Réseaux

HyperText Transfert Protocol

Sommaire

Généralités

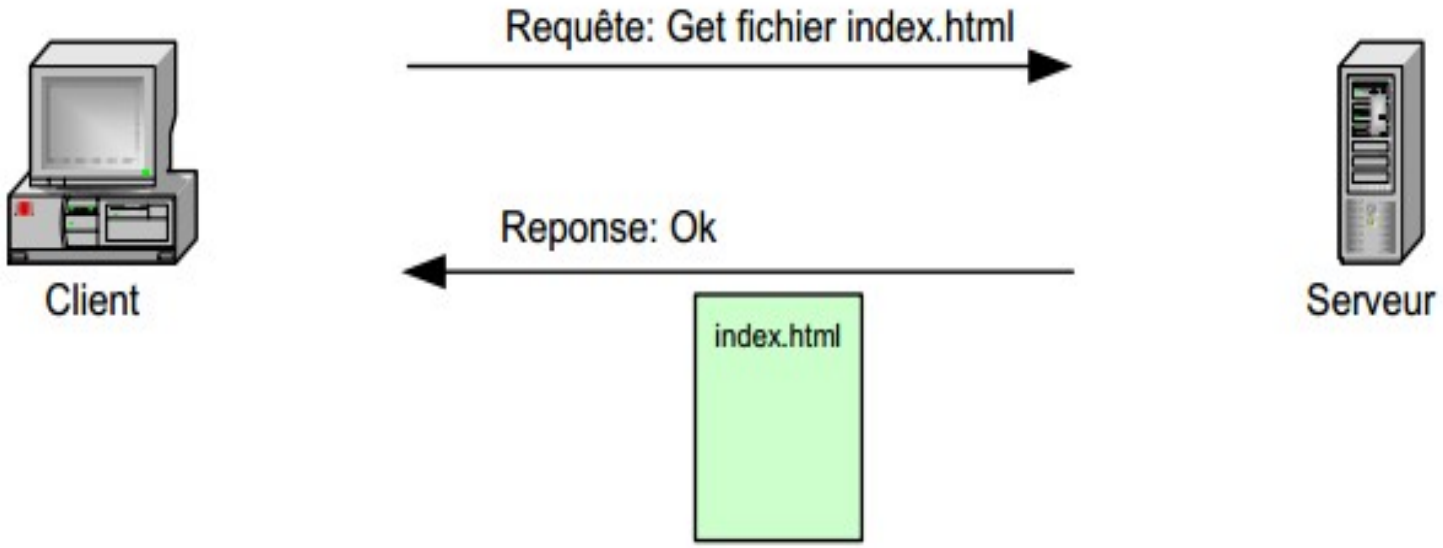
HTTP (Hypertext Transfer Protocol) permet d'accéder aux fichiers situés sur le réseau Internet.

Il est notamment utilisé pour le World Wide Web.

En matière de protocole, HTTP se place au dessus de TCP et fonctionne selon un principe de requête/réponse :

- le client transmet une requête comportant des informations sur le document demandé;
- le serveur renvoie le document si disponible ou, le cas échéant, un message d'erreur.

HTTP 1.0 est un protocole sans connexion et chaque couple requête/réponse est de ce fait indépendant.


Requête: Get fichier index.html
Reponse: Ok
index.html
Client
Serveur

Ce protocole a été créé au CERN au début des années 1990 afin de fournir au Web un protocole de transfert simple.

Deux versions du protocole existent :
- HTTP 1.0 définit en 1996 par la RFC 1945
- HTTP 1.1 définit en 1999 par la RFC 2616

La version 1.1 apporte, entre autre, les améliorations suivantes :
- Cinq nouvelles méthodes;
- Connexions persistantes;
- Digest Authentication.

Les URI

Un URI (Uniform Resource Identifier) est une chaîne de caractères structurée permettant d'identifier de manière unique une ressource dans un espace de nom donné.

Cette ressource peut être désignée soit par:

- un URN (Uniform Resource Name);
- un URL (Uniform Resource Locator);

URN et URL sont des sous-ensembles d'URI.

Une URL permet de localiser une ressource.

Dans le cas du protocole HTTP, une URL permet de localiser une page HTML, un fichier texte, un script cgi, une image...

Le format général d'une URL HTTP est le suivant :

http://<host>:<port>/<path>?<query>#<fragment>

<host> → nom d'hôte ou adresse IP;
<port> → généralement 80;
<path> → permet de désigner le fichier désiré;
<query> → suite de tuples clés/valeurs (séparés par des '&');
<fragment> → permet d'indiquer une position dans la page.

Encodage d'URL

Les caractères ne pouvant être représentés dans une URL comme ';' '/' '?' '&', …) doivent être échappés.

Afin de rendre "escaped" un caractère (par exemple '&'), il faut remplacer ce caractère par son code ASCII codé en hexadécimal (26 pour '&') et le faire précéder par le caractère '%'.

Le caractère '&' devient donc %26 et un espace ' ' devient %20.

Exemples :
http://www.exemple.com
http://www.exemple.com:80/news/search?cle=valeur
http://www.exemple.com/texte%20avec%20espace/exemple.html

Les méthodes, requêtes et réponses

Les méthodes HTTP les plus fréquemment utilisées sont :

- GET;
- POST;
- HEAD.

Cinq autres méthodes sont cependant définies par la version 1.1 du protocole.

Méthodes	1.0	1.1	Description
Get			Permet de demander un document
Post			Permet de transmettre des données (d'un formulaire par exemple) à l'URL spécifiée dans la requête. L'URL désigne en général un script Perl, PHP…
Head			Permet de ne recevoir que les lignes d'en-tête de la réponse, sans le corps du document
Options			Permet au client de connaître les options du serveur utilisables pour obtenir une ressource
Put			Permet de transmettre au serveur un document à enregistrer à l'URL spécifée dans la requête
Delete			Permet d'effacer la ressource spécifiée
Trace			Permet de signaler au serveur qu'il doit renvoyer la requête telle qu'il la reçue
Connect			Permet de se connecter à un *proxy* ayant la possibilité d'effectuer du *tunneling*

La requête transmise par le client au serveur comprend :

- une ligne de requête(request-line) contenant la méthode utilisée, l'URL du service demandé, la version utilisée de HTTP;
- une ou plusieurs **lignes d'en-têtes**, chacune comportant un nom et une valeur.

La requête peut **optionnellement** contenir une entité (contenu).

Celle-ci est notamment utilisée pour transmettre des paramètres avec la méthode POST.

L'entité est transmise après les lignes d'en-têtes, elle est séparée de la dernière en-tête par un double CRLF (carriage return et linefeed)

Voici un exemple d'en-tête:

- - - - -

GET /index.html HTTP/1.1
Host: www.example.com
Accept: */*
Accept-Language: fr
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0)
Gecko/20100101 Firefox/32.0
Connection: Keep-Alive

- - - - -

Les pointillés ne sont pas transmis, ils permettent de visualiser le double CRLF

Dans cet exemple:

- le client demande le document à l'adresse http://www.example.com/index.html;
- il accepte tous les types de document en retour mais préfère les documents en français;
- utilise un navigateur compatible Mozilla 32.0 sur un système WindowsNT 6.1 (Windows 7)
- signale au serveur qu'il faut garder la connexion TCP ouverte à l'issue de la requête (car il a d'autres requêtes à transmettre).

La réponse transmise par le serveur au client comprend :

- une ligne de statut (status-line) contenant la version de HTTP utilisée et un code d'état;
- une ou plusieurs lignes d'en-têtes, chacune comportant un nom et une valeur;
- Le corps du document retourné (les données HTML ou binaires par exemple).

Une réponse ne contient pas obligatoirement un corps comme par exemple quand il s'agit d'une réponse à une requête HEAD.

Dans ce cas, seule la ligne de statut et les en-têtes sont retournés.

```
HTTP/1.1 200 OK
Date: Mon, 15 Dec 2003 23:48:34 GMT
Server: Apache/1.3.27 (Darwin) PHP/4.3.2 mod_perl/1.26
DAV/1.0.3
Cache-Control: max-age=60
Expires: Mon, 15 Dec 2003 23:49:34 GMT
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "26206-5b0-3af1f126"
Accept-Ranges: bytes
Content-Length: 1456
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
...
```

Dans cet exemple:

- le code 200 nous indique que le document demandé a été trouvé;
- pour faciliter la gestion du cache du client, le serveur transmet la date actuelle, la date de dernière modification du document et la date d'expiration.
- Content-Type nous apprend que le document retourné est de type HTML
- ContentLength indique que le corps du document a une longueur de 1456 octets.
- Server renseigne sur le logiciel serveur utilisé. L'envoi d'une telle information n'est pas recommandé d'un point de vue sécuritaire.

En-têtes génériques

Certains en-têtes peuvent se trouver aussi bien dans la requête que dans la réponse:

Champ	Description
Content-length	Longueur en octets des données suivant les en-têtes
Content-type	Type MIME des données qui suivent
Connection	Indique si la connexion TCP doit rester ouverte (*Keep-Alive*) ou être fermée (*close*)

En-têtes de la requête

Les en-têtes suivants n'existent que dans les requêtes HTTP.
Seul l'en-tête Host est obligatoire dans la version 1.1 de HTTP

Champ	Description
Accept	Types MIME que le client accepte
Accept-encoding	Méthodes de compression supportées par le client
Accept-language	Langues préférées par le client (pondérées)
Cookie	Données de *cookie* mémorisées par le client
Host	Hôte virtuel demandé
If-modified-since	Ne retourne le document que si modifié depuis la date indiquée
If-none-match	Ne retourne le document que sil a changé
Referer	URL de la page à partir de laquelle le document est demandé
User-agent	Nom et version du logiciel client

En-têtes de la réponse

Champ	Description
Allowed	Méthodes HTTP autorisées pour cette URI (comme POST)
Content-encoding	Méthode de compression des données qui suivent
Content-language	Langue dans laquelle le document retourné est écrit
Date	Date et heure UTC courante
Expires	Date à laquelle le document expire
Last-modified	Date de dernière modification du document
Location	Adresse du document lors d'une redirection
Etag	Numéro de version du document
Pragma	Données annexes pour le navigateur (par exemple, no.cache)
Server	Nom et version du logiciel serveur
Set-cookie	Permet au serveur d'écrire un *cookie* sur le disque du client

Les en-tête Hop-by-hop

Ces en-têtes sont destinés à être examinés par tout nœud recevant le paquet

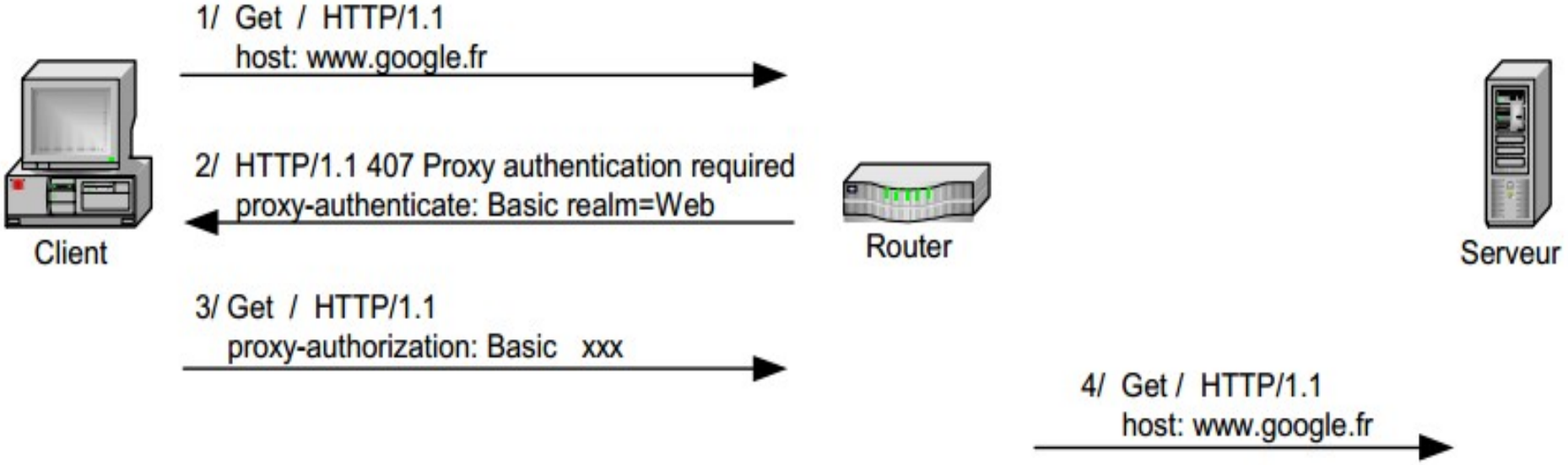
Les en-têtes suivants sont de type Hop-by-hop:

- Connection
- Proxy-authenticate
- Proxy-authorization
- TE (transfer encodings)
- Trailers
- Transfer-Encoding
- Upgrade (vers un autre protocol, eg. HTTP/2.0, SHTTP/1.3, ...)

Exemple du proxy:

Un client doit passer par un proxy pour envoyer des requêtes HTTP sur Internet.

Le proxy nécessite une authentification du client et va utiliser l'en-tête Proxy-authenticate.



Exemple du proxy:

- Le client va donc établir une connexion TCP avec le proxy pour lui envoyer sa requête HTTP;
- Le client n'étant pas encore authentifié, le proxy va lui retourner une réponse contenant l'en-tête proxy-authenticate;
- Le client réitère sa requête en y ajoutant les données d'authentification demandées par le serveur dans l'en-tête proxy-authorization;
- Une fois le client authentifié, le proxy établit la connexion TCP avec le serveur demandé puis lui transmet la requête émise par le client.
- L'en-tête proxy-authorizationn'est pas transmise par le proxy.

Les en-têtes End-to-end:

Ces en-têtes sont destinés au destinataire final de la requête (ou de la réponse) HTTP;

De ce fait, ces en-têtes doivent être ignorés par les nœuds (proxies, routeurs…) constituant le chemin suivi par le paquet;

Les nœuds vont donc simplement transmettre ces en-têtes au destinataire final.

Tous les en-têtes sont de type end-to-end exceptés ceux identifiés comme étant de type Hop-by-hop.

Les en-têtes et la sécurité:

Les en-têtes permettent de fournir aux deux protagonistes d'un échange HTTP des informations afin d'optimiser (cache, connection keep-alive…) et de personnaliser (négociation de contenu, cookies…) le contenu échangé.

Cependant, certaines informations transmises dans ces en-têtes ne devraient pas être divulguée pour des raisons de sécurité comme l'en-tête Server.

En effet, la première action entreprise par un hacker lors d'une tentative d'intrusion d'un système consiste à identifier le système en question.

Les en-têtes et la sécurité:

Cette identification peut être effectuée :

- par prise d'empreinte du système grâce à un outil du type NMAP (ou P0f pour une prise d'empreinte passive) ;
- en regardant la valeur de l'en-tête Server reçue après avoir effectué une requête HTTP.

Par défaut, un serveur Web IIS 6 aura pour en-tête Server:

Server: Microsoft-IIS/6.0

L'en-tête Server peut être modifié à l'aide d'un proxy applicatif (Blue Coat, juniper) ou en configurant le serveur en question.

Codes et status

Lorsque le serveur renvoie un document, il lui associe un code de statut renseignant le client sur le résultat de la requête (requête invalide, document non trouvé…).

Les principales valeurs des codes de statut HTTP sont détaillées dans le tableau ci-après.

Codes et statuts

Code	Nom	Description
Information 1xx		
100	Continue	Utiliser dans le cas où la requête possède un corps.
101	Switching protocol	Réponse à une requête
Succès 2xx		
200	OK	Le document a été trouvé et son contenu suit
201	Created	Le document a été créé en réponse à un PUT
202	Accepted	Requête acceptée, mais traitement non terminé
204	No response	Le serveur n'a aucune information à renvoyer
206	Partial content	Une partie du document suit
Redirection 3xx		
301	Moved	Le document a changé d'adresse de façon permanente
302	Found	Le document a changé d'adresse temporairement
304	Not modified	Le document demandé n'a pas été modifié
Erreurs du client 4xx		
400	Bad request	La syntaxe de la requête est incorrecte
401	Unauthorized	Le client n'a pas les privilèges d'accès au document
403	Forbidden	L'accès au document est interdit
404	Not found	Le document demandé n'a pu être trouvé
405	Method not allowed	La méthode de la requête n'est pas autorisée
Erreurs du serveur 5xx		
500	Internal error	Une erreur inattendue est survenue au niveau du serveur
501	Not implemented	La méthode utilisée n'est pas implémentée
502	Bad gateway	Erreur de serveur distant lors d'une requête *proxy*

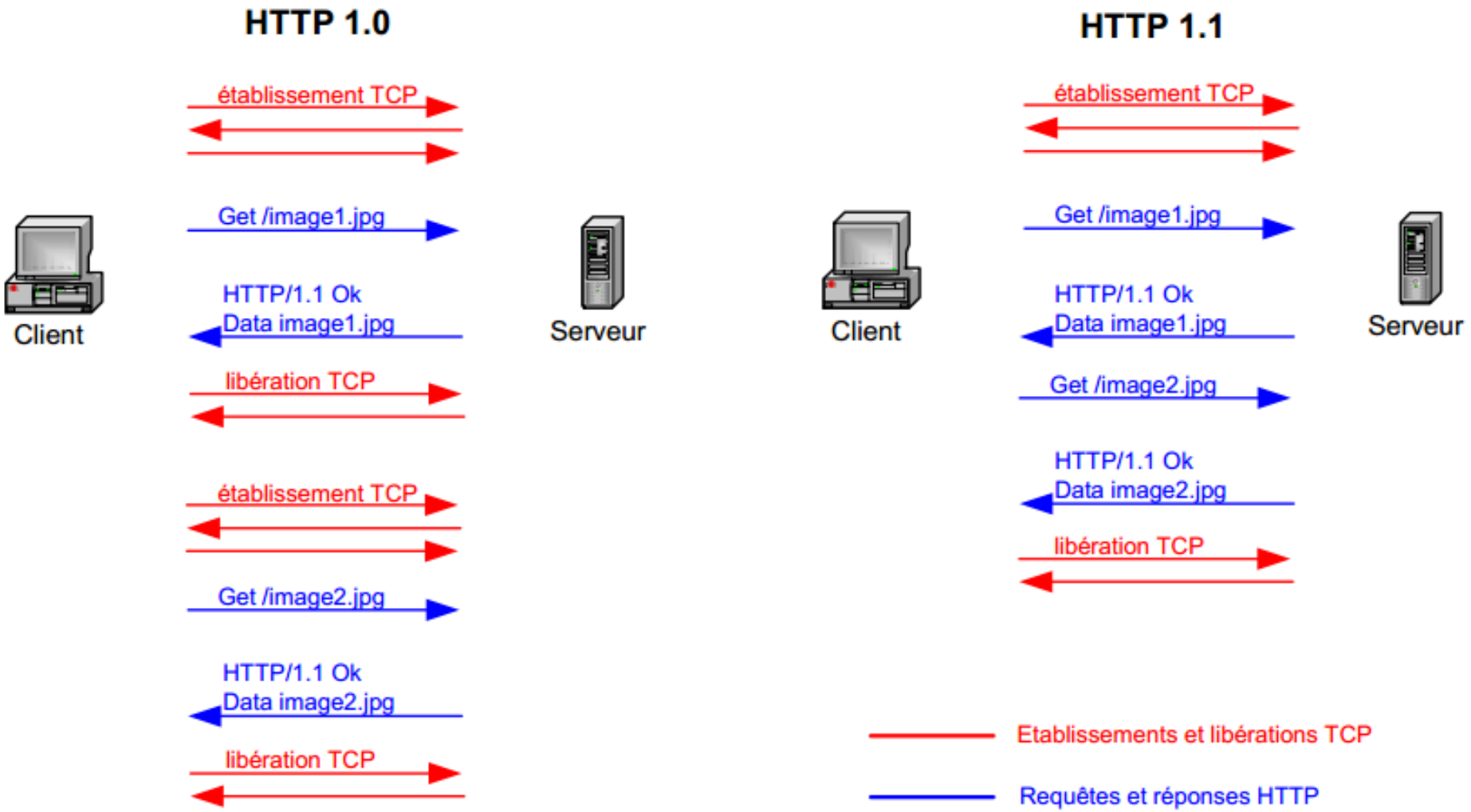
Connexions persistantes, cache et négotiation du contenu

Connexions persistantes :

- La fonctionnalité de connexions persistantes, apparue avec la version 1.1, est sans doute l'amélioration la plus notable !
- Avec la version 1.0 de HTTP, une nouvelle connexion TCP devait être établie pour chaque URL demandée.
- Avec la version 1.1, une connexion TCP existante peut être réutilisée pour demander d'autres URLs.
- Il est possible avec la version 1.1 d'HTTP d'effectuer sur une seule connexion TCP plusieurs requêtes HTTP sans avoir à attendre les réponses HTTP (Pipelining).

Réseaux : Hypertext Transfert Protocol

Connexions persistantes, cache et négotiation du contenu



Cette amélioration permet de :

- diminuer la charge CPU (pour les routeurs, serveurs Web, clients, proxies…) engendrée par les établissements TCP ;

- diminuer le nombre de paquets transitant sur le réseau dû aux établissements TCP ;

- diminuer la latence engendrée par l'établissement des connexions TCP

Les clients désirant maintenir une connexion TCP ouverte doivent inclure dans leur requête HTTP l'en-tête ***Connection: keep-alive***

Cache

Une page HTML est souvent constituée d'informations (textes ou images) ne subissant aucune modification pendant plusieurs jours.

Il devient alors intéressant de mettre en cache les objets pour lesquels nous avons déjà effectué une requête.

Il existe deux types de chache:

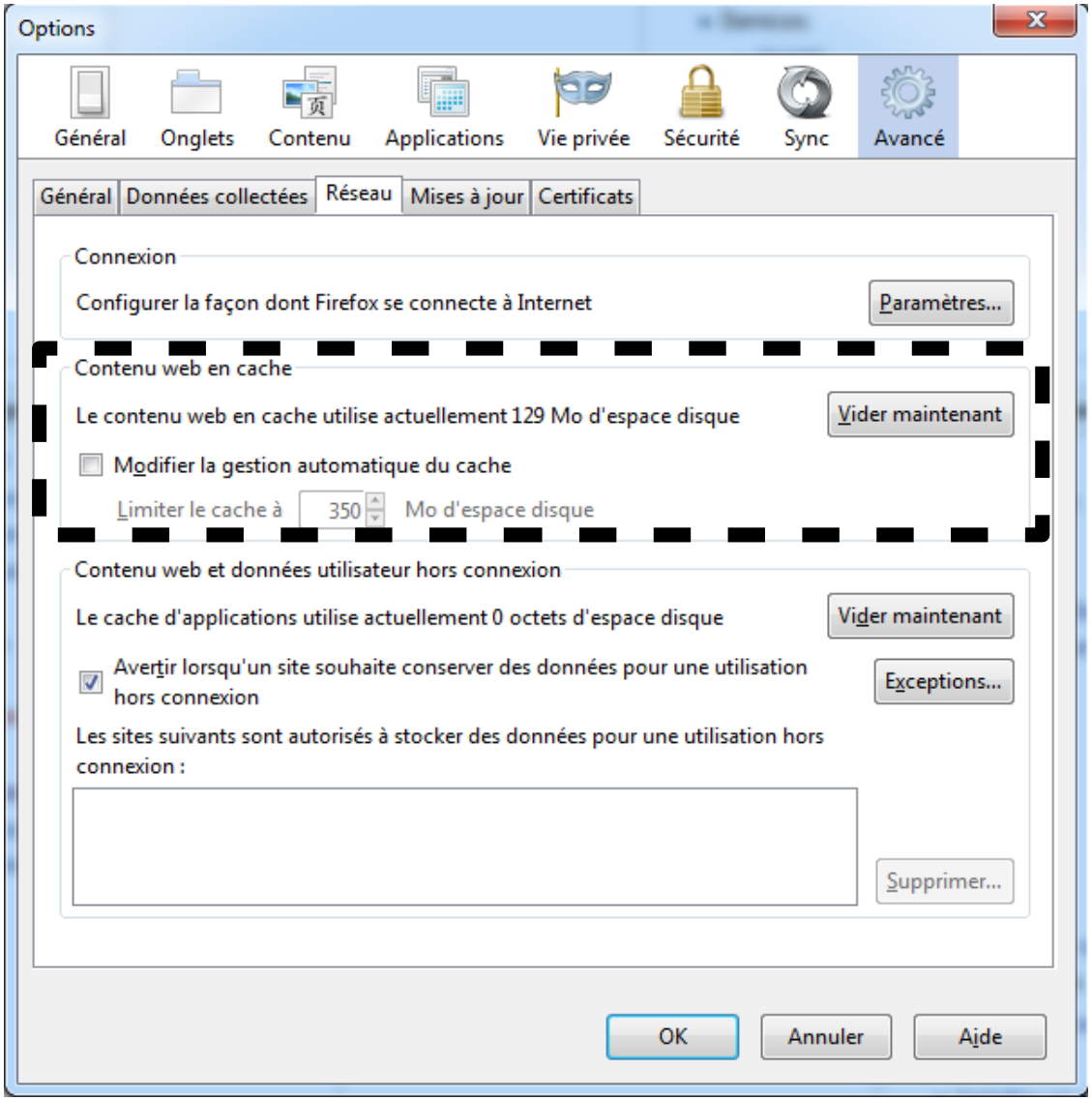
- Le cache navigateur;
- Le cahce proxy.

Cache du navigateur :

Les navigateurs actuels possèdent tous la fonctionnalité de cache.

Ce cache va permettre au navigateur de fournir immédiatement les objets présents dans le cache sans avoir à effectuer une requête auprès du serveur possédant l'objet.

Dans le cas de Firefox, les réglages conernant le cache sont disponibles de la partie réseau:



Cache proxy

Les proxies HTTP possèdent la fonctionnalité de cache. Ce cache va permettre au proxy de fournir aux clients les objets présents dans son cache sans effectuer de requête HTTP auprès du serveur possédant l'objet.

Ce cache est à la disposition de tous les clients utilisant le proxy.

Négociation de contenu :

La négociation de contenu a pour but de fournir à un client la représentation la mieux adaptée à ses
besoins.

Par exemple:

un site international propose des informations en plusieurs langues. L'utilisateur arrive sur le site sur une page index.html où il doit sélectionner la langue dans laquelle il veut consulter les informations.

Il serait intéressant de pouvoir automatiser cette phase et donc, de fournir à l'utilisateur la représentation appropriée des informations, sans qu'il ne doive sélectionner la langue désirée.

La version 1.1 de HTTP permet cette automatisation grâce à l'en-tête:

Accept-Language

Cet en-tête est, normalement, présent dans toute requête HTTP.

Cet en-tête accepte plusieurs valeurs simultanément.

Accept-Language: fr, en

Il est possible de pondérer ses préférences.

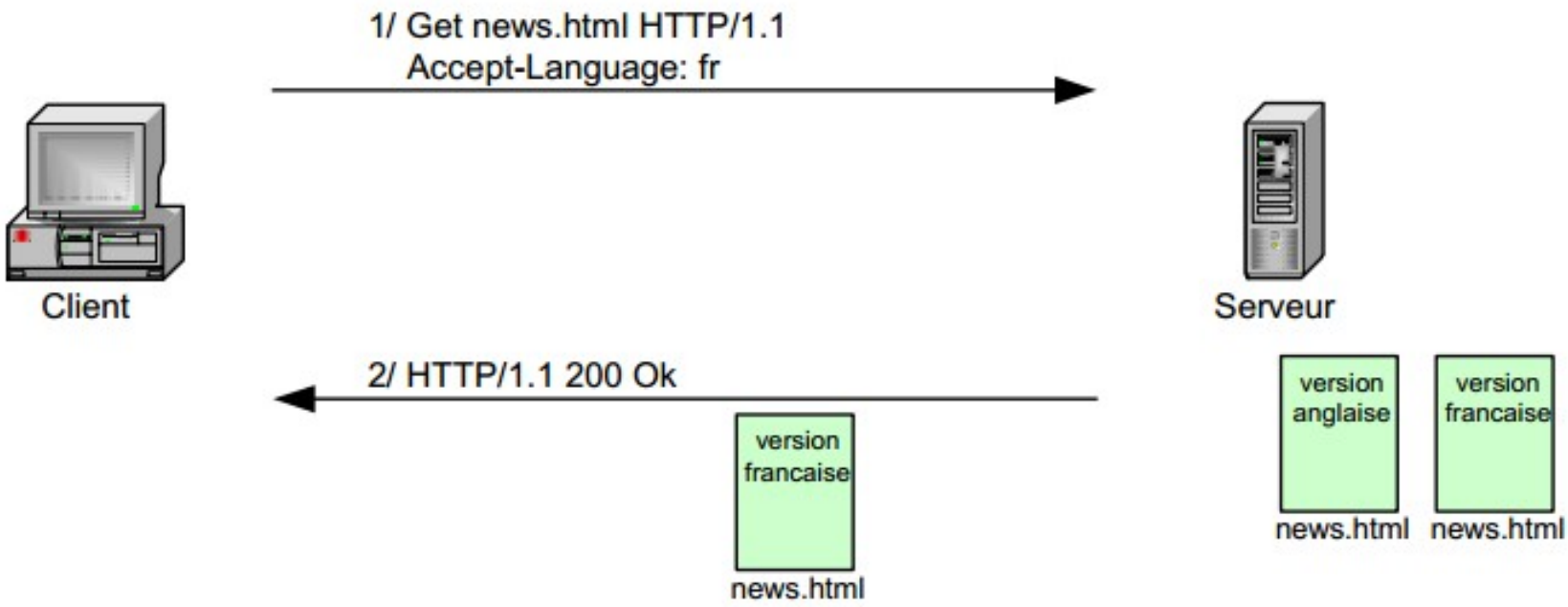
Accept-Language: fr ; q=1.0 , en ; q=0.8

Ces principes sont applicables aux jeux de caractères, types de fichiers textuel (HTML ou .txt), types d'images (JPEG ou GIF)…

Voici les en-têtes concernant la négotiation de contenu:

En-tête	Description
Accept	Permet de spécifier les préférences concernant les types audio, textuels, images...
Accept-Language	Permet de spécifier les préférences concernant les langues
Accept-Charset	Permet de spécifier les préférences concernant les jeux de caractères
Accept-Encoding	Permet de spécifier les préférences concernant les compressions et encodages

Exemple de négatiation de contenu :



Le client effectue une requête afin d'obetnir le fichier news.html en spécifiant qu'il doit être retourné en français en priorité.

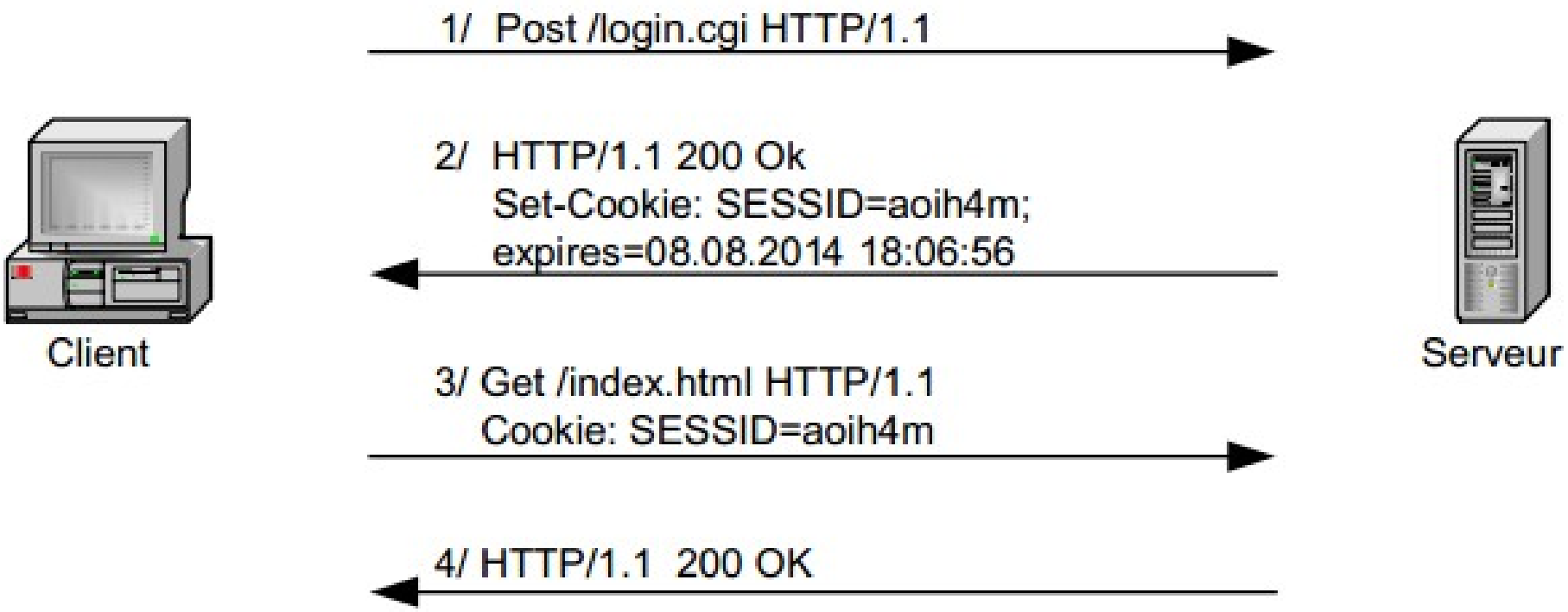
Le serveur possèdant deux versions du fichier et ayant connaissance des préférences du client grâce à l'en-tête Accept-Language, il va retourner la version française du document.

Cookie, authentification et redirection

Cookies :

- Le protocole HTTP ne possède pas de mécanisme de session.
- Pour pallier à ce manque, Netscape a créé les cookies afin de fournir à HTTP un mécanisme de gestion d'états.
- Les cookies vont permettre au serveur de mémoriser des données du côté client. Cela permet un suivi de l'utilisateur d'une requête à l'autre et d'implémenter ainsi la notion de session.
- Afin de donner l'ordre au client de créer un cookie, le serveur place un en-tête Set-Cookie dans sa réponse comprenant :
  - Le nom du cookie
  - sa valeur
  - son contexte (path)
  - sa date d'expiration

Le schéma de séquence suivant présente l'utilisation d'un cookie



Dans le schéma de séquence précédent le client après avoir rempli un formulaire (username/password) :

- Envoi (Post) les données à un script login.cgi ;
- Le serveur informe le client que les données ont été correctement reçues par le script (200 Ok) et demande au client de créer un cookie ayant pour valeur SESSID=aoih4m et expirant le 08.08.2014 à 18:06:56;
- Le client effectue une requête pour obtenir le fichier index.html en joignant à la requête le cookie créé;
- Après avoir vérifié que la valeur SESSID correspondait bien à une session valide, le serveur renvoie le fichier demandé au client.

Un cookie est souvent utilisé afin de mémoriser :

- une clé de session à validité limitée
- les préférences de l'utilisateur concernant un site Web
- les achats réalisés par l'utilisateur
- etc…

Champs d'un cookie

NAME : Obligatoire, le nom attribué par le serveur au cookie;

Value: Obligatoire, une valeur attribuée au cookie par le serveur et théoriquement dénuée de sens pour l'utilisateur. Cette valeur sert à identifier l'utilisateur;

Comment='value': Optionnel, comme les cookies peuvent être utilisés pour enregistrer des informations sensibles sur un utilisateur, ce champ est utilisé par le serveur pour décrire comment il va utiliser ce cookie.

Discard: Optionnel, sert à informer le navigateur qu'il peut ignorer le cookie dès que l'utilisateur termine sa session.

Version= 'value': Obligatoire, ce champ identifie la version du mécanisme de gestion d'état.

Champs d'un cookie

Domain= 'value': Décrit comme optionnel par la RFC 2965 mais en réalité obligatoire, sert à spécifier le domaine pour lequel le cookie est valide. Il est utilisé par le navigateur pour savoir à qui (serveur) le cookie doit être envoyé.

Max-Age='value': Optionnel, donne l'age maximum du cookie avant qu'il ne soit "périmé". Sa valeur est en seconde, depuis le 1er janvier 1970 00:00:00 GMT.

Path='value' : Optionnel, spécifie le chemin sur le serveur auquel le cookies'applique.

Secure: Optionnel, c'est un booléen qui spécifie si une connexion SSL est nécessaire pour accéder au cookie.

Authentification

L'accès à certains documents peut nécessiter une authentification du client.

Dans la version 1.0 du protocole HTTP, seule l'authentification de type Basic était disponible.

La version 1.1 ajoute l'authentification de type Digest.

Authentification Basic :

Authentification Basic :

- Pour consulter une page Web, le navigateur envoie une requête HTTP au serveur contenant l'URL du document à obtenir.
- Le serveur va retourner au navigateur une réponse HTTP possédant un code de statut 401 (Unauthorized) avec l'en-tête www-authenticate. Cet en-tête est suivie par le nom de la méthode d'authentification à utiliser (Basic) et le royaume (realm) auprès duquel l'authentification doit être faite.
- Le client réitère sa requête en y ajoutant son username:password(séparés par ':') encodé au format BASIC, comme demandé par l'en-tête WWW-Authenticate.
- Une fois le client correctement authentifié par le serveur, ce dernier enverra au client les ressources demandées.

Authentification Digest

L’authentification de type Digest est basée sur l’authentification de type Basic et sur un mécanisme de Challenge/Response.

Le Challenge/Response permet d’éviter la transmission du mot de passe sur le réseau.

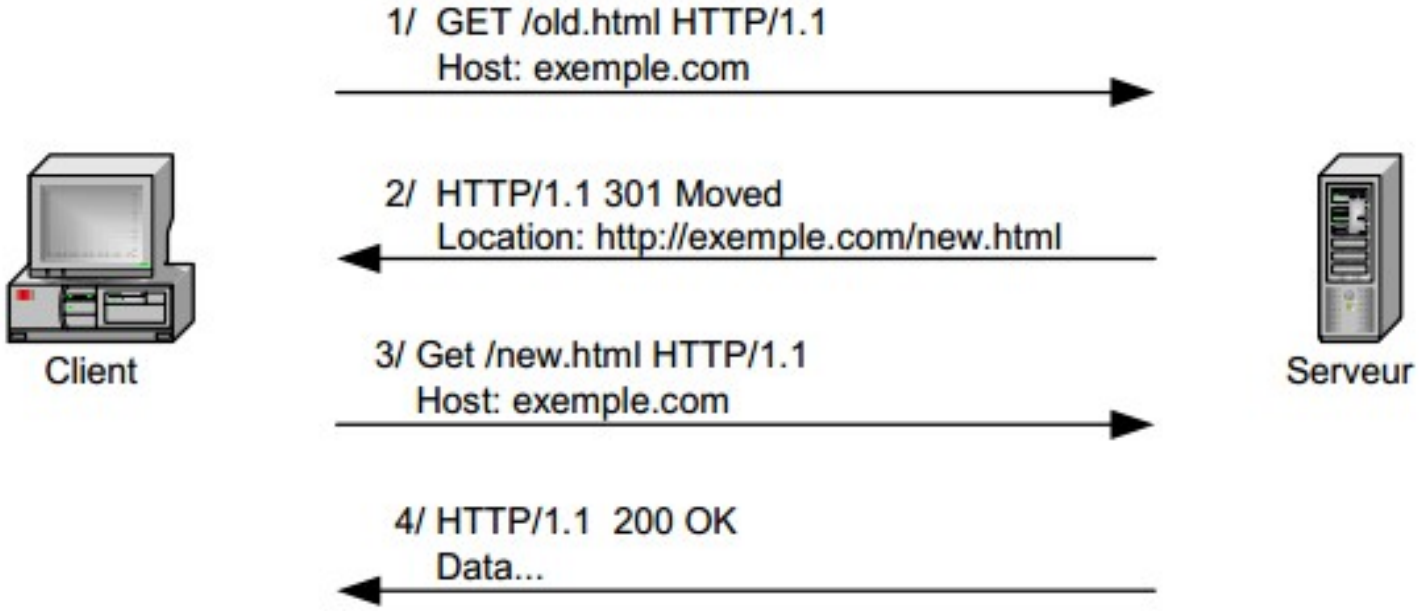
En effet, c’est un hash MD5 du mot de passe et du challenge qui est transmis à la place.

Cela permet d'éviter les attaques de type replay (grâce à la valeur aléatoire qu’est le challenge).

Redirection

Lorsque le serveur renvoie un code de statut de la série 3xx, il accompagne sa réponse d'un en-tête Location indiquant la nouvelle adresse du document.

Dans ce cas, le navigateur va automatiquement émettre une requête vers cette nouvelle adresse.



Conclusion

Conclusion

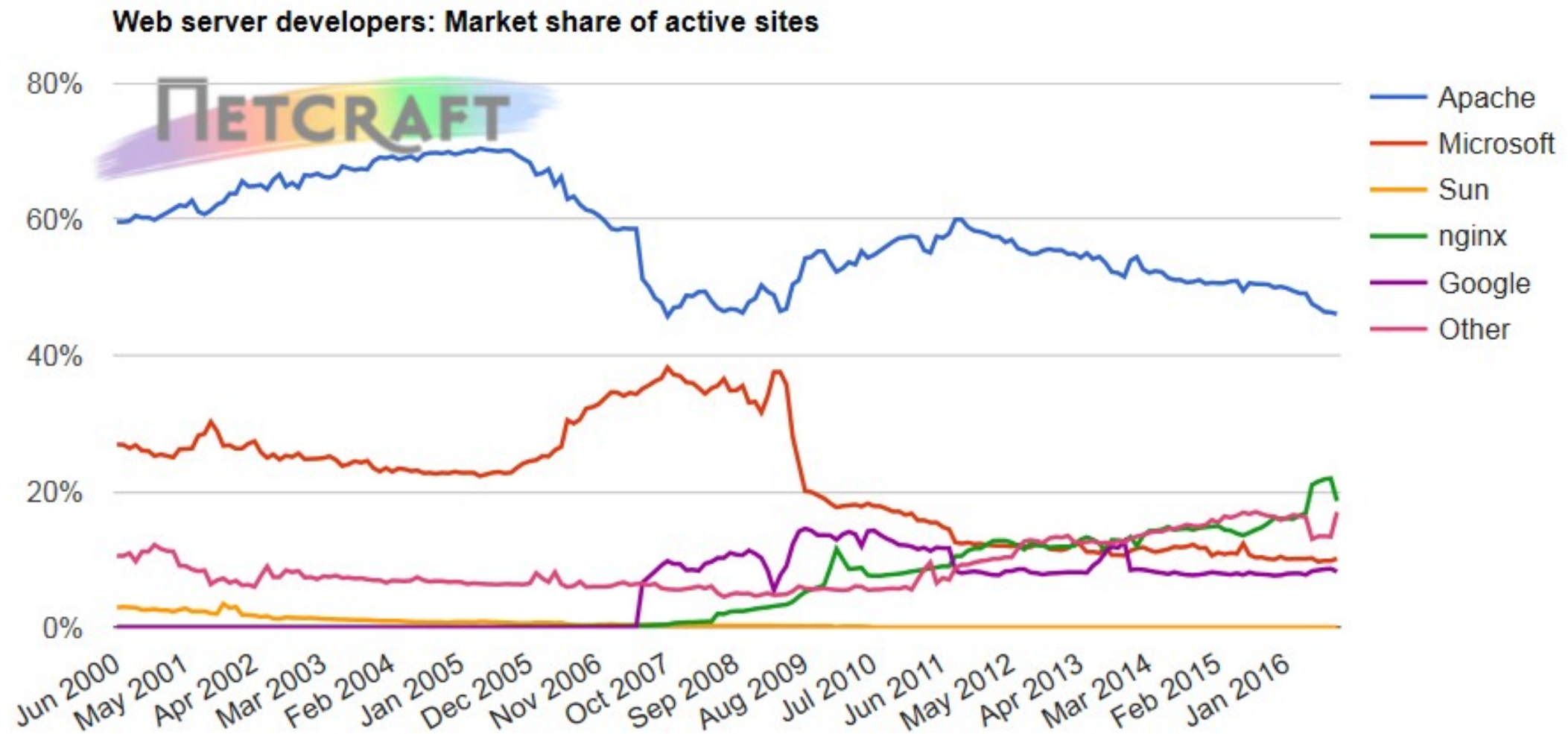
Le niveau de complexité, en terme de nombre de fonctionnalités (plugins), atteint par les navigateurs et les serveurs Web est tel que ces applications possèdent toutes de nombreuses failles.

Ces failles se situent pour la plupart au niveau applicatif et peuvent être exploitées par un script (Cross Site Scripting, Remote Directory Traversal…) pour faire des buffers overflow visant, par exemple, Windows Media Player, Internet Explorer ou même des processus système comme **lsass.exe**.

La démocratisation du firewall laisse croire aux utilisateurs que leur poste est sécurisé et exempt de tout risque alors que souvent, afin de permettre la navigation sur le Web, les firewalls laissent le port 80 ouvert (une entrée directe P2P, IM, etc. sur le poste client).

Les failles ne se situent pas au niveau du protocole HTTP mais au niveau des données transmises.

Conclusion


Web server developers: Market share of active sites
NETCRAFT
80%
60%
40%
20%
0%
Jun 2000
May 2001
Apr 2002
Mar 2003
Feb 2004
Jan 2005
Dec 2005
Nov 2006
Oct 2007
Sep 2008
Aug 2009
Jul 2010
Jun 2011
May 2012
Apr 2013
Mar 2014
Feb 2015
Jan 2016
Apache
Microsoft
Sun
nginx
Google
Other