

Les bases du réseau : NAT, Routing, UDP et TCP

Pourquoi s'y intéresser ?

En cybersécurité, chaque attaque et chaque défense repose sur le réseau. Que vous fassiez du pentesting, de l'analyse forensique (investigation faite sur le système d'information suite à une attaque informatique) ou de la défense d'infrastructure, vous devez comprendre comment les données circulent et surtout comment ce fonctionnement peut être exploité ou protégé.

Cette partie couvre quatre concepts fondamentaux : le NAT, le routing, et les deux protocoles de transport les plus utilisés, UDP et TCP.

Pour chaque concept, on verra le fonctionnement normal, les implications en sécurité, et les outils (ligne de commande et Python) qui permettent de les analyser.

Le modèle OSI en bref

Pour situer ce dont on parle, voici les couches du modèle OSI (**Open Systems Interconnection**) qui nous intéressent :

Couche	Rôle	Exemples	Surface d'attaque
7 — Application	Interface avec l'utilisateur	HTTP, FTP, DNS, SMTP	Injection, XSS, phishing
4 — Transport	Livraison fiable ou rapide des données	TCP, UDP	SYN flood, port scanning
3 — Réseau (Network)	Adressage logique et routage	IP, ICMP	Spoofing, route hijacking
2 — Liaison (Data Link)	Communication entre nœuds directement connectés	Ethernet, Wi-Fi	ARP spoofing, MAC flooding
1 — Physique	Signaux électriques, optiques, radio	Câbles, fibres	Écoute physique (wiretapping)

NAT et routing opèrent à la couche 3. TCP et UDP opèrent à la couche 4. En sécurité, les attaques les plus intéressantes exploitent souvent plusieurs couches à la fois.

Une attaque **SYN flood** (attaque semi-ouverte) est un type d'attaque par déni de service (DDoS) qui vise à rendre un serveur indisponible pour le trafic légitime en consommant toutes les ressources serveur disponibles. En envoyant à plusieurs reprises des paquets de demande de connexion initiale (SYN), le pirate est en mesure de submerger tous les ports disponibles sur une machine serveur ciblée, ce qui oblige l'appareil ciblé à répondre lentement au trafic légitime, ou l'empêche totalement de répondre.

NAT — Network Address Translation

Le problème

IPv4 utilise des adresses sur 32 bits, ce qui donne environ 4,3 milliards d'adresses possibles. En réalité, à peine 3 milliards sont utilisables (le reste est réservé pour les tests, le broadcast, des usages militaires, etc.). Avec le nombre d'appareils connectés aujourd'hui, téléphones, laptops, TV, montres, frigos, ce stock est largement insuffisant pour attribuer une adresse publique unique à chaque appareil.

La solution : le NAT. Il permet à tout un réseau privé de partager une seule adresse IP publique pour accéder à internet.

Comment ça fonctionne

Prenez un réseau domestique classique avec plusieurs appareils. Le routeur NAT fait l'intermédiaire entre le réseau privé et internet. Vu de l'extérieur, tout le trafic semble provenir d'un seul appareil : le routeur lui-même, avec son adresse IP publique.

Concrètement :

1. Votre machine **192.168.0.1** veut accéder à **example.com**. Elle envoie un paquet avec comme source **192.168.0.1:3390** et comme destination l'IP de **example.com:443**.
2. Le routeur NAT intercepte ce paquet. Il remplace l'adresse source privée par son adresse publique (disons **145.12.121.11**) et génère un nouveau port source (disons **6766**). Il note cette correspondance dans sa **table de traduction NAT** :

IP privée	Port privé	IP publique	Port public
192.168.0.1	3390	145.12.121.11	6766

3. Le paquet modifié part vers **example.com**. Le serveur reçoit un paquet qui semble venir de **145.12.121.11:6766**. Il n'a aucune idée qu'un réseau privé existe derrière.
4. Le serveur répond à **145.12.121.11:6766**. Le routeur reçoit la réponse, consulte sa table NAT, retrouve que le port **6766** correspond à **192.168.0.1:3390**, réécrit l'adresse de destination et transmet le paquet à la bonne machine.

Un port étant codé sur 16 bits, le NAT peut gérer plus de 60 000 connexions simultanées avec une seule adresse IP publique.

D'où viennent les adresses ?

Le routeur obtient son adresse publique via le serveur DHCP du FAI (fournisseur d'accès internet). En interne, le routeur fait tourner son propre serveur DHCP pour distribuer des adresses privées aux appareils du réseau.

Le NAT vu par la sécurité

Le NAT n'est **pas** un mécanisme de sécurité à proprement parler, mais il a des effets importants en défense et en attaque.

Côté défense :

- Le NAT masque la topologie interne du réseau. Un attaquant externe ne voit qu'une seule IP publique et n'a aucune visibilité sur les machines derrière.

- Les connexions entrantes non sollicitées sont bloquées par défaut : si aucune entrée n'existe dans la table NAT pour un paquet entrant, il est simplement ignoré.

Côté attaque :

- **Port forwarding mal configuré** : un administrateur qui redirige le port 22 (SSH) vers une machine interne expose cette machine à internet. Un attaquant peut la scanner et tenter un brute-force.
- **NAT slipstreaming** : une technique qui permet à un attaquant de contourner le NAT via du JavaScript malveillant dans le navigateur de la victime. Le navigateur ouvre une connexion vers un serveur contrôlé par l'attaquant, qui manipule les headers pour que le routeur NAT crée une entrée permettant l'accès direct à la machine interne.
- **Fuite d'adresse interne** : certains protocoles (SIP, FTP en mode actif) intègrent l'adresse IP dans les données applicatives, pas dans les headers. Ces adresses privées peuvent fuiter et révéler la structure du réseau.

Reconnaissance du NAT :

Depuis l'extérieur, vous pouvez détecter la présence d'un NAT en analysant les TTL des réponses. Plusieurs machines derrière un NAT auront souvent des TTL légèrement différents (car leurs OS ont des TTL par défaut différents : 64 pour Linux, 128 pour Windows).

```
# Scanner les ports ouverts derrière un NAT (port forwarding)
nmap -sV -p 1-1000 145.12.121.11
```

Limites du NAT

- **Consommation de ressources** : le routeur doit traduire chaque paquet entrant et sortant, et maintenir la table de traduction en mémoire.
- **Incompatibilité avec certains protocoles** : des applications qui intègrent l'adresse IP dans les données (pas seulement dans les headers) peuvent ne plus fonctionner.
- **Violation du modèle en couches** : le routeur est un équipement de couche 3, mais il modifie les ports, qui sont un concept de couche 4.
- **Complexité pour le tunneling** : les protocoles comme IPsec doivent être adaptés pour fonctionner à travers du NAT (NAT-T).

Routing

C'est quoi le routing ?

Le routing, c'est le mécanisme par lequel un paquet trouve son chemin d'un réseau à un autre jusqu'à atteindre sa destination. Chaque routeur sur le parcours prend une décision : "Par quelle interface je dois envoyer ce paquet pour qu'il se rapproche de sa destination ?"

La table de routage

Chaque routeur maintient une **table de routage** (routing table). C'est une sorte de carte qui associe des plages d'adresses IP de destination à des interfaces de sortie ou des routeurs voisins (next hop).

Vous pouvez voir la table de routage de votre propre machine :

```
# Sur Linux
ip route show

# Sur macOS
netstat -rn

# Sur Windows
route print
```

Exemple simplifié de table de routage :

Destination	Masque	Passerelle (Gateway)	Interface
192.168.1.0	/24	—	eth0
10.0.0.0	/8	192.168.1.1	eth0
0.0.0.0	/0	192.168.1.1	eth0

La dernière ligne (**0.0.0.0/0**) est la **route par défaut** (default route). Si aucune autre règle ne correspond à l'adresse de destination, le paquet est envoyé à cette passerelle. C'est typiquement votre routeur/box internet.

Routage statique vs dynamique

Routage statique : les routes sont configurées manuellement par un administrateur. Simple, prévisible, mais ne s'adapte pas si un lien tombe.

```
# Ajouter une route statique sur Linux
sudo ip route add 10.10.0.0/16 via 192.168.1.1
```

Routage dynamique : les routeurs échangent des informations entre eux via des protocoles dédiés (OSPF, BGP, RIP...) et mettent à jour leurs tables automatiquement. C'est ce qui fait fonctionner internet à grande échelle.

Le parcours d'un paquet

Quand vous tapez **ping 8.8.8.8** depuis votre machine :

1. Votre machine consulte sa table de routage. L'adresse **8.8.8.8** ne correspond à aucun réseau local → elle envoie le paquet à la passerelle par défaut (votre routeur).
2. Votre routeur fait la même chose : il consulte sa table, transmet au routeur suivant.
3. De routeur en routeur (on appelle chaque étape un **hop**), le paquet progresse vers sa destination.
4. Chaque routeur décrémente le champ **TTL** (Time To Live) du paquet. Si le TTL atteint 0 avant d'arriver à destination, le paquet est supprimé. Ça évite qu'un paquet tourne en boucle indéfiniment dans le réseau.

Vous pouvez visualiser ce parcours avec **tracert** (ou **tracert** sur Windows) :

```
tracert 8.8.8.8
```

Chaque ligne de la sortie représente un hop — un routeur traversé par votre paquet.

Attaques sur le routing

Le routing est une cible privilégiée en cybersécurité. Si vous contrôlez le chemin que prennent les paquets, vous contrôlez tout.

BGP hijacking : BGP (Border Gateway Protocol) est le protocole qui fait fonctionner le routing entre les réseaux autonomes d'internet. Un attaquant qui contrôle un routeur BGP peut annoncer de fausses routes et rediriger le trafic destiné à un réseau entier vers ses propres machines. C'est arrivé plusieurs fois en production — en 2018, du trafic destiné à Amazon Route 53 a été détourné pour voler de la cryptomonnaie.

Route poisoning : dans les protocoles de routage dynamique (RIP, OSPF), un attaquant qui a accès au réseau local peut injecter de fausses informations de routage pour rediriger le trafic à travers sa machine.

ARP spoofing (couche 2, mais lié au routing local) : sur un réseau local, la résolution IP → adresse MAC passe par **ARP** (Address Resolution Protocol). Quand une machine veut communiquer avec une autre sur le même réseau, elle envoie une requête ARP en broadcast : "Qui a l'IP 192.168.1.1 ? Donne-moi ton adresse MAC." La machine concernée répond avec sa MAC, et la correspondance est mise en cache. Un attaquant peut envoyer de fausses réponses ARP pour se faire passer pour la passerelle. Tout le trafic du réseau local passe alors par sa machine — c'est un man-in-the-middle classique.

```
# Détecter l'ARP spoofing : vérifier si deux IP ont la même MAC
arp -a
```

Si deux IP différentes partageaient la même MAC, ce serait un signe d'ARP spoofing.

Le routing comme outil de reconnaissance

En pentest, le tracert est un outil de reconnaissance basique mais précieux. Il révèle :

- L'infrastructure réseau entre vous et la cible
- Les firewalls (hops qui ne répondent pas : * * *)
- La segmentation réseau (changements de plage d'adresses entre les hops)

```
# Traceroute classique (ICMP)
tracert 10.0.0.1

# Traceroute TCP – passe souvent mieux les firewalls
tracert -T -p 443 10.0.0.1
```

UDP — User Datagram Protocol

Le protocole "fire and forget"

UDP est le protocole de transport le plus simple. Pas de connexion, pas d'accusé de réception, pas de garantie de livraison. Vous envoyez un **datagramme** et vous espérez qu'il arrive. C'est tout.

Un datagramme, c'est un paquet de données autonome : il contient à la fois les données et toutes les infos nécessaires pour arriver à destination (adresses, ports) dans son header. Chaque datagramme voyage indépendamment des autres, sans lien entre eux. Pensez-y comme la différence entre des cartes postales (datagrammes : chacune voyage seule, peut se perdre ou arriver dans le désordre).

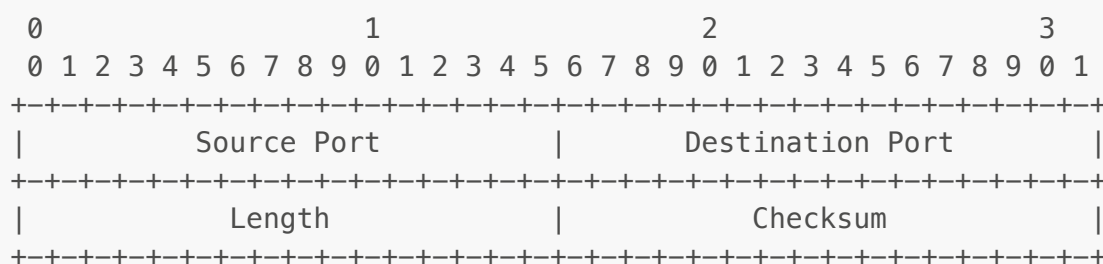
Pourquoi utiliser un protocole aussi "primitif" ? Parce que cette simplicité a un avantage énorme : la **vitesse**. Pas de handshake pour établir une connexion, pas d'attente d'accusés de réception, pas de retransmissions. Le overhead est minimal.

Caractéristiques principales

- **Connectionless** : pas de connexion à établir. Chaque datagramme est indépendant.
- **Pas de garantie de livraison** : si un paquet se perd en route, personne ne le renvoie. C'est à l'application de gérer si nécessaire.
- **Pas de garantie d'ordre** : les paquets peuvent arriver dans le désordre.
- **Rapide et léger** : très peu de overhead protocolaire.

Le header UDP

Le header UDP est minimaliste — seulement 8 octets :



Champ	Taille	Rôle
Source Port	16 bits	Port de l'application émettrice
Destination Port	16 bits	Port de l'application destinataire
Length	16 bits	Taille totale du datagramme (header + données)
Checksum	16 bits	Détection d'erreurs de transmission

Le **checksum** est une valeur calculée à partir du contenu du paquet (header + données) avant l'envoi. À la réception, le même calcul est refait : si le résultat ne correspond pas au checksum reçu, c'est que des bits ont été altérés pendant le transport (interférence, erreur matérielle...). Le

paquet est alors ignoré. Le checksum détecte les erreurs, mais ne les corrige pas et ne demande pas de retransmission — c'est juste un contrôle d'intégrité minimal.

Comparez ça au header TCP (20 octets minimum) — vous comprenez pourquoi UDP est plus rapide.

Comment fonctionne une communication UDP

1. **L'émetteur prépare le message.** Il connaît l'adresse IP et le port du destinataire.
2. **Il encapsule le message** dans un datagramme UDP avec le header approprié et calcule le checksum.
3. **Il envoie le datagramme** directement, sans établir de connexion au préalable.
4. **Le récepteur écoute** sur le port concerné. Quand le datagramme arrive, il en extrait les données.
5. **Traitement** : le récepteur traite le message. S'il en manque un, il ne le saura pas (sauf si l'application a prévu un mécanisme pour ça).

Cas d'usage typiques

UDP est utilisé quand la vitesse prime sur la fiabilité :

- **DNS** : une requête, une réponse. Rapide, pas besoin de connexion persistante.
- **VoIP et visioconférence** : mieux vaut perdre quelques paquets que d'avoir du lag à cause des retransmissions.
- **Streaming vidéo/audio** : un pixel manquant vaut mieux qu'une image figée.
- **Jeux en ligne** : la position d'un joueur il y a 200ms ne sert plus à rien, pas la peine de la retransmettre.

Attaques exploitant UDP

L'absence de connexion et de vérification fait d'UDP un vecteur d'attaque privilégié.

UDP flood (DDoS) : l'attaquant envoie un volume massif de datagrammes UDP vers des ports aléatoires de la cible. Pour chaque paquet reçu sur un port fermé, la cible génère un message ICMP "port unreachable" — ce qui consomme ses ressources. Avec assez de volume, la cible ne peut plus traiter le trafic légitime.

DNS amplification : l'attaquant envoie des requêtes DNS avec une adresse source usurpée (celle de la victime) à des serveurs DNS ouverts. Les réponses DNS sont beaucoup plus volumineuses que les requêtes (facteur d'amplification jusqu'à x50). La victime se retrouve submergée par des réponses qu'elle n'a jamais demandées. Ça fonctionne parce qu'UDP n'a pas de handshake — impossible de vérifier que l'adresse source est légitime.

DNS spoofing : un attaquant envoie une fausse réponse DNS à la victime avant que le vrai serveur DNS ne réponde. Comme UDP ne vérifie pas l'identité de l'émetteur, la victime accepte la première réponse reçue. Résultat : elle est redirigée vers un site contrôlé par l'attaquant.

UDP port scanning

Scanner les ports UDP est plus lent et moins fiable que le scan TCP, parce qu'un port ouvert ne répond pas nécessairement — il n'y a pas de handshake.

```
# Scan UDP avec nmap (nécessite les droits root)
sudo nmap -sU -p 53,67,68,69,123,161,500 10.0.0.1

# -sU : scan UDP
# Les ports courants : DNS(53), DHCP(67-68), TFTP(69), NTP(123), SNMP(161),
IKE(500)
```

La logique du scan UDP :

- Port ouvert → pas de réponse (ou réponse applicative)
- Port fermé → message ICMP "port unreachable"
- Filtré → aucune réponse (le firewall drop le paquet)

C'est pour ça que le scan UDP est lent : il faut attendre un timeout pour chaque port sans réponse.

TCP — Transmission Control Protocol

Le protocole fiable

TCP est le protocole de transport fiable. Contrairement à UDP, il garantit que les données arrivent, dans le bon ordre, sans corruption. C'est le protocole utilisé par HTTP, HTTPS, SSH, FTP, SMTP... bref, tout ce qui nécessite que chaque octet arrive intact.

Le prix de cette fiabilité : plus de complexité, plus de overhead, plus de latence. Mais cette complexité crée aussi une surface d'attaque plus large.

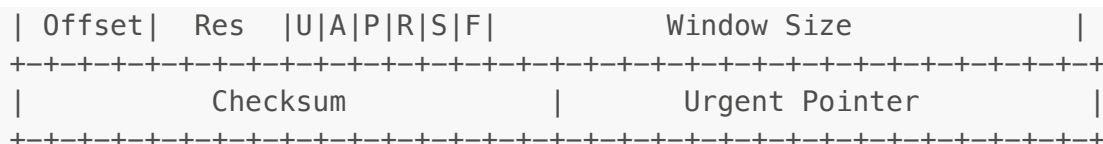
Caractéristiques principales

- **Connection-oriented** : avant d'échanger des données, les deux parties doivent établir une connexion via un handshake.
- **Livraison fiable** : chaque segment envoyé doit être acquitté. S'il ne l'est pas, il est retransmis.
- **Livraison ordonnée** : les segments sont numérotés et réassemblés dans l'ordre, même s'ils arrivent dans le désordre.
- **Contrôle de flux** : le récepteur peut dire à l'émetteur de ralentir s'il ne suit pas.
- **Contrôle de congestion** : TCP adapte son débit en fonction de l'état du réseau.

Le header TCP

Le header TCP est bien plus riche que celui d'UDP — au minimum 20 octets :

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Source Port										Destination Port																													
Sequence Number																																							
Acknowledgment Number																																							



Les champs importants :

Champ	Taille	Rôle
Source/Dest Port	16 bits chacun	Identifient les applications source et destination
Sequence Number	32 bits	Numérote chaque octet envoyé pour le réassemblage
Acknowledgment Number	32 bits	Indique le prochain octet attendu par le récepteur
Flags (URG, ACK, PSH, RST, SYN, FIN)	6 bits	Contrôlent l'état de la connexion
Window Size	16 bits	Contrôle de flux — combien d'octets le récepteur peut encore accepter
Checksum	16 bits	Détection d'erreurs

Le three-way handshake

Avant tout échange de données, TCP établit une connexion en trois étapes :



Ce mécanisme garantit que les deux parties sont prêtes à communiquer et se sont mises d'accord sur les numéros de séquence initiaux.

Livraison fiable : ACK et retransmission

Chaque segment envoyé porte un numéro de séquence. Le récepteur confirme la réception avec un ACK qui contient le numéro du prochain octet attendu.

Si l'émetteur ne reçoit pas d'ACK dans un délai donné (timeout), il retransmet le segment. Aucune donnée ne se perd silencieusement.

Gestion des paquets hors-ordre

Les paquets peuvent arriver dans le désordre sur le réseau. TCP gère ça grâce aux sequence numbers :

1. Le récepteur met en buffer les segments reçus hors-ordre.
2. Il continue d'envoyer des ACK pour le dernier octet reçu dans l'ordre.
3. Quand les segments manquants arrivent, il réassemble le tout et le passe à l'application.
4. Avec le mécanisme **SACK** (Selective Acknowledgment), le récepteur peut indiquer précisément quels blocs il a reçu, ce qui permet à l'émetteur de ne retransmettre que ce qui manque vraiment.

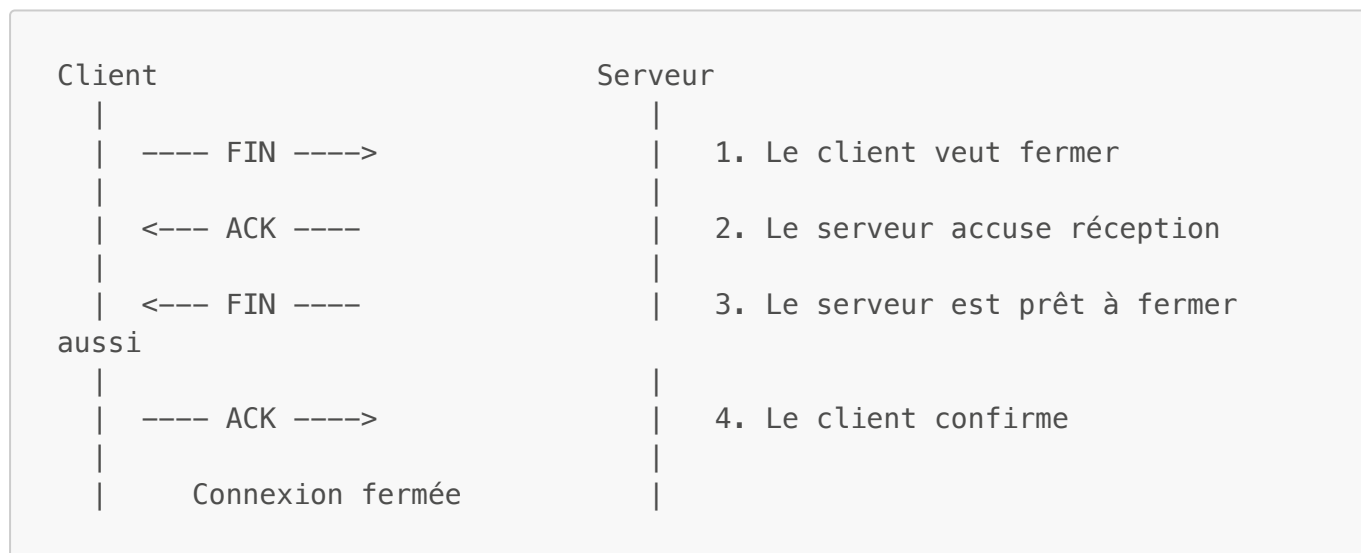
Contrôle de congestion

TCP adapte son débit pour ne pas saturer le réseau. Les mécanismes principaux :

- **Slow Start** : au début de la connexion, TCP envoie peu de données et augmente progressivement (exponentiellement) jusqu'à détecter de la congestion.
- **Congestion Avoidance** : une fois un seuil atteint, l'augmentation devient linéaire.
- **Fast Retransmit** : si l'émetteur reçoit 3 ACK dupliqués (signe qu'un segment s'est perdu), il retransmet immédiatement sans attendre le timeout.
- **Fast Recovery** : après un fast retransmit, TCP ne repart pas de zéro mais ajuste son débit à un niveau intermédiaire.

Fermeture de connexion : le four-way handshake

La fermeture est aussi explicite que l'ouverture :



Attaques exploitant TCP

Chaque étape du cycle de vie TCP est une surface d'attaque potentielle.

SYN flood : l'attaque DDoS la plus classique sur TCP. L'attaquant envoie un flot de paquets SYN avec des adresses source usurpées. Le serveur répond avec des SYN-ACK et attend le ACK final qui ne viendra jamais. Chaque connexion "half-open" consomme de la mémoire. Quand la file est pleine, le serveur ne peut plus accepter de connexions légitimes.

Défense : les **SYN cookies** permettent au serveur de ne pas stocker d'état tant que le three-way handshake n'est pas terminé. Le numéro de séquence du SYN-ACK encode les paramètres de la connexion — si le client envoie un ACK valide, le serveur peut reconstituer l'état.

TCP reset attack : un attaquant qui peut sniffer le trafic (ou deviner les sequence numbers) envoie un paquet RST forgé à l'une des parties. La connexion est coupée instantanément. C'est utilisé pour censurer des connexions (le Grand Firewall de Chine utilise cette technique) ou pour couper des sessions SSH/VPN.

Session hijacking : si un attaquant peut prédire ou observer les sequence numbers d'une connexion TCP établie, il peut injecter des données dans la session en se faisant passer pour l'une des parties. C'est pour ça que les sequence numbers initiaux sont aujourd'hui randomisés (ISN randomization) — dans les anciennes implémentations TCP, ils étaient prévisibles.

TCP port scanning

Le scan de ports TCP est la base de toute reconnaissance en pentest. Plusieurs techniques existent :

TCP connect scan : le scanner tente un three-way handshake complet sur chaque port. Simple, fiable, mais visible dans les logs du serveur.

```
# Scan connect avec nmap
nmap -sT -p 1-1000 10.0.0.1
```

SYN scan (half-open scan) : le scanner envoie un SYN. Si le port est ouvert, le serveur répond SYN-ACK. Le scanner envoie immédiatement un RST au lieu de compléter le handshake. Plus discret qu'un connect scan car la connexion n'est jamais complètement établie.

```
# SYN scan (nécessite root)
sudo nmap -sS -p 1-1000 10.0.0.1
```

Xmas scan, FIN scan, NULL scan : ces techniques envoient des paquets avec des combinaisons de flags inhabituelles. Un port fermé répond avec un RST. Un port ouvert ne répond pas. Utile pour contourner certains firewalls qui ne filtrent que les paquets SYN.

```
# Xmas scan : flags FIN, PSH, URG activés
sudo nmap -sX -p 1-1000 10.0.0.1

# NULL scan : aucun flag
sudo nmap -sN -p 1-1000 10.0.0.1
```

Banner grabbing

Une fois un port ouvert identifié, on veut savoir quel service tourne dessus et quelle version. Beaucoup de services envoient une bannière d'identification dès la connexion.

```
# Avec netcat
nc -v 10.0.0.1 22
# SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1

# Avec nmap (détection de version)
nmap -sV -p 22,80,443 10.0.0.1
```

TCP vs UDP : le résumé

	TCP	UDP
Connexion	Oui (handshake)	Non
Fiabilité	Garantie (ACK + retransmission)	Aucune
Ordre	Garanti (sequence numbers)	Non garanti
Vitesse	Plus lent (overhead du protocole)	Plus rapide
Header	20 octets minimum	8 octets
Contrôle de flux	Oui	Non
Contrôle de congestion	Oui	Non
Spoofing facile ?	Difficile (handshake oblige)	Très facile (connectionless)
Attaque DDoS typique	SYN flood	UDP flood, amplification
Scan de ports	Rapide et fiable	Lent et peu fiable
Cas d'usage	Web, email, SSH, transfert de fichiers	DNS, streaming, VoIP, jeux en ligne

Analyser le trafic réseau

Savoir capturer et lire du trafic réseau est une compétence essentielle en cybersécurité — que ce soit pour du forensique, de la détection d'intrusion, ou du debugging.

tcpdump

tcpdump est l'outil en ligne de commande de référence pour capturer du trafic :

```
# Capturer tout le trafic sur l'interface eth0
sudo tcpdump -i eth0

# Filtrer uniquement le trafic TCP sur le port 80
sudo tcpdump -i eth0 tcp port 80

# Capturer uniquement les paquets SYN (tentatives de connexion)
sudo tcpdump -i eth0 'tcp[tcpflags] & tcp-syn != 0'

# Sauvegarder la capture dans un fichier .pcap pour analyse ultérieure
```

```
sudo tcpdump -i eth0 -w capture.pcap
```

```
# Lire une capture existante
```

```
sudo tcpdump -r capture.pcap
```

Wireshark

Wireshark fait la même chose avec une interface graphique et des outils d'analyse avancés. Quelques filtres utiles :

```
# Filtrer par protocole
```

```
tcp
```

```
udp
```

```
dns
```

```
http
```

```
# Filtrer par IP
```

```
ip.addr == 10.0.0.1
```

```
# Voir uniquement les SYN (début de connexion)
```

```
tcp.flags.syn == 1 && tcp.flags.ack == 0
```

```
# Voir les retransmissions (signe de problème réseau ou d'attaque)
```

```
tcp.analysis.retransmission
```

```
# Voir les RST (connexions coupées brutalement)
```

```
tcp.flags.reset == 1
```