Webh4ck partie II

© Stéphane Rodriguez, 6 octobre 2000

La première partie du document a permis de modéliser les paramètres mis en jeu lors de la traversée d'un site. Mais on n'a pas insisté sur le fait que des sites ne respectent pas vraiment les standards. Cet article permet de détailler un cas réel et d'envisager des solutions server-side et client-side.

Les sites web utilisent des lignes de commande qui sont parfois très éloignées du type CGI, c'est-à-dire un signe & suivi par un ensemble de paires variable = valeur. Un site hébergé par une solution LotusNotes Domino en est un bon exemple. Les pages hébergées par un script serveur type Java Server Pages est aussi particulier.

SUR CADREMPLOI :
http://tf1.cadremploi.fr/Cadremploi/cadremploi.nsf/mViewTemplateVoffreswebTF1?OpenForm&Count=0&&Que
((FIELD+FonctionSaisie1+=+0205*)+OR+(FIELD+FonctionSaisie2+=+0205*))+AND+
((FIELD+Region+=+0101*)+OR+
NOT(FIELD+Region+=+010*))

SUR KELJOB:
http://www.keljob.com/recherche.jsp;jsessionid=aaazw-3vw_JmzP8HjDNZ

Cela veut dire que, dans l'absolu, la ligne de commande CGI doit être également modélisée par une règle de production. Dans le cas simple, respecté par 95% des sites web, on peut décrire une ligne de commande CGI de la façon suivante :

```
CGI      = ( '&' pair* ) | null
pair     = variable '=' value
variable = letter (letter | digit)*
value    = (letter | digit | special_code)*
```

Une nouvelle modélisation de la ligne CGI impacte le modèle de base de données à partir du moment où nous trouvons autre chose que des affectations de valeur à des variables. A partir du moment où la ligne de commande est riche de sens, elle force une implémentation particulière. Fort heureusement, la majorité des sites respectent CGI purement et simplement.

Pour bien comprendre la problématique mise en jeu, il suffit de prendre un cas réel.

Un cas réel : traversée du site www.ifrance.com pour atteindre directement la page des stats.

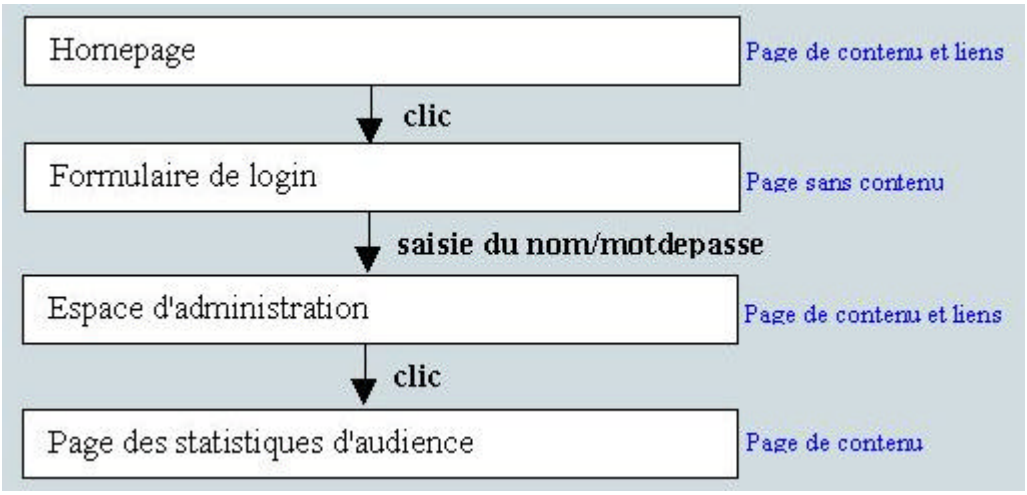
Le site ifrance est un site d'hébergement gratuit proposant des pages personnelles accessibles par des URLs du type http://www.ifrance.com/votrenom. Chaque webmaster dispose d'un accès administratif à l'espace

personnel. L'administration permet notamment de se renseigner sur l'audience du site.

En parcourant le site "à la main", on se connecte d'abord sur la homepage, dont les fonctionnalités présentées sont prévues autant pour le grand public que pour celui qui possède une page personnelle. Puis on clique sur le lien SITE. Ce qui affiche une page avec un formulaire de login. Le login et le mot de passe donnés permettent d'accéder à l'espace privatif. Une fois arrivé sur l'espace privatif, il faut cliquer sur le lien AUDIENCE pour accéder à la page des statistiques.

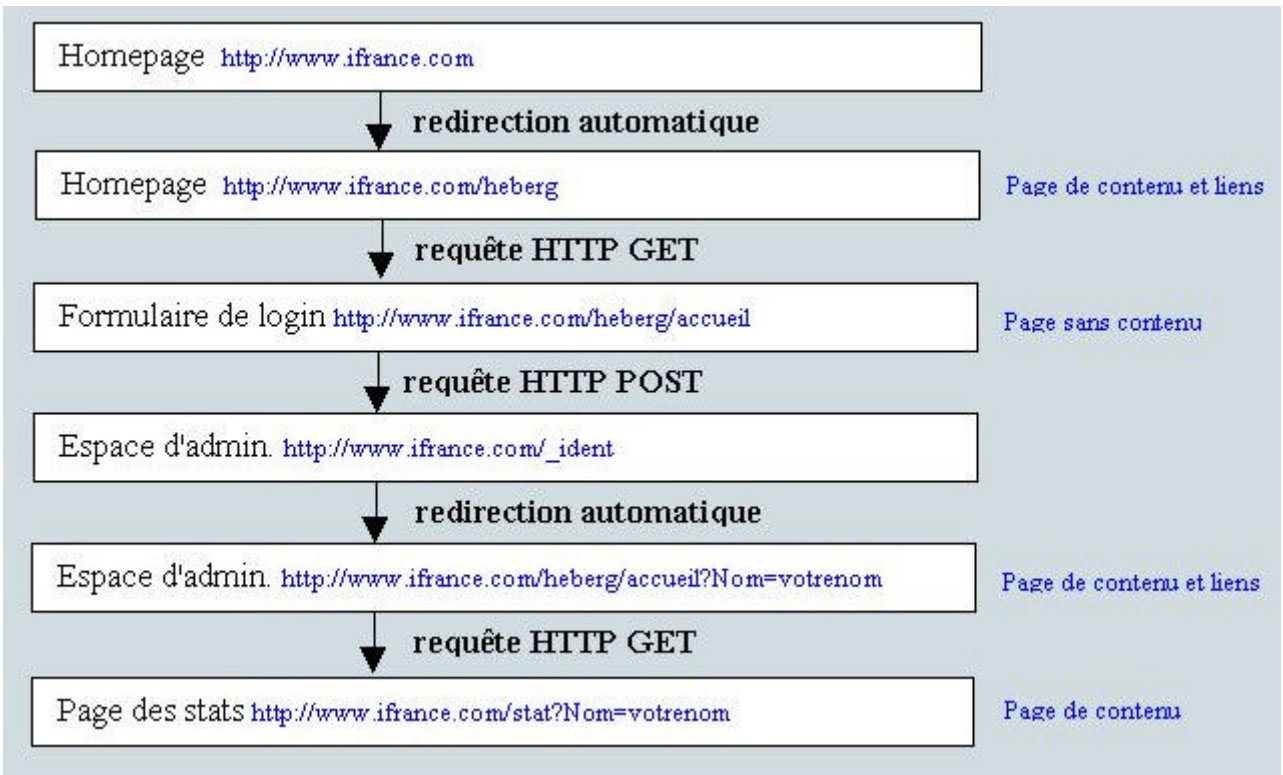
Schématiquement,

Vu de l'utilisateur :



Ceci est une description assez limpide d'un certain nombre de mécanismes invisibles imbriqués les uns dans les autres. On n'imaginerait certainement pas ce que les webmasters derrière ifrance ont imaginé pour sécuriser l'accès au site : redirections multiples, cookies multiples, identifiants de session, tout y passe.

Vu du développeur,



Les choses se compliquent un petit peu. Et on peut remarquer une chose, 6 différentes URLs sont mises en jeu et les chemins d'accès semblent sortir d'un petit peu n'importe où. On peut en déduire que le site ifrance n'en est probablement pas à sa première version et qu'un certai nombre de fonctionnalités ont été conservées tel quel. Comme dans tout développement logiciel, on garde un code historique et on tisse du nouveau code ailleurs, jusqu'au jour où il faut tout reprendre à zéro tant l'architecture est devenue ingérable.

On sera capable d'ajouter au schéma "Vu du développeur" l'ensemble des entêtes et des cookies mis en jeu, après avoir étudié le cheminement exact entre ces pages.

Techniquement, les mécanismes cachés de traversée du site

Cette étude est vérifiable à un instant t, jusqu'à ce que les administrateurs du site décident de changer les données. Le changement de données se fait en moyenne tous les deux mois.

Il y a fondamentalement quatre pages : la homepage, la page comportant le formulaire de login, la page d'administration, et la page d'audience.

L'accès à la homepage se fait en tapant http://www.ifrance.com, mais une redirection immédiate, due à une entête Location dans le retour de requête, fait charger la page http://www.ifrance.com/heberg. Cette page n'est pas indiquée en tant que ressource .HTML, ceci est probablement dû à une configuration serveur dédiée. On peut considérer que tout se passe comme si la page accédée était en fait http://www.ifrance.com/heberg/default.html.

Lorsqu'on clique sur le lien SITE, cela provoque une nouvelle requête vers http://www.ifrance.com/heberg/accueil. Si ce n'est pas la première fois qu'on accède à cette page, un cookie a été déposé pour cette page : OPS_REFERER=%3F; OPS_ORIGINE.

Les cookies

On rappelle que le principe des cookies est de sauvegarder dans un fichier du disque dur un ensemble de paires VARIABLE = VALEUR associé à une page donnée d'un site. Chaque paire est séparée par un point-virgule. Chaque paire est en fait un cookie, et peut comporter des dates d'expiration ainsi qu'un champ PATH et une mention sécurité. Le champ PATH permet de remonter au serveur les cookies de cette page et de ceux dont le chemin d'accès est concaténé à ce qui est écrit dans PATH : en d'autres termes, les cookies peuvent être partagés entre toutes les pages d'un site. Une valeur par défaut de PATH est /, ce qui veut dire que tous les cookies sont partagés, ou encore que tous les nouveaux cookies déposés seront tous renvoyés vers le serveur.

Sur le disque dur, tout se passe comme s'il y a avait une base de données mise à disposition du navigateur web (IE et Netscape n'utilisent pas la même base de données). Dans cette base de données, une table très simple relie une page web identifiée par une URL à un ensemble de cookies. Cet ensemble de cookies est ramené à un seul grâce à une grosse ligne de cookies séparés par des points-virgules. Ce qui donne :

http://www.pageweb1 cookie1;

http://www.pageweb2 cookie2; cookie3;

et ainsi de suite.

Exemples :

- un cookie simple : mycookie=54;
- un cookie avec une propriété PATH : mycookie=54; path=/;
- deux cookies : mycookieA=54; path=/; mycookieB=55; path=/;

A chaque fois que le navigateur web crée une nouvelle requête vers une page web, par un clic souris ou une redirection, il vérifie toujours si la page cible est connue dans cette base de données et transmet AUTOMATIQUEMENT le cas échéant les cookies associés, par une entête **Cookie:**

Inversement, lorsqu'un navigateur web reçoit une page web avec un ou plusieurs entêtes **set-cookie**, l'opération est inversée et la base de données est alimentée par un ou plusieurs cookies. Si pour une page web un nom de cookie existe déjà alors sa valeur est changée, alors que si le nom de cookie n'existe pas encore, le cookie est ajouté en tant que tel dans la liste des cookies connus pour cette page.

Dans cette logique très simple la propriété PATH sème un petit peu le trouble car elle décloisonne les cookies d'une page.

Par ailleurs, dans une page web on a accès fondamentalement aux cookies en lecture et en écriture, à l'aide de l'objet **document.cookie**.

Ceci, allié à la puissance de manipulation des objets HTML permet à une page HTML de connaître les cookies déposés par une autre page HTML. Voyons l'exemple suivant : une page index.html crée deux cadres, un qui sert d'espion et l'autre qui sert de support au contenu. Nous écrivons en HTML :

index.html

```
      <frameset rows="180,*">
       <frame src="readcookie.html" name="_1">
       <frame src="content.html" name="_2">
      </frameset>
```

readcookie.html

```
Cookie :
      <script language="Javascript">
      var framecontent = window.parent.frames["_2"];
      document.writeln (framecontent.document.cookie);
      </script>
```

Quant à content.html, c'est un contenu quelconque.

Ce faisant on accède au formulaire de login. Le formulaire est décrit par les champs :

NAME	TYPE	VALUE
Nom	TEXT	
Pwd	PASSWORD	
Verif	IMAGE	/_icono/b_red_valider.gif
Produit	HIDDEN	SITE
Back	HIDDEN	http://www.ifrance.com/heberg/accueil.login
ProfilId	HIDDEN	
Page	HIDDEN	AUTHENTIFICATION

L'ACTION associée au formulaire est de type POST, encodée en **application/x-www-form-urlencoded** (ce qui transforme les caractères spéciaux comme l'espace en +), et pointe vers la page **_ident** du site.

L'envoi de ces données de formulaire permettent fondamentalement de générer un identifiant, **AuthId**. Cet identifiant permet d'accéder à la partie privative du site associée au compte de l'utilisateur par l'intermédiaire d'un cookie décrit ci-après.

Remarque concernant le champ de type IMAGE : la transmission des données du formulaire n'envoit évidemment pas sous forme binaire l'image désignée. Ce qui est envoyé, c'est en réalité les coordonnées locales (x,y) où l'utilisateur a cliqué pour valider son formulaire. Ces coordonnées sont variables puisqu'au prochain login, il y a très peu de chances de cliquer exactement au même endroit. Le champ Verif n'est donc pas vraiment un champ Vérification contrairement à ce que l'on pourrait croire.

L'analyseur de trame IP scanne l'envoi des données du formulaire et indique que le navigateur transmet les données suivantes :

POST http://www.ifrance.com/_ident HTTP/1.0
Cookie : OPS_REFERER=%3F; OPS_ORIGINE
EOL EOL
Nom=votrenom&Pwd=votremotdepasse&Produit=SITE&Back=http%3A%2F%2Fwww.ifrance.com%2Fheberg%2Faccueil.login&ProfilId=&Page=AUTHENTIFICATION&Verif.x=26&Verif.y=27

On remarquera que les champs du formulaire sont passés dans la section CONTENT de la requête et non dans les entêtes. En effet, on aurait pu croire que les champs serait passés de la façon suivante Nom: votrenom EOL Pwd: votremotdepasse EOL Produit: SITE .Il n'en est rien, c'est ce qui différencie une requête POST d'une requête GET. Ces détails sont décrits dans la norme HTTP.

Pour ce qui est de l'identifiant AuthId, calculé par le serveur à partir de ces données, il faut tester si la valeur obtenue est en réalité quasi aléatoire auquel cas n'importe quelle valeur permet d'accéder au site privatif. C'est un test important et parfois on est étonné par la facilité avec laquelle on peut pénétrer dans des sites.

Si toutefois cet identifiant est calculé par le serveur pour la session courante, et donc gardé en mémoire, il sera nécessaire de transmettre correctement la valeur, après l'avoir obtenue. Un corollaire est qu'il faut absolument passer par l'étape de transmission des données de formulaire.

La question qui se pose à ce niveau est : est-il possible de simuler en HTML la transmission de ces données de formulaire ? est-il possible en sus de récupérer l'identifiant obtenu de façon à préparer l'accès à la page d'audience ?

Le serveur web répond en générant un nouveau cookie et en redirigeant la sortie vers une nouvelle page, la page d'administration : http://www.ifrance.com/heberg/accueil?Nom=votrenom. Les cookies associés à cette page sont envoyés par le serveur et sont :

OPS_SITE_CUR_SITE=votrenom; path=/;
AuthId=4588431_WRNOH; path=/;

(la valeur 4588431_WRNOH est prise comme valeur d'exemple)

La page d'administration s'affiche et l'ensemble des cookies associés (directs ou partagés) sont :

OPS_REFERER=%3F;
OPS_ORIGINE;
OPS_SITE_CUR_SITE=votrenom;
AuthId=4588431_WRNOH;

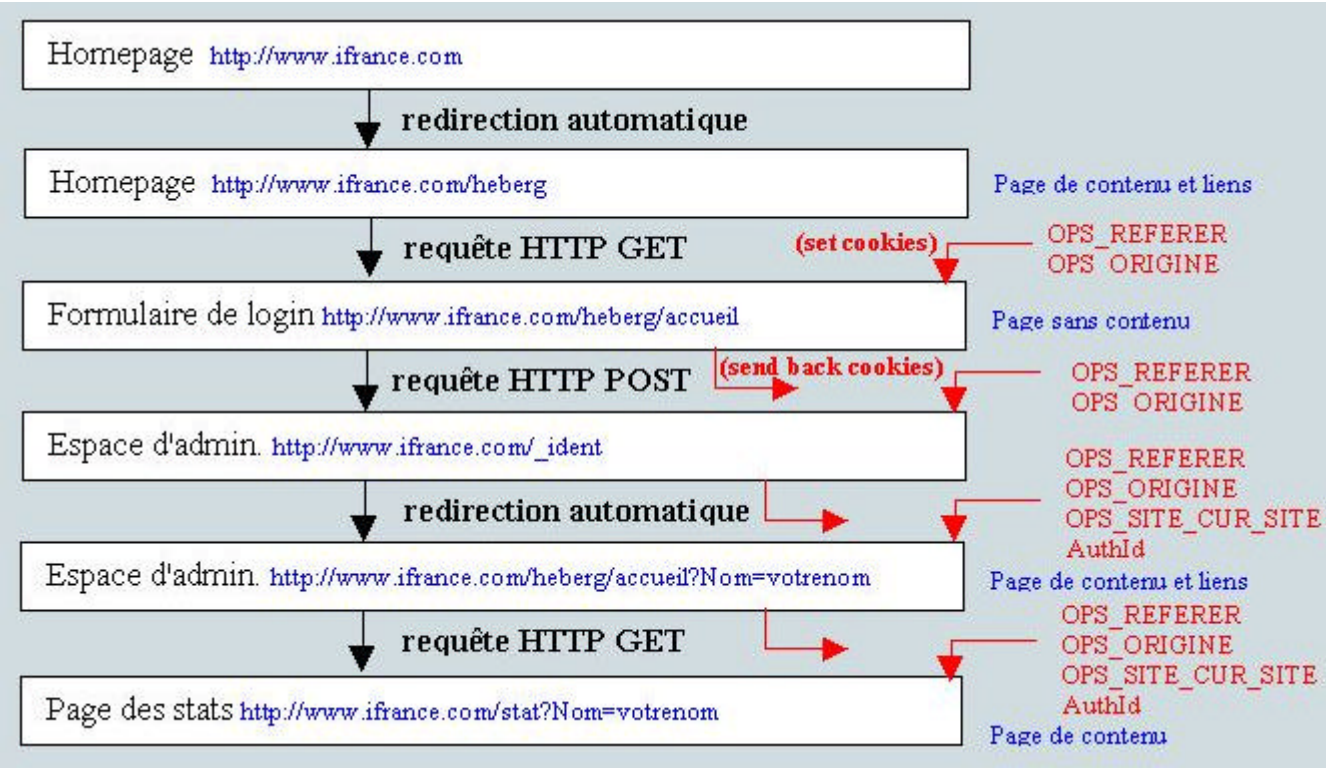
Nous pouvons alors cliquer sur le lien AUDIENCE, ce qui provoque une simple requête vers http://www.ifrance.com/stat?NomSite=votrenom.

Puisque les cookies sont partagés, les cookies cités ci-dessus sont transmis automatiquement par le navigateur, qui lui voit les cookies comme associés à la page en question, en entête de la façon suivante :

Cookie: OPS_REFERER=%3F; OPS_ORIGINE; OPS_SITE_CUR_SITE=votrenom; AuthId=4588431_WRNOH;

Et la page d'audience s'affiche.

Vu du développeur, au total :



Comment simuler la dernière étape par du code ?

Voilà une question simple, qui amène plusieurs solutions.

La première solution est de trouver un moyen de transmettre les cookies par une page HTML toute simple.

Accessoirement, la pratique peut révéler un problème d'accès car une des entêtes, l'entête Referrer, peut nous rendre la tâche difficile. En effet, si le site ifrance.com vérifie la valeur de ce Referrer, il doit logiquement trouver http://www.ifrance.com/_ident. Or, notre page étant stockée en local sur le disque dur, l'entête Referrer sera soit vide, soit reliée à une page locale. Si le serveur web contrôle cette donnée, il peut se rendre compte de la supercherie et refuser l'accès. Il faudra alors sortir une artillerie plus lourde…

La seconde solution consiste à utiliser un proxy capable d'enrichir à la volée les requêtes web à l'aide de divers entêtes, et carrément du code HTML inédit dans le code HTML. Pour obtenir en quelque sorte un trackeur.

La troisième solution consiste à utiliser un serveur intermédiaire. C'est lui qui effectue une requête

intermédiaire, hors du contexte du navigateur web, obtient la valeur AuthId, et est capable de fournir pour le compte du navigateur web une requête vers la page http://www.ifrance.com/stat?NomSite=votrenom.

Solution 1 : code HTML

Solution 1.1 : nous allons supposer dans un premier temps, pour prendre un cas simple, que l'AuthId peut avoir une valeur quelconque, du moment qu'il a une valeur, par exemple 4588431_WRNOH.

Pour accéder à la page audience, il nous suffit de générer une requête vers la page d'audience en transmettant les cookies vus ci-dessus. Mais ces cookies doivent être associés à la page d'audience, et non à la page HTML qui sert de point d'entrée. Cela implique d'utiliser la variable PATH en fournissant la valeur suivante PATH= http://www.ifrance.com/stat?NomSite=votrenom.

Il faut savoir qu'en HTML dynamique nous avons accès via Javascript au Document Object Model (DOM). Ce DOM nous permet d'accéder par l'entremise de l'objet document à **l'objet cookie**, accessible en lecture et en écriture. Par ce biais, il est possible à la fois de collecter et d'affecter un cookie à une page donnée.

Mais attention, en fonction de la valeur de PATH l'accès en lecture ou en écriture peut être interdit.

Le code HTML est le suivant :

```
<script language="Javascript">

   szURL = "http://www.ifrance.com/stat.NomSite=votrenom";
   szPath = "path=" + szURL + "; "

   document.cookie="OPS_REFERER=%3F; " + szPath
                                      + "OPS_ORIGINE; " + szPath
                                      + "OPS_SITE_CUR_SITE=votrenom; " + szPath
                                      + "AuthId=4588431_WRNOH; " + szPath

</script>
<html>
<body>
<a href="http://www.ifrance.com/stat?NomSite=votrenom">Cliquez ici pour accéder directemen
</body>
</html>
```

On peut aussi faire automatiquement :

```
<script language="Javascript">

   szURL = "http://www.ifrance.com/stat.NomSite=votrenom";
   szPath = "path=" + szURL + "; "

   document.cookie="OPS_REFERER=%3F; " + szPath
                                      + "OPS_ORIGINE;" + szPath
                                      + "OPS_SITE_CUR_SITE=votrenom; " + szPath
                                      + "AuthId=4588431_WRNOH; " + szPath

   location.href = szURL;
</script>
```

Solution 1.2 : nous devons construire une valeur valide pour AuthId, une valeur créée à la volée par le serveur web ifrance et maintenue tant que l'utilisateur est sur le site. Cela

suppose de passer par la transmission des données du formulaire. La transmission automatique des données d'un formulaire se fait sans difficulté en HTML mais il faut aussi pouvoir (re)traiter les données renvoyées par le serveur web et générer une nouvelle requête dans la foulée.

Il y a deux parties à traiter. La partie envoi automatique de données de formulaire se traite sans difficulté à l'aide du code suivant :

```
<html>
<body>

<form action="http://www.ifrance.com/_ident" method="post" enctype="application/x-www-form
  <input type="text" name="Nom" value="votrenom"></input>
  <input type="password" name="Pwd" value="votremotdepasse"></input>
  <input type="hidden" name="Produit" value="SITE"></input>
  <input type="hidden" name="Back" value="http://www.ifrance.com/heberg/accueil.login"></i
  <input type="hidden" name="ProfilId" value=""></input>
  <input type="hidden" name="Page" value="AUTHENTIFICATION"></input>
  <input type="text" name="Verif.x" value="26"></input>
  <input type="text" name="Verif.y" value="27"></input>
</form>

<script language="Javascript">
   // automatically executed during HTML parsing (IE + Netscape compatible)
   document.all.forms["myform"].submit;
</script>

</body>
</html>
```

Ceci permet d'obtenir en retour, au sein d'un cookie, la valeur de AuthId. Une fois la valeur obtenue, il suffit d'appliquer la méthode vue en 1.1 pour accéder à la page audience.

L'inconvénient de la solution 1.2 est, alors que c'est elle la plus générale, elle se fait en deux étapes distinctes. Pas d'automatisation, à moins que le navigateur web soit "configuré" pour réagir d'une certaine façon lorsqu'il reçoit un cookie pour la valeur de AuthId. En pratique, cela revient à mettre un gestionnaire d'évènements et à l'attacher au navigateur web. Cette solution n'est pas très bonne car elle est spécifique à un navigateur donné et également spécifique aux entêtes d'une page.

Si l'on désire une méthode automatique, et c'est bien le but du jeu, alors il faut se tourner de préférence vers une solution exploitant un proxy ou un serveur d'application. Ces deux solutions présentent l'avantage d'être indépendentes du navigateur.

Solution 2 : proxy enrichissant le code HTML à la volée

En voyant passer toutes les données de et vers le navigateur web, le composant proxy peut à la fois filtrer des entêtes ou du code HTML. Mieux, il peut insérer du code HTML. Cet ensemble de possibilités permet de faire à peu près tout et n'importe quoi.

< TRAITE PROCHAINEMENT >

Solution 3 : solution basée sur un serveur d'application

< TRAITE PROCHAINEMENT >

Acteurs sur le net : www.ezlogin.com , www.octopus.com, www.yodlee.com