II. Protocole HTTP

1. Rappels modèle client-serveur
2. Architectures des connexions HTTP
3. HTTP 0.9
4. HTTP 1.0
5. HTTP 1.1
   (a) Connexion persistante
   (b) Fragmentation des réponses
   (c) Sélection du contenu
   (d) Négociation de contenu
   (e) Authentification de l'accès
   (f) Proxy/cache/miroir WEB
6. Aspects liés à la sécurité

1. Rappels modèle client-serveur

Les bases

- Protocoles de base de l'Internet : TCP/IP et UDP/IP
  - accès à ces protocoles via les sockets
  - niveau d'adressage supplémentaire : numéro de port
- Service réseau identifiée par : @IP, numéro de port
  - attribution des no. des ports aux services : IANA
    * ports "privilégiés" : 0–1023

```
http 80/tcp     World Wide Web HTTP
http 80/udp     World Wide Web HTTP
```

    * ports "enregistrés" : 1024–49151

```
afs 1483/tcp    AFS License Manager
afs 1483/udp    AFS License Manager
```

    * ports "dynamiques/privés" : 49152–65535

```
ircd 6667/tcp   # Internet Relay Chat
ircd 6667/udp   # Internet Relay Chat
```

- Observation sous les systèmes Unix :
  - `netstat`
  - `/etc/services`

Caractéristiques des clients et des serveurs

Programme client :

- application conventionnelle, devient client lorsqu'un accès distant est nécessaire
- invoqué par l'utilisateur, il ne fonctionne que pour 1 session
- il s'exécute localement sur un ordinateur personnel
- il établit de manière active le contact avec le serveur
- il peut accéder à $n > 0$ services, mais 1 accès distant à la fois
- il ne nécessite pas de matériel spécial ni de S.E. complexe

Programme serveur :

- programme spécialisé pour un service
- il peut traiter plusieurs clients distants à la fois
- invoqué automatiquement par le S.E., sessions multiples
- il s'exécute sur un ordinateur partagé
- il attend de manière passive un contact depuis clients distants
- invoqué par des clients qques, mais il offre 1 service
- il requiert du matériel puissant et un S.E. évolué
  $\Rightarrow$ *ordinateur de classe serveur* ou *machine serveur*

Lancement et exécution de serveurs

Lancement :

- en utilisant le super-serveur `inetd`
  - `inetd` écoute plusieurs ports à la fois
  - demande de connexion sur $p \Rightarrow$ lancement du serveur attaché à p
  - le serveur meurt quand le client se déconnecte
  - configuration du lancement en `inetd.conf`

```
telnet stream tcp nowait root /sbin/telnetd telnet
```

- en daemon (serveur autonome)
  - le serveur écoute le port attaché
  - peut lancer plusieurs files d'exécution (*thread*)

Exécution :

- séquentielle (rare)
- simultanée (*fork* ou *thread*)

Utilisations du modèle client-serveur

Dans la plupart des services Internet :

- Prise de contrôle à distance
- Communication entre individus
- Diffusion de l'information ⇒ serveurs d'informations :
  - sites archives (ftp anonyme)
  - Finger, Whois, Netfind (recherche d'une personne)
  - Archie (recherche d'un logiciel par expr. reg.)
  - Gopher (recherche guidée par menus textuels)
  - Wais (recherche par mots clés et domaines)
  - Web (hypertexte distribué)

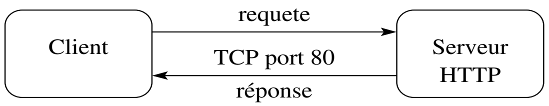
Philosophie du Web :

- partage d'information à l'échelle mondiale
- importance de la convivialité
- inter-opérabilité :
  - architectures hétérogènes (Intel, Sparc, HP, etc.)
  - S.E. hétérogènes (Unix, MS-DOS, Macintosh, etc.)
  - logiciels clients et serveurs différents

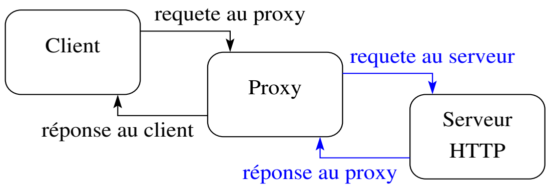
Standards du Web : HTML, URL, HTTP, MIME

2. Architectures pour les connexions HTTP

- Connexion directe client-serveur

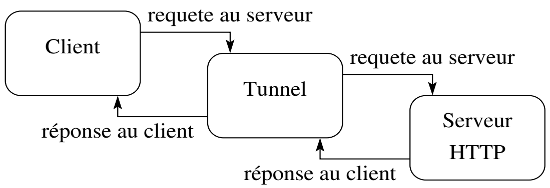


- Connexion client-proxy-serveur



Eventuel traitement sur le proxy ⇒ firewall, cache

- Connexion client-serveur via un tunnel



Acheminement du trafic HTTP sur une liaison non TCP/IP

- Combinaisons (proxy + tunnel)*

3. HTTP/0.9

- Inventé avec HTML par Tim Berners-Lee
- Première implémentation “officielle” de HTTP
- Une session client :
  1. connexion au serveur sur port TCP 80
  2. envoi requête (uniquement méthode GET)
     - normale

       `GET document.html CRLF`
     - recherche d’un ou plusieurs termes

       `GET document.html?help+me CRLF`

       si le document contient le tag ISINDEX HTML
  3. réception du document (texte uniquement)
  4. fin connexion = fin document
- Implémentation facile pour le client ou le serveur
- Limitations :
  - documents texte uniquement
  - pas d’informations du serveur sur le document
  - impossible d’envoyer des informations au serveur

Exemple connexion HTTP/0.9

```
> telnet verif.liafa.jussieu.fr 80
Trying 132.227.81.70...
Connected to verif.liafa.jussieu.fr.
Escape character is '^]'.
GET /~haberm/index.html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">

<HTML>
<HEAD>
...
</BODY>
</HTML>
Connection closed by foreign host.
```

4. HTTP/1.0

Développé entre 1992–1996, RFC 1945 paru Mai 1996

- Introduction des entêtes MIME dans
  - la requête = préférences du client, p.e. type du document, date de modification,...

```
Full-Request = Request-Line
               *( General-Header |
                  Request-Header |
                  Entity-Header   ) CRLF
               [ Entity-Body ]
Request-Line = Method SP Request-URI SP
               HTTP-Version CRLF
```

  - la réponse = informations sur la ressource, p.e. code d'état, type du contenu, ...

```
Full-Response = Status-Line
                *( General-Header  |
                   Response-Header |
                   Entity-Header    ) CRLF
                [ Entity-Body ]
Status-Line = HTTP-Version SP Status-Code SP
              Reason-Phrase CRLF
```

- Nouvelles méthodes
  - HEAD : demande d'informations sur la ressource
  - POST : entrées (non-limités) pour la ressource

4. HTTP/1.0 (cont.)

- Requêtes conditionnelles :
  - `If-Modified-Since`
  - cache client
- Code d'état pour la réponse :
  - succès `2XX`, p.e. `200 OK`
  - redirection `3XX`, p.e. `302 Found`
  - erreur client `4XX`, p.e. `404 Not Found`
  - erreur serveur `5XX`
- Accès restreint à une ressource
  - `401 Unauthorized status`
  - authentification basic : username, password
  - permet de suivre un client

Nouveaux besoins, nouvelles limitations

- rapidité du transfert et de la connexion
- traiter serveurs avec domaines virtuels
- introduction des caches Web

Exemple 1 : connexion normale HTTP/1.0

```
> telnet verif.liafa.jussieu.fr 80
Trying 132.227.81.70...
Connected to verif.liafa.jussieu.fr.
Escape character is '^]'.
GET /~haberm/index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 09 Oct 2001 07:17:50 GMT
Server: Apache/1.3.19 (Unix)
Last-Modified: Tue, 11 Sep 2001 12:58:04 GMT
ETag: "4a20d-cac-3b9e0a5c"
Accept-Ranges: bytes
Content-Length: 3244
Connection: close
Content-Type: text/html
Content-Language: fr


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
...
</BODY>
</HTML>
Connection closed by foreign host.
```


Exemple 2 : demande d'une page protégée HTTP/1.0

```
> telnet liafa1.liafa.jussieu.fr 80
Trying 132.227.81.128...
Connected to liafa1.liafa.jussieu.fr.
Escape character is '^]'.
GET /~sighirea/cours/deug1/index.html HTTP/1.0

HTTP/1.1 401 Authorization Required
Date: Tue, 09 Oct 2001 07:22:20 GMT
Server: Apache/1.3b3
WWW-Authenticate: Basic realm="Enseignants"
Connection: close
Content-Type: text/html

<HTML><HEAD>
<TITLE>401 Authorization Required</TITLE>
</HEAD><BODY>
<H1>Authorization Required</H1>
This server could not verify that you
are authorized to access the document you
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.<P>
</BODY></HTML>
Connection closed by foreign host.
```

5. HTTP/1.1

- Développé entre 1995–1996 dans IETF/HTTP WG
- Dernier RFC 2616 paru 1999
- Compatible HTTP/1.0 mais améliorations et corrections
- Nouveautés ...

Nouvelles méthodes

- PUT : demande de mise à jour d'un document sur le serveur, $\neq$ POST
- DELETE : demande d'effacer (déplacer) un document sur le serveur $\Rightarrow$ problèmes d'autorisation
- OPTION : informations sur les capacités du serveur
- TRACE : fonction d'echo pour le debug
- CONNECT : réservé pour proxy-tunnel

Exemple méthode OPTIONS

```
> telnet liafa1.liafa.jussieu.fr 80
Trying 132.227.81.128...
Connected to liafa1.liafa.jussieu.fr.
Escape character is '^]'.
OPTIONS / HTTP/1.1
Host: liafa1.liafa.jussieu.fr

HTTP/1.1 200 OK
Date: Tue, 09 Oct 2001 07:41:11 GMT
Server: Apache/1.3b3
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
```

Ou, si le serveur le permet ...

```
> telnet liafa1.liafa.jussieu.fr 80
Trying 132.227.81.128...
Connected to liafa1.liafa.jussieu.fr.
Escape character is '^]'.
OPTIONS * HTTP/1.1
Host: liafa1.liafa.jussieu.fr

HTTP/1.1 403 Forbidden
...
```

5.a Connexions persistantes

Une fois la connexion établie, elle reste ouverte jusque le client (ou le serveur) demande sa fermeture = comportement par défaut !

- Durant une connexion, le client peut faire plusieurs requêtes.
- Le client n'est pas obligé d'attendre la réponse pour envoyer une nouvelle requête.
- Le serveur doit répondre dans le même ordre.

⇒ *pipelining* des requêtes et des réponses

Attention aux opérations *non-idempotentes* !

Moyens :

- fermeture de connexion : timeout, `Connection: close`
- longueur requête/réponse : `Content-Length:`
- négociation du maintient d'une connexion
  - par le client : `Expect: 100-continue`
  - par le serveur : `HTTP/1.1 100 Continue`
- éviter les envois inutiles : “binary exponential backoff”

Exemple connexion persistante HTTP/1.1

```
> telnet liafa1.liafa.jussieu.fr 80
Trying 132.227.81.128...
Connected to liafa1.liafa.jussieu.fr.
Escape character is '^]'.
GET /~sighirea/index.html HTTP/1.1
Host: liafa1.liafa.jussieu.fr

HTTP/1.1 200 OK
Date: Tue, 09 Oct 2001 07:31:48 GMT
Server: Apache/1.3b3
Last-Modified: Thu, 14 Jun 2001 08:23:06 GMT
ETag: "5b17c-10e5-3b28746a"
Content-Length: 4325
Accept-Ranges: bytes
Content-Type: text/html


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
...
</BODY>
</HTML>

GET /~sighirea/photo.jpg HTTP/1.1
Host: liafa1.liafa.jussieu.fr
Connection: close
```

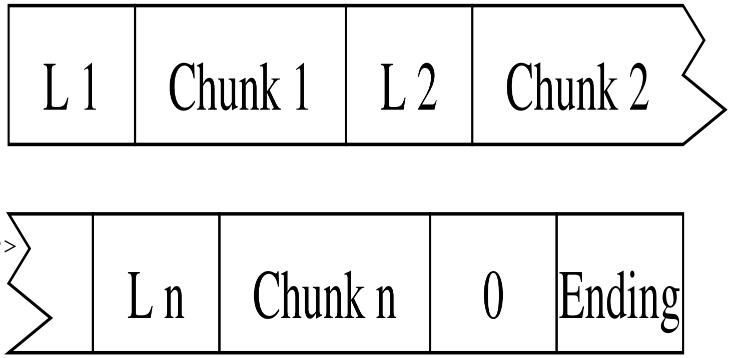
```
HTTP/1.1 404 Not Found
Date: Tue, 09 Oct 2001 07:35:29 GMT
Server: Apache/1.3b3
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

a2
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /~sighirea/photo.jpg was not found on this server.<P>
</BODY></HTML>

0

Connection closed by foreign host.
```


5.b Fragmentation des réponses

Avec le comportement par défaut, le serveur doit spécifier la taille de la ressource envoyée en réponse (`Content-Length`).

Problème : les ressources dynamiques.

Solution : “chunked encoding”



- = envoyer la ressource par “petits morceaux” de taille connue.
- dans l’en-tête `Transfer-Encoding: chunked`

5.c Sélection du contenu (“byte range”)

Problème :

- connexion interrompue durant un transfert
- ressource à transférer trop grande

Solution : transfert sélectif d'une ressource

Moyens :

- en-tête réponse (serveur) :

```
Accept-Ranges: bytes
Accept-Ranges: none
```

- code d'état (serveur) :

```
206 Partial content
```

- en-tête requête (client) :
  - spécification des portions à transmettre

```
Range: bytes=0-500
Range: bytes=10-
Range: bytes=-500
Range: bytes=0-99,500-510,4000-
```

  - requête conditionnelle

```
If-Range: Thu, 13 Jun 2001 08:23:06 GMT
```


Exemple : combinaison

```
GET /~sighirea/index.html HTTP/1.1
Host: liafa1.liafa.jussieu.fr
Range: bytes=0-2,-2
If-Modified-Since: Thu, 13 Jun 2001 08:23:06 GMT

HTTP/1.1 206 Partial Content
Date: Tue, 16 Oct 2001 12:17:00 GMT
Server: Apache/1.3b3
Last-Modified: Thu, 14 Jun 2001 08:23:06 GMT
ETag: "5b17c-10e5-3b28746a"
Content-Length: 183
Content-Type: multipart/byteranges; boundary=3bcc253c7c42

--3bcc253c7c42
Content-type: text/html
Content-range: bytes 0-2/4325

<!
--3bcc253c7c42
Content-type: text/html
Content-range: bytes 4323-4324/4325


--3bcc253c7c42--
```

5.d Négociation de contenu

Un serveur peut contenir plusieurs représentations pour une ressource, p.e. messages d'erreur en français/anglais, figures GIF ou TIFF, etc.

Négociation dirigée par le serveur

Algorithme basé sur deux paramètres :

- les représentations disponibles pour la ressource
- la requête du client contenant les représentations acceptées et leur *facteur de qualité* :

```
Accept: audio/*;q=0.2, audio/basic
Accept: text/plain;q=0.5, text/html,
        text/x-dvi;q=0.8, text/x-c
Accept-Charset: iso-8859-5,
                unicode-1-1;q=0.8
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Language: fr, en-gb;q=0.8, en;q=0.7
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

ou l'adresse IP du client.

Le serveur peut informer le client (et le cache) de ses critères de sélection :

```
Vary: User-Agent, Accept-Language
Vary: *
```


Négociation dirigée par le client

Si le serveur ne peut/veut pas faire un choix, il informe le client par une des réponses :

```
300 Multiple Choices
406 Not Acceptable
```

Le client (*user-agent*) peut spécifier un choix en envoyant une (2ème) requête contenant ses préférences.

Négociation transparente

Si un proxy/cache est interposé entre le client et le serveur :

- il connaît les représentations du serveur
- il connaît la requête du client

Hôtes virtuels (“virtual host”)

But : permettre à un même serveur, sur une même machine, de simuler plusieurs serveurs sur des machines différentes.

Exemples :

- même nom de domaine : help.data.com et cgi.data.com
- nom de domaine différent : www.help.net et www.cgi.net

Deux façons de faire :

- Hôtes virtuels basés sur une adresse IP
  - utilise différentes adresses IP de la même machine
- Hôtes virtuels nommés
  - utilise la même adresse IP, mais des noms différents
  - en-tête requête `Host:`

```
GET /index.html HTTP/1.1
Host: help.data.com:80
```

obligatoire en HTTP/1.1, sinon erreur. Dans les versions ultérieures, utilisation des URI absolus.

5.e Authentification de l’accès

Problème : on veut qu’un ensemble de pages (*procted realm* ou *realm*) soit accessible qu’aux clients qui sont autorisés.

Solution 1 : utiliser l’adresse IP du client.

Solution 2 : (RFC 2617) utiliser un schéma défi-réponse :

1. si le client demande un page $\in$ realm, le serveur répond

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: <challenge>
```

2. le client doit faire une nouvelle requête avec sa réponse

```
Authentication: <credentials>
```

3. le serveur analyse la requête et si elle correcte, il envoie la page (en clair par défaut) et l’accès est donné au realm entier, sinon retour à 1.

Deux types de défis : `Basic` et `Digest`

Authentification “Basic”

La réponse au défi est *base64*(userid:password)

Exemple :

- défi serveur :

```
WWW-Authenticate: Basic realm="Enseignants"
```

- réponse client : “Aladin:open sesame”

```
Authorization: Basic QWxhZGRpbjpvcGVuIHN1c2FtZQ==
```

Problèmes : *base64* c'est une technique d'encodage, pas une méthode de chiffrement.

Exemple : L'interception de l'en-tête réponse ci-dessus permet d'obtenir le nom de l'utilisateur et le password en utilisant

```
perl -MMIME::Base64 -e
'print decode_base64("QWxhZGRpbjpvcGVuIHN1c2FtZQ==")'
```

qui renvoie “Aladin:open sesame”.

On utilise ici le module Perl `MIME::Base64` disponible librement à `http://www.cpan.org/`

Authentification “Digest”

Améliore le schéma “Basic” en cryptant la réponse et en utilisant une valeur “unique” (*nonce*) par défi.

Algorithme de cryptage : MD5

- string $\longrightarrow$ 128 bits
- fonction difficilement réversible

Exemple :

- défi serveur :

```
WWW-Authenticate: Digest realm="Enseignants",
                  domain="/index.html",
                  nonce="a93b26",
                  algorithm="MD5", qop="auth"
```

- réponse client :

```
Authorization: Digest username="Aladin",
               realm="Enseignants",
               uri="/index.html", qop="auth",
               cnonce="0efff1", nc=01,
               response="<rd>"
```

où `<rd>` est calculé comme suit :

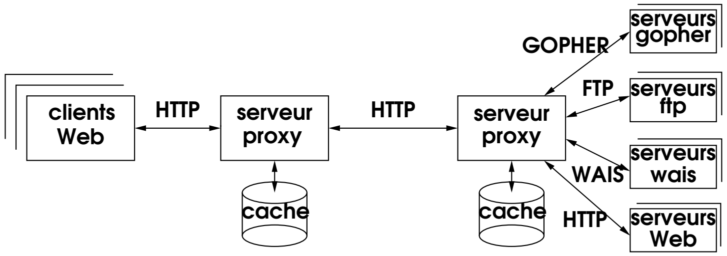
```
<A1> = MD5 (Aladin:Enseignants:open sesame)
<A2> = MD5 (GET:/index.html)
<rd> = MD5 ( <A1>:a93b26:01:0efff1:auth:<A2> )
```

Proxy/cache/miroirs WEB

Buts généraux :

- Améliorer le temps d'accès aux informations distantes pour les utilisateurs locaux.
- Diminuer le trafic global de l'Internet.
- Diminuer la charge des serveurs les plus populaires.
- Sécuriser/filtrer les accès.

Moyen : interposer des proxy/cache entre clients et serveurs



Un *proxy* est un intermédiaire : il se comporte comme un serveur vis-à-vis de ses clients et comme un client vis-à-vis des serveurs.

Pourquoi utiliser un proxy ?

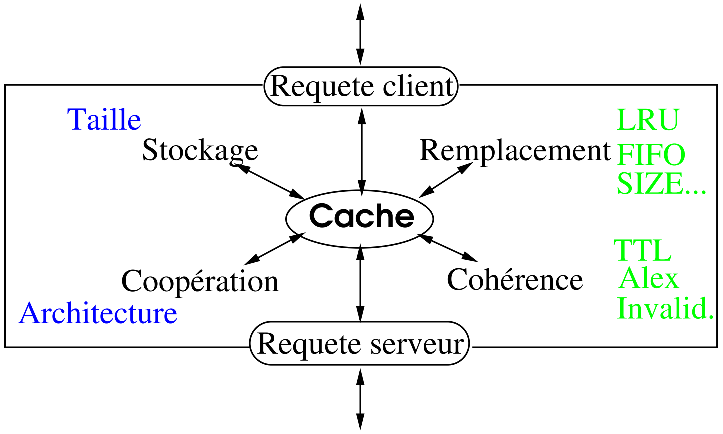
- Pour des raisons de sécurité
  - *firewall* : connexion à l'Internet sans visibilité de l'extérieur pour les machines locales
  - accélérateur http (p.e. Squid) : vu comme un unique serveur, mais masque plusieurs serveurs
- Pour contrôler l'utilisation de l'Internet
  - on peut filtrer les protocoles
  - on peut filtrer les adresses
- Pour soulager le réseaux : cache
- Pour donner accès à des protocoles que les clients Web ne connaissent pas (p.e. Wais).

Il existe des proxies non Web, p.e. CORBA.

Principe de fonctionnement des caches Web

- Il garde localement une copie des documents qu'il transfère (avec leur identification).
- Si on demande au cache un document qu'il possède déjà, il peut le délivrer sans effectuer une requête au serveur.

Architecture d'un cache Web :



- Replacement : quel document expulser quand le cache est plein ?
  - LRU : le moins récent utilisé
  - FIFO : le premier arrivé
  - SIZE : le plus gros
- Cohérence : éviter de conserver des documents périmés.
  - TTL : durée de vie maximale (24h)
  - Alex : durée de vie prop. à leur age (10%)
  - Invalidation : explicite par le serveur
- Coopération : tirer parti de caches proches.
  - Hiérarchies de caches. Exemple : Renater-cache.
  - Coopération d'égal à égal : sondage des voisins.

Caching en HTTP/1.1

- En général, uniquement les réponses à GET et HEAD sont “cacheable”.
- En-tête requête/réponse : `Cache-Control`
- Directives :
  - Est cacheable (S) : public, private, no-cache
  - Est stockable (S ou C) : no-store
  - Quel age peut avoir : max-age (S ou C), max-stale, min-fresh (C). Exemple : (temps en secondes)

    ```
    Cache-Control: max-age=300, max-stale=120
    ```
  - Ne pas dépasser cet age (S) : must-revalidate, proxy-revalidate
  - Ne pas transformer (S ou C) : no-transform
  - Ne pas faire une nouvelle requête (C) : only-if-cached

Caches clients

- De type persistant sur le disque : pratique mais gourmand !
- En mémoire : navigation avant-arrière.

Miroirs

But :

- Accélérer les accès aux sites les plus populaires
- Diminuer le trafic global de l’Internet

Moyens : créer des copies des sites stratégiques à plusieurs endroits.

- Un logiciel de miroir est un robot : il navigue automatiquement sur le Web.
- On donne une URL de départ au robot et des règles pour savoir où arrêter la récursion.
- Le robot extrait les liens des documents qu’il récupère pour explorer récursivement un serveur.

Logiciels : webcopy, wget, w3mir

Logiciels

- Domaine public :
  - Harvest Cached : développements arrêtés
  - Modules proxy inclus dans certains serveurs : Apache, Roxen, Jigsaw
  - Squid : basé sur Harvest Cached et conseillé dans Renater-cache `www.nlanr.net/Squid`
  - Logiciels spécialisés dans une seule fonction : Socks (firewall)
- Produits commerciaux : Netscape, Microsoft

Aspects liés à la sécurité

Sources d'attaques par HTTP

Fuite d'informations personnelles :

- Par les fichiers log sur les serveurs.
- Par transfert d'informations sensibles en en-têtes :
  - `Server:` version du logiciel serveur WEB utilisé
  - `User-Agent:` version du browser utilisé
  - `Via:` les informations sur les machines traversées dans un domaine protégé par firewall
  - `Referer:` l'URI contenant le URI demandé peut être un document privé
  - `From:` l'adresse mail de l'utilisateur qui a lancé le browser
  - `Accept-*:` identifie les utilisateurs par leurs préférences (p.e. linguistiques)
- Par URIs avec des données sensibles : GET, `Referer:`

Attaques basées sur les chemins dans les URI :

- Peuvent accéder à des informations privées sur le serveur.
- Exemple : URI “/../../etc/passwd” exécuté sous Unix

Attaques basées sur les DNS :

- Cause : stoker d'une association (nom,IP) expirée.
- Accès à une adresse IP qui n'est plus sûre.

Changement d'identité par redirection :

- Sur un serveur qui héberge plusieurs organisations, les redirections doivent être contrôlées afin d'éviter l'accès à des ressources non autorisées.
- Réponses : `Content-Location:`, `Location:`

Authentification illimitée :

- Quand un client obtient un autorisation d'accès par authentification, il peut la retenir indéfiniment.
- Changements prévus pour les versions ultérieures.

Attaques par proxy/cache :

- Proxy/cache = “men-in-the-middle”, donc ils doivent être sûrs et sécurisés.
- Exemple : accès aux logs sur les proxies.
- Exemple : accès aux informations stockées sur les caches long temps après leur utilisation.