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

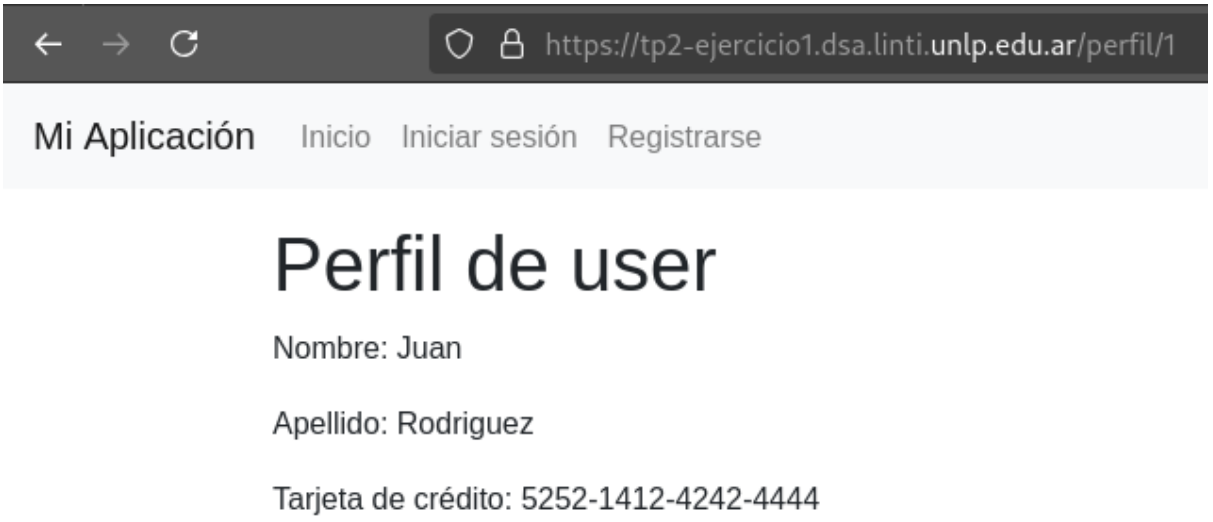
- **Fecha de entrega: 8/05/2025 a las 17:59**.
- Términos relacionados:
    - OWASP top 10 (2021): A01 (pérdida de control de acceso).
    - OWASP top 10 (2013): A04 (referencia directa insegura a objetos).
    - OWASP top 10 (2013): A07 (ausencia de control de acceso a las funciones).
- Para la realización 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

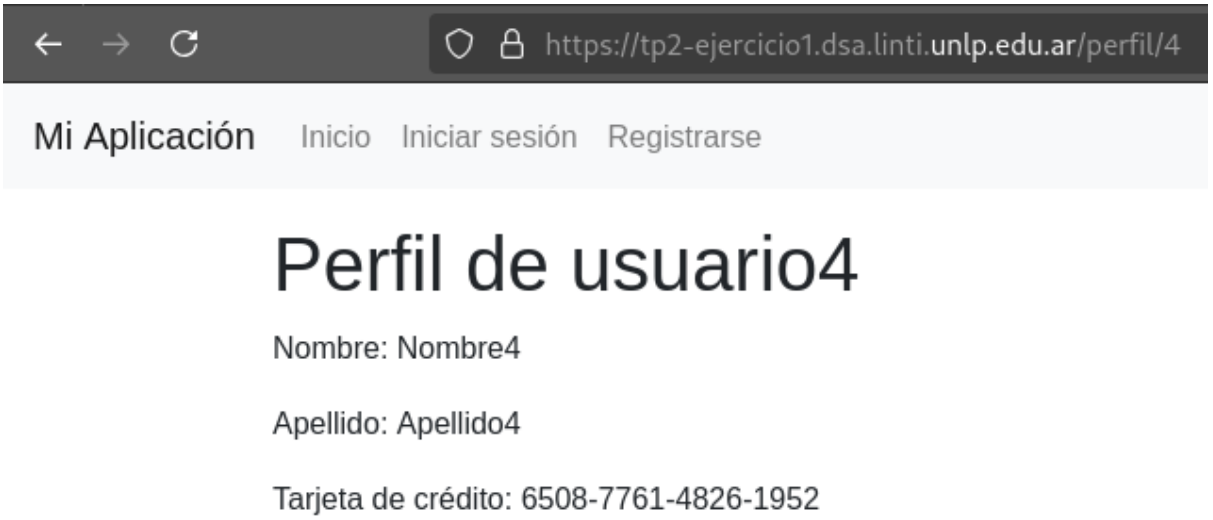
Luego de loguearnos en la página podemos ver, en la URL, que estamos posicionados en: *https://tp2-ejercicio1.dsa.linti.unlp.edu.ar/perfil/1*.



Esto nos da la pista de 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}*

De ese modo, se probó modificando la URL borrando el id del usuario “1” y escribiendo otro número entero que sería el ID de otro usuario. Al hacerlo, se comprobó que se tiene acceso a todos los datos, **incluyendo las tarjetas de crédito**, de todos los usuarios del sistema.

Ejemplo:



Al principio, se realizó la búsqueda por fuerza bruta, accediendo a los primeros 20 perfiles de usuarios. Luego de realizar sin éxito la búsqueda de la flag por el método mencionado, se desarrolló el siguiente script:

```
#!/bin/bash

# Se valida si se le ingresa al menos un argumento
if [ -z "$1" ]; then
 echo "Error 1: Ingresar como argumento la cantidad máxima de perfiles
a revisar"
 exit 1
fi

i=1
END=$1

while [[ $i -le $END ]]; do
 RESPONSE=$(curl -s
"https://tp2-ejercicio1.dsa.linti.unlp.edu.ar/perfil/$i")
 # SE verifica si es el perfil del admin
 if echo "$RESPONSE" | grep -q "admin"; then
   # Se Busca y extrae la flag en tarjeta de crédito
   FLAG=$(echo "$RESPONSE" | sed -n 's/.*Tarjeta de crédito:
\(flag{[^}]*}\).*/\1/p')

   if [ -n "$FLAG" ]; then
     echo "Flag encontrada en perfil $i: $FLAG"
     break
```

```
   fi
 fi
 (( i++ ))
done

if [ $i -gt $END ]; then
 echo "No se encontró la flag en los primeros $END perfiles"
 exit 2
else
 exit 0
fi
```

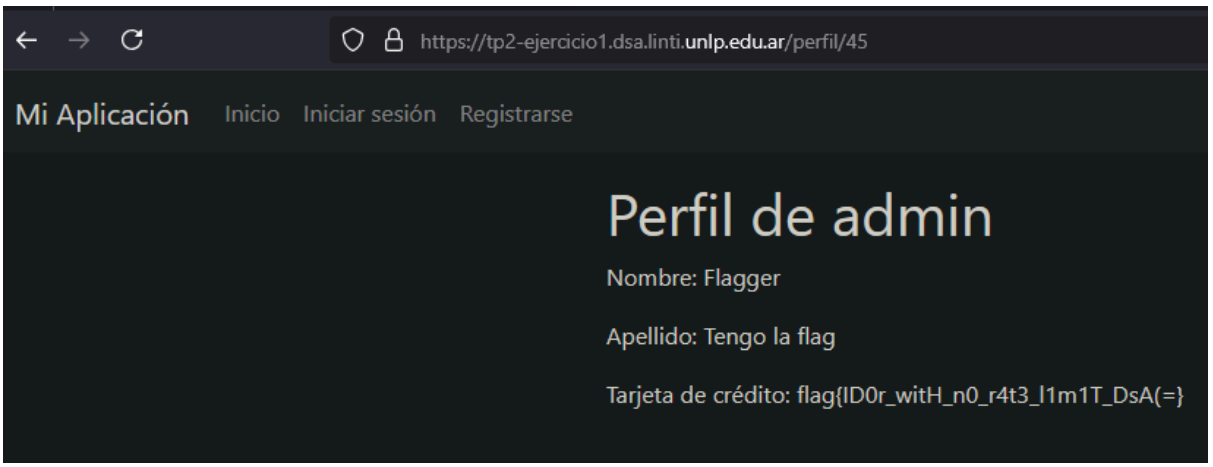
Algunas consideraciones del mismo fueron:

- Para evitar problemas de bucle indefinido, se le solicita por argumento un valor máximo de perfiles a tener en cuenta a la hora de la búsqueda.
- Se notó que la petición a la url devuelve dentro del html: “Perfil de {nombre del usuario}”. Entonces, al saber que el usuario que tiene la flag es el usuario admin, se utilizó el mismo como condición de corte.
- Por el enunciado se tuvo en cuenta que la flag se encuentra luego del campo “Tarjeta de crédito”, por lo que se guardó el contenido del mismo siguiendo el formato flag{valor de la flag}

Teniendo todo esto en cuenta, se procedió a ejecutar el script, obteniendo lo siguiente:

```
braii@debian:~/Documentos/DSA/Practica/TP2$ ./scriptP2E1v2.sh 100
Flag encontrada en perfil 45: flag{ID0r_witH_n0_r4t3_l1m1T_DsA(=}
```

Para corroborarlo, se ingresó al perfil 45 desde el navegador:



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.

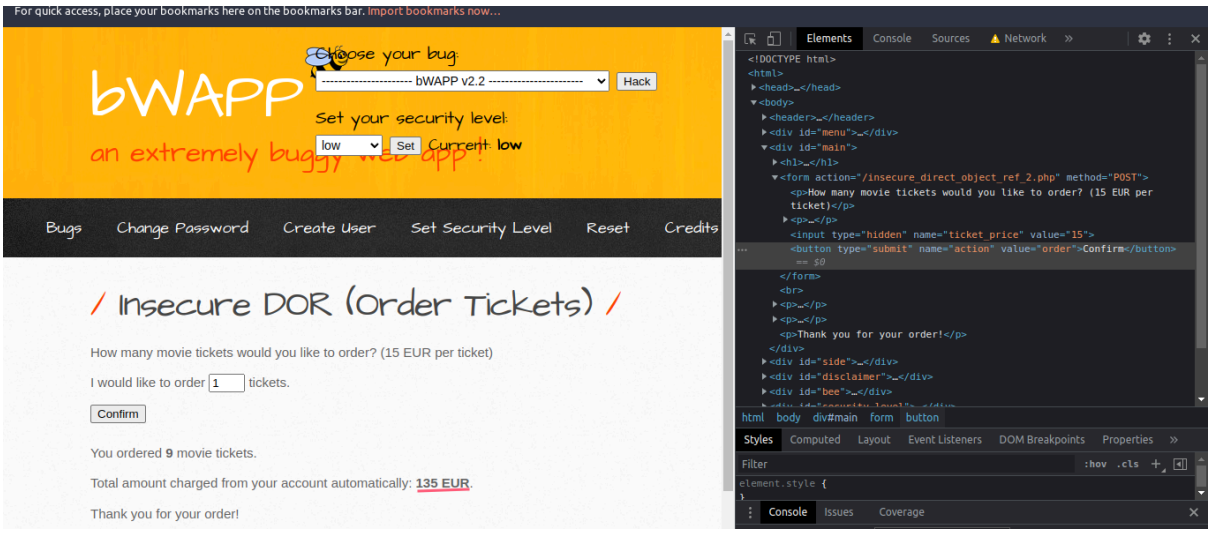
2) Sin embargo, lo de arriba **no es suficiente,** ya que a pesar de que el usuario se loguee, aún así puede acceder al perfil de cualquier otro usuario modificando la URL a mano.
    a) Esto se podría solucionar implementando algún **método de control de acceso** (de los cuales hay 4 tipos: DAC, MAC, 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)

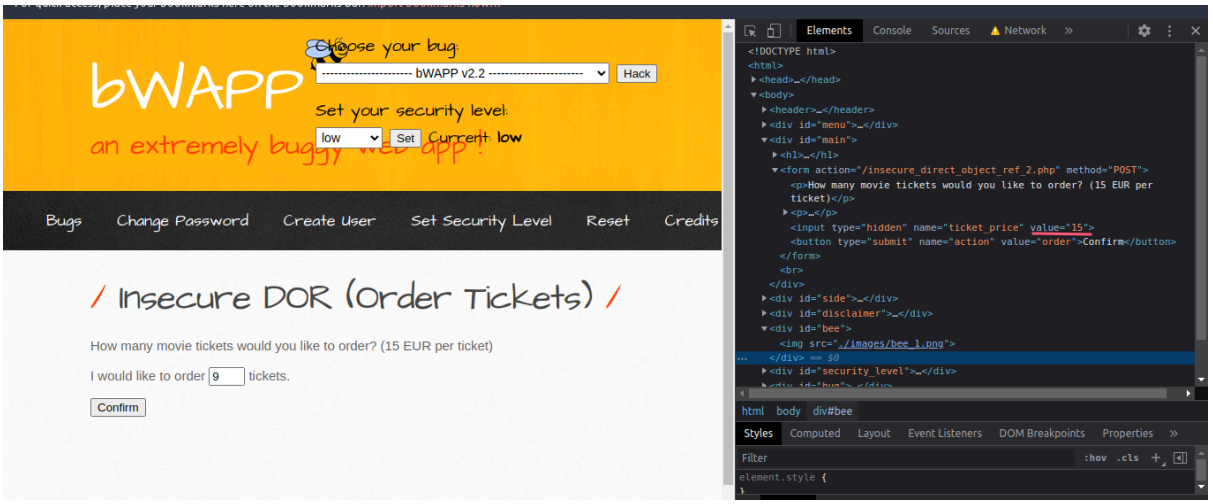
- Para los ejercicios 2, 3, 4 y 5 se utilizará la aplicación bWAPP (“an extremely buggy web app”:
    - https://sourceforge.net/projects/bwapp/files/bWAPP/
    - http://www.itsecgames.com/index.htm
- Utilizando el navegador ingresar a http://localhost:12000/install.php (por defecto los lleva a /login.php 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

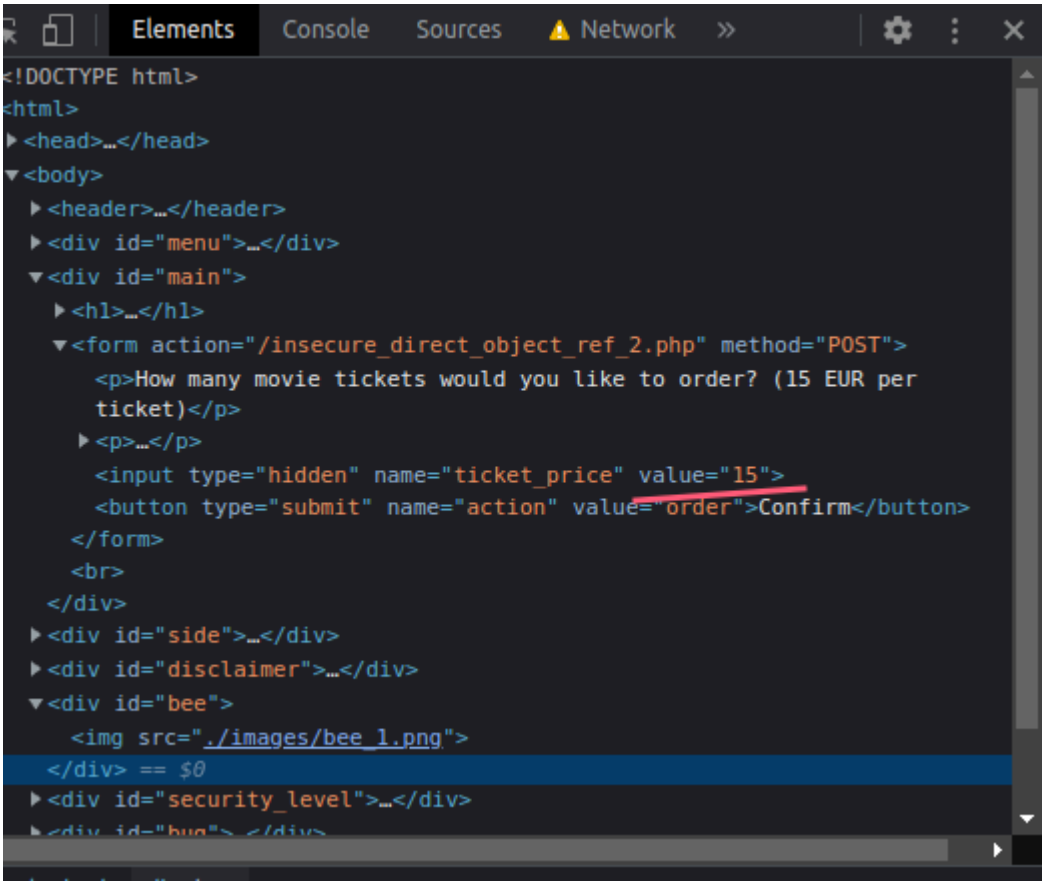
Para explotar la vulnerabilidad, primero se accedió a la aplicación bWAPP descargada, más específicamente al siguiente reto:
http://localhost:12000/insecure_direct_object_ref_2.php
En la misma, probamos la aplicación y, luego de ver que efectivamente es una página que simula la compra de entradas al cine, procedimos a inspeccionar el sitio.



Al hacerlo, se notó que dentro del formulario disponible para realizar la compra de los tickets, además del atributo “ticket_quantity” que nos permite indicar la cantidad de tickets solicitada, existe un campo oculto de entrada llamado “**ticket price**” con el valor “**value=15**”



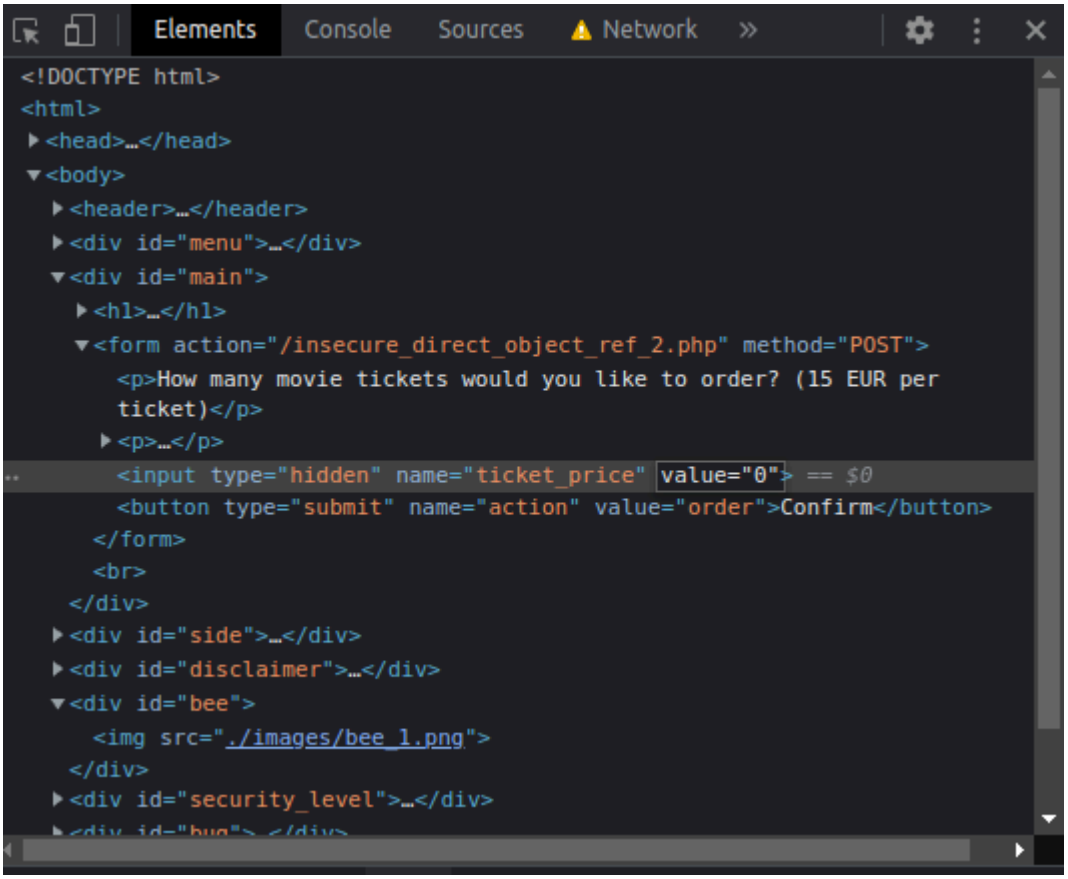
Esto último quiere decir que el precio de cada ticket no se encuentra en la base de datos accedida por el backend, si no que se encuentra “hardcodeado” en el frontend. De ese modo, se modificó el mismo cambiando el valor de 15 a 0 para obtener los tickets de manera gratuita.
La modificación de esta última se puede realizar utilizando herramientas como **burp proxy** recomendada por la cátedra, la cual intercepta las peticiones realizadas antes de ser enviadas al servidor destino para ser modificadas y reenviadas a destino. En nuestro caso, utilizamos la **modificación directa sobre el html**, obteniendo el resultado visto a continuación:


Elements
Console
Sources
Network
<!DOCTYPE html>
<html>
<head>…</head>
<body>
<header>…</header>
…

<h1>…</h1>
<form action="/insecure_direct_object_ref_2.php" method="POST">
How many movie tickets would you like to order? (15 EUR per ticket)
…
<input type="hidden" name="ticket_price" value="15">
<button type="submit" name="action" value="order">Confirm</button>
</form>

…
…

<img src="./images/bee_1.png">
 == $0
…

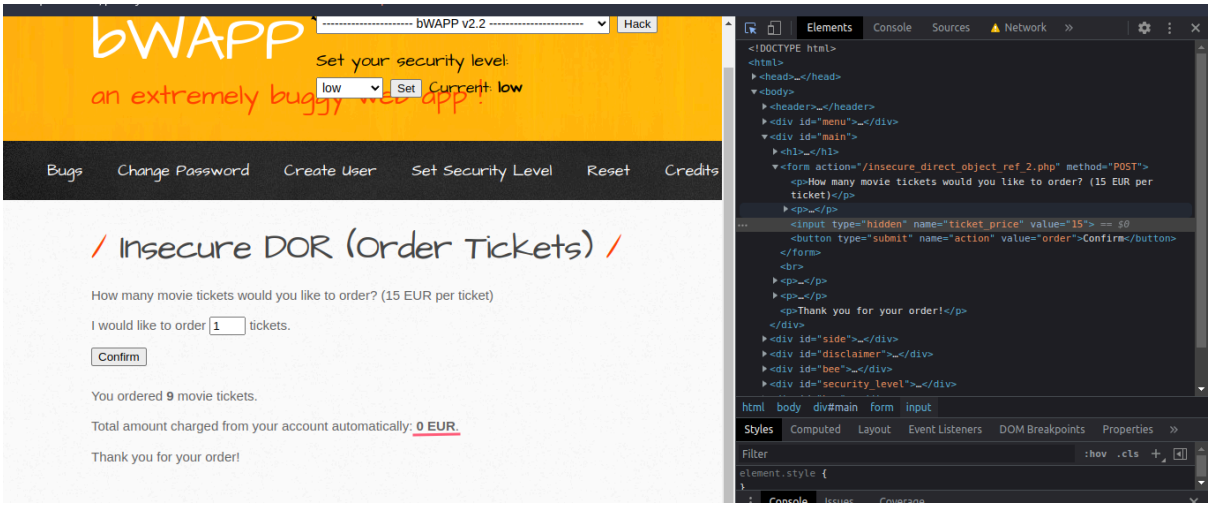

Elements
Console
Sources
Network
<!DOCTYPE html>
<html>
<head>…</head>
<body>
<header>…</header>
…

<h1>…</h1>
<form action="/insecure_direct_object_ref_2.php" method="POST">
How many movie tickets would you like to order? (15 EUR per ticket)
…
<input type="hidden" name="ticket_price" value="0"> == $0
<button type="submit" name="action" value="order">Confirm</button>
</form>

…
…

<img src="./images/bee_1.png">

…



Cómo se podría evitar este tipo de vulnerabilidad

El problema principal es que los datos como los precios de los productos **no deberían poder ser modificados por cualquier usuario**. Se recomienda en estos casos que la interacción con el valor del precio se realice internamente en el backend solicitando el precio a la base de datos de la misma, y que al usuario final solo se le permita ver el precio.

Por otra parte, no se validan los datos que envía el usuario. Otra forma de evitar esta vulnerabilidad sería añadiendo una **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.

Ejercicio 3 - Insecure DOR (Change Secret)

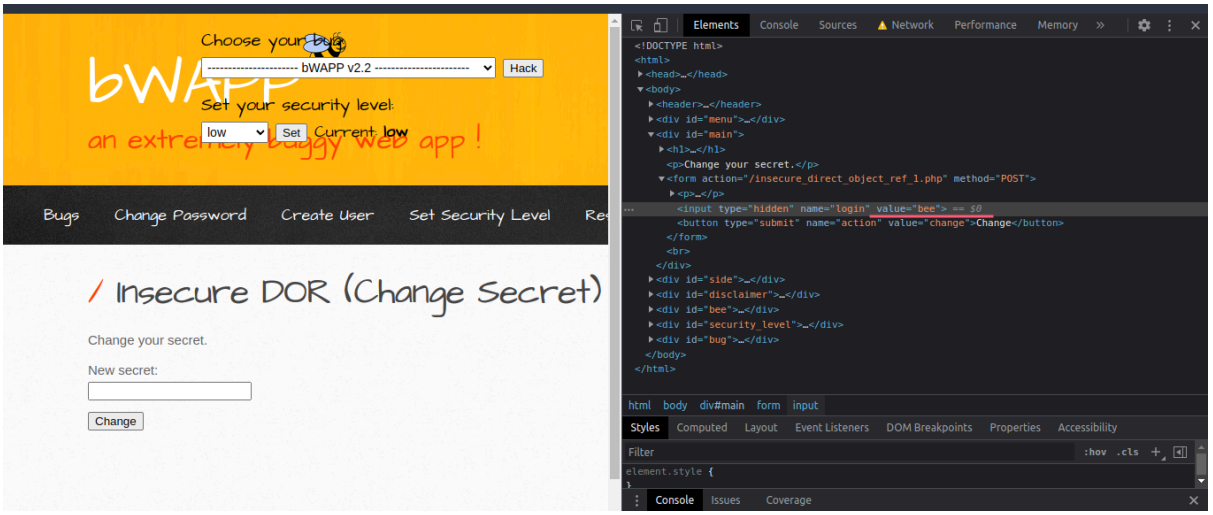
- Reto bWAPP.
- Esta página permite cambiar el "Secret" de tu usuario.
- **Objetivo**: cambiar el "Secret" de otro usuario que no sea el logueado. Verificar que se cambió correctamente el secret logueando con el usuario víctima, accediendo a http://localhost:12000/insecure_crypt_storage_1.php y viendo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

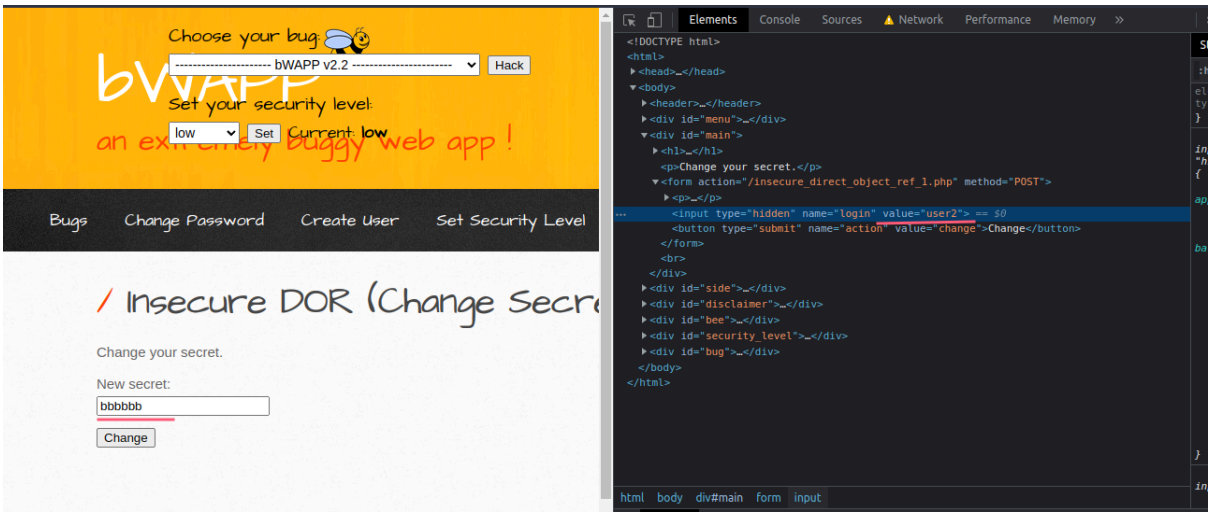
Cómo explotamos la vulnerabilidad

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

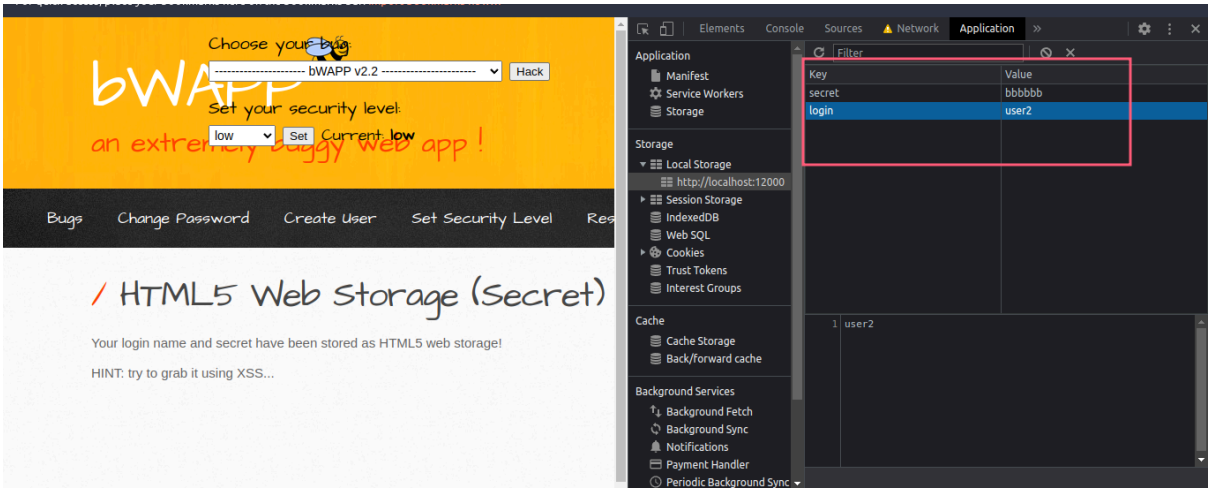
- Usuario: user2
- Contraseña: bug
- Email: NUEVO@GMAIL.COM
- Secret: aaaaaa
- 

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).

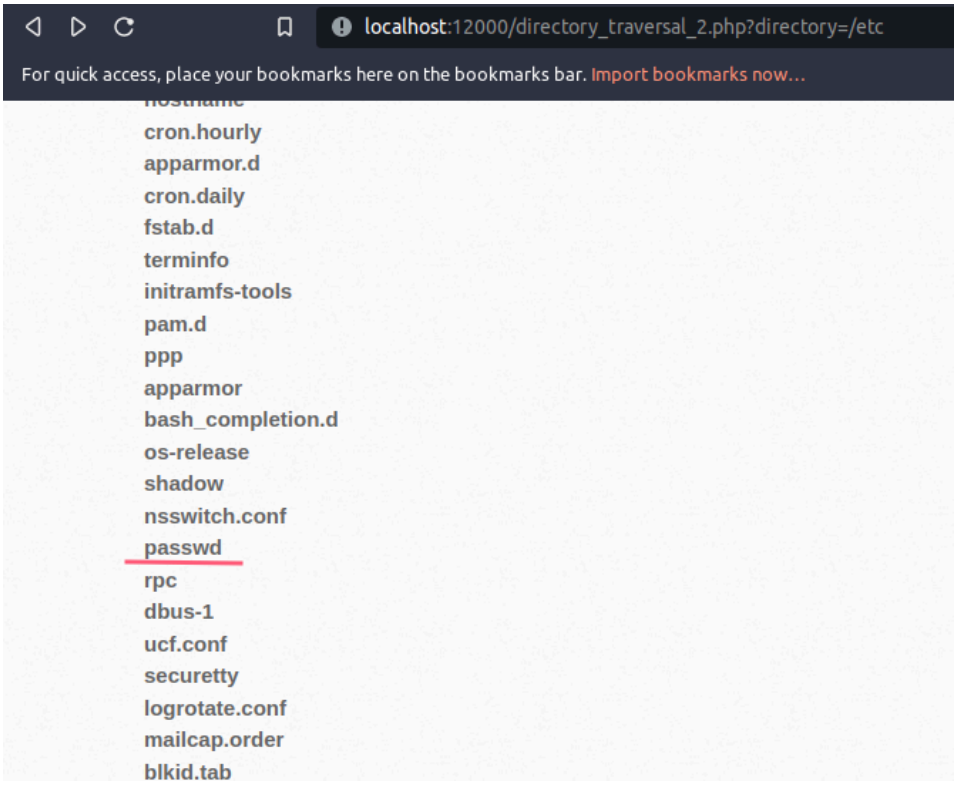
Cómo explotamos la vulnerabilidad

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

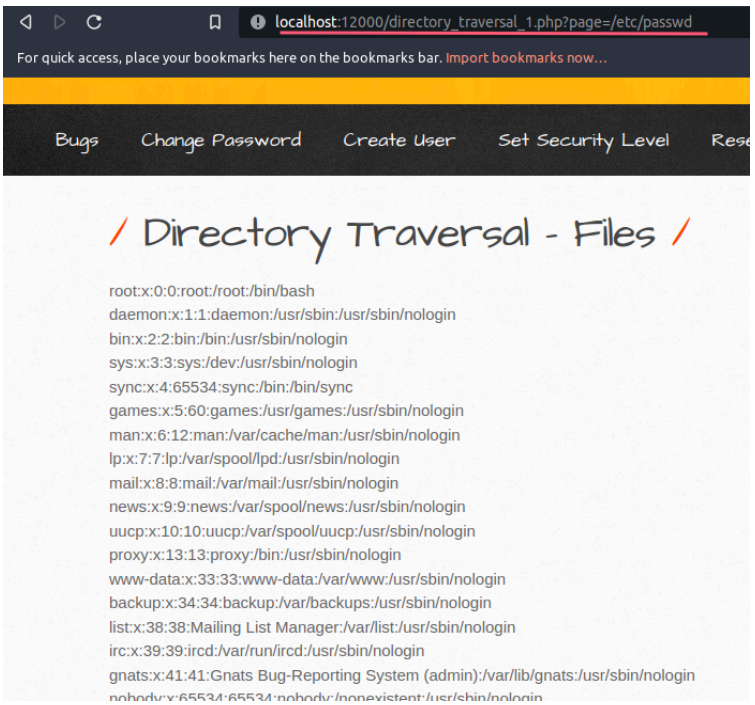
http://localhost:12000/directory_traversal_1.php?page=message.txt
Este enlace permite mostrar el contenido de un archivo de la computadora que corre en el servidor, el cual se le pasa a través del argumento “page” el nombre del archivo.

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=
Se podría armar una lista con argumentos válidos como message.txt, de tal manera que solo dicho valor sea válido.

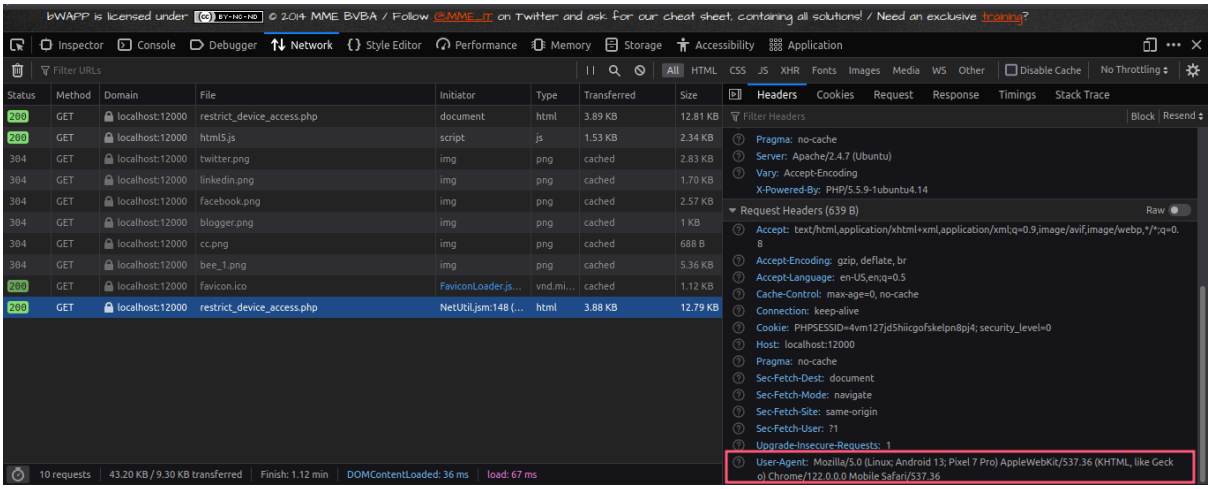
Lo mismo sería para http://localhost:12000/directory_traversal_2.php?directory= se limita valores que puede llegar a tener como argumento el parámetro “directory”.

Ejercicio 5 - Restrict Device Access

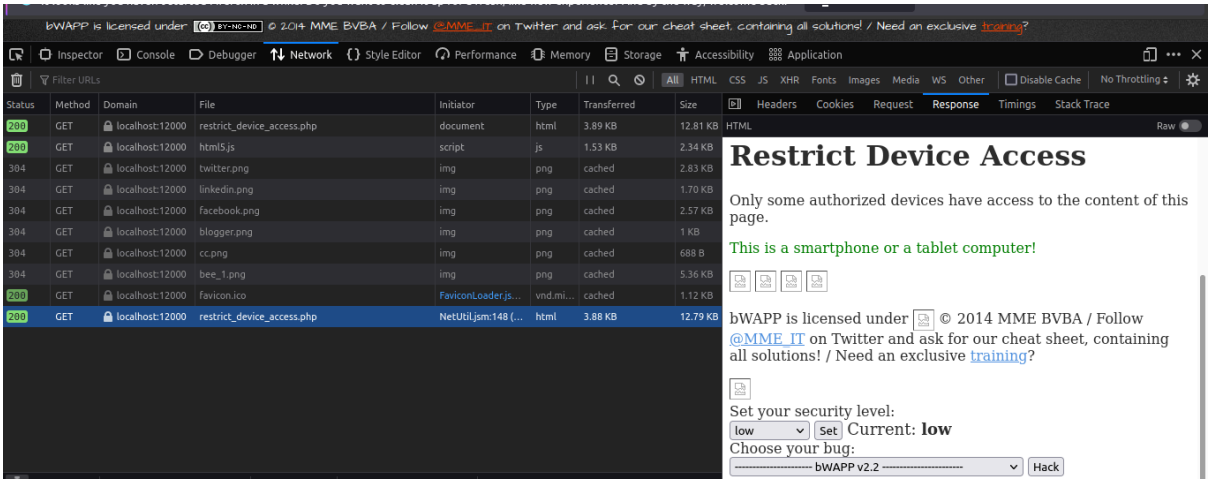
- Reto bWAPP.
- Esta página solo muestra su contenido cuando se la accede con un dispositivo móvil.
- **Objetivo**: acceder sin usar un dispositivo móvil. (Ayuda: pueden utilizar Mozilla Firefox y ver la “respuesta”).

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.