HTTP, le protocole du web

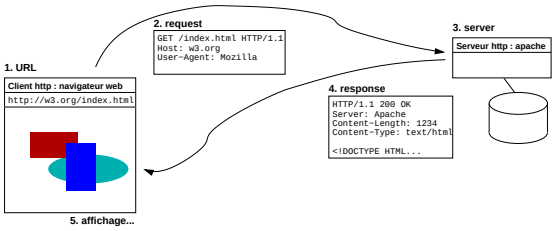
Fabien Coelho
fabien@coelho.net

$Id: http.tex 1183 2006-12-04 11:55:36Z coelho $

- protocole client-serveur (requêtes-réponses)
- construit sur TCP/IP (liaison fiable)
- récupération de documents
  désignés par une URI
  typés (text, image, son. . .)
- services : cache, négotiation, redirections,
  sécurité (authentification, SSL), *cookies*. . .
- implémentations : navigateurs (netscape, IE, hotjava, lynx),
  proxy (squid), serveurs (apache, IIS)

Protocole HTTP en action

navigateur - URL - requête - serveur - réponse



HTTP ?

P Protocol
  accord préalable pour une opération
  nécessaire à la cohérence de la communication

T Transfer
  transmission de données
  similaire à `ftp`

HT HyperText
  notion créée par *Ted Nelson* en 1965
  documents non linéaires, liaisons entre différentes parties
  transport de HTML, typage des informations
  référenceur (`Referer`), redirections (`Location`)

Origine de HTTP

- création au CERN
- défini par le World Wide Web Consortium (W3C)
  consortium MIT/INRIA/(CERN)
  `http://www.w3c.org/`
- spécifié dans un **RFC** : *Request For Comments*
  textes de loi d'internet. . .
- cohérence nécessaire avec HTML et URI

Auteurs du protocole

- Tim Berners-Lee (CERN, MIT/W3C)
- Roy T. Fielding (UC Irvine)
- Henrik Frystyk Nielsen (?)
- P. Leach (Microsoft)
- Jim Gettys (Compaq/W3C)
- J. Mogul (Compaq)
- Larry Masinter (Xerox)

Versions du protocole

(nombreux Internet Draft)

- `HTTP/0.9`, à partir de 1990 au CERN
  version historique
- `HTTP/1.0`, février 1996 : RFC 1945
  version initiale
- `HTTP/1.1`, août 1997 : RFC 2068
  + hôtes virtuels (partage numéro IP)
  + connexions persistantes (plusieurs requêtes)
- `HTTP/1.1`, juin 1999 : RFC 2616
  mise à jour, clarification du document. . .

Compléments au protocole

basic and digest authentification RFC 2069 puis 2617

state management RFC 2109 puis 2965
  il s'agit des *cookies !*
  gestion de **sessions** avec état courant

versions RFC 2145
  interprétation des numéros de version. . .

WebDAV RFC 2518

- protocole **client-serveur**
- basé sur **TCP/IP** (nom d'hôte, port)
- transfert de **données** (comme ftp)
- données **typées** (MIME, RFC 822) ($\neq$ `ftp`)
  `text/html image/gif audio/mp3 video/*`...
- **pas** d'état courant de la connexion
  requêtes indépendantes les unes des autres *(standalone)*
  ($\neq$ `ftp` : répertoire, type de transfert, login)
- connexion commune à plusieurs requêtes

- flux bi-directionnel **asymétrique** (symétrique=messages)
  requêtes (formulaire), réponses (résultat),
- identification du fournisseur par son **nom** (pas numéro IP)
  `www.ensmp.fr dev.apache.org`...
- identification du *document* au sens large
  fichier, documents dynamiques générés au vol...
- authentification de base par mot de passe
- **négociation** du contenu (format, encodage, langue, qualité)
- limite de validité (pour les caches)
- etc.

Moyens

- **deux** types de messages à entêtes : **requêtes** et **réponses**
  format en texte ASCII (lisible !) similaire à un *e-mail*
- contenu associé joint
  données attachées au résultat
  éventuellement transmis en morceau...
- maintien des connexions (plus efficace) :
  suite de requêtes et de réponses
  nécessite une synchronisation !
  doit donc connaître les quantités transférées

Exemple de requête d'un navigateur

```
GET /manual/index.html HTTP/1.1
Host: www.apache.org
User-Agent: Lynx/2.8.1
Referer: http://www.apache.org/index.html
Accept: text/html, text/plain, image/*, audio/*
```


Exemple de réponse d'un serveur

```
HTTP/1.1 200 OK
Date: Fri 02 Nov 2001 15:43:20 GMT
Server: Apache/1.3.14 (Unix)
Content-Length: 1358
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD.HTML.3.2.Final//EN">
<HTML>...
```


URI - RFC 2396 (août 1998)

`http://www.w3c.org/index.html`

- URI = *Uniform Resource Identifier*
- identification/nommage d'une ressource sur le Web
  abstraite (service) ou physique (fichier)
- Tim Berners-Lee, Roy Fielding, Larry Masinter
- historique : spécifié depuis 1994 (RFC 1738 puis 1808)
- avec une **chaîne de caractères** compacte
- 2 niveaux : syntaxe (validité) et sémantique (interprétation)
- destiné : aux utilisateurs, aux navigateurs, aux serveurs

Uniform

- commun à différents types de ressources
  fichiers, annuaires, documents web, messagerie...
- ajout facile de nouveaux types
- réutilisable dans de nouveaux contextes
- pas d'ambiguité.

Identifier

- **référence** à une entité ! (étiquette, pointeur...)
- URI : chaîne de caractères avec une syntaxe spécifique

Resource

- une entité identifiable
- exemples :
  - un document électronique : texte image son film...
  - un service (notion dynamique) : recherche
  - un livre, un CDROM...
  - une entreprise, une personne...
- ressource $=$ correspondance nom - entité
- entité (contenu) peut changer au cours du temps...

```
mailto:postmaster@ensmp.fr

telnet:leuven.ensmp.fr

file:/etc/httpd.conf

ftp://ftp.inria.fr/pub/gnu/bash-2.0.tgz

http://www.apache.org/manual/index.html

http://www/cgi-bin/test.pl?couleur=rouge&forme=rond

https://admin:secret@193.48.171.200:8080/
```


Vocabulaire UR*

- URI : *Uniform Resource Identifier*
  identification d'une ressource
- URL : *Uniform Resource Locator*
  URI avec instructions de récupération : **comment** et **ou**
- URN : *Uniform Resource Name*
  URI avec notion de persistance
- URC : *Uniform Resource Citation*
  paires attribut/valeur pour décrire une ressource

Localisation

- localisation au sens réseau : machine...
- précise le protocole principal `http ftp gopher`...
- utilisation d'autres protocoles
  typiquement, le DNS pour traduire le nom

Nommage

- `urn:`
- unicité nécessaire, même si disparition
- en cours de définition (RFC 2141)

Caractères d'une URI

- des caractères, pas des octets !
  nombre très limité et fixé
  tous supports : électronique, impression, oral...
- **3 types** : réservés, non-réservés, échappés
  non réservés `a-z A-Z 0-9 -_.!~*'()`
  réservés délimiteurs de l'UR* : `;/?:@&=+$,`
  échappés encodage US-ASCII (7 bits) :
  `% hex hex` (`hex` parmi `0-9 a-f A-F`)
  `%20` = espace, `%25` =pourcent, `%23%` =dièse
- ne pas échapper les caractères non réservés !

Caractères exclus (directement) des URI

- contrôle (US-ASCII 00-1F, 7F)
- espace car impression sur papier (publicité)
- # (fragments d'URI)
- `" < >` car utilisés pour délimiter une URI
  `"http://www.iar2m.ensmp.fr/"`
  `<http://www.iar2m.ensmp.fr/>`
- % caractère d'échappement
- `{ } | \ ^ [ ] ‘` car potentiellement transformés
- par contre ~ est bien présent !

Syntaxe complète d'une URI

`<scheme>:<scheme-specific-part>`

- pas très précise ni utilisable;-)
- exemple de schémas :
  `http ftp file mailto gopher telnet jdbc`

Les URI hiérarchiques

`<scheme>://<authority><path>?<query>`

- classe particulière bien utile
- `http ftp gopher`
- toutes les parties sont facultatives
  interprétation à préciser en cas d'absence
  le préfix annonce la partie suivante
  `//<authority>  /<path>  ?<query>`
- ordre : `:  // / ?  #`

Authorité : `authority`

`<login>:<pass>@<machine>:<port>`

- introduite par `//`
- parties facultatives : login, passe...
- nom ou numéro IP (v4) du **serveur**
- numéro de port (défaut selon le `scheme`)
- ne doit pas contenir de `/ ? #` (annonce la suite)
  `www.iar2m.ensmp.fr  deauville:8888`
  `calvin@hobbes.net  10.2.16.200:80`
  `www.cnn.com@193.48.171.200`

Chemin d'accès : path

`/<component1>/<component2>...`

- introduit par / qui fait partie du path
- composants hiérarchiques d'une arborescence
  répertoires et fichiers apparents
- peut se terminer par /
- ne doit pas contenir ? # (annonce la suite)
- exemples : `/` `/usr/local` `/eval/`

Requête particulière : query

`?<query>`

- introduite par un ?
- ne doit pas contenir de # (annonce la suite)
- souvent composée de couples attributs/valeurs (= &)
  transfert des données d'un formulaires
  `?login=lapinot&email=lapinot@trondheim.org`
- ne fait pas partie de la hiérarchie (URI relatives)

Référence à un fragment d'URI

`<uri>#<designation-du-fragment>`

- introduite par #
- format libre (fin de l'URI)
- typiquement, désigne une section de document
- exemples : `#introduction` `#reference`
- ne circule pas !
  interprétée par le navigateur pour se positionner !
  l'URI est avant le #

Exemples d'URI hiérarchiques complètes

`http://www.iar2m.ensmp.fr/main.html#objectif`

`http://palo-alto/cgi-bin/printquery?name1=val1&name2=val2`

`ftp://coelho@blonville/usr/local/src/web/apache_1.3.9.tar.gz`

`ftp://ftp:%66%63%40@ftp.cri.ensmp.fr/pub`

URI relatives

- détermination des champs manquants selon le **contexte**
- contexte : une URI de référence
- schéma : le même !, autorité : la même !
- chemin : comme un système de fichier classique
  moins dernier composant après /
  en relatif avec `..` et `.`
- exemple :
  `http://cri/doc/rapports/rapports.html#ref`
  `../images/redball.gif`
  `http://cri/doc/images/redball.gif`

Exemples d'URI relatives

`http://www.w3c.org/cgi-bin/search?key=html`

`telnet:cri`	`telnet:cri`
`info`	`http://www.w3c.org/cgi-bin/info`
`./info`	`http://www.w3c.org/cgi-bin/info`
`admin/`	`http://www.w3c.org/cgi-bin/admin/`
`/doc`	`http://www.w3c.org/doc`
`//w3.org`	`http://w3.org`
`?z=y`	`http://www.w3c.org/cgi-bin/search?z=y`
`info?quest`	`http://www.w3c.org/cgi-bin/info?quest`
`#partie`	`(current document)#partie`
`.`	`http://www.w3c.org/cgi-bin/`
`..`	`http://www.w3c.org`
`../`	`http://www.w3c.org/`
`../icon`	`http://www.w3c.org/icon`

URI spécifique http

`http://host:port/path/to/somewhere?query`

- port par défaut : 80
  `http://palo-alto/index.html`
- chemin?query ou ?query peuvent être absents
- chemin : au minimum /...
  pour les répertoires, mieux si se terminent par /
  évitent les redirections pour la gestion des URL relatives
  `http://www.apache.org/manual` + `licence.html`

Exemple de problème réel

- *Fabien, mon URL elle marche pas !*
  `<a href="/cgi-bin/tete_sn.pl?mot=&acirc;ge">...`
- URL générée automatiquement par un CGI
  le CGI destinataire ne récupère pas le mot...
- `&acirc;` interprété par le navigateur ? non
- & et ; sont séparateur des URL...
  substituer par leurs codes ASCII respectifs !
  `<a href="/cgi-bin/tete_sn.pl?mot=%26acirc%3bge">`

Conclusion sur les URI

- désignation d'un document au sens large
- utilisation en HTML (href=...) et HTTP
- au moyen d'une chaîne de caractères compacte
  caractères ASCII directs, spéciaux ou échappés
- URL : localisation du document
- différentes parties pour les URI hiérarchiques
  - schéma `:` `//` authorité `/` chemin...
  - `?` query
  - `#` référence (uniquement pour le client)

Présentation de HTTP

- spécifié dans le RFC 2616
- caractéristiques du protocole
- syntaxe des requêtes et des réponses
- syntaxe des entêtes
- fonctions et extensions disponibles
  authentification, cache, cookies...

Caractéristiques

- client-serveur : asymétrique requête/réponse
- pas d'état :
  chaque requête est complète et indépendante
  répétition intégrale des informations à chaque requête
  (authentification, préférences, capacités)
  ajout de la notion d'état avec les **cookies**
- désignation du document par URI

La chaîne d'acteurs HTTP

user-agent le client : un navigateur, un robot

intermédiaires éventuels

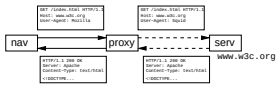
proxy intermédiaire HTTP explicite (préférences navigateur)

gateway intermédiaire HTTP, non explicite (interception)

cache intermédiaire HTTP qui garde des copies
(performances, peut être un proxy ou un gateway)

tunnel transfert TCP/IP aveugle

server le serveur



Contenu de la spécification HTTP

- au dessus de TCP/IP, mais assez indépendant
- format des requêtes et réponses
  - caractères autorisés
  - première ligne (requête ou réponse)
  - conventions générales :
    synchronisation
    continuations, fin des entêtes, dates
    langages, qualité...
    transmissions (tailles, codage éventuel...)

- définition et signification des champs d'entête
  en particulier leurs occurences possibles :
  - requêtes
  - réponses
  - les deux
- niveaux de spécification `MUST` `MAY` `SHOULD`...
- des conseils, des avertissements...

Données manipulées par le protocole

- niveau TCP/IP : flux bidirectionnel d'octets
- entêtes lisibles, similaire à un message
  - caractère US-ASCII (0-127)
  - interprétés ligne à ligne
    fin de ligne : `\r\n` (*carriage return, new line*)
  - première ligne des entêtes spécifiques
  - champs spécifiques et valeur `Champs: valeur...`
  - ligne de continuation : commencent par tab ou espace
  - fin des entêtes : une ligne vide ! (`\r\n\r\n`)
- ensuite données (binaires) associées, non interprétées

Faire des requêtes manuelles

- pour du debug d'un serveur
- pour comprendre le protocole
- pour faire le TP...

Commandes `telnet` ou nc

- commandes unix standard, parfois sous windows

  `man telnet`
- connection TCP/IP interactive
- destinataire : numéro IP / numéro de port
- source : vous même !
- entrée standard : envoyée au serveur
- sortie standard : réponse du serveur

Syntaxe : `telnet <dest-host> <dest-port>`

- `<dest-host>` hôte destination

  nom ou numéro IP
- `<dest-port>` port destination

  pour http, c'est en général 80
- côté client :

  le numéro IP est celui de la machine

  un numéro de port (> 1000) est alloué


```
prompt> telnet www.iar2m.ensmp.fr 80
Trying 192.54.172.235...
Connected to www.iar2m.ensmp.fr.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.0 200 OK
Date: Wed, 10 Nov 1999 17:11:13 GMT
Server: Apache/1.3.9 (Unix)
Last-Modified: Fri, 10 Jul 1998 16:38:46 GMT
ETag: "3e48a-3f0-35a64396"
Accept-Ranges: bytes
Content-Length: 1008
Content-Type: text/html

Connection closed by foreign host.
```


Structure des requêtes et réponses

```
Format spécifique de la première ligne
Champs1: valeur associée au champs 1
Champs2: idem pour le champs 2
Champs3: la valeur du
 champs 3 peut s'étaler sur
 plusieurs lignes...
Champs4: Hop la !

données éventuelles associées...
qui suivent les entêtes...
après une ligne vide...
par exemple un document résultat.
```


Exemple de requête

```
POST /cgi-bin/search HTTP/1.1
Host: www.w3c.org
User-Agent: Lynx/2.8.1
Referer: http://www.w3c.org/index.htm
Accept: text/html, text/plain,
  image/gif, image/jpeg
Content-Type: application/x-www-form-urlencoded
Content-Length: 9

word=html
```


Format d'une requête

- première ligne : `<method> <uri...> <version>`
- `<method>` : le type d'opération souhaitée (*case sensitive*)

  `GET POST...`
- `<uri>` : le document requis

  schéma et authorité non indispensables (sauf proxy)

  ne peut pas être vide, si non approprié : `*`
- `<version>` : la version du protocole

  `HTTP/0.9`, `HTTP/1.0` ou `HTTP/1.1`
- ensuite entêtes (*headers*) et message

Exemple de réponse

```
HTTP/1.1 401 Authorization Required
Date: Fri, 02 Nov 2001 16:31:17 GMT
Server: Apache/1.3.14 (Unix)
WWW-Authenticate: Basic
  realm="intranet du site"
Content-Type: text/html;
  charset=iso-8859-1
Content-Length: 2765

<!DOCTYPE HTML...
```


Format d'une réponse

- première ligne :

  `<version> <status-code> <explanation...>`
- `<version>` : idem requête
- `<status-code>` :

  3 chiffres décimaux, répartis dans 5 classes

  interprétation spécifiée dans le RFC
- `<explanation>` : message *lisible* d'explications

  contenu parfois fantaisiste
- ensuite entêtes et message

Les types d'entêtes

- 48 champs définis par HTTP
- *case insensitive*
- 4 types de champs
  - 19 spécifiques aux requêtes : *request*
  - 9 spécifiques aux réponse : *response*
  - 9 généraux à la connexion : *general*
  - 11 concernant l'entité, le document : *entity*
- et des extensions : netscape cookies

Requêtes de formulaires

- envoyées par un formulaire HTML

```
<FORM METHOD=GET ACTION="/cgi-bin/hello.cgi">
<INPUT TYPE="text" NAME="prenom" SIZE=20>
<INPUT TYPE="submit" VALUE="Ok">
</FORM>
```

- transmission par défaut des champs selon la **méthode**

  document joint ou dans l'URL requise
- encodage type URL

- méthode GET : partie *query* de l'URL

```
GET /cgi-bin/hello.cgi?prenom=Calvin HTTP/1.1
Host: sablons.ensmp.fr
```

- méthode POST : document joint + encodage URL

```
POST /cgi-bin/hello.cgi HTTP/1.0
Content-Length: 13
Content-Type: application/x-www-form-urlencoded

prenom=Calvin
```


Les méthodes de requêtes

- `GET` : obtention d'un document
- `POST` : soumission d'un document
- `HEAD` : consultation des entêtes (pour cache)
- `PUT` : stockage d'un document !
- `DELETE` : effacement d'un document !
- `OPTIONS` : consultation des possibilités du serveur
- `TRACE` : requête miroir (pour déboguage)
- `CONNECT` : changement de protocole
- *extensions...* (webdav)

Méthode GET

- méthode principale de HTTP !
- obtention d'un document par son URI
- statique (fichier) ou dynamique (CGI...)
- passage des options après ? dans l'URI...
- conditionnelle (pour caches) avec certains entêtes

  `If-Modified-Since`
- partielle (morceaux) avec `Range`

Exemples de petites requêtes GET

```
GET / HTTP/1.0
```

```
GET /index.html HTTP/1.0
```

```
GET /icons/redball.gif HTTP/1.1
Host: www.iar2m.ensmp.fr
```

```
GET /cgi-bin/printquery?foo=1&bla=2 HTTP/1.1
Host: palo-alto
```

```
GET http://www.ibm.com/ HTTP/1.1
Host: portail
```


Exemple d'une requête de `netscape`

```
GET http://www.netscape.com/images/bar_te/smartupdate.gif HTTP/1.0
Referer: http://www.netscape.com/
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.61 [en] (X11; I; Linux 2.2.13 i686)
Host: www.netscape.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png
Accept-Encoding: gzip
Accept-Language: fr
Accept-Charset: iso-8859-1,*,utf-8
Cookie: UIDC=193.48.171.47:0959079925:793179
```


Méthode POST

- envoi d'une entité associée
  - annotation à ajouter
  - message (news, e-mail...)
  - données provenant d'un formulaire
  - ajout à une base de données
- interprétation dépend du serveur, pas du protocole
- typiquement : formulaires + CGI

Exemple de requête POST

```
POST /cgi-bin/printquery HTTP/1.1
Host: palo-alto
User-Agent: MesPetitesMains-1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 26

hero=Calvin&peluche=Hobbes
```


GET versus POST

- possibilité similaire pour les CGI

  `GET` + URI `?<query>`

  `POST` + entité associée (corps) `query`
- interprétation spécifique au serveur
- mais conseil :
  - `GET` : simple consultation, lecture
  - `POST` : modification, mise à jour
- taille limitée par le serveur de l'URI
- requête `GET` généralement loguée

Méthode OPTIONS

- disponibilité d'options sur un serveur ou proxy
- si précise URI : à propose de la ressource

  langues, formats disponibles...
- si l'uri est `*` : à propos du serveur
- entête `Max-Forwards` :

  pour interroger un proxy sur une chaîne
- pas toujours implémentée...

Consultation des options

```
OPTIONS * HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 30 Oct 1999 12:13:22 GMT
Server: Apache/1.3.9 (Unix)
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close
```


Méthode HEAD

- comme GET, sauf l'entité retournée !
- vérification de la validité, de l'accessibilité

  pour les caches en particulier
- test de modification avec entêtes

  `Content-Length Content-MD5 ETag Last-Modified`
- pas toujours implémentée, voire mal implémentée...

Exemple de méthodes HEAD

```
HEAD / HTTP/1.0

HEAD /index.html HTTP/1.0

HEAD http://www.ibm.com/ HTTP/1.1
Host: portail

HEAD / HTTP/1.1
Host: www.apache.org
```


Méthode PUT

- remplacement d'un document
- similaire à la méthode POST

  entité attachée à la requête
- différence (suggérée) :

  `POST` : soumet une entité à une autre

  `PUT` : remplacement
- comportement effectif non spécifié par le protocole
- client : **publish** de *netscape composer*
- problèmes de sécurité, d'authentification...

  généralement non implémentée ou non disponible !

Méthode DELETE

- effacement d'une entité
- fonctionnement non garanti (heureusement...)
- devrait être prise en compte par un cache
- même problème de sécurité...

Exemples divers

```
PUT /index.html HTTP/1.0
Content-Length: 123

<!DOCTYPE...
```

```
DELETE /cgi-bin/printquery HTTP/1.0
Host: palo-alto2
```

```
DELETE /index.html HTTP/1.1
Host: www.microsoft.com
```


Méthode TRACE

- entité retournée = miroir de la requête
- utilisée pour le debug

  en particulier, entête **Via**
- permet de s'adresser à un proxy intermédiaire

  avec entête `Max-Forwards`

Exemple de requête et réponse TRACE

```
TRACE / HTTP/1.1
Host: leuven
Hello: world
Java: bien
```

```
HTTP/1.1 200 OK
Date: Sun, 31 Oct 1999 17:04:21 GMT
Server: Apache/1.3.9 (Unix)
Transfer-Encoding: 60
Content-Type: message/http

TRACE / HTTP/1.1
Hello: world
Host: leuven
Java: bien
```


Méthode CONNECT

- réservée par HTTP
- passage en mode tunnel d'un proxy

  par exemple pour SSL
- impact sur tous les intermédiaires...
- pas toujours implémentée ?

```
CONNECT www.netcraft.com:443 HTTP/1.1
User-Agent: Mozilla/4.61
```


Entêtes spécifiques aux requêtes

Accept-* formats, etc. acceptés

If-* conditions d'application

Host serveur destinataire

User-Agent navigateur (Mozilla, IE...)

From adresse e-mail de l'utilisateur

Max-Forwards contrôle des proxy

Range sélection d'une partie d'un document

Referer document source de l'URL

Authorization... montre patte blanche

Expect,TE

Requête normale ou de proxy

normale adressée directement au destinataire

- connaît son adresse !
- `Host` précise l'hôte virtuel (1.1)
- `uri` précise le document local

```
GET /index/html HTTP/1.0
```

proxy adressée explicitement à un intermédiaire

- ne connaît pas le destinataire
- `Host` désigne l'intermédiaire
- `uri` doit préciser le destinataire, protocole...

```
GET http://www.microsoft.com/index.html HTTP/1.0
```


Exemples de requêtes normale/proxy minimales

```
GET /index.html HTTP/1.0
```

```
GET http://www.gnu.org/index.html HTTP/1.0
```

```
GET /index.html HTTP/1.1
Host: www.gnu.org
```

```
GET ftp://ftp.gnu.org/ls-lR.gz HTTP/1.1
Host: portail2.ensmp.fr
```


Réponses HTTP

- première ligne avec code de retour
- ensuite entêtes, certains spécifiques

```
HTTP/1.1 200 OK
Server: Apache/1.3.9
Content-Type: text/plain
Content-Length: 10
Connection: close

Bonjour !
```

Les codes de retour des réponses

- 3 chiffres décimaux
- premier chiffre : classe de la réponse, parmi 5

1xx information, traitement en cours

2xx succès

3xx redirection (aller voir ailleurs !)

4xx erreur du client (syntaxe de la requête)

5xx erreur du serveur (problème interne)

10x Informations, 2 codes

- réponse provisoire, plutôt rare

100 Continue

continuer d'envoyer la suite d'une requête (chuncked)

101 Switching Protocols

avec entête `Upgrade`

changement dynamique de protocole, si profitable

autre version de HTTP, ou autre protocole

exemple : son ou video en temps réel...

voir requête `CONNECT`

20x Succès, 7 codes

- exprime le succès de la requête

200 OK ça marche !

interprétation selon la requête

`GET HEAD POST TRACE`...

201 Created création d'une entité

pour PUT par exemple

202 Accepted requête acceptée

traitement ultérieur ou non...

assez flou

203 Non-Authoritative Information

document non nécessairement strictement conforme

par exemple copie envoyée par un proxy-cache

204 No Content

ok, mais pas d'entité résultat

par exemple pour `DELETE`

205 Reset Content

idem, doit re-récupérer le document

e.g. : formulaire accepté, à remettre à jour (`GET`)

206 Partial Content

retour d'un `GET` + `Range`

30x Redirection, 8 codes

- nouvelle action du client nécessaire
- comme un renvoie à une autre URL par exemple
- entêtes complémentaires : `Location`

précise la nouvelle destination

- souvent opéré de manière transparente par le client

300 Multiple Choices

document en plusieurs versions...

doit engager une négotiation éventuelle !

304 Not Modified

récupérations conditionnelles

entête `If-Modified-Since`...

réponse doit préciser certaines entêtes pour vérification

305 Use Proxy

passage par un proxy pour cette requête

désignation via l'entête `Location`

sécurité de la délégation ?

306 code réservé (utilisé à la version précédente)

301 Moved Permanently

mise à jour automatique des liens ? `GET HEAD POST`...

302 Found

redirection **temporaire**

destination précisée avec l'entête `Location`

303 See Other

redirection similaire à 302

mais utiliser la méthode `GET`

307 Temporary Redirect

redirection similaire à 302

même requête (méthode) à soumettre à `Location`

utilisé pour les anciennes versions, avec html cliquable

Exemple de réponse 300

```
HTTP/1.1 307 Temporary Redirect
Date: Fri, 10 Jul 1998 11:31:03 GMT
Server: Apache/1.3.14 (Unix)
Location: /nouveau/document.html
Content-Type: text/html
Content-Length: 4253

<!DOCTYPE HTML...
```

4xx Erreur du client, 18 codes

- erreurs syntaxiques, sémantiques...
- liste la plus complète, 18 codes !
- document associé d'explication...
  normallement destiné à un utilisateur
- notamment utilisé pour les autorisations
  resoumission avec authentification du client

401 Unauthorized pas autorisé
requiert un mot de passe
challenge donné : `WWW-Authenticate` (RFC 2617)

407 Proxy Authentication Required
similaire à 401, mais authentification client-proxy

402 Payment Required Ah ah ah ah !

403 Forbidden refus
autre que problème d'authentification
par exemple, intranet limité aux locaux
implique que le document existe...

404 Not Found le document n'existe pas !
différent de non autorisé (403)

405 Method Not Allowed
méthode non autorisé pour l'entité
typiquement `DELETE` `PUT`, ou `POST` sur HTML
l'entête `Allow` doit préciser les méthodes disponibles

406 Not Acceptable
format requis par le client ne peut être produit
exemple : CGI produit du html, le client veut une image ?

412 Precondition Failed
application d'une méthode après vérification
sorte de mécanisme de lock
exemple : `PUT` + `If-Unmodified-Since`

417 Expectation Failed (avec entête `Expect`)
le client attend un certain retour du serveur

415 Unsupported Media Type
données soumises (`POST`) dans un mauvais format

408 Request Timeout patience du serveur limitée

409 Conflict (protection des versions à la RCS)

410 Gone

411 Length Required

413 Request Entity Too Large

414 Request URI Too Long

416 Requested Range Not Satisfiable

Exemple de réponse 400

```
HTTP/1.1 401 Authorization Required
Date: Fri, 10 Jul 1998 15:31:03 GMT
Server: Apache/1.3.14 (Unix)
WWW-Authenticate: Basic realm="Intranet"
Connection: closed
Content-Type: text/html

<!DOCTYPE HTML...
```


5xx Erreur du serveur

- normallement le serveur doit y survivre !
- explications requises, si possible :
  sous forme d'un document joint
  à destication d'un humain, utilisateur...

500 Internal Server Error argh !

501 Not Implemented
par exemple méthode non reconnue

502 Bad Gateway
erreur d'un intermédiaire dans une chaîne
pas du serveur visé !

503 Service Unavailable
indisponibilité temporaire (surcharge, maintenance)
entête `Retry-After`

504 Gateway Timeout
problème dans la chaîne (DNS, proxy HTTP...)

505 HTTP Version Not Supported
doit préciser les versions supportées (entitée)

Exemple de réponse 500

```
HTTP/1.1 501 Method Not Implemented
Date: Fri, 02 Nov 2001 18:17:20 GMT
Server: Apache/1.3.14 (Unix)
Allow: GET, HEAD, OPTIONS, TRACE
Connection: close
Content-Type: text/html; charset=iso-8859-1
Expires: Sun, 04 Nov 2001 18:17:20 GMT

<!DOCTYPE HTML...
```

Server identification du serveur (OS, soft. . .)

Age age en seconde d'une ressource

ETag Entity TAG, identifiant d'une **variante**

Accept-Range `Range` accepté (none, bytes)

Allow méthodes authorisées pour une ressource

Location pour les redirections

Retry-After arrêt temporaire d'un service

Vary indicateur de cachabilité pour les caches

Proxy-Authenticate WWW-Authenticate authentification

Conclusion sur les codes d'erreurs

- assez nombreux
- couvrent les besoins principaux
- explications complémentaires souvent requises
  dans le document joint

Les 48 entêtes d'HTTP/1.1

- 4 classes : general/request/response/entity
- 7 groupes logiques :
  - propre à la connexion
  - description du document attaché (entity)
  - négociation des documents
  - informations diverses et perverses. . .
  - authentification
  - conditions d'application
  - cache
- entêtes non standard. . .

Entêtes liés à la connexion

- gestion de la connexion client-serveur
- synchronisation des flux
- établissement du destinataire

`Host Connection Date Upgrade Expect TE Transfer-Encoding Trailer Warning Location Via Pragma`

Entête `Host`

- *nom* du site souhaité (nom DNS, numéro IP)
- uniquement dans la requête (*request-header*)
- **seule entête obligatoire en HTTP/1.1**
  pour les requêtes demandant un document
- nomme le serveur explicitement
- permet un serveur virtuel : partage d'un numéro IP

```
Host:  www.netcraft.com
```


Entête `Connection`

- *general-header*
- option pour une connexion donnée
  du client au premier proxy
- valeurs possibles : `Close`, `Keep-Alive`

Entête `Date`

- date de génération du **message**
- pas très utile si machines non synchronisées !

Entête `Upgrade`

- changement de protocole sur la même liaison
- voir méthode `CONNECT` et réponse `101`

Entête `Expect`

- *request-header field*
- attend un certain retour du serveur
  généralement `100 continue`

Entête `TE`

- *request-header*, Transfer-Coding (!)
- correspondant du *response-header* `Transfer-Encoding`
- exprime les possibilités du client
- valeur : *trailers*. *chuncked* toujours ok.

Entête `Transfer-Encoding`

- *response-header*, pendant de TE
- encodage appliqué pour le transfert
- *chuncked* : découpage en blocs du message
- nécessaire si taille inconnue au début de l'envoi

- indique que d'autres entêtes peuvent apparaître
- pour les transferts par paquets (*chuncked*)

Entête `Warning`

- message d'avertissement destiné à un humain
- ajouté par un proxy par exemple

Entête `Location`

- nouvelle URI pour les redirections

Entête `Via`

- proxy intermédiaires doivent s'enregistrer
- permet le debug, de suivre
- précise les transformations effectuées

Entête `Pragma`

- aspect spécifiques aux implémentations
- permet de faire sa petite sauce dans le cadre HTTP

Entête `Max-Forwards`

- *request-header*
- nombre maximum de proxy intermédiaires
- permet de les identifier ?

  doit être décrémentée par chaque intermédiaires

  réponse retournée si `0`

  le proxy/cache fabrique la réponse !

  utilisé avec la méthode TRACE...
- permet d'éviter les cycles entre proxy ?


```
TRACE / HTTP/1.0
Host: www.iar2m.ensmp.fr
User-Agent: Lynx/2.8.1
Max-Forwards: 2

HTTP/1.1 200 OK
Date: Fri, 02 Nov 2001 18:27:53 GMT
Server: Apache/1.3.14 (Unix)
Content-Length: 70
Content-Type: message/http

TRACE / HTTP/1.1
Host: www.iar2m.ensmp.fr
User-Agent: Squid/1.0
Max-Forwards: 1
```


Description du document attaché

- classe *entity-header*
- par le client (`POST` `PUT`)
- par le serveur (réponses)
- un seul document attaché par message

```
Content-Type Content-Length Content-Location
Last-Modified Content-Language Content-Encoding
Content-MD5 ETag Range Content-Range
```


Entête `Content-Type`

- typage du document retourné ou soumis
- donné par le serveur ou le client
- pas forcemment basé sur le suffixe !
- valeurs définies par le RFC 1590

  Media Type Registration Procedure

  format (case-insensitive) : type `/` sous-type

  aussi utilisé par MIME
- `text/html text/plain text/xml`
  `image/gif image/jpeg message/http`
  `audio/mpeg audio/midi video/mpeg`
  `application/EDIFACT application/msword`

Entête `Content-Length`

- taille du document envoyé
- nécessaire à la synchronisation !
- si non disponible, par morceau (*chuncked*)

Entête `Content-Location`

- URI/URL du document envoyé ou retourné
- en particulier si plusieurs versions (négotiation...)

Entête `Last-Modified`

- date de dernière modification (si approprié)

Entête `Content-Language`

- langage(s) du document
- voir RFC 1766

  `primary-subtag`

  Tags for the Identification of Languages

  administration par IANA (toujours ?)
- exemples : `en fr da en-US`...
- concerne toutes les ressources : texte, image, son...
- plusieurs langages possibles :

  `Content-Language: mi, en`

Entête Content-Encoding

- complète le `Content-Type`
- codage additionnel ⇒ décodage nécessaire
- valeurs possibles
  - `gzip` : GNU zip, RFC 1952, algo LZ77
  - `compress` : Unix, algo LZW
  - `deflate` : zlib (RFC 1950 et 1951)
  - `identity` : rien (valeur par défaut)
- exemple : `Content-Encoding: gzip`
- distinct du `Transfer-Encoding` !

- résumé *digest* du document (cf RFC 1864)
- vérifie intégrité du document reçu
- coût significatif pour le serveur, donc peu utilisé

Entête ETag

- *entity tag* pour la *variante* requise
- objectif : aide des caches
- utilisé par `If-*Match`... (conditions), `Vary` (cache)
- chaîne quotée opaque : égalité si **exactement** égale
- avec préfixe `W/` : faible (égale si équivalente)

Entête Range

- requête de tronçons d'octets
- exemple : `bytes=0-99,200-500`

Entête Content-Range

- tronçon d'un document joint
- morceau sur le total
- exemple : `bytes 0-100/1000`
- si taille indéterminée `*`
- avec réponse `206 Partial Content`

Entêtes de conditions

- pour requêtes conditionnelles
- utilisation : cache ou PUT
- contenu : date ou *entity tag*

Entête `If-Match` avec *entity tag*

typiquement avec un PUT

`*` si ne doit pas exister du tout

Entête `If-None-Match` idem...

mise à jour d'un cache

nouveau document si différent ou n'existe pas

Entête `If-Modified-Since`

date, typiquement mise à jour d'un cache

Entête `If-Unmodified-Since`

date, avec PUT

Entête `If-Range` date ou *entity tag*

retourne le complément ou nouveau document

Informations générales

Entête `From`

adresse *e-mail* du requérant !

Entête `User-Agent`

navigateur du requérant

permet au serveur de contourner des bugs...

Entête `Referer`

page pointante (marketing, facturation)

Entête `Server`

serveur utilisé pour générer la réponse

exemple `Apache/1.3.9 (Unix)`

peut inclure des options (OS, modules apaches)

Atteintes à la vie privée ?

- e-mail du requérant !
- logiciel et version utilisé

  peut inclure la langue, des options...

  le système d'exploitation
- les pages lues (URI dans `Referer`)
- circulation des préférences pour la négociation

  en particulier la langue
- l'identité par authentification (certes !)
- les cookies...

Authentification : RFC 2617

Basic and Digest Authentification

- authentification du `client` par mot de passe
- requise par un serveur, pour un royaume (*realm*)

  domaine d'authentification `Intranet IAR2M`

  destiné à l'utilisateur
- modèle : *challenge/credential*
- deux modes : basic (minable) et digest (médiocre)

`Id: http-auth.tex 1155 2005-03-13 09:42:31Z coelho`

Entêtes HTTP

- 4 (voire 5) entêtes différents
  - *end-to-end* : du client au serveur
    `WWW-Authenticate Authorization`
  - *hop-to-hop* : entre deux intermédiaires sur une chaîne
    `Proxy-Authenticate Proxy-Authorization`
- entête de réponses
  `WWW-Authenticate Proxy-Authenticate`
- entête de requêtes
  `Authorization Proxy-Authorization`

- fonctionnement : requête du serveur vers le client
  requête →
  ← 401 + `WWW-Authenticate` (plusieurs!?)
  requête + `Authorization` →
  ← 200 + résultat
- interprétation des entêtes selon le mode
  Basic : très simple
  Digest : nombreux sous-entêtes...

HTTP Basic Authentification

- introduit par le mot clef `Basic`
  dans la réponse et la requête
- réponse (requête d'authentification), précise le *realm*
  `WWW-Authenticate: Basic realm="Mon royaume"`
- requête (réponse d'authentification)
  `Authorization: Basic chaîne-de-caractères...`
- chaîne : `login:motdepasse` (pas de retour chariot !)
  encodé en **base64** (RFC 2045, MIME part I)
  voir commandes `b64`, `base64`, `uuencode`
  n'utilise que certains caractères (lettres, chiffres)
  non directement lisible, mais pas chiffré !

Exemple d'échanges d'une authentification

```
GET /eval/ HTTP/1.1
Host: palo-alto

HTTP/1.1 401 Authorization Required
Server: Apache/1.3.9 (Unix)
WWW-Authenticate: Basic realm="Intranet IAR2M"
...

GET /eval/ HTTP/1.1
Host: palo-alto
Authorization: Basic dG90bzpsYSB0ZXRlIGEgdG90bw==

HTTP/1.1 200 OK
...
```


Traduction en/de base64

- commande `base64`
- attention au retour chariot : `b64`

```
shell> b64 -e 'toto:la tete a toto'
toto:la tete a toto => dG90bzpsYSB0ZXRlIGEgdG90bw==

prompt> b64 -d 'QWxhZGRpbjpvcGVuIHNlc2FtZQ=='
QWxhZGRpbjpvcGVuIHNlc2FtZQ== => Aladdin:open sesame
```


Remarques sur le mode `Basic`

- circulation du mot de passe en (presque) clair
  exposé à tous les intermédiaires
- niveau de protection **minable**
- très facile à attaquer par un intermédiaire
  devrait être combiné avec du chiffrage (SSL)
- même authentification pour tout le *realm*
- serveur stocke chiffrage du mot de passe
  pas le mot de passe direct, qui circule;-)
- les utilisateurs ont peu de mots de passe...

HTTP Digest Authentification

- système un peu plus sérieux...
  pas de circulation du mot de passe !
- basée sur la notion de *digest*
  calcul un résumé d'une chaîne avec une fonction de hash
  typiquement : DES, MD5, SHA1
  fonctions non facilement inversibles
  résultats de taille constante
- généralement pas implémenté par les navigateurs

Principe du *digest* (résumé)

- client et serveur **partagent secret**
- protocole cryptographique
  le client doit montrer qu'il connait le **secret**
- soit **H** une fonction de hash
  ← **nonce** publique (plus ou moins aléatoire)
  code = H(nonce:secret) →
  compare **code** reçu à **H(nonce:secret)**
- si identique, montre que le client connaît **secret** !
- *nonce* indispensable, sinon **code** constant !
- **secret** ne circule pas !

for the nonce = pour la circonstance

- introduit par le mot clef `Digest`
- système de sous entêtes dans
  `WWW-Authenticate` et `Authorization`
- plusieurs versions (RFC 2069/RFC 2617)
- possibilités offertes :
  nonce du serveur **et** du client
  limitation dans le temps
  limitation à un URI, une entité...
- basée sur la fonction **MD5** : hash de 128 bits
- requiert une bonne implémentation du serveur et du client

Entête de réponse `WWW-Authenticate`

- *requête* d'authentification du serveur
- entête multi-entêtes
  séparés par des vigules
  éventuellement multi-lignes
- plusieurs challenge peuvent être proposées
- après le mode (`Digest`)
  2 sous-entêtes obligatoires : `realm nonce`
  5 facultatifs : `stale algorithm opaque domain qop`
  autres : syntaxe `token=value`

Sous-entête de réponse

realm royaume ciblé, obligatoire
`realm="Paradis"`

nonce chaîne quotée et opaque (pour le client)
- doit être hashée avec le `secret`
- encodage base64 ou hexa suggéré
- choisie par le serveur...
- choix très important pour l'efficacité !
  spécifique à l'entité (etag, uri...)
  inclusion de la date (pour limiter la validité)
- e.g. : `date H(date:etag:private)`

`nonce="492d2347c6235b4d261118"`

domain liste d'URI définissant le **realm**
quotée, relatives ou absolues
`domain="http://palo-alto/eval http://palo-alto/intra"`

opaque chaîne à retourner au serveur sans changement
peut servir de clef (traceur/biscuit/cookie) par exemple ?
`opaque="xfhujejkmznjkfr17cbds"`

stale (rassis, éventé, croupi, vieux) valeur booléenne
vrai si refus précédent du à l'âge du `nonce`
engage à resoumettre avec la nouvelle nonce
sinon : erreur de mot de passe, doit être redemandé
e.g. : `stale=false`

algorithm fonction de hash à utiliser
valeurs : `MD5` ou `MD5-sess`...
seconde valeur pour authentification tri-partite
`algorithm=MD5`

qop *quality of protection*
directive optionnelle (compatibilité avec RFC 2069)
authentification, intégrité... offerte par le serveur
`qop="auth,auth-int"`

autres... autres paramètres
format : `token=quoted-string` ou `token=token`
doivent être ignorés

Exemple de réponse

```
HTTP/1.1 401 Authorization Required
Server: Paradise/2.3 (YaveOS)
WWW-Authenticate: Digest
  realm="lost paradise",
  domain="www.god.org",
  nonce="6629fae49393a05397450978507c4ef1",
  stale=false,
  algorithm=MD5,
  qop="auth",
  opaque="559e81c8edc0ba69877711db088562"
Date: ...
...
```


Entête de requête `Authorization`

- introduit par le mot-clef `Digest`
- requête en *réponse* à la réponse du serveur
  répétition des entêtes et sous-entêtes
- 5 sous-entêtes obligatoires :
  `username realm nonce uri response`
- 5 facultatives : `algorithm cnonce opaque qop nc`
- autres paramètres...

Sous-entête de requête

username nom de l'utilisateur
exemple : `username="lapinot"`

uri répétion de l'URI requise
car peut être modifiée par un proxy dans certains cas !
e.g. : `uri=/eval/`

exemple : `qop=auth-int`

n'existe pas dans la version précédente

cnonce seconde nonce, choisie par le client

permet d'éviter les *plain-text* attaques

requise avec `qop`

`cnonce="unechaineopaque"`

nc *nounce-count*

nombre d'utilisations du nonce (synchronisation/cohérence)

requise avec `qop`

8 chiffres hexadécimaux : `nc=000001fa`

Calcul de la réponse

- chaîne quotée de 32 chiffres hexadécimaux

  en minuscules ; 128 bits
- méthode de calcul spécifiée par le RFC

  basé sur les valeurs des sous-entêtes non quotés

  composition selon *qop*
  - avec : `H(H(A1):nonce:nc:cnonce:qop:H(A2))`
  - sans (RFC 2069) : `H(H(A1):nonce:H(A2))`

- A1 : l'utilisateur

  calcul selon l'algorithme choisi
  - *MD5* : `H(user:realm:pass)`
  - *MD5-sess* : `H(user:realm:pass):nonce:cnonce`

    calculé une seule fois (authentification par un tiers...)

  `pass` passé trois fois par `H` !
- A2 : l'entité, le document

  selon le **qop** choisi
  - *auth* : `method:uri`
  - *auth-int* : `method:uri:H(entity)`

Remarques sur le calcul de la réponse

- réponse dépend de tous les paramètres de la requête
- résultat de taille constante (digest)
- authentification de
  - l'utilisateur A1 (username, realm, pass)
  - la demande A2 (method, uri, qop, entity)
  - du protocole (nonce, nc, cnonce)
- intégrité
  - ajoute `H(entity)`, entitée **soumise**
  - pour un `POST`, pas de modification des paramètres !
- par contre, pas d'intégrité du retour...

  document résultat peut être modifié !

Exemple de requête `Digest`

```
GET / HTTP/1.1
Host: www.god.org
User-Agent: AdamBrowser/1.2
Authorization: Digest
        username="adam",
        realm="lost paradise",
        nonce="6629fae49393a05397450978507c4ef1",
        uri="/", qop=auth,
        nc=00000001, cnonce="f545d129",
        algorithm=MD5,
        opaque="559e81c8edc0ba69877711db088562",
        response="8cc7ede26c042bf6cbb494bf8ed09484"
Date: ...
...
```


Entête `Authentication-Info`

- entête supplémentaire, propre au mode digest
- utilisé par le serveur (réponse)
- **1** sous-entête obligatoire : `nextnonce`
- **4** facultatives : `qop rspauth cnonce nc`

Discussion sur `Authentication-Info`

nextnonce nonce pour le message suivant

doit être utilisé par la requête suivante

empêche le pipelining des requêtes ?

si bon code mais en retard, `stale=true`

rspauth response-auth, authentification inverse !

serveur prouve qu'il connait bien le secret !

même calcul que pour les requêtes, selon `qop`

avec A2 : `:uri` ou `:uri:H(entity)`

peut donc assurer l'intégrité du document retourné !

Authentification Digest entre proxy

- fonctionnement similaire

  entêtes `Proxy-Authentication Proxy-Authorization`

  réponse `407 Proxy Authentication Required`

  complément `Proxy-Authentication-Info`
- un client peut devoir s'authentifier auprès du proxy

  permet de surveiller précisément les employés

  e.g. accès à Internet de chez EDF

Problèmes de sécurité liés à l'authentification

- nombreux;-)
  - faiblesses intrinsèques des méthodes...
  - existence même des deux méthodes...
  - nécessitent de bonne**s** implémentation**s**
    nombreux paramètres libres...
    qualité des générateurs aléatoires...
    aspects facultatifs dans les protocoles
    qualité invisible à l'utilisateur final...
- documentés dans le RFC 2617

Mode Basic

- pas de confidentialité
- un peu d'authentification
- login et mot de passe en presque clair (base64)
- sécurité minable
  pas pour protéger des documents (statistiques ?)
  ou nécessite un chiffrage supplémentaire (SSL ?)
- problème du partage des mots de passe...

Mode Digest

- mieux que CRAM-MD5 (LDAP) ou POP/IMAP (RFC 2195)
- moins que du RSA... (mot de passe connu du serveur)
- pas de confidentialité, mais authentification
- et intégrité (faible : MITM attack...)
- utilisation des nonces :
  - politique à la charge du serveur
    par document, par session, etc.
    limitation éventuelle dans le temps
  - nonce client : selon la (bonne) volonté du client
- limite les fuites aux documents retrouvés

Différentes attaques possibles

replay attack réutilisation des *credentials*

- impossible si un nonce par document/requête
- sinon, selon la qualité des nonces du serveur
  limitation dans le temps
  inclus des infos sur le client (no IP...)
  distinction méthode `GET POST PUT`...

dictionnary attack classique

- requête + réponse : test avec un dictionnaire
  trouve le mot de passe si dans le dico...
- qualité des mots de passe !

man in the middle attack argh !

- intermédiaire silencieux
  s'adresse au serveur et au client
  peut modifier les requêtes à volonté...
- affaiblissement des niveaux de protection
  `auth`, `auth-int` → `auth`
  `Digest` → `Basic` !
  (client doit avertir l'utilisateur ? compétence ?)

chosen plaintext attack

- choix du texte chiffré/hashé peu aider...
- conjonction avec un dictionnaire précalculé
- impossible si nonce client (`cnonce`)

brute force attack

- collecte de paires requêtes/réponses avec même nonce
- attaque avec un dictionnaire...
- impossible si nonce client `cnonce` ?

spoofing du serveur

- demande d'authentification d'un autre serveur !
- importance de l'unicité globale du *realm*
  inclusion de l'adresse http ? `domain`
- besoin d'authentifier le serveur pour le client...

attaque du serveur

- stockage des mots de passe sur le serveur
  nécessaire, contrairement à UNIX
- ne doit pas être compromis...

Conclusion sur l'authentification HTTP

- authentification seulement (confidentialité, intégrité ?)
- deux modes : `Basic` et `Digest`
- qualité dépend
  des implémentations des clients et serveurs
  Digest de Netscape 4.7 = Basic !
  des aspects facultatifs du protocole
  de la compétence de l'utilisateur
- attaques assez aisées...

- exemples occurrences :
  - redirection `300 Multiple Choice`
  - erreur `405 Method Not Allowed`
- sélection de documents équivalents
  - formats (gif, mpeg)
  - langues différentes
- entêtes :

  `Allow` : entity-header

  `Accept-Ranges` : response-header

  `Accept`... : request-header

- méthodes autorisées sur un document
- aussi sur un serveur (requête `OPTIONS`)
- retournée avec erreur `405`
- suggestion du client avec un PUT

Entête `Accept-Ranges`

- response-header
- autorise requêtes `Ranges:` du client
- types de tronçons acceptés : `bytes` ou `none`

Entête `Accept`

- types de media acceptés
- plus une notion de préférence
- syntaxe :
  - entrées séparées par des `,`
  - type/sous-type `*/* audio/* text/html`
  - indique les types acceptés, par défaut tous !
  - qualité facultative (défaut 1) : `; q=0.3`

    indique le degré de préférence
  - éventuellement autres paramètres spécifiques

```
Accept: text/plain;q=0.5, text/html,
  image/jpeg, image/gif;q=0.5, image/tiff;q=0.1
```



```
GET /index HTTP/1.0
Accept: text/plain;q=0.9, text/html;q=0.5


HTTP/1.1 200 OK
Date: Wed, 02 Oct 2002 18:33:29 GMT
Server: Apache/2.0.42 (Unix)
Content-Location: index.plain
Vary: negotiate,accept
Content-Length: 41
Content-Type: text/plain; charset=ISO-8859-1

...
```


Entête `Accept-Charset`

- syntaxe similaire à `Accept`
- charactères acceptés (selon fontes, langues...)

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

Entête `Accept-Encoding`

- similaire à `Accept`
- encodages de l'entité... tous/autres : `*`

```
Accept-Encoding: gzip, identity;q=0.9, *;q=0
```


Entête `Accept-Language`

- similaire à `Accept`
- langues et variantes acceptées, préférences
- syntaxe : `langue` ou `langue-variante`

  noms 1 et 8 caractères, voir RFC 1766

  administrés par IANA (?)
- exemples : `en en-US en-gb fr`

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



```
GET /index.html HTTP/1.0
Accept-Language: de;q=0.6, fr;q=0.8, en;q=0.5


HTTP/1.1 200 OK
Content-Location: index.html.fr
Vary: negotiate,accept-language
Content-Length: 35
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-Language: fr

...
```


Conclusion

- permet de négocier différents aspects

  langues, formats, codages...
- implémentation ?
- informations privées

- *proxy* (=mandataire) ou *gateway* imposé
- objectif **performance**
  réduction de la bande passante requise
  pour une population **homogène**, accès souvent communs
- contrôle, filtrage, audit des accès
  login/mot de passe spécifique (cf authentification)
  détection éventuelle de virus (pas fiable ?)
- interdiction de certains sites ou types de documents
  porno, pub, bourse... selon politique locale
  remplacement par des images blanches par exemple
  détection d'image pornographique au vol ?

- `https` (HTTP/SSL) : liaison directe
  contournement du cache/filtre ?
- propriété de transparence
  même résultats avec et sans le cache
- les niveaux de caches
  - **mémoire** et **disque** au niveau du navigateur !
  - au niveau d'un **site**
  - au niveau d'un **fournisseur** *(provider)*

- support du protocole
  - meta-informations sur les documents
    entêtes de taille, digest, tag...
  - requêtes et réponses spécifiques
  - mécanismes conditionnels
  - critères de cachabilité des documents
    requiert de la souplesse...
- implémentations
  - proxy (explicite) ou gateway (caché, au niveau routeur)
  - *shared* (navigateur) ou *private*
  - logiciels : `squid apache/mod_proxy`...

2 Mécanismes

expiration

- politique de vieillissement des documents
- entêtes `Age Expires` explicites
- modes de caculs...

validation

- décider si un document est valide
- *weak/strong* validation faible ou forte
- vérification : requêtes conditionnelles
- variations possibles selon préférences (`Vary`)
- mais aussi `Last-Modified ETag`

Entête `Age`

- *response-header*
- age estimé du document en secondes
- généré par un cache

Entête `Expires`

- date limite d'un document

Entête `Vary`

- liste des entêtes pouvant faire varier la réponse

```
Vary: Accept-Language
```


Entête `Cache-Control`

- *general-header* : client ou serveur
- contrôle des caches intermédiaires
- nombreuses sous directives/entêtes
  - requêtes/réponses
  - gestion de la cachabilité, des âges...

Sous-entêtes de réponses

public réponse cachable

private=... réponse privée, cachable par un cache privé

no-cache=... entêtes à revalider, ou pas de cache

no-store pas de stockage, info sensibles

no-transform pas de changement de type...

must-revalidate *shared* et *private*

proxy-revalidate *shared* seulement

max-age=s pour forcer une revalidation

s-maxage=s idem, cache *shared*

Sous-entêtes de requête

no-cache clair !

no-store idem !

max-age=s

max-stale accepte un document agé

min-fresh=s demande un document frais (*fresh*)

no-transform

only-if-cached document si présent, sinon tant pis

Remarques sur les caches

- maintien de la cohérence...
  expiration et validation des documents
- utilité pour les sites :
  paiement au *hit* ;-)
  importance des statistiques d'utilisation...
  expiration immédiate des documents !
- qui doit définir tout ça ?
  auteur du site ? mainteneur ?

Requêtes à un proxy (éventuellement cache)

- syntaxe particulière
- URI complète dans la requête
- requêtes/réponses spécifiques
  authentification : `Proxy-*` + `407`
  traçage : `Via`

Les Cookies (biscuits)

- maintien d'une état : notion de session
  important pour le *shopping* !
  éventuellement pour de l'authentification (basique)
- nécessite la coopération **active** du client
- requête/réponse inversée, comme authentification
- proposé dans le RFC 2109 (initiative de netscape)
- entête `Set-Cookie` : response-header
- entête `Cookie` : request-header
- peut mettre/envoyer plusieurs biscuits

Entête `Set-Cookie`

- proposition du serveur de mettre un cookie
- si le client l'accepte
- syntaxe : `nom=valeur`
  par exemple `TOTO=fjioaf219denwdfwe`
- et options après `;` et séparées par `;`
- éventuellement multiples

Options de `Set-cookie` (v1)

Comment=... commentaire éventuel (pour qui ?)

Domain=... précise le domaine de validité
commence toujours pas un `.`

Max-Age=s durée limite de validité du biscuit
par défaut, disparaît à la fin de la session

Path=... sous ensemble des URL concernées

Secure conseille de garder en sécurité le cookie

Version=1 version, **doit** être spécifiée

Mais aussi, ancienne version (v0)

- **implémentée** par IE, netscape...
- pas de `Version`
- **expires** date limite de validité
  plus logique ?

Exemple de réponse `Set-cookie`

```
Set-Cookie: vgnvisitor=D06EM0000VM000l0fnNL564KRe;
  path= /; expires=Saturday, 06-Sep-2014 23:50:08 GMT

Set-Cookie: RMID=c036acc038245920;
  expires=Friday, 31-Dec-99 23:59:59 GMT;
  path=/; domain=.tf1.fr

Set-cookie: Customer="Lapinot";
 Version="1"; Domain=".trondheim.org"; Path="/"
```


Entête Cookie

- réponse gracieuse du client
  si implémenté
  si autorisé/validé par l'utilisateur (préférences)
  si domaine et path correspondent
- syntaxe : `Cookie: $Version="1"; nom=valeur`
- options après `;` et entre `;`
  `$Path=...` et `$Domain=...`
  `$` empêche de confondre avec un cookie
- éventuellement plusieurs cookies retournés

Sécurité des biscuits pour le client

- bien vérifier le `domaine` cible
- bien vérifier le `path`
- bien vérifier la date d'expiration `max-age`

Exemple de requête `Cookie`

```
Cookie: Hello=10.2.14.100.23046972757787726

Cookie: $Version="1";
  Customer="Lapinot";
    $Domain=".trondheim.org";
    $Path="/";
  Album="Pour de vrai";
    $Domain=".trondheim.org";
    $Path="/albums/";
```


Remarques sur les cookies

- caches : **no-cache=set-cookie** nécessaire
- contenu du biscuit
  - pas réellement important !
  - clef pour les informations associées
- biscuits définis : `.netscape/cookies`
- passage aux CGI par environnement
- version décrite non implémentée en général
  mais version précédente : `expires`
  même principe pour `Cookie`, sans la version
- problème : connexions parallèles du client...

Conclusion sur les cookies

- maintient d'un état
- nécessite la coopération active du client
  sinon : path pour chaque client !
  module `mod_session` du serveur apache
- ne doit pas contenir d'information...
  clef pour retrouver les informations sur le serveur
  normallement doit être opaque et aléatoire
  peut être *volé* facilement
- configuration de la politique des navigateurs

Conclusion sur HTTP

- autres entêtes
- versions...
- SSL
- extensions

Autres entêtes

- ignorés si inconnus
- entête `X-pad`
  commentaires ajoutés par **apache**
  eg : *avoid browser bug* : passage en `HTTP/1.0`
  devrait utiliser `Warning` ?

Extension de HTTP : WebDAV

Distributed Authoring and Versioning

- ajouts de méthodes diverses `RENAME DELETE LOCK`...
- manipulations de fichiers, vérouillage, versions...
- clients WebDAV : IE, DreamWaever, `cadaver nd`

Différences entre HTTP/1.0 et HTTP/1.1

- versions successives de HTTP
  bon support de HTTP/1.1
- virtualisation des serveurs :
  `Host:` obligatoire
- connexions multi-requêtes par défaut
  `Connection: close` pour fermer

SSL

- cryptographie et authentification au niveau TCP/IP
- conséquence : pas de serveurs virtuels...

Extensions

- webdav
- shttp

Implémentation de serveurs WEB

versions libres/semi-libres/commerciales

Apache basé sur NCSA

IIS de Microsoft

iPlanet Netscape

Roxen Challenger

thttp

Jigsaw implémentaton JAVA du W3C

khttpd directement dans l'OS pour requêtes simples

Évaluation des performances

- tests standards : *WebSTONE, SpecWeb99 WebBench*
- commande `ab` *Apache Bench*
- évaluation difficile
  - OS : système de fichier, TCP/IP, processes...
  - paramétrabilité/paramétrage
  - serveur lui-même
  - type de site : statique, dynamique...
  - tuning...

SpecWeb99 de SpecBench `http://www.spec.org/`

- organisation spécialisé dans la production de benchmarcks
  CPU, Web, JVM, JBB, Mail...
- seconde version (SpecWeb96, SpecWeb99 v1.02)
- résultats sur différents matériels
  de différents logiciels : Zeus, IIS, Tux
  précision de tous les paramétrages du test
- mesure la charge (délais, nombre de connexions servies...)
- exprimé sous forme d'un entier synthétique...

Compaq ProLiant DL380, Microsoft IIS 5.0, SPECweb99=1098

IBM RS/6000 7044-270, Zeus 3.3.6, SPECweb99=2175

WebSTONE de Mindcraft

`http://www.mindcraft.com/webstone/`

- entreprise spécialisée également : bench répertoires, web...
- version actuelle 2.5
- résultats : connexions, débit pour CGI, HTML...
- Compaq ProLiant 5000 1p, Win NTS 4.0, Netscape ES 2.0
  connexions : HTML 904/s (500 clients), CGI 29/s (20)
  débits : HTML 133Mb/s (500), CGI 4.45Mb/s (30)

Autre benchmarks

- WebBench 4.0 par ZDnet/E testing labs
  `http://www.webbench.com/`
  freeware de test pour serveur web
- NetBench, même fournisseur
- Netperf `http://www.netperf.org/`
  tests orientés réseau (TCP/IP, UDP/IP...)
- `http://www.benchmarkfactory.com/`

Statistiques sur les serveurs HTTP

`http://www.netcraft.com/survey/`
`http://www.securityspace.com/`

- d'après entête `Server` (selon configuration)

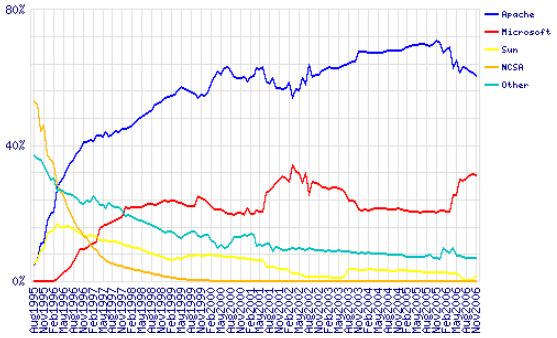
```
Server:  Apache/1.3.9 (Unix) PHP/4.0B2
Server:  Apache/2.0.48 (Unix) DAV/2 PHP/5.0.1
```

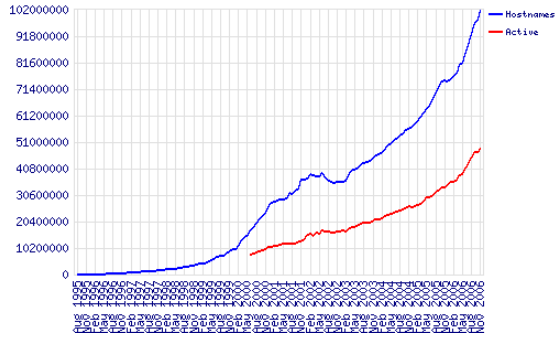
- statistiques par domaines :
  - serveurs HTTP
  - serveurs SSL avec certificats valides
  - utilisation des modules pour apache
  - dénombrement pour les serveurs virtuels

Marché des serveurs HTTP par noms NETCRAFT

Date	Total	Apache	Microsoft	Netscape
09/1995	20 000	5%	0%	4%
09/1996	350 000	43%	8%	14%
09/1997	1 364 714	44%	17%	7%
09/1998	3 156 324	52%	22%	8%
09/1999	7 370 929	58%	22%	8%
09/2000	21 166 912	60%	19%	7%
09/2001	32 398 046	59%	27%	4%
09/2002	35 756 436	60%	29%	1%
09/2003	43 144 374	64%	24%	3.5%
09/2004	54 407 216	68%	21%	3%
09/2005	71 723 098	69%	20%	2.6%
09/2006	96 854 877	63%	30%	0%

par numéro, 53% Apache et 35% Microsoft





Marché des clients HTTP

- information collectable par les serveurs !

  entête `User-Agent:` de HTTP

  statistiques possibles sur les versions. . .
- dépend des sites webs. . .
- immense majorité de Internet Explorer (Microsoft)

Future version de HTTP

- HTTP-NG (New Generation)
- discussions en cours, IETF

List of Slides