. Проте ця відкритість і доступність одночасно створюють умови для появи нових загроз, оскільки REST API часто стають мішенню для кіберзлочинців, які прагнуть отримати несанкціонований доступ до систем, викрасти конфіденційні дані або порушити роботу сервісів.

Однією з найбільш поширених і небезпечних загроз для REST API є атаки на основі підбору сесій, які можуть призвести до компрометації облікових записів, втрати конфіденційності та порушення цілісності даних. Ці атаки зазвичай реалізуються шляхом викрадення або вгадування сесійних токенів, які використовуються для ідентифікації користувача під час взаємодії з API.

Відсутність належного захисту сесійних даних або слабкі механізми управління сесіями створюють вразливості, що дозволяють зловмисникам отримати доступ до системи під виглядом легітимного користувача. Наслідки таких атак можуть бути критичними, включаючи фінансові втрати, шкоду репутації компанії, а також юридичні санкції.

У зв’язку з цим, забезпечення надійного захисту REST API від атак на основі підбору сесій є надзвичайно актуальним завданням для розробників, інженерів з безпеки та організацій загалом. Це вимагає комплексного підходу, який включає не лише впровадження технічних засобів захисту, але й розробку політик безпеки, навчання персоналу та постійний моніторинг систем. Методи захисту повинні охоплювати різні аспекти, починаючи від аутентифікації і авторизації користувачів, використання надійних протоколів шифрування, контролю доступу до ресурсів, і закінчуючи управлінням життєвим циклом сесійних токенів.

Метою даної дипломної роботи є всебічне дослідження методів захисту REST API від атак на основі підбору сесій. У рамках роботи буде здійснено глибокий аналіз існуючих вразливостей, які найчастіше експлуатуються зловмисниками, а також розглянуто ефективні стратегії і технології, що дозволяють мінімізувати ризики. Особлива увага буде приділена питанням аутентифікації та авторизації, оскільки саме ці механізми є фундаментальними для забезпечення безпеки API. Додатково буде проаналізовано роль шифрування даних, контролю доступу, а також моніторингу і логування подій, що допомагає виявляти і реагувати на підозрілі дії. Також ця робота має на меті не лише теоретичне дослідження проблеми, а й формування практичних рекомендацій щодо впровадження сучасних заходів безпеки для REST API. Результати дослідження можуть бути корисними як для розробників програмного забезпечення, так і для спеціалістів у сфері інформаційної безпеки, які прагнуть забезпечити високий рівень захисту своїх систем. Впровадження отриманих рекомендацій сприятиме підвищенню надійності, стійкості та довіри до цифрових сервісів, що є критично важливим у сучасному світі, де інформація є одним із найцінніших ресурсів.

Важливо відзначити, що безпека REST API - це не лише питання технічних рішень, а й організаційних процесів. Вона включає створення політик безпеки, навчання персоналу, аудит систем і регулярне тестування на вразливості. Успішний захист API потребує комплексного підходу, який враховує як технологічні, так і людські фактори. Це дозволяє не лише запобігати атакам, але й швидко реагувати на інциденти, мінімізуючи їхній вплив.

Таким чином, дана дипломна робота спрямована на всебічне вивчення проблеми захисту REST API від атак на основі підбору сесій, що є актуальним і важливим напрямом у сфері інформаційної безпеки. Вона покликана допомогти розробникам і фахівцям з безпеки краще розуміти загрози, що існують у сучасному цифровому середовищі, та ефективно застосовувати сучасні методи і технології для їх подолання. Це сприятиме підвищенню загальної безпеки інформаційних систем і захисту критично важливих даних у сучасному світі.

1 ТЕОРЕТИЧНІ ТА ПРАКТИЧНІ ОСНОВИ ПОСТАВЛЕНОГО ЗАВДАННЯ

1.1 Аналіз предметної області і виявлення наявних проблем та завдань

Захист REST API від атак на осанові підбору сесій є надзвичайно важливим аспектом забезпечення безпеки сучасних веб-сервісів, адже REST API все частіше виступають ключовою точкою входу для користувачів і додатків до цифрових ресурсів. Основна мета захисту полягає у запобіганні несанкціонованому доступу до даних і функцій через викрадення або підбір сесійних токенів, які використовуються для ідентифікації користувача під час взаємодії з API [1]. Атаки на основі підбору сесій передбачають спроби зловмисників вгадати, викрасти або повторно використати сесійні токени, що дає їм можливість отримати доступ до чужих облікових записів чи ресурсів. Для протидії таким атакам застосовуються різноманітні методи, які включають як технічні, так і організаційні заходи. Принцип впровадженого REST API можливо переглянути рисунку 1.1.

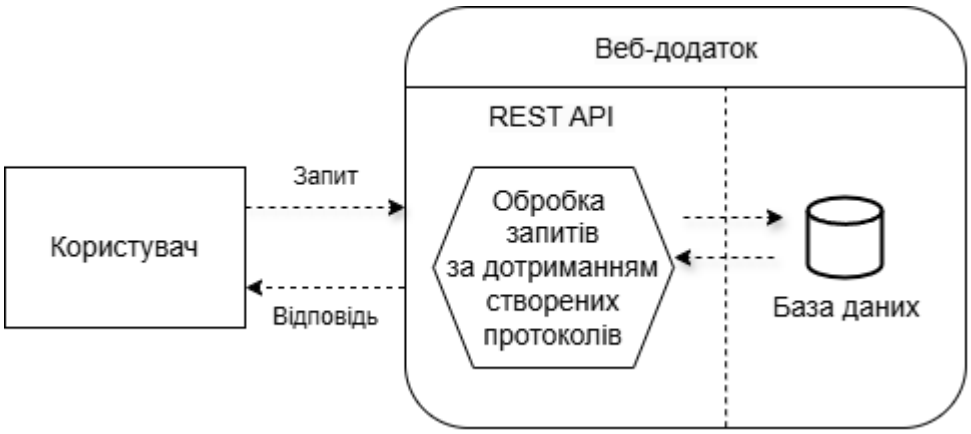


Рисунок 1.1 – Застосування REST API веб-додатку

Захист REST API є критично важливим через стрімке зростання кількості API-ендпоінтів у цифровій інфраструктурі сучасних організацій. API стали одним із найпривабливіших векторів атак, оскільки вони відкривають доступ до бізнес-логіки, даних користувачів і внутрішніх сервісів. За статистикою, близько 31% клієнтських API не використовують базове шифрування, що робить їх вразливими до перехоплення трафіку та викрадення сесійних токенів. Окрім того, з’являються

новітні загрози, такі як BOLA (Broken Object Level Authorization), які дозволяють зловмисникам отримувати доступ до чужих даних шляхом маніпуляцій з ідентифікаторами об’єктів у запитах. Це підкреслює, що навіть при наявності автентифікації можуть існувати критичні вразливості, пов’язані з недостатньою перевіркою прав доступу [2].

Регуляторні вимоги, наприклад австралійський закон SOCI (Security of Critical Infrastructure Act) та стандарти APRA CPS 234, встановлюють суворі норми безпеки для API, що підкреслює не лише технічну, а й юридичну відповідальність організацій за захист своїх цифрових сервісів. Порушення цих вимог може призвести до значних штрафів і репутаційних втрат. Потенційні загрози для REST API включають неавторизований доступ через маніпуляції з параметрами запитів, такі як BOLA або SSRF (Server-Side Request Forgery), а також ін’єкційні атаки, що виникають через недостатню валідацію вхідних даних. Ще одним ризиком є надмірне розкриття інформації у відповідях API, що може призвести до втрати конфіденційності даних.

DoS-атаки (Denial of Service) можуть бути реалізовані через складні GraphQL-запити або масові одночасні звернення до API, що призводить до перевантаження серверів і недоступності сервісів для легітимних користувачів. Також існує ризик зловживання функціоналом API через автоматичне присвоєння параметрів запитів до моделей даних, що може зачіпати конфіденційні поля і призводити до витоків інформації або порушення цілісності даних.

Наслідки недотримання заходів безпеки можуть бути катастрофічними. За даними 2024 року, середні фінансові втрати від витоку даних сягали $4.88 млн, а інциденти, пов’язані з API, часто призводять до штрафів, втрати клієнтів і серйозної шкоди репутації [3]. Зокрема, близько 60% компаній втрачають клієнтів після публікації інформації про порушення безпеки. Юридичні санкції можуть включати штрафи до 4% від глобального доходу компанії за порушення таких законів, як GDPR чи SOCI. Операційні простої, спричинені DoS-атаками, можуть паралізувати роботу критичних сервісів, що підтверджується прикладами авіакомпаній, які зазнали втрат мільйонів записів через вразливості в API [4].

Сучасні підходи до захисту REST API базуються на впровадженні Zero Trust-моделі, яка передбачає постійну перевірку ідентифікації користувачів і пристроїв, а також шифрування даних на всіх етапах передачі і зберігання. Використання штучного інтелекту та машинного навчання для моніторингу трафіку дозволяє виявляти аномалії в режимі реального часу, що допомагає швидко реагувати на потенційні загрози. Наприклад, застосування JWT (JSON Web Tokens) з алгоритмом ES256 і обмеження часу життя токенів до 15 хвилин значно знижує ризик перехоплення сесій, оскільки навіть у разі компрометації токен швидко стає недійсним.

Ефективний захист REST API також включає регулярне тестування на вразливості за допомогою інструментів, таких як OWASP ZAP, які дозволяють автоматизувати пошук слабких місць у реалізації API. Важливо також постійно оновлювати протоколи безпеки та відстежувати нові загрози через спеціалізовані платформи ,по типу Traceable.ai, що забезпечують проактивний захист і адаптацію до змін у ландшафті кіберзагроз [5].

Таким чином, захист REST API від атак на основі підбору сесій є багатошаровим процесом, який включає використання сучасних технологій шифрування, надійних механізмів аутентифікації і авторизації, постійного моніторингу і аналізу трафіку, а також регулярного тестування і оновлення систем безпеки. Впровадження цих заходів є необхідною умовою для забезпечення цілісності, конфіденційності та доступності даних у сучасних цифрових сервісах, а також для дотримання законодавчих і нормативних вимог.

1.2 Створення, впровадження та розвиток REST API

API або розширено Application Programming Interface рахується фундаментальним концептом у сучасній розробці програмного забезпечення, який визначає набір правил і протоколів, що дозволяють різним програмним системам взаємодіяти між собою. Спершу API можна розглядати як інтерфейс,

який описує, як одна програма може звертатися до функцій іншої, отримувати від неї дані або керувати її поведінкою. Мета API полягає у створенні стандартизованого способу обміну інформацією та функціональністю між різними програмними компонентами, незалежно від їхньої внутрішньої реалізації. Це дозволяє розробникам будувати складні системи, де різні частини можуть ефективно взаємодіяти, не заглиблюючись у деталі роботи одна одної [6].

Історично концепція API виникла ще в 1960-х роках, коли зростаюча складність комп'ютерних систем вимагала чітких способів взаємодії між різними модулями та програмами. Перші API були внутрішніми інтерфейсами операційних систем або бібліотек, які дозволяли програмам викликати функції одна одної. Згодом, із розвитком мережевих технологій, API почали використовувати для зв'язку між віддаленими системами, що дало поштовх до створення нових протоколів і стандартів, таких як RPC (Remote Procedure Call) у 1980-х, який дозволяв викликати функції на віддалених машинах, наче вони були локальними. Це стало важливим кроком у напрямку розподілених обчислень і заклало основу для сучасних веб-сервісів.

З розвитком Інтернету з'явилася потреба у стандартизованих способах обміну даними між веб-додатками. Тоді домінували протоколи на кшталт SOAP (Simple Object Access Protocol), які базувалися на XML і надавали суворі правила для обміну повідомленнями. SOAP API були потужними, але складними у реалізації та підтримці, що обмежувало їхню гнучкість і швидкість розробки. Водночас з'являлися більш легкі та прості підходи, які прагнули використовувати переваги протоколу HTTP, що вже широко застосовувався у інтернеті.

Саме в цей час, у 2000 році, доктор Рой Філдінг у своїй дисертації запропонував архітектурний стиль, який отримав назву REST (Representational State Transfer). REST став відповіддю на потребу у простому, масштабованому та гнучкому способі організації взаємодії між розподіленими системами через Інтернет. Основна ідея REST полягає у тому, що всі ресурси системи ,як дані, об'єкти та сервіси мають бути представлені у вигляді унікальних URI (Uniform Resource Identifier), а взаємодія з ними відбувається через стандартизовані HTTP-

методи, такі як GET, POST, PUT, DELETE. Це дозволяє клієнтам і серверам спілкуватися у зрозумілій і передбачуваній формі, що значно спрощує розробку, тестування і масштабування систем [7]. Принцип застосування HTTP-методів наведено на рисунку 1.2.

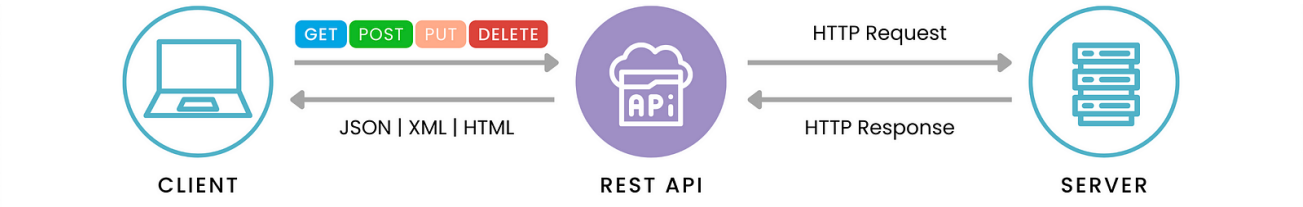


Рисунок 1.2 – Використання HTTP-методів

REST API, або RESTful API, - це конкретна реалізація API, яка дотримується принципів REST. Вона забезпечує простий і ефективний спосіб доступу до ресурсів через веб-протокол HTTP. REST API відрізняється від традиційних API тим, що він є stateless (безстанним), тобто кожен запит від клієнта до сервера містить всю необхідну інформацію для його обробки, без збереження стану сесії на сервері. Це підвищує масштабованість і надійність системи. Крім того, REST API підтримує кешування відповідей, що оптимізує використання мережевих ресурсів і покращує продуктивність. Важливою особливістю REST API є уніфікований інтерфейс, який базується на чотирьох основних принципах: ідентифікація ресурсів через URI, маніпуляція ресурсами через їхні представлення, наприклад, у форматах JSON або XML, самодокументовані повідомлення, які містять всю необхідну інформацію для обробки, та використання HATEOAS (Hypermedia as the engine of application state), що дозволяє клієнтам динамічно орієнтуватися у доступних діях і ресурсах. Ці обмеження забезпечують простоту, гнучкість і послідовність у роботі з API.

Застосування REST API стало надзвичайно популярним завдяки своїй сумісності з існуючими веб-технологіями, простоті використання і можливості легко інтегрувати різні системи. REST API забезпечують уніфікований і стандартизований спосіб взаємодії між клієнтськими додатками та серверними

сервісами, що робить їх незамінними у сучасній розробці програмного забезпечення. Вони широко використовуються у мобільних додатках, веб-сервісах, хмарних платформах, Інтернеті речей (IoT), мікросервісній архітектурі та багатьох інших сферах, де потрібна гнучка, масштабована і ефективна комунікація між різними компонентами системи [8].

REST API дозволяє розробникам швидко створювати масштабовані і легко підтримувані сервіси, які можуть взаємодіяти між собою незалежно від мови програмування чи платформи. Завдяки використанню стандартних протоколів, таких як HTTP, і форматів даних, наприклад JSON або XML, REST API забезпечує простоту інтеграції і сумісність із широким спектром клієнтських технологій. Це дозволяє створювати розподілені системи, де різні сервіси можуть ефективно обмінюватися інформацією, підтримуючи високу продуктивність і надійність [9].

Історично першим із потенційних засобів, що сприяли розвитку API, були внутрішні бібліотеки та модулі в операційних системах, які надавали стандартизовані функції для взаємодії з апаратним забезпеченням або іншими програмами. Ці локальні інтерфейси дозволяли програмам ефективно використовувати ресурси системи, але були обмежені рамками однієї машини. Згодом з’явилися протоколи віддаленого виклику процедур (RPC), які дозволяли викликати функції на віддалених машинах, що стало основою для подальших веб-сервісів і розподілених систем. RPC надавав можливість більш гнучкої взаємодії між додатками, але часто супроводжувався складністю у реалізації та обмеженнями в масштабованості [10].

Не менш важливим етапом розвитку стали веб-сервіси на основі SOAP (Simple Object Access Protocol), який хоча й був складним, але забезпечував формалізований і стандартизований спосіб обміну повідомленнями між системами. SOAP підтримував суворі стандарти безпеки, транзакційності та надійності, що робило його популярним у корпоративних середовищах. Проте його складність і надмірність призвели до пошуку більш легких і гнучких рішень.

Згодом з’явилися REST API, які, завдяки своїй простоті, гнучкості та легкості впровадження, стали домінуючим стандартом у веб-розробці. REST

базується на принципах архітектури, описаних Роєм Філдінгом, і використовує стандартні HTTP-методи (GET, POST, PUT, DELETE) для роботи з ресурсами, представленими у вигляді унікальних URL. Відсутність стану (statelessness) в REST API дозволяє створювати масштабовані системи, де кожен запит містить всю необхідну інформацію для обробки, що спрощує розподіл навантаження і підвищує надійність [11].

Варто також зазначити, що розвиток API тісно пов'язаний із появою нових форматів даних для обміну інформацією. Спершу домінував XML, який був багатим за можливостями, але важким для обробки і менш ефективним у передачі. Згодом JSON став стандартом де-факто завдяки своїй легкості, читабельності та простоті парсингу, особливо у веб-додатках і мобільних клієнтах. Це значно спростило роботу з REST API і сприяло їхній широкій популярності, оскільки JSON легко інтегрується з JavaScript і багатьма іншими мовами програмування.

Крім того, сучасні REST API часто доповнюються механізмами аутентифікації та авторизації, такими як OAuth, JWT (JSON Web Tokens) та інші, що дозволяє безпечно контролювати доступ до ресурсів [12]. Це особливо важливо у світі, де дані і сервіси розподілені між численними клієнтами і серверами, а безпека є пріоритетом. Використання таких механізмів забезпечує захист від несанкціонованого доступу, підвищує довіру користувачів і відповідає сучасним стандартам інформаційної безпеки.

Таким чином, API як концепція пройшла довгий шлях від простих локальних інтерфейсів до складних, масштабованих і безпечних веб-сервісів, а REST API стала ключовим стандартом, який забезпечує ефективну взаємодію між сучасними програмними системами. Загалом, розуміння природи API і REST API є ключовим для сучасних розробників, оскільки це дозволяє створювати ефективні, масштабовані і безпечні системи, що відповідають вимогам сучасного цифрового світу, сприяючи розвитку інновацій і підвищенню якості програмних продуктів [13].

1.3 Розглядання можливих методів захисту REST API

REST API є популярним способом взаємодії між клієнтами та серверами, що забезпечує гнучкість та масштабованість. Однак, зростання використання API також призводить до збільшення ризиків безпеки, таких як несанкціонований доступ, атаки типу "людина посередині", а також вразливості на рівні даних.

З метою забезпечення безпеки REST API необхідно впроваджувати різноманітні методи захисту, які можуть включати аутентифікацію, авторизацію, шифрування даних та моніторинг активності. Кожен із цих методів відіграє важливу роль у формуванні комплексної системи безпеки, яка здатна ефективно протистояти різним загрозам і атакам. Аутентифікація гарантує, що доступ до API отримують лише легітимні користувачі або системи, авторизація визначає, які саме дії дозволені для кожного користувача, шифрування захищає дані під час їх передачі, а моніторинг активності допомагає своєчасно виявляти підозрілі дії та реагувати на них [14].

Для розуміння необхідності захисту REST API буде розглянуто найбільш ефективні стратегії та технології, які застосовуються на практиці, а також їх переваги та недоліки. Це дозволить не лише знизити ризики компрометації системи, але й підвищити довіру користувачів до сервісів, що використовують API, що є критично важливим у сучасному цифровому середовищі. Впровадження цих заходів забезпечує цілісність, конфіденційність і доступність сервісів, що позитивно впливає на репутацію організації та безпеку її даних.

Першим розглянутим методом захисту буде використання токенів з коротким терміном життя. У сучасних системах аутентифікації токени доступу з коротким терміном життя стали важливим елементом безпеки, особливо в контексті REST API. Цей підхід дозволяє знизити ризики, пов'язані з компрометацією токенів, оскільки навіть якщо зловмисник отримає доступ до токена, його термін дії буде обмежений. Перевагами швидкого закінчення терміну дії є зменшенням ризику отримання тривалого несанкціонованого доступу до захищених ресурсів. Навіть у випадку можливого викрадення сесії зловмисник не

матиме змоги завдати великої шкоди в роботі системи при правильному налаштуванні довготи життя токенів та їх перевірки [15]. Такий підхід також спонукає системи регулярно оновлювати токени, що підвищує загальний рівень безпеки, адже будь-який викрадений токен швидко втрачає актуальність і стає непридатним для використання.

Крім того, короткоживучі токени сприяють більш ефективному контролю доступу, оскільки сервер може частіше перевіряти дійсність сесії і вчасно виявляти підозрілі дії. Це особливо важливо у випадках, коли користувачі працюють з API через публічні або ненадійні мережі, де ризик перехоплення токенів є вищим. Водночас, правильне налаштування часу життя токенів і механізмів їх оновлення дозволяє зберегти баланс між безпекою і зручністю користувачів, уникаючи надмірних перерв у роботі або необхідності частих повторних входів [16].

Таким чином, впровадження токенів з коротким терміном життя є одним із ключових кроків у забезпеченні безпеки REST API, що дозволяє знизити потенційні ризики і підвищити стійкість системи до атак, пов’язаних із викраденням або компрометацією сесійних даних [17].

Освіження (рефреш) токенів є важливим елементом безпеки REST API, що дозволяє підтримувати активність сесій користувачів без шкоди для безпеки системи. Цей механізм дозволяє підтримувати активність сесії користувача, не вимагаючи повторної аутентифікації при кожному запиті. Основна ідея полягає в тому, що користувач отримує два типи токенів: access-токен та refresh-токен [18]. Access-токен - це короткочасний токен, який надає доступ до ресурсів API. Як правило, він має обмежений термін дії. Після закінчення терміну дії access-токена, користувач не зможе виконувати запити до API без повторної аутентифікації. Refresh-токен: Цей токен використовується для отримання нового access-токена без повторної аутентифікації. Refresh-токени зазвичай мають більш тривалий термін дії і зберігаються на стороні клієнта. Використання даного методу дозволяє користувачам залишатися в системі без необхідності повторного введення логіну та пароля, що покращує загальний досвід використання та покращує зручність

користування. Використання короткочасних access-токенів зменшує ризик несанкціонованого доступу. Навіть якщо access-токен буде скомпрометований, його термін дії швидко закінчиться. Системи також можуть використовувати refresh-токени для контролювання доступу до ресурсів, анулюючи їх у разі підозрілої активності або при зміні прав доступу [19].

Multi-Factor Authentication (MFA) є методом аутентифікації, який вимагає від користувача надання двох або більше різних форм підтвердження особи для доступу до ресурсів, таких як додатки, онлайн-акаунти або VPN. Цей підхід значно підвищує рівень безпеки, оскільки зменшує ймовірність несанкціонованого доступу навіть у разі компрометації пароля. MFA додає додаткові шари захисту, що ускладнює завдання зловмисникам, адже для успішного злому недостатньо лише отримати один фактор, наприклад, пароль. Особливо ефективним цей метод стає при застосуванні для чутливих операцій, таких як зміна конфіденційної інформації, проведення фінансових транзакцій або доступ до адміністративних функцій, де ризики компрометації мають найсерйозніші наслідки [20].

Багатофакторна аутентифікація базується на використанні різних типів аутентифікаційних факторів, які можна класифікувати за природою підтвердження:

– фактори знань (Knowledge factors) - це інформація, яку знає користувач, наприклад, пароль, PIN-код або відповіді на контрольні запитання. Вони є найпоширенішим і традиційним способом аутентифікації, але самі по собі часто недостатньо надійні через вразливість до фішингових атак або перебору паролів;

– фактори володіння (Possession factors) - це фізичні об'єкти, які користувач має при собі, наприклад, мобільний телефон, апаратний токен, смарт-картка або USB-ключ безпеки. Ці фактори додають рівень захисту, оскільки доступ до них фізично обмежений, і навіть якщо зловмисник дізнається пароль, без фізичного пристрою він не зможе пройти аутентифікацію;

– фактори впізнавання (Inherence factors) - це фактор, що базуються на унікальних біометричних характеристиках користувача, таких як відбитки

пальців, розпізнавання обличчя, голосу або сітківки ока. Вони забезпечують високий рівень безпеки, оскільки біометричні дані складно підробити або викрасти, але водночас потребують належної обробки і захисту для запобігання витоку чутливої інформації;

– локаційні фактори (Location factors) - це фактори, які визначаються на основі географічного положення користувача, яке може бути встановлене за IP-адресою, GPS-даними або іншими мережевими параметрами. Вони дозволяють додатково оцінити ризики доступу, наприклад, блокуючи спроби входу з незвичних або підозрілих регіонів, що може свідчити про спроби несанкціонованого доступу [21,22].

Використання багатофакторної аутентифікації істотно підвищує загальний рівень безпеки системи, оскільки поєднує переваги різних типів факторів, зменшуючи залежність від одного лише пароля. Це особливо важливо в умовах зростаючих кіберзагроз, коли методи компрометації облікових даних стають все більш витонченими. Багатофакторна аутентифікація допомагає не лише захистити користувачів і їхні дані, але й знижує ризики фінансових втрат, репутаційних збитків і порушення конфіденційності.

Валідація токенів по IP-адресах та пристроях є важливим механізмом контролю доступу, який дозволяє значно підвищити безпеку REST API шляхом обмеження використання сесійних токенів лише до тих мережевих адрес і пристроїв, з яких вони були видані. Цей метод передбачає, що під час аутентифікації користувача система не лише генерує токен доступу, а й фіксує додаткові параметри контексту, зокрема IP-адресу та характеристики пристрою, з якого здійснюється вхід. Ця інформація може бути безпосередньо закодована в самому токені або зберігатися окремо на сервері у вигляді сесійних даних [23].

При кожному подальшому запиті до API система виконує перевірку відповідності поточної IP-адреси і типу пристрою тим, що були зафіксовані під час видачі токена. Якщо виявляється невідповідність - наприклад, запит надходить з іншої IP-адреси або з пристрою, який не був зареєстрований раніше - доступ до ресурсів автоматично блокується або вимагається додаткова

аутентифікація. Такий підхід ефективно запобігає ситуаціям, коли викрадений токен використовується зловмисником на іншому пристрої або в іншій мережі, що є однією з поширених тактик атак на сесії.

Переваги цього методу полягають у тому, що він створює додатковий бар'єр для зловмисників, унеможливлюючи використання викрадених токенів поза межами початкового контексту їх видачі. Це значно знижує ризики несанкціонованого доступу навіть у випадках компрометації токена. Крім того, валідація по IP-адресах та пристроях дозволяє адміністраторам системи вести більш детальний моніторинг і контроль за тим, які саме пристрої і з яких мереж здійснюють доступ до ресурсів, що допомагає вчасно виявляти підозрілу активність і реагувати на потенційні загрози [24].

Водночас цей механізм стимулює більш відповідальне ставлення до безпеки з боку користувачів, адже вони розуміють, що доступ до їх сесій суворо контролюється і прив'язаний до конкретних умов. Це також може допомогти у випадках розслідування інцидентів безпеки, оскільки система зберігає інформацію про контекст використання токенів. Загалом, валідація токенів за IP-адресою та пристроєм є ефективним і практичним інструментом підвищення захищеності REST API, що дозволяє зменшити ризики викрадення сесій та несанкціонованого доступу до критичних ресурсів [25].

Відстеження підозрілої активності є ключовим елементом забезпечення безпеки в системах, що використовують REST API, оскільки дозволяє своєчасно виявляти потенційні загрози, шахрайство або несанкціонований доступ до ресурсів [26]. Цей процес передбачає постійний моніторинг дій користувачів, аналіз їх поведінки та виявлення аномалій, які відрізняються від звичайних моделей використання системи. Вчасне виявлення таких відхилень дозволяє оперативно реагувати, мінімізуючи ризики і запобігаючи можливим інцидентам безпеки. Основні методи відстеження підозрілої активності включають кілька взаємодоповнюючих підходів.

По-перше, системи виявлення аномалій застосовують статистичні методи та алгоритми машинного навчання для глибокого аналізу поведінки користувачів.

Вони порівнюють поточну активність із заздалегідь визначеними або динамічно сформованими моделями нормальної поведінки, що дозволяє виявити незвичні або підозрілі дії. Наприклад, якщо користувач раптово починає виконувати запити з незвичних IP-адрес або у незвичний час доби, система може автоматично сповістити адміністратора або ініціювати додаткові заходи безпеки. Такий підхід дозволяє виявляти навіть складні атаки, які не можуть бути зафіксовані простими правилами.

По-друге, моніторинг шаблонів входу зосереджується на відстеженні частоти, географічного розташування та інших характеристик спроб входу користувачів до системи. Наприклад, багаторазові невдалі спроби входу або одночасні сесії з різних географічних локацій можуть свідчити про спроби перехоплення сесії або використання викрадених облікових даних. Виявлення таких патернів на ранніх етапах дозволяє запобігти подальшому поширенню атаки і захистити систему від компрометації [27].

Третім важливим методом є застосування алгоритмів машинного навчання, які здатні автоматично обробляти великі обсяги даних і виявляти складні патерни шахрайства або аномальної поведінки. Ці алгоритми навчаються на історичних даних, розпізнаючи відмінності між нормальними і підозрілими діями. Вони забезпечують високу точність у виявленні загроз і суттєво знижують кількість помилкових спрацьовувань, що покращує ефективність роботи системи безпеки і зменшує навантаження на аналітиків.

Четвертий метод – аналіз графів – дозволяє досліджувати взаємодії між користувачами або об’єктами у системі, що особливо корисно для виявлення шахрайських мереж або скоординованих атак. Аналізуючи зв’язки і взаємодії, можна виявити групи користувачів, які діють спільно для здійснення шахрайства, що часто буває складно виявити традиційними методами. Цей підхід допомагає не лише ідентифікувати потенційних зловмисників, а й розуміти структуру і масштаби загроз.

Нарешті, реальний моніторинг у режимі реального часу забезпечує миттєве виявлення підозрілої активності відразу після її виникнення. Це дозволяє

оперативно реагувати на загрози, блокувати небезпечні дії і запобігати можливим збиткам. Системи реального моніторингу часто інтегруються з іншими засобами безпеки, такими як системи управління подіями безпеки (SIEM), що підвищує загальну ефективність захисту [28].

Таким чином, відстеження підозрілої активності є комплексним процесом, що об'єднує різні технології і підходи для забезпечення високого рівня безпеки REST API. Використання цих методів дозволяє не лише виявляти потенційні загрози, а й швидко реагувати на них, що є критично важливим у сучасних умовах зростаючих кіберзагроз [29].

Ще один метод забезпечення безпеки REST API є використання токенів на основі HTTPS. Це дозволяє захистити дані користувачів під час передачі і зменшити ризики несанкціонованого доступу до ресурсів системи. HTTPS (Hypertext Transfer Protocol Secure) - це протокол, який забезпечує шифрування всього трафіку між клієнтом і сервером за допомогою технологій SSL (Secure Sockets Layer) або TLS (Transport Layer Security). Завдяки цьому всі дані, включно з токенами аутентифікації, передаються у зашифрованому вигляді, що унеможливлює їх перехоплення або модифікацію зловмисниками під час передачі.

Використання HTTPS гарантує не лише конфіденційність переданих даних, а й автентичність сервера, що дозволяє клієнтам впевнено встановлювати з'єднання саме з тим сервером, до якого вони звертаються, уникаючи атак типу «Man-in-the-Middle» (людина посередині). Під час встановлення HTTPS-з'єднання відбувається процес «рукостискання» (handshake), під час якого клієнт і сервер узгоджують параметри шифрування і перевіряють сертифікати, що підтверджують автентичність сервера. Після цього весь обмін даними відбувається через захищений канал, що забезпечує цілісність і недоторканність інформації.

Для REST API, які часто передають чутливу інформацію такі, як токени доступу, паролі, персональні дані або платіжні реквізити, використання HTTPS є обов'язковою умовою безпеки. Без цього навіть найкращі механізми

аутентифікації і авторизації можуть бути скомпрометовані через перехоплення трафіку.

Впровадження HTTPS також підвищує довіру користувачів і клієнтів сервісу, оскільки браузери і клієнтські додатки відображають індикатори безпеки, які сигналізують про захищене з’єднання. Це особливо важливо у комерційних і фінансових сервісах, де безпека даних є пріоритетом [30].

Після успішної аутентифікації користувача сервер генерує унікальний токен, який містить інформацію про користувача та його права доступу. Цей токен зазвичай підписується за допомогою криптографічних методів для забезпечення цілісності. Користувач включає токен у заголовки HTTP-запитів або передає його через параметри URL. Важливо, щоб цей процес відбувався через захищене з'єднання HTTPS, що запобігає перехопленню токена під час передачі [31]. Тоді сервер перевіряє дійсність отриманого токена, включаючи перевірку його підпису та терміну дії. Якщо токен є дійсним, сервер надає доступ до запитуваних ресурсів. HTTPS шифрує дані, що передаються між клієнтом і сервером, роблячи їх недоступними для зловмисників. Це особливо важливо для токенів, які можуть містити чутливу інформацію про користувача. Використання HTTPS забезпечує цілісність даних, що передаються, запобігаючи їх зміні під час передачі. HTTPS дозволяє клієнту перевірити автентичність сервера, з яким він взаємодіє, що знижує ризик атак "людина посередині" (MITM).

Контроль часу активності та сповіщення користувача у системах, що використовують REST API, є важливим аспектом безпеки, який допомагає забезпечити захист облікових записів і зменшити ризики несанкціонованого доступу. Цей процес дозволяє виявляти періоди неактивності користувачів, своєчасно інформувати їх про можливе завершення сесії, а також оперативно реагувати на підозрілу або аномальну активність, що може свідчити про спроби компрометації облікового запису.

Основні елементи контролю часу активності включають кілька взаємопов’язаних механізмів. По-перше, визначення періоду активності – системи встановлюють певний часовий інтервал, протягом якого користувач повинен

взаємодіяти з API, наприклад, 15-30 хвилин. Якщо протягом цього часу жодної активності не зафіксовано, сесія автоматично завершується. Такий підхід дозволяє мінімізувати ризики, пов’язані з тим, що користувач може залишити свій пристрій без нагляду, і зловмисник отримає можливість використати відкриту сесію для несанкціонованого доступу до конфіденційних даних або функцій системи.

По-друге, сповіщення про неактивність є важливим елементом, який покращує взаємодію з користувачем і підвищує рівень безпеки. Перед автоматичним завершенням сесії система може надсилати попереджувальні повідомлення, які відображаються на екрані користувача, інформуючи його про наближення завершення сесії через відсутність активності. У випадках, коли мова йде про особливо чутливі дії або сервіси, додатково можуть використовуватися сповіщення через Email або SMS, що дозволяє користувачу бути в курсі стану свого облікового запису навіть якщо він не перебуває безпосередньо в інтерфейсі додатку.

По-третє, у разі завершення сесії через неактивність система повинна вимагати повторної аутентифікації для відновлення доступу. Це забезпечує додатковий рівень захисту, оскільки підтверджує, що саме легітимний користувач намагається продовжити роботу із системою. Такий підхід запобігає можливості використання відкритої сесії сторонніми особами і знижує ризики компрометації.

Четвертим важливим аспектом є моніторинг активності користувачів. Системи ведуть детальний журнал дій, фіксуючи всі спроби входу, виходу, змін налаштувань та інші значущі події. Це дає змогу адміністраторам аналізувати поведінку користувачів, виявляти нетипові або підозрілі дії, а також проводити аудит безпеки. Наявність таких логів є критичною для розслідування інцидентів і прийняття своєчасних заходів.

Нарешті, реакція на підозрілу активність є ключовим елементом системи безпеки. У разі виявлення аномалій, наприклад, частих невдалих спроб входу, доступу з незвичних географічних локацій або одночасного використання сесії з різних пристроїв, система може автоматично блокувати обліковий запис,

надсилати сповіщення адміністраторам для подальшого розслідування або ініціювати додаткові заходи безпеки, такі як вимога багатофакторної аутентифікації. Такий підхід дозволяє оперативно реагувати на потенційні загрози і мінімізувати можливі збитки від атак.

Таким чином, контроль часу активності та сповіщення користувача є комплексним механізмом, який не тільки підвищує безпеку REST API, але й покращує користувацький досвід, забезпечуючи баланс між захистом і зручністю використання системи [32].

Запобігання XSS-атакам є критично важливим для забезпечення безпеки веб-додатків. XSS-атаки дозволяють зловмисникам впроваджувати шкідливі скрипти на веб-сторінки, що може призвести до крадіжки чутливої інформації, такої як “cookie” користувачів або сесійні токени. Приклад роботи XSS-атаки наведено на рисунку 1.3.

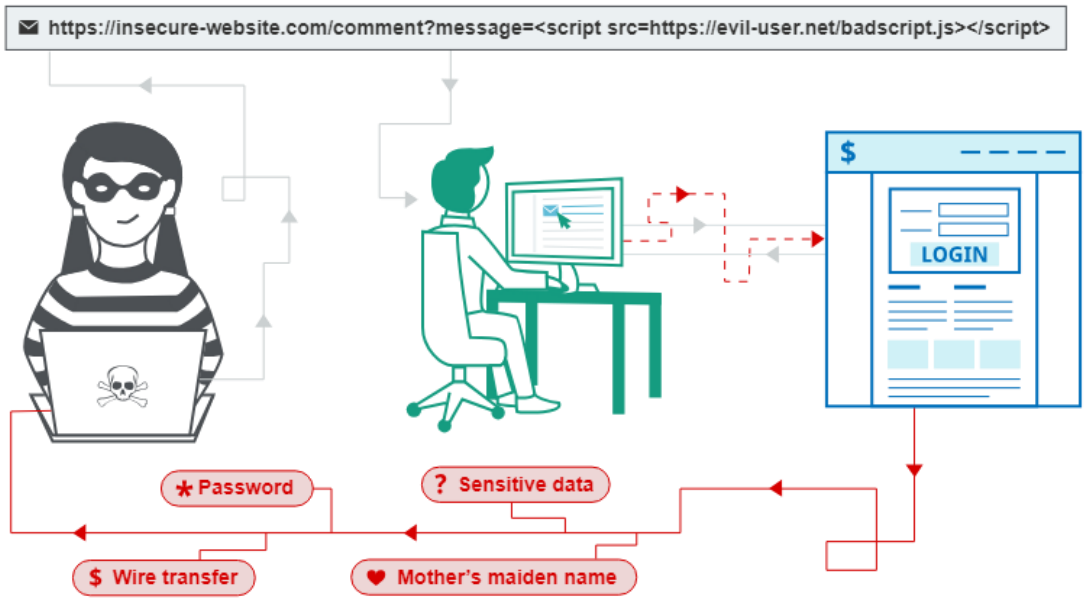


Рисунок 1.3 – Ілюстрація роботи XSS-атаки

Для ефективного захисту від XSS-атак рекомендується впроваджувати кілька ключових методів, які разом формують комплексний підхід до запобігання виконанню шкідливого коду у веб-додатках [33]. Ці методи охоплюють різні

етапи обробки даних - від моменту їх отримання від користувача до відображення у браузері.

Першим і одним із найважливіших кроків є валідація введення. Всі дані, що надходять від користувача, повинні проходити ретельну перевірку на відповідність визначеним критеріям, таким як тип, довжина, формат та допустимі символи. Валідація включає очищення або фільтрацію потенційно небезпечних символів і послідовностей, які можуть бути використані для впровадження шкідливого JavaScript-коду. Для цього часто застосовують регулярні вирази, які дозволяють виявляти і блокувати небезпечні патерни у вхідних даних. Такий підхід допомагає запобігти проникненню шкідливих скриптів на ранньому етапі обробки, зменшуючи ризик їх подальшого виконання.

Другим важливим заходом є кодування виводу, яке запобігає інтерпретації браузером потенційно небезпечних символів як виконуваного коду. Цей процес передбачає заміну спеціальних символів, таких як <, >, &, ", ' на відповідні HTML-сутності. Завдяки цьому браузер відображає ці символи як текст, а не як частину скрипта, що виконується. Для спрощення цього процесу часто використовують спеціалізовані бібліотеки безпеки контенту, які автоматично виконують кодування і допомагають уникнути помилок, що можуть призвести до вразливостей.

Третім методом є використання Content Security Policy (CSP) – потужного механізму безпеки, який дозволяє розробникам визначати політики, що контролюють, які ресурси можуть завантажуватися і виконуватися на веб-сторінці. CSP дозволяє обмежити виконання скриптів лише з довірених джерел, блокуючи будь-які неавторизовані або підозрілі скрипти. Це значно знижує ризик успішної XSS-атаки, оскільки навіть якщо шкідливий код буде впроваджений, браузер не дозволить його виконання без відповідного дозволу.

Четвертий аспект захисту полягає у дотриманні безпечних практик програмування. Розробники повинні уникати використання inline-скриптів, які є однією з найбільш вразливих точок для XSS. Важливо застосовувати сучасні фреймворки і бібліотеки, які забезпечують автоматичне перетворення і кодування

даних, а також мають вбудовані механізми захисту від XSS. Регулярні аудити безпеки, код-рев'ю і тестування на вразливості допомагають виявляти потенційні проблеми на ранніх стадіях розробки і своєчасно їх усувати.

П'ятий і не менш важливий метод – це регулярні оновлення безпеки систем і фреймворків. Вчасне впровадження останніх патчів і оновлень дозволяє закривати відомі вразливості, які можуть бути використані зловмисниками для здійснення XSS-атак. Постійне підтримання актуальності програмного забезпечення є запорукою збереження високого рівня безпеки веб-додатків.

Таким чином, поєднання валідації введення, кодування виводу, впровадження CSP, дотримання безпечних практик програмування та регулярних оновлень безпеки створює надійний захист від XSS-атак. Ці заходи дозволяють мінімізувати ризики впровадження шкідливих скриптів і забезпечують безпечну взаємодію користувачів із веб-додатками, що особливо важливо для REST API, які часто обробляють чутливі дані [34].

Усі ці різні методи захисту REST API від атак на основі підбору сесій є багатогранними і вимагають комплексного підходу. Впровадження зазначених стратегій дозволяє значно підвищити рівень безпеки системи та забезпечити захист чутливих даних від несанкціонованого доступу.

2 ВИБІР, АНАЛІЗ ТА ЗАСТОСУВАННЯ ІНСТРУМЕНТІВ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

2.1 Порівняльний аналіз різних інструментів для тестування REST API

Для виконання даної роботи знадобиться веб-додаток на якому будуть проводитися тестування різних методів захисту REST API та інструмент завдяки якому будемо проводити данні тести. Перед початком роботи я провів аналіз різних платформ API та доступних додатків завдяки яким можливо провести потрібні мені тестування із отриманням бажаного результату.

Серед основних інструментів для тестування безпеки API можна виділити такі, як SoapUI, JMeter, Insomnia REST, Nessus, Postman, Akto, Checkmarx, а також ReadyAPI. Кожен з них має свої переваги та недоліки, залежно від специфіки задачі, тому задля уточнення вибору інструменту для проведення якісної оцінки поставленої задачі, буде проведений більш детальніше порівняння вибраних можливостей.

Першим із потенційних інструментів є SoapUI. Це доволі потужний інструмент для тестування API, який підтримує як функціональне, так і безпекове тестування. Його можливості безпекового тестування включають виявлення вразливостей, таких як SQL-ін’єкції, XSS, а також перевірку автентифікації та авторизації. З версії 4.0 SoapUI має вбудовані функції безпекового тестування, що робить процес перевірки функціональної безпеки цільових сервісів простим і зручним. Інструмент підтримує тестування, як REST, так і SOAP та GraphQL, що робить його універсальним для різних типів API. Однак SoapUI іноді може бути складним для початківців через насичений функціонал та інтерфейс. SoapUI дозволяє створювати проекти для тестування API, де можна налаштувати різні запити та перевірки. Наприклад, можна створити тест, який надсилає запит до API і перевіряє, що отримана відповідь містить очікувані дані або відповідає певним критеріям. Інструмент підтримує автоматизацію тестування, генерацію тестових даних і аналіз результатів, що робить його ефективним для комплексного тестування веб-сервісів. Для REST API можна легко створювати запити з різними

HTTP-методами , як GET, POST, PUT, DELETE, а також працювати з різними форматами даних, такими як JSON чи XML. Однією з переваг SoapUI є можливість інтеграції з CI/CD процесами, що дозволяє запускати тести автоматично при кожному оновленні коду, забезпечуючи безперервний контроль якості. Крім того, SoapUI підтримує створення складних сценаріїв тестування з використанням скриптів на Groovy, що дає змогу адаптувати тести під специфічні вимоги проекту. Інструмент також дозволяє легко переключатися між різними середовищами розробки, тестування, що спрощує управління конфігураціями.

Наступним із можливих засобів для тестування вибір пав на JMeter. Даний інструмент є доволі відомим для навантажувального тестування, але він також має можливості для безпекового тестування. Він дозволяє виконувати такі види безпекового тестування, як Site Spidering (збір інформації про доступні ресурси), Fuzzing (введення випадкових або некоректних даних для виявлення вразливостей), а також імітацію атак типу DDoS. JMeter є відкритим і безкоштовним рішенням, що робить його привабливим варіантом у порівнянні з дорогими комерційними продуктами. Проте його використання для безпеки API вимагає додаткової настройки і знання в області скриптування, що може ускладнити процес для більшості звичайних користувачів.

Ще одним інструментом можна відокремити Insomnia. Insomnia є найбільш сучасним на момент проведення даного аналізу та оцінки інструментом для розробки, тестування та налагодження API, який підтримує автоматизоване тестування безпеки. Він дозволяє організовувати запити у структуровані колекції, використовувати скрипти до і після запитів, а також працювати з різними середовищами (development, staging, production). Insomnia також має CLI для інтеграції в CI/CD процеси, що дозволяє автоматизувати запуск тестів. Важливою перевагою є простий та інтуїтивний інтерфейс, що робить його зручним для команд, які швидко ітеративно розробляють API. Однак Insomnia більше орієнтований на функціональне тестування і налагодження, ніж на комплексне безпекове тестування [35].

Наступним інструментом буде Nessus Expert. Даний продукт є потужнім сканером вразливостей, який включає динамічне тестування безпеки веб-додатків і API (DAST). Він дозволяє виявляти відомі вразливості, включаючи OWASP Top 10, проблеми з SSL/TLS, HTTP-заголовками та інші типові помилки конфігурації. Nessus забезпечує високоточний аналіз із мінімумом хибних спрацьовувань і підходить для глибокого аудиту безпеки. Проте значним недостатком даного вибору інструменту полягає в тому, що він більше орієнтований на безпекових аналітиків і пентестерів, ніж на розробників, і часто вимагає значних ресурсів та налаштувань.

Оскільки дана робота пов’язана на роботі із певним стилем побудови архітектури API через HTTP запроси, я вважаю необхідним обговорити Postman, як один із найпопулярніших і найуніверсальніших інструментів для роботи з API, що поєднує в собі простоту використання, потужний функціонал та широкі можливості інтеграції, що робить його ідеальним вибором для тестування безпеки REST API. Його інтуїтивний та зручний інтерфейс дозволяє швидко створювати, запускати та автоматизувати тести навіть користувачам без глибоких технічних знань, що значно спрощує процес тестування. Postman підтримує різні типи API, включно з REST, SOAP та GraphQL, завдяки чому він є універсальним інструментом, здатним задовольнити потреби різних проектів. Особливо важливою є можливість написання складних тестових сценаріїв на JavaScript, що дає змогу реалізувати перевірки автентифікації, авторизації, обробки помилок, захисту від ін’єкцій та інших аспектів безпеки, що є критично необхідним для комплексного тестування API. Інтеграція CI/CD через Newman , що являє собою інструмент командного рядка для запуску колекцій Postman, дозволяє автоматизувати безпекове тестування на різних етапах розробки, забезпечуючи безперервну перевірку якості API, що підвищує надійність продукту і скорочує час виявлення помилок [36]. Крім того, Postman підтримує колаборативні функції, такі як спільні робочі простори, контроль версій, можливість обміну колекціями та середовищами, що сприяє ефективній командній роботі і полегшує передачу знань між розробниками, тестувальниками та іншими учасниками проекту.

Широка спільнота користувачів і активна підтримка забезпечують доступ до численних ресурсів, шаблонів тестів, документації та регулярних оновлень, що робить Postman живим і постійно вдосконалюваним інструментом.

Важливо також відзначити, що Postman пропонує безкоштовну базову версію з можливістю розширення функціоналу за рахунок платних планів, що робить його доступним для проектів різного масштабу - від стартапів до великих корпорацій. Цей інструмент дозволяє не лише надсилати запити і отримувати відповіді, а й аналізувати їх у зручному форматі (JSON, XML, та інші), писати тести, які перевіряють статуси HTTP, час відгуку, наявність конкретних значень у відповіді, а також складні бізнес-логіки, що значно підвищує якість і надійність API. Можливість використання змінних і середовищ робить автоматизацію тестування гнучкою і адаптивною до різних сценаріїв, а функції Pre-request Script дозволяють генерувати токени, керувати логікою тестів і створювати ланцюжки запитів. Крім того, Postman підтримує автоматичну генерацію документації API на основі колекцій запитів, що спрощує процес створення та оновлення документації, роблячи API більш зрозумілим для команди і користувачів. Можливість емулювати серверні відповіді за допомогою mock-серверів дозволяє тестувати UI та інші компоненти системи ще до повної готовності бекенду, що пришвидшує розробку і знижує ризики. Завдяки цим широким можливостям Postman поєднує в собі простоту, гнучкість, потужність і можливість інтеграції в сучасні DevOps процеси, що відрізняє його від інших інструментів, які часто або надто складні, або орієнтовані на вузькі задачі. Таким чином, Postman забезпечує оптимальний баланс між функціональністю і зручністю, що робить його найкращим вибором для виконання поставленої задачі тестування методів захисту REST API.

Незважаючи на детальний аналіз всіх попередньо перечислених додатків для виконання поставленої задачі, ще досі залишилися нерозглянуті інструменти, які мають потенціал у задоволені виконання необхідного тестування, тому наступним розгляденим елементом виступає Akto. Akto є спеціалізованою платформою для безпеки API, яка може запропонувати автоматичне виявлення

API, понад 400 вбудованих тестів безпеки, управління безпековим станом і захист у реальному часі. Вона інтегрується з CI/CD і дозволяє ефективно управляти безпекою API на всіх етапах життєвого циклу. Akto добре підходить для великих організацій, які потребують комплексного рішення для безпеки API, але може бути надмірним для невеликих проектів.

Ще одним нерозглянутим інструментом є Checkmarx, який спеціалізується на статичному аналізі коду (SAST) і динамічному тестуванні (DAST) для виявлення вразливостей у коді API без його виконання. Це дозволяє розробникам виявляти проблеми безпеки на ранніх етапах розробки. Платформа підтримує широкий спектр тестів, включаючи ін’єкції, проблеми автентифікації та витік даних [37]. Checkmarx підходить для організацій, які прагнуть інтегрувати безпеку безпосередньо у процес розробки, але вимагає значних ресурсів для впровадження.

На кінець буде розглянута, хоча і останньою, але не найгіршим вибором, комплексна платформа ReadyAPI, що об’єднує функціональне, безпекове та навантажувальне тестування API в одному інтерфейсі. Вона підтримує різні протоколи такі ,як REST, SOAP, GraphQL, Kafka. Має вбудовані шаблони безпекових сканів і дозволяє створювати складні сценарії тестування без глибоких знань програмування. ReadyAPI також інтегрується з CI/CD, підтримує віртуалізацію сервісів і командну роботу. Проте це комерційний продукт з відповідною вартістю, що може бути недоцільним для невеликих проектів.

Після детальної оцінки та порівняння різноманітних інструментів, які могли б бути використані для проведення необхідних тестів, вибір був зроблений на користь Postman. Цей інструмент вирізняється серед інших завдяки своїй універсальності, адаптивності та широкому спектру можливостей, які дозволяють ефективно реалізовувати тестування REST API. Порівняння враховувало як функціональні характеристики, так і зручність використання, а також інтеграційні можливості, що є важливими для забезпечення безперервності процесу тестування в сучасних розробницьких середовищах. Інші популярні рішення, такі як SoapUI, Rest-Assured, Nessus чи Akto, мають свої сильні сторони, проте часто

вони або занадто спеціалізовані, або вимагають значних зусиль для налаштування, що може знижувати швидкість і гнучкість роботи. Постійна підтримка, активна спільнота користувачів і регулярні оновлення Postman також стали вагомими факторами при виборі, оскільки це гарантує актуальність інструменту та доступність широкого спектру ресурсів для навчання і вирішення можливих проблем. Враховуючи співвідношення простоти, потужності та можливостей інтеграції, Postman є оптимальним інструментом для виконання поставленої задачі, що забезпечує баланс між ефективністю, зручністю та масштабованістю тестування.

2.2 Обґрунтування, створення та використання веб додатку

Незважаючи на проведений попередньо аналіз та уточнення вибору інструменту для проведення тестів, для проведення цих самих тестів необхідний об’єкт тестування. У даному випадку об’єктом проведення випробувань на різні методи захисту REST API від викрадення сесії буде спеціально створений веб-додаток, на якому будуть проведені відповідні тести. У даному випадку таким об’єктом є спеціально створений веб-додаток за основу якого взято OWASP Juice Shop, який широко використовується у сфері інформаційної безпеки для навчання, демонстрацій та тестування вразливостей, зокрема REST API. Даний веб-додаток був обраний оскільки OWASP Juice Shop вважається одним із найсучасніших і найскладніших навмисно вразливих веб-додатків, що імітує реальні умови сучасних веб-сервісів, включаючи різноманітні вразливості з OWASP Top Ten та інші помилки безпеки, які часто зустрічаються у реальних додатках [38]. Цей додаток створений з метою навчання розробників, тестувальників і фахівців із безпеки, дозволяючи їм на практиці вивчати типові помилки в реалізації захисту, а також методи їх виявлення і усунення.

Особливо важливим для нашого дослідження є те, що взятий за основу OWASP Juice Shop містить численні вразливості, пов’язані з REST API, що дає

змогу комплексно тестувати різні методи захисту від викрадення сесій, авторизаційних помилок, ін’єкційних атак, а також інших типів атак, що можуть бути спрямовані на API. Даний сайт включає в себе спеціальні REST-ендпоінти, які надають доступ до уразливих фрагментів коду, пов’язаних із різними викликами безпеки. Ці ендпоінти використовуються, наприклад, у CTF-розширенні Juice Shop для надання підказок учасникам, що дозволяє глибше розуміти суть вразливостей і механізми їх експлуатації. Така інтеграція робить Juice Shop не лише навчальним інструментом, а й платформою для практичного відпрацювання навичок безпеки API. З точки зору захисту REST API, даний веб-додаток демонструє низку ключових аспектів безпеки, а також типових помилок, які часто зустрічаються в реальних проєктах. Зокрема, у додатку реалізовано авторизацію через JSON Web Token, але при цьому існують відомі проблеми з перевіркою прав доступу, що дозволяє здійснювати обходи авторизації. Вразливість, пов’язана з можливістю створення адміністративних облікових записів через необроблені POST-запити без належної автентифікації, є яскравим прикладом критичних помилок у захисті API. Це ілюструє важливість послідовної та комплексної перевірки прав доступу на всіх рівнях API.

Крім того, під час проведення тестів можливо буде скористуватися вразливості, пов’язані з ін’єкціями, зокрема SQL-ін’єкції, які дозволяють зловмиснику отримати доступ до даних або обійти механізми автентифікації. Такі вразливості є типовими для REST API, які працюють із базами даних, і демонструють необхідність належної обробки вхідних даних і використання параметризованих запитів. Веб-додаток також схильний до XSS-атак , у тому числі DOM-базованих, що відображається у вразливості пошукової функції, де введення шкідливого коду не проходить належної санітизації. Це підкреслює важливість як валідації вхідних даних, так і безпечного кодування вихідних даних у відповідях API. Ще одним важливим елементом захисту в додатку є використання механізмів CAPTCHA для підтвердження дій користувача, що допомагає запобігати автоматизованим атакам, наприклад, брутфорс-атакам на автентифікацію. Веб-додаток також реалізує allowlist для перенаправлень, що

знижує ризик атак типу open redirect, які можуть бути використані для фішингу чи викрадення сесій. Архітектура його REST API побудована на сучасних технологіях, таких як Node.js, Express і Sequelize, що дозволяє ефективно управляти даними і забезпечувати взаємодію між клієнтською частиною і сервером через стандартизовані HTTP-запити. Використання middleware finale-rest автоматизує створення кінцевих точок API для основних операцій CRUD , як створення, читання, оновлення, видалення, що робить API гнучким і легко масштабованим.

REST API веб-додатку організовано за принципами RESTful, де кожен ресурс має свій унікальний URI, а взаємодія відбувається через стандартні HTTP-методи: GET для отримання даних, POST для створення, PUT і PATCH для оновлення, DELETE для видалення. Формат обміну даними – JSON – забезпечує високу сумісність із різними клієнтами, включаючи веб-браузери, мобільні додатки та інші сервіси. Така організація дозволяє легко інтегрувати даний веб-додаток з іншими системами або використовувати його API для автоматизованого тестування.

Однією з ключових особливостей REST API веб-додатку є реалізація аутентифікації та авторизації. Після успішного входу користувач отримує токен сесії, який використовується для доступу до захищених ресурсів. Проте, оскільки даний веб-додаток створений як навчальний елемент із навмисними вразливостями та недоробками, він демонструє типові проблеми безпеки, такі як відсутність шифрування трафіку, по типу використання HTTP замість HTTPS, слабкі налаштування cookie, можливості для XSS і сесійного викрадення. Це робить його ідеальним майданчиком для практики виявлення і експлуатації вразливостей, а також для відпрацювання методів їх усунення. Проте структура API відповідає загальноприйнятим стандартам, що дозволяє легко створювати запити вручну або автоматизовано за допомогою інструментів, таких як Postman. Це сприяє гнучкості у тестуванні і розробці, а також дозволяє користувачам зосередитися на вивченні безпеки, не витрачаючи час на вивчення складної

документації. Переглянути імплементовані об’кти для подальшого використання можливо у рисунку 2.1.

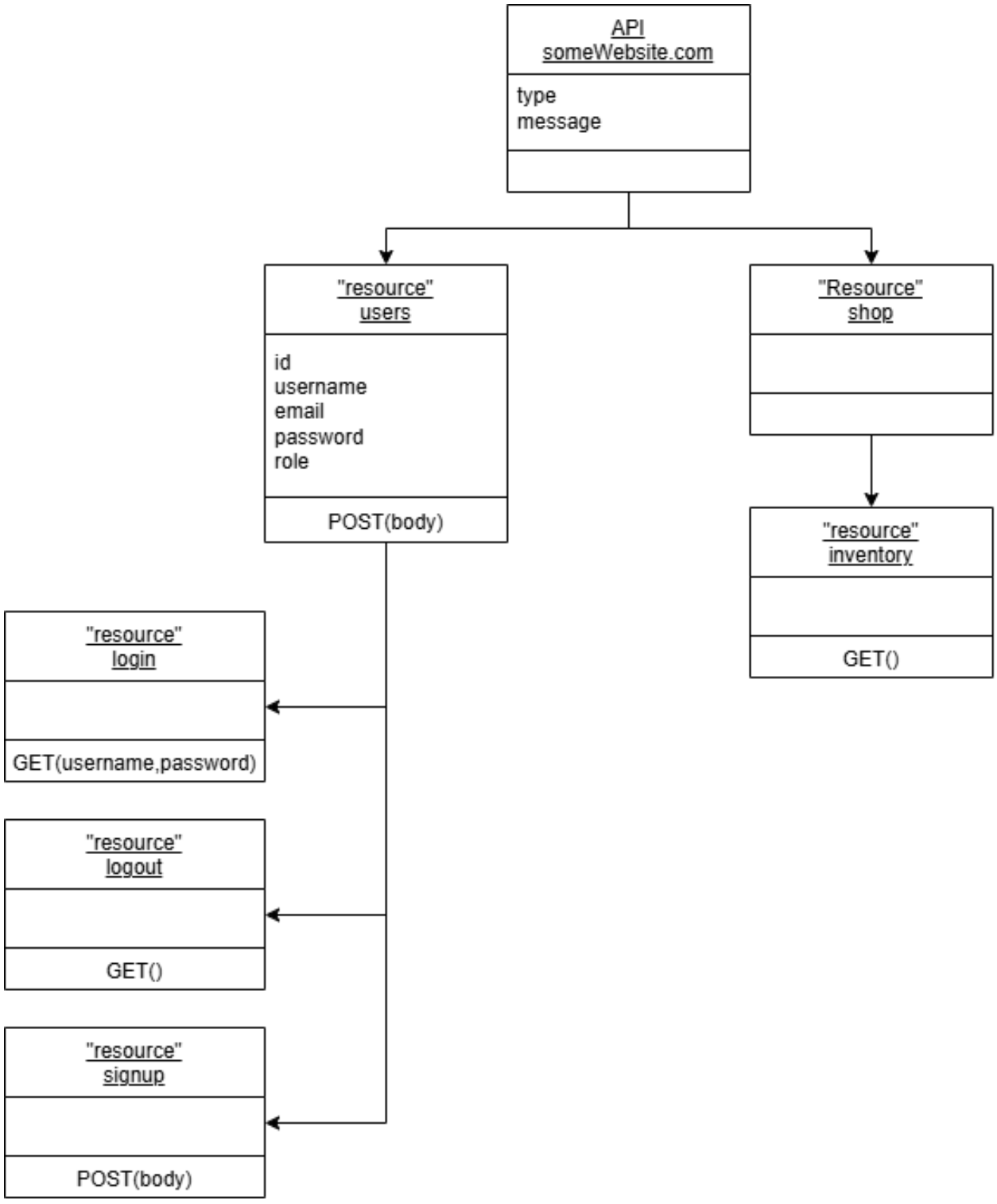


Рисунок 2.1 – Приклад впровадження шкідливого скрипту

Важливо також відзначити, що REST API веб-додатку підтримує роботу з різними типами ресурсів, включаючи користувачів, продукти, замовлення,

відгуки, що імітує реальні бізнес-процеси. Це дозволяє моделювати різноманітні сценарії атак і захисту, наприклад, викрадення сесій у контексті замовлень або зміну даних користувача. Такий підхід робить створений веб-додаток не просто демонстраційним проектом, а й надає можливості покращити свої практичні навички у схожих сферам безпеки.

Загалом, REST API веб-додатку є типовим прикладом сучасної веб-сервісної архітектури з усіма необхідними компонентами для роботи з ресурсами, а також із навмисними вразливостями. Він ілюструє, як правильно організувати API, які операції мають бути підтримані, і одночасно демонструє потенційні слабкі місця, які часто зустрічаються у реальних проектах. Це дозволяє розробникам, тестувальникам і фахівцям із безпеки глибше розуміти механізми роботи REST API, виявляти і усувати вразливості, а також впроваджувати ефективні заходи захисту. Таким чином, веб-додаток слугує не лише навчальним майданчиком, а й практичним інструментом для відпрацювання навичок тестування безпеки REST API, що є критично важливим у сучасному цифровому світі, де безпечна взаємодія між клієнтом і сервером є основою довіри користувачів і стабільності бізнесу.

У модулі insecurity.js, який містить більшість функцій, пов’язаних із безпекою, реалізовані як слабкі, так і відносно сильні хеш-функції, а також логіка авторизації маршрутів через JWT з використанням функцій denyAll(), isAuthorized() та authorize(). Проте, оскільки цей модуль навмисно містить помилки, він ілюструє, як неправильна або непослідовна реалізація цих механізмів може призводити до серйозних проблем безпеки.

Загалом, даний відкритий веб-додаток був взятий за основу та перероблений для проведення необхідних випробувань, оскільки OWASP Juice Shop є комплексним середовищем, яке імітує реальні умови роботи веб-додатків із REST API, демонструючи як типові, так і складні уразливості, що пов’язані з автентифікацією, авторизацією, обробкою даних, захистом сесій, а також іншими аспектами безпеки. Завдяки цьому він є ідеальним об’єктом для проведення тестів на різні методи захисту REST API від викрадення сесій та інших атак, що дозволяє

не лише виявляти вразливості, а й відпрацьовувати ефективні стратегії їх усунення.

2.3 Обґрунтування вибору та використання додаткових інструментів

Для проведення більшості тестів можуть знадобитися додаткові інструменти, завдяки яким отримаємо змогу отримувати необхідні вхідні данні для проведення поставлених тестів із ціллю покращити безпеку прости певних методів викрадення сесії.

Першим важливим інструментом буде розглянуто Burp Suite. Даний додаток є комплексним інструментом для тестування безпеки веб-додатків, який широко використовується пентестерами та фахівцями з інформаційної безпеки. Його популярність для задач, пов’язаних із виявленням і експлуатацією вразливостей REST API, зумовлена багатьма факторами.

По-перше, Burp Suite пропонує потужний проксі-сервер, який дозволяє перехоплювати, аналізувати і змінювати HTTP/HTTPS трафік у реальному часі. Це дає змогу детально досліджувати механізми аутентифікації, сесійного управління і виявляти потенційні уразливості, такі як сесійне викрадення, XSS, SQL-ін’єкції та інші [39].

Причиною вибору Burp Suite для задачі тестування сесійного викрадення є його гнучкість і багатофункціональність. Інструмент містить різні модулі, зокрема Proxy для перехоплення трафіку, Intruder для автоматизованого перебору параметрів і вразливостей, Repeater для повторного відправлення і модифікації запитів, а також Scanner для автоматичного пошуку вразливостей. Особливо корисною є можливість створення макросів та правил обробки сесій, що дозволяє автоматизувати роботу з токенами авторизації, наприклад, оновлювати JWT-токени під час сканування. Це робить Burp Suite незамінним для глибокого і комплексного тестування REST API.

Переваги Burp Suite над іншими альтернативами полягають у його інтегрованості, широкому функціоналі і підтримці розширень. На відміну від

більш простих проксі або сканерів, Burp Suite дозволяє не лише виявляти вразливості, а й гнучко керувати процесом тестування, налаштовувати автоматичне оновлення сесійних токенів, аналізувати відповіді сервера, виконувати складні атаки типу брутфорс або ін'єкції. Крім того, Burp Suite має активну спільноту і великий вибір плагінів, що розширюють його можливості. Професійна версія інструменту підтримує паралельне сканування, інтеграцію з CI/CD і дозволяє працювати з сучасними механізмами аутентифікації, такими як JWT.

Хоча існують альтернативи, наприклад OWASP ZAP, Fiddler або Postman, вони часто не мають такого ж рівня інтеграції і гнучкості для комплексного пентестингу. OWASP ZAP, хоч і безкоштовний і відкритий, іноді поступається Burp Suite за зручністю і швидкістю роботи. Postman більше орієнтований на тестування API з точки зору функціональності, а не безпеки, і не має вбудованих засобів для автоматизованого виявлення вразливостей. Fiddler є потужним мережевим аналізатором, але не спеціалізується на пентестингу.

Отже, Burp Suite є оптимальним вибором для задачі тестування REST API на предмет сесійного викрадення завдяки своїй комплексності, можливостям автоматизації, підтримці сучасних технологій аутентифікації і широкому набору інструментів для аналізу і модифікації трафіку. Його використання дозволяє проводити глибокий аудит безпеки, швидко виявляти і експлуатувати вразливості, що робить його незамінним інструментом у роботі пентестера та фахівця з кібербезпеки.

Не зважаючи на весь перелічений функціонал Burp Suite та його потенційні використання для виконання поставленої задачі, вибір пав на ще один інструмент, який спеціалізується на зчитуванні мережевих пакетів, Wireshark. Даний застосунок є одиним із найпотужніших і найпопулярніших інструментів для захоплення та аналізу мережевого трафіку, який широко застосовується фахівцями з безпеки, системними адміністраторами та розробниками. Його основна перевага полягає у можливості глибокого аналізу пакетів на різних рівнях мережевого стеку - від канального до прикладного, що дозволяє детально

досліджувати як загальний стан мережі, так і специфічний трафік, наприклад, HTTP-запити REST API.

Для задачі тестування захисту REST API від сесійного викрадення Wireshark є незамінним інструментом, оскільки він дозволяє перехоплювати трафік у реальному часі або аналізувати раніше записані дампи. Це дає змогу виявити, чи передаються сесійні токени у відкритому вигляді, чи використовуються захищені протоколи, а також простежити поведінку клієнта і сервера на рівні мережевих пакетів. Особливо корисною є можливість застосування складних фільтрів для відбору трафіку за IP-адресами, портами, протоколами або конкретними параметрами HTTP, що спрощує локалізацію і аналіз потрібних даних.

Причини використання Wireshark у цій задачі полягають у його універсальності і гнучкості. Він не обмежується лише HTTP-трафіком, а підтримує сотні мережевих протоколів, що дозволяє виявляти проблеми не лише на рівні додатку, а й у мережевій інфраструктурі. Wireshark дозволяє детально дослідити, як саме передаються сесійні токени, чи немає витоків через незашифровані канали, а також виявити аномалії, які можуть свідчити про атаку типу MITM або інші загрози.

Переваги Wireshark над іншими альтернативами, такими як tcpdump, Fiddler чи Charles Proxy, полягають у його потужному графічному інтерфейсі, що робить аналіз мережевого трафіку більш інтуїтивним і зручним. На відміну від tcpdump, який є командним рядком і більше орієнтований на автоматизацію, Wireshark дає змогу візуалізувати пакети, застосовувати багаторівневі фільтри, переглядати структуру протоколів і слідувати за потоками TCP чи HTTP. У порівнянні з проксі-інструментами, які фокусуються переважно на HTTP/HTTPS, Wireshark охоплює весь спектр мережевих протоколів, що особливо важливо для комплексного аналізу безпеки [40].

Ще однією важливою перевагою є підтримка захоплення трафіку на різних типах мережевих інтерфейсів, включно з Ethernet, Wi-Fi, VPN, а також спеціальними адаптерами для перехоплення loopback-трафіку. Це дає можливість

аналізувати навіть внутрішні виклики REST API, які не виходять у зовнішню мережу.

Загалом, Wireshark є незамінним інструментом для глибокого і всебічного аналізу мережевого трафіку, що робить його ідеальним для виявлення проблем із сесійним викраденням у REST API. Його потужний функціонал, гнучкість у налаштуванні фільтрів і зручний інтерфейс забезпечують ефективну роботу як початківцям, так і досвідченим фахівцям із безпеки. Використання Wireshark у поєднанні з іншими інструментами, такими як Burp Suite і Postman, дозволяє отримати повну картину безпеки API і своєчасно виявляти та усувати вразливості.

3 ПРОВЕДЕННЯ ТЕСТІВ ТА ВВЕДЕННЯ ЗАХИСТУ REST API

3.1 Спроби викрадення сесії

Тестування на можливість викрадення сесії є критично важливим етапом оцінки безпеки веб-додатків, оскільки сесійні токени часто виступають ключем до доступу користувачів і можуть стати привабливою мішенню для зловмисників. Проведення таких тестів допомагає виявити слабкі місця в механізмах аутентифікації та управління сесіями, що дозволяє своєчасно впровадити ефективні заходи захисту і запобігти несанкціонованому доступу до конфіденційних даних. Крім того, імітація реальних атак дає змогу краще зрозуміти, як саме можуть бути викрадені сесійні дані, а також оцінити ефективність існуючих механізмів безпеки веб-додатку.

Перший метод викрадення сесії базується на перехопленні та повторному використанні токена сесії. Для створення умов перехоплення даних між користувачем та веб-додатком буде симульовано ситуацію підслуховування запитів за допомогою Burp Suite, який є потужним проксі-інструментом для перехоплення, аналізу та модифікації HTTP/HTTPS-запитів між браузером і сервером. Задля проведення тестування буде проведено налаштування Burp Suite та проведено спробу зайти до веб-додатку у ролі користувача. Для цього спочатку потрібно запустити Burp Suite і налаштувати його як проксі, щоб він слухав локальний порт. Потім у браузері потрібно вказати цей проксі-сервер або скористатися вбудованим браузером Burp Suite. Після цього увімкнути перехоплення трафіку, щоб інструмент міг зупиняти запити і показувати їх зловмиснику. Тепер при спробі заходу користувача до тестувального веб-додатку маємо отримати інформацію про здійснені запити користувачем та їх результати. Вигляд заходу до веб-додатку на якому проводиться дане тестування зображено на рисунку 3.1.

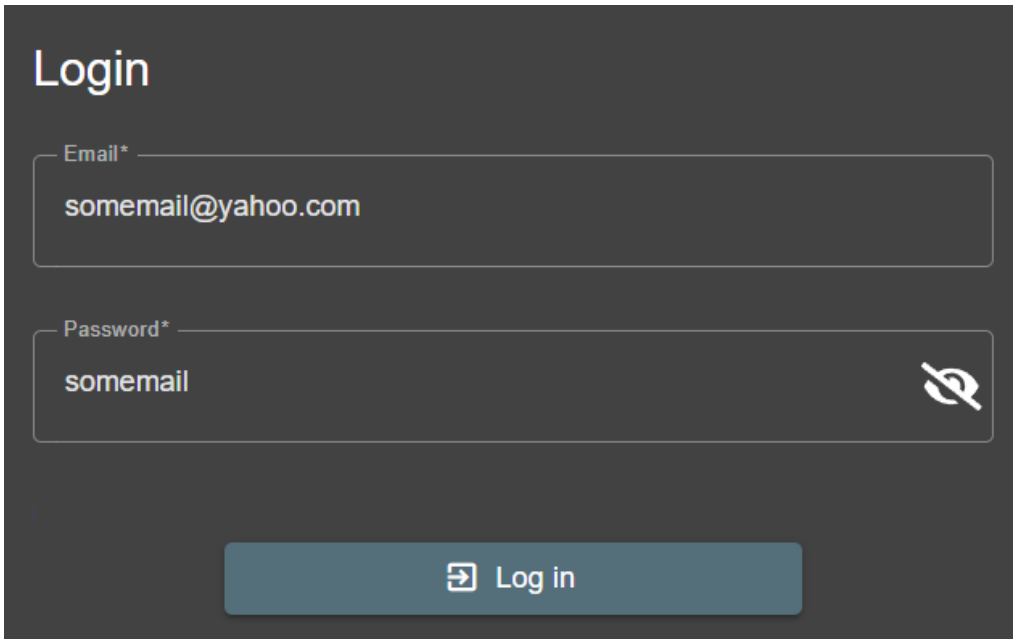


Рисунок 3.1 – Проведення аутентифікації користувачем

Після успішної аутентифікації на сайті, під час аналізу трафіку за допомогою застосунку Burp Suite з'являється можливість детально переглянути повний HTTP-запит, включно з усіма його заголовками та тілом. Це дозволяє виявити всі параметри, які передаються від клієнта до сервера, а також оцінити, які дані надсилаються під час взаємодії з API. Після того, як запит був пропущений через проксі Burp Suite, у відповіді сервера вдалося знайти та ідентифікувати сесійний токен, який був створений спеціально для цього користувача. Цей токен є надзвичайно важливим, оскільки він слугує ключем для підтримки сесії користувача і забезпечує авторизацію подальших запитів без повторної аутентифікації. За допомогою Burp Suite можна не лише переглядати, а й редагувати перехоплені запити, змінюючи токени, параметри або заголовки, що дає змогу моделювати різноманітні атаки, наприклад повторне використання токенів або спроби їх підробки. Це допомагає оцінити рівень захищеності API від таких загроз і виявити потенційні вразливості. Приклад вигляду знайденого токена користувача наведено на рисунку 3.2.

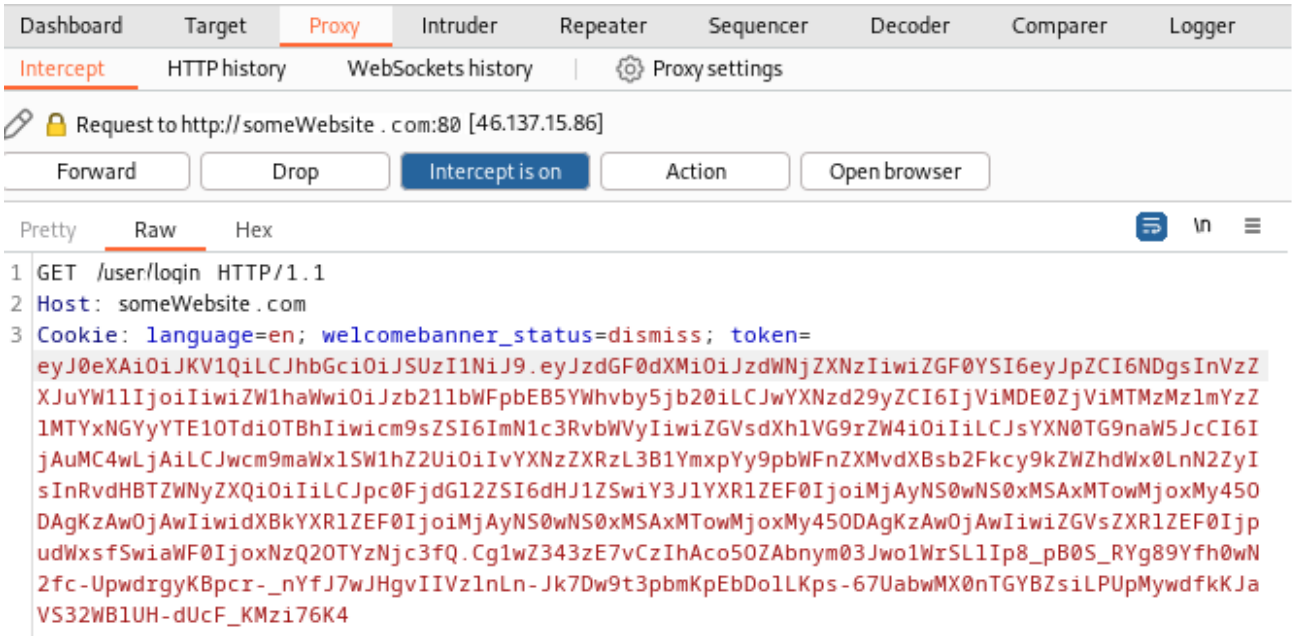


Рисунок 3.2 – Перехоплена критична інформація користувача

Після отримання токена з перехопленого запиту можна використати його для несанкціонованого доступу до веб-додатку, видаючи себе за легітимного користувача. Для цього у середовищі Postman створюється новий запит до захищеного API-ендпоінту, наприклад, для отримання даних профілю користувача, куди в заголовках додається скопійований токен авторизації. Надісланий таким чином запит фактично імітує сесію жертви, що дозволяє отримати доступ до її конфіденційної інформації без повторної аутентифікації. В даному випадку використовуємо скопійований токен. Цей метод демонструє вразливість системи до повторного використання викрадених токенів і підкреслює необхідність впровадження додаткових заходів захисту, таких як короткочасні токени, механізми їх відкликання, а також багатофакторна аутентифікація і моніторинг аномальної активності. На рисунку 3.3 наведено приклад успішного відтворення сесії за допомогою Postman із використанням викраденого токена.

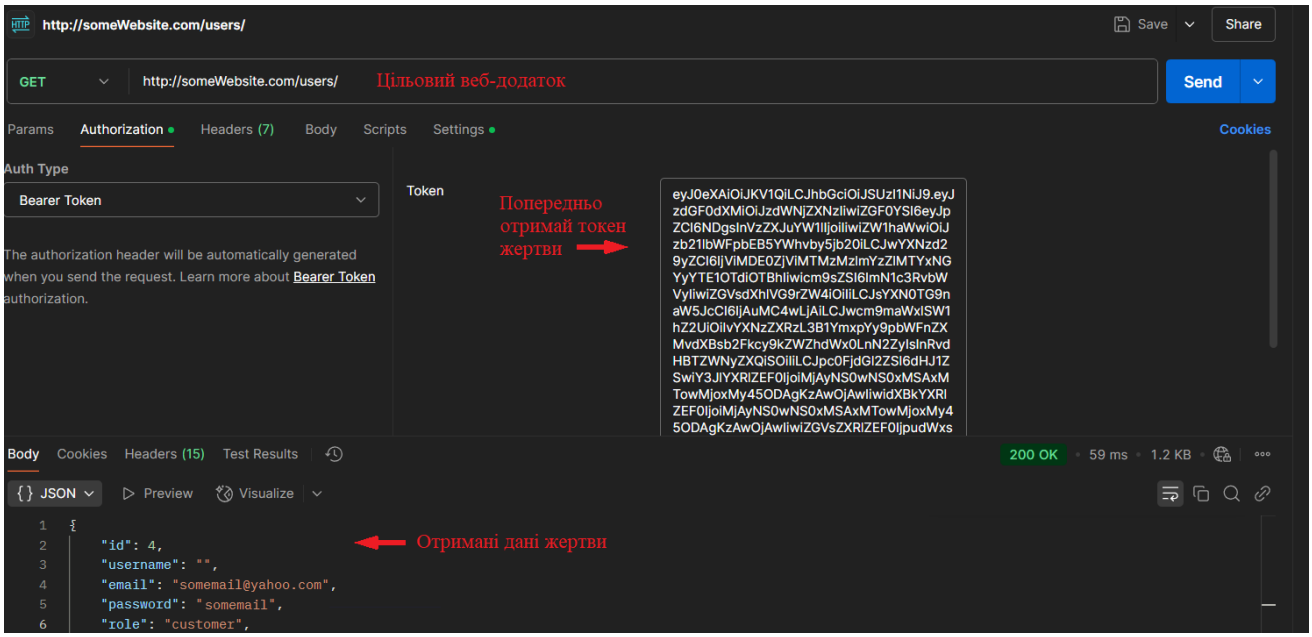


Рисунок 3.3 – Отримана більш детальна інформація про користувача

Цей метод є ефективним, оскільки даний веб-додаток працює через HTTP, токени сесії передаються відкритим текстом і їх легко перехопити. Відсутність прапорців Secure і HttpOnly у cookie дозволяє викрадати токени через браузерні атаки. Сервер, який не перевіряє додаткові параметри, такі як IP-адреса чи User-Agent, дозволяє використовувати викрадений токен з будь-якого пристрою. Burp Suite дає можливість не лише побачити токен, а й гнучко керувати запитами для глибокого тестування. Також важливо тестувати, чи сервер інвалідизує токени після виходу або тривалої неактивності, а також чи прив'язує сесію до IP-адреси або User-Agent.

Отже, перехоплення і повторне використання токена сесії є доволі простою, але дуже потужною атакою на REST API, особливо якщо не застосовуються базові заходи безпеки. Burp Suite є незамінним інструментом для її реалізації, оскільки дозволяє перехоплювати, аналізувати і змінювати трафік, що робить його обов'язковим для тестувальників безпеки і пентестерів.

Ще один не менш розповсюджений спосіб викрадення сесії це атака типу "Man-In-The-Middle" (людина посередині). Даний вид кібер-атаки зосереджений на перехоплені зловмисником даниї, дає змогу змінювати або підробляти комунікацію між сторонами клієнта і сервера, не будучи при цьому поміченим.

Атаки типу "Man-In-The-Middle" особливо небезпечні, оскільки REST API часто використовують токени для аутентифікації, які передаються у заголовках або cookie. Якщо токен перехоплено, зловмисник може отримати повний доступ до облікового запису. Відсутність шифрування дозволяє не лише викрадати відкриту інформацію, а й підробляти запити, наприклад, зміна замовлень, підвищення прав користувачів та інше. Атаки типу "Man-In-The-Middle" можуть бути непомітними для жертви, якщо сервер не впроваджує додаткові механізми захисту. У контексті REST API це означає, що зловмисник може перехопити HTTP-запити та відповіді, отримати конфіденційну інформацію ,як токени сесії, а також змінити або повторити запити. Приклад виконання даного методу перехоплення даних можливо повторити використовуючи доступний програмний застосунок по типу Wireshark, який аналізує всі пакети мережі до якої він є підключений. Приклад знаходження необхідного пакету перехоплених даних за допомогою застосунку Wireshark наведене на рисунку 3.4.

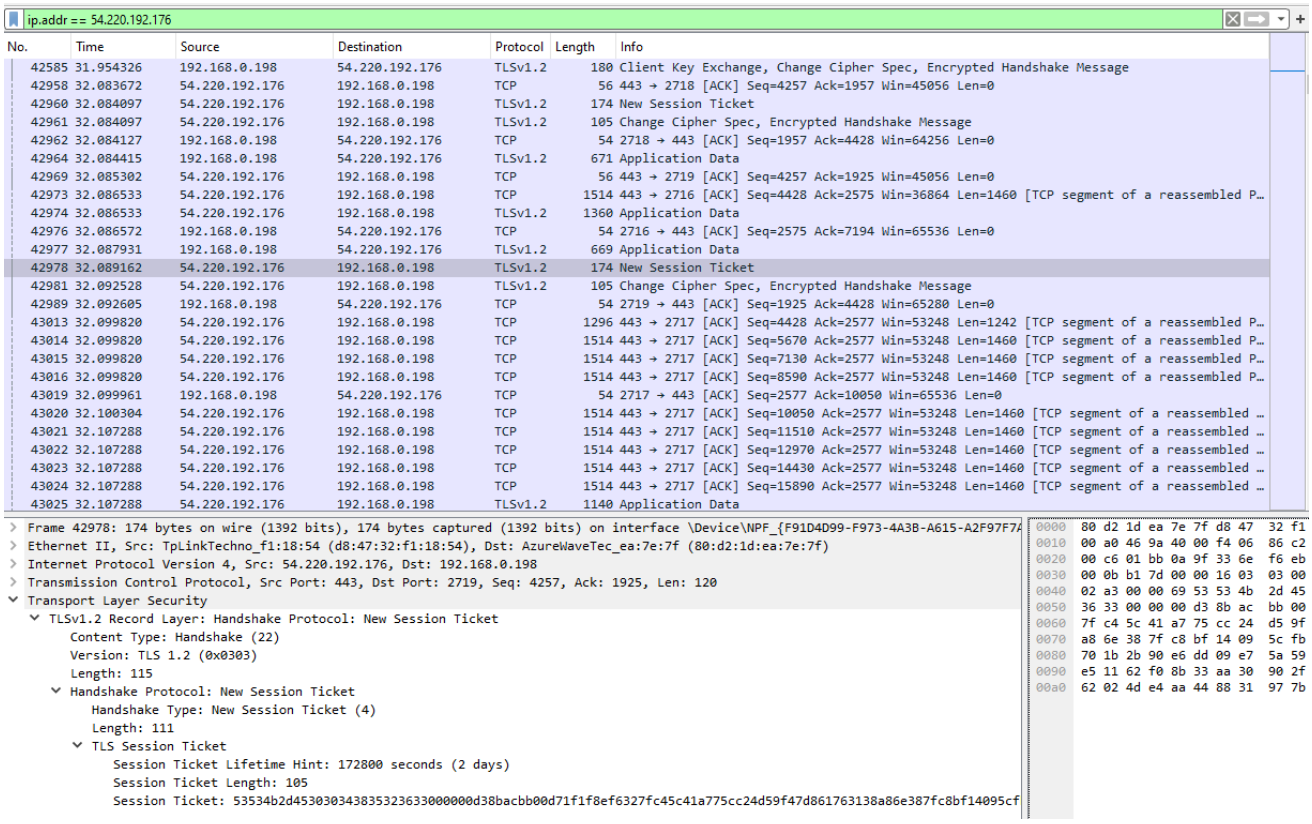


Рисунок 3.4 – Перехоплення пакетів через застосунок Wireshark

Після перехоплення та перегляду пакетів в застосунку Wireshark можливо відфільтрувати усі необхідні нам пакети, які можуть стосуватися, як і потрібних нам веб-додатків так і окремих користувачів, які працюють через таку ж саме мережу, що і зловмисник. В даному прикладі при використанні даного методу було знайдено пакет, який містить у собі токен сесії потрібного нам користувача, тому ми можемо використати його щоб видати себе за нього у потрібному нам веб-додатку. Результат такого використання токенів можливо переглянути на рисунку 3.3, оскільки даний процес виведення необхідних даних є ідентичним із попереднім методом, оскільки використовується однаковий метод авторизації через систему токенів.

Наступний метод викрадення сесійних токенів пов’язаний із атакою типу XSS (Cross-Site Scripting) – однією з найпоширеніших і найнебезпечніших вразливостей у веб-додатках. Ця атака дає змогу зловмиснику впровадити і виконати шкідливий JavaScript-код безпосередньо у браузері жертви. У контексті REST API це особливо критично, оскільки сесійні токени, які використовуються для аутентифікації користувача, часто зберігаються у cookie або localStorage - обох цих місцях доступних для виконання JavaScript. Якщо cookie не мають прапорця HttpOnly, шкідливий скрипт може легко їх прочитати і викрасти, отримуючи таким чином повний контроль над сесією користувача. Навіть у випадках, коли токени зберігаються у localStorage, XSS-атака залишається ефективним інструментом викрадення, оскільки цей механізм зберігання повністю доступний для JavaScript на стороні клієнта. Викрадення токенів через XSS обходить багато інших механізмів захисту, зокрема ті, що працюють на рівні мережі або сервера, адже атака відбувається всередині браузера жертви, використовуючи довіру до легітимного сайту. Це робить XSS однією з найсерйозніших загроз для безпеки REST API.

Для проведення атаки даним методом зловмиснику необхідно знайти у веб-додатку поле вводу, яке некоректно обробляє введені дані і дозволяє вставити шкідливий JavaScript-код. Це може бути форма відгуків, коментарів, пошуку або будь-яке інше поле, де введений текст відображається без належного екранування

та фільтрації. Далі зловмисник вводить у вразливе поле спеціальний скрипт, який виконається у браузері будь-якого користувача, хто перегляне сторінку з цим вмістом, та коли жертва заходить на сторінку з ін’єкцією, браузер виконує шкідливий скрипт. Приклад впровадженого шкідливого скрипта можливо переглянути у рисунку 3.5.

Рисунок 3.5 – Приклад впровадження шкідливого скрипту

Він зчитує токен сесії, який зберігається у cookie або localStorage, і відправляє його на сервер зловмисника, який він зазначив у шкідливому скрипті.

Вже із отриманим токеном сесії жертви зловмисник може використовувати його для аутентифікації у REST API. Приклад механізму впровадження XSS-атаки в веб-додатки наведено на рисунку 3.6.

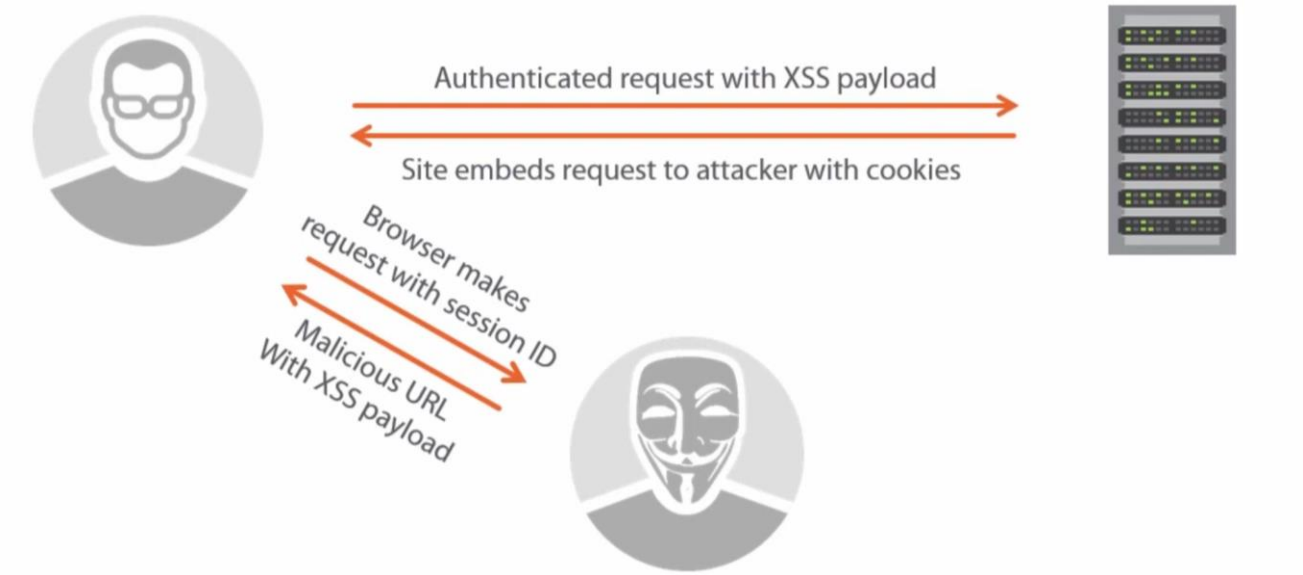


Рисунок 3.6 – Механізм впровадження XSS-атаки в веб-додатки

Викрадення токена через XSS-атаки на REST API є однією з найнебезпечніших загроз, оскільки зловмисник, впроваджуючи шкідливий JavaScript-код у вразливі поля введення або відповіді сервера, може отримати доступ до сесійних токенів користувачів. Ці токени дають змогу повністю контролювати сесію жертви, що відкриває шлях до несанкціонованого доступу до конфіденційних даних і функціоналу веб-додатку. Застосування інструментів на кшталт Postman дозволяє зловмиснику використати викрадений токен для відтворення сесії і отримання захищеної інформації, що підкреслює критичну важливість впровадження надійних заходів захисту від XSS, таких як валідація та кодування введених даних, а також використання безпечних заголовків.

Ще одним серйозним ризиком є атака методом передбачення токенів сесії, яка, хоч і трапляється рідше, може мати катастрофічні наслідки за умови використання слабких або передбачуваних алгоритмів генерації токенів. Зловмисник аналізує структуру токенів, шукаючи закономірності, наприклад інкрементальні ідентифікатори або прості шаблони, що дозволяє автоматизувати

перебір можливих значень токенів за допомогою спеціалізованих скриптів або інструментів. У разі успіху така атака відкриває доступ до чужих сесій та конфіденційної інформації, що робить необхідним застосування криптографічно стійких методів генерації токенів, обмеження часу їх дії та впровадження додаткових механізмів контролю доступу. Таким чином, захист від XSS та передбачення токенів є невід'ємною частиною комплексної стратегії безпеки REST API, що включає регулярний аудит, моніторинг аномалій і застосування сучасних технологій захисту.

Для реалізації такої атаки часто застосовують інструменти типу Postman у поєднанні з функціями автоматизації, або спеціалізовані скрипти, які послідовно змінюють значення токена і відправляють запити до захищених ендпоінтів API. Відповіді сервера аналізуються на предмет статусу доступу: якщо сервер повертає успішний статус із номером 200 , це означає, що токен дійсний і сесія активна. Якщо ж відповідь містить помилку авторизації із кодом 401, то токен недійсний або сесія закінчилася. Приклад відповідного скрипту який застосовувався для підбору токену можливо переглянути у лістингу 3.7 який наведено в додатку Б.

Цей метод є особливо ефективним у випадках, коли сервер не обмежує кількість спроб аутентифікації і не застосовує захист від брутфорс-атак, наприклад, обмеження швидкості запитів або тимчасове блокування IP-адрес після кількох невдалих спроб. Відсутність таких механізмів дозволяє зловмиснику безперешкодно перебирати токени, що значно підвищує ризик компрометації сесій. Для ілюстрації застосування автоматизованого перебору токенів у Postman наведено рисунок 3.8, яка демонструє процес відправлення послідовних запитів із різними значеннями токена та аналіз відповідей сервера. Впровадження таких заходів безпеки, як обмеження кількості спроб, моніторинг аномальної активності та використання складних, криптографічно стійких токенів, є критично важливими для запобігання успішним атакам методом перебору.

Рисунок 3.8 – Приклад скриптованого підбір токену

Регулярний аудит і тестування безпеки API з використанням інструментів, що імітують перебір токенів, допомагає виявити подібні вразливості і усунути їх до того, як їх використають зловмисники. Таким чином, передбачення токенів є серйозною загрозою, але її можна ефективно нейтралізувати за допомогою належних практик генерації токенів і контролю доступу.

3.2 Можливі вдосконалення системи задля перешкоджання повторення проведених атак

Захист REST API від сесійного викрадення та інших кіберзагроз є комплексним і багатогранним процесом, що вимагає впровадження низки взаємодоповнюючих заходів безпеки. Одним із найважливіших кроків є забезпечення шифрування всього трафіку між клієнтом і сервером за допомогою протоколу HTTPS. Приклад Реалізація HTTPS наведено у рисунку 3.9.

Рисунок 3.9 – Налаштування HTTPS з’єднання

Це дозволяє захистити дані від перехоплення і модифікації зловмисниками, які можуть перебувати в тій же мережі або використовувати методи атаки "Man-In-The-Middle". Впровадження HTTPS супроводжується налаштуванням

заголовка HTTP Strict Transport Security (HSTS), який змушує браузери автоматично переходити на захищене з'єднання, навіть якщо користувач вводить адресу без префікса https://. Це значно знижує ризик випадкового або навмисного використання незашифрованого каналу.

Наступним важливим аспектом є правильне налаштування cookie, у яких зберігаються сесійні токени. Приклад захищеної конфігурації файлів cookie наведено у рисунку 3.10.

Рисунок 3.10 – Безпечна конфігурація файлів cookie

Встановлення прапорців Secure, HttpOnly та SameSite є ключовим для запобігання XSS-атакам та CSRF (Cross-Site Request Forgery). Прапорець Secure гарантує, що cookie передаються лише через захищене HTTPS-з'єднання, що унеможливлює їх перехоплення через незашифровані мережі. HttpOnly забороняє доступ до cookie зі сторони JavaScript, що захищає від викрадення токенів через XSS-атаки. SameSite обмежує відправку cookie у крос-доменних запитах, що допомагає уникнути CSRF-атак, коли зловмисний сайт намагається виконати дії від імені користувача. Не менш важливим є те, щоб токени сесії були короткоживучими і криптографічно стійкими. Задання короткого часу життя токена , наприклад, 15-30 хвилин, обмежує вікно можливого використання

викраденого токена. Криптографічна стійкість забезпечується використанням надійних генераторів випадкових чисел і складних алгоритмів шифрування, що унеможливлює передбачення або підробку токенів. Використання JSON Web Tokens з підписом і шифруванням є сучасною практикою, але важливо контролювати їх час дії і механізми відкликання.

Додатковими механізмами захисту можна відокремити ротацію і інвалідизацію токенів. Приклад виконання даного методу наведено у рисунку 3.11.

Рисунок 3.11 – Імплементація ротації токенів

Після кожного входу користувача, зміни прав доступу або інших критичних подій сервер повинен генерувати новий токен і анулювати старий. Це унеможливлює повторне використання викрадених токенів. Також при виході з

системи або тривалій неактивності токен слід інвалідизувати, щоб запобігти доступу за допомогою застарілих сесій. Впровадження механізмів відкликання токенів є важливим елементом безпеки, особливо для довгоживучих токенів. Для захисту від передбачення токенів необхідно використовувати криптографічно стійкі генератори випадкових чисел при створенні токенів сесії. Токени повинні бути довгими, містити випадкові символи і бути унікальними для кожної сесії. Також важливо впроваджувати обмеження на кількість запитів з однієї IP-адреси або користувача за певний проміжок часу, а також застосовувати механізми блокування або додаткової аутентифікації при підозрілих спробах доступу.

Ще одним важливим заходом у забезпеченні безпеки сесій є прив’язка сесії до контексту користувача, що передбачає додаткову перевірку параметрів запиту, таких як User-Agent та IP-адреса. Приклад імплементації перевірки методів наведено у рисунках 3.12. та 3.13.

Рисунок 3.12 – Імплементація перевірки параметру User-Agent

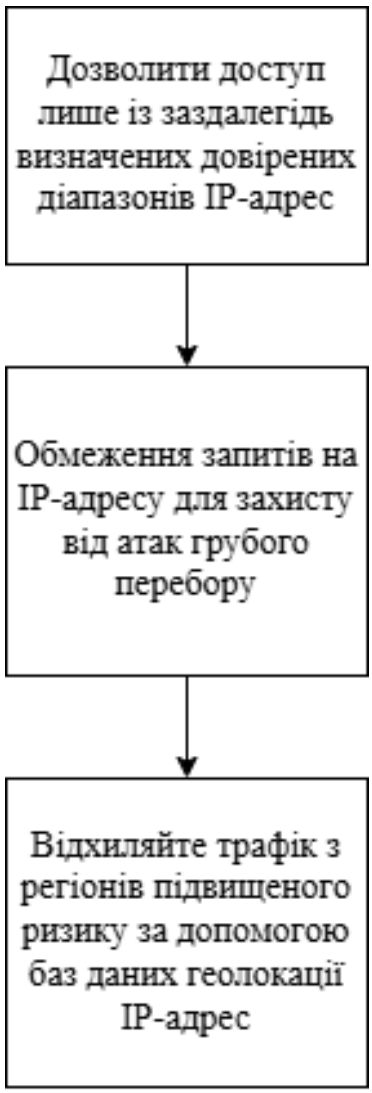


Рисунок 3.13 – Імплементація перевірки параметру IP-адреси

Сервер може відстежувати ці параметри під час кожного запиту і виявляти аномалії, які можуть свідчити про несанкціоноване використання сесійного токена. Наприклад, якщо токен починає використовуватися з іншої IP-адреси або браузера, ніж ті, що були зафіксовані під час аутентифікації, це може свідчити про викрадення сесії або атаку типу "людина посередині". У таких випадках сервер може автоматично заблокувати сесію, ініціювати її інвалідизацію або вимагати від користувача повторної аутентифікації для підтвердження своєї особи.

Водночас, реалізація цього механізму повинна бути гнучкою і враховувати особливості поведінки користувачів. Наприклад, IP-адреса користувача може легітимно змінюватися при переході між різними мережами, як домашньою,

мобільною або корпоративною, що є звичайною практикою для сучасних мобільних пристроїв. Аналогічно, користувачі можуть використовувати різні браузери або пристрої, що також впливає на параметри User-Agent. Тому надмірно суворі обмеження можуть призвести до помилкових спрацьовувань, погіршуючи користувацький досвід і викликаючи зайві запити на повторну аутентифікацію.z

Щоб уникнути таких проблем, системи безпеки часто впроваджують адаптивні алгоритми, які враховують історію поведінки користувача, геолокацію, частоту змін IP-адрес і типи пристроїв. Наприклад, невеликі зміни IP-адреси або використання схожих User-Agent можуть не викликати блокування, тоді як різкі і несподівані зміни навпаки, призводять до додаткових перевірок. Також застосовують багатофакторну аутентифікацію як додатковий рівень захисту у випадках виявлення підозрілої активності.

Впровадження багатофакторної аутентифікації значно підвищує безпеку, додаючи другий рівень перевірки користувача. Необхідно запропонувати користувачам реєструвати другий фактор під час створення облікового запису або першого входу після увімкнення багатофакторної аутентифікації. Вимагати багатофакторну аутентифікацю не тільки при вході в систему, але й для операцій з високим рівнем ризику, таких як зміна паролів, доступ до конфіденційних даних або здійснення фінансових операцій. Така поетапна автентифікація знижує ризик несанкціонованих дій, навіть якщо паролі скомпрометовані. Навіть якщо зловмисник викрав токен або пароль, без другого окремого підтвердження, наприклад, одноразового коду з мобільного додатку, через поштову скриню чи SMS, доступ до акаунта буде неможливим. Багатофакторна аутентифікація є ефективним засобом протидії багатьом типам атак, включно з фішингом і сесійним викраденням. Приклад імплементації методу багатофакторної аутентифікації наведено у рисунку 3.14.

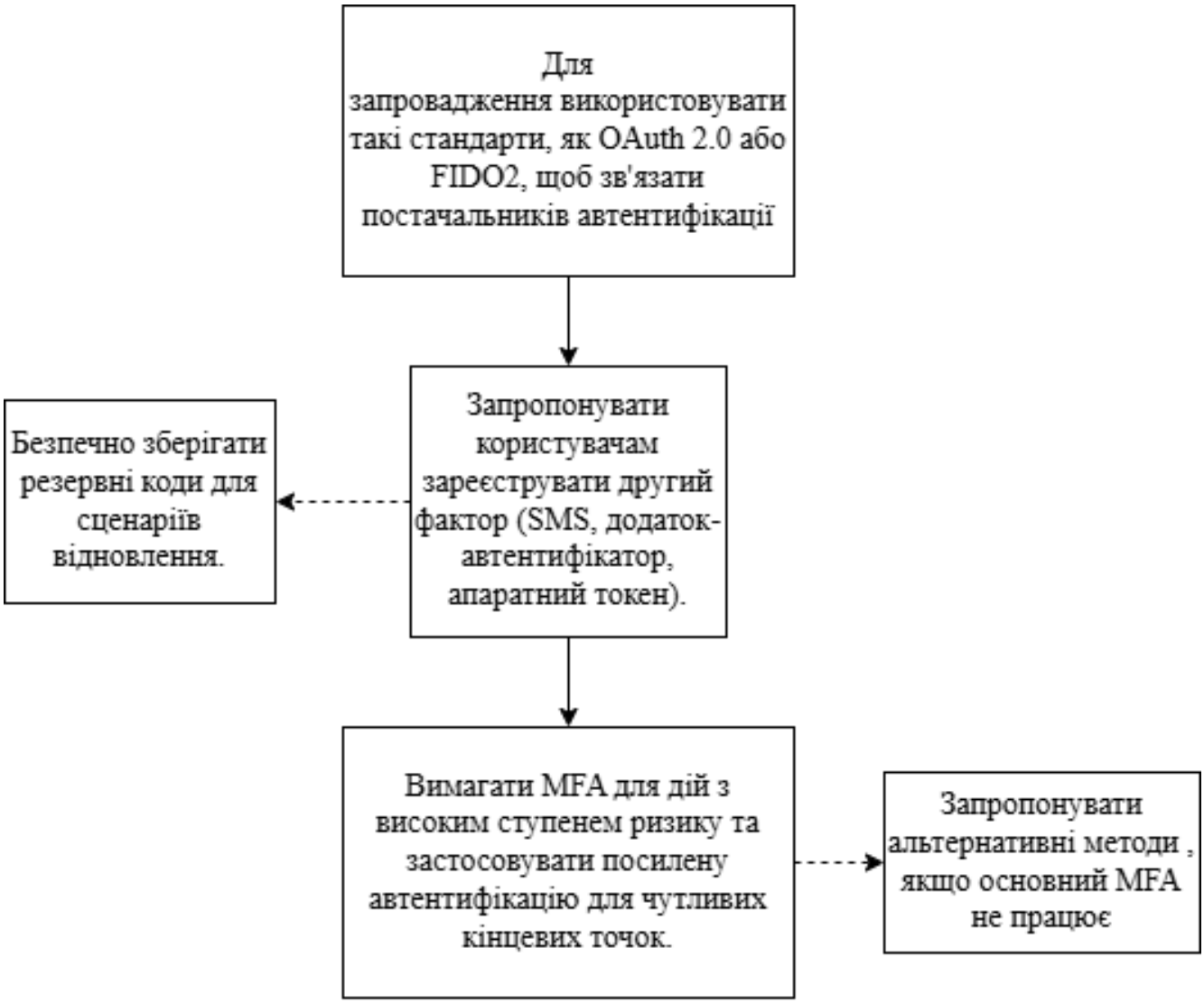


Рисунок 3.14 – Імплементація перевірки параметру IP-адреси

Критично важливим для захисту сесій є усунення XSS-вразливостей. Для цього потрібно ретельно санітизувати всі вхідні дані, застосовувати CSP (Content Security Policy), яка обмежує виконання сторонніх і небезпечних скриптів, та регулярно оновлювати всі компоненти додатку і залежності. Під час отримання даних від користувача застосовувати суворі правила валідації, які дозволяють вводити лише очікувані та безпечні формати даних. Наприклад, якщо форма очікує введення номера телефону, відхиляйте будь-які дані, що містять HTML-теги або спеціальні символи. Це зменшує ймовірність потрапляння шкідливих скриптів у вашу систему. Використовувати перевірку за білим списком замість того, щоб вносити підозрілі символи до чорного списку, оскільки зловмисники

можуть легко обійти чорні списки. Зберігання токенів у HttpOnly cookie замість localStorage значно знижує ризик викрадення через XSS, оскільки JavaScript не зможе отримати доступ до токена. Також можливою мірою захисту виступить розгортання брандмауера веб-додатку, який може виявляти та блокувати шкідливе корисне навантаження ще до того, як воно потрапить до вашого додатку. Це додає додатковий рівень захисту, фільтруючи поширені шаблони XSS-атак і підозрілий трафік. Не менш важливим використовувати безпечні редактори розширеного тексту або розмітку у разі надання дозволу користувачам надсилати відформатований вміст, використовуйте безпечні редактори або Markdown, які дезінфікують вхідні дані та обмежують небезпечні HTML-теги та атрибути, запобігаючи впровадженню скриптів та дозволяючи безпечне форматування. Також слід уникати передачі токенів у URL, оскільки URL можуть зберігатися в логах сервера, браузера або передаватися через заголовок Referer на сторонні сайти. Це створює додаткові ризики витоку токенів. Тому токени слід передавати виключно в заголовках або cookie.

Застосування HTTPS є фундаментальним кроком, оскільки без шифрування будь-які інші заходи можуть бути марними через можливість перехоплення трафіку. Впровадження SSL/TLS сертифікатів сьогодні є стандартом і не створює значних технічних бар'єрів, тому цей метод має найвищий пріоритет і є базовою вимогою для безпеки.

Використання атрибутів cookie, таких як Secure, HttpOnly і SameSite, суттєво знижує ризики атак на клієнтські сесії. Вони ефективно блокують доступ до токенів зі сторони JavaScript і обмежують їх передачу, що ускладнює експлуатацію XSS та CSRF. Впровадження цих прапорців є відносно простим і не впливає на продуктивність, тому цей метод є дуже доцільним.

Короткоживучі і криптографічно стійкі токени зменшують час, протягом якого викрадений токен може бути використаний. Це підвищує безпеку, але вимагає налаштування механізмів оновлення токенів, що може ускладнити реалізацію і вплинути на зручність користувачів, особливо якщо оновлення відбувається надто часто.

Ротація і інвалідизація токенів рахується, як додатковий рівень захисту, який дозволяє швидко нейтралізувати скомпрометовані сесії. Однак це вимагає більш складної логіки на сервері і може призводити до проблем із синхронізацією, якщо клієнт не встигає отримати новий токен вчасно.

Прив'язка сесії до IP-адреси і User-Agent допомагає виявляти аномальну активність і запобігати використанню токенів з незнайомих пристроїв. Проте цей метод може створювати незручності для користувачів із динамічними IP або тих, хто часто змінює пристрої, тому його слід застосовувати з обережністю і гнучкими налаштуваннями.

Багатофакторна аутентифікація значно підвищує рівень безпеки, адже навіть при викраденні токена зловмиснику буде складно повністю отримати доступ. Водночас багатофакторна аутентифікація може ускладнити користувацький досвід і збільшити час входу, що варто враховувати при її впровадженні.

Усунення XSS-вразливостей є критичним заходом, який не тільки захищає сесійні токени, а й запобігає багатьом іншим атакам. Однак повне усунення XSS вимагає ретельного аналізу коду, санітизації даних і постійного моніторингу, що може бути трудомістким і ресурсозатратним.

Уникнення передачі токенів у URL – простий, але дуже ефективний захід, який запобігає випадковому витоку токенів через логи або заголовки Referer. Цей метод не має негативного впливу на роботу системи і легко реалізується.

Впровадження вищезазначених заходів у комплексі формує надійний захист REST API від сесійного викрадення і інших атак. Важливо також проводити регулярний аудит безпеки, моніторинг аномальної активності і автоматизоване тестування, що дозволяє вчасно виявляти і усувати потенційні вразливості. Це забезпечує безпеку користувачів, збереження конфіденційних даних і стабільну роботу системи.

Загалом, найефективнішими і найбільш критичними є впровадження HTTPS, правильне налаштування cookie і усунення XSS-вразливостей. Методи, пов'язані з управлінням токенами і перевіркою контексту, підвищують безпеку,

але потребують більш складної реалізації і можуть впливати на зручність користувачів. Багатофакторна аутентифікація є потужним додатковим захистом, але вимагає балансування між безпекою і комфортом. Найкращі результати досягаються при поєднанні кількох методів у комплексній стратегії безпеки.

Таким чином, лише комплексний підхід, який поєднує шифрування, правильне управління сесіями, перевірку контексту, багатофакторну аутентифікацію та захист від вразливостей клієнтської сторони, дозволяє ефективно протистояти сесійному викраденню і забезпечити високий рівень безпеки REST API.

ВИСНОВКИ

REST API на сьогодні є одним із ключових компонентів сучасних веб-додатків і сервісів. Він забезпечує стандартизований, масштабований і гнучкий спосіб взаємодії між клієнтськими застосунками, мобільними пристроями, веб-серверами та іншими системами. REST API дозволяє розробникам створювати модульні, розподілені архітектури, де різні компоненти можуть незалежно розвиватися, масштабуватися і взаємодіяти через прості HTTP-запити.

Проведений всебічний аналіз проблеми сесійного викрадення із використанням створеного сайту для проведення необхідних тестувань, який був створений на основі OWASP Juice Shop, а також детальне розглядання методів тестування з використанням таких інструментів, як Postman і Burp Suite, дозволяє глибше усвідомити масштаб і складність загроз, які стоять перед сучасними REST API. Ця проблема є однією з найпоширеніших та найнебезпечніших у сфері веб-безпеки, оскільки сесійні токени слугують ключем до доступу користувачів і часто є єдиним механізмом аутентифікації у багатьох додатках.

Вразливості, що виникають через відсутність шифрування трафіку, по типу використання HTTP замість HTTPS, неправильне налаштування cookie, а також через XSS-атаки, створюють численні можливості для зловмисників викрадати токени сесій. Це дозволяє їм отримувати повний контроль над обліковими записами, виконувати несанкціоновані дії, отримувати доступ до конфіденційної інформації та навіть змінювати дані. Особливо небезпечними є атаки типу "Man-In-The-Middle", коли зловмисник може перехоплювати і змінювати трафік у реальному часі, а також фіксація сесії, яка дозволяє атакуючому примусити жертву використовувати контрольований токен.

Практичні методи, такі як перехоплення і повторне використання токенів, фіксація сесії, викрадення токенів через XSS, MITM-атаки та передбачення токенів, можна ефективно моделювати і тестувати за допомогою Burp Suite і Postman. Ці інструменти дають змогу не лише виявити уразливості, а й глибоко

проаналізувати механізми аутентифікації і управління сесіями, що є критично важливим для розробників і фахівців із безпеки.

Для забезпечення високого рівня захисту REST API необхідно впроваджувати комплекс заходів. Найважливішим є використання HTTPS, яке забезпечує шифрування трафіку і захищає від перехоплення даних. Правильне налаштування cookie з прапорцями Secure, HttpOnly і SameSite допомагає захистити токени від викрадення через клієнтські скрипти і крос-домени атаки. Використання короткоживучих і криптографічно стійких токенів зменшує ризик тривалого використання викрадених токенів.

Ротація і інвалідизація токенів дозволяють швидко нейтралізувати скомпрометовані сесії, а прив’язка сесії до IP-адреси і User-Agent допомагає виявляти аномалії і запобігати несанкціонованому доступу. Впровадження багатофакторної аутентифікації значно підвищує безпеку, додаючи додатковий рівень перевірки користувача. Усунення XSS-вразливостей і уникнення передачі токенів у URL є додатковими, але не менш важливими заходами, що допомагають мінімізувати ризики.

Оцінка ефективності цих методів показує, що найбільший вплив на безпеку має комплексне поєднання технологічних рішень і налаштувань, які не лише ускладнюють життя зловмисникам, а й не створюють зайвих незручностей для користувачів. Водночас деякі заходи, наприклад, прив’язка сесії до IP або часта ротація токенів, можуть впливати на зручність використання, тому їх потрібно впроваджувати з урахуванням специфіки аудиторії і сценаріїв використання.

Регулярний аудит безпеки, моніторинг аномальної активності, автоматизоване тестування і навчання команди розробників і тестувальників є невід’ємною частиною підтримки високого рівня захисту. Використання OWASP Juice Shop як навчального майданчика у поєднанні з Postman і Burp Suite дозволяє не лише виявляти уразливості, а й глибоко розуміти механізми атак і захисту, що є критично важливим для підвищення кваліфікації фахівців і покращення безпеки реальних систем.

Отже, лише системний, багаторівневий підхід, що включає сучасні технології, налаштування безпеки, автоматизацію тестування і постійний контроль, може гарантувати ефективний захист REST API від сесійного викрадення і забезпечити надійний захист даних і довіру користувачів у цифровому середовищі.

Глибоке розуміння принципів роботи REST API, потенційних загроз і методів їхнього запобігання є необхідним для розробників, тестувальників і фахівців із безпеки. Лише комплексний підхід до проектування, реалізації і захисту REST API дозволяє створити надійні, безпечні і масштабовані системи, які відповідають вимогам сучасного цифрового світу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. REST API Security: Best Practices Guide URL: https://www.stackhawk.com/blog/rest-api-security-best-practices/ (дата звернення: 14.02.2025)

2. How to Secure a REST API URL: https://tyk.io/learning-center/how-to-secure-rest-api/ (дата звернення: 14.02.2025)

3. Cost of a Data Breach Report 2024 URL: https://table.media/wp-content/uploads/2024/07/30132828/Cost-of-a-Data-Breach-Report-2024.pdf (дата звернення: 14.02.2025)

4. GDPR Fines: Understanding Percentages and Penalties URL: https://gdprlocal.com/gdpr-fines-understanding-percentages-and-penalties/ (дата звернення: 14.02.2025)

5. Security Strategies for Microservices-based Application Systems URL: https://csrc.nist.gov/pubs/sp/800/204/final (дата звернення: 03.03.2025)

6. How session identifiers help protect APIs URL: https://blog.barracuda.com/2025/04/10/session-identifiers-protect-apis (дата звернення: 03.03.2025)

7. What is REST API? URL: https://www.ibm.com/think/topics/rest-apis?mhsrc=ibmsearch_a&mhq=What%20is%20REST%20API%26quest%3B (дата звернення: 03.03.2025)

8. Building a modern API security strategy — API protection URL: https://www.contrastsecurity.com/security-influencers/building-a-modern-api-security-strategy-api-protection (дата звернення: 18.03.2025)

9. REST API — Basic components of Authorization and Authentication URL: https://medium.com/@KeyurRamoliya/rest-api-basic-components-of-authorization-and-authentication-d6145d188359 (дата звернення: 18.03.2025)

10. What Is Zero-Trust API Security? URL: https://prophaze.com/learn/api/what-is-zero-trust-api-security/ (дата звернення: 18.03.2025)

11. Step-by-Step Guide to Implementing JWT in Your REST API for Enhanced Security URL: https://moldstud.com/articles/p-step-by-step-guide-to-implementing-jwt-in-your-rest-api-for-enhanced-security (дата звернення: 18.03.2025)

12. Building RESTful APIs That Scale URL: https://www.leverture.com/post/building-restful-apis-that-scale-best-practices-for-2025 (дата звернення: 26.03.2025)

13. REST API Security Best Practices URL: https://appsentinels.ai/blog/rest-api-security-best-practices/ (дата звернення: 26.03.2025)

14. Security Strategies for Microservices-based Application Systems URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf (дата звернення: 26.03.2025)

15. Best practices for RESTful web API design URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 26.03.2025)

16. The Complete Guide to Secure API Authentication URL: https://auth0.com/learn/api-security (дата звернення: 28.03.2025)

17. What Are Refresh Tokens and How to Use Them URL: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/ (дата звернення: 28.03.2025)

18. The Importance of Securing RESTful Services URL: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-application-security-appsec/what-is-api-security/rest-api-security-foundations-and-best-practices/ (дата звернення: 28.03.2025)

19. Refresh Tokens URL: https://auth0.com/docs/secure/tokens/refresh-tokens (дата звернення: 28.03.2025)

20. REST API Security: 4 Design Principles and 10 Essential Practices URL: https://www.cycognito.com/learn/api-security/rest-api-security.php (дата звернення: 28.03.2025)

21. What is multi-factor authentication (MFA) URL: https://www.ibm.com/think/topics/multi-factor-authentication (дата звернення: 07.04.2025)

22. What is Multi-Factor Authentication URL: https://www.cyberark.com/what-is/mfa/ (дата звернення: 07.04.2025)

23. How to secure a REST API URL: https://tyk.io/learning-center/how-to-secure-rest-api/ (дата звернення: 07.04.2025)

24. MFA for REST APIs URL: https://softwareengineering.stackexchange.com/questions/439702/oauth-2-0-mfa-for-rest-apis (дата звернення: 07.04.2025)

25. User Authentication Using JWT Tokens With IP Bound Sessions URL: https://medium.com/@palsagnik2001/authentication-and-renewal-of-jwt-tokens-with-ip-bound-sessions-51613156f32b (дата звернення: 17.04.2025)

26. REST API Monitoring URL: https://www.manageengine.com/products/applications_manager/rest-api-monitoring.html (дата звернення: 17.04.2025)

27. Activity Monitor REST API URL: https://helpcenter.netwrix.com/bundle/ActivityMonitor_6.0/page/Content/SAM/Appendices/RestAPI.htm (дата звернення: 17.04.2025)

28. Why is REST API monitoring important? URL: https://www.site24x7.com/learn/api-performance-guide.html (дата звернення: 17.04.2025)

29. How to Detect Suspicious API Traffic URL: https://www.akamai.com/blog/security/how-to-detect-suspicious-api-traffic (дата звернення: 23.04.2025)

30. What Is Token-Based Authentication? URL: https://www.okta.com/identity-101/what-is-token-based-authentication/ (дата звернення: 23.04.2025)

31. Securing a REST API by using HTTPS URL: https://www.ibm.com/docs/en/integration-bus/10.0?topic=apis-securing-rest-api-by-using-https (дата звернення: 27.04.2025)

32. What is User Activity Monitoring? URL: https://www.activtrak.com/user-activity-monitoring/ (дата звернення: 27.04.2025)

33. Best Practice For XSS Attacks in Rest Api URL: https://stackoverflow.com/questions/64373470/best-practice-for-xss-attacks-in-rest-api (дата звернення: 27.04.2025)

34. Cross-site scripting URL: https://portswigger.net/web-security/cross-site-scripting (дата звернення: 02.05.2025)

35. Test Your REST APIs Using Insomnia REST Client URL: https://www.codemag.com/Article/2107051/Test-Your-REST-APIs-Using-Insomnia-REST-Client (дата звернення: 02.05.2025)

36. Postman Echo Documentation URL: https://www.postman.com/postman/published-postman-templates/documentation/ae2ja6x/postman-echo?ctx=documentation (дата звернення: 02.05.2025)

37. Checkmarx as a global software security company URL: https://www.westconcomstor.com/id/en/vendors/checkmarx.html (дата звернення: 11.05.2025)

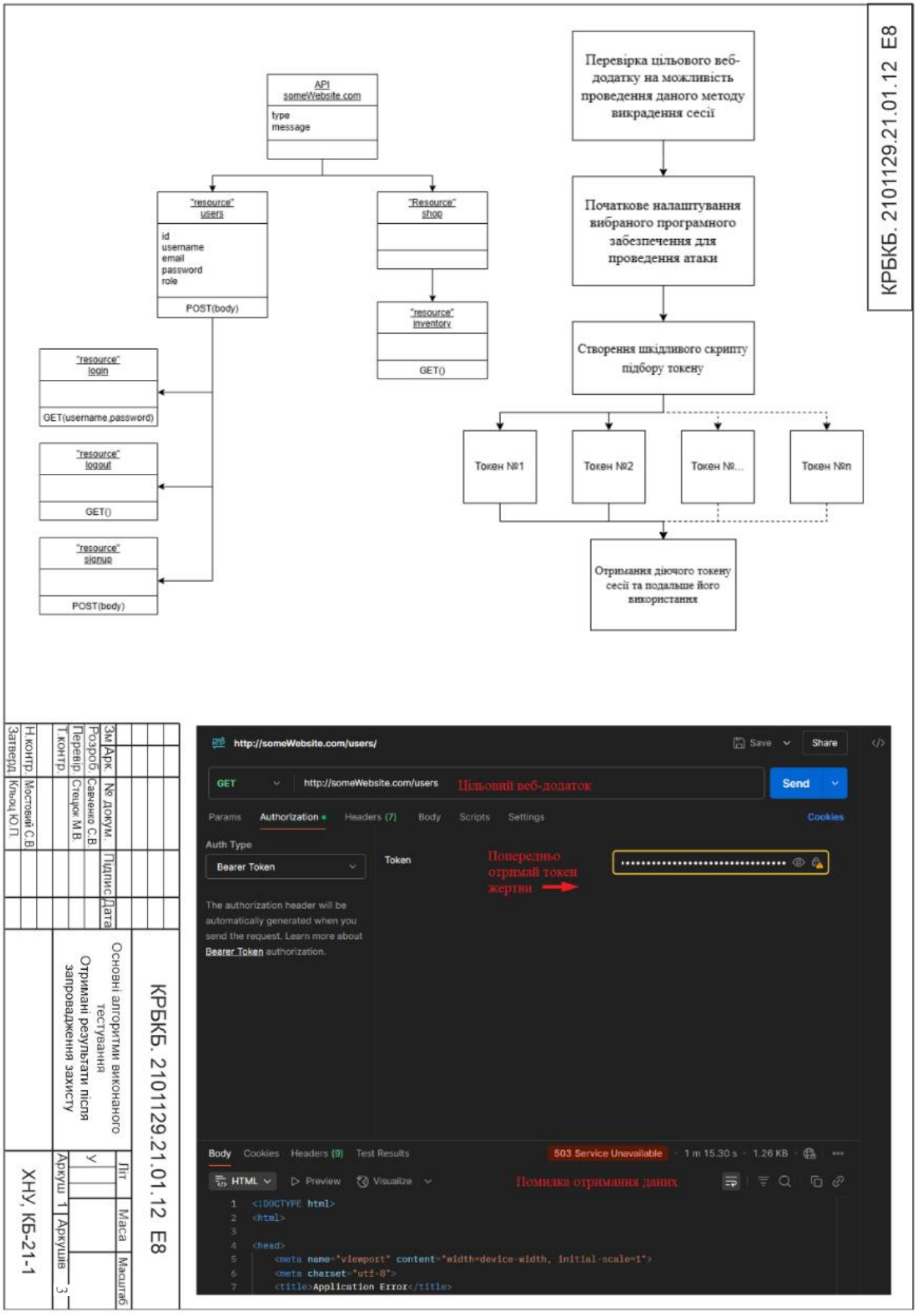
38. OWASP Juice Shop URL: https://owasp.org/www-project-juice-shop/ (дата звернення: 11.05.2025)

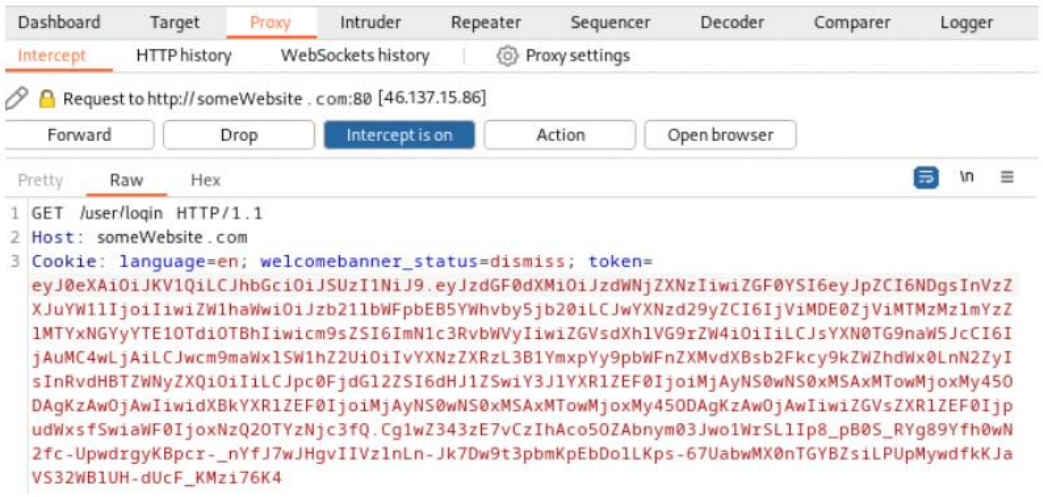
39. Scale your AppSec, unburden your security team URL: https://portswigger.net/burp/dast (дата звернення: 11.05.2025)

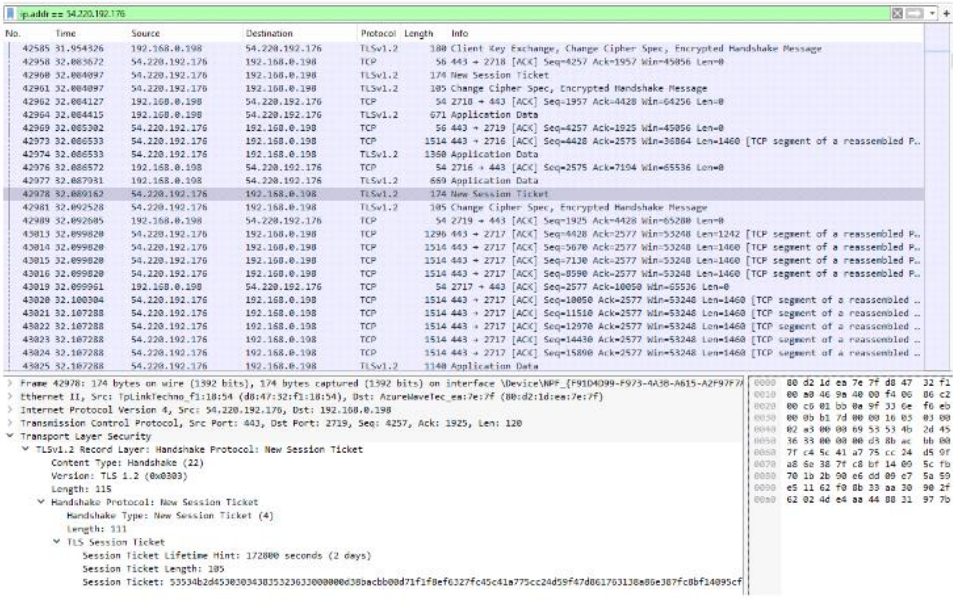
40. How to Use Wireshark URL: https://www.varonis.com/blog/how-to-use-wireshark (дата звернення: 11.05.2025)

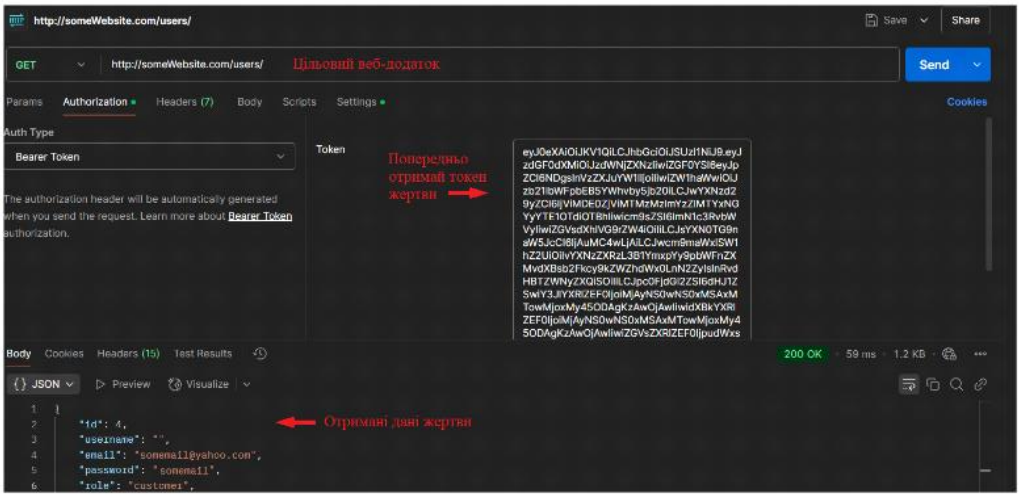
ДОДАТОК А

Копія графічної частини



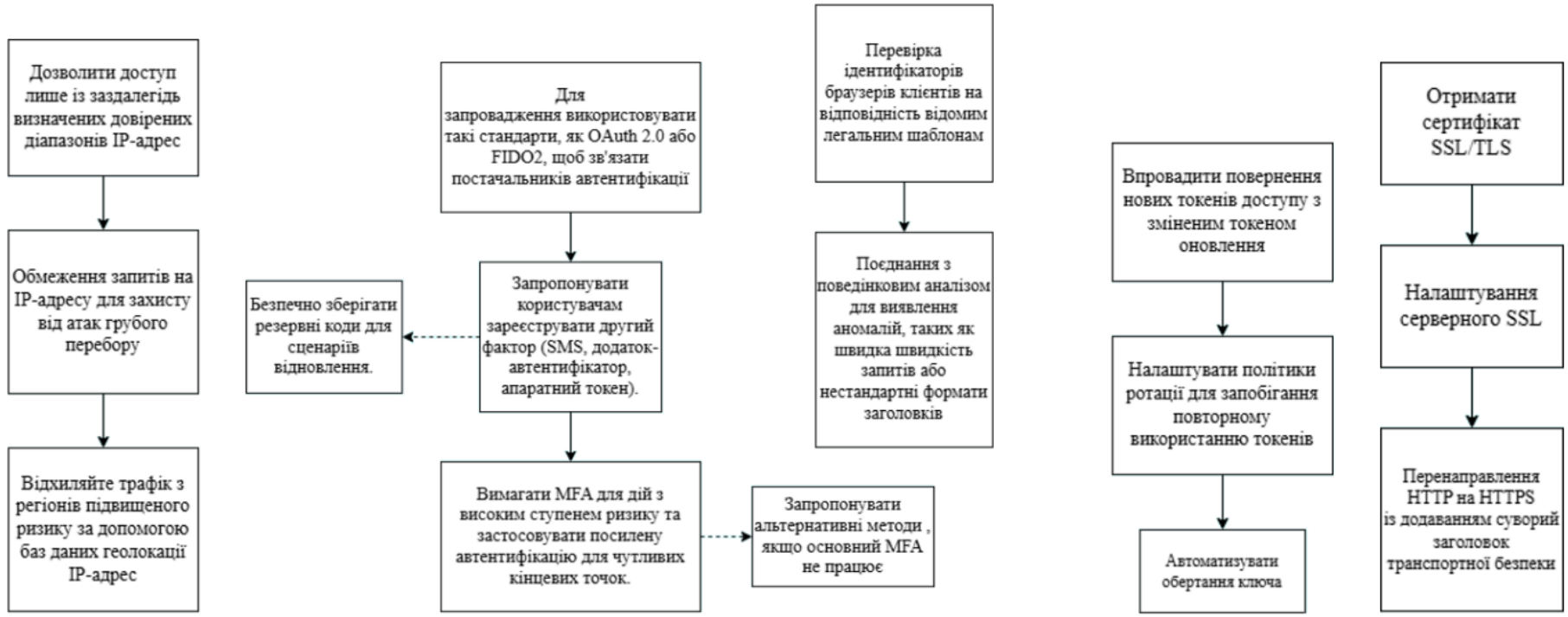


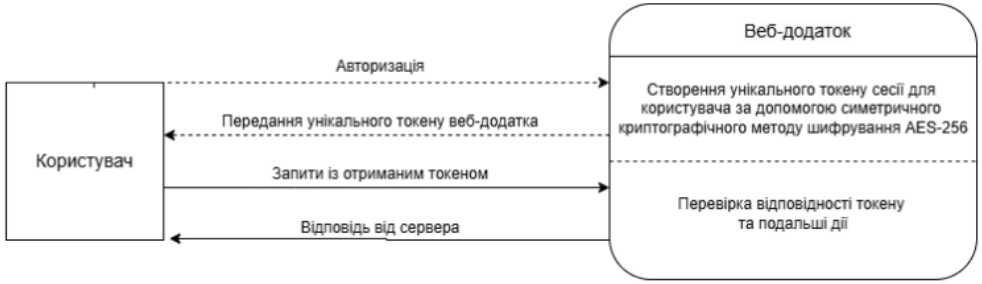




Зм.	Арк.	№ докум.	Підпис	Дата	КРБКБ.2101129.21.01.12 Е8		
Розроб.	Савченко С.В.				Перехоплення сесійного токену жертви за допомоги BurpSuite, Wireshark та подальше його використання	Літ.	
Перевір.	Стецюк М.В.						
Т.контр.						Маса	Масштаб
Н.контр.	Мостовий С.В.						
Затверд.	Кльоц Ю.П.				ХНУ, КБ-21-1	Аркуш 2	Аркушів 3

КРБКБ.2101129.21.01.12 Е8





					КРБКБ.2101129.21.01.12 Е8	
Зм	Арк.	№ докум.	Підпис	Дата	Впровадженні методики покращення захисту REST API додатків	Літ: У; Маса; Масштаб
Розроб.		Савченко С.В.				Аркуш 3 / Аркушів 3
Перевір.		Стецюк М.В.				
Т.контр.						
Н.контр.		Мостовий С.В.			ХНУ, КБ-21-1	
Затверд.		Кльоц Ю.П.				

ДОДАТОК Б

Лістинг 3.7 – Скрипт підбору токену

```
// Змінні - токен і час
const tokenVar = 'access_token';
const tokenTimeVar = 'token_time';
const tokenLifetimeVar = 'token_lifetime';
const token = pm.environment.get(tokenVar);
const tokenTime = pm.environment.get(tokenTimeVar);
const tokenLifetime = pm.environment.get(tokenLifetimeVar) || 300;

// отримання нового токена
function getNewToken() {
    const authRequest = {
        url: 'https://jsonplaceholder.typicode.com/users',
        method: 'POST',
        header: {
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: {
            mode: 'urlencoded',
            urlencoded: [
                { key: 'client_id', value: pm.environment.get('client_id') },
                { key: 'client_secret', value: pm.environment.get('client_secret') },
                { key: 'grant_type', value: 'client_credentials' }
            ]
        }
    };

    pm.sendRequest(authRequest, function (err, res) {
        if (err || res.code !== 200) {
            console.error('Помилка:', err || res.status);
            return;
        }
        const json = res.json();
        pm.environment.set(tokenVar, json.access_token);
        pm.environment.set(tokenTimeVar, Date.now());
        pm.environment.set(tokenLifetimeVar, json.expires_in);
        console.log('Новий токен:', json.access_token);
    });
}

// Перевірка оновлення токена
if (!token || !tokenTime || (Date.now() - tokenTime) / 1000 > tokenLifetime) {
    console.log('Оновлення...');
    getNewToken();
} else {
    console.log('Токен дійсний.');
}
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Савченка Сергія Володимировича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КБ-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

27.05.2025
дата

підпис

Anti-Plagiarism v-15.258 (global version)

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

ID: 242312 Title: Комплексна систему захисту REST API додатків від атак на основі підбору сесій Added in a DB: 2025-05-28 Authors: Савченко Сергій Володимирович Heads: Стецюк М.В. Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	95535	608	526 (1%)	6 (1%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Савченко Сергій Володимирович

Співавтор:

Назва: Комплексна систему захисту REST API додатків від атак на основі підбору сесій

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:2.3%

Коефіцієнт подібності 2:0%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-28 19:44:14.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1. Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

29.05.2025р. СМБ

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ КІБЕРБЕЗПЕКИ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Комплексна систему захисту REST API додатків від атак на основі підбору сесій

Автор: Савченко Сергій Володимирович

Спеціальність: 125 – Кібербезпека

Освітня програма: Кібербезпека

Науковий керівник: Микола СТЕЦЮК д-р філософії, ст. викладач

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 97.7%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Виявлені модифікації стосуються математичних формул і не є порушенням академічної доброчесності.

Керівник роботи _____ Микола СТЕЦЮК

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Савченко Сергій Володимирович

Тема Комплексна систему захисту REST API додатків від атак на основі підбору сесії

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 69.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі, відповідно до поставленого завдання, проведене дослідження предметної області, ознайомлення із архітектурою REST API та проведені випробування викрадення сесії та покращення захисту системи для запобігання подальшої компрометації даних.

2. Висновок про відповідність кваліфікаційної роботи завданню У кваліфікаційній роботі повністю виконано поставлене завдання як у теоретичній, так і в практичній частині

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У розділі 1 розглянуті існуючі проблеми, створення, розвиток та впровадження REST API. Також було розглянуто потенційні методи захисту REST API. У розділі 2 був проведений аналітичний вибір необхідних інструментів для проведення поставлених тестів та обговорено мету створення та використання веб-додатку. У розділі 3 розглянуто основні методи викрадення токенів сесії та заволодіння бажаної інформації завдяки цим токенам, також був проведений аналіз можливих захистів для запобігання втрати персональних даних та методи їх імплементації

4. Позитивні сторони роботи Робота добре структурована, чітко висвітлені всі етапи дослідження. Використані сучасні інструменти та підходи, що свідчить про якість проведеного дослідження.

5. Негативні сторони роботи В роботі не повністю розкрито вплив застосування комплексного поєднання захисту на загальну продуктивність системи та створення потенційних вразливостей через впровадження даних методів захисту

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні сторони кваліфікаційної роботи, а також негативні сторони, які не зменшують практичну цінність отриманих результатів і загальну якість роботи, рекомендованою оцінкою є «відмінно»

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Констянтинович, завідувач кафедри ТМІТ

«02» 06 2025. (підпис)