

Desarrollo Seguro de Aplicaciones -

Práctica 2

Autores - Grupo 'Error 404'

- Juan Cruz Cassera Botta, 17072/7
- Brian Llamocca, 02037/9
- Franco Camacho, 16194/1

Notas

- **Fecha de entrega: 8/05/2025 a las 17:59.**
- Términos relacionados:
 - OWASP top 10 (2021): A01 (pérdida de control de acceso).
 - OWASP top 10 (2013): A04 (referencia directa insegura a objetos).
 - OWASP top 10 (2013): A07 (ausencia de control de acceso a las funciones).
- Para la realización de esta práctica, cada grupo tendrá un repositorio de código en GitHub, el cual podrán clonar con "git clone <URL git del repositorio del grupo>".
- Se requiere instalar docker y docker compose.
- Luego de clonar el repositorio y cambiar directorio a "practica2/", corremos el comando ***docker-compose up***.

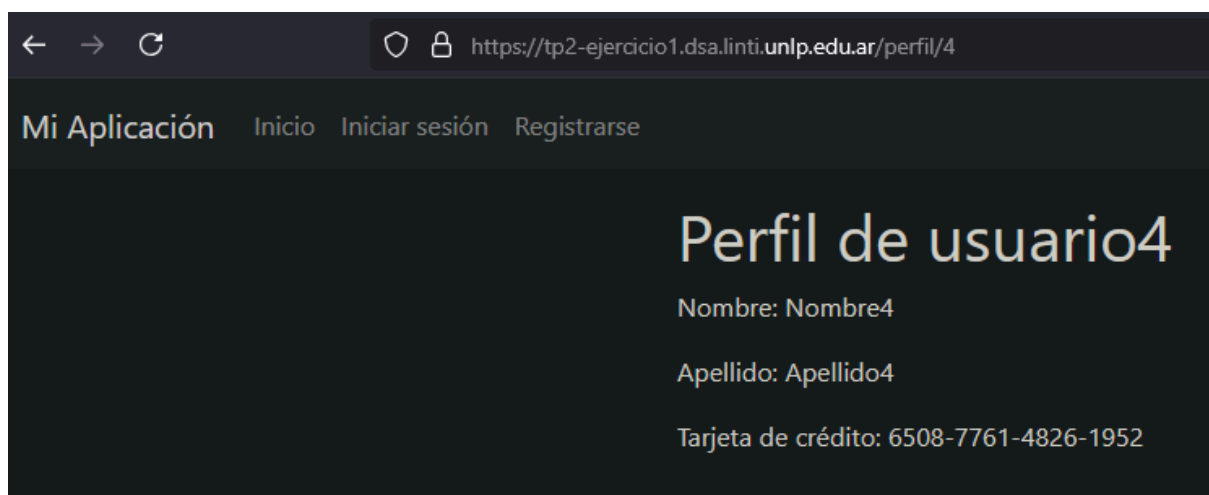
Ejercicio 1 - Broken Access Control

- [Reto ejercicio1](#).
- Loguearse con:
 - Usuario: **user**
 - Password: **pass**
- **Objetivo:** Obtener las tarjetas de crédito de otros usuarios. En el CTF la flag estará en la tarjeta de crédito del usuario "admin".

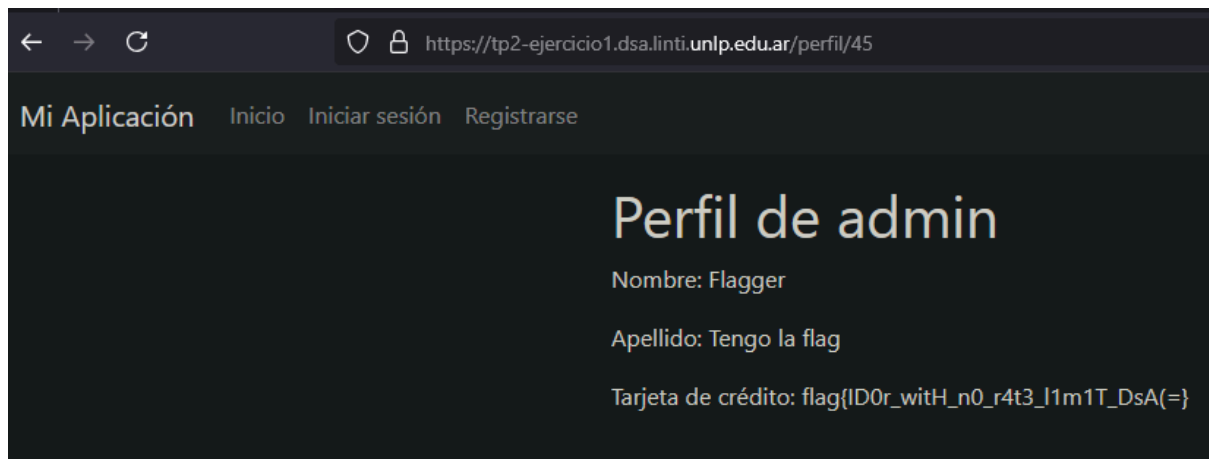
Cómo explotamos la vulnerabilidad

Luego de loguearnos en la página podemos ver, en la URL, que estamos posicionados en: <https://tp2-ejercicio1.dsa.linti.unlp.edu.ar/perfil/1>. Esto nos indica que los perfiles de las cuentas creadas en la página se encuentran en la URL <https://tp2-ejercicio1.dsa.linti.unlp.edu.ar/perfil/{id-del-usuario}>. Por ende, podemos directamente modificar la URL borrando el "1" y escribiendo otro número entero que sería el ID de otro usuario. Haciendo esto, tenemos acceso a los datos, **incluyendo las tarjetas de crédito**, de todos los usuarios del sistema.

Ejemplo:



La flag se encuentra en el perfil del admin que tiene id = 45:



flag{ID0r_witH_n0_r4t3_l1m1T_DsA(=)}

Cómo se podría evitar este tipo de vulnerabilidad

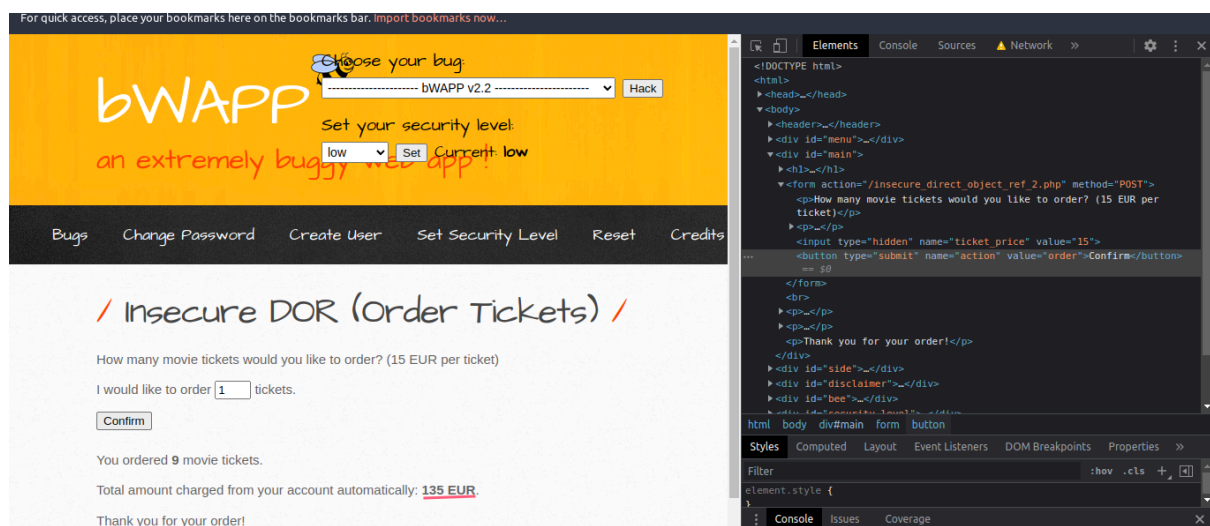
En este ejercicio se encontraron **2 vulnerabilidades**:

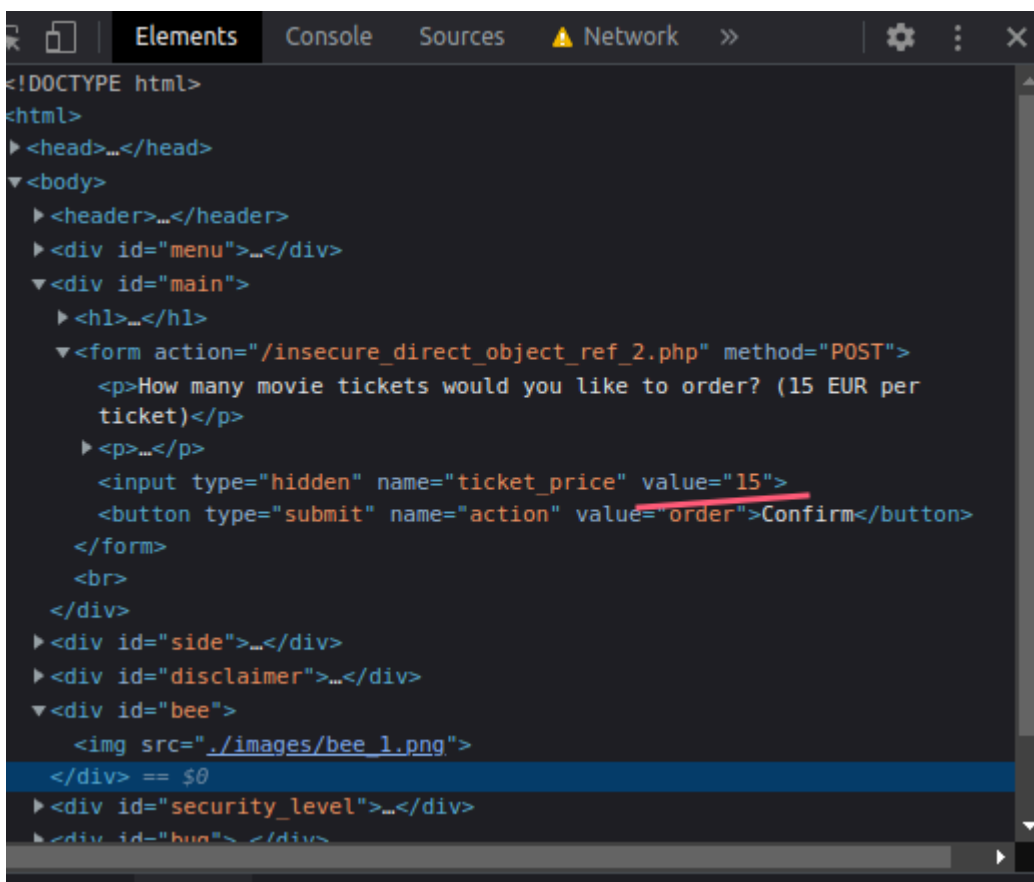
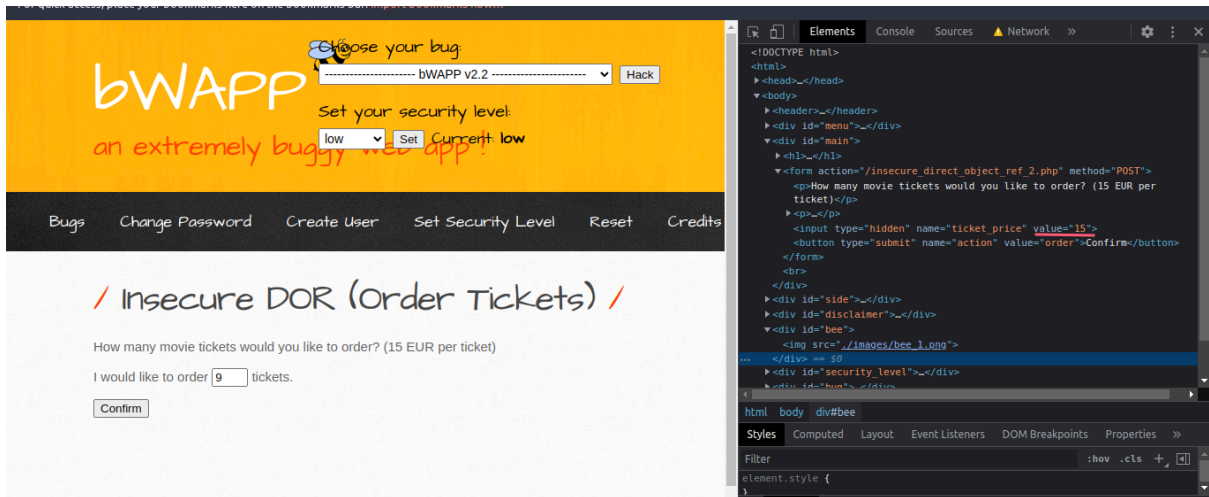
- 1) Se puede acceder al perfil de cualquier usuario sin estar logueados, usando la URL *https://tp2-ejercicio1.dsa.linti.unlp.edu.ar/perfil/X*.
 - a) Esto se podría solucionar validando que para poder ingresar a esa URL, el usuario **debe** estar logueado.
- 2) Sin embargo, lo de arriba **no es suficiente**, ya que a pesar de que el usuario se loguee, aún así puede acceder al perfil de cualquier otro usuario modificando la URL a mano.
 - a) Esto se podría solucionar implementando algún **método de control de acceso** (de los cuales hay 4 tipos: DAC, MAC, RBAC y ABAC) para asegurar que cada usuario pueda ingresar solo a **SU perfil** y no al de ningún otro usuario.

Ejercicio 2 - Insecure DOR (Order Tickets)

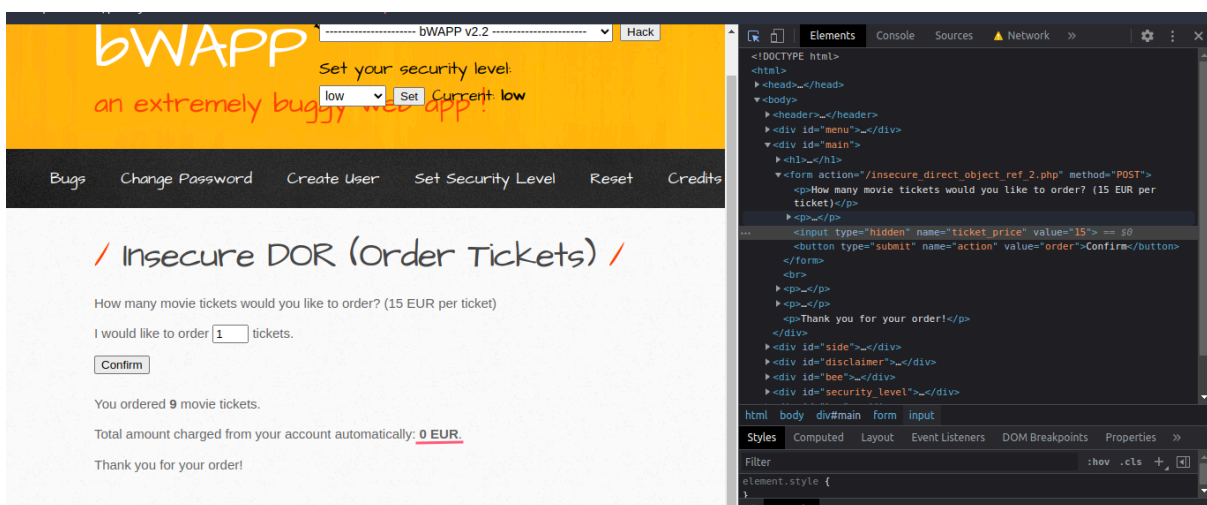
- Para los ejercicios 2, 3, 4 y 5 se utilizará la aplicación bWAPP ("an extremely buggy web app":
 - <https://sourceforge.net/projects/bwapp/files/bWAPP/>
 - <http://www.itsecgames.com/index.htm>
- Utilizando el navegador ingresar a <http://localhost:12000/install.php> (por defecto los lleva a /login.php y tira un error porque todavía no está creada la DB y seteada la config de la aplicación).
- Clic en el enlace "**Click here to install**".
- Login con user/pass: **bee/bug**.
- Luego ingresar a http://localhost:12000/security_level_set.php y configurar el nivel de seguridad en "**low**".
- En el menú "Bugs" (Portal) buscar en la lista los retos de las categorías:
 - "**A4 - Insecure Direct Object References**"
 - "**A7 - Missing Functional Level Access Control**"
- [Reto bWAPP](#).
- Esta página simula un sitio para comprar/encargar entradas de cine por un valor de "15 EUR".
- **Objetivo:** encargar tickets gratis o por un valor distinto.

Cómo explotamos la vulnerabilidad





```
Elements Console Sources Network >>
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <header>...</header>
    <div id="menu">...</div>
    <div id="main">
      <h1>...</h1>
      <form action="/insecure_direct_object_ref_2.php" method="POST">
        <p>How many movie tickets would you like to order? (15 EUR per ticket)</p>
        <p>...</p>
        <input type="hidden" name="ticket_price" value="0"> == $0
        <button type="submit" name="action" value="order">Confirm</button>
      </form>
      <br>
    </div>
    <div id="side">...</div>
    <div id="disclaimer">...</div>
    <div id="bee">
      
    </div>
    <div id="security_level">...</div>
    <div id="bun">...</div>
```



En este caso, explotamos la vulnerabilidad chequeando el código HTML de la página http://localhost:12000/insecure_direct_object_ref_2.php y dándonos cuenta de que el precio de cada ticket no está en el backend (base de datos) si no que está hardcoded en el frontend, en el atributo **value del tag input**. Por ende, como cualquiera puede modificar el código HTML de una página, modificamos el "15" por "0" y tenemos tickets gratis.

Cómo se podría evitar este tipo de vulnerabilidad

El problema principal es que no se validan los datos que envía el usuario. Se podría evitar añadiendo **validación** de los inputs del usuario, es decir, si el usuario envía una solicitud por tickets de \$0, o por tickets con valor menor a \$15, debería ser rechazado. Además de esto, el precio del ticket debería estar almacenado en el servidor y no en el cliente.

Ejercicio 3 - Insecure DOR (Change Secret)

- [Reto bWAPP](#).
- [Esta página permite cambiar el "Secret" de tu usuario](#).
- **Objetivo:** cambiar el "Secret" de otro usuario que no sea el logueado. Verificar que se cambió correctamente el secret logueando con el usuario víctima, accediendo a http://localhost:12000/insecure_crypt_storage_1.php y viendo el localStorage.

Cómo explotamos la vulnerabilidad

Primero creamos un nuevo usuario que será la víctima, con los siguientes datos:

- Usuario: user2
- Contraseña: bug
- Email: NUEVO@GMAIL.COM
- Secret: aaaaaa

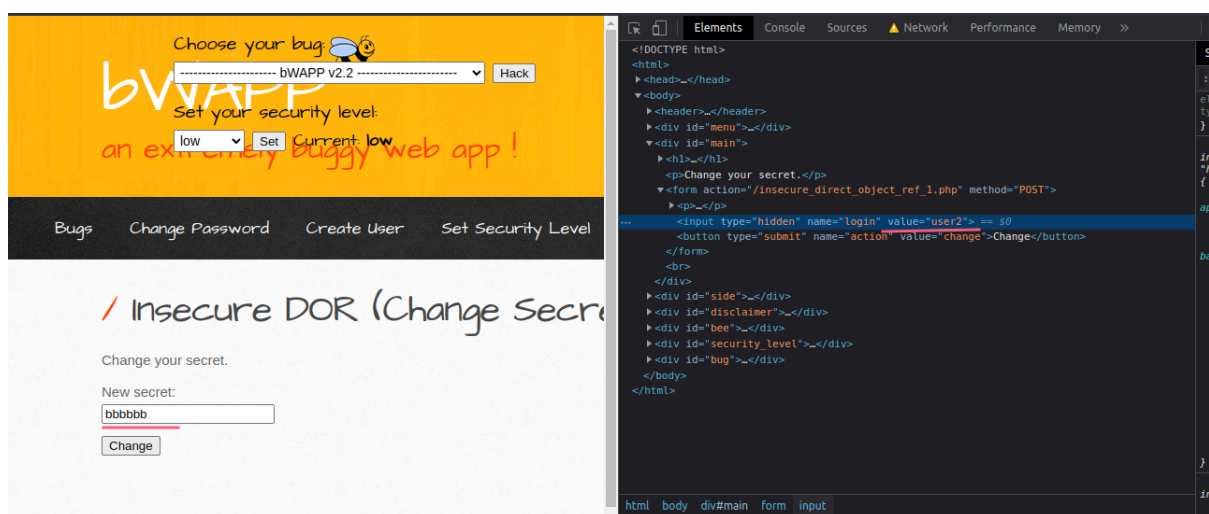
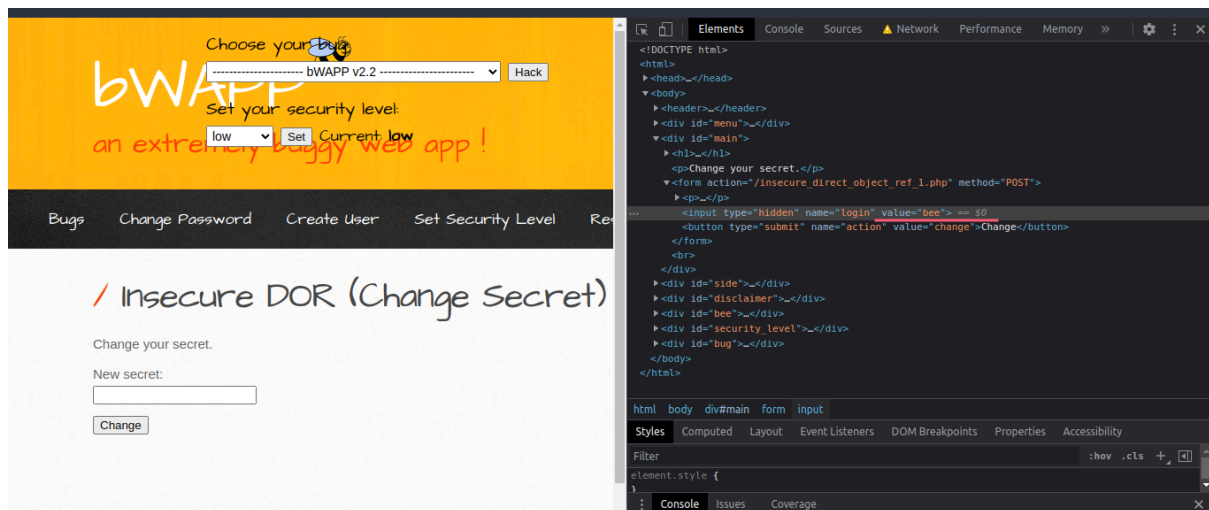


The screenshot shows a web form titled "Create User" with the subtitle "Create an extra user." The form contains the following fields and controls:

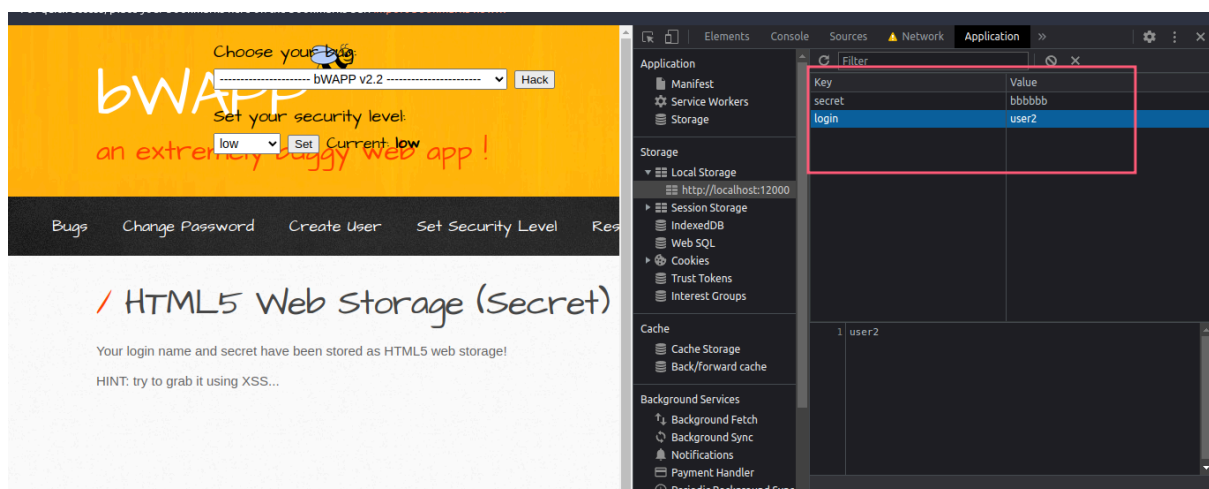
- Login:** A text input field containing "user2".
- E-mail:** A text input field containing "NUEVO@GMAIL.COM".
- Password:** A password input field with masked characters "..."
- Re-type password:** A password input field with masked characters "..."
- Secret:** A text input field containing "aaaaaa".
- E-mail activation:** A checkbox that is currently unchecked.
- Create:** A button to submit the form.

Below the form, a red error message states: "User not successfully created! An e-mail could not be sent..."

Luego nos logueamos con el usuario original, **bee/bug** y vamos a la página http://localhost:12000/user_extra.php y viendo el código HTML podemos ver que el nombre de usuario al cual se le cambiará su secret está expuesto en el tag **input**, atributo **value**. Por ende, si lo modificamos de **bee** a **user2**, como el servidor no realiza validaciones, esto modificará el secret de **user2** y no de **bee**.



Al loguearnos con user2, podemos ver que nuestro secret ya no es más “aaaaaa”, si no que ahora es “bbbbbb”, porque **bee** lo modificó por nosotros sin nuestro permiso.



Cómo se podría evitar este tipo de vulnerabilidad

Como en el ejercicio anterior, el problema está en no validar el nombre de usuario que envía el usuario logueado en la petición de cambio de secret. El servidor debería validar que si ese **nombre es distinto del nombre del usuario logueado**, la petición se rechaza.

Otra alternativa es que no figure el nombre de usuario en el HTML y que el cambio de secret se realice 100% desde el servidor.

Ejercicio 4 - Directory Traversal

- [Reto bWAPP.](#)
- [Reto bWAPP.](#)
- [Esta página permite cambiar el "Secret" de tu usuario.](#)
- **Objetivo:** listar los usuarios del sistema operativo subyacente (sistema del contenedor docker).

Cómo explotamos la vulnerabilidad

Para iniciar a conocer la vulnerabilidad, se exploró los siguientes enlaces:

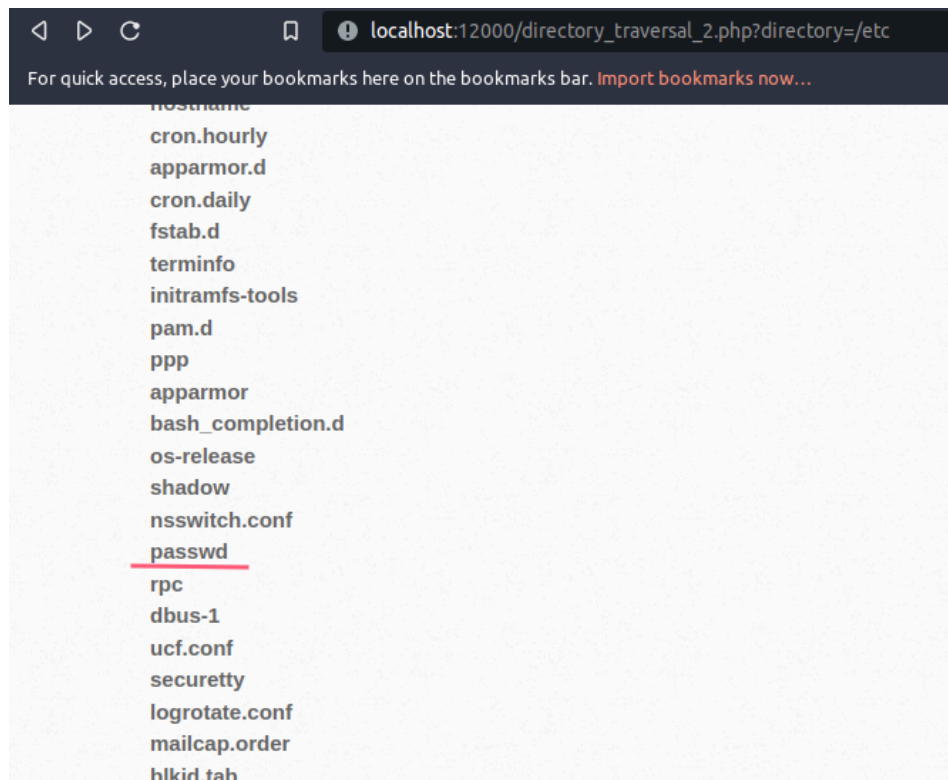
http://localhost:12000/directory_traversal_1.php?page=message.txt

Este enlace permite mostrar el contenido de un archivo de la computadora que corre en el servidor, el cual se le pasa a través del argumento "page" el nombre del archivo.

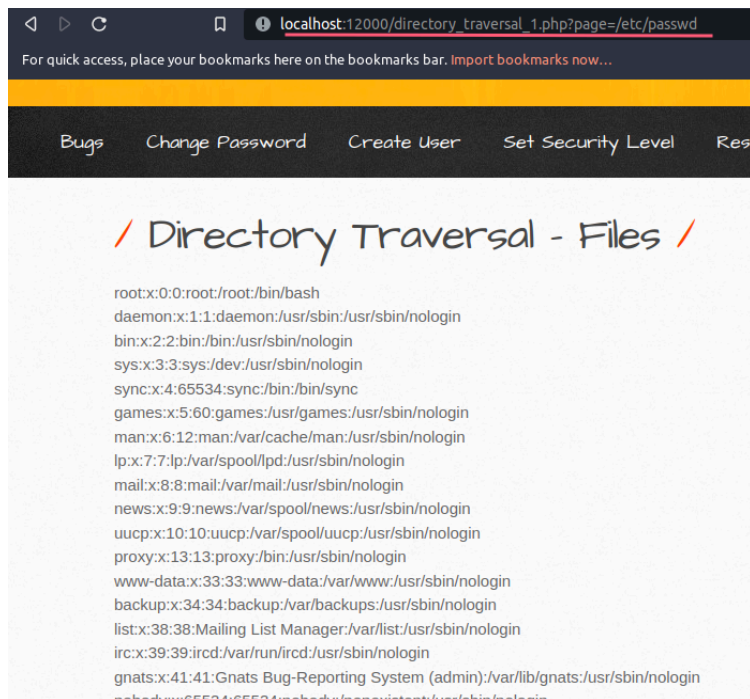
http://localhost:12000/directory_traversal_2.php?directory=documents

Este enlace permite mostrar el contenido de un directorio en particular, el cual se pasa por parámetro a través del argumento "directory".

Una vez descubierto lo anterior, en primer lugar se procede a explorar el contenido del directorio "/etc", el cual posee configuraciones del SO, de usuarios, entre otras.



En la imagen se puede apreciar que dentro de este directorio se encuentra el archivo “passwd”, archivo en el cual se encuentra información de cada usuario del sistema. Finalmente, se procede a abrir el mismo para lograr el objetivo.



Cómo se podría evitar este tipo de vulnerabilidad

Esta vulnerabilidad se podría evitar si se limitan los valores que puede llegar a tener como argumento los parámetros “page” y “directory”. De este modo, se armaría una lista de valores válidos para cada uno de ellos y así, cualquier otro valor que ingrese el usuario sería rechazado o inválido.

Por ejemplo, para el siguiente caso

[http://localhost:12000/directory_traversal_1.php?page=](http://localhost:12000/directory_traversal_1.php?page=message.txt)

Se podría armar una lista con argumentos válidos como [message.txt](#), de tal manera que solo dicho valor sea válido.

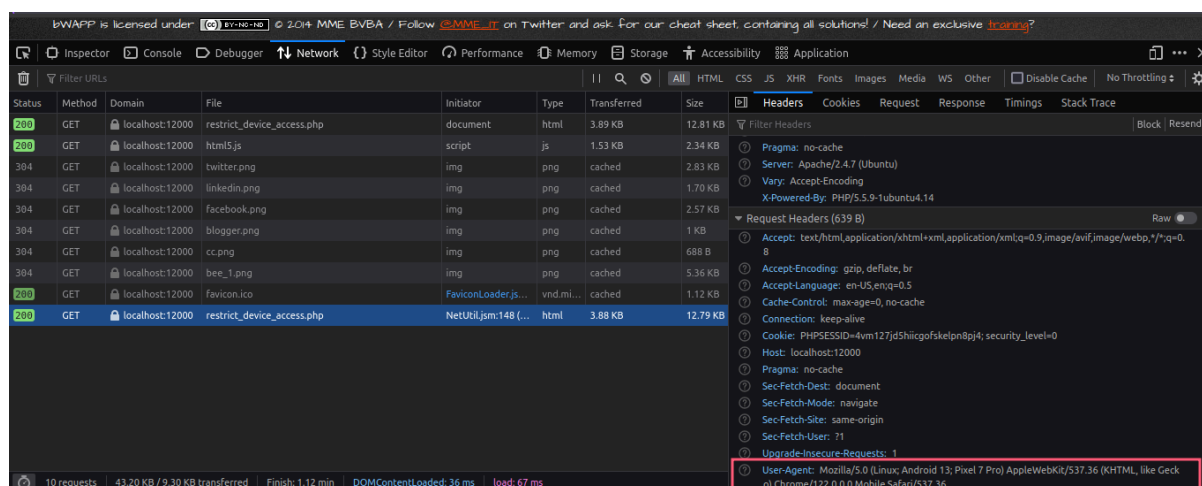
Lo mismo sería para [http://localhost:12000/directory_traversal_2.php?directory=](http://localhost:12000/directory_traversal_2.php?directory=/etc/passwd) se limita valores que puede llegar a tener como argumento el parámetro “directory”.

Ejercicio 5 - Restrict Device Access

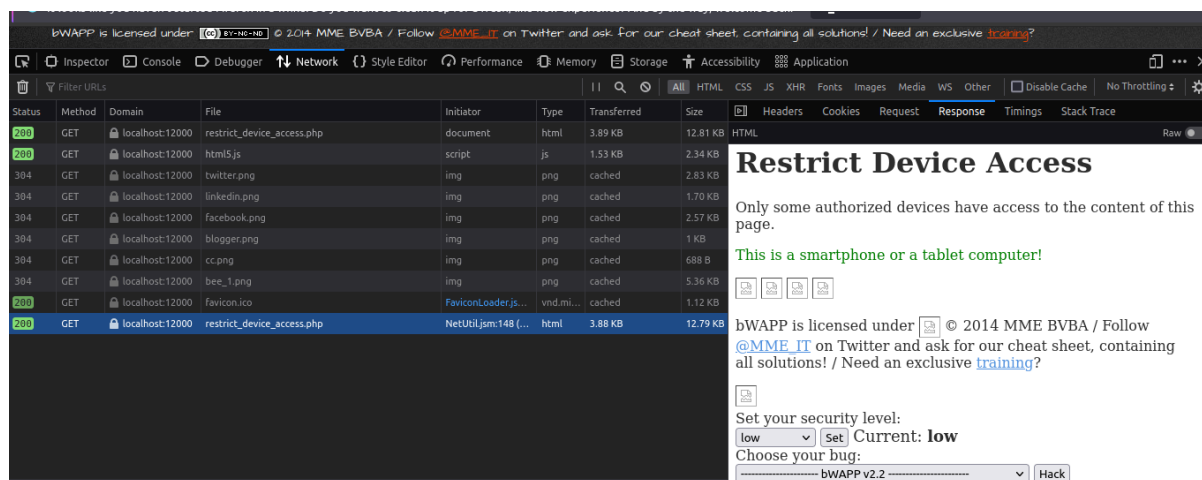
- [Reto bWAPP](#).
- Esta página solo muestra su contenido cuando se la accede con un dispositivo móvil.
- **Objetivo:** acceder sin usar un dispositivo móvil. (Ayuda: pueden utilizar Mozilla Firefox y ver la “respuesta”).

Cómo explotamos la vulnerabilidad

Para la siguiente vulnerabilidad, se revisaron las peticiones “get” realizadas. Se puede apreciar que en el header de las peticiones se envía la siguiente clave “User-Agent”, el cual tiene como valor el nombre del buscador web y dispositivo desde el que se realiza la petición.



Para poder explotar la vulnerabilidad, se modificó el valor de dicho argumento del header con uno que sea propio de un dispositivo móvil. En este caso, se utilizó el siguiente valor: “Mozilla/5.0 (Linux; Android 12; SAMSUNG SM-G991B) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Mobile Safari/537.36” para el cual se obtuvo como respuesta la página vista en la siguiente captura:



Cómo se podría evitar este tipo de vulnerabilidad

No existiría forma confiable de poder evitar este tipo de vulnerabilidad, ya que todos los posibles datos que se puedan pasar al servidor (como la clave "user-agent", tamaño de pantalla, soporte para pantalla táctil, entre otro tipo de información) son vulnerables a ser manipulados por el propio usuario.

Podría realizarse una validación más robusta teniendo en cuenta varios datos pasados por el usuario para tener un perfil más acertado del mismo, pero esto sólo retrasaría la posibilidad de ser vulnerada.

Ejercicio 6 - Subir un fix

- Cree un branch llamado **"fix-practica2"**.
- Modifique el código del ejercicio 1 para que no sea posible obtener los datos personales ni las tarjetas de crédito de otros usuarios. Solamente que se pueda acceder a los datos del usuario logueado.