Documentation technique NTL-Systoolbox

Date : Février 2026

Projet : MSPR TPRE511 - Développement d'application NordTransit Logistics

Contexte académique

Ce document a été réalisé dans le cadre du projet MSPR TPRE511 par le Groupe 2 composé de Yohan PETIT, Paul LUGIEN, Floyd LAFORGE et Mathéo CORRE de la promotion B3 ASRBD de l 'EPSI Auxerre.

Projet académique - Année 2025-2026

Sommaire

1. Architecture globale 4
1.1 Vue d'ensemble 4
1.2 Architecture modulaire 4
1.3 Principe de conception 5
1.4 Flux d'exécution général 5
2. Module 1 : Diagnostic AD/DNS + MySQL 5
2.1 Objectif 5
2.2 Architecture technique 6
2.3 Objectif 7
2.4 Architecture technique 7
2.5 Mécanismes de test 7
3.1 Description fonctionnelle 10
Connexion à la base 10
Sauvegarde complète SQL 11
Export d'une table en CSV 11
Horodatage au format json 12
3.2 Architecture technique 12
3.3 Organisation des répertoires 13
3.4 Sécurité 13
3.5 Gestion des erreurs 13
3.6 Contraintes techniques 13
3.7 Perspectives d'évolution 14
4.1. Présentation Générale 15
4.2. Architecture Globale 15
4.3. Moteur de Découverte : Nmap 15
4.3.1. Pourquoi Nmap ? 15
4.3.2. Fingerprinting OS : Mécanisme Technique 15
4.3.3. Options de Scan Retenues 16
4.3.4. Choix du Format XML pour le Parsing 16
4.4. Normalisation et Mapping des Données OS 16
4.4.1. Le Problème de l'Hétérogénéité 16
4.4.2. Stratégie de Matching à Double Niveau 17
4.4.3. Extraction de Version par Expression Régulière 17
4.5. Gestion des Données EOL (End of Life) 17
4.5.1. Architecture Hybride : Cache Local + API 17
4.5.2. Format de Stockage : JSON 17

4.5.3. Calcul du Statut EOL 18
4.6. Export et Reporting 18
4.7. Considérations de Sécurité et Limitations 18
4.7.1. Prérequis de Privilèges 18
4.7.2. Limitations de la Détection OS 18
4.7.3. Impact Réseau 18
4.8. Prérequis et Déploiement 19
4.9. Référentiel de Conformité EOL 19

1. Architecture globale

1.1 Vue d'ensemble

NTL-SysToolbox est une suite d'outils CLI (Command Line Interface) modulaire développée en Python 3.7+ pour automatiser les opérations d'administration système critiques de NordTransit Logistics.

1.2 Architecture modulaire

Élément	Description
main.py	Point d’entrée principal (menu CLI)
Élément	**Description**
Module1.py	Diagnostic AD/DNS + MySQL + OS
Module2.2.py	Sauvegarde base WMS
Module3.py	Audit obsolescence réseau (EOL)
backups/	Sauvegardes SQL générées
exports/	Exports CSV tables
reports/	Rapports d’analyse
audits/	Résultats scans nmap + EOL
nmap_scan.xml	Sortie brute scan nmap
eol_data_local.json	Cache local données EOL
README.md	Documentation utilisateur

1.3 Principe de conception

Modularité :

• Chaque module est un script Python autonome exécutable indépendamment

• `main.py` sert de menu central mais n'est pas obligatoire

• Pas de dépendances inter-modules (couplage faible)

Auto-installation des dépendances :

• Chaque module vérifie et installe automatiquement ses dépendances Python manquantes

• Utilisation de `subprocess` pour appeler `pip install` si nécessaire

Portabilité multi-OS :

• Compatible Linux (Debian, Ubuntu, CentOS, RHEL) et Windows (10, 11, Server)

• Détection automatique de l'OS via `platform.system()`

• Chemins fichiers gérés avec `os.path` pour compatibilité Windows/Linux

1.4 Flux d'exécution général

Utilisateur lance main.py

□

Menu principal affiché

□

Choix module (1, 2 ou 3)

□

Exécution du module sélectionné

□

Affichage résultats + génération fichiers

□

Retour au menu principal (ou sortie)

2. Module 1 : Diagnostic AD/DNS + MySQL

2.1 Objectif

Vérifier l'état de santé des services critiques de l'infrastructure NTL :

• Contrôleurs de domaine (Active Directory + DNS) : Ports LDAP (389), Kerberos (88), DNS (53)

• Base de données MySQL WMS : Connexion, authentification, requête test

• Informations OS : Systèmes d'exploitation des serveurs et version MySQL

2.2 Architecture technique

Composant	Description
Langage	Python 3.7+
Dépendances	mysql-connector-python (auto-installation)
Bibliothèques standard	socket, sys, subprocess, getpass
Fichier source	Module1.py (□180 lignes)
Configuration	DOMAIN_CONTROLLERS / MYSQL_STATIC

Présentation du module 1

2.3 Objectif

Vérifier l'état de santé des services critiques de l'infrastructure NTL :

• Contrôleurs de domaine (Active Directory + DNS) : Ports LDAP (389), Kerberos (88), DNS (53)

• Base de données MySQL WMS : Connexion, authentification, requête test

• Informations OS : Systèmes d'exploitation des serveurs et version MySQL

2.4 Architecture technique

Composant	Description
Langage	Python 3.7+
Dépendances	mysql-connector-python (auto-installation)
Bibliothèques standard	socket, sys, subprocess, getpass
Fichier source	Module1.py (□180 lignes)
Configuration	DOMAIN_CONTROLLERS / MYSQL_STATIC

2.5 Mécanismes de test

2.3.1 Test de disponibilité des ports (AD/DNS)

Fonction : `check_tcp_port(host, port, timeout=3)`

Principe :

• Tentative de connexion TCP bas niveau via `socket.create_connection()`

• Timeout de 3 secondes par port (configurable)

• Retourne `True` si le port répond, `False` sinon

Ports testés :

• 389/tcp : LDAP (Active Directory)

• 88/tcp : Kerberos (authentification AD)

• 53/tcp : DNS

Avantages :

• Pas besoin de privilèges élevés (contrairement à ICMP ping)

• Test réel de la disponibilité du service (pas seulement du host)

• Rapide (3 secondes max par port)

2.3.2 Test de connexion MySQL

Fonction : `test_mysql()`

Principe :

1. Demande interactive des identifiants MySQL (user + password)

2. Connexion via `mysql.connector.connect()`

3. Exécution requête test `SELECT 1`

4. Vérification du résultat (doit retourner `1`)

Sécurité :

• Utilisation de `getpass.getpass()` pour masquer le mot de passe à la saisie

• Identifiants stockés uniquement en mémoire pendant l'exécution

• Variable globale `MYSQL_CREDENTIALS` pour éviter de redemander les identifiants à chaque test

Gestion des erreurs :

• `mysql.connector.Error` capturée pour afficher message d'erreur MySQL précis

• Fermeture propre de la connexion dans bloc `finally`

2.3.3 Récupération version MySQL

Fonction : `get_mysql_version()`

Principe :

• Connexion MySQL identique à `test_mysql()`

• Exécution requête `SELECT VERSION()`

• Parsing et retour de la chaîne version (ex: `8.0.35-0ubuntu0.20.04.1`)

Utilité :

• Permet de tracker l'obsolescence MySQL

• Aide au diagnostic de compatibilité applicative

Séparation configuration/identifiants :

• `MYSQL_STATIC` : Paramètres non sensibles (host, port, base)

• `MYSQL_CREDENTIALS` : User + password (en mémoire uniquement, jamais stockés)

2.6 Menu interactif

Choix	Action
1	Test AD/DNS (389, 88, 53)
2	Test connexion MySQL (SELECT 1)
3	Diagnostic OS + version MySQL
0	Quitter le programme

Présentation du module 2

Le Module 2 permet d'assurer la sauvegarde et l'export des données d'une base MariaDB utilisée par le système WMS.

Objectifs principaux :

- Sauvegarde complète de la base au format SQL

- Export d'une table spécifique au format CSV

- Génération d'un rapport d'exécution au format JSON

3.1 Description fonctionnelle

Connexion à la base

Au lancement du script, l'utilisateur doit renseigner :

L'adresse IP du serveur MariaDB
Le nom de la base de données
Le nom d'utilisateur
Le mot de passe

```
NTL-SysToolbox - Menu principal
1 - Module 1 : Diagnostic AD/DNS + MySQL + OS serveurs
2 - Module 2 : Sauvegarde de la base WMS
3 - Module 3 : Audit d'obsolescence reseau (EOL)
0 - Quitter
Votre choix : 2

=== Connexion a la base WMS ===
IP du serveur MariaDB : 10.5.20.113
Nom de la base : wms
Utilisateur : backup
Mot de passe :
Connexion a la base reussie
```

Une tentative de connexion est immédiatement effectuée.

Si la connexion échoue :

- Un message d'erreur est affiché
- Les informations sont redemandées

Si la connexion réussit :
Le menu principal s'affiche

Sauvegarde complète SQL

La fonctionnalité permet d'extraire l'ensemble de la structure et des données de la base WMS afin de générer un fichier SQL horodaté.

Processus :

1. Récupération des tables (SHOW TABLES)

2. Extraction des structures (SHOW CREATE TABLE)

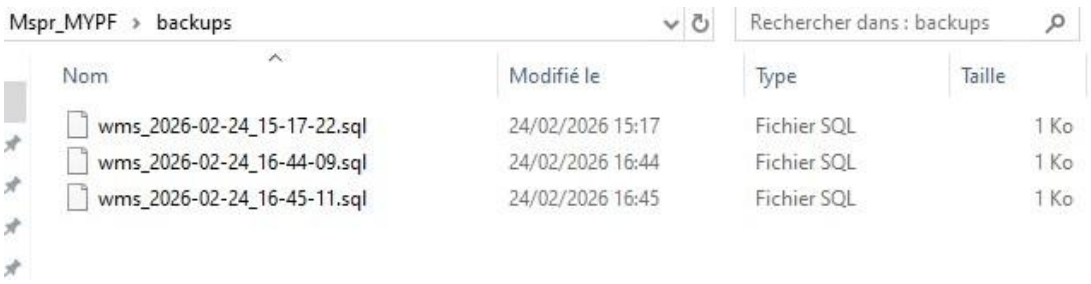
3. Extraction des données (SELECT *)

4. Génération des requêtes INSERT

5. Création du fichier .sql dans le dossier backups/

```
=== Module Sauvegarde WMS ===
1 - Sauvegarde complète (SQL)
2 - Export d'une table (CSV)
0 - Quitter
Choix : 1
 Sauvegarde SQL réussie
```

Visuellement on ne voit que le message de succès ou d'échec de la sauvegarde, on peut ensuite la récupérer dans le dossier backups au format sql.



Export d'une table en CSV

L'utilisateur peut sélectionner une table parmi celles disponibles et exporter son contenu au format CSV.

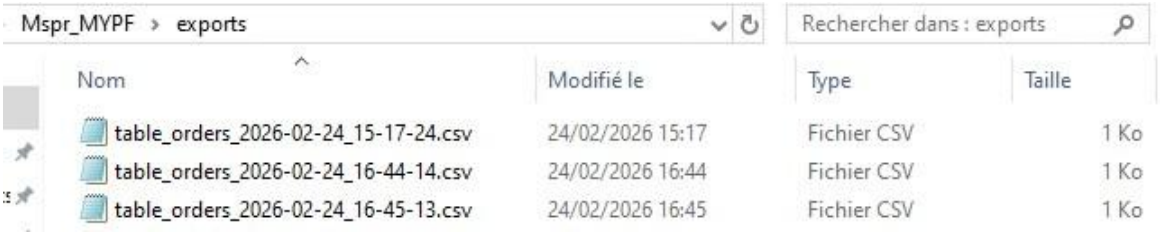
Processus :

Une fois la table sélectionné, le module exécute (SELECT * FROM nom_table;)

Toutes les lignes et colonnes du fichier sont récupérées et générées dans un fichier au format csv.

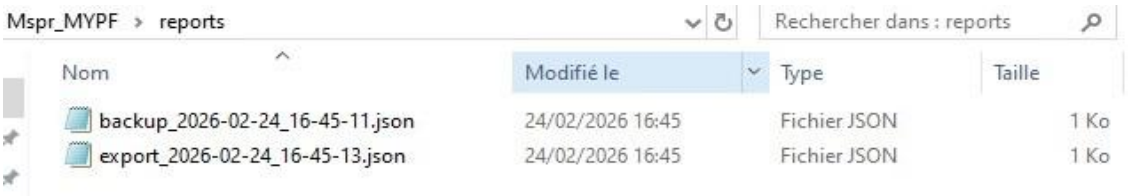
```
=== Module Sauvegarde WMS ===
1 - Sauvegarde complète (SQL)
2 - Export d'une table (CSV)
0 - Quitter
Choix : 2

Tables disponibles :
1 - orders
Choisis le numéro de la table : 1
 Export CSV réussi (orders)
```

Le fichier est généré dans le dossier exports/. Au format csv.



Horodatage au format json

Chaque opérations produit un fichier json pour suivre de manière plus lisible nos sauvegarde et export.



3.2 Architecture technique

Langage : Python 3

Base de données : MariaDB

Connecteur : mysql-connector-python

Modules utilisés :

mysql.connector
os
datetime
csv
json
getpass

3.3 Organisation des répertoires

À l'exécution, le module crée automatiquement l'arborescence suivante :

├── backups/ (sauvegardes SQL)

├── exports/ (exports CSV)

└── reports/ (rapports JSON)

3.4 Sécurité

Mot de passe masqué via getpass : permet de masquer la saisie à l'écran et l'affichage en clair dans le terminal.

Pas de stockage du mot de passe

Vérification de la connexion avant exécution, via le connecteur python officiel mysql-connector-python : Cela permet une connexion tcp/ip vers le serveur MariaDB, d'authentifier l'utilisateur pour pas que n'importe qui y est accès.

3.5 Gestion des erreurs

Le module gère les erreurs de connexion, les erreurs SQL, les choix invalides utilisateur et les exceptions système. Il envoie des messages d'erreur à l'utilisateur.

```
Tables disponibles :
1 - orders
Choisis le numéro de la table : 2
Choix invalide
```

```
=== Module Sauvegarde WMS ===
1 - Sauvegarde complète (SQL)
2 - Export d'une table (CSV)
0 - Quitter
Choix : 1
 ERREUR DÉTAILLÉE : 2003: Can't connect to MySQL server on ':3306' (Errno 10061: Aucune connexion n'a pu être établie car l'o
rdinateur cible l'a expressément refusée)
```

3.6 Contraintes techniques

- Accès réseau au serveur MariaDB

- Port 3306 ouvert

- Compte disposant des droits SELECT et SHOW CREATE

3.7 Perspectives d’évolution

- Chiffrement des sauvegardes

- Compression automatique

- Sauvegarde distante (SFTP)

- Planification automatique via cron

Présentation du module 3

4.1. Présentation Générale

Ce module est un outil d'inventaire et d'audit de conformité du cycle de vie logiciel à destination des administrateurs système et réseau. Son objectif principal est d'automatiser la détection des systèmes d'exploitation (OS) actifs sur un réseau et de croiser ces informations avec les dates de fin de support officielles (End of Life – EOL) publiées par les éditeurs.
Un système non supporté représente un vecteur d'attaque critique : absence de correctifs de sécurité (CVE), non-conformité aux référentiels ANSSI, ISO 27001, ou RGPD. Cet outil permet d'identifier et prioriser ces risques de manière proactive.

4.2. Architecture Globale

Le script suit une architecture pipeline en trois étapes distinctes :

1. Découverte réseau (Scanning) : Énumération des hôtes actifs et détection de leur OS.
2. Corrélation de données (Data Mapping) : Normalisation des résultats bruts et interrogation de la base EOL.
3. Reporting : Production d'un rapport CSV et affichage d'un résumé de conformité.

Chaque étape est découplée : il est possible d'alimenter le module directement via un fichier CSV externe (option 3 du menu), sans forcément lancer de scan réseau. Cela rend l'outil compatible avec des données issues d'autres outils d'inventaire (GLPI, OCS Inventory, etc.).

4.3. Moteur de Découverte : Nmap

4.3.1. Pourquoi Nmap ?

Nmap (Network Mapper) est l'outil de référence en matière de découverte réseau. Il intègre deux moteurs distincts mobilisés dans ce script : le scanner de ports TCP/UDP, et le moteur de fingerprinting d'OS. Des alternatives Python pures existent (Scapy, python-nmap), mais elles ne permettent pas d'atteindre la maturité et la précision de la base de signatures de Nmap, qui contient plusieurs milliers d'empreintes d'OS référencées.

4.3.2. Fingerprinting OS : Mécanisme Technique

Le fingerprinting d'OS de Nmap repose sur une technique d'analyse de la pile TCP/IP. Chaque implémentation du protocole TCP/IP par un OS laisse des comportements réseaux légèrement différents. Nmap exploite ces écarts en envoyant une série de sondes (probes) TCP, UDP, et ICMP craftées, puis en comparant les réponses obtenues à sa base de signatures. Les indicateurs analysés incluent notamment :

- La valeur initiale de TTL (Time To Live) des paquets retournés (Linux = 64, Windows = 128, Cisco = 255 typiquement).
- La taille de la fenêtre TCP (TCP Window Size) dans les paquets SYN-ACK.
- Les options TCP proposées et leur ordre (MSS, SACK, Timestamps, NOP).
- Le comportement face aux paquets invalides ou malformés (réponses ICMP Unreachable).

Le résultat est une correspondance probabiliste avec un score de précision (0-100%). L'option `--osscan-guess` abaisse le seuil d'acceptation pour forcer un résultat même lorsque la correspondance est partielle — comportement préférable dans un contexte d'audit où une estimation vaut mieux qu'aucune information.

4.3.3. Options de Scan Retenues

Le script appelle Nmap avec les flags suivants, chacun justifié techniquement :

- -O (OS Detection) : Active le moteur de fingerprinting OS basé sur les probes TCP/IP. Requiert des privilèges root/admin car il utilise des raw sockets.
- -sV (Version Detection) : Interroge les services ouverts pour identifier le produit et sa version. Nmap envoie des payloads spécifiques et analyse les bannières renvoyées. Cela permet de détecter la distribution Linux via un service SSH qui se présente par exemple en 'OpenSSH 8.2p1 Ubuntu 4ubuntu0.5'.
- -Pn (No Ping) : Désactive la phase de découverte ICMP préalable. Indispensable sur des réseaux où le ping est filtré par le pare-feu, sans quoi Nmap considèrerait l'hôte comme inactif et le passerait.
- --script ssl-cert : Exécute le script NSE (Nmap Scripting Engine) d'extraction des certificats TLS. Les certificats SSL/TLS exposent souvent le nom d'hôte (CN) et des métadonnées utiles pour l'identification.
- -T4 (Timing Template 4) : Accélère le scan en réduisant les délais inter-sondes. C'est un compromis entre vitesse et fiabilité, adapté aux LANs. Sur un WAN ou un réseau congest, T3 serait préférable pour éviter les faux négatifs.
- -oX (XML Output) : Force la sortie dans un fichier XML structuré plutôt qu'en texte brut. Voir section 3.4.

4.3.4. Choix du Format XML pour le Parsing

Un choix architectural structurant est d'utiliser la sortie XML de Nmap (`-oX`) plutôt que le parsing du stdout texte. Ce choix est justifié par plusieurs raisons :

- Robustesse : Le format texte de Nmap peut varier selon la version, les options actives, ou la locale système. Le schéma XML est stable et documenté, ce qui rend le parser insensible à ces variations.
- Richesse des données : Le XML embarque des informations inaccessibles dans le stdout, notamment les scores de précision par classe d'OS (osmatch > osclass), les tables de scripts NSE, et l'état précis de chaque port.
- Séparation des préoccupations : Le XML est persisté sur disque (`nmap_scan.xml`). Cela permet de relancer l'analyse sans refaire le scan, utile lors du débogage ou pour rejouer l'analyse avec une configuration différente.

Le module `xml.etree.ElementTree` de la bibliothèque standard est utilisé pour le parsing. Une gestion explicite des namespaces XML est implémentée car certaines versions de Nmap incluent un namespace URI dans le tag racine, ce qui casse un parser naïf.

4.4. Normalisation et Mapping des Données OS

4.4.1. Le Problème de l'Hétérogénéité

Nmap retourne des chaînes libres comme 'Microsoft Windows 10 1809', 'Linux 4.15 - 5.6', ou 'FreeBSD 12.1'. L'API endoflife.date attend des identifiants normalisés ('windows', 'ubuntu', 'freebsd'). Un moteur de normalisation est donc indispensable.

4.4.2. Stratégie de Matching à Double Niveau

La normalisation fonctionne en cascade :

1. Niveau 1 – Dictionnaire statique (OS_MAPPING) : Un mapping explicite couvre les cas les plus fréquents ('windows server' → 'windows-server', 'red hat' → 'rhel'). Ce dictionnaire est ordonné des termes les plus spécifiques aux plus génériques pour éviter les faux positifs.
2. Niveau 2 – Extraction du premier mot (Fallback) : Si aucune correspondance n'est trouvée, le premier mot de la chaîne OS est utilisé comme identifiant. Les valeurs génériques ('linux', 'unknown') sont explicitement exclues pour éviter des appels API inutiles.

4.4.3. Extraction de Version par Expression Régulière

Une fois l'OS identifié, la version doit être extraite pour interroger le bon cycle de vie. La stratégie Regex est hiérarchisée :

- Windows (prioritaire) : Les versions Windows sont exprimées sous deux formats. Les versions 'semi-annuelles' (1809, 20H2, 21H2) sont gérées par un pattern spécifique. Ce choix est justifié car Microsoft lie les dates EOL aux 'builds' et non aux versions commerciales.
- Fallback générique : Capture des patterns de type 'YYYY' (années) ou 'X.Y' (versions sémantiques) pour les distributions Linux, FreeBSD, etc.

4.5. Gestion des Données EOL (End of Life)

4.5.1. Architecture Hybride : Cache Local + API

L'accès aux données EOL:

- Lors d'une requête, le module vérifie d'abord le cache local (fichier JSON).
- Si la donnée est absente, il interroge l'API REST d'endoflife.date.
- La réponse API est persistée dans le cache pour les accès futurs.

Ce pattern est particulièrement adapté aux environnements d'entreprise où les réseaux de production sont segmentés (DMZ, VLAN sécurisé) et n'ont pas forcément accès à internet. L'administrateur peut initialiser le cache depuis une machine avec accès internet, puis opérer en mode 'offline' sur le réseau cible.

4.5.2. Format de Stockage : JSON

Le JSON est retenu comme format de cache pour plusieurs raisons : compatibilité native avec les réponses de l'API REST (pas de transformation), lisibilité humaine pour le débogage, et support natif en Python via le module `json` de la bibliothèque standard. La clé de premier niveau est le nom de l'OS (identifiant API), et la valeur est la liste complète des cycles de vie retournée par l'API.

4.5.3. Calcul du Statut EOL

La comparaison est arithmétique : delta = Date EOL - Date du jour. Trois états sont définis :

- CRITIQUE (delta < 0 jours) : Système hors support, aucun patch de sécurité disponible.
- AVERTISSEMENT (0 < delta < 365 jours) : Fin de support imminente, migration à planifier.
- OK (delta >= 365 jours) : Système sous support actif.

4.6. Export et Reporting

Deux fichiers CSV sont produits :

- `scan_results.csv` : Résultat brut du scan nmap avec IP, OS détecté, précision et ports ouverts. Ce fichier peut être édité manuellement pour corriger les détections incorrectes, avant d'être réinjecté dans le pipeline d'analyse.
- `eol_audit_report.csv` : Rapport de conformité final avec IP, OS, version, date EOL, et statut. Ce format est directement exploitable dans un tableur (Excel, LibreOffice) ou un outil ITSM comme GLPI.

L'encodage UTF-8 est explicitement forcé sur tous les fichiers pour assurer la compatibilité avec les caractères accentués présents dans les bannières de services.

4.7. Considérations de Sécurité et Limitations

4.7.1. Prérequis de Privilèges

Le fingerprinting OS de Nmap (-O) nécessite des raw sockets pour forger des paquets IP arbitraires. Sur Linux, cela implique une exécution en root ou avec la capability `CAP_NET_RAW`. Sur Windows, des droits Administrateur sont requis. Sans ces droits, le scan reste fonctionnel mais la détection OS sera indisponible.

4.7.2. Limitations de la Détection OS

Le fingerprinting OS n'est pas infaillible. Plusieurs facteurs peuvent dégrader la précision :

- Les pare-feux applicatifs qui normalisent les paquets TCP/IP masquent les empreintes.
- Les équipements réseau (switches managés, imprimantes) retournent des piles TCP/IP non référencées.
- Les machines virtuelles ou conteneurs Docker partagent le noyau de l'hôte, ce qui peut induire des faux positifs sur la version du kernel.

C'est pourquoi le script implémente une couche de 'Banner Grabbing' supplémentaire : les bannières de services (SSH, HTTP) révèlent souvent la distribution Linux de manière fiable, compensant ainsi les limitations du fingerprinting réseau pur.

4.7.3. Impact Réseau

Le mode -T4 avec détection de version génère un volume de trafic non négligeable sur de larges plages. Il est recommandé de limiter les scans aux heures creuses ou de réduire la plage scannée. L'option -Pn traite tous les hôtes comme actifs, ce qui peut générer du trafic vers des IPs non utilisées.

4.8. Prérequis et Déploiement

• Logiciels : Python 3.6 minimum, Nmap installé et accessible dans le PATH système
• Dépendance Python : `requests` (auto-installé au premier lancement)
• Réseau : Accès internet optionnel (requis uniquement pour la mise à jour du cache EOL)
• Privilèges : Root/Administrateur pour la détection OS complète

4.9. Référentiel de Conformité EOL

L'intégrité de l'audit repose sur la fiabilité des données de cycle de vie. Ce module utilise comme source de vérité le référentiel communautaire endoflife.date, qui agrège et normalise les politiques de support officielles des éditeurs (Microsoft Lifecycle Policy, Ubuntu Release Cycle, Red Hat Life Cycle, etc.).

Les dates de validité sont évaluées selon deux métriques standard :

EOL (End of Life) : Date d'arrêt de la commercialisation ou du développement actif.

EOS (End of Support) : Date critique d'arrêt des correctifs de sécurité (Security Maintenance).

Dans le cadre de cet audit, c'est la date EOS qui fait foi pour le calcul du risque. Toute date antérieure à la date du scan (aujourd'hui, 24 février 2026) classe l'actif en statut critique ("Obsolète"). Une marge de tolérance de 90 jours avant échéance déclenche un statut d'avertissement ("À planifier") pour anticiper les migrations conformément aux bonnes pratiques ITIL.