HELHa

Haute École
Louvain en Hainaut

Laboratoire de sécurité internet

Découverte du protocole HTTP

Jean-Louis Gouwy

Plan

HELHa
Haute École
Louvain en Hainaut

- Le protocole HTTP
  - Introduction
  - Les versions
  - Interprétation d'une URL
  - Le modèle client/serveur
  - La requête client
  - La réponse serveur
- Ateliers de découverte
  - Mise en place
  - Atelier 1: Envoi d'une requête HTTP par telnet
  - Atelier 2: Envoi de cette requête en HTTP 1.0
  - Atelier 3: Codification d'une requête HTTP dans un script
  - Atelier 4: Requête vers une ressource qui n'existe pas
  - Atelier 5: Redirection d'une requête
  - Atelier 6: Accès à un site protégé
  - Atelier 7: Refus d'accès à une ressource
  - Atelier 8: Accès sécurisé à un site
  - Atelier 9: Les connexions persistantes
- Autres champs

Plan

HELHa
Haute École
Louvain en Hainaut

- Les méthodes des clients
  - Les méthodes classiques
  - La méthode GET
  - La méthode POST
  - GET vs POST
  - Ateliers d'espionnage
- Les codes de retour
  - Un exemple de code de retour de catégorie 5
- Référence

Le protocole HTTP

HELHa
Haute École
Louvain en Hainaut

- **INTERPRETATION D'UNE URL *(Uniform Resource Locator)***

Soit l'URL : **`http://hypothetical.ora.com:80/`**

Le navigateur interprète cette URL de la façon suivante :

`http://`

Utiliser HTTP comme protocole.

`hypothetical.ora.com`

Contacter un ordinateur sur le réseau portant ce nom.

`:80`

Se connecter sur son port 80. S'il est omis le navigateur le rajoute Automatiquement.

`/`

Chemin du document.

Le protocole HTTP

HELHa
Haute École
Louvain en Hainaut

- **INTRODUCTION**

– HyperText Transfert Protocol (proto. de transfert de document hypertexte)

– Inventé par Tim Berners-Lee avec les adresses Web et le langage HTML pour créer le World Wide Web.

- **LES VERSIONS**

– 1990: HTTP/0.9

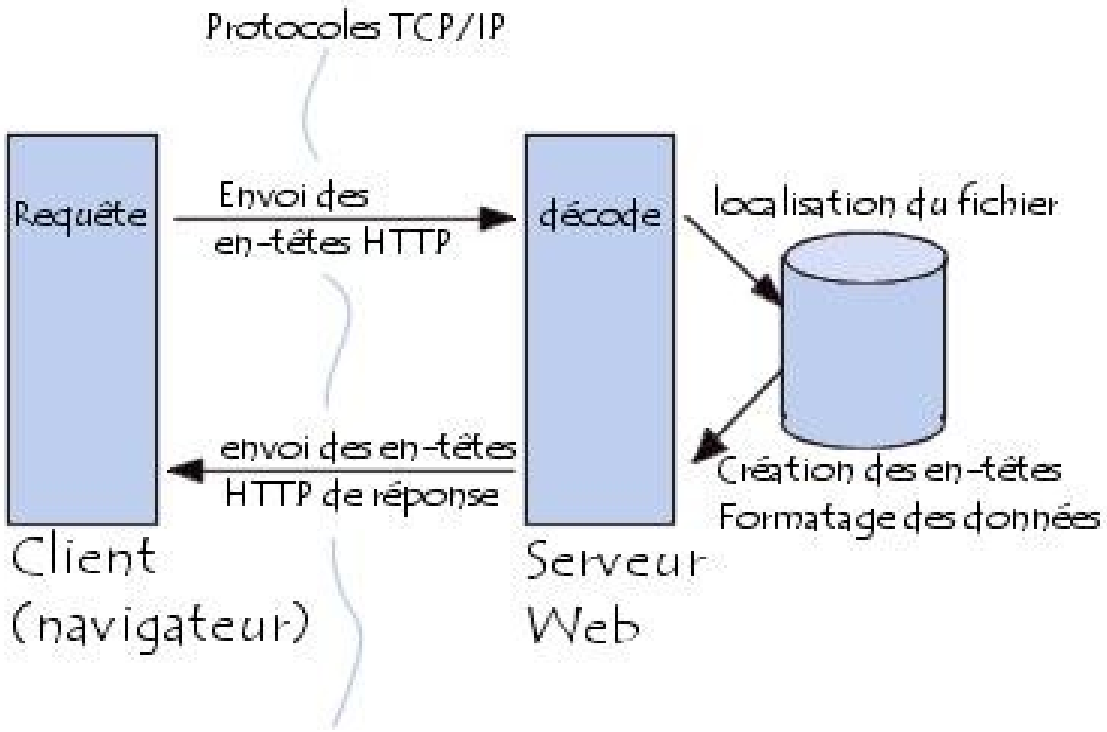
– 1996: HTTP/1.0 devient un standard. Cette version supporte les sites web virtuels hébergés par ip, la gestion de cache, l'identification.

– 1997: HTTP/1.1 devient le nouveau standard . Cette version ajoute notamment les connexions persistantes, la négociation de type de contenu (format de données, langue), les sites web virtuels hébergés par noms.

– 2015: HTTP/2 est poussé par Google mais encore très controversé. Cette version se targue d'améliorer sensiblement les performances de chargement de pages lourdes. A suivre….

Le protocole HTTP

- **LE MODELE CLIENT/SERVEUR**



Source: http://www.commentcamarche.net/contents/internet/http.php3

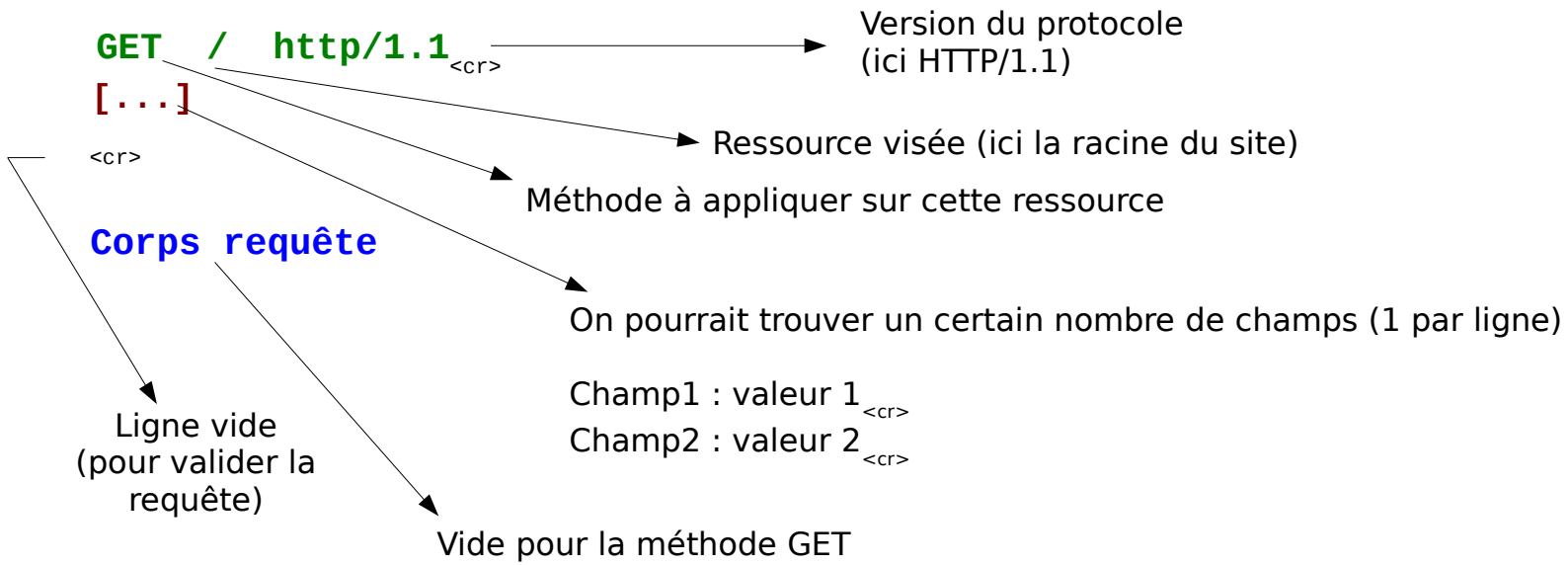
Le protocole HTTP

HELHa
Haute École
Louvain en Hainaut

- **LA REQUETE CLIENT**

Syntaxe d'une requête client

Ligne de commande (Commande, chemin du document, Version de protocole)
En-tête de requête
<cr>
Corps de requête



Le protocole HTTP

HELHa
Haute École
Louvain en Hainaut

- **LA REPONSE SERVEUR**

Syntaxe d'une réponse serveur

Ligne de statut (Version, Code-réponse, Texte-réponse)
En-tête de réponse
<nouvelle ligne>
Corps de réponse

```
HTTP/1.1 200 OK
Date: Mon, 24 Sep 2017 14:03:19 GMT
Server: Apache
...
Connection: close
Content-Type: text/html

<html>
…
</html>
```

Vers.http utilisée

Code d'état — *Utilisé par le programme client.*

Texte explicatif — *Informer l'utilisateur en cas d'erreur.*

Nouvelle ligne

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **MISE EN PLACE**

– Pour ces ateliers, nous travaillerons avec 4 sites `www.test.be`, `bank.test.be`, `get.test.be` et `post.test.be` hébergés par un serveur Apache 2.4.27 tournant sur la machine d'Ip `10.103.0.x`

Les sites préfixés par `www`, `get` et `post` sont accessibles par le port 80.
Le site préfixé par `bank` est accessible par le port 443.

– Votre VM Linux sera connectée sur ce réseau.

– Les lignes suivantes seront rajoutées au fichier `/etc/hosts`:

```
10.103.0.x www.test.be bank.test.be get.test.be post.test.be
```

– Toutes les commandes seront lancées à partir cette VM.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 1:** Envoi d'une requête HTTP par telnet

```
#telnet www.test.be 80
```
→ adresse et port d'écoute du serveur web

```
…
GET / HTTP/1.1
host: www.test.be
```
→ Requête du client (à formuler via le protocole http car on frappe sur le port 80 d'un serveur web qui ne comprend que de l'http...

```
HTTP/1.1 200 OK
Date: Wed, 13 Sep 2017 13:08:06 GMT
Server: Apache/2.4.27 (Fedora) OpenSSL/1.1 …
…
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title> Site web de test </title>
</head>
<body>
<h1> Site web de test. Ca marche !!! </h1>
</body>
</html>Connection closed by foreign host.
#
```
→ Réponse du serveur

→ Code HTML transmis au client et fermeture de la connexion effectuée par le serveur.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 1:** Envoi d'une requête HTTP par telnet (suite)

Champs de la requête

`GET / HTTP/1.1` → Méthode GET pour obtenir la ressource se trouvant à la racine du site. La requête est envoyée par le protocole HTTP 1.1

`host: www.test.be` → Le champ 'host' a été introduit dans la version 1.1 du protocole. Nécessaire pour que le client puisse toucher des sites hébergés par noms (voir Chapitre 'Serveur Web Apache')

Champs de la réponse

`HTTP/1.1 200 OK` → Code de retour renvoyé par le serveur. Code 200 : requête accomplie avec succés.

`Date: Wed, 13 Sep 2017 13:08:06 GMT` → Date d'envoi de la réponse

`Server: Apache/2.4.27 (Fedora) OpenSSL/1.1 …` → Informations sur le serveur

…

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 2:** Envoi de cette requête en HTTP 1.0

```
#telnet www.test.be 80
…
GET / HTTP/1.0
```
→ La requête est envoyée via le protocole 1.0 (le champ 'host' serait ignoré même s'il était présent)

```
HTTP/1.1 200 OK
…
```
→ Le serveur répond via le protocole 1.1 en signalant que tout est accompli avec succès ...

```
<html>
<head>
<title> Site web principal </title>
</head>
<body>
<h1> Site web principal. </h1>
</body>
</html>Connection closed by foreign host.
 #
```

→ … excepté que la page renvoyée n'est pas celle attendue.

Pourquoi ?
voir Chapitre 'Serveur Web Apache'

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 3: Codification d'une requête HTTP dans un script**

```
# cat test.sh
#!/bin/bash
echo "open www.test.be 80"
sleep 2
echo "GET / HTTP/1.1"
echo "host: www.test.be"
echo
sleep 2
```

→ contenu du script

→ génération d'un <enter>

→ pour que le serveur ait le temps retourner ses résultats avant la fermeture du script et donc de la communication http...

```
# ./test.sh | telnet
```

→ pour l'exécuter

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 4:** Requête vers une ressource qui n'existe pas

```
#telnet www.test.be 80
…
GET /notexist/ HTTP/1.1
host: www.test.be
```

→ On tente de récupérer la page d'accueil d'un site se trouvant dans le dossier 'notexist' du site racine...

```
HTTP/1.1 404 Not Found
…
```

→ Code de retour : 404 (Not Found)
La ressource demandée n'a pas été trouvée.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /notexist/ was not found on this server.</p>
</body></html>
Connection closed by foreign host.
#
```

→ Code HTML d'une page d'erreur standard transmise au client.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 5:** Redirection d'une requête

```
#telnet www.test.be 80
…
GET /service/ HTTP/1.1
host: www.test.be
```

→ On tente de récupérer la page d'accueil d'un site se trouvant dans le dossier 'service' du site racine...

```
HTTP/1.1 301 Moved Permanently
…
Location: http://www.test.be/staff/
```

Code de retour : 301
→ La ressource demandée a été déplacée dans le dosssier 'staff'

Pour retourner la valeur du champ 'Location', le serveur a besoin de s'autoréférencer.
C'est là que la directive 'ServerName' interviendra (voir Chapitre 'Serveur Web Apache').

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.test.be/staff/">here</a>.</p>
</body></html>
Connection closed by foreign host.
#
```

→ Code HTML d'une page de redirection standard transmise au client.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 5:** Redirection d'une requête (suite)

```
#telnet www.test.be 80
…
GET /staff/ HTTP/1.1
host: www.test.be

HTTP/1.1 200 OK → Ca marche !
…

<html>
<head>
<title> Site web de notes de service </title>
</head>
<body>
<h1> Site web des notes de service. Ca marche !!! </h1>
</body>
</html>Connection closed by foreign host.
#
```

→ Alors, tentons de récupérer la page d'accueil dans ce dossier…

→ Code HTML de cette page d'accueil.

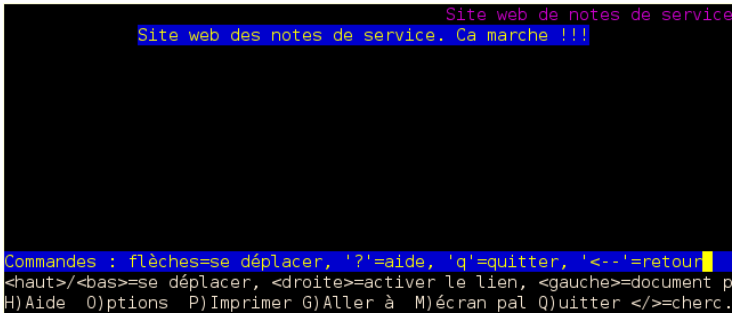
Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 5:** Redirection d'une requête (suite)

Tous les clients web sont programmés de façon à réitérer une requête lorsqu'ils recoivent une réponse avec un champ 'Location'.

```
# lynx www.test.be/service
```



URL saisie: http://www.test.be/service

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 6:** Accès à un site protégé

```
#telnet www.test.be 80
…
GET /controle/ HTTP/1.1
host: www.test.be
```

→ On tente de récupérer la page d'accueil d'un site se trouvant dans le dossier 'controle' du site racine...

```
HTTP/1.1 401 Unauthorized
…
WWW-Authenticate: Basic realm="Acces au site sous contrôle"
…
```

→ Code de retour 401. On a besoin de s'identifier pour accéder à cette ressource.

Le serveur renvoie dans sa réponse un champ 'WWW-Authenticate' indiquant au client le type d'authentification, ici: Basic (voir Chapitre 'Serveur Web Apache').

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Unauthorized</title>
</head><body>
<h1>Unauthorized</h1>
<p>This server could not verify that you are authorized to access the document
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.
#
```

→ Code HTML de cette page d'erreur.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 7:** Refus d'accès à une ressource

```
#telnet www.test.be 80
…
GET /accueil.html HTTP/1.1
host: www.test.be
```

→ On tente de récupérer la page accueil.html du site racine...

```
HTTP/1.1 403 Forbidden
…
```

→ Code de retour 403. L'accès à cette ressource est refusé. Le serveur Apache n'a probablement pas le droit de lire ce fichier (droit 'r' sous Linux).

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /accueil.html
on this server.<br />
</p>
</body></html>
Connection closed by foreign host.
#
```

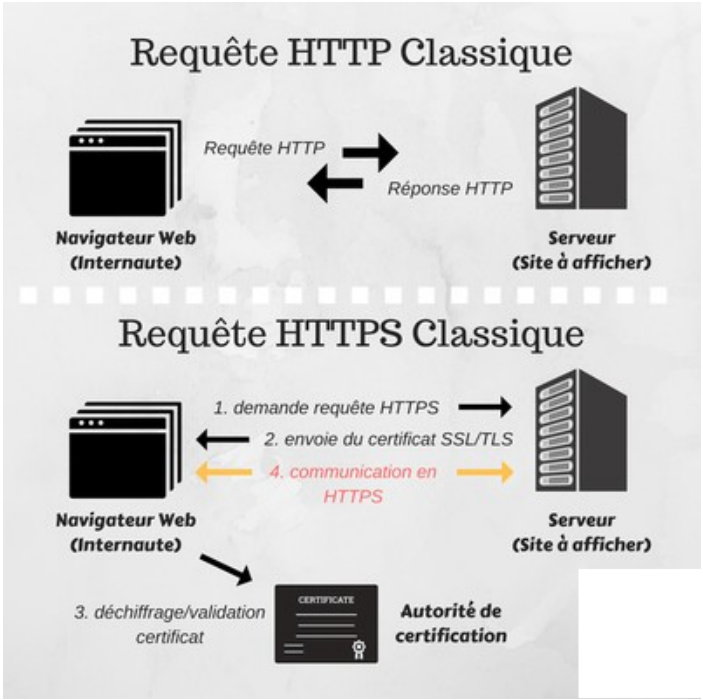
→ Code HTML de cette page d'erreur.

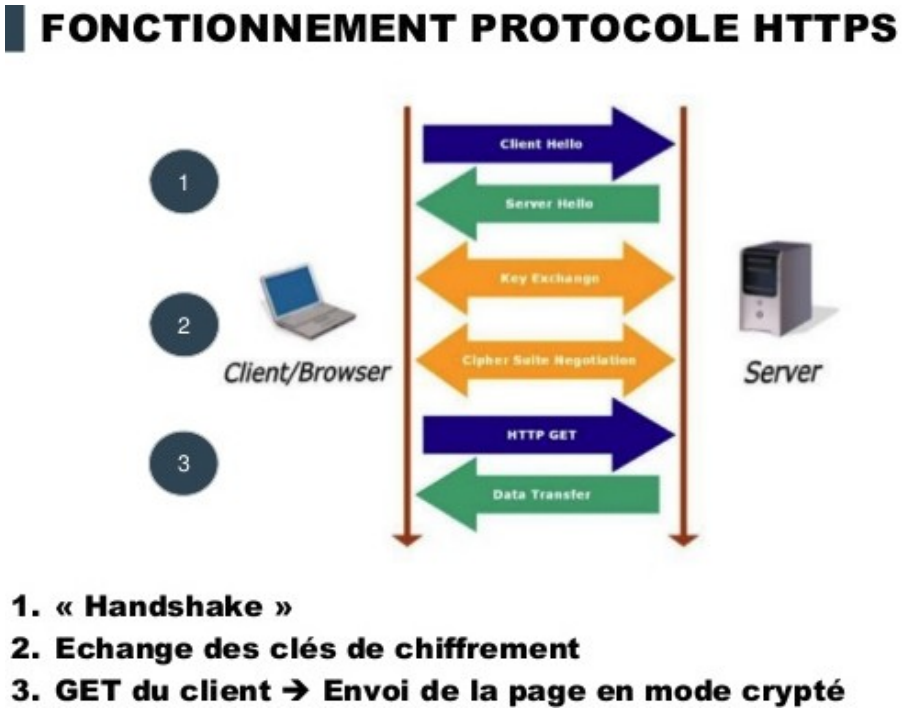
Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site

  - Cas classique: l'accès à un site bancaire se fait via le protocole HTTPS.
  - Les data échangées entre le client et le serveur sont chiffrées.





- Le serveur de test héberge un site sécurisé accessible via l'URL: `https://bank.test.be`

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

```
#telnet bank.test.be 80
…
GET / HTTP/1.1
host: bank.test.be

HTTP/1.1 200 OK→ Le serveur répond  en signalant que tout est accompli avec succès ...
…

<html>
<head>
<title> Site web principal </title>      → … excepté que la page reçue n'est pas celle
</head>                                      du site bancaire !!!
<body>
<h1> Site web principal. </h1>
</body>
</html>Connection closed by foreign host.
#
```

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

#telnet bank.test.be 443 → 443 étant le port d'écoute par défaut d'un serveur https.

```
…
GET / HTTP/1.1
host: bank.test.be

HTTP/1.1 400 Bad Request
…
```

Le serveur ne comprend pas la requête car il est configuré en https sur le port 443.
Donc, toute requête entrant par ce port doit être initiée par un message 'hello' et non 'GET'.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
Reason: You're speaking plain HTTP to an SSL-enabled server port.<br />
Instead use the HTTPS scheme to access this URL, please.<br />
</p>
</body></html>
Connection closed by foreign host.
#
```

→ Code HTML de cette page d'erreur.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

Dans ce cas, il est donc nécessaire d'utiliser un utilitaire de la bibliothèque 'openssl' se chargeant de réaliser tous les échanges nécessaires afin de pouvoir créer un canal sécurisé au travers lequel la conversation HTTP pourra circuler.

```
# openssl s_client -connect bank.test.be:443
...
...
...
```

→ Le certificat est téléchargé mais n'est pas validé. En effet, ici, nous travaillons avec un certificat auto-signé qui n'est pas reconnu par le client (voir Chapitre 'Serveur Web Apache').

```
GET / HTTP/1.1
HTTP/1.1 400 Bad Request
```

→ Le canal sécurisé n'est donc pas créé et l'échange HTTP ne peut donc se poursuivre.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
</body></html>
closed
#
```

→ Code HTML de la page d'erreur.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

Codification d'une requête HTTPS dans un script

```
# cat testhttps.sh
#!/bin/bash
echo "s_client -connect www.kernel.org:443"
sleep 2
echo "GET / HTTP/1.1"
echo "host: www.kernel.org"
echo
sleep 2
```

```
# ./testhttps.sh | openssl > out      → Pour l'exécuter
```

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

```
# ./testhttps.sh | openssl > out
…
# cat out
…
…
…
```

→ Le certificat est téléchargé et validé. En effet, ici, le certificat a été certifié conforme par une autorité de certification reconnue par le client (voir Chapitre 'Serveur Web Apache')

```
HTTP/1.1  200 OK
Server: nginx
…

--- Code html de la page d'accueil du site ---
```

Ce code est chiffré par le serveur, puis transporté et déchiffré par le client.

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 8:** Accès sécurisé à un site (suite)

curl (client URL library) permet de récupérer le contenu d'une ressource accessible par un réseau informatique

```
# curl https://bank.test.be
```

```
# curl -k -v https://bank.test.be
```

Pour' bypasser' la validation du certificat.

Mode bavard.

```
# curl -v http://www.test.be
```

…

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 9:** Les connexions persistantes

Une connexion persistante garde la connexion réseau ouverte afin de faire passer plusieurs transactions entre le client et le serveur à l'intérieur de celle-ci (pipelining).

Sans connexion persistante, les *x* objets (images, css...) d'une page seront transférés par *x* connexions réseaux différentes → lenteur

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 9:** Les connexions persistantes (cp)

# telnet www.test.be 80<cr> GET / HTTP/1.1<cr> host : www.test.be<cr> <cr> → / ↑ CLIENT		SERVEUR **Ouvre une CP (1)**	SERVEUR **N'ouvre pas de CP (2)**
CLIENT	**Ne demande pas de CP**	Le serveur envoie la page et fermera la connexion après 30 secondes. Durant ce temps, le processus fils httpd est mobilisé en état d'attente au lieu de pouvoir servir des requêtes. On conseille une persistance allant de 2 à 5 sec. maximum.	Le serveur envoie la page et signale au client qu'il fermera immédiatement la connexion après l'envoi de la page. **Réponse** **Connection: close** Le serveur ferme la connexion directement après téléchargement de la page.
CLIENT	**Demande une CP** **Requête** **Connection: keep-alive**	Le serveur accepte la demande mais signale au client qu'il fermera la connexion après 30 sec. **Réponse** **Keep-Alive : timeout=30, max=100** **Connection: Keep-Alive** La connexion reste ouverte durant 30 sec. Puis est fermée par le serveur.	Le serveur n'accepte pas la demande du client et lui signale qu'il fermera immédiatement la connexion après l'envoi de la page. **Réponse** **Connection: close** Le serveur ferme la connexion directement après téléchargement de la page.

(1) KeepAlive On
KeepAliveTimeOut 30
MaxKeepAliveRequest 100

(2) KeepAlive Off

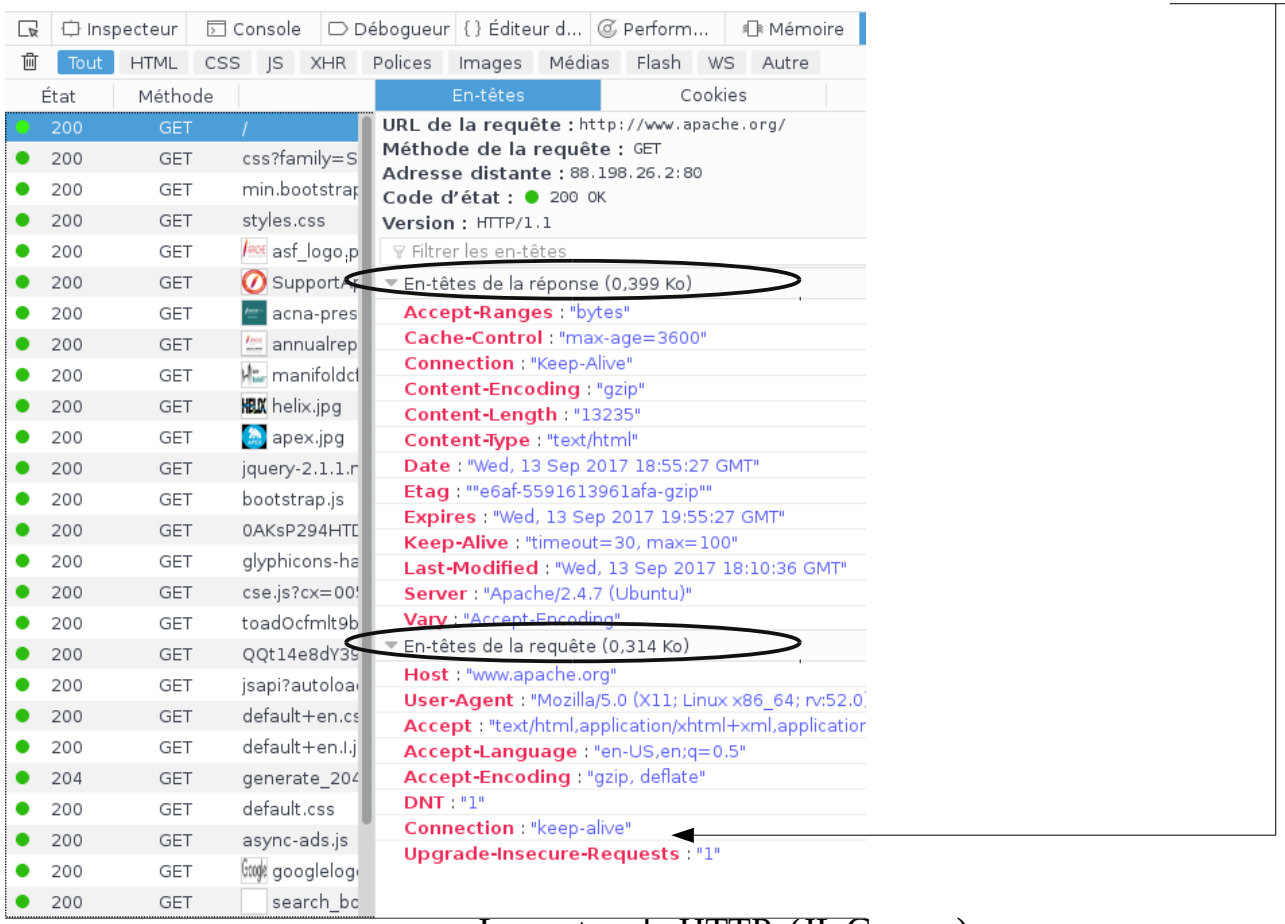
https://httpd.apache.org/docs/2.4/fr/mod/core.html#keepalive

Ateliers de découverte

HELHa
Haute École
Louvain en Hainaut

- **ATELIER 9:** Les connexions persistantes (suite)

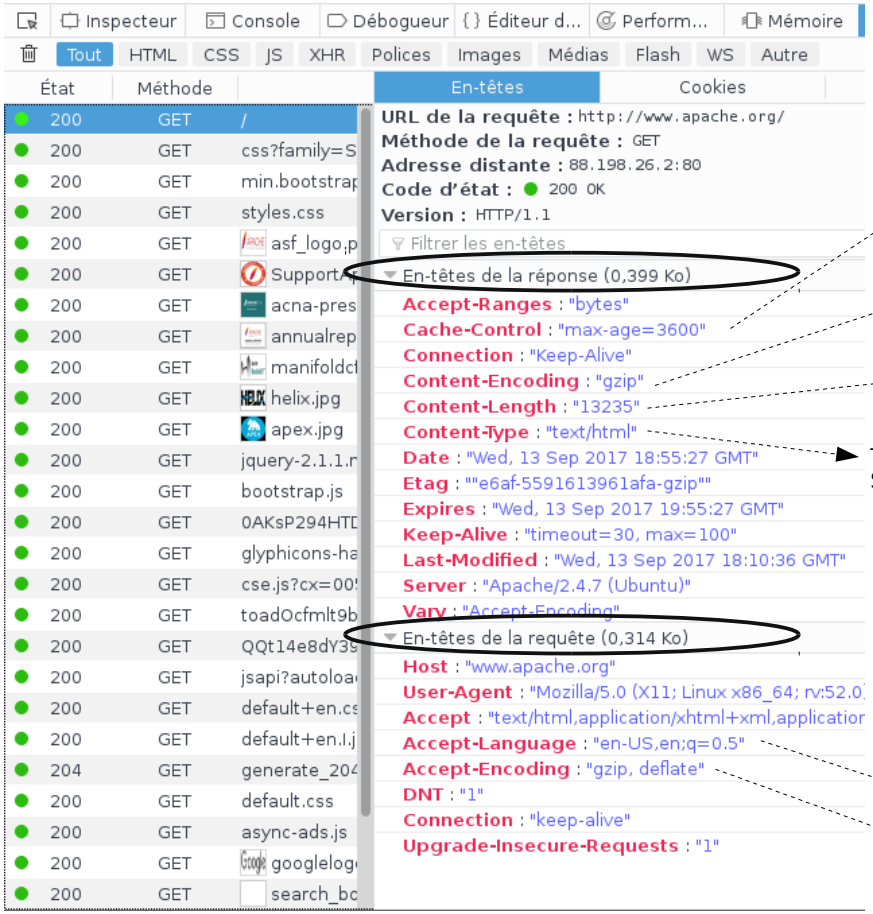
Les clients HTTP 1.1 (ils le sont tous actuellement) envoient toujours leur requête en demandant une connexion persistante.



Autres champs

HELHa
Haute École
Louvain en Hainaut

Plus d'info: http://abcdrfc.free.fr/rfc-vf/rfc2616.html
https://www.w3.org/Protocols/rfc2616/rfc2616.html



Le client mettra en cache cette ressource durant 3600 sec.

L'entité transférée est compressée en format gzip. Le client possède l'algo. de décompression (voir champ 'Accept-Encoding'

Taille du corps du message

Type et sous-type du média utilisé par le corps du message. Souvent en corrération avec l'Accept de la requête.

Le client se présente.

Type de média préféré par le client

Si le serveur le possède, le document sera envoyé dans cette langue.

Le serveur pourra compresser des entités dans un algo. que le client comprend.

Conseil sur la sécurité: https://httpd.apache.org/docs/2.4/fr/misc/security_tips.html

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LES METHODES CLASSIQUES**

GET	Obtenir la ressource située à l'URL spécifiée. Elle peut-être le contenu d'un fichier statique ou elle peut invoqué un programme qui génère des données (script CGI, PHP...).
HEAD	Obtenir la ressource située à l'URL spécifiée (la réponse ne contient que l'entête, et pas le contenu de la ressource).
POST	Envoi de données au programme situé à l'URL spécifiée (le corps de la requête peut être utilisé).
PUT	Ajouter une ressource sur le serveur.
DELETE	Suppression de la ressource située à l'URL spécifiée.

Dans la pratique peu de serveurs autorisent les actions de type PUT et DELETE pour des raisons évidentes de sécurité.

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE GET**

– Pour récupérer une ressource statique ou dynamique sur le serveur.

Statique : ex. une page HTML
Dynamique : ex. une page HTML générée par un script PHP auquel on peut passer des arguments

```
#telnet get.test.be 80
…
GET /index.php?user=cobaye&pass1=1234&pass2=1234 HTTP/1.1
host: get.test.be
<cr>
```

❶ *Envoi requête*

❷ *Exécution*

```
index.php          [----]  0 L:[  1+12  13/ 13] *(
<html>
<head>
<title> Test methode GET </title>
</head>
<body>
<?php
    echo "<p>Hello world...".$_GET['user']."</p>";
    if ( $_GET['pass1'] == $_GET['pass2']) {
        echo "<p>MOT DE PASSE CONFIRME</p>";
    }
 ?>
</body>
</html>
```

❸ *Envoi réponse*

```
HTTP/1.1 200 OK
…
<html>
<head>
<title> Test methode GET </title>
</head>
<body>
<p>Hello world...cobaye</p><p>MOT DE PASSE CONFIRME</p></body>
</html>
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE GET**
  - Invocation avec du code html :

```
# lynx get.test.be/formget.html
```

❶

```
formget.html          [----]  4 L:[  1+11  12/ 16] *(279
<html>
<title> Test methode GET </title>
<center>
<hr><h1>Formulaire de test de la methode GET</h1><hr>
</center>
<form method="get" action="/index.php">
<pre>
<b>
Nom de l'utilisateur: <INPUT NAME="user">
Mot de passe: <INPUT NAME="pass1">
Confirmation <INPUT NAME="pass2">
</b>
</pre>
<INPUT TYPE="submit" VALUE="Envoyer">
</form>
</html>
```

❷

```
                                                Test methode GET
   ------------------------------------------------------------------------

                Formulaire de test de la methode GET
   ------------------------------------------------------------------------

Nom de l'utilisateur: _____________________
Mot de passe: _____________________
Confirmation _____________________

   Envoyer
```

❸

```
                                                Test methode GET
   ------------------------------------------------------------------------

                Formulaire de test de la methode GET
   ------------------------------------------------------------------------

Nom de l'utilisateur: cobaye_______________
Mot de passe: 1234_________________
Confirmation 1234__________________

   Envoyer
```

❹

```
index.php             [----]  0 L:[  1+12  13/ 13] *(2
<html>
<head>
<title> Test methode GET </title>
</head>
<body>
<?php
    echo "<p>Hello world...".$_GET['user']."</p>";
    if ( $_GET['pass1'] == $_GET['pass2']) {
        echo "<p>MOT DE PASSE CONFIRME</p>";
    }
 ?>
</body>
</html>
```

❺

```
                                                Test methode GET
   Hello world...cobaye

   MOT DE PASSE CONFIRME
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE GET**

– Invocation avec la commande curl :

```
# curl "http://get.test.be/index.php?user=cobaye&pass1=1234&pass2=1234"
<html>
 <head>
 <title> Test methode GET </title>
 </head>
 <body>
 <p>Hello world...cobaye</p><p>MOT DE PASSE CONFIRME</p></body>
#
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE POST**

– Pour envoyer des données à des programmes sur le serveur.

```
# testpost.sh
echo "open post.test.be 80"
sleep 2
echo "POST /index.php HTTP/1.1"
echo "host: post.test.be"
echo "Content-Length: 33"
echo "Content-Type: application/x-www-form-urlencoded"
echo
echo "user=cobaye&pass1=1234&pass2=1234"
sleep 2
```

Longueur des données (corps du message)

Les valeurs sont encodées sous forme de couples clé-valeur séparés par '&', avec un '=' entre la clé et la valeur.

```
# ./testpost.sh | telnet > out
```

❶ *Envoi requête*

❷ *Exécution*

```
index.php                [-M--]  0 L:[  1+13  14/ 14] *(2
<html>
<head>
<title> Test methode POST </title>
</head>
<body>
<?php
    echo "<p>Hello world...".$_POST['user']."</p>";
    if ( $_POST['pass1'] == $_POST['pass2']) {
        echo "<p>MOT DE PASSE CONFIRME</p>";
    }
 ?>
</body>
</html>
```

❸ *Envoi réponse*

```
# cat out
HTTP/1.1 200 OK
…
<html>
<head>
<title> Test methode POST </title>
</head>
<body>
<p>Hello world...cobaye</p><p>MOT DE PASSE CONFIRME</p></body>
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE POST**

  - Invocation avec du code html :

```
# lynx post.test.be/formpost.html
```

❶

```
formpost.html       [----]  0 L:[  1+ 4   5/ 16] *(106 /
<html>
<title> Test methode POST </title>
<center>
<hr><h1>Formulaire de test de la methode POST</h1><hr>
</center>
<form method="post" action="/index.php">
<pre>
<b>
Nom de l'utilisateur: <INPUT NAME="user">
Mot de passe: <INPUT NAME="pass1">
Confirmation <INPUT NAME="pass2">
</b>
</pre>
<INPUT TYPE="submit" VALUE="Envoyer">
</form>
</html>
```

❷

```
                                                Test methode POST
  ------------------------------------------------------------------------

                  Formulaire de test de la methode POST
  ------------------------------------------------------------------------

Nom de l'utilisateur: _____________________
Mot de passe: _____________________
Confirmation _____________________

   Envoyer
```

❸

```
                                                Test methode POST
  ------------------------------------------------------------------------

                  Formulaire de test de la methode POST
  ------------------------------------------------------------------------

Nom de l'utilisateur: cobaye_____________
Mot de passe: 1234_____________
Confirmation 1234_____________

   Envoyer
```

❹

```
index.php           [-M--]  0 L:[  1+13  14/ 14] *(2
<html>
<head>
<title> Test methode POST </title>
</head>
<body>
<?php
    echo "<p>Hello world...".$_POST['user']."</p>";
    if ( $_POST['pass1'] == $_POST['pass2']) {
        echo "<p>MOT DE PASSE CONFIRME</p>";
    }
 ?>
</body>
</html>
```

❺

```
                                                Test methode POST
  Hello world...cobaye

  MOT DE PASSE CONFIRME
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE POST**

– Invocation avec la commande curl :

```
# curl --data "user=cobaye&pass1=1234&pass2=1234" http://post.test.be/index.php
<html>
 <head>
 <title> Test methode POST </title>
 </head>
 <body>
 <p>Hello world...cobaye</p><p>MOT DE PASSE CONFIRME</p></body>
#
```

Les méthodes des clients

HELHa
Haute École
Louvain en Hainaut

- **GET vs POST**

	Taille des données	Type de données	Invocation	Localisation des données	Visiblité dans l'URL et historique du browser et dans les logs du serveur	Sniffing des données sensibles
GET	1024 car. maximum	ASCII	Souvent à partir d'un code HTML (1)	En argument dans la méthode de l'en-tête de la requête	Oui → danger si on passe des données sensibles (ex.mdp) (3)	Oui car le protocole HTTP circule en clair (3)
POST	Aucune restriction	ASCII ou binaire (image, son ...)	Souvent à partir d'un code HTML (2)	Dans le corps de l'en-tête de la requête	Non	Oui car le protocole HTTP circule en clair (3)

(1) form method="get" action="/index.php"
(2) form method="post" action="/index.php"

(3) Pistes pour sécuriser un échange GET ou POST de données sensibles :

- Salage (chiffrement) de ces données
- Utiliser HTTPS
- ...

Ateliers d'espionnage

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE GET**

  Récupération via `tshark` les mots de passe passés en arguments.

Terminal 2

```
#tshark -V -i eth0 port http > sniff.get
…
```

Terminal 1

```
# telnet get.test.be 80
 …
GET /index.php?user=cobaye&pass1=1234&pass2=1234 HTTP/1.1
host: get.test.be
 <cr>
 …
```

Terminal 2

```
# cat sniff.get| grep "GET"
    GET /index.php?user=cobaye&pass1=1234&pass2=1234 HTTP/1.1\r\n
    …
#
```

Ateliers d'espionnage

HELHa
Haute École
Louvain en Hainaut

- **LA METHODE POST**

  Récupération via `tshark` les mots de passe passés dans le corps de l'en-tête.

Terminal 2

```
# tshark -V -i eth0 port http > sniff.post
…
```

Terminal 1

```
# ./testpost.sh | telnet
…
```

Terminal 2

```
# ^C
# n=`grep -n "Line-based" sniff.post| cut -d: -f1`; tail --lines=+$n
                                                        sniff.post | more
Line-based text data: application/x-www-form-urlencoded
    user=cobaye&pass1=1234&pass2=1234
…
#
```

Les codes de retour

HELHa
Haute École
Louvain en Hainaut

10x	Codes d'informations.
20x	L'opération s'est correctement effectuée. Le plus courant est le code 200 (OK).
30x	L'objet demandé a été déplacé (ex. redirection).
40x	Lorsque le client a commis une erreur. Erreur '404 Not found' lorsque l'objet demandé n'existe plus...
50x	Erreur du coté du serveur tel qu'un bug dans un script CGI.

Plus d'info: http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

Les codes de retour

HELHa
Haute École
Louvain en Hainaut

- **Un exemple de code de retour de catégorie 5**

  Codons volontairement une erreur dans le fichier `index.php` de la méthode `GET` :

  Soit : `echo "<p>Hello world…".$_GET['user'] "</p>`

  Et invoquons ce script avec la commande `curl` :

Pour visualiser les en-têtes.

```
# curl -i "http://get.test.be/index.php?user=cobaye&pass1=1234&pass2=1234"
HTTP/1.0 500 Internal Server Error
    …
    #
```

Erreur lors de l'exécution d'un script côté serveur..

Références

- **HTTP précis & concis**
  Editions O'Reilly, Paris 2000
  Clinton Wong
  Traduction de Laurent Bourdron
  ISBN 2-84177-115-6