DE LA RECHERCHE À L'INDUSTRIE

Les protocoles web

HTTP

Florent MONJALET

Commissariat à l'énergie atomique et aux énergies alternatives
Direction des applications militaires

26 février 2018

Plan

Sommaire

HTTP : *Hyper Text Transfer Protocol*

- Fournit un moyen d'accèder à tout type de contenu du World Wide Web (WWW)
- À l'origine essentiellement des pages HTML et des éléments multimédia ;
- Spécifications « mixtes » :
  - W3C (World Wide Web Consortium)
  - IETF
- Versions du protocole :
  - HTTP 0.9 (1991)
  - HTTP 1.0 : RFC 1945 (1996)
  - HTTP 1.1 : RFC 2068 puis 2616 (1997, puis 1999)
  - HTTP 2.0 : RFC 7540 (Mai 2015)

- Utilise le port TCP/80 par défaut, non chiffré
- Port 443 pour HTTPS (HTTP Secure, *i.e.* HTTP over TLS)
- Protocole stateless (gestion de l'état laissé aux couches supérieures)
- Basé sur un scénario de requêtes/réponses entre un client et un serveur

Remarques

- Le client est appelé « user agent » et ce n'est pas forcément un navigateur ;
  - Web crawler
  - `curl`
  - La lib `requests` de python
  - …
- Intermédiaires possibles (*proxies*).

La version 1.0 du protocole HTTP

La première version (`HTTP/0.9`) ne spécifie qu'un moyen simpliste de transférer du contenu sur Internet (GET de HTML uniquement)
À partir de la version 1.0 :

- prend en charge les messages au format MIME ;
- contient des métainformations sur les données transférées ;
- permet de spécifier des paramètres sur les requêtes des clients.

La version 1.1 du protocole HTTP

Résolution de problèmes apportés avec la version 1.0, meilleure gestion :

- Des intermédiaires (*proxies*) ;
- De la mise en cache des informations ;
- De la persistance des connexions ;
- Des virtual hosts ou vhosts :
  - Comment héberger plusieurs sites web sur une même adresse IP ?
  - Hébergement basé sur le nom vs. hébergement basé sur l'IP

Support simultané des deux versions du protocole.

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)
- Serveur web (Nginx, Apache, lighttpd)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)
- Serveur web (Nginx, Apache, lighttpd)
- Application web (tomcat, uwsgi + django, mongrel + rails, serveur web go)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)
- Serveur web (Nginx, Apache, lighttpd)
- Application web (tomcat, uwsgi + django, mongrel + rails, serveur web go)
- Autres services :

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)
- Serveur web (Nginx, Apache, lighttpd)
- Application web (tomcat, uwsgi + django, mongrel + rails, serveur web go)
- Autres services :
  - Bases de données (Postgresql, Oracle, Mongodb, Elasticsearch, LDAP...)

Éléments impliqués

- User Agent (script ou navigateur, html, js, css, plugins...)
- Proxy (squid, burp, zap)
- Reverse-proxy (cache, load-balancer... Nginx, HAProxy, Varnish)
- Serveur web (Nginx, Apache, lighttpd)
- Application web (tomcat, uwsgi + django, mongrel + rails, serveur web go)
- Autres services :
  - Bases de données (Postgresql, Oracle, Mongodb, Elasticsearch, LDAP...)
  - Applications tierces (web ou autre)

Sommaire

Parenthèse sur les URIs

URI : Uniform Resource Identifer

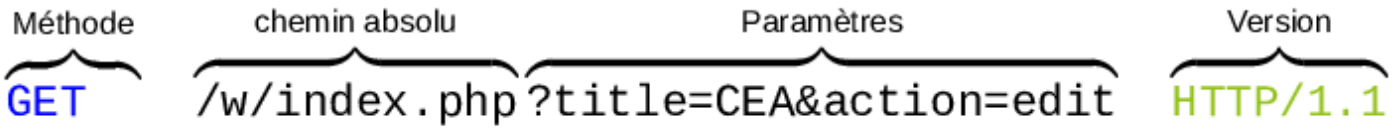
- Une chaîne de caractères identifiant une ressource par différents moyens (nom, adresse réseau, chemin d'accès, paramètres, etc.)
- RFC 3986 (contient des exemples très clairs)
- Classiquement de la forme suivante :

scheme : [//[**user**[: **password**]@]**host**[: **port**]]/**path**[?**query**][#**fragment**]

- En réalité notion plus générique :
  - `http://example.com/toto`
  - `svn+ssh://mylogin@myserver/mydir/myrepo`
  - `git://github.com/cea-sec/miasm.git`
  - `cpe:/o:linux:linux_kernel:2.6.39` (Common Platform Enumeration, description d'OS/logiciel/matériel)
  - `urn:isbn:0-486-27557-4` (International Standard Book Number)

URL : Universal Resource Locator, cas particulier d'une URI désignant un moyen d'atteindre une ressource.

Requête HTTP

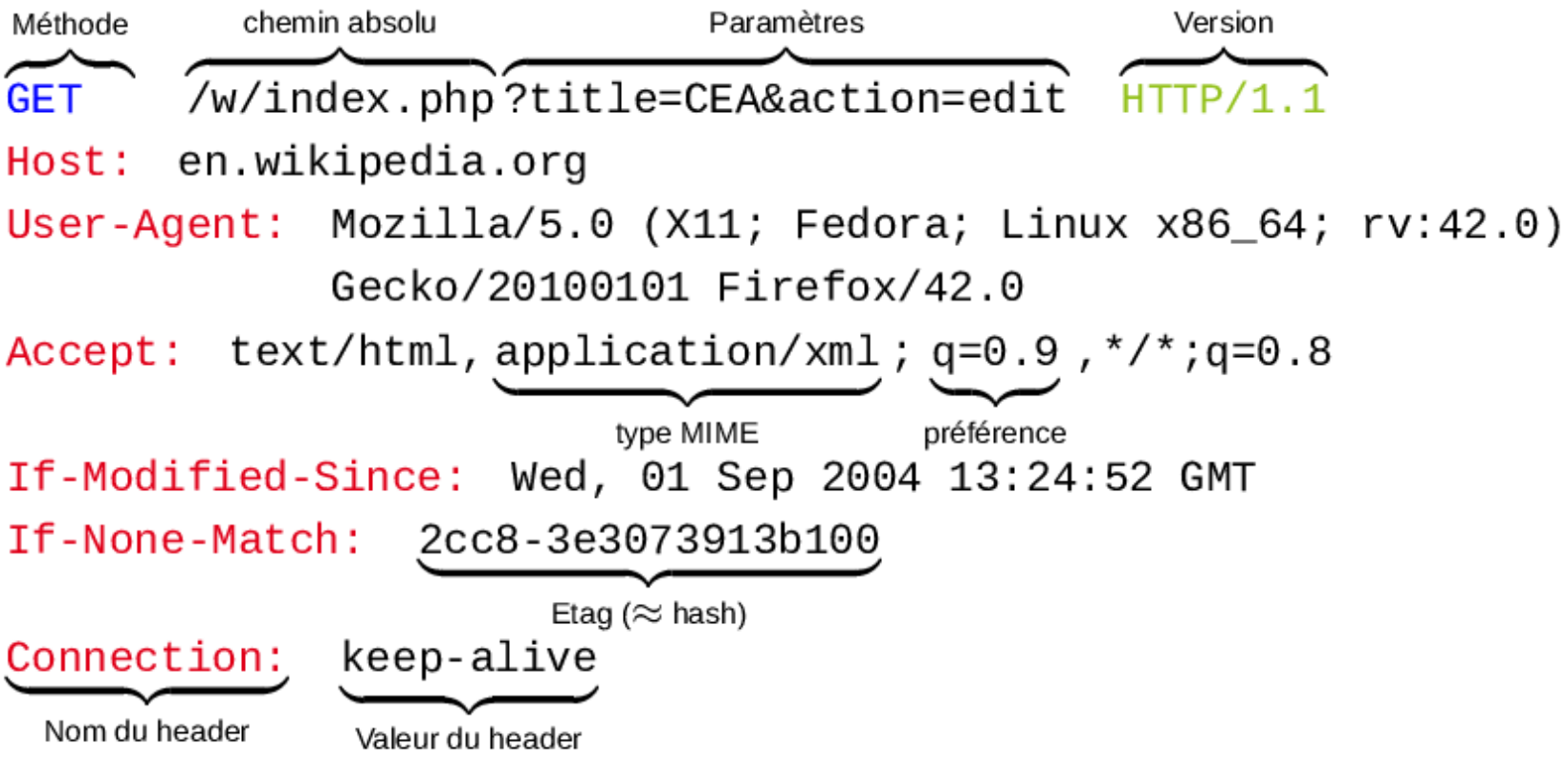


Requête HTTP

Méthode | chemin absolu | Paramètres | Version

```
GET  /w/index.php?title=CEA&action=edit  HTTP/1.1
Host: en.wikipedia.org
```

Requête HTTP



Requête HTTP

Méthode — chemin absolu — Paramètres — Version

```
GET /w/index.php?title=CEA&action=edit HTTP/1.1
Host: en.wikipedia.org
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:42.0)
            Gecko/20100101 Firefox/42.0
Accept: text/html, application/xml ; q=0.9 ,*/*;q=0.8
If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT
If-None-Match: 2cc8-3e3073913b100
Connection: keep-alive
```

type MIME — préférence

Etag (≈ hash)

Nom du header — Valeur du header

Note : Lorsqu'un proxy est utilisé, `Host` est facultatif et la première ligne devient une URL complète :

```
GET http://en.wikipedia.org/w/index.php?title=CEA HTTP/1.1
```

Requête HTTP

La requête envoyée par le client contient :

- une méthode (ex : GET, POST, etc.) ;
- un chemin ou une URL
- une version du protocole (typiquement : `HTTP/1.1`) ;
- des informations optionnelles encodées dans les entêtes HTTP, qui permettent de :
  - positionner des paramètres relatifs à la connexion,
  - transmettre des informations à propos du client (note : tous les entêtes sont optionnels à l'exception de « `Host:` » pour la version 1.1 du protocole) ;
- (éventuellement) un corps de message.

De plus :

- Chaque ligne se termine par <CRLF>
- Les *headers* sont séparés du contenu par une ligne vide

Réponse HTTP

Version | Status code | Descriptif

```
HTTP/1.1 404 Not Found
Content-Encoding: gzip
Content-Language: en
Content-Length: 42
Content-Type: text/html; charset=UTF-8
Date: Sun, 13 Dec 2015 14:24:54 GMT
Last-Modified: Sat, 12 Dec 2015 00:29:30 GMT
Server: Apache/1.3.37

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /test was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.37 Server at 192.168.0.1 Port 80</ADDRESS>
</BODY></HTML>
```

Réponse HTTP

Le serveur répond avec :

- Une ligne (status line) contenant la version du protocole, un code numérique de statut et éventuellement un texte bref ;
- Le reste (*i.e.* entêtes + corps) du message de réponse au format MIME.

Exemples de headers

Quelques exemples de *headers* de requête HTTP :

Host Hostname du serveur. Obligatoire en HTTP 1.1 si non spécifié dans la première ligne. Sert pour différentier plusieurs vhosts.

User-Agent Description de l'*user agent*.

Accept Types MIME acceptés par le client, avec éventuellement un indice de préférence (q=x, avec $x \in [0., 1.]$).

Accept-Encoding Encodages particuliers acceptés par l'*user agent* (gzip, deflate...).

Content-Type Dans le cas où de la donnée est associée à la requête, son type MIME.

Content-Length La taille en octets de l'éventuelle donnée.

Content-Encoding L'encodage de l'éventuelle donnée.

Connection Sert pour demander explicitement de maintenir une connexion TCP ouverte (utilisé lorsque l'*user agent* fait plusieurs requêtes successives).

Exemples de headers

D'autres exemples liés à la gestion du cache :

If-Modified-Since Le serveur ne retourne le contenu que s'il a été modifié depuis une date donnée (sinon l'*user agent* utilise son cache)

If-None-Match Idem mais en comparant à l'*ETag*, un identifiant pour un contenu.

Autre exemple :

X-Some-Header Header (généralement) non standard.

Exeples de headers

Quelques exemples de *headers* de réponse :

Content-*	Comme pour les requêtes avec données.
Date	Date à laquelle le contenu a été envoyé.
Last-Modified	Date de dernière modification de la ressource (sert pour les caches).
Vary	Forcer à ne pas utiliser le cache du serveur ou du mandataire si certains headers de la requête ont changé (typiquement `Accept-Encoding`, pour éviter par exemple de resservir un contenu *gzipé* à un *user agent* qui ne sait pas le lire).
Cache-Control	Donne des directives que les caches doivent respecter (`max-age: N`, `no-cache`...).

Sommaire

Les méthodes HTTP

GET	Récupération d'une ressource
HEAD	Idem que GET mais le corps de la réponse n'est pas transmis
POST	Soumettre (*i.e.* « envoyer ») des données (formulaire), ces dernières étant encodées dans le corps de la requête
PUT, DELETE	Création et suppression de ressources
TRACE	Affichage de la requête reçue par le serveur ($\approx$ echo)
OPTIONS	Retourne les méthodes supportées et d'autres informations complémentaires
CONNECT	Mise en place d'un tunnel applicatif

Notion de méthodes *sûres*

Idée : l'utilisateur (client) et l'*user agent* doivent pouvoir prévoir la portée d'une action

- `GET/HEAD` : aucune incidence (récupération d'informations) $\Rightarrow$ Méthodes sûres
- `POST/PUT/DELETE` : modifications possibles (effets de bord) $\Rightarrow$ Méthodes non sûres

En conséquence, le navigateur doit représenter les méthodes différemment d'un point de vue visuel :

- par exemple, pour POST, par l'utilisation d'un bouton « submit » à la place d'un lien hypertexte.

Notion de méthodes *sûres*

Plus important, c'est un code de conduite pour l'*user agent* :

- Un crawler ne doit utiliser que des méthodes sûres pour ne pas avoir d'incidence
- Un navigateur ne peut rejouer automatiquement que des méthodes *sûres* (*e.g.* HEAD puis GET)
- Si une requête a un effet de bord (non sûre), un proxy doit transférer la requête au serveur au lieu de servir une réponse en cache
- Considérations pour la protection contre le CSRF

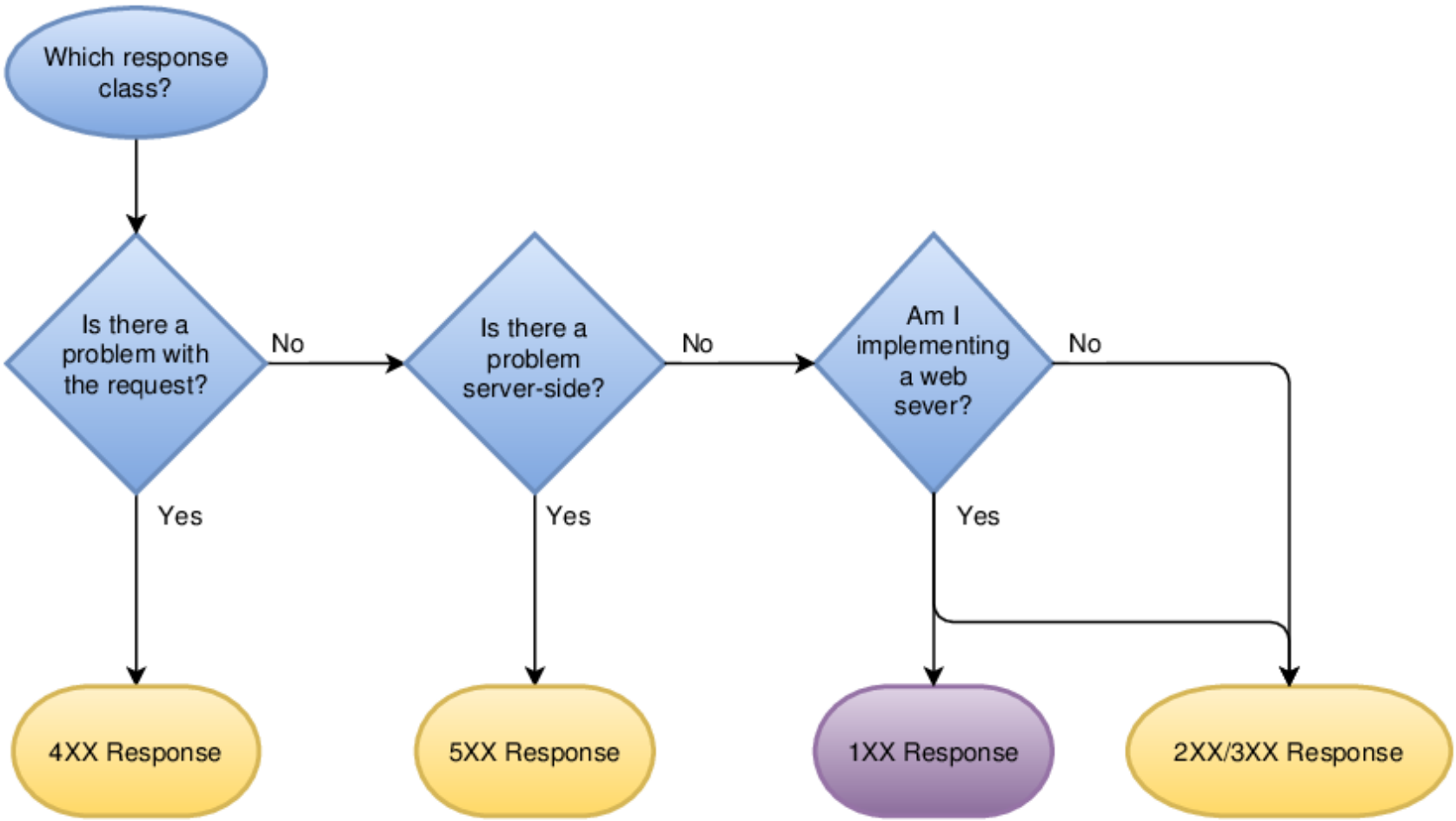
Et pour les implémenteurs d'API web :

- Cohérence des effets de bord des méthodes
- Sémantique des méthodes (*e.g.*PUT, POST et DELETE différents)

Les codes numériques de statut

- 1xx : information (`100 Continue`, `101 Switching Protocols`)
- 2xx : succès (la requête a été correctement reçue, analysée et traitée, par exemple : `200 OK`)
- 3xx : redirection (le client doit accomplir une action complémentaire, par exemple : 301 *Moved Permanently*, 304 *Not Modified*)
- 4xx : erreur de la requête client, par exemple 404 (Not Found) ;
- 5xx : erreur au niveau du serveur, par exemple 500 *Internal Server Error*

Les codes numériques de statut



Source : `http://racksburg.com/choosing-an-http-status-code/`

Codes d’erreur atypiques

- `418 I’m a teapot` (RFC 2324, poisson d’avril)
- `420 Enhance Your Calm` (utilisé par Twitter pour du *rate limiting* sur leur API)
- `450 Blocked by Windows Parental Controls`

Sommaire

Connexions persistantes

Par défaut, les connexions HTTP/1.0 ne sont pas persistantes :

- Une nouvelle session TCP est ouverte pour chaque nouvelle ressource à télécharger sur le serveur ;
- Entête `Connection: keep-alive`

En HTTP/1.1 les connexions sont persistantes par défaut

- Découpage des données transférées (*chunked encoding*)
- Support d'un pipeline de requêtes (en mode FIFO)
- Problème du *head-of-line blocking*

En HTTP/2 :

- Vrai multiplexage des requêtes/réponses

La notion d'état en HTTP

HTTP est un protocole « sans état » (*stateless*) :

- Les requêtes sont indépendantes les unes des autres ;
- Non adapté aux applications web modernes :
  - La gestion de l'état est déléguée à l'application
  - Variables transmises dans les formulaires, l'URL et les *headers*
  - Utilisation de *cookies*.

Le serveur peut fournir différentes versions pour une même ressource en fonction des capacités du client (langue, encodage).

Exemple : entête *Accept-Language*

```
Accept-Language: fr; q=1.0, en; q=0.5
```

HTTPS : HTTP Secure (ou HTTP over SSL), encapsulation des échanges HTTP dans une connexion SSL ou TLS

- https ://domaine.com/index.html

SHTTP : autre mode basé sur un entête `Connection: Upgrade`, ne chiffre que le *contenu* des pages servies ou du contenu envoyé. **Quasiment pas utilisé.**

Remarques

- HTTPS n'assure QUE la confidentialité des échanges réseau ;

HTTPS : HTTP Secure (ou HTTP over SSL), encapsulation des échanges HTTP dans une connexion SSL ou TLS

- https ://domaine.com/index.html

SHTTP : autre mode basé sur un entête `Connection: Upgrade`, ne chiffre que le *contenu* des pages servies ou du contenu envoyé. **Quasiment pas utilisé.**

Remarques

- HTTPS n'assure QUE la confidentialité des échanges réseau ;
- Assure également l'authenticité du serveur, à condition de n'avoir que des AC racine de confiance ;

HTTPS : HTTP Secure (ou HTTP over SSL), encapsulation des échanges HTTP dans une connexion SSL ou TLS

- https ://domaine.com/index.html

SHTTP : autre mode basé sur un entête `Connection: Upgrade`, ne chiffre que le *contenu* des pages servies ou du contenu envoyé. **Quasiment pas utilisé.**

Remarques

- HTTPS n'assure QUE la confidentialité des échanges réseau ;
- Assure également l'authenticité du serveur, à condition de n'avoir que des AC racine de confiance ;
- SSL protège la communication réseau, pas le contenu applicatif (aucune protection contre les XSS, par exemple)

HTTPS : HTTP Secure (ou HTTP over SSL), encapsulation des échanges HTTP dans une connexion SSL ou TLS

- https ://domaine.com/index.html

SHTTP : autre mode basé sur un entête `Connection: Upgrade`, ne chiffre que le *contenu* des pages servies ou du contenu envoyé. **Quasiment pas utilisé.**

Remarques

- HTTPS n'assure QUE la confidentialité des échanges réseau ;
- Assure également l'authenticité du serveur, à condition de n'avoir que des AC racine de confiance ;
- SSL protège la communication réseau, pas le contenu applicatif (aucune protection contre les XSS, par exemple)
- *sslstrip* et `Strict-Transport-Security`

HTTPS : HTTP Secure (ou HTTP over SSL), encapsulation des échanges HTTP dans une connexion SSL ou TLS

- https ://domaine.com/index.html

SHTTP : autre mode basé sur un entête `Connection: Upgrade`, ne chiffre que le *contenu* des pages servies ou du contenu envoyé. **Quasiment pas utilisé.**

Remarques

- HTTPS n'assure QUE la confidentialité des échanges réseau ;
- Assure également l'authenticité du serveur, à condition de n'avoir que des AC racine de confiance ;
- SSL protège la communication réseau, pas le contenu applicatif (aucune protection contre les XSS, par exemple)
- *sslstrip* et `Strict-Transport-Security`
- Interception SSL et `Public-Key-Pins`

HTTPS et proxy

Usage avec un proxy :

- Création d'un tunnel sur HTTP avec la méthode `CONNECT`
- Après le `CONNECT`, le proxy ne fait plus que relayer les paquets HTTPS (paquets TLS avec un payload chiffré HTTP)
- Cela permet d'appliquer un filtrage sur un nom de domaine malgré HTTPS

Alternativement : interception SSL

- Délégation de la vérification du certificat au proxy
- Problèmes de confidentialité
- Le certificat du proxy doit être accepté par les clients pour tous les domaines

Sommaire

Authentification HTTP

Authentification :

1. L'*user agent* demande l'accès à une ressource
2. Le serveur répond avec un `401 Unauthorized` et un *header* `WWW-Authenticate: <method> [<args>]`
3. L'*user agent* répond avec un header `Authorization: <auth_data>`, où la donnée d'authentification dépend de la méthode.
4. - Si l'authentification est réussie, le serveur retourne le code de statut adapté (classiquement `200 OK`) et la ressource demandée
   - Si l'authentification échoue (ou nécessite des étapes supplémentaires), `goto step 2`
   - Si l'authentification réussit mais que l'utilisateur n'a pas le droit d'accéder à la ressource, le serveur renvoie un `403 Forbidden`

Authentification HTTP

Les principales méthodes d'authentification HTTP sont :

- **Basic :** `base64(user:password[:realm])` :
  - Dans un navigateur, boîte de dialogue utilisateur/mot de passe
  - **Passe en clair sur le réseau si HTTPS n'est pas utilisé**.
- **Digest :** challenge/response. Le mot de passe ne passe pas en clair, mais le serveur doit stocker l'information d'authentification de manière réversible.
- **Negotiate :** authentification GSSAPI (par exemple avec Kerberos), NTLM ou autre.

Authentification HTTP

Les principales méthodes d'authentification HTTP sont :

- **Basic :** `base64(user:password[:realm])` :
  - Dans un navigateur, boîte de dialogue utilisateur/mot de passe
  - **Passe en clair sur le réseau si HTTPS n'est pas utilisé**.
- **Digest :** challenge/response. Le mot de passe ne passe pas en clair, mais le serveur doit stocker l'information d'authentification de manière réversible.
- **Negotiate :** authentification GSSAPI (par exemple avec Kerberos), NTLM ou autre.

Souvent, l'authentification est prise en charge par l'application :

- Pas d'authentification au niveau HTTP
- Souvent : `POST` du login/mot de passe
- Utilisation d'un header `Cookie` permettant de tracer la session que l'utilisateur a authentifié
- Authentification via OpenID, autorisation via OAuth
- L'authentification comme la transmission du cookie repose très souvent sur la confidentialité apportée par HTTPS

Sommaire

Évolutions récentes

- WebSocket
  - Protocole full-duplex sur TCP/80 ;
  - Communication bi-directionnelle permanente ;
  - Utilise des en-têtes HTTP pour la compatibilité avec les proxies ;
  - Orienté applications dynamiques.

Évolutions récentes

- WebSocket
  - Protocole full-duplex sur TCP/80 ;
  - Communication bi-directionnelle permanente ;
  - Utilise des en-têtes HTTP pour la compatibilité avec les proxies ;
  - Orienté applications dynamiques.
- Google SPDY
  - S'appuie sur HTTPS ;
  - Multiplexage des requêtes dans une connexion ;
  - Compression des headers (deflate), priorisation des requêtes ;
  - Réduction forte du temps de chargement des pages ;
  - Implémentation dans les navigateurs majeurs (théoriquement plus supporté en faveur d'HTTP/2).

Évolutions récentes

- WebSocket
  - Protocole full-duplex sur TCP/80 ;
  - Communication bi-directionnelle permanente ;
  - Utilise des en-têtes HTTP pour la compatibilité avec les proxies ;
  - Orienté applications dynamiques.
- Google SPDY
  - S'appuie sur HTTPS ;
  - Multiplexage des requêtes dans une connexion ;
  - Compression des headers (deflate), priorisation des requêtes ;
  - Réduction forte du temps de chargement des pages ;
  - Implémentation dans les navigateurs majeurs (théoriquement plus supporté en faveur d'HTTP/2).
- Microsoft SM (Speed+Mobility)
  - S'appuie sur SPDY + WebSocket ;
  - Orienté mobilité : moins gourmand en processeur (suppression de compression, chiffrement...)
  - Expérimental, non utilisé à ma connaissance

Évolution standardisée : HTTP/2

HTTP/2 :

- Groupe de travail IETF httpbis ;
- RFC 7540, mai 2015
- Nécessité de standardiser les fonctionnalités actuelles :
- Multiplexage (notions de streams entrelacés) ;
- Compression des en-têtes : HPACK, algorithme dédié pour éviter certaines attaques par taille de texte compressé ;
- Gestion de congestion ;
- Gestion de priorités et de dépendances entre requêtes ;
- Lien avec SSL (les grands navigateurs ne l'implémentent *que* sur SSL, même si la norme ne l'impose pas) ;
- Basée sur SPDY ($\Rightarrow$ fin du support de SPDY) ;
- Utilisé par de plus en plus de sites (Wikipedia, Google, Twitter, Trainline...)

Démarrer un échange HTTP/2 :

- **Depuis TLS :** protocole h2 annoncé
- **Depuis HTTP/1.1 :** via header `Upgrade:` h2c et headers spécifiques (non implémenté en pratique)
- **En connection directe en clair :** magic packet :

```
PRI * HTTP/2.0

SM
```

  - Requête HTTP avec un verbe invalide pour HTTP/1.X
  - Testée sur $\sim 2\%$ des serveurs web sur internet au moment de la spécification
  - But : être rejetée proprement par un serveur ne supportant pas HTTP/2 sans tentative de parser le reste du flux

- RFC 7541
- Algorithme conçu pour la compression de headers HTTP
- Tire profit de la redondance d'HTTP
- Génération dynamique d'un dictionnaire de taille fixe associant une paire `header: value` à un nombre
- Dictionnaire maintenu et synchronisé implicitement entre le client et le serveur
- Utilisation d'un code de Huffman pour encoder les chaînes de caractère
- Les headers couramment utilisés sont généralement réduits à un ou deux octet
- Le mécanisme associant un nombre à un header exact limite les attaques par taille de texte compressé
- Possibilité d'empêcher la compression d'un header

Commissariat à l'énergie atomique et aux énergies alternatives CEA/DAM
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019