Rapport du projet python

Membres du groupe :

MAHOUNGOU Bryan

BERSON Maxence

1. Introduction

Ce rapport présente une analyse technique d'un outil Python conçu pour la sécurité et l'analyse réseau. L'objectif principal est de fournir des fonctionnalités pour :

- Analyser les logs d'authentification afin d'identifier des adresses IP suspectes.
- Effectuer des scans de ports sur des adresses IP cibles.
- Combiner les deux fonctionnalités pour automatiquement scanner les IPs identifiées comme suspectes.

Le projet est structuré en plusieurs modules Python distincts, chacun ayant une responsabilité spécifique.

2. Étapes Techniques - Découpage du Projet

Le projet est logiquement divisé en plusieurs modules, chacun gérant une tâche spécifique. Cela favorise la clarté du code, la réutilisation des composants et facilite le débogage.

2.1. `main.py` (Orchestrateur - Interface Utilisateur)

- **Rôle :** Point d'entrée principal de l'application. Il présente un menu interactif à l'utilisateur, gère les choix et orchestre l'appel des fonctions des autres modules en fonction de la sélection de l'utilisateur.
- **Fonctionnalités clés :**
    - Affichage du menu principal (`afficher_menu_principal`).
    - Gestion des entrées utilisateur avec validation et valeurs par défaut (`input_utilisateur`).
    - analyse de logs, scan de ports, ou combinaison des deux.

Extrait de code (`main.py`) : Structure principale du menu interactif

```
# main.py
# ... (imports)

def afficher_menu_principal():
    #Affichage menu principal
    print("\n**************************")
    print("    Menu interactif")
    print("**************************")
    print("1 - Analyser des logs ")
    print("2 - Scan de ports")
    print("3 - Analyser & scanner les IPs suspectes")
    print("4 - Quitter")
    print("**************************")

# ... (input_utilisateur, menu_analyse_log, menu_scan_ports, effectuer_scan)

def main():
    # Menu principal.
```

```
    while True: # Ajout de la boucle while True pour maintenir le menu
        afficher_menu_principal()
        choix = input_utilisateur("Votre choix", type_func=int)

        if choix == 1:
            # Analyser les logs
            menu_analyse_log()

        elif choix == 2:
            # Effectuer un scan de ports sur des IPs spécifiées manuellement
            menu_scan_ports()

        elif choix == 3:
            # Analyser les logs ET scanner les IPs suspectes
            print("\n--- Étape 1 : Analyse des logs ---")
            top_ips = menu_analyse_log()

            if top_ips:
                print("\n--- Étape 2 : Scan des IPs suspectes ---")
                print(f" Les IPs suspectes suivantes seront scannées : {',
'.join(top_ips)}")
                menu_scan_ports(ips_predefinies=top_ips)
            else:
                print(" Pas d'IPs suspectes trouvées dans les logs pour
lancer le scan.")

        elif choix == 4:
            print("Bye !")
            sys.exit(0)
        else:
            print("Choix invalide. Veuillez entrer un nombre entre 1 et 4.")

        input("\nAppuyez sur Entrée pour revenir au menu principal...")

if __name__ == "__main__":
    main()
```

2.2. `log_parser.py` (Analyseur de Logs)

- **Rôle :** Ce module est dédié à l'extraction d'informations pertinentes à partir de fichiers de logs système, spécifiquement les échecs d'authentification SSH.
- **Fonctionnalités clés :**
    - Lecture d'un fichier de log (`auth.log`).
    - Utilisation d'expressions régulières (`re`) pour identifier les lignes d'échec d'authentification (`Failed password`).
    - Extraction des adresses IP associées à ces échecs.
    - Comptage des occurrences de chaque IP suspecte (`collections.defaultdict`).
    - Gestion des erreurs de fichier (`FileNotFoundError`).

Extrait de code (`log_parser.py`) : Logique d'analyse des échecs d'authentification

```
# log_parser.py
import re
from collections import defaultdict

def analyser_log_authentification(chemin_fichier_log):
    """
    Analyse un fichier de log d'authentification (auth.log) pour extraire
    les adresses IP associées aux échecs de connexion.
    Retourne un dictionnaire où les clés sont les IPs suspectes et les
valeurs
    sont le nombre d'occurrences.
    """
    occurrences_ip = defaultdict(int)
    # Regex pour capturer les lignes "Failed password" et extraire l'adresse
IP
    regex = r"Failed password.*from (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"

    try:
        with open(chemin_fichier_log, 'r') as f:
            for ligne in f:
                if "Failed password" in ligne:
                    correspondance = re.search(regex, ligne)
                    if correspondance:
                        adresse_ip = correspondance.group(1)
                        occurrences_ip[adresse_ip] += 1
    except FileNotFoundError:
        print(f"Erreur : Le fichier de log '{chemin_fichier_log}' est
introuvable.")
        return None
    except Exception as e:
        print(f"Une erreur s'est produite lors de la lecture du fichier de
log : {e}")
        return None

    return dict(occurrences_ip)
```

2.3. `data_analyzer.py` (Analyse et Visualisation de Données)

- **Rôle :** Ce module prend les données d'IPs et leurs occurrences fournies par le `log_parser.py` et les traite pour identifier les IPs les plus actives, puis visualise et exporte ces données.
- **Fonctionnalités clés :**
    - Utilisation de la bibliothèque `pandas` pour la manipulation de données (création de DataFrame, tri des occurrences).
    - Identification du "Top 5" des adresses IP les plus suspectes.
    - Génération de visualisations (graphiques à barres) des Top IPs via `matplotlib.pyplot`.
    - Enregistrement du graphique au format PNG.
    - Exportation des données triées au format CSV.
    - Gestion de la création du répertoire de sortie (`results`).

Extrait de code (`data_analyzer.py`) : Traitement Pandas et génération de graphique

```
# data_analyzer.py
import pandas as pd
import matplotlib.pyplot as plt
import os

def analyser_et_visualiser_ips(donnees_ip, repertoire_sortie="results"):
    #Conversion des données en DataFrame + trie des IPs les plus actives +
graphique/export.

    if not donnees_ip:
        print("Aucune donnée IP à analyser ou visualiser.")
        return None, None

    df_ips = pd.DataFrame(list(donnees_ip.items()), columns=['Adresse_IP',
'Occurrences'])

    #(Top 5)
    df_trie = df_ips.sort_values(by='Occurrences', ascending=False)
    top_ips = df_trie.head(5)

    print("\nTop 5 IPs suspectes:")
    print(top_ips)

    # Creation répertoire
    if not os.path.exists(repertoire_sortie):
        os.makedirs(repertoire_sortie)

    # Générer un graphique à barres
    plt.figure(figsize=(10, 6))
    plt.bar(top_ips['Adresse_IP'], top_ips['Occurrences'], color='skyblue')
    plt.xlabel('Adresse IP')
    plt.ylabel('Nombre d\'occurrences')
```

```
    plt.title('Top 5 des adresses IP suspectes (Tentatives de connexion
échouées)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

    # Enregistrer le graphique
    chemin_graphique = os.path.join(repertoire_sortie, 'top_ips_chart.png')
    plt.savefig(chemin_graphique)
    print(f"\nGraphique enregistré sous : {chemin_graphique}")

    # Exporter les résultats au format CSV
    chemin_csv = os.path.join(repertoire_sortie, 'suspect_ips.csv')
    df_trie.to_csv(chemin_csv, index=False)
    print(f"Données exportées au format CSV sous : {chemin_csv}")

    return top_ips['Adresse_IP'].tolist(), chemin_graphique
```

2.4. `network_scanner.py` (Scanner de Ports)

- **Rôle :** Ce module est responsable de la réalisation des scans de ports sur des adresses IP spécifiées.
- **Fonctionnalités clés :**
    - Fonction `scanner_port` : Tente d'établir une connexion TCP à un port spécifique en utilisant le module `socket`. Gère les timeouts et retourne l'état "ouvert" ou "fermé".
    - Fonction `scan_mono_thread` : Exécute des scans de ports de manière séquentielle (un port après l'autre).
    - Fonction `scan_multi_thread` : Implémente le scan multi-threadé pour des performances améliorées, en utilisant `concurrent.futures.ThreadPoolExecutor` pour gérer un pool de threads.
    - Option de mode verbeux pour afficher les ports fermés.

Extrait de code (`network_scanner.py`) : Fonction `scanner_port` et structure de `scan_multi_thread`

```
# network_scanner.py
import socket # Module socket importé
import threading
from concurrent.futures import ThreadPoolExecutor

def scanner_port(adresse_ip, port, delai_attente=1, verbeux=False):
    """
    Tente une connexion sur un port spécifique d'une adresse IP.
    Retourne True si le port est ouvert, False sinon.
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(delai_attente)
    resultat = sock.connect_ex((adresse_ip, port))
    sock.close()
```

```
    if resultat == 0:
        if verbeux:
            print(f"Port {port} ouvert sur {adresse_ip}")
        return True
    else:
        if verbeux:
            print(f"Port {port} fermé/filtré sur {adresse_ip} (Erreur:
{resultat})")
        return False

# ... (scan_mono_thread)

def scan_multi_thread(adresse_ip, ports, delai_attente=1, max_threads=20,
verbeux=False):
    """
    Effectue un scan de ports sur une adresse IP donnée en mode multi-thread.
    Retourne un dictionnaire des ports ouverts.
    """
    print(f"\nLancement du scan multi-thread sur {adresse_ip} avec
{max_threads} threads...")
    ports_ouverts = {}

    with ThreadPoolExecutor(max_workers=max_threads) as executeur:
        # Soumet chaque scan de port comme une tâche distincte à un thread
        futurs = {executeur.submit(scanner_port, adresse_ip, port,
delai_attente, verbeux): port for port in ports}
        for futur in futurs: # Itère sur les objets Future (représentant les
tâches)
            port = futurs[futur]
            try:
                if futur.result(): # Bloque jusqu'à ce que la tâche soit
terminée et récupère le résultat
                    ports_ouverts[port] = "Ouvert"
            except Exception as exc:
                if verbeux:
                    print(f"Port {port} a généré une exception: {exc}")
    return ports_ouverts
```

3. Choix Techniques Effectués

Plusieurs choix techniques ont été faits pour la conception et l'implémentation de cet outil :

- **Modularité du Code :** Le découpage en modules distincts (`log_parser.py`, `data_analyzer.py`, `network_scanner.py`, `main.py`) est un choix architectural.
    - **Avantages :** Améliore la lisibilité, la maintenabilité, la réutilisabilité du code (chaque module peut être utilisé indépendamment si nécessaire) et facilite le travail en équipe sur des fonctionnalités spécifiques.
    - **Alternative :** Un seul fichier, ce qui aurait rendu le projet plus difficile à gérer et à étendre.

- **Utilisation du Module `socket` pour le Scan de Ports :**
    - **Justification :** Le module `socket` est la brique de base pour la communication réseau en Python. Il offre un contrôle fin sur les connexions TCP/IP, ce qui est essentiel pour un scanner de ports.
    - **Méthode `connect_ex()` :** Ce choix est judicieux car `connect_ex()` ne lève pas d'exception en cas d'échec de connexion (par exemple, port fermé), mais retourne un code d'erreur (0 pour succès), permettant une gestion plus propre du flux du programme sans avoir besoin de nombreux blocs `try-except` pour chaque tentative de connexion.
    - **`settimeout()` :** Essentiel pour éviter que le scanner ne reste bloqué indéfiniment sur un port inaccessible, améliorant ainsi la robustesse et la rapidité du scan.
- **Multi-threading avec `concurrent.futures.ThreadPoolExecutor` :**
    - **Justification :** Le scan de ports est une opération I/O-bound (liée aux entrées/sorties), où le programme passe beaucoup de temps à attendre les réponses du réseau. Le multi-threading permet d'exécuter plusieurs tentatives de connexion simultanément, réduisant considérablement le temps total de scan, surtout pour un grand nombre de ports ou d'IPs.
    - **`ThreadPoolExecutor` vs. `threading` direct :** `ThreadPoolExecutor` offre une abstraction de haut niveau pour gérer les threads. Il gère automatiquement la création, la réutilisation et la destruction des threads dans un pool, simplifiant grandement la gestion des tâches concurrentes par rapport à la création et au démarrage manuel de threads avec le module `threading` pur.
- **Traitement des Logs avec `re` (Expressions Régulières) et `collections.defaultdict` :**
    - **`re` :** Les expressions régulières sont l'outil standard et puissant pour l'analyse de texte structuré ou semi-structuré comme les fichiers de logs. Elles permettent de cibler précisément les motifs (`Failed password.*from (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})`) et d'extraire les données souhaitées (l'adresse IP).
    - **`defaultdict(int)` :** `defaultdict` est une classe pratique de la collection `collections` qui simplifie le comptage d'occurrences. Au lieu de vérifier `if ip in occurrences_ip: occurrences_ip[ip] += 1 else: occurrences_ip[ip] = 1`, on peut simplement faire `occurrences_ip[ip] += 1`.
- **Analyse de Données avec `pandas` :**
    - **Justification :** `pandas` est la bibliothèque de référence pour la manipulation et l'analyse de données tabulaires en Python. Elle facilite le chargement, la transformation, le tri et la sélection des données. L'utilisation d'un `DataFrame` pour les IPs et leurs occurrences rend les opérations comme le

tri (`sort_values`) et la sélection du Top 5 (`head(5)`) très concises et efficaces.

- **Visualisation avec `matplotlib.pyplot` :**
    - **Justification :** `matplotlib` est la bibliothèque de traçage la plus populaire en Python, offrant une grande flexibilité pour créer des graphiques de qualité. Un graphique à barres est un choix approprié pour visualiser les occurrences d'IPs, rendant l'information rapidement compréhensible.
- **Exportation des Résultats (CSV et HTML) :**
    - **CSV :** L'exportation en CSV (`df_trie.to_csv`) est un format universellement reconnu pour l'échange de données, permettant d'analyser les résultats plus en détail avec d'autres outils (tableurs, bases de données).
    - **HTML :** L'exportation HTML des résultats de scan (`main.py`) est une excellente idée pour un rapport visuellement accessible et facile à partager dans un navigateur web.
- **Interface Utilisateur en Ligne de Commande (CLI) Interactive :**
    - **`input_utilisateur` :** Cette fonction personnalisée permet une interaction utilisateur robuste, avec des options de valeurs par défaut et de validation de type, ce qui améliore l'expérience utilisateur et la robustesse du programme face aux entrées invalides.

4. Exemples de Résultats

Pour illustrer les capacités de l'outil, voici des exemples de résultats obtenus en exécutant le programme.

(Note : Les résultats réels dépendent du contenu de `auth.log` et des IPs/ports scannés.)

4.1. Analyse des Logs (Option 1 ou 3 du menu principal)

Après avoir exécuté l'analyse des logs sur un fichier `auth.log` contenant des tentatives de connexion échouées, le `log_parser.py` collecte les IPs, et `data_analyzer.py` traite et affiche le Top 5.

Exemple de sortie console après analyse des logs :

```
Analyse du fichier de logs : auth.log
Analyse des logs terminée. Résultats:

Top 5 IPs suspectes:
      Adresse_IP  Occurrences
0  192.168.1.12            8
1    203.0.113.5            6
2  198.51.100.23            4
```

```
3     10.0.0.10            3
4   172.16.0.25            2
```

Graphique enregistré sous : results/top_ips_chart.png
Données exportées au format CSV sous : results/suspect_ips.csv

Exemple de Visualisation (`top_ips_chart.png` - généré dans le répertoire `results/`) :

4.2. Scan de Ports (Option 2 ou 3 du menu principal)

Si on choisit l'option de scan de ports, que ce soit manuellement ou avec les IPs suspectes, l'outil affichera la progression et un résumé final.

Exemple de sortie console pour un scan :

```
Lancement du scan multi-thread sur 192.168.1.1 avec 20 threads...
Ports à scanner : 22, 80, 443, 21, 23, 25, 110, 3389
Mode multi-thread : Oui
Mode verbeux : Non
Délai d'attente par port : 1.0 secondes
Port 22 fermé/filtré sur 192.168.1.1 (Erreur: 10061)
Port 80 ouvert sur 192.168.1.1
Port 443 ouvert sur 192.168.1.1
Port 21 fermé/filtré sur 192.168.1.1 (Erreur: 10061)
...

**************************
    RÉSUMÉ DES PORTS OUVERTS
**************************
IP: 192.168.1.1
  - Port 80: Ouvert
  - Port 443: Ouvert
  - Port 3389: Ouvert
**************************

Résultats du scan de ports exportés en HTML :
results/scan_resultats.html
```

Exemple de Fichier `scan_resultats.html` (généré dans le répertoire `results/`) :

```
<!DOCTYPE html>
<html>
<head><title>Résultats du Scan de Ports</title>
<style>table, th, td {border: 1px solid black; border-collapse: collapse;}
th, td {padding: 8px; text-align: left;} th {background-color:
#f2f2f2;}</style>
</head>
<body>
<h1>Résultats du Scan de Ports</h1>
<table>
<tr><th>Adresse IP</th><th>Ports Ouverts</th></tr>
<tr><td>192.168.1.1</td><td>80, 443, 3389</td></tr>
<tr><td>203.0.113.5</td><td>22</td></tr>
</table>
</body>
</html>
```

5. Conclusion

Ce projet Python constitue un outil de sécurité réseau complet et modulaire. En découpant les fonctionnalités en modules spécialisés, il offre une structure claire et facile à comprendre. Les choix techniques, notamment l'utilisation de `socket` pour le scan, `concurrent.futures` pour le multi-threading, `re` pour le parsing de logs, `pandas` pour l'analyse de données et `matplotlib` pour la visualisation, sont pertinents et optimisés pour les tâches à accomplir. La capacité à analyser les logs et à utiliser ces informations pour initier des scans de ports ciblés représente une fonctionnalité puissante pour l'identification proactive de menaces potentielles sur un réseau.