ЛАБОРАТОРНА РОБОТА №6
АНАЛІЗ ТИПОВИХ ВРАЗЛИВОСТЕЙ ВЕБ-ЗАСТОСУНКІВ ТА МЕХАНІЗМІВ ЇХ ЕКСПЛУАТАЦІЇ (ЧАСТИНА 1)

Мета роботи:

1. Дослідження механізмів функціонування веб-застосунків у моделі клієнт-сервер та проаналізувати структуру HTTP/HTTPS-запитів і відповідей.

2. Ознайомлення з принципами роботи intercepting proxy та набуття практичних навичок перехоплення, аналізу й модифікації HTTP-запитів за допомогою Burp Suite.

3. Аналіз типових вразливостей веб-застосунків та дослідження механізмів їх експлуатації в контрольованому середовищі.

4. Дослідження принципів роботи механізмів автентифікації та авторизації на основі JSON Web Token та аналіз можливих помилок реалізації.

5. Отримання практичних навичок виявлення та аналізу вразливостей типу Path Traversal та File Upload Vulnerabilities.

Інструменти та ПЗ: ВМ Kali Linux, Burp Suite.

Теоретичні відомості

У сучасних інформаційний системах веб-застосунок є одним із найбільш поширених форматів програмного забезпечення. Він функціонує у моделі кілєнт-сервер та взаємодіє з користувачем через веб-браузер із використанням протоколу HTTP/HTTPS.

Базові компоненти веб-інтерфейсу включають:

1. HTML – структурна розмітка сторінки.
2. CSS – оформлення та візуальний стиль.
3. JavaScript – динамічна поведінка та клієнтська логіка.

У більшості сучасних систем застосовуються додаткові бібліотеки та фреймворки (наприклад, Angular або React), що забезпечують складніший інтерфейс, односторінкові застосунки (SPA) та розширену взаємодію з сервером.

Найпоширеніша архітектурна модель складається з трьох ключових компонентів:

1. Веб-сервер.
2. Серверна логіка (Server-Side Scripting).
3. Система зберігання даних.
4. Веб-сервіси та API.

Веб-сервер відповідає за обробку HTTP-запитів, передачу статичних ресурсів, маршрутизацію запитів до серверної логіки.

Серверна логіка реалізує прикладну логіку системи:

1. Аутентифікація та авторизація користувачів.
2. Перевірка вхідних даних.
3. Обробка бізнес-логіки.
4. Взаємодія з базою даних.
5. Формування відповіді клієнту.

Часто використовуються фреймворки (Django, Flask, Symfony, Ruby on Rails тощо), які структурують код відповідно до архітектурних патернів, зокрема MVC (Model-View-Controller).

API (Application Programming Interface) забезпечує стандартизований механізм взаємодії між застосунками. За допомогою API веб-система може:

1. Зберігати або отримувати дані.
2. Надсилати повідомлення.
3. Інтегруватися з платіжними сервісами.
4. Працювати з хмарними сервісами.
5. Взаємодіяти з мобільними застосунками.

OWASP Top 10

Для систематизації ризиків у сфері веб-безпеки використовується класифікація, розроблена OWASP (Open Worldwide Application Security Project).

OWASP є міжнародною відкритою спільнотою, що займається дослідженням вразливостей та розробкою рекомендацій щодо безпечної

розробки застосунків. Одним із ключових документів є OWASP Top 10 – перелік найбільш критичних ризиків безпеки веб-застосунків та API.

OWASP Top 10 відображає найбільш поширені та небезпечні категорії вразливостей, серед яких (станом на 2025 рік):

1. Broken Access Control.
2. Security Misconfiguration.
3. Software Supply Chain Failures.
4. Cryptographic Failures.
5. Injection.
6. Insecure Design.
7. Authentication Failures.
8. Software or Data Integrity Failures.
9. Security Logging and Alerting Failures.
10. Mishhandling of Exceptional Conditions.

Burp Suite

Burp Suite – це інтегрований програмний комплекс для тестування безпеки веб-застосунків. Інструмент має графічний інтерфейс і підтримує повний цикл аналізу безпеки – від дослідження поверхні атаки до експлуатації виявлених вразливостей.

Burp Suite використовується для:

1. Перехоплення HTTP/HTTPS-запитів.
2. Аналізу структури веб-зстосунку.
3. Модифікації параметрів запитів.
4. Тестування механізмів автентифікації та авторизації.
5. Перевірки коректності серверної валідації.

Інструмент працює як intercepting proxy, тобто виступає посередником між браузером та веб-сервером, що дозволяє аналізувати і змінювати трафік у режимі реального часу.

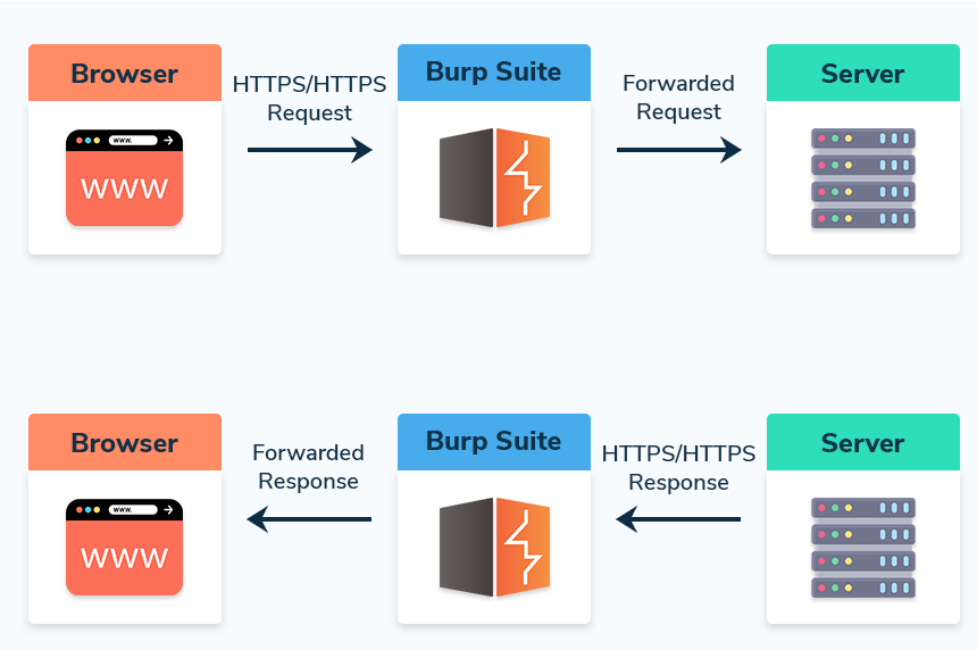

Рисунок 1 – Схема роботи Proxy-посередника

Для спрощення роботи в межах лабораторної роботи використовується вбудований браузер Burp Suite. У такому випадку додаткове налаштування проксі у зовнішньому браузері не потрібне, оскільки весь трафік автоматично проходить через внутрішній proxy-механізм інструмента.

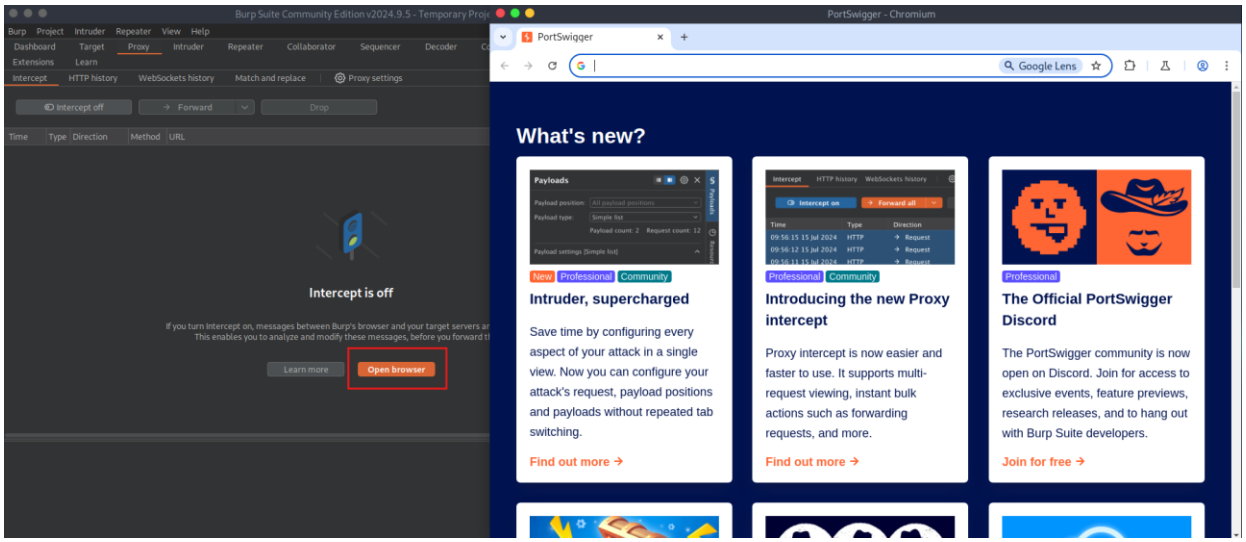

Рисунок 2 – Запуск вбудованого веб-браузера Burp Suite

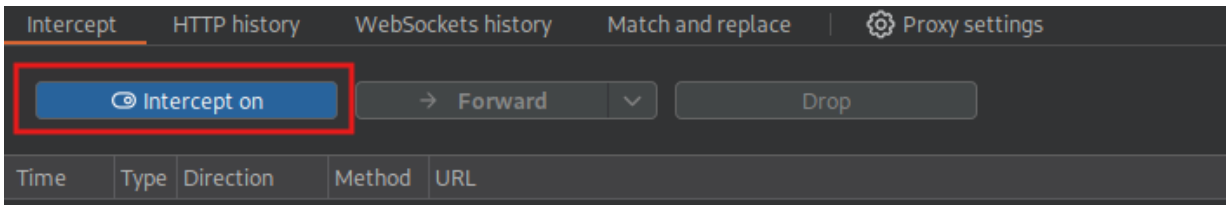

Рисунок 3 – Активація функції перехоплення HTTP-запитів

Після запуску вбудованого браузера та активації режиму Intercept у вкладці Proxy кожен HTTP-запит зупиняється до його надсилання на сервер. Це дозволяє переглянути повний вміст запиту, проаналізувати його структуру та, за потреби, внести зміни перед передачею на сервер.

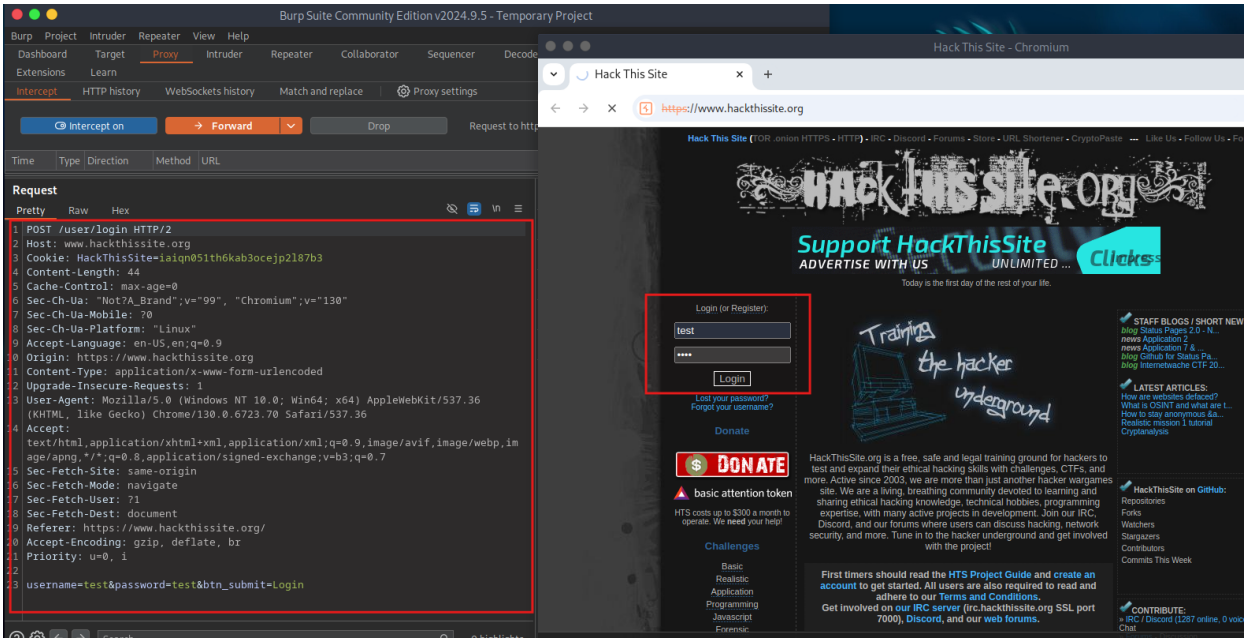

Рисунок 4 – Приклад перехопленого HTTP-запиту під час автентифікації на веб-сайті

Під час аналізу запиту можна переглянути його структуру, яка складається з HTTP-методу (GET, POST тощо), адреси ресурсу, заголовків (Headers) та тіла повідомлення (Body). Особлива увага приділяється таким заголовкам, як Cookie, Authorization, Content-Type, оскільки саме вони часто містять маркери сесії або токени доступу.

Після перехоплення запиту можна керувати його подальшою обробкою. Кнопка **“Forward”** використовується для передачі перехопленого запиту на сервер без змін або після його редагування. Якщо режим Intercept увімкнений, кожен наступний HTTP-запит також буде зупинятися до моменту ручного підтвердження. Це дозволяє контролювати трафік, що надсилається з клієнтської сторони.

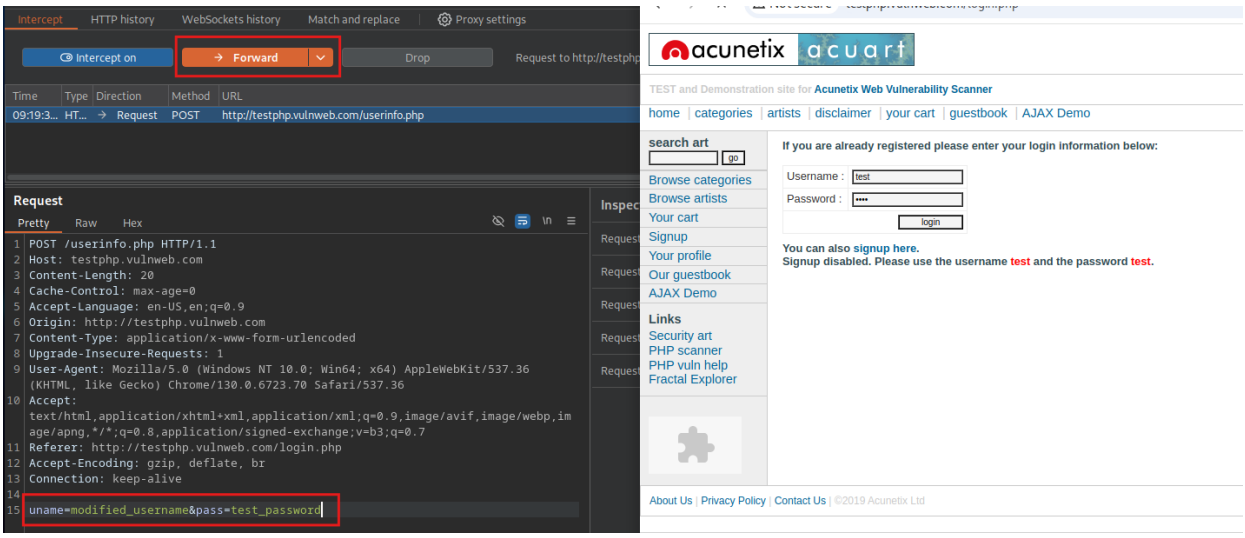

Рисунок 5 – Надсилання модифікованого HTTP-запиту

Після відправлення запиту та отримання відповіді вся взаємодія автоматично зберігається у вкладці HTTP History. Цей розділ містить журнал усіх HTTP-запитів і відповідей, які проходили через проксі. HTTP History дозволяє переглядати повну хронологію роботи з веб-застосунком, аналізувати параметри запитів, вивчати структуру відповідей сервера та повертатися до будь-якого етапу взаємодії.

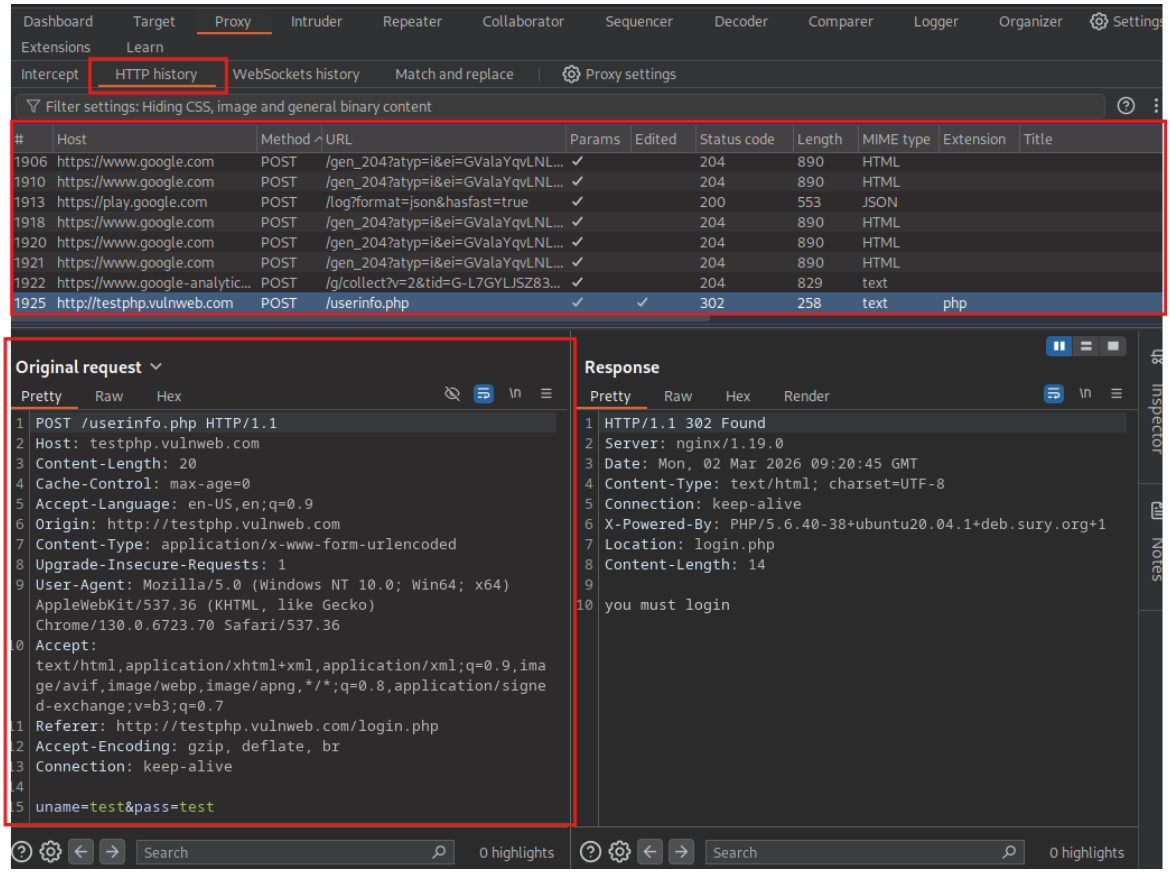

Рисунок 6 – Вкладка History

Для цілеспрямованого тестування конкретного запиту використовується модуль **“Repeater”**. Він дозволяє вручну змінювати будь-яку частину HTTP-запиту та повторно надсилати його на сервер необмежену кількість разів. На відміну від режиму Intercept, Repeater застосовується для цілеспрямованої модифікації запиту та аналізу реакції сервера на внесені зміни.

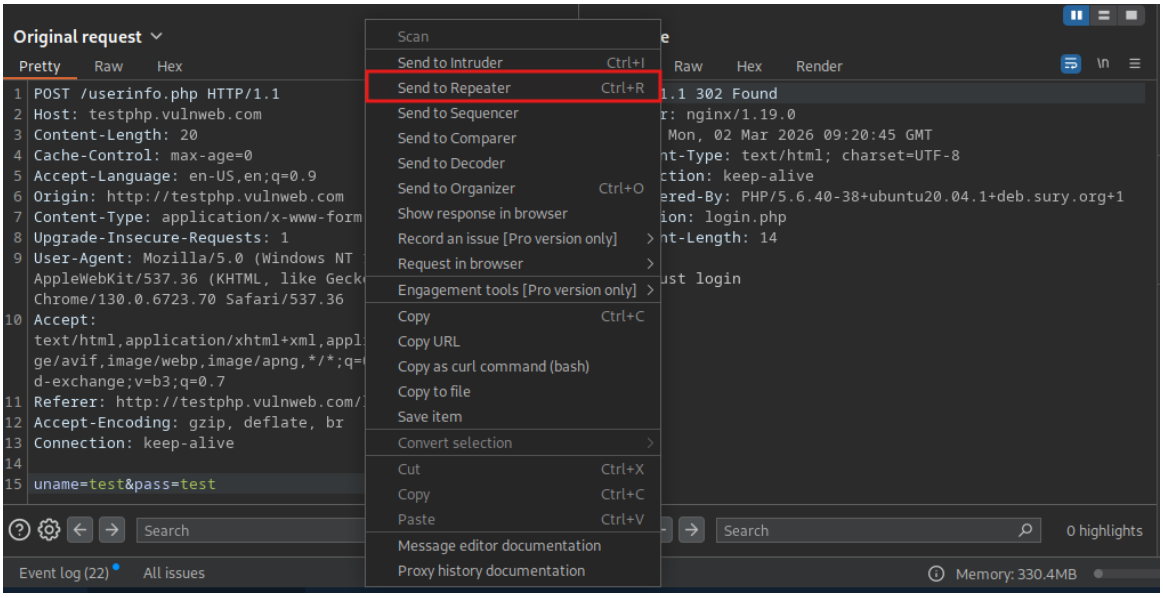

Рисунок 7 – Надсилання HTTP-запиту на вкладку Repeater (ПКМ – Send to Repeater)

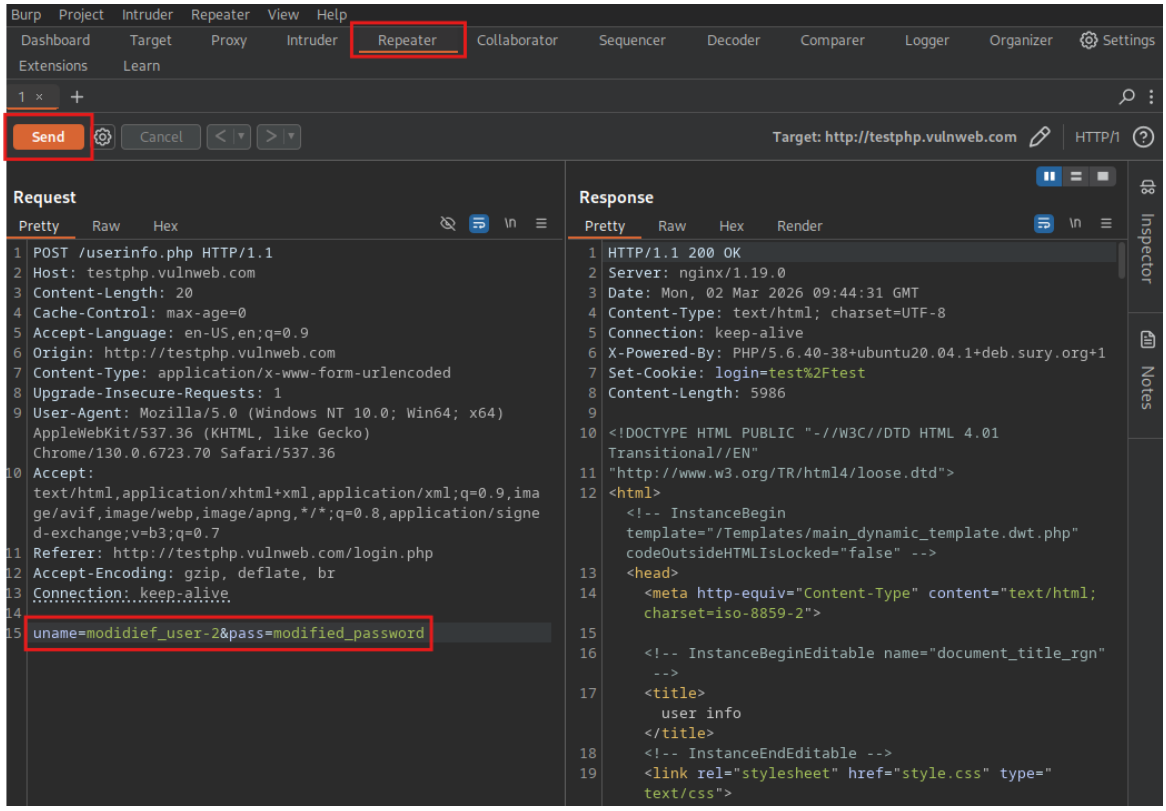

Рисунок 8 – Надсилання модифікованого HTTP-запиту з вкладки Repeater

JSON Web Tokens (JWT) та Broken Access Control

JSON Web Tokens (JWT, RFC 7519) – це відкритий стандарт для безпечного представлення “заявок” (claims) між двома сторонами. Вони часто використовуються як механізм підтримки сесій користувача, альтернативно до традиційних session cookies.

JWT складається з трьох частин:

1. Header – JSON-об’єкт, який описує алгоритм підпису та тип токена.
2. Payload – JSON-об’єкт, що містить дані про користувача або інші дані.
3. Signature – криптографічний підпис, який підписує цілісність Header і Payload.

Всі частини кодуються за допомогою Base64URL і розділяються крапками.

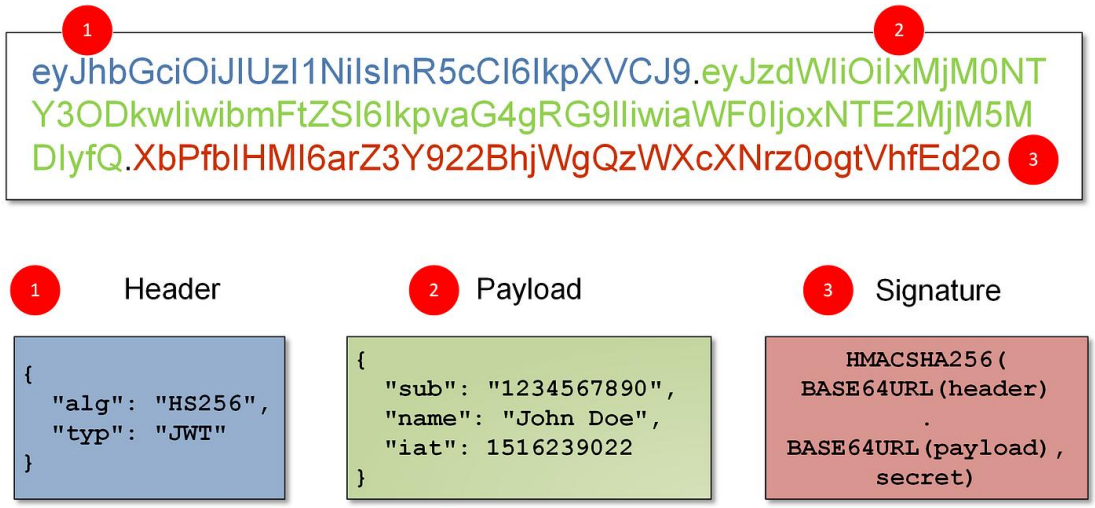

Рисунок 9 – Структура JWT-токена

Для забезпечення цілісності та автентичності токена зазвичай застосовуються алгоритми підпису, наприклад RS256 (HMAC-SHA256 + RSA з приватним ключем сервера) або інші симетричні та асиметричні алгоритми. Сервер перевіряє підпис, щоб упевнитися, що Payload не був змінений клієнтом.

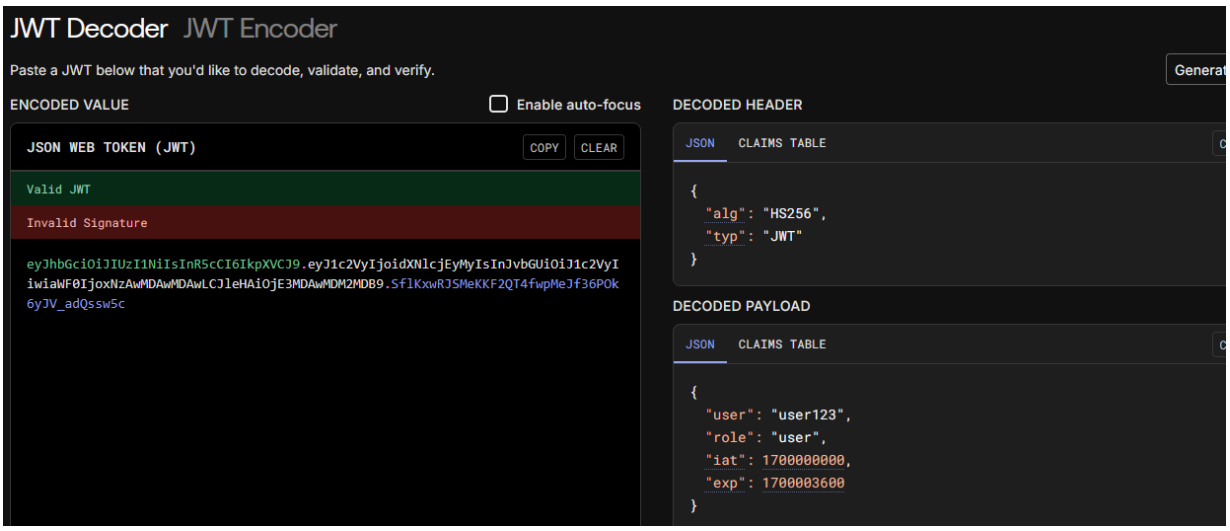

Рисунок 10 – Приклад декодованого JWT-токена (веб-сайт jwt.io)

У старих реалізаціях JWT існували критичні вразливості, які відносяться до категорії **A01:2025 – Broken Access Control** у OWASP Top 10. Однією з таких проблем була атака, що передбачала зміна алгоритму підпису до ***none***, що дозволяло повністю прибрати підпис і змінювати Payload довільно.

Ще один спосіб атаки полягав у зміні алгоритму підпису з RS256 на HS256, що дозволяло використовувати відкритий ключ RSA як симетричний ключ HMAC. Для цього потрібен був доступ до відкритого ключа сервера, який інколи можна було знайти у вихідному коді веб-застосунку.

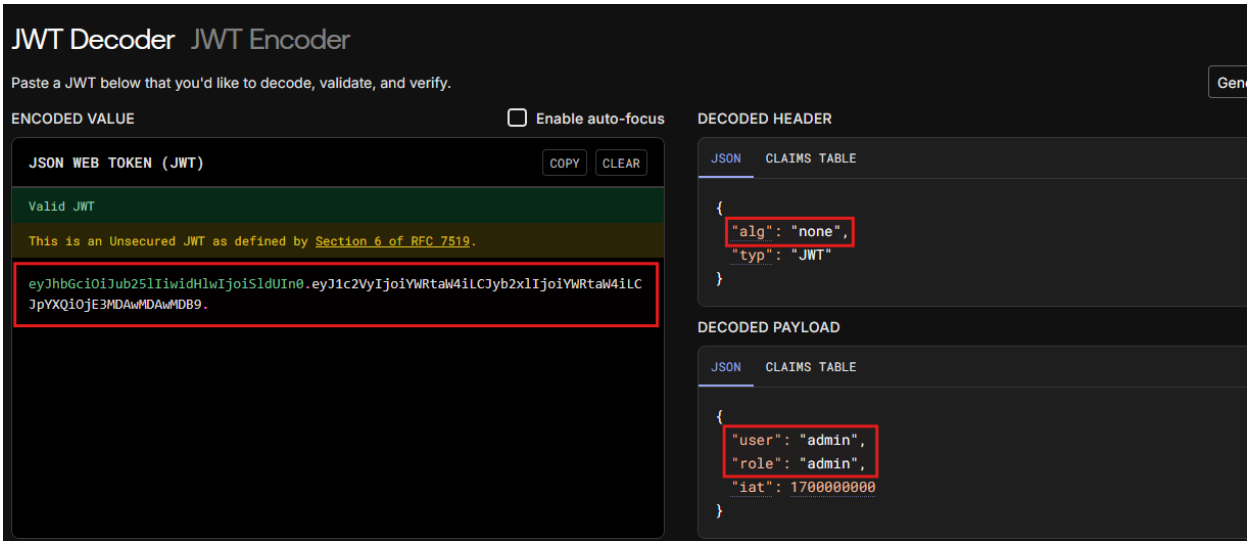

Рисунок 11 – Приклад реалізації атаки на JWT-токен із підміною алгоритму підпису на ***“none”***

Path Traversal

Path Traversal (Directory Traversal) - це тип вразливості, що належить до категорії **A01:2025 – Broken Access Control** за класифікацією OWASP. Вона виникає у випадках, коли веб-застосунок некоректно обробляє введені користувачем шляхи до файлів і не обмежує доступ до ресурсів файлової системи.

Суть атаки полягає у формуванні спеціально сконструйованого запиту, який дозволяє вийти за межі дозволеного каталогу та отримати доступ до файлів або директорій, які не повинні бути доступними користувачу. Зазвичай це відбувається через маніпуляцію параметрами URL або формою введення.

Наприклад, якщо застосунок завантажує зображення за параметром ***filename***, запит може виглядати так:

https://insecure-website.com/loadImage?filename=../../../etc/passwd

У цьому випадку використовується послідовність **../**, яка означає перехід до батьківської директорії. Якщо сервер не виконує нормалізацію шляху та не перевіряє допустимість доступу, можливо отримати вміст системного файлу /etc/passwd (для UNIX-подібних систем).

Аналогічна атака для середовища Windows може виглядати так:

https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini

В обох випадках експлуатується довіра сервера до вхідних даних без належної перевірки.

Рисунок 12 – Приклад реалізації атаки Path Traversal

У реальних системах пряме використання **../** часто блокується фільтрами. Однак обмеження можуть бути реалізовані некоректно, що дозволяє обійти їх за допомогою кодування.

Одним із методів є URL-кодування спеціальних символів:

1. %2e%2e%2f → ../
2. %2e%2e/ → ../
3. %2e%2e%5c → ..\
4. ..%5c → ..\

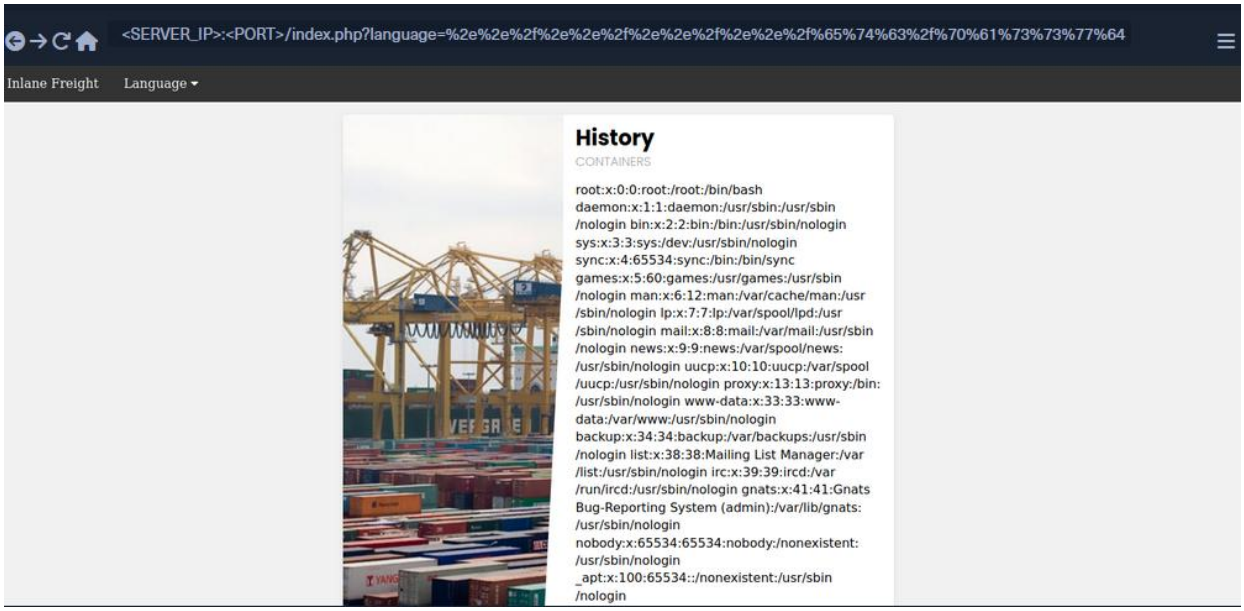

Рисунок 13 – Приклад реалізації атаки Path Traversal з використанням URL-кодування

File Upload Vulnerabilities

Вразливості, пов’язані із завантаженням файлів, належать одразу до кількох категорій OWASP Top 10. Найчастіше вони пов’язані з посилками валідації вхідних даних, некоректною перевіркою типу файлу або порушенням контролю доступу.

Проблема виникає у випадках, коли сервер дозволяє завантаження файлів без належної перевірки їх типу, вмісту або розташування збереження. Якщо механізм валідації відсутній або може бути обійдений, зловмисник отримує можливість завантажити шкідливий файл на сервер.

Основні ризики включають:

1. Unchecked Uploads – сервер приймає файли будь-якого типу без перевірки або перевірка реалізована лише на стороні клієнта.

2. Malicious File Types – завантаження серверних скриптів (наприклад, PHP або ASP), які можуть бути виконані веб-сервером.

3. Content Tampering – підміна або модифікація вмісту веб-застосунку чи даних користувачів.

В окремих випадках навіть сам факт збереження файлу може становити загрозу, якщо сервер автоматично обробляє або індексує завантажені дані. В інших сценаріях для експлуатації необхідно виконати додатковий HTTP-запит до завантаженого файлу, щоб ініціювати його виконання сервером.

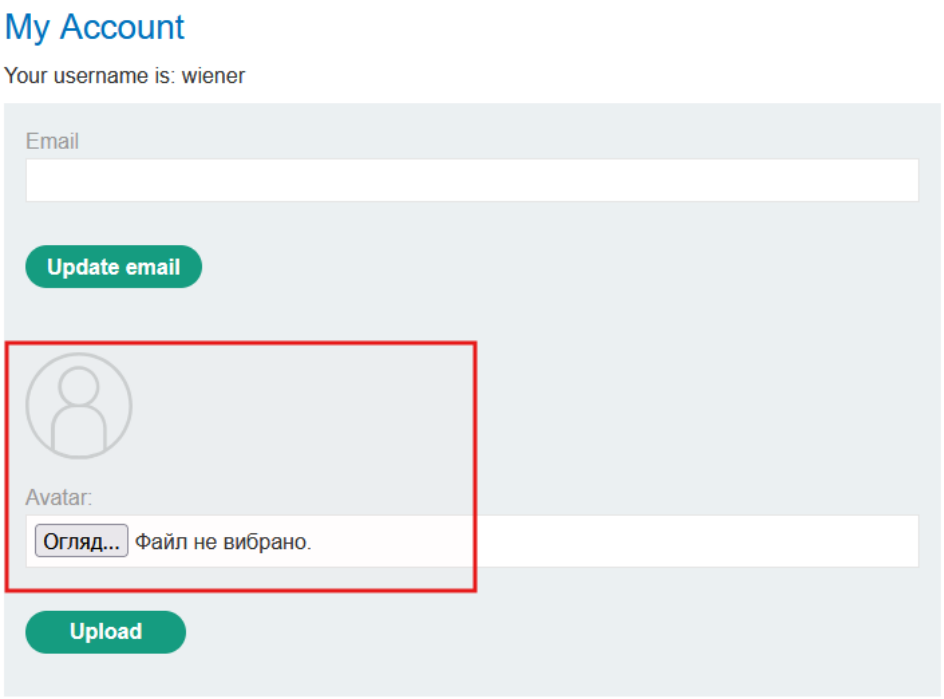

Рисунок 14 – Форма завантаження файлу у веб-застосунку

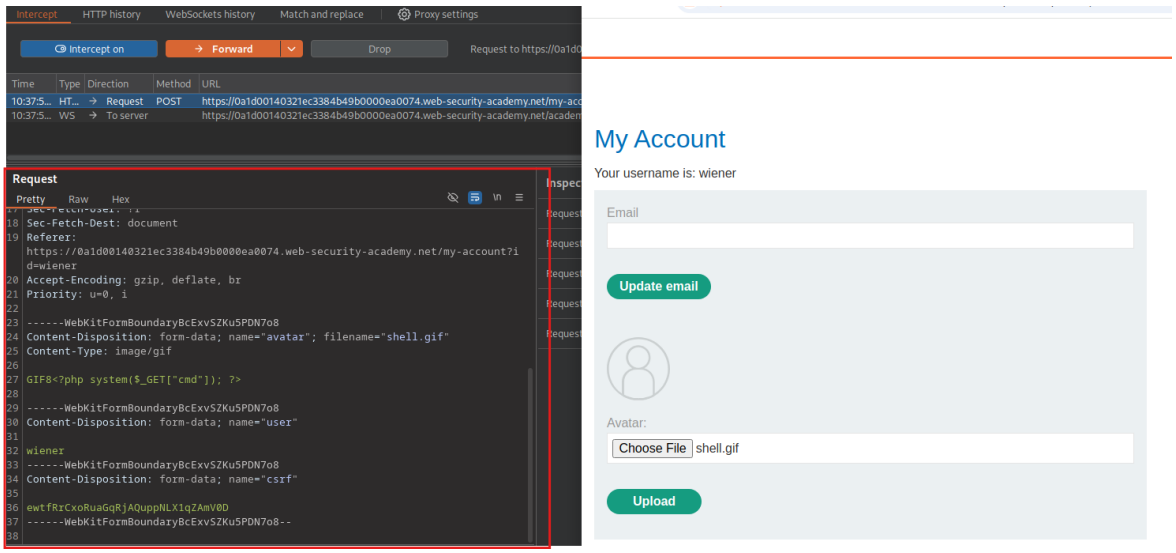

Рисунок 15 – Приклад перехоплення HTTP-запиту від час завантаження зображення на веб-сервер

Якщо сервер зберігає завантажені файли у доступному каталозі та дозволяє виконання скриптів, можливо завантажити серверний файл, який виконує команди або зчитує локальні ресурси.

Приклади мінімальних PHP-фрагментів, що ілюструють потенційний ризик неконтрольованого виконання коду:

```
<?php echo file_get_contents('/proc/self/environ'); ?>
Або
<?php echo system($_GET['command']); ?>
```

У разі некоректної реалізації механізмів валідації можливі такі способи обходу обмежень:

1. Подвійне розширення файлу (Double Extension). Якщо сервер перевіряє лише наявність дозволеного розширення в імені файлу, можливе використання конструкцій типу:

- image.jpg.php
- document.pdf.php

2. Маніпуляція заголовком Content-Type. Під час завантаження файл передається у форматі *multipart/form-data* із MIME-типом (наприклад, image/jpeg). Якщо сервер довіряє лише цьому заголовку без перевірки

реального вмісту файлу, його можна змінити через перехоплення HTTP-запиту (наприклад, за допомогою Burp Suite).

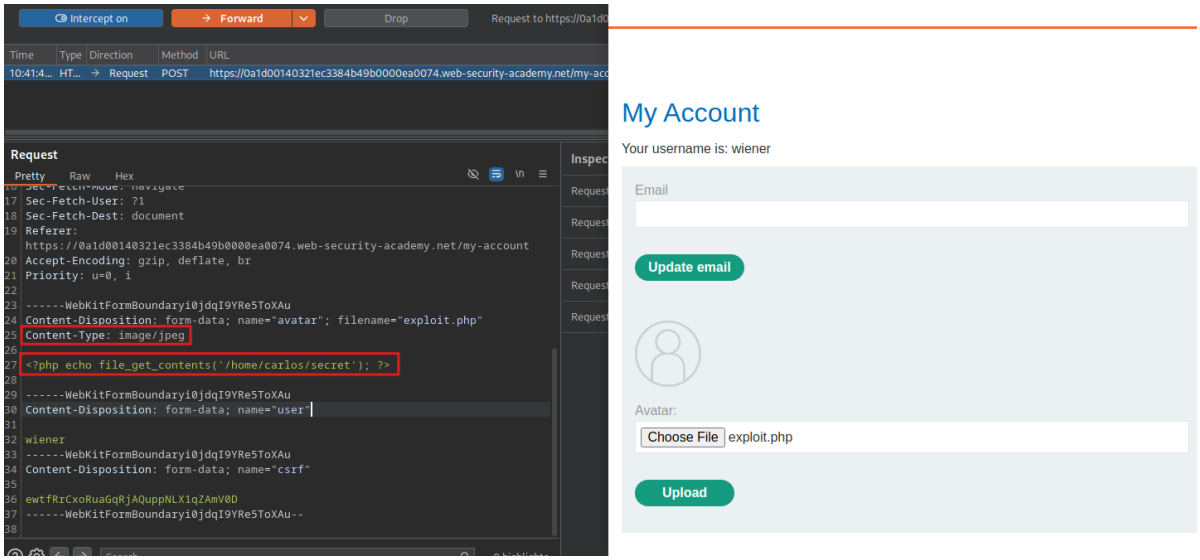

Рисунок 16 – Приклад реалізації File Upload Attack (модифікація заголовка Content-Type)

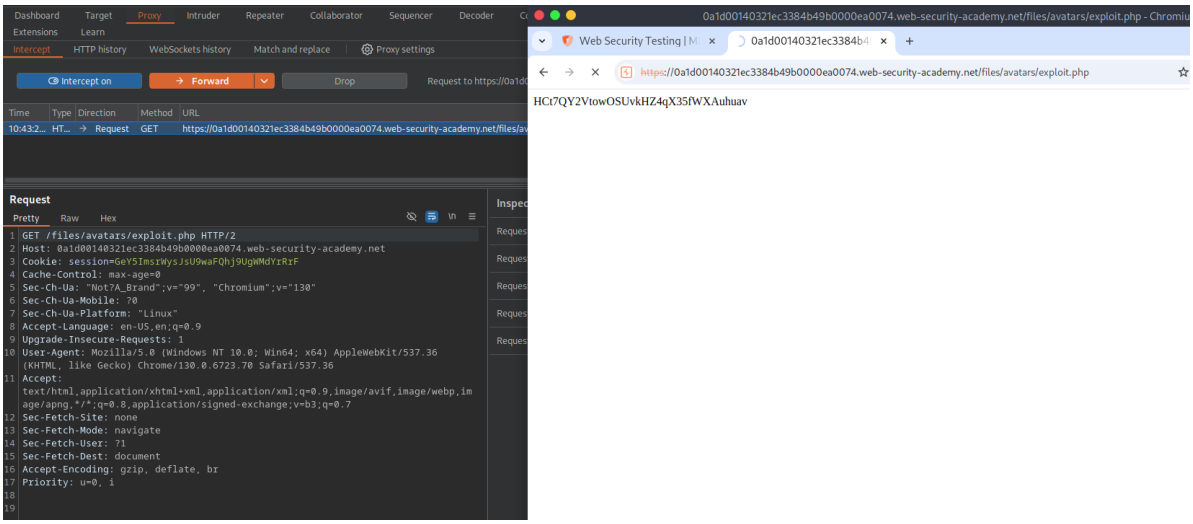

Рисунок 17 – Успішно реалізована атака File Upload

3. Використання альтернативних або нестандартних рішень. Наприклад:

- .phtml
- .php5
- .phar

Якщо фільтр перевіряє лише .php, інші інтерпретовані розширення можуть залишитися поза контролем.

4. Зміна регістру символів. У випадку регістрозалежної перевірки можливі варіанти:

- .PHP
- .Php
- .pHp

5. Додавання спеціальних символів або пробілів. Наприклад:
- shell.php
- shell.php.

6. Вбудований шкідливий код у допустимий формат (Polyglot-файли). Наприклад, зображення, яке починається з валідного JPEG-заголовка, але містить серверний код далі. Якщо сервер перевіряє лише сигнатуру файлу, такий файл може пройти перевірку.

Завдання на лабораторну роботу

Завдання 1. Дослідження вразливості Broken Access Control на основі JWT.

У даному завданні досліджується вразливість контролю доступу, пов’язана з некоректною перевіркою цифрового підпису JSON Web Token.

У разі, якщо сервер довіряє значенням, що містяться у Payload токена, некоректно перевіряє алгоритм підпису, або приймає токени з алгоритмом ***none***, клієнт може змінити роль користувача (наприклад, на ***admin***) та отримати несанціонований доступ до привілейованих ресурсів.

1. Визначте IP-адресу контейнера з веб-застосунком OWASP Juice Shop, виконавши в терміналі команду:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' juice-shop.vm
```

2. Запустіть інструмент **Burp Suite** та відкрийте вбудований браузер.

3. Перейдіть до веб-застосунку за адресою:

```
http://<IP-адреса>:3000
```

4. Переконайтеся, що режим перехоплення HTTP-запитів (Intercept) вимкнений.

5. Зареєструйте новий обліковий запис на веб-сайті Juice Shop.

6. Увімкніть режим перехоплення HTTP-запитів (Intercept).

7. На веб-сайті Juice Shop перейдіть до:

http://<IP-адреса>:3000/rest/user/whoami

8. Зафіксуйте перехоплений HTTP-запит та проаналізуйте його структуру.

9. Передайте запит до вкладки Repeater для подальшої модифікації:

10. Видаліть із HTTP-запиту заголовок (може впливати на механізм кешування та завадити отриманню повної відповіді сервера):

If-None-Match: ...

11. Надішліть HTTP-запит (Send) та переконайтеся, що сервер повертає інформацію про поточного користувача:

12. HTTP-запит містить JWT-токен (значення після ***token***=), скопіюйте його та вставте на веб-сайт https://jwt.io для декодування:

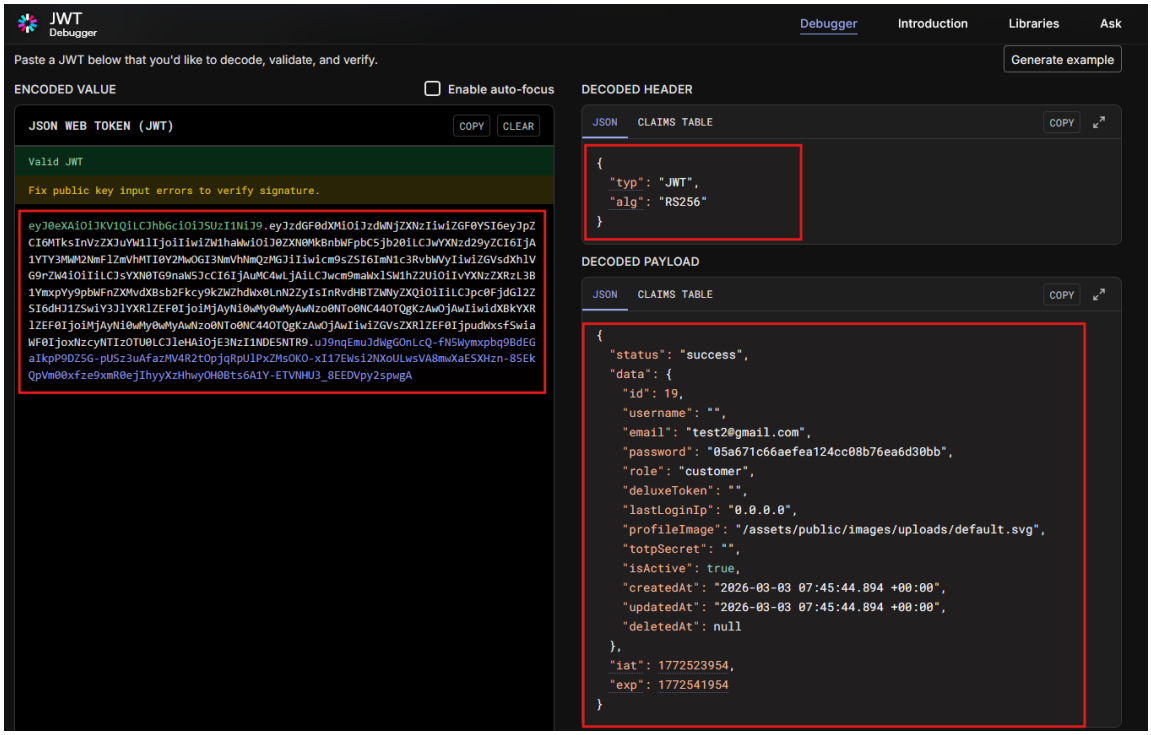

Рисунок 18 – Декодований JWT-токен

13. Проаналізуйте декодовану структуру JWT-токена (Header, Payload, Signature).

14. Скопіюйте декодоване значення Header та відкрийте сервіс Cyber Chef (https://gchq.github.io/CyberChef/).

15. На веб-сайті Cyber Chef із списку **Operations** оберіть **To Base64** та перетягніть його в поле **Recipe**.

16. Приберіть символ = з поля **Alphabet**.

17. В полі **Input** з декодованим Header JWT-токена змініть значення параметра **“alg”** з “**RS256”** на **“none”.**

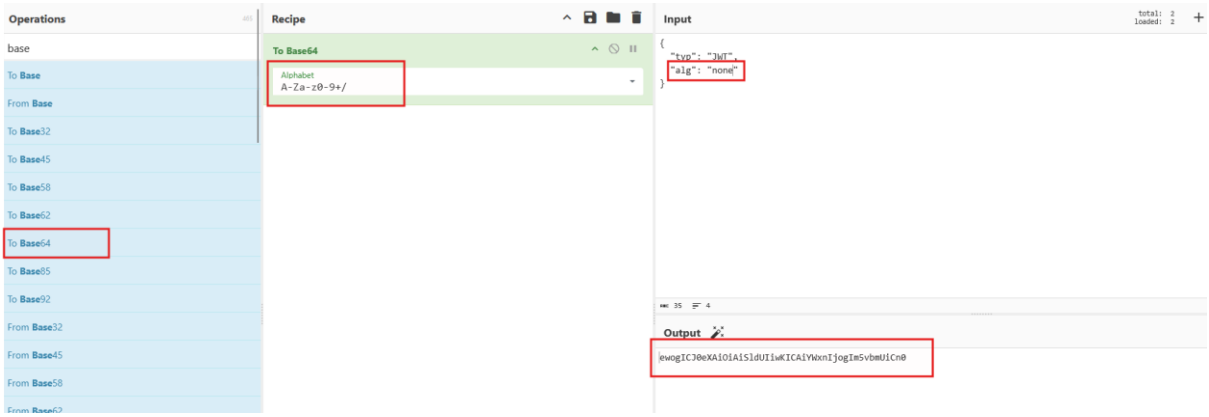

Рисунок 19 – Успішно закодований Header JWT-токена

18. Скопіюйте отримане закодоване значення з поля **Output** в будь-який текстовий редактор, додавши в кінці символ крапки.

19. Аналогічно скопіюйте Payload JWT-токена.

20. Змініть значення полів:

- “id” → 1 (ID адміністратора).
- “email” → jwtn3d@juice-sh.op.
- “role” → “admin”.

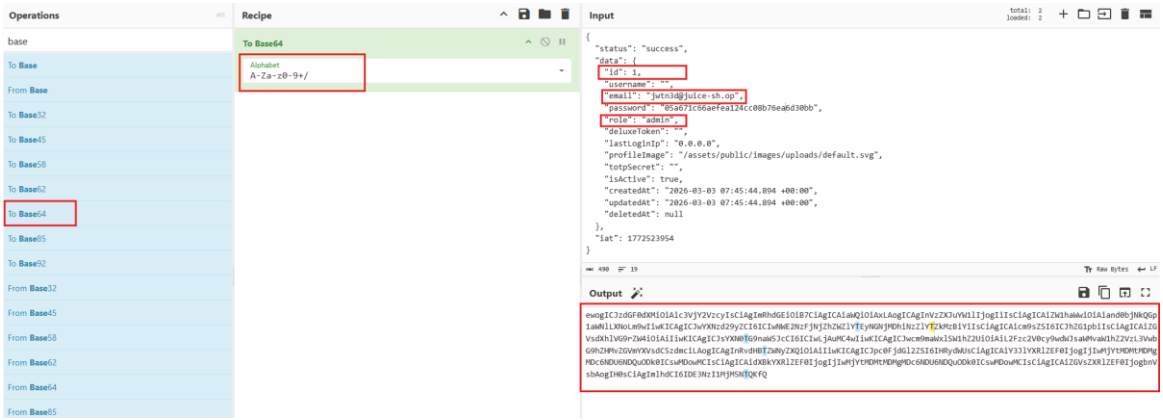

Рисунок 20 – Зміна параметрів Payload JWT-токена

21. Додайте закодоване значення Payload в текстовий редактор (п. 18). В кінці сформованого JWT-токена додайти символ крапки. Оскільки алгоритм встановлено в значення ***none***, третя частина токена (Signature) відсутня.

22. У вкладці Repeater (Burp Suite) замініть початкове значення параметра ***token=*** на сформований токен.

23. Надішліть модифікований HTTP-запит на сервер (Send).

24. Проаналізуйте відповідь сервера. Якщо сервер повертає інформацію про адміністратора, це свідчить про відсутність належної перевірки підпису токена та наявність вразливості Broken Access Control.

Завдання 2. File Upload Attack.

Мета – дослідити вразливість неконтрольованого завантаження файлів та реалізувати атаку обходу перевірки розширення з подальшим виконання довільного коду на сервері. Очікуваний результат – отримання віддаленого виконання команд (RCE) та захоплення прапора.

1. Зареєструйте обліковий запис на платформі PicoCTF:

https://picoctf.org/

2. Перейдіть до навчального завдання категорії Web Exploitation:

https://play.picoctf.org/practice/challenge/445?category=1&page=1&search=Trickster

3. Натисніть Launch Instance для запуску вразливого веб-сайту.

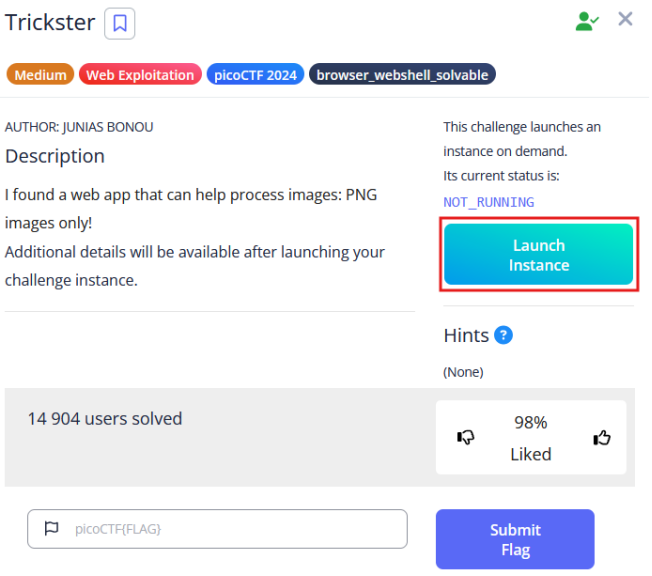

Рисунок 21 – Запуск вразливого веб-сайту

4. Отримайте URL-адресу запущеного застосунку (**Try it here!**) та відкрийте її у вбудованому браузері Burp Suite. Вимкніть режим перехоплення запитів.

5. Відкрийте вихідний код веб-додатку та проаналізуйте функціонал форми завантаження файлів. Визначте:

- Які типи файлів дозволені.
- Чи здійснюється перевірка розширення.
- Чи присутня клієнтська валідація.

6. За допомогою інструмента ffuf виконайте пошук каталогу, який використовується для зберігання завантажених файлів.

```
ffuf -u <URL_веб-сайту>/FUZZ -w /usr/share/wordlists/dirb/common.txt -s
```

7. Проаналізуйте отримані результати та визначте каталог, доступний для перегляду через веб-браузер.

8. Завантажте будь-який файл з розширенням .png та проаналізуйте:

- Чи приймає сервер файл.
- Чи відображається він у веб-застосунку.
- Чи змінюється ім’я або шлях збереження.

9. Створіть порожній .php файл з розширенням .php:

```
touch test.php
```

10. Спробуйте завантажити його на веб-сайт та проаналізуйте відповідь застосунку. Визначте, чи реалізована перевірка розширення файлу.

11. Перейдіть за посиланням та скопіюйте готовий Web Shell PHP cmd:

```
https://www.revshells.com/
```

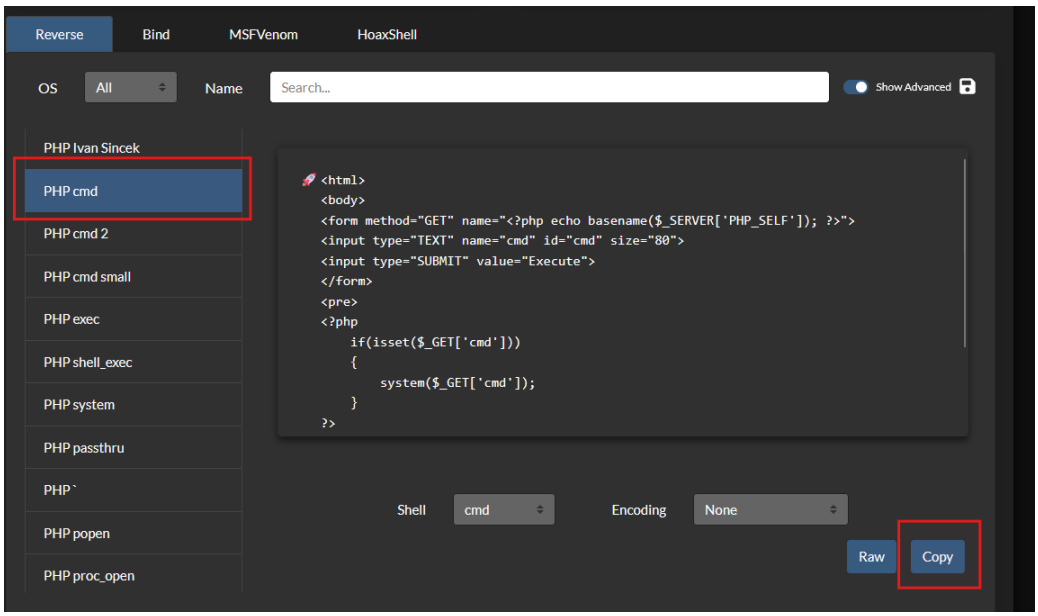

Рисунок 22 – Web Shell PHP cmd

12. За допомогою текстового редактора nano створіть файл shell.png.

nano shell.png

13. Вставте скопійований Web Shell та збережіть файл.

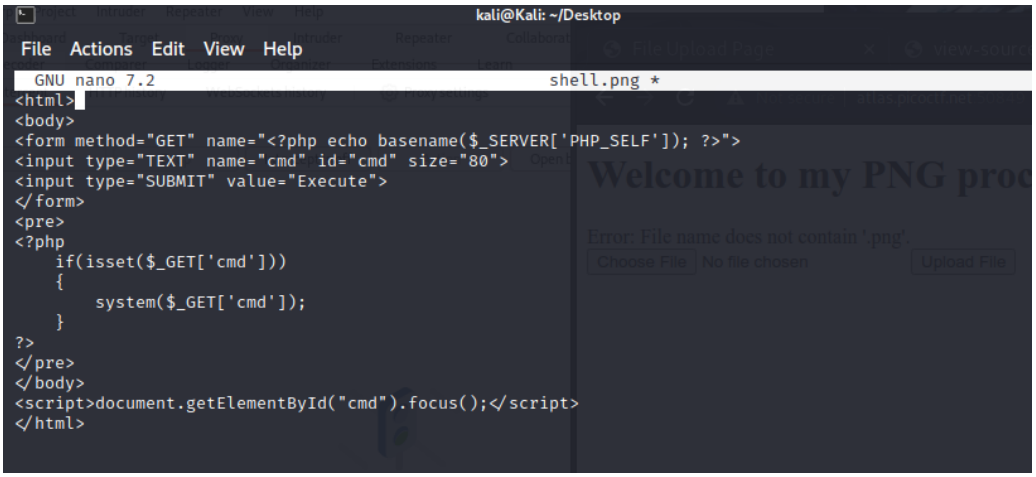

Рисунок 23 – Вміст файлу shell.png

14. Увімкніть режим перехоплення запитів в Burp Suite.

15. Завантажте створений файл shell.png на веб-сайт та перейдіть до перехопленого запиту в Burp Suite.

16. Модифікуйте параметри запиту:

- Змініть значення filename на **shell.png.php**

- Додайте сигнатуру ***PNG*** перед html-кодом для імітації файлу зображення.

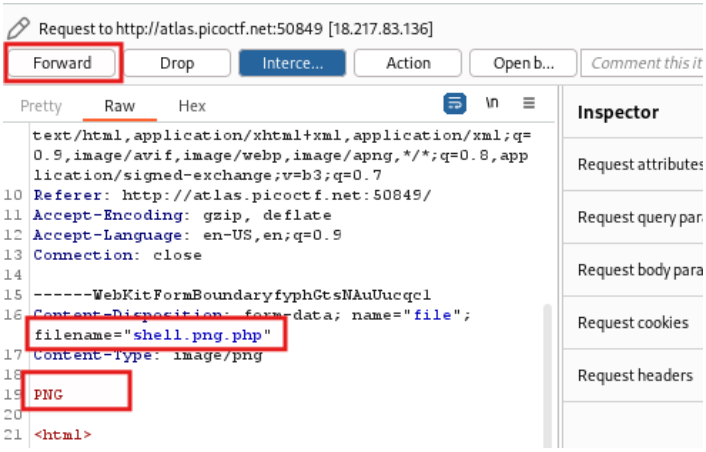

Рисунок 24 – Використання подвійного розширення для обходу фільтру

17. Надішліть модифікований HTTP-запит на сервер (Forward), після чого вимкніть режим перехоплення. Переконайтеся в успішності завантаження файлу.

18. Перейдіть до завантаженого файлу:

http://<URL-адреса-завдання>/<виявлений-каталог>/shell.png.php>

19. Перевірте, чи виконується серверний код.

20. Виведіть вміст поточного каталогу та каталогу на рівень вище (../), ввівши команду ***ls*** та натиснувши Execute. Знайдіть текстовий файл з випадковою назвою.

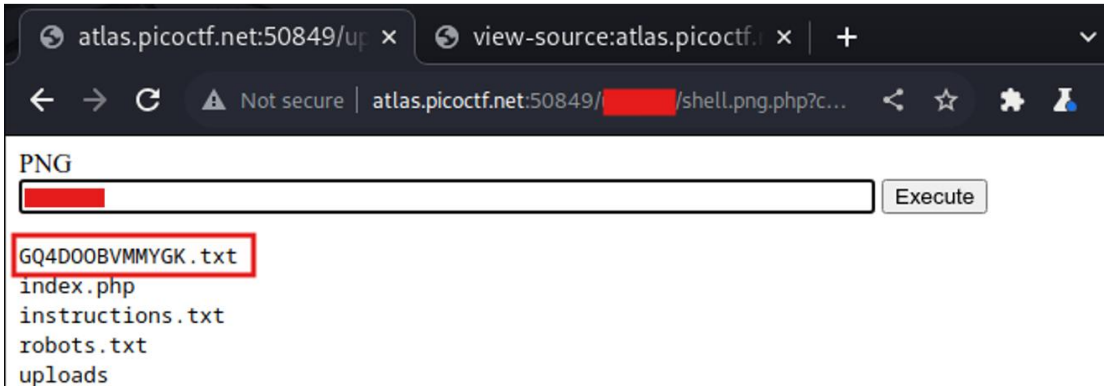

Рисунок 25 – Приклад знайденого текстового файлу

21. Виведіть вміст знайденого текстового файлу за допомогою команди ***cat***. У разі успішного виконання, ви отримаєте прапор у форматі picoCTF{}.

22. Введіть знайдений прапор на платформі picoCTF та натисніть Submit Flag. Переконайтеся, що завдання зараховано.

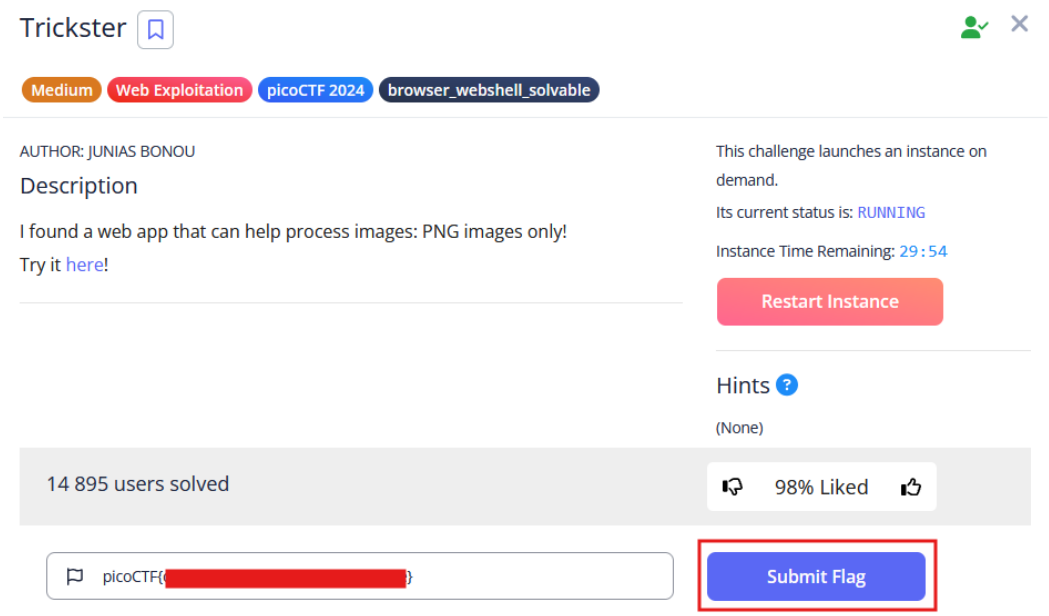

Рисунок 26 – Приклад коміту прапора

Завдання 3. Дослідження вразливості Path Traversal.

1. Зареєструйте обліковий запис на платформі PortSwigger:

https://portswigger.net/

2. Перейдіть до наступного завдання:

https://portswigger.net/web-security/file-path-traversal/lab-simple

3. Ознайомтеся з описом завдання та запустіть вразливий веб-сайт.

4. Визначте параметр HTTP-запиту, який відповідає за формування шляху до файлу на сервері.

5. Дослідіть можливість маніпуляції значенням цього параметра з метою доступу до файлів поза межами дозволеного каталогу.

6. Реалізуйте атаку Path Traversal та отримайте доступ до системного файлу сервера (відповідно до умов завдання).

Контрольні запитання

1. Яка основна функція веб-сервера у моделі клієнт-сервер?
2. Який компонент HTTP-запиту містить дані, що передаючться методом POST?
3. Яке призначення інструмента Burp Suite?
4. Який модуль Burp Suite дозволяє багаторазово змінювати та повторно надсилати HTTP-запити?
5. До якої категорії OWASP Top 10 належить Path Traversal?
6. З яких частин складається JWT-токен?
7. У чому полягає вразливість при використанні алгоритму “none” у JWT?
8. У чому полягає атака Double Extension під час File Upload?
9. Чому перевірка типу файлу лише на стороні клієнта є небезпечною?
10. Який заголовок HTTP визначає тип переданих даних під час завантаження файлу?
11. Який ризик виникає, якщо сервер зберігає завантажені файли у доступному каталозі та дозволяє виконання скриптів?

Список джерел

1. File uploads | Web Security Academy. *PortSwigger*. URL: https://portswigger.net/web-security/file-upload.
2. OWASP Top 10:2025. *OWASP*. URL: https://owasp.org/Top10/2025/.
3. JWT: Vulnerabilities, Attacks & Security Best Practices. *VAADATA - Ethical Hacking Services*. URL: https://www.vaadata.com/blog/jwt-json-web-token-vulnerabilities-common-attacks-and-security-best-practices/ (date of access: 03.03.2026).
4. Path Traversal | OWASP Foundation. *OWASP*. URL: https://owasp.org/www-community/attacks/Path_Traversal.