Projet Réseaux : DNS, TCP, HTTP, HTTPS

DNS – Résolution de noms

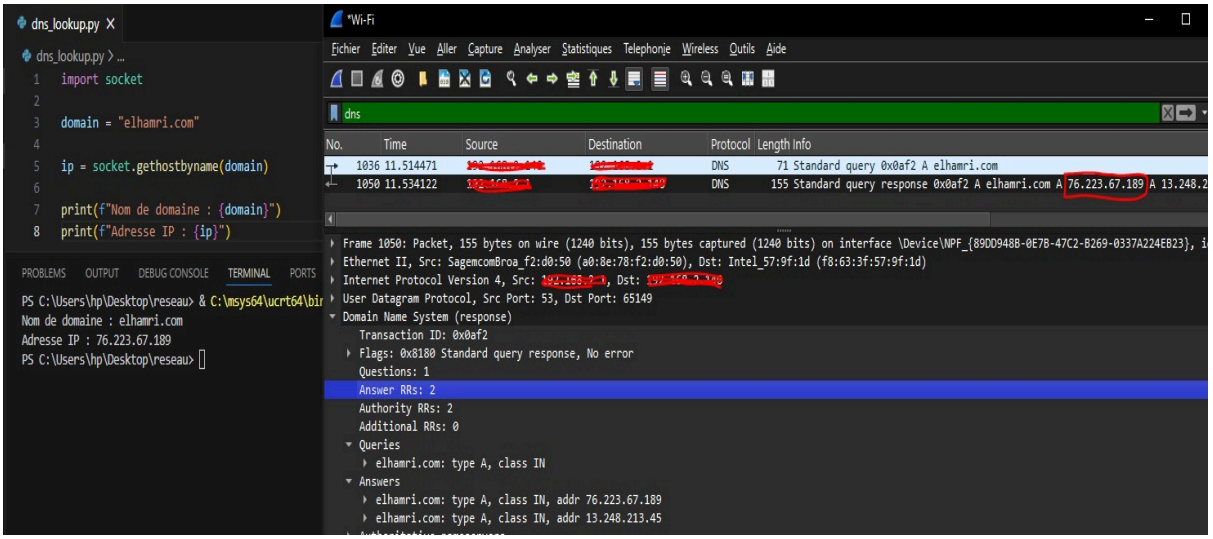
Ce script illustre le fonctionnement du **DNS (Domain Name System)**, qui est un service fondamental d'Internet. Le DNS permet de traduire un **nom de domaine** compréhensible par l'utilisateur en une **adresse IP**, nécessaire pour établir une communication réseau.

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.



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.

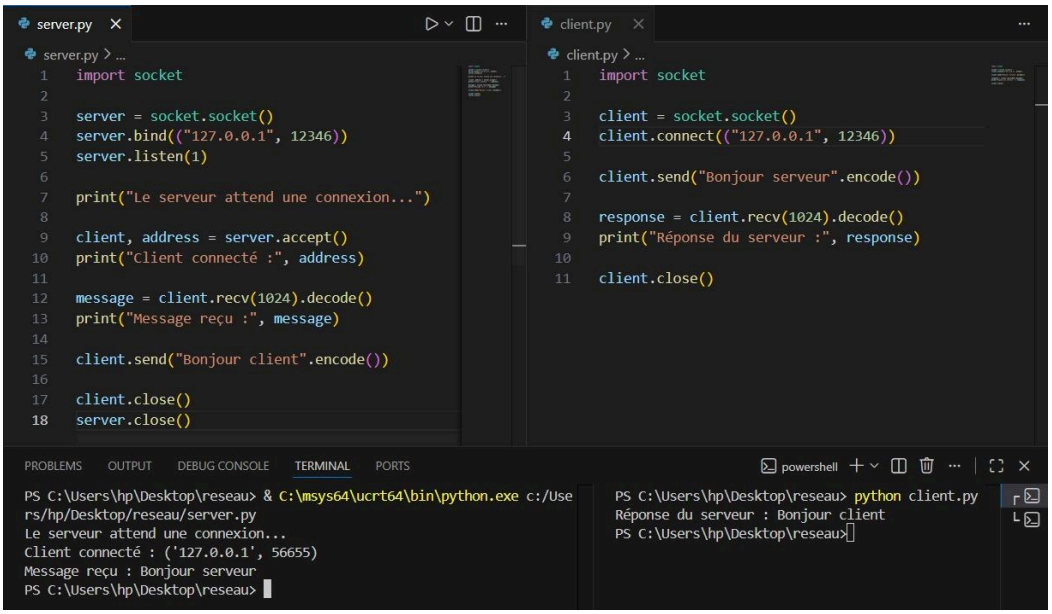
Le **serveur TCP** commence par créer un socket, puis se met en écoute sur une adresse IP et un port spécifiques. Il attend qu’un client se connecte. Lorsque la connexion est établie, le serveur accepte la requête du client, reçoit les données envoyées, les affiche, puis renvoie une réponse. Une fois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



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 (`/`).

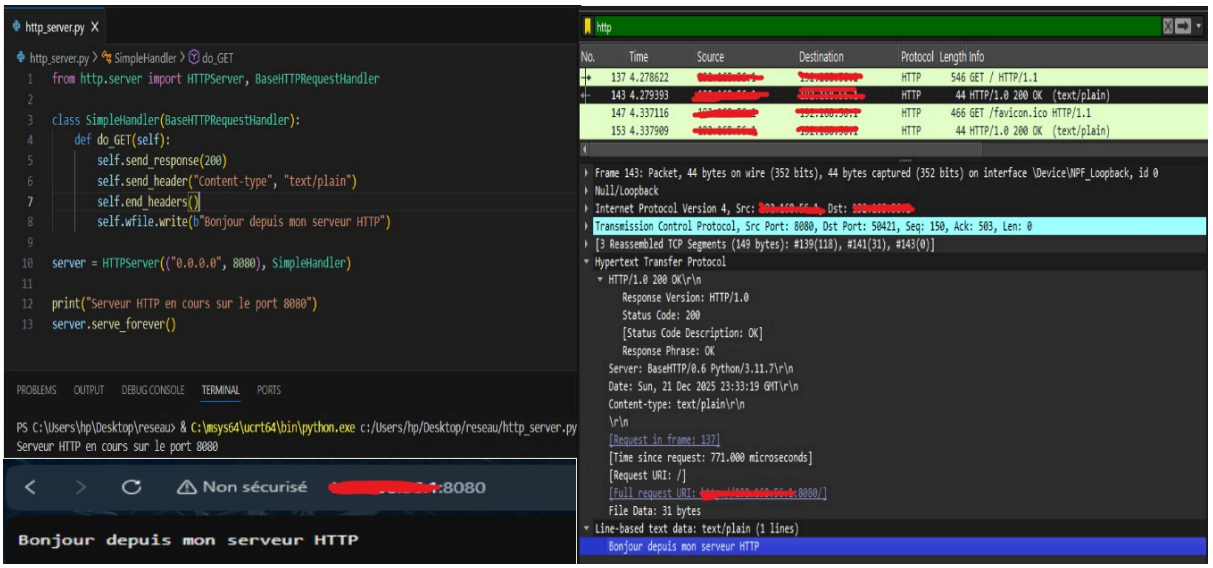
Le serveur reçoit cette requête, l'analyse, puis renvoie une **réponse HTTP** contenant :

- 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.



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".

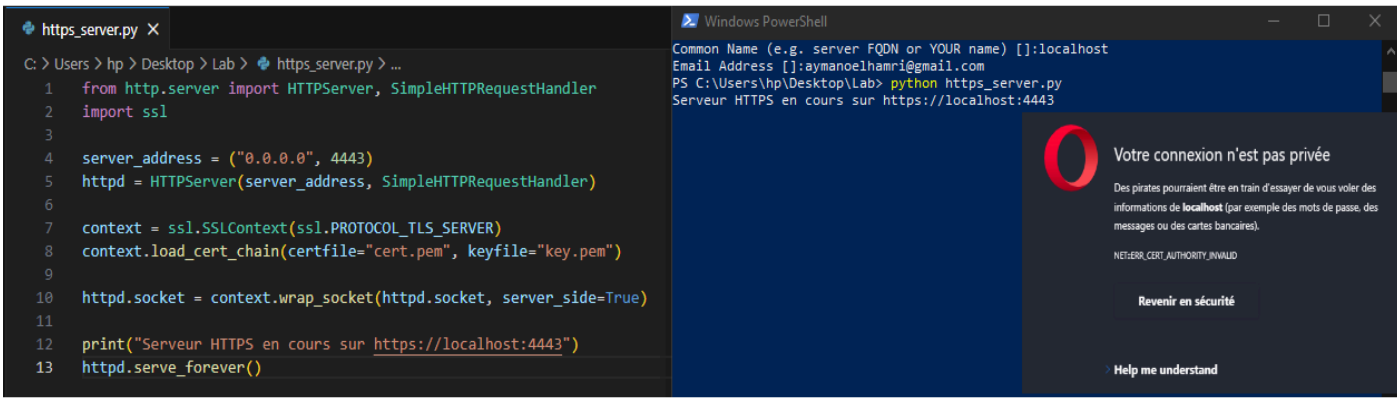
Un certificat numérique auto-signé est généré à l'aide d'OpenSSL. Ce certificat permet au serveur de chiffrer les données échangées avec le client. Lorsqu'un navigateur se connecte au serveur via HTTPS, un **handshake TLS** est effectué afin d'établir une connexion sécurisée.

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.



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.