

Projet Réseaux : DNS, TCP, HTTP, HTTPS

DNS – Résolution de noms

Ce script illustre le fonctionnement du **DNS (Domain Name System)**, qui est un service fondamental d'Internet. Le DNS permet de traduire un **nom de domaine** compréhensible par l'utilisateur en une **adresse IP**, nécessaire pour établir une communication réseau.

Le programme utilise les fonctions réseau de Python pour effectuer une requête DNS et récupérer l'adresse IP associée à un nom de domaine. Cette étape est indispensable avant toute connexion TCP, HTTP ou HTTPS, car les machines communiquent entre elles uniquement à l'aide d'adresses IP.

Ce code permet de comprendre :

- le rôle du DNS dans les communications réseau,
- la séparation entre noms de domaine et adresses IP,
- et l'ordre logique des protocoles utilisés sur Internet.

Le DNS constitue ainsi la **première étape** d'une communication réseau, avant l'établissement d'une connexion TCP et l'échange de données applicatives.

The screenshot shows a dual-pane interface. On the left, a terminal window titled 'dns_lookup.py X' displays the following Python code and its output:

```
dns_lookup.py > ...
1 import socket
2
3 domain = "elhamri.com"
4
5 ip = socket.gethostbyname(domain)
6
7 print(f"Nom de domaine : {domain}")
8 print(f"Adresse IP : {ip}")

PS C:\Users\hp\Desktop\reseaux> & C:\msys64\ucrt64\bins
Nom de domaine : elhamri.com
Adresse IP : 76.223.67.189
PS C:\Users\hp\Desktop\reseaux> 
```

On the right, the Wireshark interface shows captured DNS traffic. A table of captured packets is visible, with the last two entries highlighted:

No.	Time	Source	Destination	Protocol	Length	Info
1036	11:51:44.71	192.168.0.100	192.168.0.100	DNS	71	Standard query 0x0af2 A elhamri.com
1050	11:53:41.22	192.168.0.100	192.168.0.100	DNS	155	Standard query response 0x0af2 A elhamri.com A 76.223.67.189 A 13.248.213.45

Below the table, the packet details and bytes panes show the DNS request and response frames. The response frame is expanded to show the query and answer sections, confirming the resolution of 'elhamri.com' to both '76.223.67.189' and '13.248.213.45'.

2. TCP – Client / Serveur (socket)

Ce code illustre le fonctionnement **fondamental du protocole TCP** à travers une communication client/serveur réalisée avec la bibliothèque “socket” de Python.

Le **serveur TCP** commence par créer un socket, puis se met en écoute sur une adresse IP et un port spécifiques. Il attend qu'un client se connecte. Lorsque la connexion est établie, le serveur accepte la requête du client, reçoit les données envoyées, les affiche, puis renvoie une réponse. Une fois l'échange terminé, la connexion est fermée.

Le **client TCP**, de son côté, crée également un socket et se connecte au serveur en utilisant son adresse IP et son port. Il envoie un message au serveur, attend la réponse, l'affiche, puis ferme la connexion.

Ce programme permet de comprendre les concepts essentiels du protocole TCP :

- l'établissement d'une connexion fiable entre deux machines,
- l'échange bidirectionnel de données,
- la notion de port réseau,
- et la fermeture propre de la connexion.

Cette communication TCP constitue la **base de nombreux protocoles réseau**, comme HTTP et HTTPS, étudiés dans les parties suivantes du projet.

```
server.py
1 import socket
2
3 server = socket.socket()
4 server.bind(("127.0.0.1", 12346))
5 server.listen(1)
6
7 print("Le serveur attend une connexion...")
8
9 client, address = server.accept()
10 print("Client connecté : ", address)
11
12 message = client.recv(1024).decode()
13 print("Message reçu : ", message)
14
15 client.send("Bonjour client".encode())
16
17 client.close()
18 server.close()
```

```
client.py
1 import socket
2
3 client = socket.socket()
4 client.connect(("127.0.0.1", 12346))
5
6 client.send("Bonjour serveur".encode())
7
8 response = client.recv(1024).decode()
9 print("Réponse du serveur : ", response)
10
11 client.close()
```

```
powershell
PS C:\Users\hp\Desktop\reseau> python client.py
Réponse du serveur : Bonjour client
PS C:\Users\hp\Desktop\reseau>
```

3. HTTP – Serveur HTTP simple

Ce code met en œuvre un **serveur HTTP simple** en utilisant le module `http.server` fourni par Python. Il permet de comprendre le fonctionnement du protocole HTTP, qui est le protocole utilisé par les navigateurs web pour communiquer avec les serveurs.

Le serveur écoute sur un port défini et attend les requêtes envoyées par un client, généralement un navigateur web. Lorsqu'un client accède à l'adresse du serveur, une **requête HTTP de type GET** est envoyée pour demander une ressource, par exemple la page d'accueil (/).

Le serveur reçoit cette requête, l'analyse, puis renvoie une **réponse HTTP** contenant :

- un **code de statut** (indique que la requête a été traitée avec succès),
- des **en-têtes HTTP**, qui fournissent des informations sur le type de contenu envoyé,
- et un **corps de réponse**, qui correspond au contenu affiché dans le navigateur.

Ce programme permet d'observer concrètement :

- le format des requêtes HTTP GET,
- le rôle des en-têtes HTTP,
- le fonctionnement des réponses HTTP,
- et la communication entre un navigateur et un serveur web.

Grâce à l'analyse du trafic avec Wireshark, il est possible de voir que les requêtes et réponses HTTP sont envoyées **en clair**, ce qui met en évidence l'absence de chiffrement et prépare la transition vers le protocole HTTPS.

The screenshot shows a development environment with three main windows:

- Code Editor:** Displays the Python script `http_server.py` which defines a simple HTTP server using the `http.server` module. The script creates a server on port 8080 and prints a message indicating it's running.
- Terminal:** Shows the command `python http_server.py` being run, and the output "Serveur HTTP en cours sur le port 8080".
- Browser:** Shows a single-line response "Bonjour depuis mon serveur HTTP" from the server at port 8080.

On the right, a Wireshark capture window titled "Http" shows the raw network traffic. It lists several HTTP requests and responses. One request (Frame 143) is highlighted in yellow, showing the details of the GET / HTTP/1.1 request. The response (Frame 144) is also highlighted in yellow, showing the OK status code and the response body "Bonjour depuis mon serveur HTTP".

```
http_server.py X
http_server.py > SimpleHandler > do_GET
1  from http.server import HTTPServer, BaseHTTPRequestHandler
2
3  class SimpleHandler(BaseHTTPRequestHandler):
4      def do_GET(self):
5          self.send_response(200)
6          self.send_header("Content-type", "text/plain")
7          self.end_headers()
8          self.wfile.write(b"Bonjour depuis mon serveur HTTP")
9
10 server = HTTPServer(("0.0.0.0", 8080), SimpleHandler)
11
12 print("Serveur HTTP en cours sur le port 8080")
13 server.serve_forever()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\hp\Desktop\reseaux> & C:\sys64\ucrt64\bin\python.exe c:/Users/hp/Desktop/reseaux/http_server.py
Serveur HTTP en cours sur le port 8080

< > C Non sécurisé 8080
Bonjour depuis mon serveur HTTP

Http
No. Time Source Destination Protocol Length Info
137 4.278622 192.168.56.1 192.168.56.1 HTTP 546 GET / HTTP/1.1
143 4.279393 192.168.56.1 192.168.56.1 HTTP 44 HTTP/1.0 200 OK (text/plain)
147 4.337116 192.168.56.1 192.168.56.1 HTTP 466 GET /favicon.ico HTTP/1.1
153 4.337909 192.168.56.1 192.168.56.1 HTTP 44 HTTP/1.0 200 OK (text/plain)

Frame 143: Packet, 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF_Loopback, id 0
Null/Lopback
Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.1
Transmission Control Protocol, Src Port: 8080, Dst Port: 50421, Seq: 150, Ack: 503, Len: 0
[3] Reassembled TCP Segments (149 bytes): #139(118), #141(31), #143(0)
Hypertext Transfer Protocol
HTTP/1.0 200 OK\r\n
    Response Version: HTTP/1.0
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
    Server: BaseHTTP/0.6 Python/3.11.7\r\n
    Date: Sun, 21 Dec 2023 23:33:19 GMT\r\n
    Content-type: text/plain\r\n
    \r\n
    [Request in frame: 137]
    [Time since request: 771.000 microseconds]
    [Request URI: /]
    [Full request URL: http://192.168.56.1:8080/]
    File Data: 31 bytes
    Line-based text data: text/plain (1 lines)
    Bonjour depuis mon serveur HTTP
```

4. HTTPS – Serveur HTTP sécurisé avec TLS

Cette partie du projet consiste à sécuriser les communications HTTP en utilisant le protocole **HTTPS**, qui repose sur le chiffrement **TLS (Transport Layer Security)**. Le serveur HTTPS est implémenté en Python à l'aide des modules “http.server” et “ssl”.

Un certificat numérique auto-signé est généré à l'aide d'OpenSSL. Ce certificat permet au serveur de chiffrer les données échangées avec le client. Lorsqu'un navigateur se connecte au serveur via HTTPS, un **handshake TLS** est effectué afin d'établir une connexion sécurisée.

Le navigateur affiche un avertissement de sécurité indiquant que la connexion n'est pas privée. Cet avertissement est normal dans un environnement de test, car le certificat utilisé n'est pas signé par une autorité de certification reconnue. Malgré cet avertissement, les échanges sont bien chiffrés.

L'analyse du trafic réseau avec Wireshark montre que, contrairement à HTTP, les données HTTPS ne sont plus lisibles. Les requêtes HTTP sont encapsulées dans TLS, ce qui empêche toute interception ou lecture du contenu échangé.

Cette partie du projet permet de comprendre :

- le rôle du chiffrement TLS,
- la différence entre HTTP et HTTPS,
- le fonctionnement des certificats numériques,
- et l'importance de la sécurité des communications réseau.

The screenshot shows two windows side-by-side. On the left is a terminal window titled 'https_server.py' containing Python code to run an HTTPS server using the 'ssl' module. On the right is a browser window titled 'Windows PowerShell' showing the command to run the script and the resulting output: 'Serveur HTTPS en cours sur https://localhost:4443'. Below this, a browser alert dialog is displayed, stating 'Votre connexion n'est pas privée' (Your connection is not private) with a red exclamation mark icon. It explains that pirates could try to steal information like passwords or bank cards. Buttons for 'Revenir en sécurité' (Return to security) and 'Help me understand' are visible at the bottom of the dialog.

```
https_server.py X
C:\Users\hp\Desktop\Lab> https_server.py ...
1  from http.server import HTTPServer, SimpleHTTPRequestHandler
2  import ssl
3
4  server_address = ("0.0.0.0", 4443)
5  httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)
6
7  context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
8  context.load_cert_chain(certfile="cert.pem", keyfile="key.pem")
9
10 httpd.socket = context.wrap_socket(httpd.socket, server_side=True)
11
12 print("Serveur HTTPS en cours sur https://localhost:4443")
13 httpd.serve_forever()
```

```
Windows PowerShell
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:aymanoelhamri@gmail.com
PS C:\Users\hp\Desktop\Lab> python https_server.py
Serveur HTTPS en cours sur https://localhost:4443
```

Votre connexion n'est pas privée
Des pirates pourraient être en train d'essayer de vous voler des informations de localhost (par exemple des mots de passe, des messages ou des cartes bancaires).
NET::ERR_CERT_AUTHORITY_INVALID
Revenir en sécurité
Help me understand

4. Conclusion

Ce projet met en évidence la chaîne complète d'une communication réseau moderne :

DNS → TCP → HTTP → HTTPS

Il permet de comprendre comment les données sont acheminées, interprétées et sécurisées sur Internet, depuis la résolution d'un nom de domaine jusqu'au chiffrement des échanges entre un client et un serveur.