NSI première - IHM sur le web

Protocole HTTP

Protocole HTTP

Revenons sur l'adresse qui s'affiche dans la barre d'adresse d'un navigateur web et plus précisément sur le début de cette adresse c'est-à-dire le "http"

Selon les cas cette adresse commencera par http ou https (nous verrons ce deuxième cas à la fin de cette activité).

Le protocole (un protocole est ensemble de règles qui permettent à 2 ordinateurs de communiquer ensemble) HTTP (HyperText Transfert Protocol) va permettre au client d'effectuer des requêtes à destination d'un serveur web. En retour, le serveur web va envoyer une réponse.

Voici une version simplifiée de la composition d'une requête HTTP (client vers serveur) :

- la méthode employée pour effectuer la requête
- l'URL de la ressource
- la version du protocole utilisé par le client (souvent HTTP 1.1)
- le navigateur employé (Firefox, Chrome) et sa version
- le type du document demandé (par exemple HTML)
- ...

Certaines de ces lignes sont optionnelles.

Voici un exemple de requête HTTP :

```
GET /mondossier/monFichier.html HTTP/1.1
User-Agent : Mozilla/5.0
Accept : text/html
```

Nous avons ici plusieurs informations :

- "GET" est la *méthode* employée (voir ci-dessous)
- "/mondossier/monFichier.html" correspond l'*URL* de la ressource demandée
- "HTTP/1.1" : la version du *protocole* est la 1.1
- "Mozilla/5.0" : le *navigateur* web employé est Firefox de la société Mozilla
- "text/html" : le client s'attend à *recevoir du HTML*

Revenons sur la *méthode* employée :

Une requête HTTP utilise une méthode (c'est une commande qui demande au serveur d'effectuer une certaine action). Voici la liste des méthodes disponibles :

```
GET, HEAD, POST, OPTIONS, CONNECT, TRACE, PUT, PATCH, DELETE
```

Détaillons 4 de ces méthodes :

- **GET** : C'est la méthode la plus courante pour demander une ressource. Elle est sans effet sur la ressource.
- **POST** : Cette méthode est utilisée pour soumettre des données en vue d'un traitement (côté serveur). Typiquement c'est la méthode employée lorsque l'on envoie au serveur les données issues d'un formulaire.
- **DELETE** : Cette méthode permet de supprimer une ressource sur le serveur.
- **PUT** : Cette méthode permet de modifier une ressource sur le serveur

Réponse du serveur à une requête HTTP

Une fois la requête reçue, le serveur va renvoyer une réponse, voici un exemple de réponse du serveur :

```
HTTP/1.1 200 OK
Date: Thu, 15 feb 2019 12:02:32 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Voici mon site</title>
<head>
  <body>
    <h1>Hello World! Ceci est un titre</h1>
    <p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
  </body>
</html>
```

Nous n'allons pas détailler cette réponse, voici quelques explications sur les éléments qui nous seront indispensables par la suite :

Commençons par la fin (à partir de `<!doctype html>`): le serveur renvoie du code HTML, une fois ce code reçu par le client, il est interprété par le navigateur qui affiche le résultat à l'écran. Cette partie correspond au *corps de la réponse*.

La 1re ligne se nomme la ligne de statut :

- HTTP/1.1 : version de HTTP utilisé par le serveur
- 200 : code indiquant que le document recherché par le client a bien été trouvé par le serveur. Il existe d'autres codes dont un que vous connaissez peut-être déjà : le code 404 (qui signifie «Le document recherché n'a pu être trouvé»).

Les 5 lignes suivantes constituent l'en-tête de la réponse, une ligne nous intéresse plus particulièrement :

```
Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
```

Le serveur web qui a fourni la réponse http ci-dessus a comme système d'exploitation une distribution GNU/Linux nommée "Debian" (pour en savoir plus sur GNU/Linux, c'est celle que nous utiliserons d'ailleurs). "Apache" est le coeur du serveur web puisque c'est ce logiciel qui va gérer les requêtes http (recevoir les requêtes http en provenance des clients et renvoyer les réponses http).

Il existe d'autres logiciels capables de gérer les requêtes http (nginx, microsoft-iis...). NGINX (engine x) dépasse depuis peu (juin 2019) Apache. NGINX et Apache sont installés sur 85% des serveurs web mondiaux.

Le "HTTPS" est la version "sécurisée" du protocole HTTP. Par "sécurisé" en entend que les données sont chiffrées avant d'être transmises sur le réseau et que leur provenance est certifiée.

Les étapes du protocole HTTPS

Voici les différentes étapes d'une communication client - serveur utilisant le protocole HTTPS :

- le client demande au serveur une connexion sécurisée (en utilisant "https" à la place de "http" dans la barre d'adresse du navigateur web)
- le serveur répond au client qu'il est OK pour l'établissement d'une connexion sécurisée. Afin de prouver au client qu'il est bien celui qu'il prétend être, le serveur fournit au client un certificat prouvant son "identité".

  En effet, il existe des attaques dites "man in the middle", où un serveur "pirate" essaye de se faire passer, par exemple, pour le serveur d'une banque : le client, pensant être en communication avec le serveur de sa banque, va saisir son identifiant et son mot de passe, identifiant et mot de passe qui seront récupérés par le serveur pirate. Afin d'éviter ce genre d'attaque, des organismes délivrent donc des certificats prouvant l'identité des sites qui proposent des connexions "https".
- à partir de ce moment-là, les échanges entre le client et le serveur seront chiffrés grâce à un système de "clé publique - clé privée" (nous aborderons le principe du chiffrement par "clé publique - clé privée" en terminale).

Même si un pirate arrivait à intercepter les données circulant entre le client et le serveur, ces dernières ne lui seraient d'aucune utilité, car totalement incompréhensible à cause du chiffrement (seuls le client et le serveur sont aptes à déchiffrer ces données)

D'un point vu strictement pratique il est nécessaire de bien vérifier que le protocole est bien utilisé (l'adresse commence par "https") avant de transmettre des données sensibles (coordonnées bancaires...). Si ce n'est pas le cas, passez votre chemin, car toute personne qui interceptera les paquets de données sera en mesure de lire vos données sensibles.

Concernant un site statique, comme le mien, https n'apporte pas grand chose, seulement l'assurance de l'origine. Vous ne transmettez aucune donnée au site.

Complément

Etablir une connexion HTTP directement dans Python, sans utiliser de navigateur

Il est parfaitement possible d'utiliser Python (ou n'importe quel langage moderne) pour établir une connexion avec un serveur web :

```
>>> import requests # librairie qui gère les connexion HTTP
>>> reponse = requests.get("http://qkzk.xyz")
>>> # on établi une connexion avec mon site
>>> reponse
<Response [200]>
>>> # la connexion est établie correctement (200 signifie OK)
>>> reponses.headers # detail de la reponse du serveur
{'Server': 'GitHub.com', 'Content-Type': 'text/html; charset=utf-8',
'Last-Modified': 'Wed, 09 Oct 2019 16:10:57 GMT', 'ETag': 'W/"5d9e0691-e152"',
'Access-Control-Allow-Origin': '*', 'Expires': 'Sat, 12 Oct 2019 09:20:09 GMT',
'Cache-Control': 'max-age=600', 'Content-Encoding': 'gzip',
# réponse tronquée
}
```

- On peut voir que la connexion est bien établie entre mon client (Python lui même) et le serveur (Github.com) qui héberge mon site.
- Le contenu est en html, comme on s'y attend
- L'encodage utf-8
- Et tout un tas d'information moins pertinantes.

Le contenu de la réponse

```
>>> reponse.content
b'<!DOCTYPE html>
<html lang="en">
<head>
<meta name="generator" content="Hugo 0.57.2" />
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>accueil| qkzk</title> ...'
# tronqué. La suite est le contenu complet de la page d'accueil du site...
</body></html>'
```

Il est possible d'examiner la requête transmise par *le client* lui même au serveur :

```
>>> reponse.request
<PreparedRequest [GET]>
>>> reponse.request.headers
{'User-Agent': 'python-requests/2.22.0',
'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*',
'Connection': 'keep-alive'}
```

C'est la requête qui a été transmise quand on a tapé :

```
>>> reponse = requests.get("http://qkzk.xyz")
```

On a bien transmis une requête GET au serveur, avec toutes les informations voulues. C'est bien Python qui a transmis cette requête.

À faire vous même

1. En utilisant Thonny reproduire les commandes présentées ci-dessus pour joindre successivement : `https://google.com` et `https://google.com/azeaze`
2. Comparez les codes réponses obtenus dans les deux cas. Que signifient-ils ?
3. Que peut-on en déduire concernant la page https://google.com/azeaze ?
4. Mesurez la longueur du contenu de la réponse dans les deux cas.

   Comment expliquer cette différence ?