جامعة هواري بومدين
للعلوم والتكنولوجيا
USTHB

RAPPORT DU PROJET DE TP

Anti-Rootkit

FAIT PAR :

CHALAL LYNA

MEROUANI ITHRI

ELKHECHINE ABDELKADER

Année : 2024/2025

M1 SSI

Table des matières :

Introduction générale

Dans un monde de plus en plus dépendant des technologies numériques, la sécurité informatique est devenue une priorité majeure. Les menaces évoluent constamment, et parmi elles, les rootkits se démarquent par leur capacité à rester cachés tout en compromettant profondément les systèmes informatiques.

Un rootkit est un logiciel malveillant spécialement conçu pour s'infiltrer dans un système et y maintenir un accès non autorisé, tout en dissimulant ses activités. Ces outils de cyberattaque sont particulièrement dangereux car ils permettent aux attaquants de contrôler un système de manière invisible, mettant en péril les données sensibles, les ressources système et l'intégrité de l'environnement informatique.

Face à ces menaces, les anti-rootkits jouent un rôle crucial. Contrairement aux antivirus classiques qui se concentrent principalement sur les fichiers malveillants, les anti-rootkits visent à détecter et supprimer ces programmes furtifs en analysant les couches du système, y compris le noyau et la mémoire.

Ce projet a pour objectif de développer un outil d'analyse anti-rootkit capable de détecter différents types de rootkits, qu'ils opèrent au niveau de l'utilisateur, des librairies systèmes, du noyau, etc. En combinant plusieurs méthodes de détection – signatures, comportement, analyse mémoire, et vérification d'intégrité – cet outil offre une approche globale pour renforcer la sécurité des systèmes.

Dans ce rapport, nous présenterons les leurs types de rootkits, et les principes des anti-rootkits, avant de détailler les méthodes utilisées pour leur détection.

C'est quoi un rootkit ?

Un rootkit est un logiciel malveillant conçu pour obtenir et maintenir un accès non autorisé à un système informatique tout en restant indétectable. Il tire son nom de "root" (l'utilisateur administrateur avec les droits les plus élevés sous Unix/Linux) et de "kit" (un ensemble d'outils).

Les rootkits sont utilisés pour exécuter diverses activités malveillantes telles que l'espionnage, le vol de données, ou encore le contrôle total du système infecté. Leur force réside dans leur capacité à rester invisibles, en masquant leurs fichiers, leurs processus, ou leur présence dans la mémoire.

Le principal danger des rootkits réside dans leur furtivité et leur capacité à contourner les outils de sécurité traditionnels, rendant leur détection et leur suppression complexes.

Quelques types de rootkits

Les rootkits se classent selon leur emplacement dans le système et leur niveau d'accès. Voici principaux types :

- **Rootkits en mode utilisateur (User/Application Mode) :** Ils agissent dans l'espace utilisateur, affectant les applications et les processus visibles par l'utilisateur. Ils interceptent les appels système et manipulent les bibliothèques partagées pour cacher des fichiers, des processus ou des connexions réseau.
- **Rootkits en mode noyau (Kernel Mode) :** Ils fonctionnent au niveau du noyau, la partie centrale du système d'exploitation, avec les privilèges les plus élevés. Ils modifient les structures ou fonctions du noyau pour cacher leur présence et permettre l'exécution de code malveillant. Ce type de rootkit est particulièrement dangereux car il contrôle entièrement le système.
- **Rootkits de démarrage (Bootkits) :** Ils infectent le chargeur de démarrage ou la séquence de démarrage pour s'exécuter avant le système d'exploitation. Ils modifient des composants critiques comme le Master Boot Record (MBR) pour rester actifs même après une réinstallation de l'OS.
- **Rootkits mémoire ou mappés (Memory-Based Rootkits)** : Ces rootkits opèrent exclusivement en mémoire vive (RAM), ce qui les rend très difficiles à détecter, car ils ne laissent aucune trace sur le disque. Ils injectent du code directement dans les processus en cours d'exécution ou exploitent des failles pour s'exécuter uniquement en mémoire.
    - N'écrivent pas de fichiers sur le disque, évitant ainsi d'être détectés par des analyses basées sur des signatures.
    - Utilisent des techniques comme l'injection de code dans un processus ou la manipulation de la table des appels système (System Call Table).
    - Volatiles : disparaissent lorsque le système redémarre.
- **Rootkits au niveau des bibliothèques (Library-Level Rootkits) :** Ces rootkits modifient ou remplacent les bibliothèques partagées utilisées par les applications. Les bibliothèques partagées (comme les libc sous Linux, Dynamic Link Libraries (DLL)

pour Windows) fournissent des fonctions couramment utilisées par les programmes, et un rootkit peut les exploiter pour manipuler les comportements des applications.

- Fonctionnent en injectant ou modifiant des bibliothèques dynamiques.
- Utilisent des techniques comme le hooking (redirection de fonctions système) pour intercepter des appels système et retourner de fausses informations.

Ces types de rootkits représentent des niveaux variés de menace et de complexité. Leur détection nécessite des outils capables d'analyser les couches profondes du système et de contourner leurs mécanismes de dissimulation.

C'est quoi un Anti-Rootkit ?

Un anti-rootkit est un logiciel conçu pour détecter et prévenir les rootkits, qui sont des programmes malveillants permettant à un attaquant de prendre le contrôle d'un système informatique tout en restant caché. L'anti-rootkit analyse les composants du système, tels que les processus, fichiers et registres, pour identifier des comportements suspects associés à un rootkit.

Les méthodes que nous avons utilisées pour détecter les rootkits sur Windows

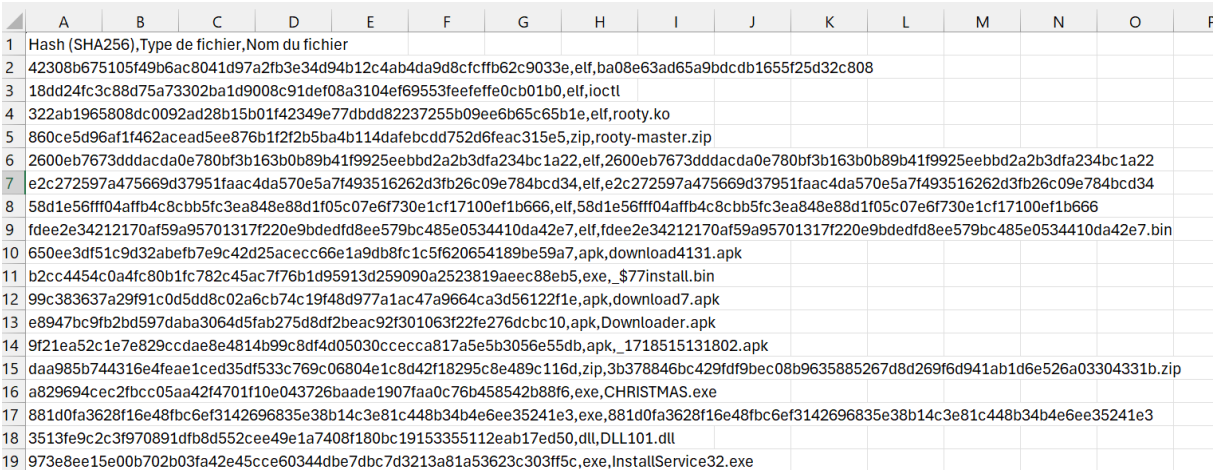
1. Détection basée sur les signatures de Rootkits connus

La détection de rootkits par signature repose sur l'idée de comparer les fichiers suspects avec une base de données contenant les empreintes uniques (hashes) de rootkits connus. Chaque fichier a un hash, calculé à l'aide d'un algorithme comme SHA-256, qui sert d'identifiant unique. Lorsqu'un fichier est analysé, son hash est calculé et comparé à ceux de la base de données. Si une correspondance est trouvée, le fichier est identifié comme étant un rootkit. Cette méthode est rapide et efficace pour détecter des rootkits connus, mais elle présente des limitations, notamment son incapacité à détecter les nouvelles menaces ou les rootkits polymorphiques qui modifient leur apparence.

Les étapes principales du processus (fichier : *signature_verification.py*) sont les suivantes :

- Charger les hashes connus des rootkits depuis le fichier CSV (*rootkit_data.csv*).
- Calculer le hash SHA-256 du fichier choisi par l'utilisateur.

- Comparer ce hash avec ceux de la base de données pour déterminer si le fichier est un rootkit.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Hash (SHA256),Type de fichier,Nom du fichier															
2	42308b675105f49b6ac8041d97a2fb3e34d94b12c4ab4da9d8cfcffb62c9033e,elf,ba08e63ad65a9bdcdb1655f25d32c808															
3	18dd24fc3c88d75a73302ba1d9008c91def08a3104ef69553feefeffe0cb01b0,elf,ioctl															
4	322ab1965808dc0092ad28b15b01f42349e77dbdd82237255b09ee6b65c65b1e,elf,rooty.ko															
5	860ce5d96af1f462acead5ee876b1f2f2b5ba4b114dafebcdd752d6feac315e5,zip,rooty-master.zip															
6	2600eb7673dddacda0e780bf3b163b0b89b41f9925eebbd2a2b3dfa234bc1a22,elf,2600eb7673dddacda0e780bf3b163b0b89b41f9925eebbd2a2b3dfa234bc1a22															
7	e2c272597a475669d37951faac4da570e5a7f493516262d3fb26c09e784bcd34,elf,e2c272597a475669d37951faac4da570e5a7f493516262d3fb26c09e784bcd34															
8	58d1e56fff04affb4c8cbb5fc3ea848e88d1f05c07e6f730e1cf17100ef1b666,elf,58d1e56fff04affb4c8cbb5fc3ea848e88d1f05c07e6f730e1cf17100ef1b666															
9	fdee2e34212170af59a95701317f220e9bdedfd8ee579bc485e0534410da42e7,elf,fdee2e34212170af59a95701317f220e9bdedfd8ee579bc485e0534410da42e7.bin															
10	650ee3df51c9d32abefb7e9c42d25acecc66e1a9db8fc1c5f620654189be59a7,apk,download4131.apk															
11	b2cc4454c0a4fc80b1fc782c45ac7f76b1d95913d259090a2523819aeec88eb5,exe,_$77install.bin															
12	99c383637a29f91c0d5dd8c02a6cb74c19f48d977a1ac47a9664ca3d56122f1e,apk,download7.apk															
13	e8947bc9fb2bd597daba3064d5fab275d8df2beac92f301063f22fe276dcbc10,apk,Downloader.apk															
14	9f21ea52c1e7e829ccdae8e4814b99c8df4d05030ccecca817a5e5b3056e55db,apk,_1718515131802.apk															
15	daa985b744316e4feae1ced35df533c769c06804e1c8d42f18295c8e489c116d,zip,3b378846bc429fdf9bec08b9635885267d8d269f6d941ab1d6e526a03304331b.zip															
16	a829694cec2fbcc05aa42f4701f10e043726baade1907faa0c76b458542b88f6,exe,CHRISTMAS.exe															
17	881d0fa3628f16e48fbc6ef3142696835e38b14c3e81c448b34b4e6ee35241e3,exe,881d0fa3628f16e48fbc6ef3142696835e38b14c3e81c448b34b4e6ee35241e3															
18	3513fe9c2c3f970891dfb8d552cee49e1a7408f180bc19153355112eab17ed50,dll,DLL101.dll															
19	973e8ee15e00b702b03fa42e45cce60344dbe7dbc7d3213a81a53623c303ff5c,exe,InstallService32.exe															

2. *Détection et Vérification de l'Intégrité du Système*

Le premier script *create_sys_sauvegarde.py* crée une base de référence en sauvegardant les hachages SHA-256 des fichiers, dossiers et clés de registre critiques du système Windows dans des fichiers CSV. Son but principal est de détecter des modifications non autorisées (potentiellement dues à des rootkits) dans :

- Les fichiers critiques du système (par ex. ntoskrnl.exe, winlogon.exe).
- Les dossiers critiques (par ex. C:\Windows\System32\drivers).
- Les clés de registre critiques (par ex. SYSTEM\CurrentControlSet\Services).

Le deuxième script *verify_sys_folder.py* vérifier l'intégrité des fichiers, dossiers et clés de registre critiques en comparant leurs hachages actuels avec ceux sauvegardés pour aider à détecter des altérations malveillantes.

Fonctionnement :

- Chargement des hachages de référence : Lit les CSV générés par le premier script.
- Comparaison des hachages : Détecte les modifications (hachages différents), suppressions (fichiers/clés manquants) ou ajouts (nouveaux éléments).
- Classification des résultats : Signale les fichiers/clés modifiés, manquants ou ajoutés.

3. Détection des hooks malveillants dans la table IAT d'un processus

La table Import Address Table (IAT) est utilisée pour lier dynamiquement les fonctions d'une bibliothèque partagée (comme une DLL) avec le programme. Les rootkits de niveau utilisateur peuvent détourner ces adresses pour les rediriger vers des fonctions malveillantes, compromettant ainsi le fonctionnement du programme.

Dans notre programme, la fonction *int DetectIATHooks(DWORD pid)* ouvre le processus spécifié en mode lecture, parcourt les modules chargés (DLLs et exécutables), et analyse les sections de la table IAT de chaque module. L'objectif est de vérifier si des adresses pointent en dehors du module auquel elles appartiennent, ce qui pourrait indiquer la présence de hooks malveillants.

Nous avons utilisé les fonctions API Windows telles que EnumProcessModules pour lister les modules et GetModuleInformation pour obtenir des informations détaillées sur chacun d'eux. Ces fonctions facilitent l'identification des anomalies dans les adresses importées par les modules.

4. Surveillance des Contextes des Threads d'un Processus

Les rootkits de niveau utilisateur peuvent injecter du code directement dans la mémoire d'un processus et exécuter ce code via des threads créés ou détournés. Une telle activité peut permettre à un rootkit de s'exécuter de manière furtive et d'échapper à la détection classique.

La fonction *int MonitorThreadContexts(DWORD pid)* permet d'identifier ces threads suspects en effectuant les étapes suivantes :

1. Lister les threads du processus spécifié : Les threads actifs associés au processus sont identifiés.
2. Récupérer le contexte d'exécution de chaque thread : Les informations contextuelles (comme l'adresse actuelle d'exécution des instructions) sont extraites.
3. Vérifier la légitimité des adresses d'instructions : L'adresse d'instruction actuelle est comparée aux plages mémoire des modules chargés (DLLs ou exécutables légitimes).

Si un thread exécute du code situé en dehors des modules légitimes chargés, un score est incrémenté. Ce score permet d'évaluer la probabilité qu'un ou plusieurs threads soient utilisés pour des activités malveillantes.

La détection repose sur l'utilisation des API Windows suivantes :

- Thread32First et Thread32Next : Pour parcourir les threads d'un processus.
- GetThreadContext : Pour obtenir le contexte d'exécution d'un thread spécifique.

Cette approche fournit une méthode robuste pour repérer les threads exécutant des instructions non autorisées, un indicateur clé de la présence de rootkits.

5. Détection des Hooks dans la Table d'Adresses d'Exportation

La table Export Address Table (EAT) est utilisée pour exporter les fonctions d'une DLL. Les rootkits peuvent exploiter cette table en redirigeant les adresses d'exportation vers du code malveillant, compromettant ainsi la sécurité du système.

La fonction *int DetectEATHooks(HMODULE dll)* effectue les étapes suivantes :

1. Analyse la structure IMAGE_EXPORT_DIRECTORY d'une DLL donnée.
2. Vérifie si les adresses d'exportation pointent en dehors des limites de la DLL.
3. Incrémente un score pour chaque anomalie détectée.

Cette méthode aide à identifier les DLLs manipulées par des hooks malveillants.

6. Détection des Injections de DLL

L'injection de DLL est une technique couramment utilisée par les rootkits pour manipuler les processus à des fins malveillantes. Cette fonction vise à identifier les DLLs injectées dans un processus spécifique.

La fonction : *int DetectDLLInjection(DWORD pid)* effectue les étapes suivantes :

1. Lister les modules (DLLs) chargés par le processus.
2. Vérifier si les adresses des DLLs sont légitimes et appartiennent réellement aux modules chargés.
3. Incrémenter un score lorsqu'une DLL injectée est détectée.

Cette fonction est cruciale pour détecter les processus compromis par des injections de DLL.

7. Identification des Régions Mémoire Suspectes dans une DLL

Les rootkits peuvent utiliser des zones mémoire privées associées à une DLL pour exécuter du code malveillant. Cette fonction cherche à identifier ces régions mémoire suspectes.

La fonction *int CheckLibraryMemoryRegions(HMODULE dll*) fonctionne ainsi :

1. Parcourt les régions mémoire associées à une DLL en utilisant l'API VirtualQuery.
2. Inspecte les propriétés des régions mémoire (par exemple, MEM_PRIVATE, MEM_COMMIT).
3. Incrémente un score lorsqu'une zone mémoire privée et engagée est détectée.

Cette méthode aide à repérer les comportements anormaux dans les bibliothèques partagées.

8. Analyse de la Mémoire Système

Les rootkits exploitent souvent de grandes régions mémoire privées pour stocker du code ou des données malveillantes. Cette fonction analyse la mémoire système pour détecter de telles zones suspectes.

La fonction *void ScanSystemMemory()* inclue les étapes suivantes :

1. Parcourir toutes les régions mémoire du système à l'aide de l'API VirtualQuery.
2. Identifier les régions mémoire engagées en mode privé (MEM_PRIVATE).
3. Signaler les zones suspectes, notamment celles ayant une taille inhabituelle.

Cette approche est essentielle pour une surveillance systématique de la mémoire.

9. Validation des Régions Mémoire Chargées en tant qu'Images

Les rootkits peuvent charger des sections d'images illégitimes pour dissimuler leur code. Cette fonction vérifie la légitimité des régions mémoire chargées en tant qu'images.

Voici les étapes réalisées par la fonction *void ValidateMemoryRegions()* :

1. Parcourt les régions mémoire associées à des images (MEM_IMAGE).
2. Utilise l'API VirtualQuery pour examiner leurs propriétés.
3. Valide les régions en fonction de leurs types (MEM_IMAGE).
4. Signale les anomalies dans la taille ou la structure des images.

Cette validation permet de détecter les régions mémoire illégitimes associées à des activités malveillantes.

Les méthodes que nous avons utilisées pour détecter les rootkits sur Linux

1. Détection basée sur les signatures de Rootkits connus

La méthode de détection par signature est comme expliqué précédemment pour Windows.

2. Analyse des Régions Mémoire Suspectes d'un Processus

La fonction *CheckMaps(pid_t pid, std::vector<std::string>& suspiciousRegions)* analyse le fichier /proc/<pid>/maps d'un processus pour identifier des régions mémoire potentiellement suspectes. Les régions de mémoire anonymes ([anon]) ou celles de type heap ([heap]) sont souvent utilisées pour injecter du code malveillant, car elles ne sont pas associées à des fichiers ou bibliothèques externes.

Le processus suit les étapes suivantes :

1. Recherche dans le fichier /proc/<pid>/maps des régions mémoire de type [anon] ou [heap].
2. Ajoute les lignes correspondantes à un vecteur de régions suspectes.
3. Appelée dans la fonction ScanProcesses pour analyser les régions mémoire des processus actifs.

Cela permet de repérer des régions mémoire potentiellement exploitées par des rootkits ou des malwares.

3. Détection du Code Injecté dans un Processus

La fonction *DetectInjectedCode(pid_t pid)* analyse le fichier mémoire /proc/<pid>/mem d'un processus pour rechercher des indices d'injection de code malveillant. Un des signes typiques d'injection de code est la présence de l'instruction de point d'arrêt (\xcc), fréquemment utilisée dans le débogage ou par des rootkits pour marquer le début du code injecté.

Le processus suit les étapes suivantes :

1. Parcourt la mémoire du processus à la recherche du caractère spécial \xcc, qui indique une instruction de point d'arrêt.

2. Si une telle instruction est détectée, la fonction retourne true, ce qui suggère la présence de code malveillant injecté dans le processus.

Cela permet d'identifier les processus compromis par l'injection de code.

4. Scan des Processus du Système

La fonction *ScanProcesses()* parcourt tous les processus présents dans le répertoire /proc et effectue une série de vérifications pour détecter les processus potentiellement compromis. Elle effectue les opérations suivantes pour chaque processus :

1. Identifie le chemin de l'exécutable du processus à partir de /proc/<pid>/exe.
2. Analyse les régions mémoire du processus à l'aide de la fonction CheckMaps.
3. Vérifie la présence de code injecté à l'aide de la fonction DetectInjectedCode.
4. Affiche les processus suspects avec leurs informations pertinentes (nom, PID, et régions mémoire suspectes).

Cela permet de dresser une liste de processus potentiellement compromis par des rootkits ou des injections de code.

5. Analyse de la Mémoire du Processus Courant

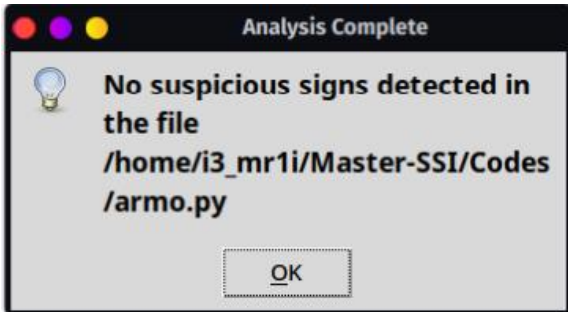
La fonction *ScanMemory()* analyse la mémoire du processus courant en lisant le fichier /proc/self/maps, qui contient les informations sur les régions mémoire du processus en cours d'exécution. Elle détecte les régions suspectes comme celles de type [anon], [heap], [stack] ou [vdso], qui peuvent être exploitées par des rootkits.

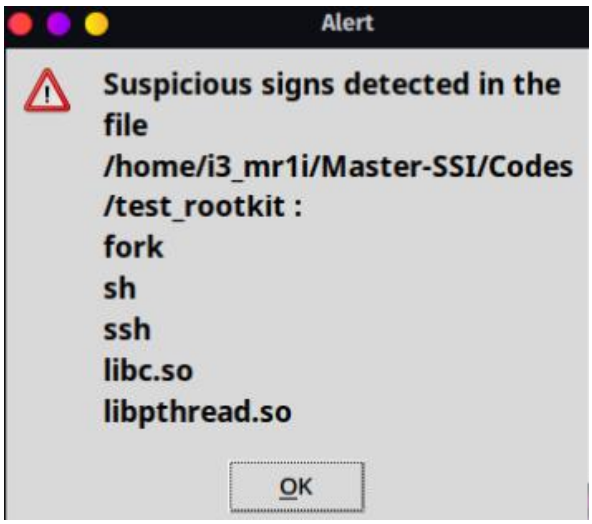
Les étapes de la fonction sont :

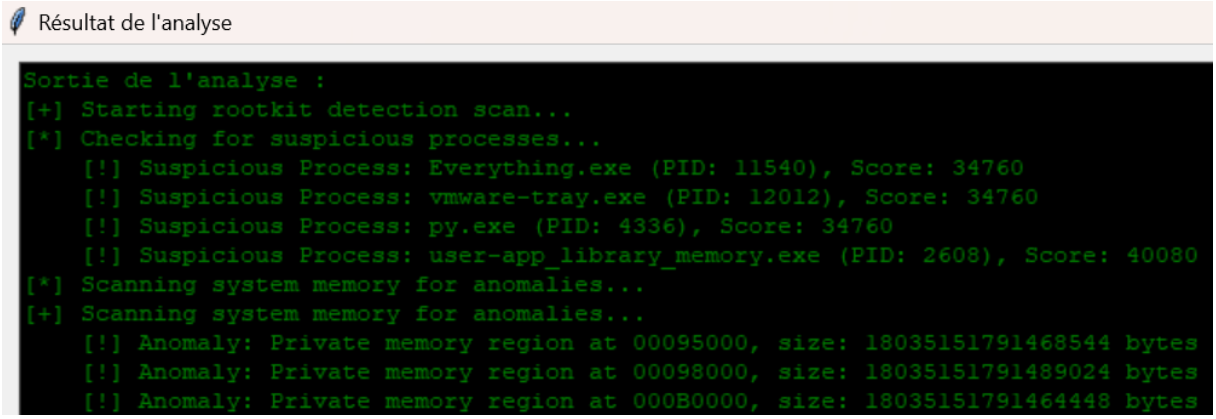
1. Lit le fichier /proc/self/maps pour identifier les régions mémoire de type [anon], [heap], [stack], ou [vdso].
2. Si des régions suspectes sont détectées, leurs détails (nom et informations associées) sont affichés.
3. Si aucune région suspecte n'est trouvée, un message indique qu'aucune anomalie n'a été détectée.

Cela permet de surveiller les régions mémoire suspectes dans le processus courant pour identifier des injections ou manipulations malveillantes de mémoire.

Présentation de l'interface graphique







Conclusion générale

Dans un contexte de cybermenaces de plus en plus sophistiquées, les rootkits représentent des dangers majeurs pour l'intégrité des systèmes informatiques. Leur capacité à rester dissimulés tout en permettant un contrôle non autorisé des systèmes les rend particulièrement difficiles à détecter et à éradiquer. Ce projet a permis de mettre en évidence l'importance des outils anti-rootkits qui, grâce à des méthodes de détection avancées, telles que l'analyse des hooks, la surveillance des processus et l'examen des régions mémoire suspectes, offrent une défense proactive contre ces menaces invisibles.

En combinant plusieurs approches de détection adaptées aux environnements Windows et Linux, nous avons développé un outil qui permet d'identifier efficacement les rootkits et de restaurer la sécurité des systèmes affectés. La diversité des méthodes utilisées montre l'importance d'une approche multi-niveaux pour renforcer la sécurité informatique, car chaque méthode a sa propre spécificité et efficacité.

Face à l'évolution constante des rootkits, il est essentiel de poursuivre la recherche et l'amélioration des techniques de détection afin de rester en avance sur les cybercriminels. Les anti-rootkits représentent un pilier essentiel dans la défense des systèmes informatiques, et leur rôle ne cessera de croître à mesure que les menaces deviennent plus complexes et furtives.