HTTP[s]
Hyper Text Transfert Protocol
Apache2 ~ nginx

Pierre Bettens

février 2023
v0.1 ~ May 26, 2023


HTTP(s)
Hyper Text Transfert Protocol
Apache 2 ~nginx

HTTP

C'est **le** protocole du WEB[1]

- Sur TCP
- Port 80 (par défaut)
- 1990 v0.9, 1996 v1.0

RFC
https://www.rfc-editor.org/rfc/rfc9110.html

[1] Du **WEB** (*World Wide Web*), pas de l'**Internet**

HTTP

```
$ nc -C localhost 80
GET / HTTP/1.1
host: localhost
```

Réponse...

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2023 11:02:18 GMT
Server: Apache/2.4.54 (Debian)
Last-Modified: Mon, 20 Jan 2020 09:21:02 GMT
ETag: "44-59c8ecf63ee1e"
Accept-Ranges: bytes
Content-Length: 68
Content-Type: text/html

<h1>Think about</h1>
<p>There are no failures, only lessons.</p>
```

HTTP

```
$ nc -C example.org 80
GET / HTTP/1.1
host: example.org
```

Extrait...

```
HTTP/1.1 200 OK
Age: 541560
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 01 Mar 2023 11:33:27 GMT
(cut)

<!doctype html>
(cut)
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Structure de la requête HTTP

```
GET / HTTP/1.1 host: example.org
```

- `GET`, la méthode
- `/`, la page demandée
- `HTTp/1.1`, le protocole
- `host: example.org`, précise le nom du site (cfr. *vhost*)

À partir de la deuxième ligne, `field-name: token`

Les **méthodes**:

Method Name	Description	Section
GET	Transfer a current representation of the target resource.	9.3.1
HEAD	Same as GET, but do not transfer the response content.	9.3.2
POST	Perform resource-specific processing on the request content.	9.3.3
PUT	Replace all current representations of the target resource with the request content.	9.3.4
DELETE	Remove all current representations of the target resource.	9.3.5
CONNECT	Establish a tunnel to the server identified by the target resource.	9.3.6
OPTIONS	Describe the communication options for the target resource.	9.3.7
TRACE	Perform a message loop-back test along the path to the target resource.	9.3.8

Table 4

Figure 1: Extrait de rfc-editor.org

Les **entêtes**.

L'entête `host`

```
host: example.org
```

- précise quel est le nom de l'hôte
- obligatoire
- permet de distinguer les ressources lorsque le serveur gère plusieurs hôtes virtuels (*vhosts*)

Il existe beaucoup d'autres entêtes de requête (*request context fields*), par exemple

- `referer: <value>`, URI de la ressource d'origine
- `user-agent: <value>`, identifie le client
- `accept-charset: <value>`, jeu de caractères accepté
- ...

... et des entêtes de réponse (*response context fields*)

- `server: <value>`, le type de serveur HTTP
- `content-type: <value>`, le type de document échangé
- ...

La **réponse** HTTP est constituée de *context fields* et commence par un **code de retour** (*status code*).

Extrait d'une réponse

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2023 11:02:18 GMT
Server: Apache/2.4.54 (Debian)
```

Statut 200

OK Standard response for succesful HTTP requests

HTTP

Les *status code* sont catégorisés, le premier chiffre décrit le type de réponse; succès, erreur...

Il y a 5 catégories :

- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

HTTP

Quelques codes de retour (*status code*) fréquents :

- `200` OK
- `301` *Move permanently*
- `307` *Temporary redirect*
- `403` *Forbidden*
- `404` *Not found*
- `500` *Internal server error*
- `502` *Bad gateway*

Know your HTTP [pdf]

HTTP - Cache

HTTP fournit un mécanisme de **cache**; des messages de réponses HTTP peuvent être sauvegardés par le navigateur (*private*) ou un proxy (*public*) sous certaines conditions.

`Cache-Control` est un en-tête HTTP qui dicte le comportement de mise en cache du navigateur.

Exemple

```
Cache-Control: public, max-age=604800
```

HTTP - Cache

Quelques entêtes :

- `max-age=[secondes]` — indique la quantité de temps maximale où la représentation sera considérée fraîche
- `s-maxage=[secondes]` — similaire à `max-age`, sauf qu'elle ne s'applique qu'aux caches partagés (*proxy*)
- `public` — marque les réponses authentifiées comme cachables
- `no-cache` — force à chaque fois les caches à soumettre la requête au serveur original
- `no-store` — instruit les caches de ne pas garder de copie de la représentation dans toutes les conditions
- ...

Cfr.

- https://fr.wikipedia.org/wiki/Cache-Control
- https://developer.mozilla.org/fr/docs/Web/HTTP/Caching

HTTP - Cookie

Un cookie HTTP est une donnée de petite taille envoyée par le serveur au navigateur web de l'utilisatrice ou de l'utilisateur.

Le navigateur peut alors enregistrer le cookie et le renvoyer au serveur lors des requêtes ultérieures.

Les cookies ont trois usages principaux :

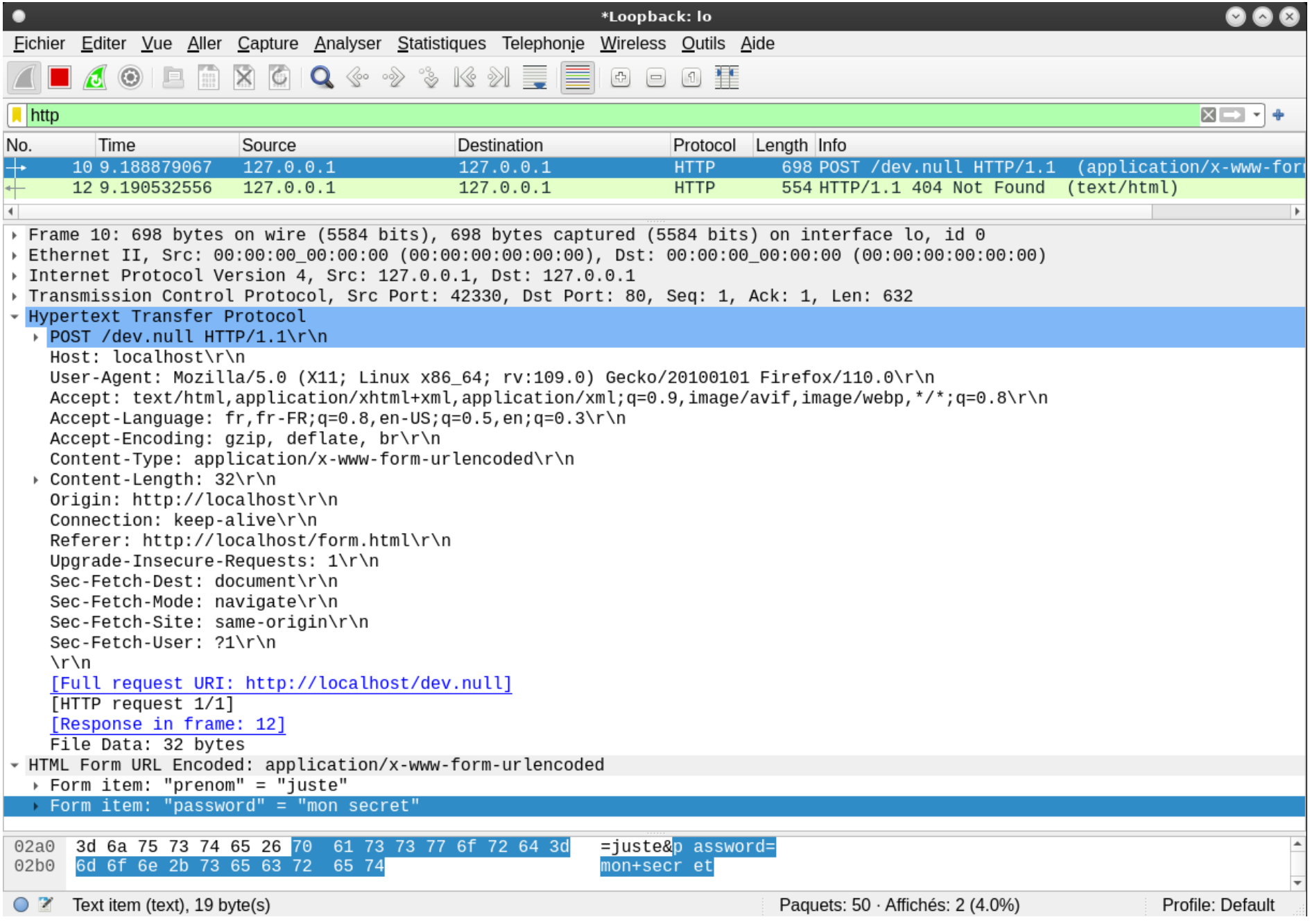
- La gestion de session
- La personnalisation
- Le pistage

Les cookies sont actuellement « remplacés » par *localstorage*, *sessionstorage* et *IndexedDB*.

Cfr. https://developer.mozilla.org/fr/docs/Web/HTTP/Cookies

HTTP - Envoi d'un formulaire

Envoi d'un formulaire


*Loopback: lo
Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide
http
No. Time Source Destination Protocol Length Info
10 9.188879067 127.0.0.1 127.0.0.1 HTTP 698 POST /dev.null HTTP/1.1 (application/x-www-for
12 9.190532556 127.0.0.1 127.0.0.1 HTTP 554 HTTP/1.1 404 Not Found (text/html)
Frame 10: 698 bytes on wire (5584 bits), 698 bytes captured (5584 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 42330, Dst Port: 80, Seq: 1, Ack: 1, Len: 632
Hypertext Transfer Protocol
POST /dev.null HTTP/1.1\r\n
Host: localhost\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate, br\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 32\r\n
Origin: http://localhost\r\n
Connection: keep-alive\r\n
Referer: http://localhost/form.html\r\n
Upgrade-Insecure-Requests: 1\r\n
Sec-Fetch-Dest: document\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-Site: same-origin\r\n
Sec-Fetch-User: ?1\r\n
\r\n
[Full request URI: http://localhost/dev.null]
[HTTP request 1/1]
[Response in frame: 12]
File Data: 32 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "prenom" = "juste"
Form item: "password" = "mon secret"
02a0 3d 6a 75 73 74 65 26 70 61 73 73 77 6f 72 64 3d =juste&p assword=
02b0 6d 6f 6e 2b 73 65 63 72 65 74 mon+secr et
Text item (text), 19 byte(s)
Paquets: 50 · Affichés: 2 (4.0%)
Profile: Default


HTTP...S

HTTPS

- Sur TCP encapsulé dans TLS
- Port 443 (par défaut)

Fonctionnement

1. Le client envoie une demande de connexion HTTPS au serveur.
2. Le serveur répond en envoyant son certificat SSL/TLS au client.
3. Le client vérifie la validité du certificat SSL/TLS en vérifiant l'autorité de certification qui l'a émis.
4. Si le certificat est valide, le client et le serveur établissent une connexion sécurisée.

HTTPS

```
$ openssl s_client -connect example.org:443

(cut)
GET / HTTP/1.1
Host: example.org
```

Voir *outssl*

Serveur web

Serveur web

Mise à disposition de l'information sur le réseau

- Apache2, nginx...
- HTML (CSS/JS)
- PHP, Ruby on rails, ASP...

https://news.netcraft.com/archives/category/web-server-survey/

Apache 2

Apache2

Installation

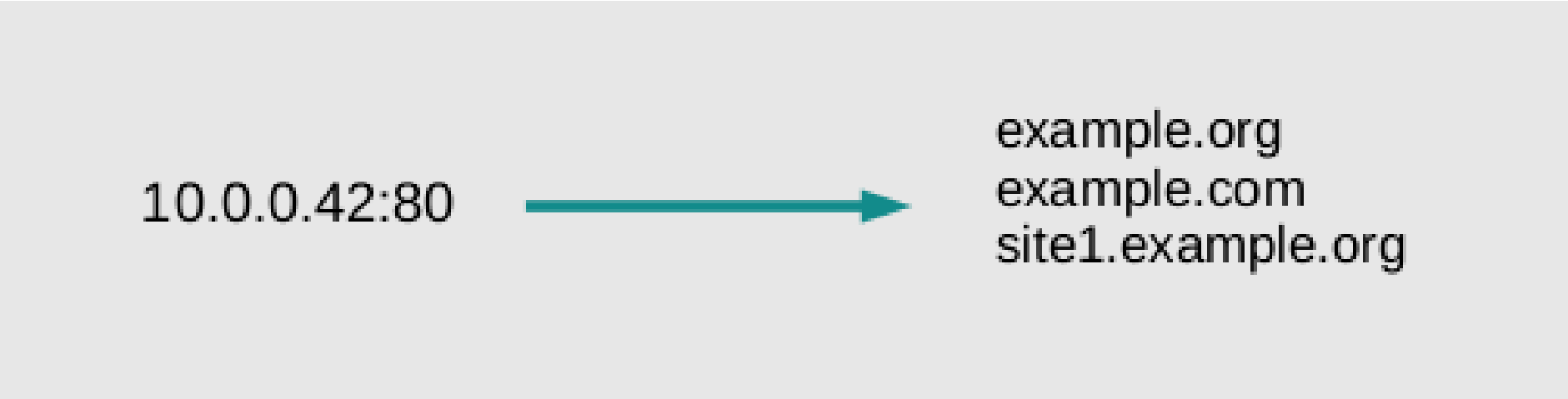
- Paquets à installer : `apache2`, `libapache2-mod-php5`…

Configuration

- Fichiers
  - `apache2.conf`, `conf-[available|enabled]`
  - `sites-[available|enabled]`, gestion des hôtes virtuels (voir plus loin)
  - `mods-[available|enabled]`, gestion des modules
- Commandes `a2en[conf|site|mod]`, `apache2ctl`


Les hôtes virtuels permettent à un serveur web de servir plusieurs sites

Apache2 - Virtual host



Un fichier de configuration par site (*vhost*). Extrait :

```
ServerName      example.org
DocumentRoot    /var/www/html/org.example
ServerAdmin     webmaster@example.org
```

Que fait `a2ensite example.org` ?

Apache2 - MPM

MPM - Modules multi-processus

- Apache fonctionne avec un essaim de *processus*
- Un module MPM, gère le nombre de *processus* et de *threads* lancés par Apache2
- Il existe plusieurs modules MPM: *prefork*, *event*...

Exemple

```
<IfModule mpm_prefork_module>
    StartServers             5
    MinSpareServers          5
    MaxSpareServers          10
    MaxRequestWorkers        150
    MaxConnectionsPerChild   0
</IfModule>
```

Apache2 - ACL

Gestion du contrôle d'accès (*access control list*)

- spécifie les autorisations (*allow*) et les interdictions (*deny*) d'accès à une arborescence d'un serveur
- peut se faire dans la configuration de *vhost* ou dans un fichier `.htaccess`

Un fichier `.htaccess` est un extrait de configuration du serveur se trouvant dans un répertoire servi par le le serveur web.

```
AccessFileName .htaccess
```

Apache2 - ACL

L 'authentification dans HTTP peut se faire grâce aux ACL de différentes manières

- BASIC (RFC 2617)
- DIGEST (RFC 2617)
- SSL/TLS

Une authentification *basic* à travers une connexion TLS est sûre. Une alternative est l'utilisation de certificat pour une authentification *via* ces certificats (voir exemples).

Apache2 - ACL

Exemple

```
Order deny, allow
Deny from all
Allow from 127.0.0.1/8
```

Exemple 2

(Fichier .htaccess à placer dans un répertoire)

```
AuthUserFile access/.htaccess_passwd
AuthGroupFile /dev/null
AuthName "Accès membres"
AuthType Basic

<FilesMatch verysecretfile.html>
<LIMIT GET POST>
require valid-user
</LIMIT>
</FilesMatch>

<FilesMatch .htaccess>
<LIMIT GET POST>
deny from all
</LIMIT>
</FilesMatch>
```

Exemple 3

```
SSLCACertificateFile "conf/ssl.crt/company-ca.crt"

<Directory "/usr/local/apache2/htdocs">
#   En dehors de subarea, seul l'accès depuis l'intranet est
#   autorisé
    Require                 ip 192.168.1.0/24
</Directory>

<Directory "/usr/local/apache2/htdocs/subarea">
#   Dans subarea, tout accès depuis l'intranet est autorisé
#   mais depuis l'Internet, seul l'accès par HTTPS + chiffrement fort + Mot de passe
#   ou HTTPS + chiffrement fort + certificat client n'est autorisé.

#   Si HTTPS est utilisé, on s'assure que le niveau de chiffrement est fort.
#   Autorise en plus les certificats clients comme une alternative à
#   l'authentification basique.
    SSLVerifyClient         optional
    SSLVerifyDepth          1
    SSLOptions              +FakeBasicAuth +StrictRequire
    SSLRequire              %{SSL_CIPHER_USEKEYSIZE} >= 128
```

Apache2 - ACL

Exemple 3 (suite)

```
    #   ON oblige les clients venant d'Internet à utiliser HTTPS
    RewriteEngine        on
    RewriteCond          "%{REMOTE_ADDR}" "!^192\.168\.1\.[0-9]+$"
    RewriteCond          "%{HTTPS}" "!=on"
    RewriteRule          "." "-" [F]

    #   On permet l'accès soit sur les critères réseaux, soit par authentification Basique
    Satisfy              any

    #   Contrôle d'accès réseau
    Require              ip 192.168.1.0/24

    #   Configuration de l'authentification HTTP Basique
    AuthType             basic
    AuthName             "Protected Intranet Area"
    AuthBasicProvider    file
    AuthUserFile         "conf/protected.passwd"
    Require              valid-user
</Directory>
```

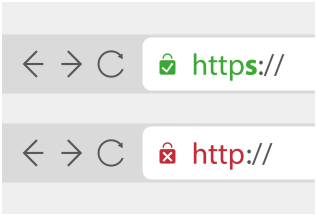
Source https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html

Apache2 - Authentification

Actuellement, l'authentification est plutôt **applicative**

- interface de *login* est intégrée à l'application et peut être paramétrée
- authentification passe à travers TLS puis est (bien) gérée par le *framework*
- peut-être déléguée (*SSO*)
- « facile » pour l'application de gérer ensuite la session
- ...

Qu'est-ce que HTTPS ?



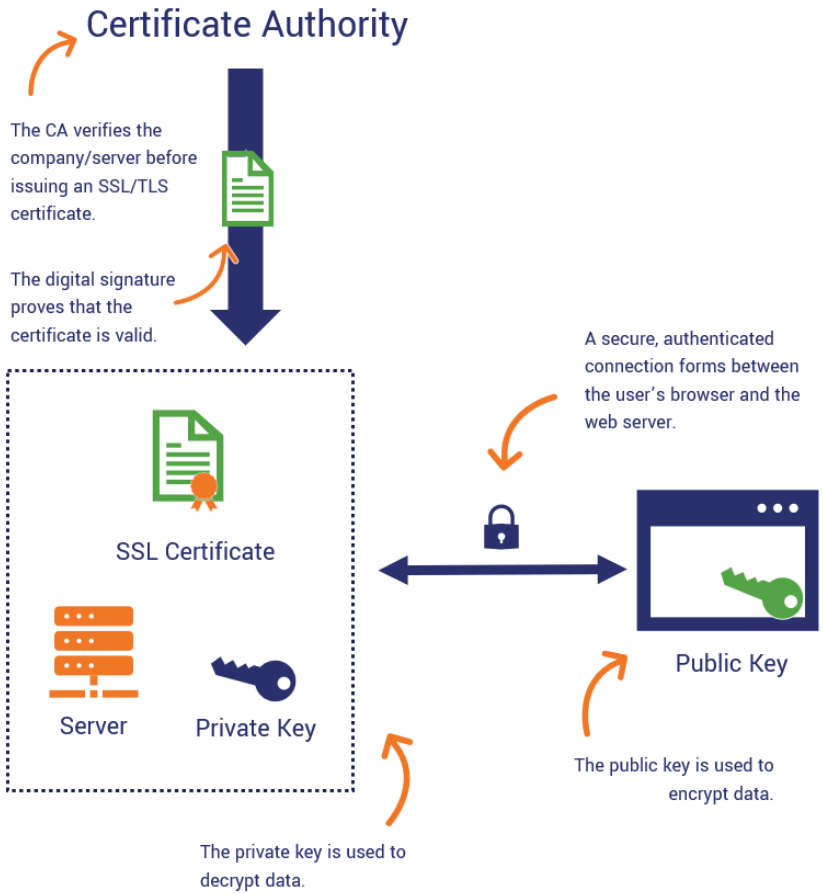
HTTPS, c'est HTTP encapsulé dans TLS.
La connection est sécurisée. Quelle **confiance** ?

- confiance dans le protocole (TLS)
- confiance dans le navigateur (Firefox, Chrome, Edge, Netscape...)
- confiance dans l'autorité de certification (CA)

L'autorité de certification

Les autorités de certification

- connues des navigateurs (*root CA*)
- certifie suivant
  - http
  - dns
  - mail
- il existe EV SSL (*extended validation*)



Source

Obtention du certificat

- payer une autorité de certification
  - créer une CSR (*certificate signing request*) contenant l'identité et une PK (*primary key*)

```
openssl req -nodes -newkey rsa:2048 -sha256
    -keyout myserver.key -out server.csr
```

  - le CA signe avec sa clé privée
- utiliser Let's Encrypt
  - `certbot`
  - `dehydrated`

Configurer Apache pour servir un site en HTTPS

Configurer le *vhost* pour écouter sur le port 443.
Extrait

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerName example.org
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        SSLEngine on
        SSLCertificateFile  /elsewhere/cert.pem
        SSLCertificateKeyFile /elsewhere/key.key
    </VirtualHost>
</IfModule>
```

Rendre le module `mod_ssl` disponible

nginx

Nginx

Installation

- Paquets à installer : nginx, libnginx-mod-*foo*...

Configuration

- Fichiers
  - `nginx.conf`
  - `sites-[available|enabled]`, gestion des hôtes virtuels (*servers blocks*)
  - `modules-[available|enabled]`, gestion des modules

Nginx - *Servers blocks*

Le premier *server block* est celui par défaut

```
http {
    server {
        root /data/www;
    }
}
```

- `server` détermine la configuration du *server block*
- `root` l'endroit où se trouvent les fichiers (`/var/www` est très bien aussi)

Nginx - *Servers blocks*

```
http {
    server {
        location / {
            root /data/www;
        }
        location /images {
            root /data/images;
        }
    }
}
```

- plusieurs *locations* peuvent être choisies pour un même *server block*

Nginx - *Servers blocks*

L'ajout d'une directive server_name permet de répondre pour un hôte virtuel

```
server {
    listen       80;
    server_name  example.org;
    root /data/www/org.example;
    ;...
}
```

Nginx - *Servers blocks*

```
server {
    listen       80;
    server_name  _;
    root /data/www/default;
    ;...
}

server {
    listen       80;
    server_name  example.org;
    root /data/www/org.example;
    ;...
}
```

- server_name _ est un *catch_all*

Nginx - proxy

Un simple *proxy* est un changement de *location*

```
server {
    location / {
        proxy_pass http://localhost:8080;
    }
}
```

Nginx

to be continued next year...

Défi

Écrire dans le langage de son choix, un programme qui ouvre un *socket* sur `example.org:80` et demande la page `index.html`.

Slides pour mes cours.

Pierre Bettens
bettensp@helha.be ~ http://blog.namok.be

Sources

todo

Crédits

Linux, pandoc, beamer, LaTeX