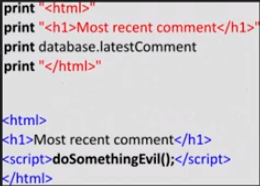
qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm

|  |
| --- |
| Seguridad PHP  05/01/2019  Yeison Y Jose María |

* 1. Filtrado de formularios.
  + *a. Ataques XSS.*
  + *b. Filtrado de datos.*
  + *c. Escape de datos.*
* 2. Encriptado de contraseñas.
  + *a. Seguridad web.*
  + *b. Encriptación de contraseñas PHP.*
* 3. Inyección de SQL.
  + *a. Ejemplos inyección SQL.*
  + *b. Ataques SQL Injection.*
* 4. Ataques en las sesiones.
  + *a. Seguridad sesiones PHP.*
* 5. Seguridad en la subida de archivos.
  + *a. Seguridad subida archivos.*

1. ***Filtrado de formularios.***

Primero vamos a explicar que es XSS o Cross-site Script, esto es básicamente inyección de código java script en aplicaciones web, un ejemplo básico sería el siguiente.

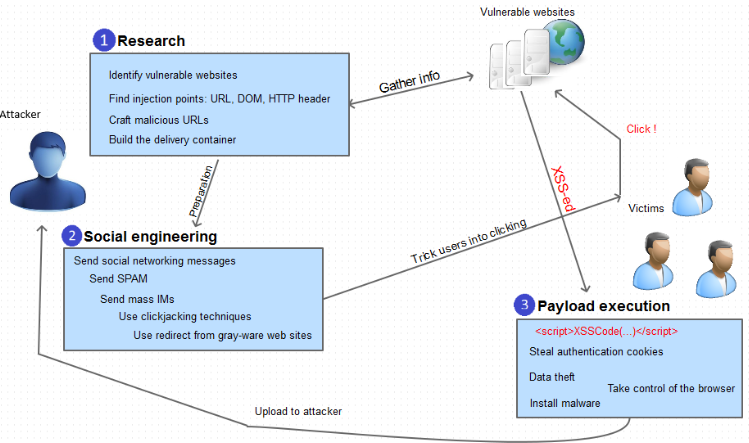


Podemos observar que tenemos varios print (código rojo) que básicamente imprimen código HTML y el “database.latestComment” se refiere a que nos imprima el último comentario que haya en base de datos.

Si ese último comentario tiene código y no validamos ni la entrada del comentario ni como modificar el comentario en la salida, podría ocurrir que alguien dejase como comentario un script (código azul) y que al imprimir ese HTML se ejecute el código Jasa Script.

***Estos tipos de ataques se agrupan en dos categorías:***

***Non-persistent XSS o reflected XSS***, no almacenan el código malicioso en el servidor sino que lo pasan y presentan directamente a la víctima. El ataque se lanza desde una fuente externa, mediante email o un sitio de terceros.



En esta etapa, el atacante busca sitios web vulnerables que se pueden usar para llevar a cabo el ataque. Se realiza una verificación visual para determinar si la entrada del usuario se usa en la página HTML de respuesta, como se muestra en los siguientes ejemplos:

Sitios web que tienen funcionalidad de búsqueda y muestran el término buscado en la página HTML devueltos con los resultados

Sitios web con funcionalidad de inicio de sesión, que muestran el nombre de usuario registrado en la página HTML devuelta

Sitios web que muestran información codificada en los encabezados HTTP, como el tipo de navegador y la versión

Sitios web que utilizan valores de parámetros DOM, como: “document.URL”.

Una vez que un sitio web se identifica como potencialmente vulnerable, los atacantes intentan insertar código de script en las áreas relevantes y verificar si el script se devuelve en su forma original (y se ejecuta). Este proceso puede ser manual o automático, según el sitio web / aplicación web y los posibles puntos de inyección encontrados.

El código malicioso (JavaScript) se puede entregar en una URL en un formato codificado, solicitud HTTP POST, video Flash o imagen, por lo que no es obvio para la víctima o el navegador de la víctima.

Ejemplos de entrega de código malicioso

***Punto de inyección potencial: URL***

Ejemplo:

*www.mywebsite.com/search.asp?term=my+search+term*

Ejemplo malicioso:

*www.mywebsite.com/search.asp?term= <script> MaliciousFunction (...) </script>*

***Punto de inyección potencial: DOM***

Ejemplo:

*www.mywebsite.com/logon.asp?user=Mary*

Ejemplo malicioso:

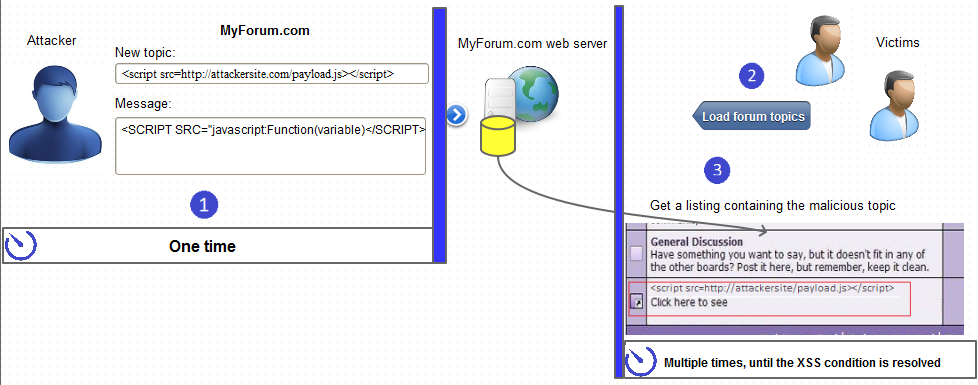
*www.mywebsite.com/logon.asp?user= <script> MaliciousFunction (...) </script>*

***Defensa contra el no persistente XSS***

***del lado del servidor***

La mejor manera de evitar las secuencias de comandos entre sitios es asegurarse de que la aplicación web no haga uso de la información del usuario en las páginas HTML de devolución, sin validarla primero. La validación implica la verificación de la entrada del usuario para determinar si la entrada es válida, de acuerdo con su propósito. En caso de que las funciones de validación encuentren etiquetas de secuencia de comandos, ya sea en texto sin formato o codificadas, deben desinfectar la entrada antes de pasarla al HTML de respuesta y asegurarse de que la secuencia de comandos sea inofensiva. La fuerza de la limpieza depende de la capacidad de las funciones de validación para identificar scripts en la entrada del usuario.

A diferencia del XSS no persistente, ***el XSS persistente*** no requiere una fase de ingeniería social, ya que las víctimas de este ataque no necesitan ser atraídas para hacer clic en un enlace creado. Sin embargo, al explotar las vulnerabilidades persistentes de XSS, los atacantes intentarán que cada vez más víctimas visiten la página web vulnerable, por lo que es muy probable que aún envíen mensajes de SPAM o los promocionen en sitios web de redes sociales.



***Consecuencias***

Las consecuencias son enormes, ya que el ataque permite la ejecución de código arbitrario, generalmente con privilegios elevados: la mayoría de los usuarios domésticos todavía usan la cuenta de "administrador" predeterminada y, aunque los últimos sistemas operativos de Windows cuentan con control de acceso de usuario y políticas de navegador reforzadas, generalmente están desactivados Con el fin de mejorar la experiencia del usuario.

***Objetivos típicos de los ataques persistentes XSS:***

Robo de cookies, robo de datos.

***Defensa contra XSS persistente / XSS almacenado***

***Del lado del servidor:***

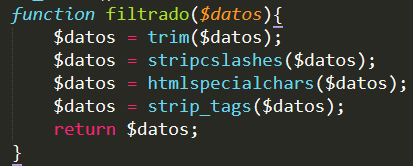
La mejor manera de prevenir el XSS persistente es asegurarse de que todas las entradas de los usuarios estén correctamente desinfectadas antes de que se almacenen permanentemente en el servidor web, y como segunda línea de defensa, asegurarse de que el contenido estático presentado a los usuarios también esté desinfectado. Como los scripts maliciosos se pueden codificar de varias maneras, los analizadores de desinfección deben tener en cuenta la codificación, así como varias formas de inyectar código, al buscar cargas útiles en el contenido para ser almacenadas / servidas nuevamente.

Las aplicaciones web pueden mantenerse libres de XSS mediante la realización de pruebas de evaluación con regularidad utilizando un escáner de vulnerabilidades web que detecta vulnerabilidades de scripts entre sitios mientras le proporciona detalles sobre cómo solucionarlos.

En cualquier formulario de cualquier aplicación siempre hay que validar los datos que introducen los usuarios. Una validación simple puede consistir en asegurarse de que se rellena un campo, se puede hacer fácilmente con empty().



Para solo comprobar que el campo del nombre o el apellido no lleven caracteres especiales o numéricos podemos escribir el siguiente código:



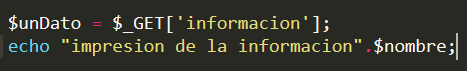
La extensión de filtrado de PHP permite validar o sanitizar los datos, especialmente cuando vienen de fuentes externas (como usuarios). La extensión se divide entre filtros de validación, para comprobar que los datos cumplen ciertos requisitos, y en filtros de saneamiento, que limpiará los datos de forma que se eliminen los caracteres no deseados.

Las banderas u opciones de filtrado sirven para modificar el comportamiento en el saneamiento o la validación según las necesidades. Por ejemplo, si se pasa FILTER\_FLAG\_PATH\_REQUIRED en una URL, se exige que se precise una ruta concreta: ***/sección*** en <http://www.ejemplo.com/seccion>.

La mayoría de aplicaciones producen output de algún tipo, ya sea en formato HTML, un archivo XML o un JSON feed, y para cualquier formato de output los datos de salida han de ser escapados.

Para entender el output escaping es necesario saber cómo funciona HTML. Los comandos marcan las secciones del contenido (cabeceras, scripts, negritas...) o insertan caracteres especiales, imágenes o scripts. Para separar el contenido y los comandos, HTML reserva los caracteres <, > y &. Las etiquetas HTML están envueltas con los caracteres < y >, por lo que no se pueden usar para nada más.

Este tipo de vulnerabilidad viene en conjunto con los ataques XSS ya que gracias a esta vulnerabilidad se pueden ver fallos como el siguiente código.



Este código no está ni filtrado y el input no se ha escapado, esto deriva a la posibilidad de que alguien pueda insertar un script en la URL y enviarla a alguien, por ejemplo: http://ejemplo.es/jasonPortal.php?informacion=

Este ejemplo da consecuencia al robo de datos por medio de cookies, etc.

Para evitar este fallo hay un método básico que es el que nombramos anteriormente arriba. “htmlSpecialChars()”.

1. ***Encriptado de contraseñas.***

Primero hablar de SSL/SSH, es un protocolo que protege los datos mientras viajan por el cable.

Los atacantes pueden obtener acceso directo a nuestra base de datos eludiendo el servidor web, para ello una forma sencilla para evitar este problema es crear primero un paquete de cifrado propio y utilizarlo en los scripts de PHP. Hay muchas extensiones de PHP que pueden ser de ayuda para esto, tales como “Mcrypt” y “Mhash”, cubriendo así una amplia variedad de algoritmos de cifrado.

Ahora hablaremos del Hashing:

Funciones básicas del hashing para PHP:

***md5***

Ejemplo 🡪 ***string md5 (string $str [, bool $raw\_output = false ])***

Calcula un hash con el algoritmo md5. Si se establece \_$rawoutput como true se devolverá en raw binario con una longitud de 16. De normal devuelve un hash de 32 caracteres hexadecimal.

***sha1***

Ejemplo 🡪***string sha1 (string $str [, bool $raw\_output = false ])***

Calcula un hash con el algoritmo sha1. Si se establece \_$rawoutput como true se devolverá en raw binario con una longitud de 20. De normal devuelve un hash de 40 caracteres hexadecimal.

***hash***

Ejemplo 🡪***string hash ( string $algo, string $data [, bool $raw\_output = false ]***

La función toma primero el algoritmo que se desea emplear, $algo, y después el string que se desea encriptar, $data. El algoritmo puede ser md5, sha128, sha256...

En caso de datos que deban estar realmente ocultos, si no fuera necesaria su representación real, (es decir, que no sean mostrados), quizás convenga utilizar algoritmos hash. El ejemplo más típico del uso del hash es a la hora de almacenar el hash criptográfico de una contraseña en una base de datos, en lugar de almacenar la contraseña en sí.

Una función de PHP es password\_hash(),se emplea para usar un hash con una cadena dada utilizando el algoritmo más fuerte actualmente disponible, mientras que la función password\_verify() comprueba si la contraseña dada coincide con el hash almacenado en la base de datos.

1. ***Inyección de SQL.***

***¿Qué es?***

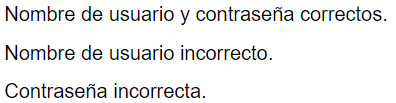
La inyección directa de comandos SQL es una técnica donde un atacante crea o altera comandos SQL existentes para exponer datos ocultos, sobrescribir los valiosos, o peor aún, ejecutar comandos peligrosos a nivel de sistema en el equipo que hospeda la base de datos.

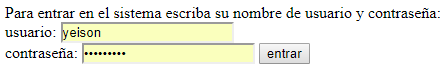
***¿Cómo se logra?***

A través de la práctica de tomar la entrada del usuario y combinarla con parámetros estáticos para elaborar una consulta SQL. También debido a la falta de validación en la entrada de datos y a la conexión a la base de datos con privilegios de superusuario o de alguien con privilegios para crear usuarios, el atacante podría crear un superusuario en la base de datos.

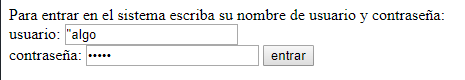
Ejemplos:

***Cuando el usuario escriba un nombre de usuario y contraseña, la aplicación responde uno de estos tres mensajes:***

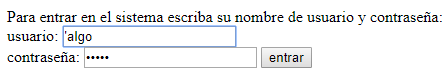
**



***Para comprobar si la aplicación incluye los datos enviados por el usuario sin ningún tratamiento previo, podemos enviar una comilla (simple o doble) como dato.***



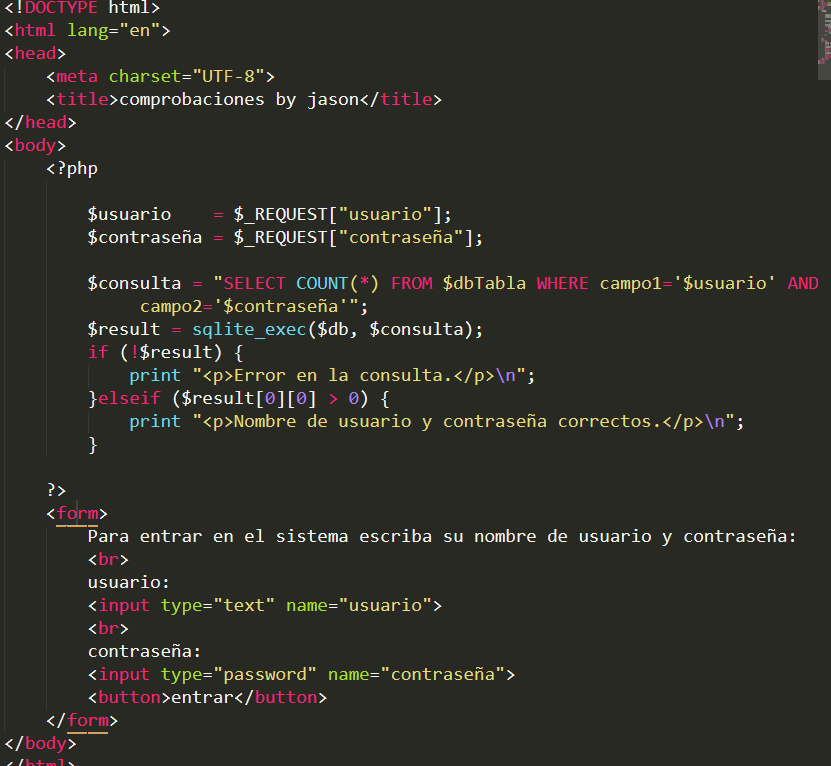




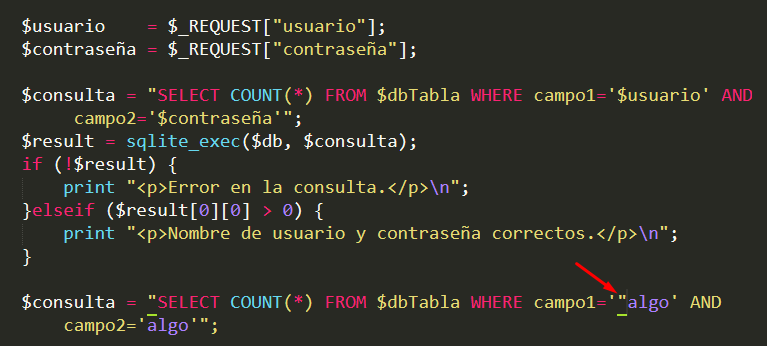


***Este último mensaje ("Error en la consulta") nos hace saber que no se tratan los datos y que además las consultas están delimitadas por dobles comillas.***

***Probablemente el código sea algo como esto:***

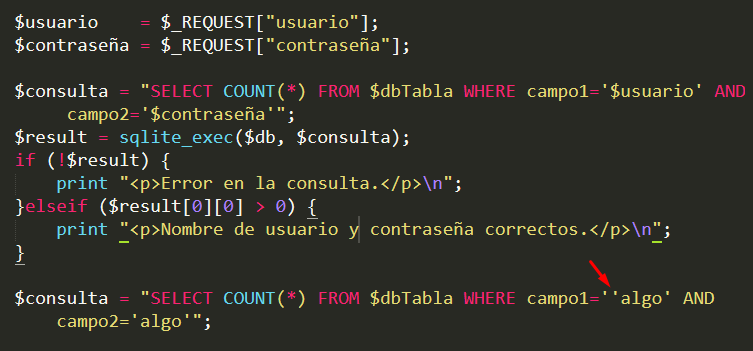
******

***Al escribir una comilla doble al principio del nombre de usuario, la consulta se convierte en:***

******

***Esta consulta es correcta (no contiene errores de sintaxis) y cuando se ejecuta, la base de datos simplemente devuelve 0.***

***Sin embargo, al escribir una comilla simple al principio del nombre de usuario, la consulta se convierte en:***

******

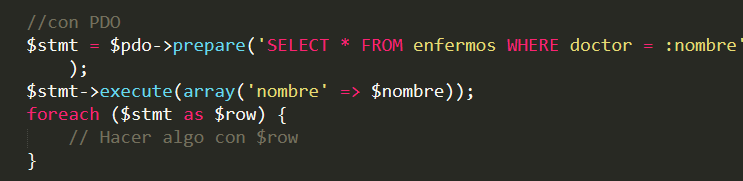
***Esta consulta no es correcta (contiene un error de sintaxis por culpa de la comilla dentro de las comillas de la segunda línea y cuando se ejecuta, la base de datos da error.***

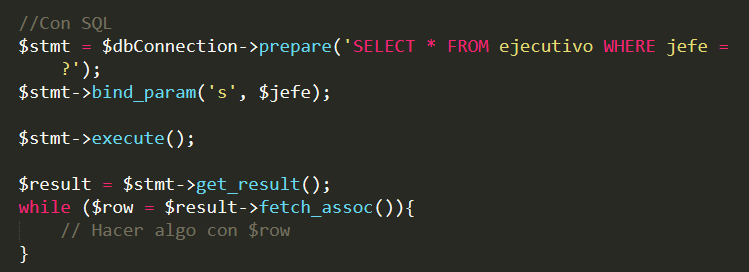
***Ahora que sabemos que la consulta está delimitada por comillas dobles podemos escribir unos datos que modificarán la consulta y harán que la aplicación crea que hemos introducido datos de un usuario registrado.***

***Aquí ya no puedo dar más ejemplos porque no tengo la base de datos creada…***

***¿Cómo evitarlo?***

Emplear sentencias preparadas con variables vinculadas. Son proporcionadas por PDO, MySQLi y otras bibliotecas.





Si la expresión espera una entrada numérica, considerar verificar los datos con la función ctype\_digit(), o silenciosamente cambiar su tipo utilizando settype(), o emplee su representación numérica por medio de sprintf().

1. ***Ataques en las sesiones.***

Primero explicar que es el robo de sesión o Hijacking:

Se refiere a que un individuo (atacante) consigue el identificador de sesión entre una página web y un usuario, de forma que puede hacerse pasar por este y acceder a su cuenta en esa página web.

*Tipos:*

***ATAQUE POR FUERZA BRUTA***

El ataque por fuerza bruta significa probar identificadores aleatoriamente hasta encontrar uno que esté siendo usado. Es como intentar abrir una caja fuerte sin saber la combinación, poniendo números al azar.

Como en el caso de la caja fuerte, cuantos más números tenga la combinación (en este caso el identificador de sesión), más difícil será de adivinar. También ayuda el hecho de que el número o identificador sea aleatorio, y no algo que se pueda predecir. El sistema de identificadores de sesión de PHP es aceptable en este sentido.

***ROBO POR SNIFFING***

Este tipo de ataque se da cuando el atacante tiene un programa de sniffing en la red del usuario y puede interceptar el tráfico destinado al mismo, incluido su identificador de sesión.

La única forma de prevenir estos ataques es utilizando cifrado HTTPS en toda la página web.

***PROPAGACIÓN EN URL***

Si el identificador de sesión se propaga utilizando la URL en lugar de las cookies, cualquier atacante puede robarlo desde muchos sitios:

* Un enlace que el propio usuario ponga en un lugar público. Los usuarios típicos no saben para que sirve ese identificador y no le dan importancia.
* El historial del navegador.
* El referrer, que es un encabezado que envían muchos navegadores a las páginas web en el que les indican la URL de la que vienen.

La forma de prevenir esto es no utilizar la URL para el identificador de sesión; utilizar únicamente las cookies. En PHP esto se consigue con la instrucción:

Ejemplo🡪ini\_set('session.use\_only\_cookies', 1);

***ROBO EN SERVIDOR COMPARTIDO***

Si tenemos nuestra página web alojada en un servidor compartido, los archivos físicos de las sesiones se guardan, por defecto, en un directorio común para todas las páginas web del servidor. Esto quiere decir que todas las personas que tengan su página web en ese mismo servidor, tienen acceso a todos los archivos de sesiones. Dado que el nombre de los archivos es "sess\_" más el identificador de sesión, cualquier atacante tendrá una lista de identificadores de sesión válidos con sólo leer la lista de archivos del directorio común.

Esta vulnerabilidad se puede combatir de dos formas:

Usando la función session\_save\_path para guardar los archivos de sesión de la página web en un directorio dentro de su cuenta al que sólo pueda acceder PHP (ya sea por estar fuera del directorio web o con un .htaccess con la instrucción deny from all). Este método no es demasiado fiable, ya que el resto de usuarios seguirá pudiendo leer en ese directorio, sólo tienen que averiguar su localización.

Guardando las sesiones en base de datos en lugar de en archivos. Esto se consigue fácilmente usando la función session\_set\_save\_handler. Esta solución es la más segura ya que la página web será la única que tendrá acceso a la base de datos y, por tanto, a las sesiones.

***ROBO POR CROSS-SITE SCRIPTING***

Si la página web es vulnerable a XSS el atacante puede insertar un código javascript que envíe las cookies de un usuario a su cuenta.

Este tipo de ataque se puede prevenir (además de evitando los ataques XSS) haciendo que las cookies de sesión tengan el atributo HttpOnly, que evita que puedan ser manejadas por javascript en la mayoría de navegadores. En PHP esto se consigue con la instrucción:

$\_SESSION[‘fingerprint’] = md5($\_SERVER[‘HTTP\_USER\_AGENT’]);

*Como Evitarlo:*

* Limitar tiempo de inactividad: eliminar la sesión si está cierto tiempo sin ser usada (de 5 a 30 minutos, según el nivel de seguridad de la página web).
* Cambiar el identificador de sesión: cada cierto tiempo o después de cada acción, cambiar el identificador de la sesión por otro distinto y eliminar la sesión antigua.
* Sistema de logout: dar a los usuarios una forma de salir de su cuenta y destruir la sesión.
* Verificación doble: usar un segundo método para intentar reconocer al usuario de la sesión. Esto puede hacerse guardando cabeceras como HTTP\_USER\_AGENT (navegador del usuario) o REMOTE\_ADDR (IP del usuario) cuando se crea la sesión, de esta forma:

1. ***Seguridad en la subida de archivos.***

Puntos a tratar en la seguridad con la subida de archivos:

***Seguridad en la subida de Archivos***

Lo primero es evitar que el usuario pueda subir cualquier archivo para ello se pueden seguir las siguientes reglas:

* El archivo subido nunca debe ser accesible inmediatamente por el cliente.
* Generar nombres aleatorios a los archivos subidos.
* Filtrar los archivos para no permitir archivos PHP.
* Subir los archivos a carpetas fuera del directorio de publicación.
* Utilizar is\_uploaded\_file para verificar que el archivo se ha subido.
* Utilizar move\_uploaded\_file para copiar el archivo al directorio final.

***Seguridad en la subida de Imágenes***

En el caso de imágenes, estas generalmente deben ser publicador inmediatamente, entonces hay que hacer varias verificaciones adicionales:

* Verificar el Content-Type del archivo para asegurarse que el archivo es una imagen.
* Verificar si el archivo es imagen con la función getimagesize
* Limitar el archivo a un máximo de tamaño.

***Implementado Clase PHP para subida de archivos***

Siguiendo las reglas mencionadas se puede crear una pequeña clase que facilita el subir archivos e incluye la verificación por tipo de archivo, extensión, tamaño de archivo. Entonces creamos un archivo llamado upload.php donde incluimos el siguiente código por ejemplo:

