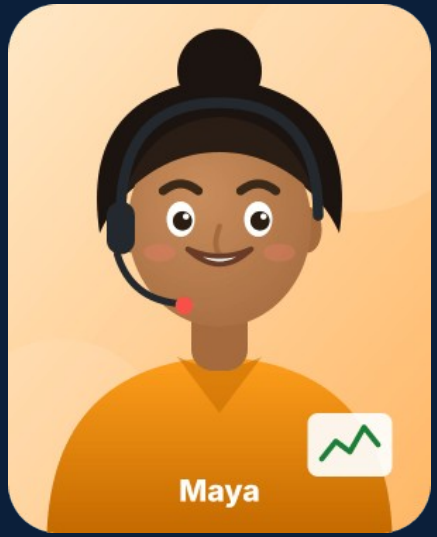
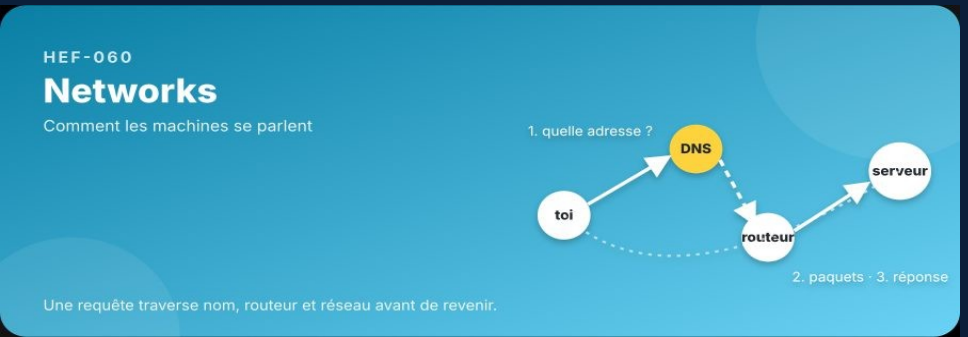
HEF-060 · MISSION 10 / 10

Analyse de trafic et dépannage

« Internet ne marche pas » n'est pas un diagnostic. Voyons comment le transformer en méthode.





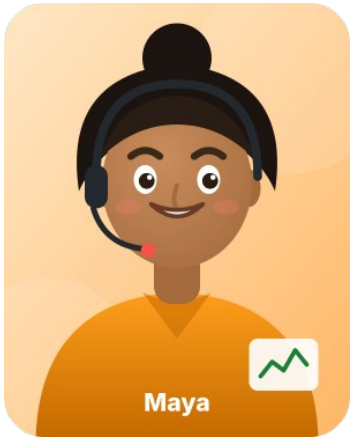
Badge : Détective du réseau

90 min · intermédiaire

L'histoire

Lundi matin, une dizaine de personnes à HashBox disent la même phrase : « Internet ne marche pas ! » Kodi ouvre un terminal. Les premières minutes, il essaie des choses au hasard : redémarrer le routeur, changer de câble, relancer le navigateur.

« Reprends à la base, dit Sam. Une couche après l'autre. À chaque étape, un test, une conclusion. »



Dans cette dernière mission, tu assembles tout le module en une méthode de dépannage.

Ta mission

À la fin, je serai capable de…

1. appliquer une démarche de dépannage couche par couche
2. utiliser ping, traceroute, nslookup/dig, curl et ss pour isoler un problème
3. lire une capture de paquets de façon élémentaire
4. écrire un outil de diagnostic réseau en Python
5. rédiger un rapport d'incident réseau

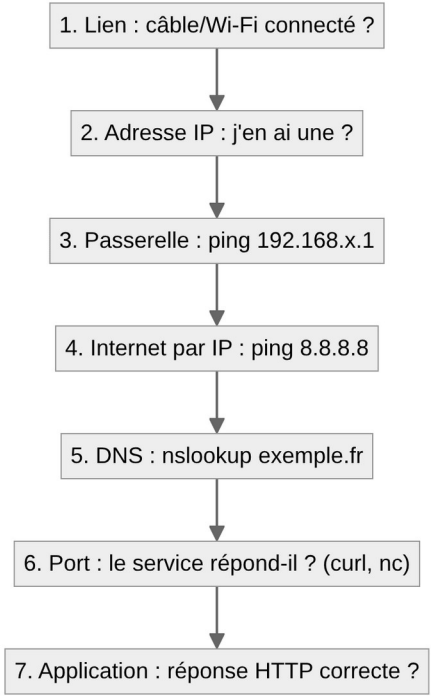
Défi d'échauffement

Défi d'échauffement

« Je n'arrive pas à ouvrir le site de ma banque. » Liste dans l'ordre **cinq vérifications** que tu ferais, du plus simple au plus complexe (câble ou Wi-Fi, adresse, passerelle, nom, site). Pour chacune, qu'apprend-on si elle réussit ? Si elle échoue ?

Le concept

La méthode : de bas en haut (suite)



Le concept

La méthode : de bas en haut (suite)

Description du schéma : sept vérifications successives, du lien physique à l'application. On s'arrête à la première qui échoue : c'est là que se trouve le problème.

Ce que chaque test nous apprend

Test	Succès signifie	Échec signifie
Lien actif	le support physique est bon	câble, Wi-Fi, carte
Adresse IP obtenue	DHCP a répondu	DHCP en panne, `169.254.x.x` (auto-adresse)
`ping` passerelle	le réseau local marche	problème local (switch, Wi-Fi)
`ping 8.8.8.8`	la route vers internet marche	routage ou opérateur
`nslookup exemple.fr`	DNS fonctionne	résolveur injoignable ou mauvais
`curl -v https://exemple.fr`	transport + TLS + HTTP OK	pare-feu, port, certificat, service

Le concept

Les outils

Outil	Rôle
`ping`	joignabilité et latence (ICMP)
`traceroute` / `tracepath`	chemin et sauts
`nslookup` / `dig`	requêtes DNS
`curl -v`	requête HTTP détaillée (TLS, en-têtes, codes)
`ss -tulpn`	ports en écoute et connexions
`ip addr`, `ip route`	adresses et routes
`tcpdump` / Wireshark	capture et analyse de paquets

Le concept

Lire une capture

Une capture montre chaque paquet : horodatage, source, destination, protocole, informations. Exemple d'une connexion TCP qui échoue :

```
10:00:00.001 192.168.1.10:51234 > 203.0.113.50:443  SYN
10:00:01.002 192.168.1.10:51234 > 203.0.113.50:443  SYN   (retransmission)
10:00:03.004 192.168.1.10:51234 > 203.0.113.50:443  SYN   (retransmission)
```

Des SYN répétés **sans SYN-ACK** : le serveur ne répond pas (pare-feu qui jette, serveur éteint, mauvais routage). À l'inverse, un RST immédiat signifie « personne n'écoute sur ce port ».

Symptôme dans la capture	Cause probable
SYN sans réponse	paquets filtrés ou machine injoignable
SYN puis RST	port fermé, rien n'écoute

Le concept

Lire une capture (suite)

Symptôme dans la capture	Cause probable
Requête DNS sans réponse	résolveur injoignable
Beaucoup de retransmissions	perte de paquets, lien saturé
Alerte TLS	certificat ou version non supportée

Un diagnostic en Python

On peut automatiser les tests élémentaires avec les modules standard :

```
import socket

def resoudre(nom):
    try:
        return socket.gethostbyname(nom), None
    except socket.gaierror as e:
```

Le concept

Un diagnostic en Python (suite)

```
        return None, f"échec DNS : {e}"

def port_ouvert(ip, port, delai=1.0):
    try:
        with socket.create_connection((ip, port), timeout=delai):
            return True, "connexion acceptée"
    except ConnectionRefusedError:
        return False, "refusée (rien n'écoute, RST)"
    except socket.timeout:
        return False, "délai dépassé (filtré ou injoignable)"
    except OSError as e:
        return False, f"erreur : {e}"

print(resoudre("localhost"))
print(port_ouvert("127.0.0.1", 9))      # le port 9 est en général fermé
```

Le concept

Un diagnostic en Python (suite)

🧠 ***Astuce*** *: « refusée » et « délai dépassé » n'ont pas du tout la même cause. La première vient d'une machine qui répond « non », la seconde du silence.*

Exemple pas à pas

Problème : écrire un outil de diagnostic qui teste un service local et explique ce qu'il trouve.

```
import json, socket, threading, urllib.request
from http.server import BaseHTTPRequestHandler, HTTPServer

class H(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200); self.send_header("Content-Length", "2"); self.end
   _headers(); self.wfile.write(b"ok")
    def log_message(self, *a): pass

serveur = HTTPServer(("127.0.0.1", 0), H)
port_ok = serveur.server_address[1]
threading.Thread(target=serveur.serve_forever, daemon=True).start()

def diagnostiquer(hote, port):
    etapes = []
    try:
```

Exemple pas à pas (suite)

```
        ip = socket.gethostbyname(hote)
        etapes.append(("DNS", "OK", ip))
    except socket.gaierror as e:
        etapes.append(("DNS", "ÉCHEC", str(e)))
        return etapes
    try:
        with socket.create_connection((ip, port), timeout=1):
            etapes.append(("TCP", "OK", f"port {port} ouvert"))
    except ConnectionRefusedError:
        etapes.append(("TCP", "ÉCHEC", "connexion refusée : rien n'écoute"))
        return etapes
    except socket.timeout:
        etapes.append(("TCP", "ÉCHEC", "délai dépassé : paquets filtrés ?"))
        return etapes
    try:
        with urllib.request.urlopen(f"http://{ip}:{port}/", timeout=1) as r:
            etapes.append(("HTTP", "OK", f"code {r.status}"))
    except Exception as e:
        etapes.append(("HTTP", "ÉCHEC", str(e)))
```

Exemple pas à pas (suite)

```
    return etapes

def afficher(titre, etapes):
    print(titre)
    for couche, etat, detail in etapes:
        print(f"  {couche:5s} {etat:6s} {detail}")

afficher("service sain", diagnostiquer("localhost", port_ok))
afficher("port fermé", diagnostiquer("localhost", 1))
afficher("nom inconnu", diagnostiquer("ce-nom-n-existe-pas.invalid", 80))
serveur.shutdown()
```

Lecture : l'outil s'arrête à la **première étape en échec** et explique la signification, exactement comme le ferait un technicien. Le troisième cas échoue dès le DNS, il ne sert donc à rien de tester le port.

À toi, avec aide

Voici trois situations. Pour chacune, dis à quelle étape de la méthode le problème se situe, et propose la prochaine commande.

1. `ping 192.168.1.1` → « Destination Host Unreachable ». L'ordinateur a l'adresse 169.254.7.22.
2. `ping 8.8.8.8` répond ; `ping exemple.fr` → « Name or service not known ».
3. `curl -v https://hashbox.example` → « Connection timed out after 10001 ms » ; `ping` du serveur répond.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Une adresse 169.254.x.x signifie que le DHCP n'a pas répondu.

Indice 2

Si l'IP marche mais pas les noms, c'est le DNS.

Indice 3

Le ping répond mais le port 443 ne répond pas : quelque chose filtre ou le service est arrêté.

À toi, avec aide · réponse

Réponse

1. **Étape 2 (adresse IP)** : pas de réponse du DHCP (auto-adresse). Prochaine commande : vérifier le lien et le service DHCP (`ip addr`, journaux DHCP, essayer un autre port/Wi-Fi).
2. **Étape 5 (DNS)** : la connectivité IP est bonne. Prochaine commande : `nslookup exemple.fr 8.8.8.8` pour tester un autre résolveur, puis vérifier `/etc/resolv.conf`.
3. **Étape 6 (port 443)** : le serveur est joignable mais le port ne répond pas : pare-feu qui jette les paquets, ou service arrêté (un service arrêté renverrait plutôt un `RST`/refus rapide). Prochaine commande : `nc -zv hashbox.example 443` depuis plusieurs endroits, puis vérifier les règles du pare-feu et `ss -tlnp` côté serveur.

À toi, en solo

Analyseur de capture. Voici une capture simplifiée sous forme de lignes `temps source destination protocole détail`. Écris `analyser(lignes)` qui détecte automatiquement :

1. les connexions TCP qui n'aboutissent jamais (SYN répétés sans SYN-ACK) ;
2. les connexions refusées (SYN suivi de RST) ;
3. les requêtes DNS sans réponse ;
4. le temps de réponse HTTP (requête → réponse) pour chaque connexion réussie.

Teste avec une capture de ton invention contenant au moins un cas de chaque.

À toi, en solo · réponse

Pistes de correction

```
from collections import defaultdict

CAPTURE = [
    "0.000 10.0.0.2:5000 203.0.113.50:443 TCP SYN",
    "1.000 10.0.0.2:5000 203.0.113.50:443 TCP SYN",
    "3.000 10.0.0.2:5000 203.0.113.50:443 TCP SYN",
    "0.100 10.0.0.2:5001 203.0.113.60:80 TCP SYN",
    "0.101 203.0.113.60:80 10.0.0.2:5001 TCP RST",
    "0.200 10.0.0.2:5002 203.0.113.70:80 TCP SYN",
    "0.230 203.0.113.70:80 10.0.0.2:5002 TCP SYN-ACK",
    "0.231 10.0.0.2:5002 203.0.113.70:80 TCP ACK",
    "0.232 10.0.0.2:5002 203.0.113.70:80 HTTP GET /",
    "0.400 203.0.113.70:80 10.0.0.2:5002 HTTP 200 OK",
    "0.500 10.0.0.2:5003 10.0.0.53:53 DNS requete exemple.fr",
]

def analyser(lignes):
    syn, synack, rst, dns_req, dns_rep = defaultdict(list), set(), set(), {}, set(
   )
```

Pistes de correction (suite)

```
    http_req, http_rep = {}, {}
    for ligne in lignes:
        t, src, dst, proto, *detail = ligne.split()
        t, detail = float(t), " ".join(detail)
        if proto == "TCP" and detail == "SYN":
            syn[(src, dst)].append(t)
        elif proto == "TCP" and detail == "SYN-ACK":
            synack.add((dst, src))
        elif proto == "TCP" and detail == "RST":
            rst.add((dst, src))
        elif proto == "DNS" and detail.startswith("requete"):
            dns_req[(src, dst)] = t
        elif proto == "DNS" and detail.startswith("reponse"):
            dns_rep.add((dst, src))
        elif proto == "HTTP" and detail.startswith("GET"):
            http_req[(src, dst)] = t
        elif proto == "HTTP":
            http_rep[(dst, src)] = t
    rapport = []
```

Pistes de correction (suite)

```
    for cle, temps in syn.items():
        if cle in rst:
            rapport.append(f"{cle[0]} → {cle[1]} : connexion refusée (RST)")
        elif cle not in synack:
            rapport.append(f"{cle[0]} → {cle[1]} : {len(temps)} SYN sans réponse (
   filtré ou injoignable)")
    for cle in dns_req:
        if cle not in dns_rep:
            rapport.append(f"{cle[0]} → {cle[1]} : requête DNS sans réponse")
    for cle, t in http_req.items():
        if cle in http_rep:
            rapport.append(f"{cle[0]} → {cle[1]} : réponse HTTP en {1000 * (http_r
   ep[cle] - t):.0f} ms")
    return rapport

for r in analyser(CAPTURE):
    print("-", r)
```

Pistes de correction (suite)

Le raisonnement est celui d'un outil comme Wireshark : on regroupe les paquets par connexion, puis on cherche les motifs qui trahissent une panne.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Tester au hasard	On passe à côté de la cause	Une couche après l'autre, de bas en haut
Confondre refus et silence	Causes opposées	Refus (RST) = rien n'écoute ; silence = filtré ou injoignable
Accuser le réseau trop vite	Souvent DNS, certificat ou application	Tester chaque couche
Ne pas noter	On refait les mêmes tests	Journal de diagnostic

Mini-boss

Dossier de dépannage final du module HEF-060. Scénario : « Depuis ce matin, les postes du bureau B n'accèdent plus à l'application interne `app.hashbox.example` (port 8443). Le bureau A fonctionne. »

1. Écris un programme qui simule trois causes possibles : (a) règle de pare-feu entre le bureau B et le serveur, (b) enregistrement DNS erroné pour les clients du bureau B (résolveur interne différent), (c) route manquante vers le sous-réseau du serveur. Chaque cause produit des « sorties de commandes » différentes (ping, nslookup, connexion TCP).
2. Écris `diagnostiquer(sorties)` qui identifie la cause à partir de ces sorties en appliquant la méthode couche par couche.
3. Rédige le rapport : chronologie, constats, cause racine, correction, vérification, prévention.
4. Relie chaque constat à une mission du module (DNS, routage, pare-feu, TCP…).

Mini-boss · réponse

Pistes de correction

Un bon dossier : pour (a), `ping` du serveur répond, DNS résout bien, la connexion TCP au port 8443 expire (silence) → pare-feu ; pour (b), `nslookup` donne une mauvaise adresse ou NXDOMAIN depuis B mais la bonne depuis A → DNS ; pour (c), `ping` du serveur échoue (« network unreachable »), `ip route` n'a pas de route vers le sous-réseau → routage. `diagnostiquer` s'arrête à la première étape en échec et nomme la couche. Le rapport inclut la comparaison A/B (qui est la meilleure démarche pour isoler), la correction minimale, la vérification (les trois tests au vert depuis B) et la prévention (supervision synthétique du service depuis chaque site, versionner les règles de pare-feu et les zones DNS).

Quiz éclair

Quiz éclair

1. Une connexion TCP « refusée » (RST) signifie :

A. rien n'écoute sur ce port
B. paquets filtrés en silence
C. DNS en panne

2. Une adresse 169.254.x.x indique :

A. un réseau d'entreprise
B. que le DHCP n'a pas répondu
C. une adresse publique

3. La démarche de dépannage recommandée va :

A. au hasard

Quiz éclair

Quiz éclair (suite)

B. de l'application vers le câble sans test
C. de la couche basse vers la couche haute

4. 🔁 *Révision mission 09* : Le NAT permet de :

A. faire partager une adresse publique à plusieurs machines privées
B. chiffrer le trafic
C. trouver l'adresse MAC

5. 🔁 *Révision mission 09* : La politique « default deny » consiste à :

A. tout accepter sauf le danger
B. tout refuser sauf ce qui est explicitement autorisé
C. désactiver le pare-feu

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

Je réfléchis

1. Quel réflexe de dépannage veux-tu garder (ne pas redémarrer au hasard, comparer deux situations) ?
2. Que changerais-tu à ta façon de décrire un problème réseau à quelqu'un d'autre ?

À retenir

À retenir

1. Dépanner = tester de bas en haut, une couche après l'autre.
2. Refus rapide ≠ silence : causes différentes.
3. ping, traceroute, nslookup, curl, ss, capture de paquets.
4. Comparer une situation qui marche et une qui échoue.
5. Un rapport clair permet de prévenir la récidive.

Félicitations

Félicitations

Tu as terminé **HEF-060 — Réseaux**. Tu sais lire un modèle en couches, calculer des sous-réseaux, raisonner sur le routage, écrire des clients et serveurs TCP/HTTP, comprendre DNS et TLS, configurer des règles de filtrage et dépanner un service couche par couche.
→ Prochain module : **HEF-070 — Bases de données**

MISSION ACCOMPLIE

+120 XP

Badge débloqué : Détective du réseau

Fin du module HEF-060. Bravo !

