## *Cross script scripting (XSS)*

### Introduction

Parler de la "same origin policy" http://en.wikipedia.org/wiki/Same_origin_policy .

### Comment cela fonctionne

Supposons que l'url suivante: http://www.monsite.com/?search=paris retourne ce fragment de HTML:

```
<p>Votre recherche pour 'paris' n'a pas renvoyé de résultat.</p>
```

Si aucune validation ou aucun échappement de caractères spéciaux n'est fait, un attaquant pourrait construire un lien comme celui-ci: http://www.monsite.com/?search=paris+%3Cscript%3Escript_malicieux()%3C/script%3E

Lorsque la victime cliquera sur le lien, le code suivant sera interprété par son navigateur:

```
<p>Votre recherche pour '<script>script_malicieux()</script>' n'a pas
renvoyé de résultat.</p>
```

Puisque la page vient du domaine www.monsite.com, le code exécuté a accès à l'environnement du navigateur pour ce domaine et notemment aux cookies. A partir de là, par exemple, un vol de session est possible.

Notons au passage qu'il n'est même pas nécessaire que la victime clique sur le lien, puisqu'une balise <iframe> peut pointer sur ce lien, et le code sera exécuté de façon totalement trasparente.

// Rappeler le fonctionnement de iframe, donner un exemple de iframe sans bordure et de 0px (invisible).

Exemple

// Mettre un exemple de code JavaScript malicieux et un code serveur qui réceptionne les cookies.

### Comment éviter le problème (changer ce titre …)

// Mettre un exemple de code qui purge les entrées (par exemple une sous classe de httpservlet???).

## *Accès aux fichiers*

### Méthode rapide: via l'URL

Voir: Insecure web app

#### *Code*

Voir: insecure/source/asc/ReportServlet.java

```
String reportName = request.getPathInfo();
InputStream is = getServletContext().getResourceAsStream("WEB-INF/reports/"
      +reportName.trim());
```

Changer le code et mettre quelque chose de plus simple et de plus grossier.

#### *Exemple*

http://localhost:8080/insecure/resports/reportxxx.pdf

### *Attaque possible*

Téléchargement d'un autre fichier en entrant son chemin.

http://localhost:8080/insecure/resports/fichierxxx.pdf

http://localhost:8080/insecure/resports/../../../config.xxx (à tester)

## Méthode naïve: choix limité via une liste dans un formulaire

### *Code*

```
<form name="form" method="post" action="report.php">
    <select name="report">
        <option value="report1.pdf" selected="selected">Report1
        <option value="report2.pdf">Report2
        <option value="report3.doc">Report3
    </select>
    <p>
        <input type="submit" name="Submit" value="Submit">
    </p>
</form>
```

### *Attaque possible*

Interception de la requête POST avant l'envoi au serveur, et modification des données pour accéder à n'importe quel fichier.

### *Exemple*

`….&report=report1.pdf&..... ==> ….&report=fichierxxx&.....`
Mettre des copies d'écran avec un proxy local (e.g. Paros).

### *Conclusion*

Ne pas se fier à la validation côté client.

## Méthodes plus sécurisées

### *Epuration des paramètres reçus*

Il s'agit par exemple d'éliminer les caractères interdits dans les noms de fichier, ce qui interdira la navigation dans le système de fichiers.

### Code

```
reportName = reportName.replaceAll("[\\/...]", "");
```

### *Utilisation d'une liste de fichiers autorisés*

Avoir un fichier contenant la liste des fichiers autorisés est une solution élégante, puisqu'il suffira d'ajouter ou de retirer des entrées à la volée pour autoriser ou non des fichiers.

### Code

```
boolean found = false;

// Charge les fichiers autorisés dans une liste de String.
```

```
List<String> reports = loadReports("reports.xml");
for (String report : reports) {
    if (report.equals(reportName) {
        found = true;
        break;
    }
}

if (found == false) {
    throw new Exception (…);
}

// Code d'envoi du fichier.
```