1 Cómo evitar ataques XSS y CSRF con PHP

1.1 Sumario

2 Introducción a XSS

Un ataque XSS (Cross Site Scripting) consiste en introducir código HTML o Javascript en las cajas de texto de formularios, y si éstos no controlan los datos que se introducen, pueden hacer que ese código se ejecute en la web.

Por ejemplo, si un usuario escribe el siguiente código en una caja de texto y no se controlan los datos introducidos, la web se redireccionaría hacia donde le hemos indicado.

```
<script>window.location = "http://www.google.com";</script>
```

2.1 Cómo evitar ataques XSS

De forma básica, se puede prevenir la inclusión de código por medio de la función de PHP "strip_tags": http://php.net/manual/es/function.strip-tags.php

```
echo strip_tags('<script>window.location = "http://www.google.com";</script>');

// resultado: window.location = "http://www.google.com";
```

Si lo que queremos es un control más exhaustivo de lo que queremos filtrar, podremos hacer uso de la clase de PHP **Input Filter**, la cuál nos permite hacer el filtrado de datos de forma sencilla, empleando los métodos proporcionados.

Para descargarse esa clase lo podéis hacer (previo registro) desde:
http://www.phpclasses.org/package/2189-PHP-Filter-out-unwanted-PHP-Javascript-HTML-tags-.html

O también la podéis descargar desde aquí: Archivo:Clase-inputfilter.zip

Para usar la clase tendremos que insertarla en nuestro documento:

```
require_once 'class.inputfilter.php';
$filtro = new InputFilter();
$nombre = $filtro->process($_POST['nombre']);
```

Si lo que queremos es filtrar todos los datos recibidos por el método POST, pasaremos el array $_POST.

Por ejemplo:

```
require_once 'class.inputfilter.php';
$filtro = new InputFilter();
$_POST = $filtro->process($_POST);
```

También se le puede indicar a la clase, si queremos que no filtre algunas etiquetas para que no las elimine del contenido, para ello haremos:

```
require_once 'class.inputfilter.php';
$filtro = new InputFilter(array('strong','bold'));
$nombre = $filtro->process($_POST['nombre']);
```

Incluso podremos también indicar atributos que no queremos que filtre. Para ello lo que hacemos es definir un segundo array con los atributos que no queremos que filtre.

```
require_once 'class.inputfilter.php';
$filtro = new InputFilter(array('a'), array('href'));
$comentarios = $filtro->process($_POST['comentarios']);
```

Información de atributos a usar durante la instanciación del objeto InputFilter.

```
1° (array etiquetas):    Opcional (desde la versión 1.2.0)
2° (array atributos):    Opcional
3° (métodos etiquetas):  0 = eliminar TODO EXCEPTO estas etiquetas (por defecto)
                         1 = eliminar SOLO estas etiquetas
4° (métodos atributos):  0 = eliminar TODO EXCEPTO estos atributos (por defecto)
                         1 = eliminar SOLO estos atributos
5° (xss autostrip):      1 = eliminar todos los problemas con etiquetas (por defecto)
                         0 = desactivar esta característica

Ejemplo: $mifiltro = new InputFilter($tags, $attributes, 0, 0);
```

2.2 Otras páginas de referencia

- http://www.funcion13.com/2012/10/15/entiende-ataques-xss-aprende-prevenir-php/
- http://www.programacion.com/articulo/como_evitar_ataques_xss_con_php_462
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

3 CSRF - Cross Site Request Forgery

Obtenido de la Fuente: http://www.funcion13.com/2012/08/21/preven-falsificacion-peticion-sitios-cruzados-csrf/

El CSRF o XSRF (del inglés Cross-site Request Forgery o falsificación de petición en sitios cruzados) es un tipo de exploit malicioso de un sitio web en el que se consigue que se ejecuten comandos no autorizados, por un usuario autorizado del sitio web, sin que éste lo pretenda. Esta vulnerabilidad es conocida también por otros nombres como XSRF, enlace hostil, ataque de un click, cabalgamiento de sesión, y ataque automático.

Esto puede ocurrir cuando, por ejemplo, el usuario ha iniciado sesión en una de sus webs y hace click sobre un enlace aparentemente inofensivo. Por detrás, la información de su perfil es actualizada y se cambia su correo electrónico con el de un atacante. El atacante ahora puede solicitar un reseteo de contraseña sin que nadie se entere y ha robado la cuenta con éxito.

3.1 Cómo funciona un ataque CSRF

Para entender bien cómo funciona un ataque CSRF, mejor (como dice el gran Goyo Jiménez) no lo cuento, lo hago. Para ilustrar un ataque, vamos a crear un sencillo ejemplo que te permite cerrar una sesión activa. Pero antes de cerrar una sesión activa, vamos a necesitar una página de login (login.php), un script que se encargue de iniciar y cerrar la sesión (procesar.php) y finalmente nuestro ataque de ejemplo (gatito.html). Los gatitos son inocentes, ¿verdad?

Código de la página **login.php**:

```
<?php
session_start();
?>
<html>
 <body>
<?php
if (isset($_SESSION["usuario"])) {
    echo "<p>¡Bienvenido, " . $_SESSION["usuario"] . "!<br>";
    echo '<a href="procesar.php?accion=logout">Salir</a></p>';
}
else {
?>
  <form action="procesar.php?accion=login" method="post">
   <p>El usuario es: admin</p>
   <input type="text" name="usuario" size="20">
   <p>Contraseña: prueba</p>
   <input type="password" name="pass" size="20">
   <input type="submit" value="Entrar">
  </form>
```

```
<?php
}
?>
 </body>
</html>
```

El script inicializa primero los datos de sesión. Luego revisa si la variable $_SESSION["usuario"] está establecida. De ser así, nos muestra un mensaje de bienvenida y un enlace para cerrar la sesión. Si no, muestra el formulario de inicio de sesión.

Vamos ahora con el procesado en **procesar.php**:

```
<?php
session_start();

switch($_GET["accion"]) {
    case "login":
        // Si viene por POST
        if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $usuario = (isset($_POST["usuario"]) &&
                $_POST["usuario"]) ? $_POST["usuario"] : null;
            $pass = (isset($_POST["pass"])) ? $_POST["pass"] : null;
        $salt = '#l0S.p4Nd4s.Pr0T3g3N.3sT4.cL4v3#';

            if (isset($usuario, $pass) && (crypt($usuario . $pass, $salt) ==
                crypt("adminprueba", $salt))) {
                $_SESSION["usuario"] = $_POST["usuario"];
            }
        }
        break;

    case "logout":
        $_SESSION = array();
        session_destroy();
        break;
}

header("Location: login.php");
?>
```

Este script comienza también iniciando los datos de sesión, y luego revisa si hay alguna acción sobre la que trabajar. Revisamos que los datos vengan rellenos con operadores ternarios y hacemos una sencilla comprobación con la función crypt(). Si esto fuera un caso real, lo comprobaríamos seguramente con los datos que tengamos almacenados en base de datos. Finalmente, el usuario es redirigido a la página login.php al final del script.

Finalmente, lleguemos a la parte de los **gatitos.html**:

```
<html>
 <body>
  <p>Ohh mira un gatito... ¡Qué bonito!</p>
  <img src="procesar.php?accion=logout" style="display: none;">
 </body>
</html>
```

Si visitas la página login.php e inicias sesión, y mientras has iniciado sesión visitas esa página, quedarás automáticamente deslogado incluso sin que hayas hecho nada. El navegador envía una petición al servidor para acceder al script de procesar.php, esperando que sea una imagen.

Nuestro script inicial no tiene forma de diferenciar entre una petición válida, iniciada por el click de un usuario sobre el enlace de ?Salir?, y una petición diseñada para hacer el mal.

Lo peor, es que el archivo gatito.html puede estar en un servidor completamente diferente al de tu aplicación y funcionar correctamente porque la página atacante hace una petición en tu nombre usando la sesión que abriste antes. Incluso si la web está en una red interna funcionará porque la petición se envía desde tu IP como si tú hubieras hecho la petición por tu cuenta.

Además, si permites a tus usuarios enlazar a imágenes como avatares de perfil sin escapar apropiadamente, tendrás el enemigo en casa.

Vale vale? cerrar la sesión de alguien no es muy grave. Pero bien podríamos aprovechar cualquier acción a nuestro alcance, o incluso un formulario oculto que se autoenvíe cuando se cargue la página.

¿Ves ahora la seriedad de un ataque CSRF? Veamos pues cómo podemos resolverlo.

3.2 Cómo protegerse de ataques CSRF

Para poder asegurarte de que una acción la está llevando acabo un usuario en vez de un script malicioso, tenemos que asociar un identificador único para que podamos verificarlo en cada acción.

Vamos a modificar el archivo login.php para que prevenir el ataque:

```
<?php
session_start();
?>
<html>
 <body>
<?php
if (isset($_SESSION["usuario"])) {
    // Generando ID único, podríamos usar la id de sesión
    $_SESSION["token"] = md5(uniqid(mt_rand(), true));

    echo "<p>¡Bienvenido, " . $_SESSION["usuario"] . "!<br>";
    echo '<a href="procesar.php?accion=logout&token=' . $_SESSION["token"] . '">Salir</a></p>';
}
else {
?>
  <form action="procesar.php?accion=login" method="post">
   <p>El usuario es: admin</p>
   <input type="text" name="usuario" size="20">
   <p>Contraseña: prueba</p>
   <input type="password" name="pass" size="20">
   <input type="submit" value="Entrar">
   <input type="hidden" value="<?php echo $_SESSION['token']; ?>">
  </form>
<?php
}
?>
 </body>
</html>
```

Luego, para verificarlo en **procesar.php**:

```
<?php
session_start();
if (isset($_GET["token"]) && $_GET["token"] == $_SESSION["token"]) {
    switch($_GET["accion"]) {
        case "login":
            // Si viene por POST
            if ($_SERVER["REQUEST_METHOD"] == "POST") {
                $usuario = (isset($_POST["usuario"]) &&
                    $_POST["usuario"]) ? $_POST["usuario"] : null;
                $pass = (isset($_POST["pass"])) ? $_POST["pass"] : null;
            $salt = '#l0S.p4Nd4s.Pr0T3g3N.3sT4.cL4v3#';

                if (isset($usuario, $pass) && (crypt($usuario . $pass, $salt) ==
                    crypt("adminprueba", $salt))) {
                    $_SESSION["usuario"] = $_POST["usuario"];
                }
            }
            break;

        case "logout":
            $_SESSION = array();
            session_destroy();
            break;
    }
}
header("Location: login.php");
?>
```

Con estas sencillas modificaciones, gatito.html ya no funcionará porque el atacante tendría que adivinar un token adicional que es aleatorio. Como véis, hemos protegido también el formulario para que el token se envíe junto al resto de datos.

Cabe destacar, que esta medida puede ser aplicada de forma similar a las peticiones AJAX.

3.3 Otras referencias

http://www.cgisecurity.com/csrf-faq.html

--Veiga (discusión) 20:26 13 may 2013 (CEST)