M3102-- TD 2

Les protocoles HTTP et HTTPS

1. Introduction

A sa base, l'Internet se compose de deux composants :

1. **Le protocole HTTP** : pour le transfert d'informations en formes de données
2. **DNS** : une arborescence de nom

Le fonctionnement actuel des sites web a évolué avec le temps, les technologies qui permettent de mettre en place le transfert de données ont changé également ; malgré cela, le fonctionnement de l'Internet n'a pas vraiment changé.

Le protocole HTTP fonctionne dans une architecture **client-serveur**. Le client HTTP est le navigateur qu'on utilise. Le processus serveur se trouve sur un serveur Apache ou IIS (Internet Information Services).

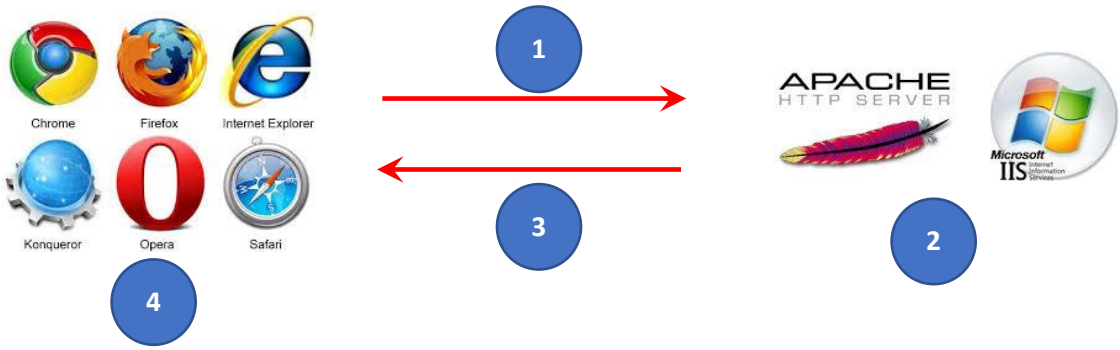


Figure 1 : le fonctionnement de HTTP

Le protocole HTTP se déroule en quatre étapes, montrées aussi dans la Figure 1 :

1. Le client initie une connexion TCP/IP. Puis il demande un document au serveur en langage HTTP.
2. Le serveur cherche le document demandé ou peut le générer en utilisant un script PHP
3. Le serveur envoie le document sur la connexion TCP/IP
4. Le client reçoit le document. En interprétant le langage HTML/CCS, le client peut la rendre dans un document visuel affiché dans le navigateur.

2. La syntaxe URL

Les informations cherchées par le client peuvent avoir plusieurs formes et être accédées de différentes façons. Elles peuvent être accueillies sur des machines différents. C'est pourquoi toute ressource disponible sur le Web est référencée par le biais d'une ***Uniform Ressource Locator*** (URL). Une URL est un type spécial de URI (***Uniform Ressource Identifier***).

Une ressource peut être un document HTML, mais également une image, une Applet Java, un fichier qui doit être transféré. En tant que référence unique à une ressource, un URL inclut :

- ***Un préfix*** constant : "URL`:`" ;
- Un mode d'accès à la ressource, appelé également un ***schéma URI*** -- `http`, `ftp`, `mailto`, `irc`, etc. On peut enregistrer un schéma auprès de la IANA -- Internet Assigned Numbers Authority. Le schéma est suivi par un ":" ;
- La partie ***protocoles Internet***, qui commence par "//" et finit par "/". Les éléments de cette partie incluent :
    - (optionnellement) Un ***nom d'utilisateur***, si besoin. Cette option est utile lorsque la ressource se trouve par exemple sur un serveur FTP. Nous pouvons rajouter un mot de passe, en le séparant par un ";" du nom d'utilisateur. A la fin de cette section il faut ajouter un "@" ;
    - Un ***nom de machine***. De préférence ceci est le nom du domaine de la machine dans le format donné par le standard RFC1037. Parfois on peut avoir une adresse IP plutôt qu'un nom de domaine, mas ceci est moins encouragé ;
    - Un ***numéro de port***. Cette valeur-ci est obligatoire si on veut que l'utilisateur joigne la machine par un port différent à celui qui est standard pour le schéma donné ;
    - Un ***chemin***. Le chemin indique où la ressource se trouve-t-elle dans le domaine indiqué ;
    - (optionnellement) Une ***requête de type chaine de caractères*** (query string). Cette partie suit le chemin. Le… commence par un "?" et consiste en des tuples nom et valeur, séparés par des "&". Par exemple "?term=bluebird&source=browser-search" ;

3. Les protocoles HTTP et HTTPS

Le transfert de données peut s'effectue d'une façon non-sécurisée (en http) ou d'une façon sécurisée (en https). Nous allons premièrement regarder la variante non-sécurisée, puis expliquer le fonctionnement du protocole sécurisé.

3.1 Le protocole HTTP

Le protocole HTTP se déroule sur TCP à la couche transport. Ce protocole représente un dialogue entre un client et un serveur Web. Les échanges entre ces deux parties se concrétisent dans des requêtes et réponses.

Plusieurs versions du protocole HTTP existent.

- **HTTP V0.9** : la première version de HTTP documentée, publiée en 1991. Cette version est actuellement obsolète.
- **HTTP V1.0** : décrit par le RFC 1945.
- **HTTP/1.1** : défini actuellement en RFC 7230.
- **HTTP/2** : défini en 2015 dans le RFC 7540. Cette version modifie la façon dont les données sont transportées entre un client et un serveur. De plus, elle permet également le fonctionnement avec TLS (version 1.2 ou ultérieure) pour les URI en HTTPS. Actuellement, autour de 41% des sites Web fonctionnent en HTTP/2 (Source : w3techs.com).
- **HTTP/3** : HTTP over QUIC. Cette version utilise UDP plutôt que TCP. En septembre 2019 Cloudflare et Chrome ont rajouté du support pour cette version.

Si nous regardons la communication entre le client et le serveur, les premiers messages sont ceux qui établissent la connexion TCP. Ensuite il y aura une négociation de version du protocole. Puis, le client va proposer une version de HTTP qu'il souhaite utiliser et le serveur va confirmer si cette version est acceptable ou non. Le client et le serveur vont échanger des requêtes et des réponses qui seront envoyés en clair. Finalement, la connexion TCP sera fermée.

Quelques commandes intéressantes HTTP sont :

- GET : permet de récupérer un document sur HTTP
- HEAD : permet de ne récupérer que les informations concernant une ressource
- OPTIONS : récupère des informations concernant le serveur HTTP
- PUT : permet d'ajouter ou de remplacer une ressource
- DELETE : supprime une ressource

3.2 Le protocole HTTPS

Lorsqu'on utilise le protocole HTTP, le contenu du dialogue entre le client et le serveur sont publiquement visibles. Si on veut que ces contenus soient privés, on peut utiliser le protocole HTTPS.

Le client et le serveur commencent en établissant une connexion TCP. Puis, les deux parties vont négocier une session du protocole Transport Layer Security (TLS). Les messages de TLS sont échangés en clair sur TCP. Une fois la connexion TLS négociée, les messages HTTP que le client et le serveur échangent seront chiffrés et authentifiés en utilisant des outils cryptographiques.

L'échange de clé authentifié -- son rôle et son fonctionnement

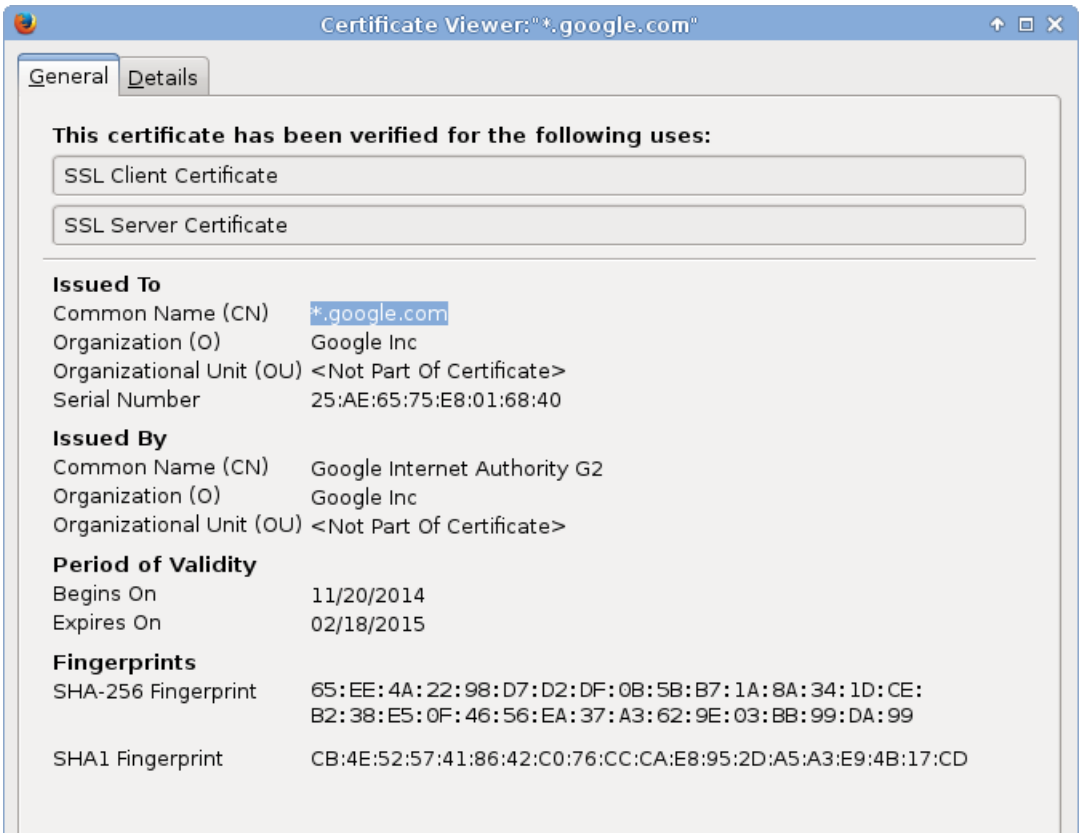
Le protocole TLS est un exemple de protocole d'échange de clé authentifié. D'autres exemples très utilisés aujourd'hui sont SSH (utilisé pour sécuriser l'accès à distance à une machine) et IPSec (utilisé principalement pour sécuriser la connexion par VPN).

Ces protocoles fonctionnent dans un modèle client-serveur. Pour TLS, le client HTTP est souvent également le client TLS. Le protocole marche en deux temps : premièrement le client et le serveur échangent des messages en clair pour établir une clé secrète, et puis les deux parties utilisent cette clé secrète pour chiffrer et authentifier leurs communications.

Le fonctionnement du protocole TLS est difficile à expliquer -- on le verra en plus de détail au cours du module OS01. Toutefois, nous allons explorer quelques éléments de ce protocole ici.

Dans la première partie du protocole TLS, ainsi appelée un poignet de main (handshake) le client et les serveurs s'envoient des messages qui peuvent être interceptés par une entité écoutant sur la ligne (qu'on appelle Man-in-the-Middle -- MitM -- ou Person-in-the-Middle --PitM). La « magie » du protocole consiste en permettant le client et le serveur de calculer une clé inconnue par le MitM malgré le fait que chaque composante de l'échange de clé soit mise à la disposition de l'attaquant.

Un aspect subtil du protocole est l'authentification. Pour que le protocole marche de façon sécurisée, il est essentiel qu'au moins le serveur authentifie les messages qu'il envoie. A ce but il aura besoin d'une paire de clés : une clé privée et une clé publique, par exemple une clé de signatures. En signant une partie de ses messages avec sa clé secrète, il convaincra le client qu'il est bien un serveur légitime. Dans notre cas, le client doit vérifier la signature du serveur en utilisant sa clé publique. Pour s'assurer que la clé



publique est associée au serveur cherché, le client aura besoin d'un certificat, établi par une autorité qui garantit l'association entre la clé publique et un nom de domaine.

Le navigateur vérifiera les certificats à chaque fois qu'on initialise une connexion sur TLS. Il faut surtout ne jamais faire confiance à un serveur sans un certificat valide.

Le chiffrement authentifié

A la fin du poignet de main (handshake) le client et le serveur calculent un nombre de clés de session. Ces clés sont connues seulement par les deux bouts de la communication. Dans la deuxième partie du protocole, le client et le serveur utiliseront ces clés pour sécuriser leur communication en utilisant un schéma de chiffrement authentifié. On appelle ce type de schéma « à clé symétrique » : notamment, on chiffre et on déchiffre avec la même clé.

Si la clé générée est « bonne » (d'une bonne taille et indistinguable d'une clé aléatoire de la même taille), alors les propriétés garanties par ces schémas sont :

- La confidentialité : Le contenu des messages échangés reste caché par rapport à un attaquant
- L'authenticité : Le receveur d'un message peut être sûr de l'identité de son envoyeur
- L'intégrité : Le receveur d'un message est assuré que le message n'a pas été modifié

Exercice 1 : URL

Regardez les exemples d'URLs suivants. Relevez dans chaque cas les éléments de chaque URL par rapport aux composants listés ci-dessus :

- `http://164.81.20.1/doc.html`

- `http://www.unilim.fr:8080/doc.html`

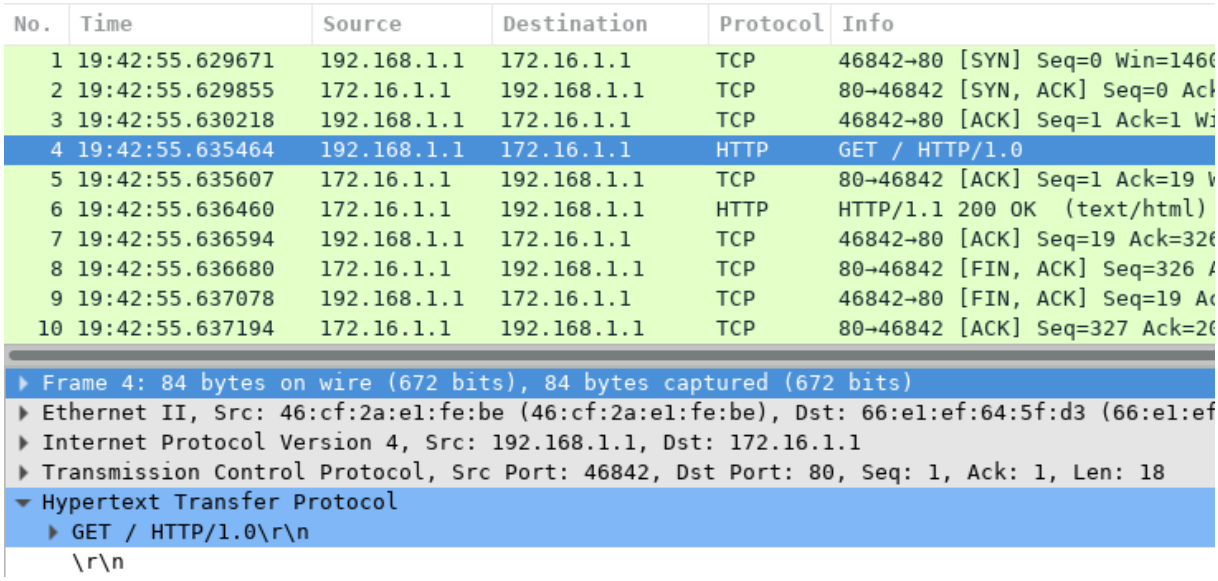
- `http://www.exemple.com:1030/software/index.html`

Exercice 2 : HTTP

1. Un utilisateur saisit l'adresse `monsite.fr` dans son navigateur. Qu'est-ce qu'on peut savoir sur les paramètres suivants du site à partir de son URL ?

    - Le protocole utilisé :
    - Le port :
    - La machine en question :
    - Le document demandé :

2. Nous avons fait la capture suivante sur Wireshark :

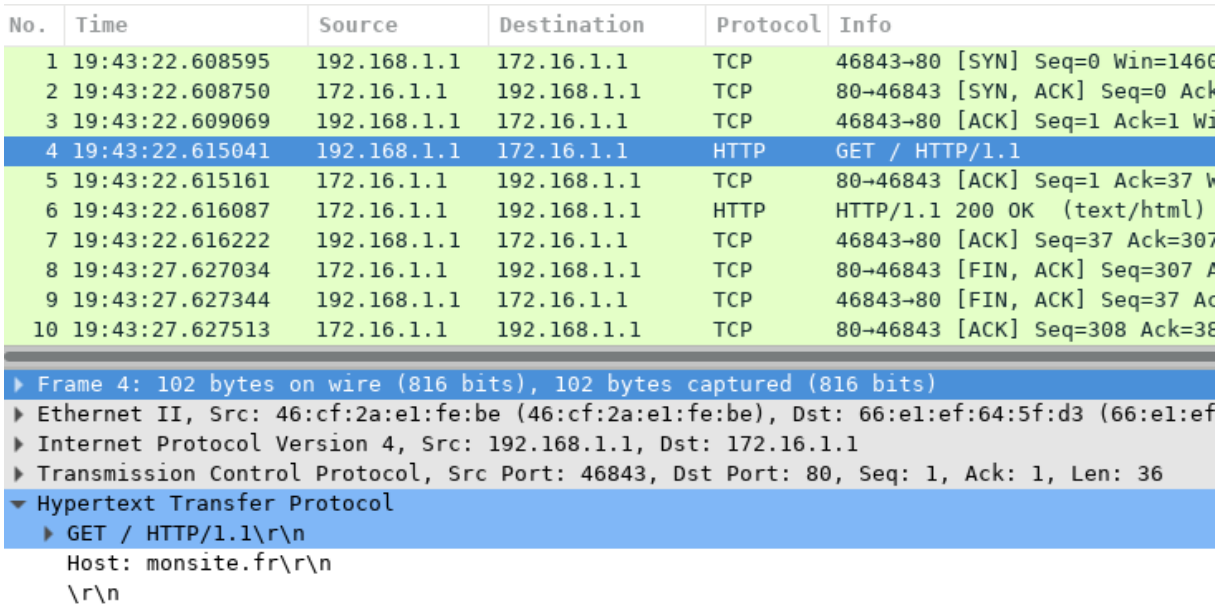


No.	Time	Source	Destination	Protocol	Info
1	19:42:55.629671	192.168.1.1	172.16.1.1	TCP	46842→80 [SYN] Seq=0 Win=1460
2	19:42:55.629855	172.16.1.1	192.168.1.1	TCP	80→46842 [SYN, ACK] Seq=0 Ack
3	19:42:55.630218	192.168.1.1	172.16.1.1	TCP	46842→80 [ACK] Seq=1 Ack=1 Wi
4	19:42:55.635464	192.168.1.1	172.16.1.1	HTTP	GET / HTTP/1.0
5	19:42:55.635607	172.16.1.1	192.168.1.1	TCP	80→46842 [ACK] Seq=1 Ack=19 W
6	19:42:55.636460	172.16.1.1	192.168.1.1	HTTP	HTTP/1.1 200 OK (text/html)
7	19:42:55.636594	192.168.1.1	172.16.1.1	TCP	46842→80 [ACK] Seq=19 Ack=326
8	19:42:55.636680	172.16.1.1	192.168.1.1	TCP	80→46842 [FIN, ACK] Seq=326 A
9	19:42:55.637078	192.168.1.1	172.16.1.1	TCP	46842→80 [FIN, ACK] Seq=19 Ac
10	19:42:55.637194	172.16.1.1	192.168.1.1	TCP	80→46842 [ACK] Seq=327 Ack=20

```
▸ Frame 4: 84 bytes on wire (672 bits), 84 bytes captured (672 bits)
▸ Ethernet II, Src: 46:cf:2a:e1:fe:be (46:cf:2a:e1:fe:be), Dst: 66:e1:ef:64:5f:d3 (66:e1:ef
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 172.16.1.1
▸ Transmission Control Protocol, Src Port: 46842, Dst Port: 80, Seq: 1, Ack: 1, Len: 18
▾ Hypertext Transfer Protocol
  ▸ GET / HTTP/1.0\r\n
    \r\n
```

- Adresse IP et port du client :
- Adresse IP et port du serveur :
- Protocole de transport utilisé :
- Longueur du message de demande :
- Version du protocole utilisée :
- Document demandé :
- Éléments de protocole HTTP :
  - Question :
  - Réponse :
- Marqueur de fin de message :
- Durée totale de l'échange :
- Nombre de messages « utiles » :

3. Dans le message ci-dessous vous voyez les mêmes échanges que dans l'exercice précédent. Cette fois-ci on utilise une autre version du protocole HTTP.

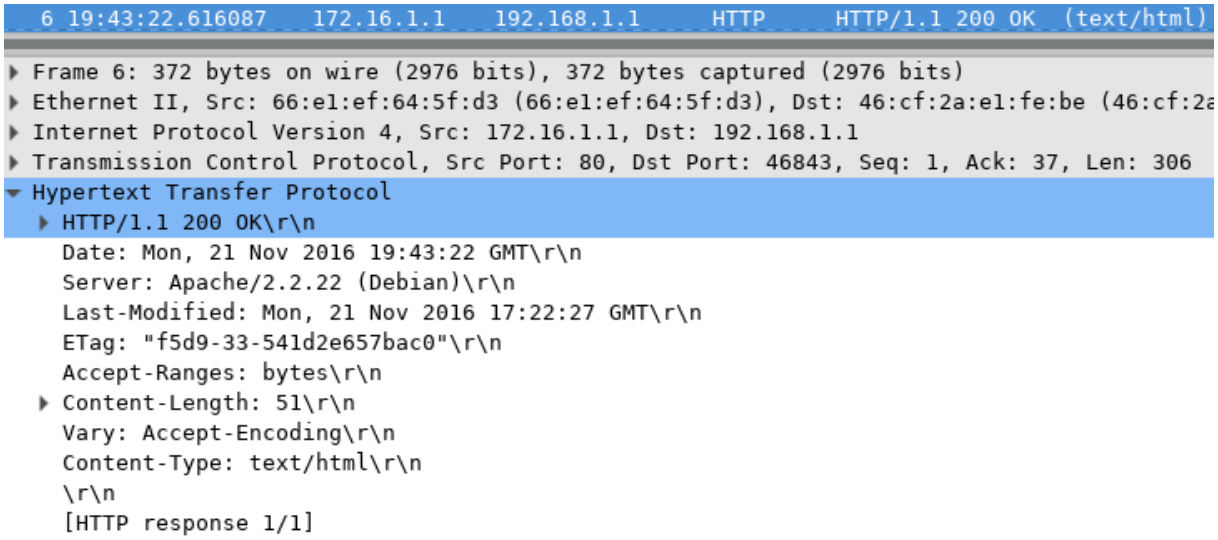


No.	Time	Source	Destination	Protocol	Info
1	19:43:22.608595	192.168.1.1	172.16.1.1	TCP	46843→80 [SYN] Seq=0 Win=1460
2	19:43:22.608750	172.16.1.1	192.168.1.1	TCP	80→46843 [SYN, ACK] Seq=0 Ack
3	19:43:22.609069	192.168.1.1	172.16.1.1	TCP	46843→80 [ACK] Seq=1 Ack=1 Wi
4	19:43:22.615041	192.168.1.1	172.16.1.1	HTTP	GET / HTTP/1.1
5	19:43:22.615161	172.16.1.1	192.168.1.1	TCP	80→46843 [ACK] Seq=1 Ack=37 W
6	19:43:22.616087	172.16.1.1	192.168.1.1	HTTP	HTTP/1.1 200 OK (text/html)
7	19:43:22.616222	192.168.1.1	172.16.1.1	TCP	46843→80 [ACK] Seq=37 Ack=307
8	19:43:27.627034	172.16.1.1	192.168.1.1	TCP	80→46843 [FIN, ACK] Seq=307 A
9	19:43:27.627344	192.168.1.1	172.16.1.1	TCP	46843→80 [FIN, ACK] Seq=37 Ac
10	19:43:27.627513	172.16.1.1	192.168.1.1	TCP	80→46843 [ACK] Seq=308 Ack=38

- De quelle version s'agit-il ?
- Quel est l'élément supplémentaire de la demande ?

- Quelle est la durée de l'échange cette fois-ci ?
- Pourquoi peut-on vouloir un échange plus long ?

4. Voici la réponse du serveur Apache. Relevez les éléments demandés ci-dessous :



- Version du serveur et OS :
- Code retour HTTP :
- Longueur du document envoyé :
- Marqueur de fin de l'en-tête :
- Version du protocole utilisée :

La configuration d'un serveur Apache

Les fichiers de configuration

Apache est le serveur HTTP le plus utilisé sur le Web aujourd'hui. C'est un logiciel libre, distribué sous une licence particulière, dont la version actuelle est la version 2.5. Étant très utilisé, Apache est amplement documenté sur Internet, même en langue française.

Le serveur Apache peut se démarrer/arrêter sous Linux comme n'importe quel serveur en utilisant init.d :

```
$ /etc/init.d/apache2 <start|stop|status|restart>
```

Les fichiers de configuration se trouvent en général dans le dossier ***/etc/apache2*** sous Linux. Selon les versions, on va trouver un seul ou plusieurs fichiers de configuration principaux, qui peuvent faire appel à

un certain nombre de modules optionnels. Du chargement ou non de ces modules vont dépendre les fonctionnalités proposées par le serveur. Sous Debian, ce répertoire se présente de la manière suivante :

```
root@interventions:/etc/apache2# ls --color
apache2.conf  envvars     magic           mods-enabled  sites-available
conf.d        httpd.conf  mods-available  ports.conf    sites-enabled
```

En ce qui concerne les directives principales, définissant les fonctionnalités de base du serveur HTTPD, on trouve notamment les directives suivantes qui seront réparties dans les différents répertoires/fichiers :

- `Listen` : précise le port sur lequel le serveur va écouter
- `ServerName` : adresse IP ou nom DNS du serveur
- `DocumentRoot` : répertoire des fichiers proposés par le serveur
- `AllowOverride` : permet de modifier les droits pour un dossier en particulier
- `DirectoryIndex` : nom du fichier par défaut
- `Alias` : définit les alias utilisables dans l'URL
- `ScriptAlias` : définit les alias utilisables pour les scripts CGI

Exercice 3 : Configuration d'un serveur HTTP

On considère le fichier de configuration (extrait) suivant :

```
Listen          80
ServerName      164.81.20.1
DocumentRoot    /var/www/html
DirectoryIndex  page.html

Alias           /images/        "/var/www/images/"
Alias           /perso/         "/var/www/perso/"

ScriptAlias     /cgi/           "/var/www/cgi-bin/"
```

Pour chaque URL, donnez le chemin du fichier transmis par le serveur HTTP, sachant que ***limds1*** est le nom de la machine 164.81.20.1 :

URL	Chemin du fichier transmis
http://164.81.20.1/document.html	
http://164.81.20.1:8080/document.html	

ftp://limds1/page.html	
http://limds1:80	
http://164.81.20.1:80/images/fond.jpg	
http://limds1/perso/perso_html.html	
http://164.81.20.1/cgi/prog	

Réglementer l'accès à un site : *.htaccess*

Dans la plupart des cas aujourd'hui, si un site demande l'authetification d'un utilisateur, ceci sera géré par le site (en https). Cependant, dans des certains cas, nous pouvons vouloir gérer l'authentification par le serveur. Pouvez-vous vous imaginer pourquoi, par exemple ?
L'utilisation de fichier .***htaccess*** permet de limiter très simplement l'accès à tout ou partie d'un site web.

Authentification pour l'accès à un répertoire

La mise en place de l'authentification dans ces cas se fait en 2 étapes. D'un côté le fichier .***htaccess*** qui définit les modalités d'accès, et de l'autre le fichier .***htpasswd*** qui contient les utilisateurs autorisés.

Exemple de fichier .***htaccess***

```
AuthUserFile /etc/httpd/conf/.htpasswd
AuthName Mapage
AuthType basic
<Limit GET>
require valid-user
</Limit>
```

Remarques :

- le fichier caché *.htpasswd*, contenant des mots de passe cachés par une fonction de hachage ou un algorithme de chiffrement assez simpliste, peut porter un nom quelconque et doit se trouver hors de l'arborescence du site Web pour des raisons de sécurité (les méthodes de chiffrement/fonctions de hachage ne sont pas vraiment sécurisées).
- S'il est nécessaire de protéger plusieurs répertoires, il est recommandé de placer tous les fichiers des mots de passe dans le même répertoire.
- La protection d'un répertoire est propagée à tous les sous-répertoires de celui-ci.

Le fichier des mots de passe

Le fichier de mots de passe chiffrés pour chacun des utilisateurs autorisés est créé avec les droits de root. La commande pour créer le fichier *.htpasswd* est `htpasswd` : .

- Si le fichier n'existe pas encore, il faut préciser qu'on désire le créer avec l'option `-c` :

```
htpasswd -c /etc/apache2/.htpasswd Nom_Utilisateur
```

- Pour ajouter un utilisateur au fichier existant, **surtout ne pas utiliser** l'option –c :

```
htpasswd -c /etc/apache2/.htpasswd Nom_Utilisateur
```

Filtrage par adresse IP d'origine

Au lieu de filtrer les usagers en leur demandant un nom d'utilisateur et un mot de passe, le filtrage par adresse IP d'origine (et/ou nom de domaine d'origine) peut se faire de manière encore plus simple en plaçant simplement le ***.htaccess*** approprié dans le répertoire à protéger.

Exemple de fichier ***.htaccess***

```
order allow,deny
allow from all
deny from domaine.tld
deny from 172.16.35.45
```

Le principe des vhosts

Le terme de « Virtual Host » (serveurs virtuels) fait référence à un principe qui permet d'héberger plusieurs sites web (www.domaine1.org, www.domaine2.org ...) sur une même machine. Ceci se fait souvent lorsqu'on demande à un tiers (un fournisseur de services) d'héberger notre site Web. Le principe des vhosts peut s'appuyer sur la technique des alias d'adresses ip « **ip-based** », ou sur la technique de multiples noms adressables sur la même adresse : « **name-based** ».

Technique "IP-based"

La technique IP-based est la plus simple à mettre en œuvre, mais elle est beaucoup plus contraignante au niveau des adresses, puisqu'elle consiste simplement à utiliser une adresse IP différente pour chacun des sites hébergés. Le gain se situe donc au niveau du nombre de machines physique et au niveau du nombre de processus nécessaires.

Technique "named-based"

Il s'agit de faire héberger plusieurs sites sur une même adresse. Le serveur associe le nom d'hôte aux entêtes des requêtes *http* passées par le client. En utilisant cette technique plusieurs « serveurs web» peuvent se partager la même adresse IP. C'est donc celle qui est privilégiée.

Les 2 méthodes nécessitent bien entendu de déclarer correctement les multiples noms de machines au niveau du serveur DNS, et de les associer aux bonnes adresses IP (adresse unique dans le cas de la technique « name-based »).

Mise en oeuvre

Pour la mise en place de cette technique, il suffit de configurer le serveur Apache afin qu'il reconnaisse les différents noms de site, et de déclarer la correspondance entre ces noms et les répertoires contenant les pages des sites (DocumentRoot).

Extrait de *ports.conf*

```
NameVirtualHost *:80
```

Extrait de *001-domain*

```
<VirtualHost *:80>
        ServerName www.domain.tld
        DocumentRoot /www/domain
</VirtualHost>
```

Extrait de *002-otherdomain*

```
<VirtualHost *:80>
        ServerName www.otherdomain.tld
        DocumentRoot /www/otherdomain
</VirtualHost>
```

Les scripts CGI

CGI (Common Gateway Interface) est une méthode standard d'extension des fonctionnalités d'un serveur Web par l'exécution de scripts sur ce serveur, en réponse aux requêtes d'un navigateur Web. Un script CGI peut être un script Shell ou autres (PERL, PhP…), ou bien un programme exécutable classique.

Un script CGI va être lancé par le daemon HTTPD en réponse à une demande provenant du client, soit directement, soit par l'intermédiaire d'une page *html*. Dans les deux cas, le script doit avoir été déclaré comme tel via la directive ScriptAlias dans la configuration d'apache.

Quand un script CGI (c'est-à-dire un exécutable) est démarré par le serveur HTTP, il se situe dans un contexte particulier :

- il a reçu du serveur HTTP un certain nombre de variables d'environnement :
  - l'@IP du client,
  - une méthode de passage de paramètres,
  - les valeurs envoyées par le client,
  - ...
- sa sortie standard est reliée au processus serveur HTTP qui se chargera de transmettre les résultats au navigateur,
- le script CGI s'exécute avec comme UserName Apache,

Voici un exemple de script CGI simple, écrit en shell :

```
#!/bin/bash
echo "Content-type:text/plain"
echo ""
echo -n "Nous sommes le : "
date
echo "Votre adresse IP est $REMOTE_ADDR"
echo "Vous utilisez le navigateur $HTTP_USER_AGENT"
```

Le script CGI ci-dessus doit constituer un document résultat. Il suffit de faire des affichages sur sa sortie standard (commande echo) car elle est indirectement reliée au navigateur. Comme premier affichage, le script doit indiquer la nature du document résultat, en précisant l'entête MIME :

- Content-type: text/plain si le document résultat est un document texte,
- Content-type: text/html si le document résultat est un document html.

Dans ce dernier cas, le script devra constituer un document html valide en mettant des balises, c'est-à-dire en faisant afficher les balises HTML.

Ce script a vocation à être lancé directement par le navigateur, il va simplement utiliser la fonction système date ainsi que des **variables d'environnement** pour créer un affichage simple mais personnalisé pour chaque client web.

L’utilisation d'un script avec un formulaire

Il est plus intéressant encore d'utiliser un script en l'appelant depuis une page HTML de manière à pouvoir lui passer plus de paramètres via une requete de type ***GET*** ou ***POST :***

- méthode **GET** : les paramètres sont dans la variable QUERY_STRING,
- méthode **POST** : les paramètres sont sur l'entrée standard (stdin).

Dans les deux cas, les paramètres sont passés sous la forme d'une *string* de la forme :

```
param1=value1&param2=value2&param3=value3
```

Exercice 4 : Les scripts

Écrivez un script shell capable de renvoyer le contenu du répertoire d'un utilisateur dont le nom est passé en paramètre en utilisant la méthode GET.