MSPR 1
Rapport de travail

epsi®

l'école d'ingénierie informatique

Mathis / Khalifa / Kerim

CYB3R XP 2025/2026

Contexte

NFL IT – National Football League Information Technology est une société de services numériques (SSII) créée au début des années 2020 par deux associés français. Elle fournit des prestations d'infogérance, d'audit, de conseil et d'intégration orientées infrastructures informatiques pour des organisations multisites à forts enjeux de disponibilité.

Historiquement, NFL IT s'est spécialisée dans l'accompagnement des équipes de football américain (gestion d'équipes, statistiques, applicatifs métiers), domaine que les fondateurs connaissent bien pour y avoir exercé comme développeurs chez l'éditeur de référence du secteur.

Depuis 2020, NFL IT est titulaire d'un contrat d'exclusivité de plusieurs millions de dollars avec la National Football League (NFL), qui regroupe 32 franchises. L'entreprise compte plus de 100 collaborateurs (environ 30 techniciens/ingénieurs itinérants, 30 personnels support, et ~40 fonctions supports). Le siège est situé à Kansas City (Missouri) ; les clients sont répartis sur tout le territoire américain.

Pour ses besoins d'hébergement, NFL IT s'appuie également sur un datacenter en France (Roubaix) pour certaines applications clients et internes. Dans la continuité de cette stratégie, l'entreprise initie le programme "Seahawks monitoring", destiné à réduire les interventions sur site, industrialiser la supervision et améliorer la maintenabilité du parc chez les franchises, avec une perspective d'extension vers la ligue européenne (ELF).

Enjeu business : aujourd'hui, l'absence d'outils unifiés de maintenance et de visibilité entraîne des déplacements coûteux et des délais de résolution élevés. NFL IT porte donc une feuille de route visant à standardiser la collecte d'informations techniques, centraliser la supervision, et outiller le support pour intervenir à distance lorsque c'est possible.

Démarche Suivie

Pour réaliser notre projet, on a d'abord pris le temps de bien définir ce qu'on voulait faire et comment organiser notre travail. On a ensuite réparti les rôles selon les compétences de chacun pour être plus efficaces. On a utilisé GitHub pour centraliser nos fichiers, suivre les tâches et collaborer plus facilement.
Puis, on a avancé étape par étape : création des scripts, mise en place des machines virtuelles, configuration des outils (Flask, Nginx, SCP, PM2…), et enfin les tests pour vérifier que tout fonctionnait bien.
Cette organisation nous a permis d'avancer de manière claire, malgré plusieurs difficultés rencontré cours du projet, notamment lors de la configuration et de l'automatisation des échanges entre les machines.

Choix Technologique Motivé

Nous avons utilisé plusieurs technologies et outils au cours du projets :

PM2 : Sert à lancer l'environnement NodeJS au démarrage de la machine virtuel « Nester », garder l'application toujours en ligne, redémarre automatiquement l'application si elle plante ou si le serveur redémarre, faciliter la gestion et le monitoring.

SCP : Le partage d'information et de fichiers entre nos deux machines utilise SCP. C'est une commande qui utilise SSH et avec une couche de sécurité en plus lors de l'envoie

Python3 / Flask : Les scripts pour le scan réseau est écrit en Python3 et utilise la bibliothèque « nmap ». Le site ne devant pas spécialement être héberger, nous avons choisi d'utiliser Flask car plus compatible avec le script.

HTML / CSS / JavaScript : Dashboard simple en HTML / CSS. On utilise un script JavaScript pour traiter les fichiers json reçu de la part des franchises (script python nmap qui sort un json) et afficher de façon dynamique dans le DOM les informations.

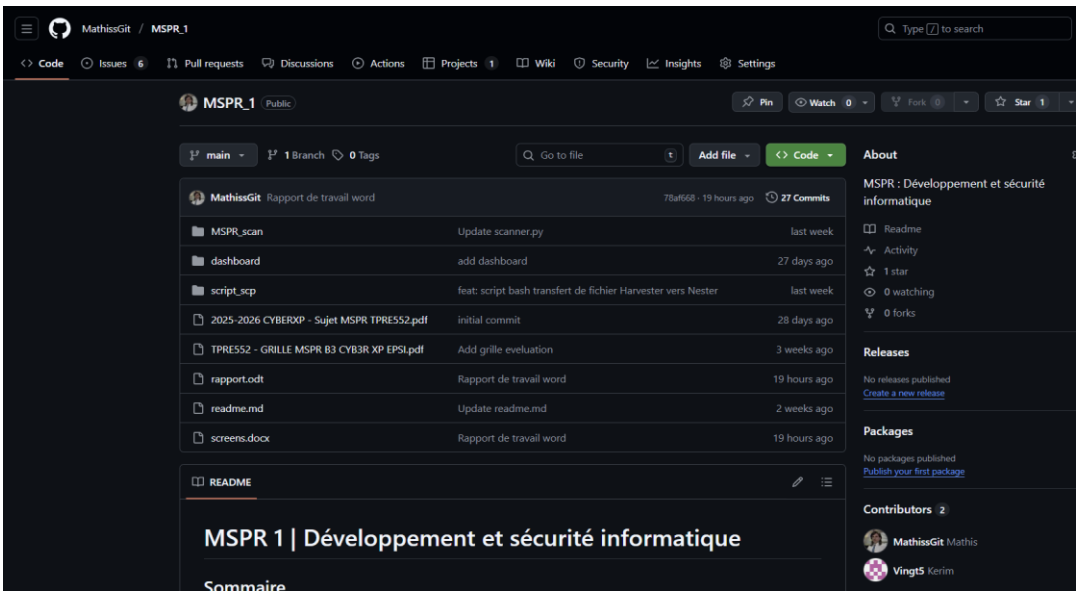
Nginx : Hébergement du dashboard sur la machine « Nester », Reverse-Proxy. Nous avons choisi Nginx car nous avions besoin d'un site pouvant être héberger simplement, avec une configurabilité fine pour la souveraineté de notre projet et qui puisse utiliser un reverse-proxy.

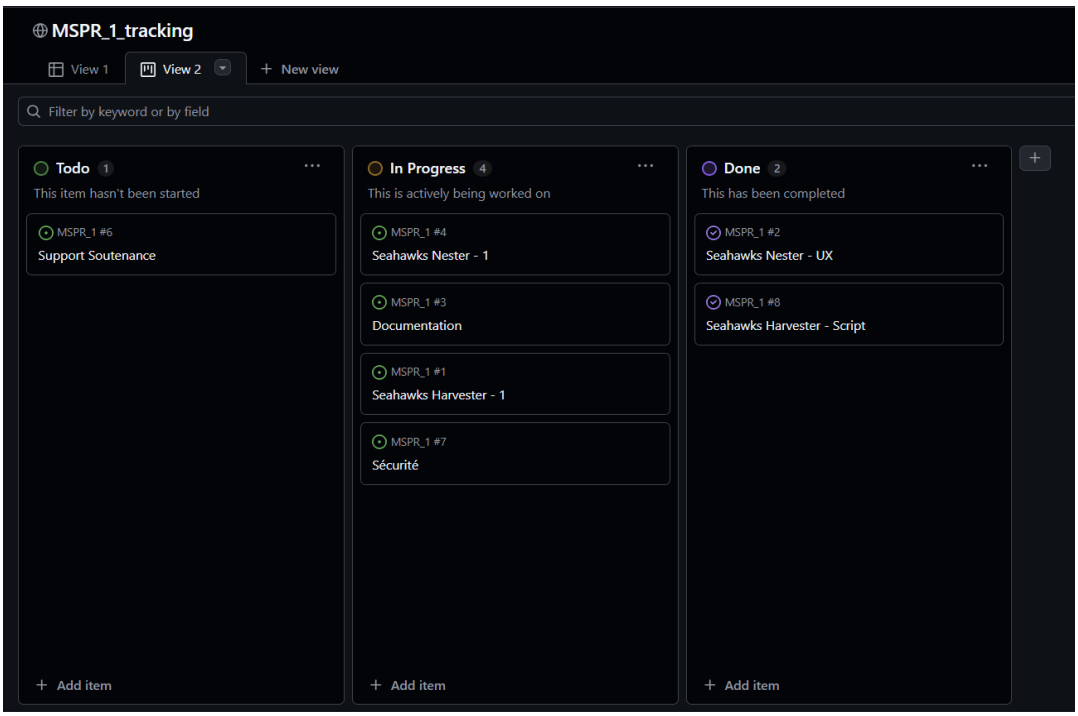
Git / GitHub : Pour la gestion de projet, des taches et le partage des documents relatifs au projet.

Organisation de l’équipe

Workflow

Nous avons utilisé un repository GitHub pour regrouper les fichiers, scripts, machines virtuels, rapports etc… afin de facilité la gestion globale de notre projet.
Le repository permet de fournir une documentation README supplémentaire au projet.
La plateforme GitHub intègre aussi un espace de gestion de projet (Kanban) qui nous a servi à lister nos taches, assignés des missions et suivre notre projet en temps réel.

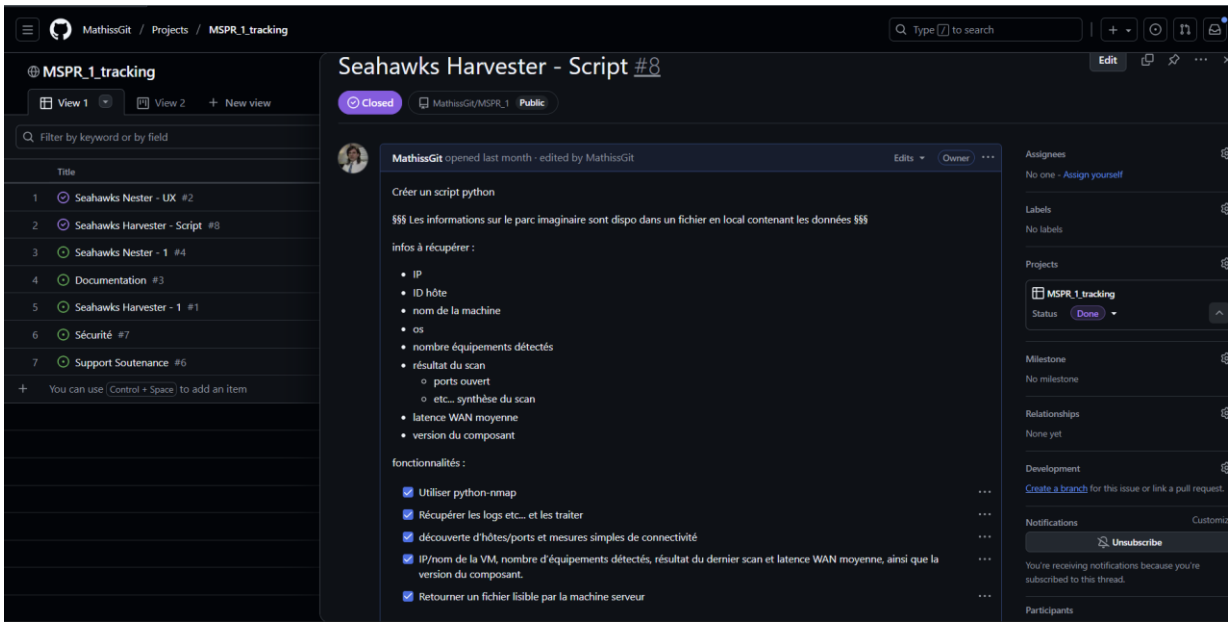




Rôles et Taches

Nous avons répartie les rôles selon les compétences dont chacun dispose. En parallèle, nous avons documenter nos taches pour que ceux qui n'ont pas travailler dessus puisse comprendre.

Mathis : Machine « Nester », documentation, gestion de projet, automatisation, dashboard…
Khalifa : Machine « Harvester », automatisation, documentation, SCP, dashboard, logs…
Kerim : Script Python scan nmap, documentation & supports…



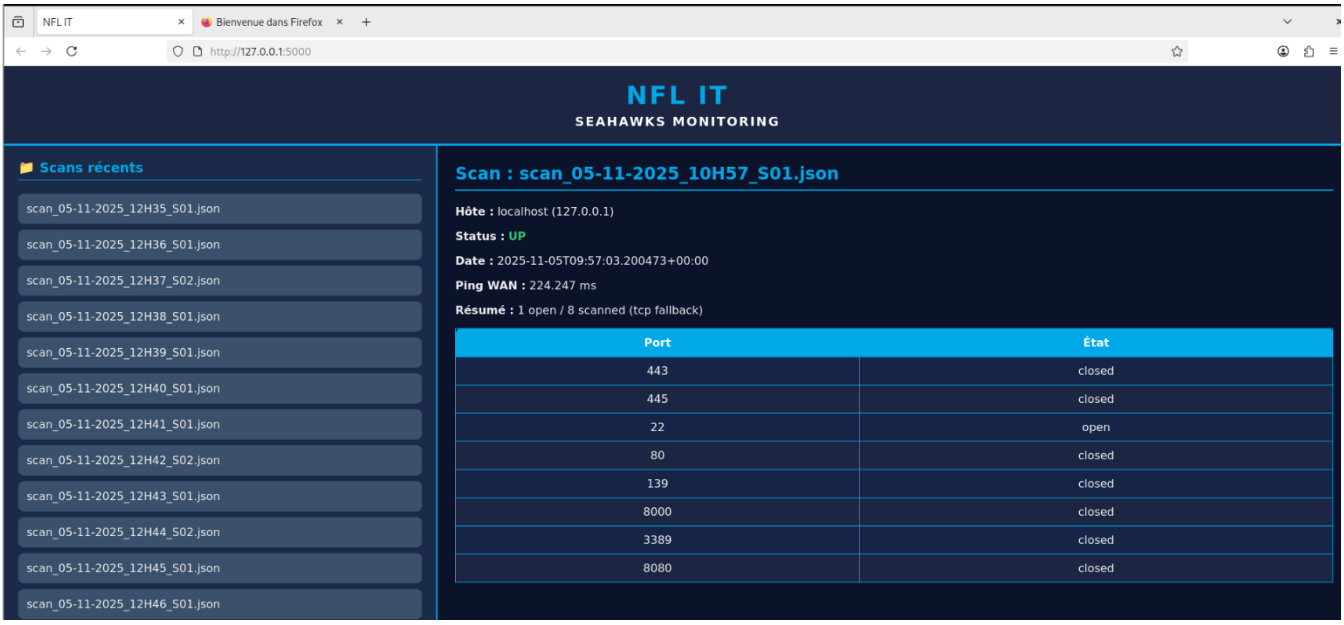
Ce qui a été fait

Seahawks Harvester (franchise)

Script Python : automatisation du scan-nmap sur l'infrastructure de la franchise.

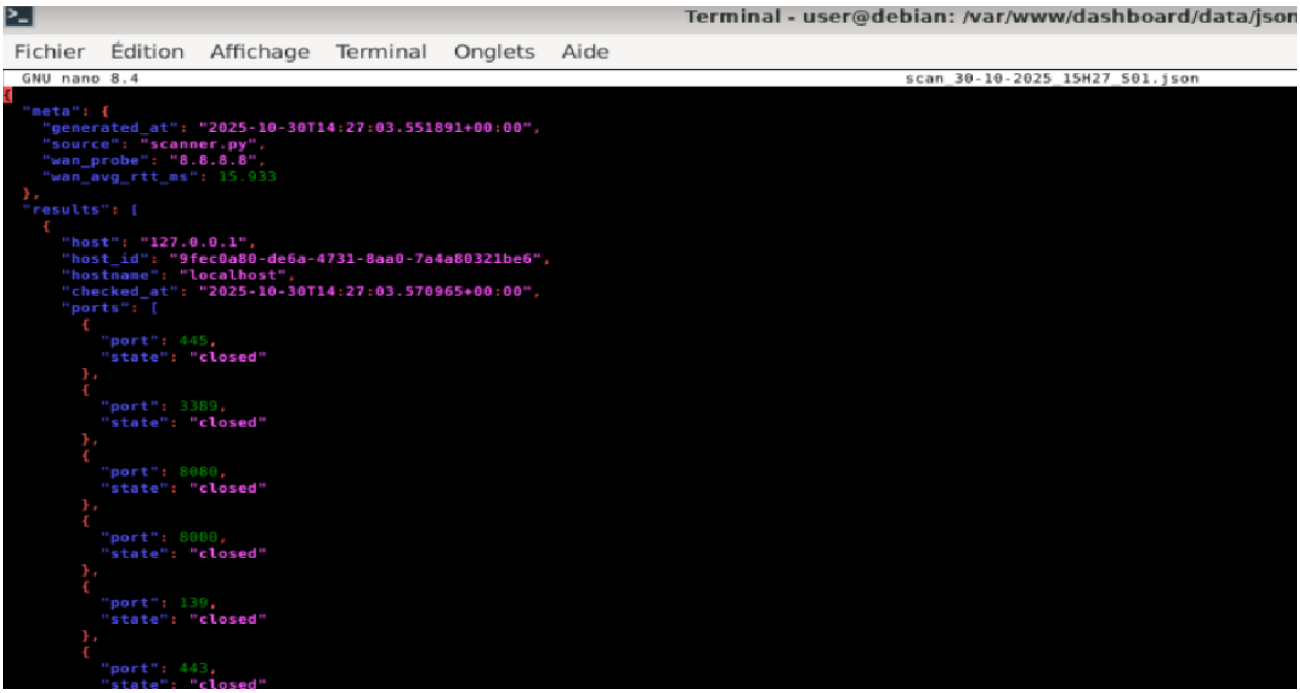
- https://github.com/MathissGit/MSPR_1/tree/main/MSPR_scan

Tableau de bord local minimal pour afficher les derniers rapports :



Fichier contenant les données stockées en local (JSON).

- Version avec identifiant et date/heures



Automatisation avec un script pour le transfert des fichiers avec SCP (chiffrement des informations par défaut en utilisant SCP).

```
 GNU nano 8.4                                    /root/MSPR_1/MSPR_scan/Script/watch_and_send.sh
#!/bin/bash
# Surveillance automatique du dossier reports et envoi SCP vers Nester

SRC_DIR="/root/MSPR_1/MSPR_scan/Script/reports"
DEST_USER="adminNester"
DEST_HOST="172.20.10.4"
DEST_DIR="/var/www/dashboard/data/json"
SSH_KEY="/root/.ssh/id_ed25519_reports"
SSH_PORT=22334
LOG_FILE="/root/MSPR_1/MSPR_scan/Script/watch_and_send.log"

echo "=== Surveillance démarrée à $(date '+%Y-%m-%d %H:%M:%S') ===" >> "$LOG_FILE"

# Vérifie que le dossier de destination existe à distance
ssh -i "$SSH_KEY" -p "$SSH_PORT" -o StrictHostKeyChecking=no "${DEST_USER}@${DEST_HOST}" \
  "mkdir -p '$DEST_DIR' && chmod 755 '$DEST_DIR'"

# Surveillance continue du dossier source
inotifywait -m -e create --format '%w%f' "$SRC_DIR" | while read NEW_FILE
do
    if [[ "$NEW_FILE" == *.json ]]; then
        echo "[$(date '+%H:%M:%S')] Nouveau fichier détecté : $NEW_FILE" >> "$LOG_FILE"

        # Tentatives d'envoi avec retries
        for i in {1..3}; do
            scp -P "$SSH_PORT" -i "$SSH_KEY" -o StrictHostKeyChecking=no "$NEW_FILE" \
                "${DEST_USER}@${DEST_HOST}:${DEST_DIR}/" >> "$LOG_FILE" 2>&1 && {
                echo "[$(date '+%H:%M:%S')] ✅ Fichier transféré : $(basename "$NEW_FILE")" >> "$LOG_FILE"
                break
            }
            echo "[$(date '+%H:%M:%S')] ⚠️Tentative $i échouée, nouvelle tentative dans 5s..." >> "$LOG_FILE"
            sleep 5
        done
    fi
done
```

Ce script utilise aussi le programme linux « Inotify » qui permet de détecter la présence ou non d'un nouveau fichier de rapport Json afin de pour pouvoir l'envoyer grâce au script.

Seahawks Nester (centre)

Installation de NodeJS et NPM pour pouvoir lancer le dashbord et les scripts JavaScript :

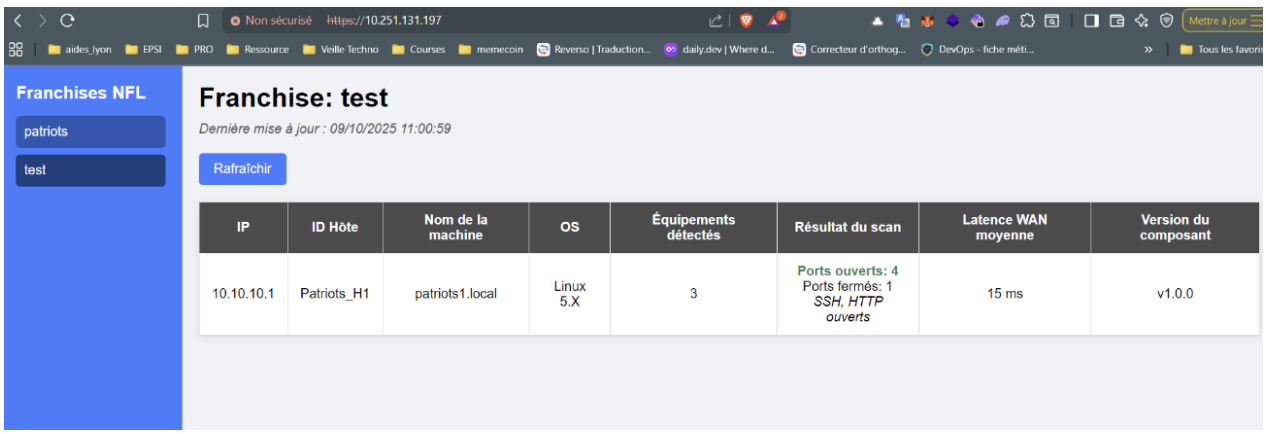
```
user@debian:~$ wget https://nodejs.org/dist/v22.20.0/node-v22.20.0-linux-x64.tar
.xz
--2025-10-09 11:47:20--  https://nodejs.org/dist/v22.20.0/node-v22.20.0-linux-x6
4.tar.xz
Résolution de nodejs.org (nodejs.org)… 104.20.1.252, 172.66.128.70, 2606:4700:10
::ac42:8046, ...
Connexion à nodejs.org (nodejs.org)|104.20.1.252|:443… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 30737976 (29M) [application/x-xz]
Sauvegarde en : « node-v22.20.0-linux-x64.tar.xz »

node-v22.20.0-linux 100%[===================>]  29,31M  10,3MB/s    ds 2,8s

2025-10-09 11:47:31 (10,3 MB/s) – « node-v22.20.0-linux-x64.tar.xz » sauvegardé
[30737976/30737976]
```

Serveur Nginx et reverse-proxy pour l'hébergement de notre application web.

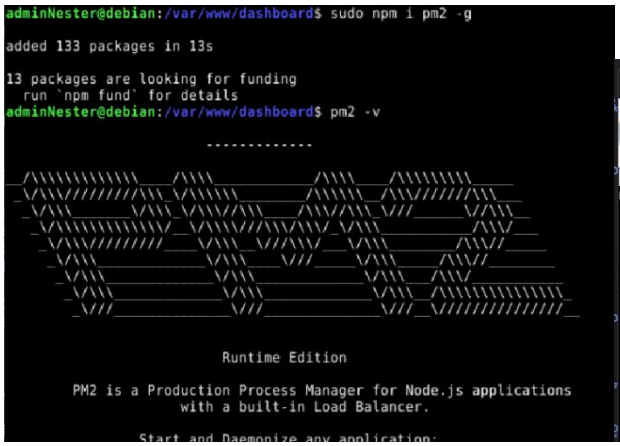
```
Fichier  Édition  Affichage  Terminal  Onglets  Aide
  GNU nano 8.4                          /etc/nginx/sites-available/default *

server {

        listen 80;
        server_name 10.251.131.197;
        return 301 https://$host$request_uri;
}

server {

        listen 443 ssl;
        server_name 10.251.131.197;

        ssl_certificate /etc/ssl/nestersite/ca.crt;
        ssl_certificate_key /etc/ssl/nestersite/mykeynester.pem;

        location / {
                proxy_pass http://localhost:3000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }

        location /static/ {

                root /var/www/dashboard/public;
                index index.html index.htm;
                try_files $uri $uri/ =404;
        }

}
```

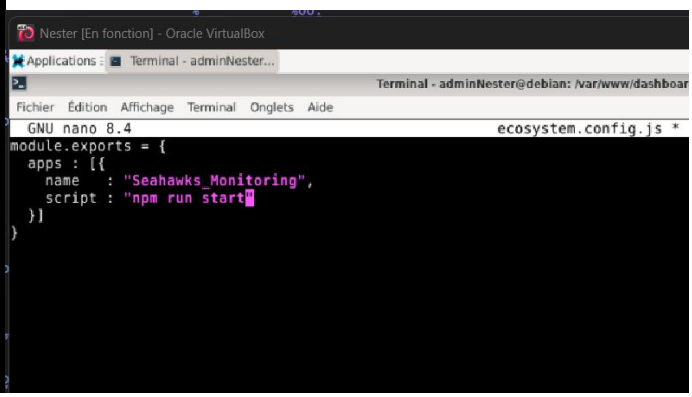
Application de consultation type dashboard permettant de :

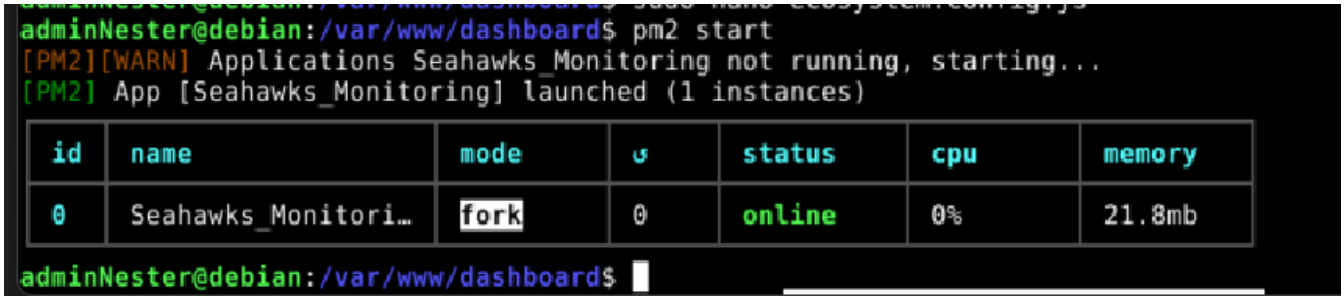
- Lister les sondes
- Obtenir les détails d'une sonde (lecture directe du rapport)



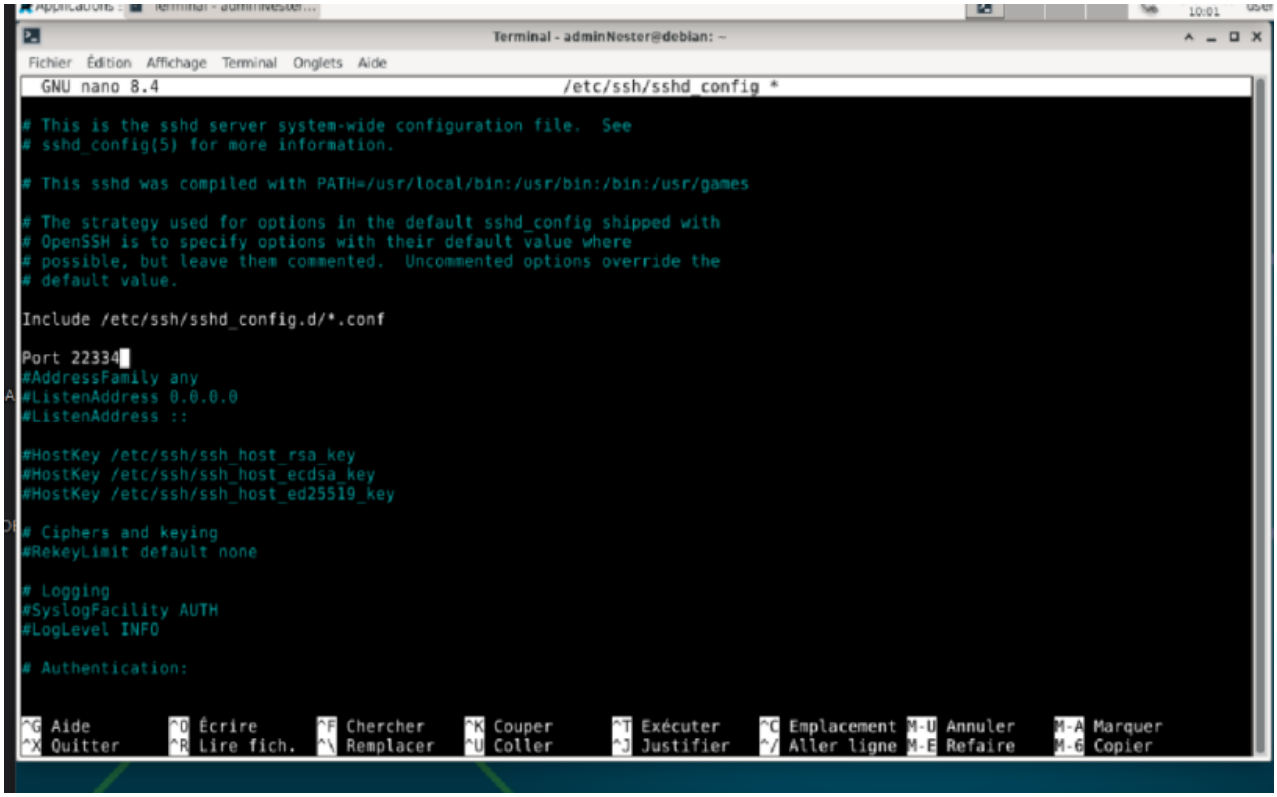
Gestion et monitoring de l’application dashboard (PM2)





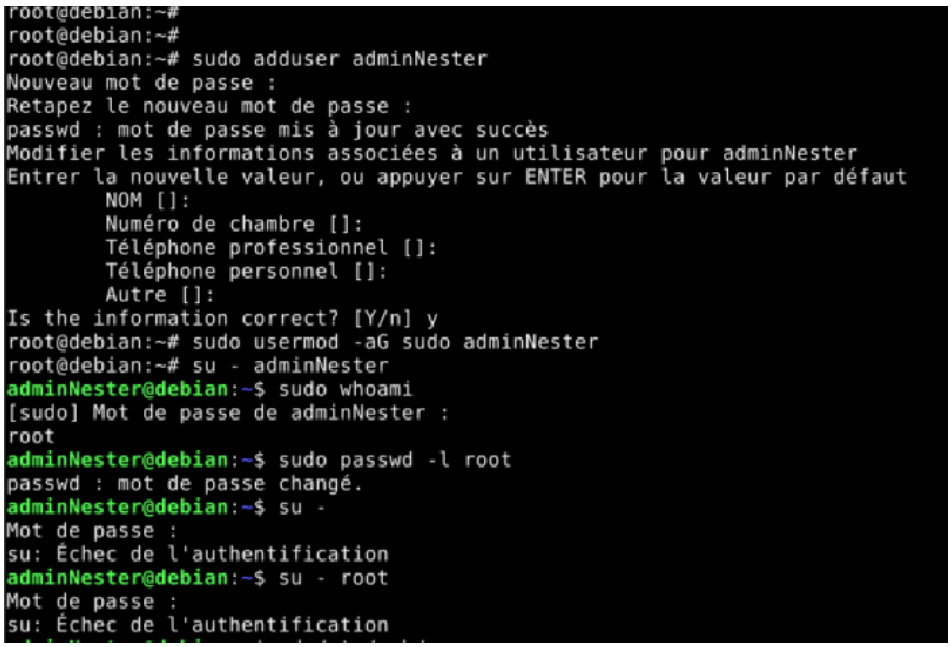


Changement du port par défaut SSH (22 -> 22334)

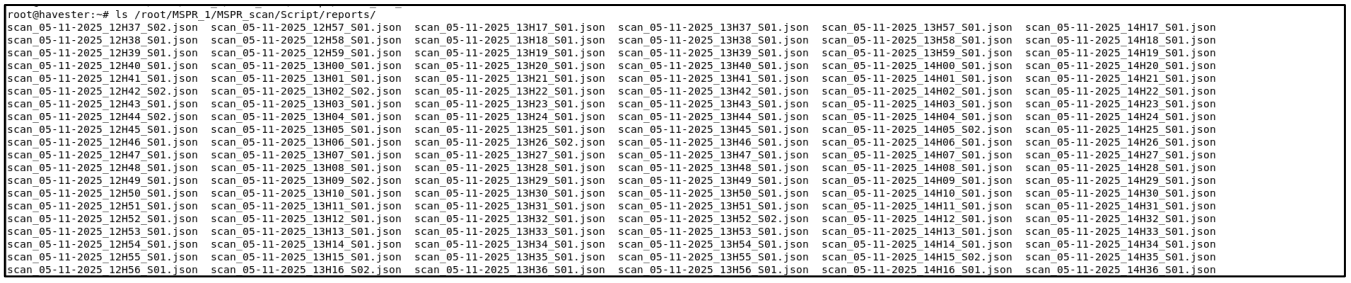


Sécurité

- Suppression du compte root par défaut : Nous avons désactivé le compte root par défaut pour éviter les attaques par brute force. À la place, on a créé un nouvel utilisateur avec les mêmes privilèges, mais seulement quand c'est nécessaire. Cette mesure permet de limiter les risques de piratage liés aux attaques par brute force et de restreindre l'accès aux privilèges d'administration.



- Intégrité et traçabilité des rapports : Chaque rapport généré par le système est sous forme binaire et possède des informations clés qui en garantissent l'intégrité. Chaque fichier est horodaté et versionné, ce qui permet de garantir sa traçabilité et de faciliter l'audit en cas de besoin. Toute modification ou mise à jour du rapport est ainsi identifiable grâce à son horodatage et à son numéro de version, assurant la fiabilité des données collectées.



- Sécurisation des échanges via SCP : Pour le transfert de fichiers entre nos machines, nous utilisons SCP (Secure Copy Protocol) en combinaison avec une paire de clés public/privé. Ce mécanisme garantit que les fichiers sont transmis de manière sécurisée, en cryptant les échanges pour éviter toute interception malveillante. Les fichiers transférés sont stockés dans un répertoire spécifique, utilisé par le dashboard pour afficher les logs du système en toute sécurité.

- Éthique & périmètre : collectes limitées aux environnements de lab/franchise autorisés ; pas de tests intrusifs.

Runbook d'exploitation N1/N2 (franchise & centre)

- Runbook d'exploitation scanner Python NMAP : https://github.com/MathissGit/MSPR_1/blob/main/MSPR_scan/Runbook_Scan.pdf

Rapport de travail

Document actuel.

Support de soutenance

Support de présentation synthétisant les principaux éléments du travail réaliser :

- Démarche suivie
- Difficultés rencontrées
- Solutions mises en place
- Résultats obtenus
- Perspectives.

Perspectives d'améliorations

- Ajouter les runbook d'exploitation et la documentation sur les dashboard client.
- Créer et mettre en place un certificat SSL avec un nom de domaine pour les dashboard hébergé en ligne
- Authentification et gestion de session (OWASP A2) :
    - Mettre en place un système d'authentification forte pour les diffèrent sites (Nester/Harvester) en utilisant OAuth ou des tokens JWT
    - Utilisation de sessions sécurisées avec des cookies (''session'' dans Flask)
- Chiffrement des informations sensible (OWASP A3) qui peuvent être présent dans les scripts ou en ligne -> utilisation de AES
- Utiliser des mots de passe fort en production
- Contrôle des rôles attribué aux utilisateurs (administrateur, exécution des scripts, etc…)
- Faille XSS : Ajouter des headers CSP dans la configuration Nginx pour restreindre les sources de scripts exécutables et éviter l'injection de JavaScript malveillant.
- Logging et surveillance (OWASP A10) :
    - Activer la journalisation dans Flask et Nginx
    - Surveiller les tentatives de connexion avec ''fail2ban''