Livre blanc

Stratégies et Techniques pour échapper aux Antivirus modernes et contourner les EDR

NBS System
Groupe CELESTE

SOMMAIRE

Partie 1

Partie 2

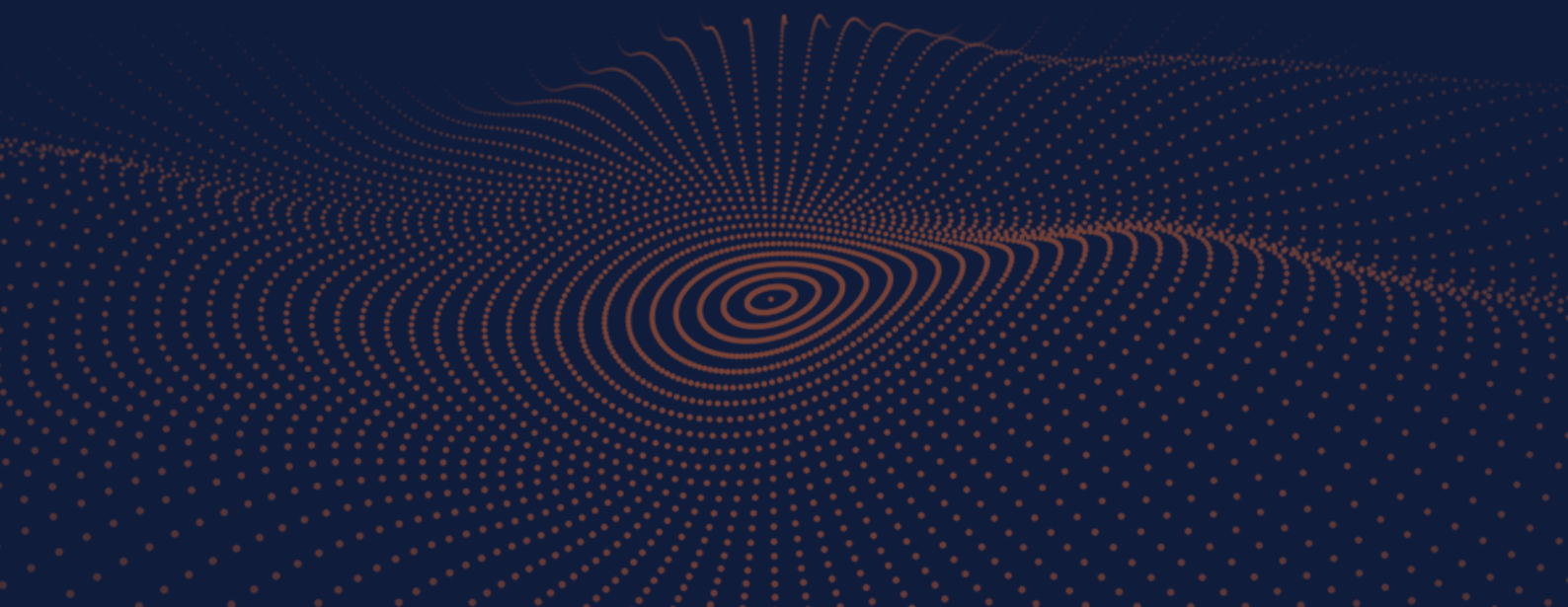
Partie 3

SOMMAIRE

Partie 4

NBS System
Groupe CELESTE

PARTIE 1



1. CONTEXTE ET PÉRIMÈTRE DES EDR ET ANTIVIRUS

Les **EDR (Endpoint Detection and Response)** et les **antivirus** sont des outils de sécurité essentiels pour protéger les systèmes informatiques contre les attaques malveillantes.

Cependant, ces outils ne sont pas infaillibles et peuvent être contournés par des attaquants expérimentés. Les missions de **pentest** sont des exercices visant à évaluer la sécurité d'un système informatique en testant sa vulnérabilité aux attaques.

Dans le cadre de ces missions, nous utilisons souvent des techniques de contournement des antivirus et des EDR pour évaluer leur efficacité et mettre en évidence les points faibles du système de sécurité.

Ce livre blanc explore ces différentes techniques utilisées dans les missions de test d'intrusion, en mettant en évidence leurs avantages et leurs limites. Le livre blanc sera découpée en trois parties.

2. OBJECTIFS ET MÉTHODOLOGIE

Notre projet vise à développer des techniques d'évasion avancées pour contourner les antivirus et les EDR. L'objectif principal de ce projet est d'optimiser nos **tests d'intrusion internes** pour nos clients, qui utilisent des solutions de sécurité avancées telles que les EDR pour renforcer leur environnement **Active Directory**.

Ce projet nous permettra de contourner les solutions de sécurité qui peuvent bloquer certaines activités malveillantes, assurant ainsi le succès de notre mission visant à obtenir les droits d'administrateur du domaine.

Par exemple, lors d'un pentest interne pour un client, nous pourrions utiliser cette approche pour développer une charge utile exécutable capable d'extraire les informations sensibles stockées dans la **mémoire LSASS** sans être détectée par les solutions de sécurité.

2. OBJECTIFS ET MÉTHODOLOGIE

Le **Local Security Authority Server Service (LSASS)** est un processus du système d'exploitation Windows responsable de l'authentification des utilisateurs : il vérifie les mots de passe, gère les changements de mots de passe, crée les tokens d'accès des processus utilisateurs, etc.

Ce processus est lancé au démarrage du système d'exploitation par le compte NT AUTHORITY/SYSTEM. En 2007, il a été découvert que les mots de passe des utilisateurs se trouvaient en clair dans la mémoire du processus LSASS.exe.

Cela signifie que si un attaquant parvenait à compromettre une machine Windows, il pourrait accéder aux mots de passe des utilisateurs connectés depuis le dernier redémarrage de la machine.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Pour comprendre comment les antivirus et les solutions EDR fonctionnent, il est important de se familiariser avec les méthodes qu'ils utilisent pour détecter et neutraliser les menaces.

Les antivirus utilisent généralement des bases de données de signatures pour identifier les logiciels malveillants connus, ainsi que des techniques de détection comportementale pour identifier les comportements suspects des programmes en cours d'exécution.

Les solutions EDR, quant à elles, surveillent les activités du système et des utilisateurs en temps réel, en utilisant des techniques de détection comportementale avancées pour identifier les comportements anormaux.

Dans les prochaines parties, nous explorerons plus en détail les méthodes de détection utilisées par les antivirus et les solutions EDR, ainsi que les avantages et les limites de chaque type de logiciel de sécurité.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

A. PRINCIPE DE FONCTIONNEMENT DES ANTIVIRUS

Afin de mettre au point des techniques d'évasion efficaces, il est essentiel de bien comprendre le fonctionnement des antivirus. Pour cela, nous avons consacré une part importante de notre temps à étudier les techniques de détection des antivirus, notamment les algorithmes de signature et l'analyse dynamique.

Cela nous a permis de mieux comprendre les mécanismes de détection et les limites des solutions de sécurité actuelles. Voici les principes fondamentaux du fonctionnement des antivirus :

Analyse statique des signatures :

L'analyse de signature est basée sur une liste noire de méthodes et de séquences d'octets. Lorsqu'un nouveau malware est détecté par un analyste, une signature spécifique est créée pour celui-ci.

La signature peut être basée sur un code ou des données spécifiques (par exemple, en utilisant une chaîne de caractères ou un prototype de fonction).

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Très souvent, les signatures sont construites sur les premiers octets exécutables d'un fichier malveillant. L'antivirus contient des millions de signatures dans sa base de données, qui sont comparées aux séquences d’octets des fichiers suspects afin de détecter une correspondance.

Les premiers antivirus utilisaient cette méthode, mais elle est encore efficace aujourd'hui en combinaison avec l'analyse dynamique. Le plus gros problème avec l'analyse basée sur les signatures est qu’elle ne peut pas être utilisée pour détecter de nouveaux logiciels malveillants.

Pour contourner l'analyse basée sur la signature, il nous suffit donc de créer un nouveau code ou de modifier considérablement le code existant pour contourner la signature actuelle.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Analyse dynamique :

De nos jours, de plus en plus d'antivirus reposent sur une approche dynamique. Lorsqu'un fichier exécutable est analysé, il est lancé pendant une courte durée dans une machine virtuelle (bac à sable).

Combiné à la vérification des signatures, il peut détecter les logiciels malveillants inconnus, même ceux qui reposent sur le chiffrement. En effet, l'analyse dynamique est une fonctionnalité complexe capable d'analyser des millions de fichiers, de les exécuter dans un environnement émulé et de vérifier toutes les signatures. Cependant, elle a aussi quelques limitations :

- L'analyse est trop rapide, l'antivirus a donc une limite par rapport au nombre d'opérations qu'il peut effectuer pour chaque analyse.

- L'environnement étant émulé, les spécifications de l'appareil et l'environnement du malware sont inconnus.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

B. Principe de fonctionnement des EDR

Pourquoi un EDR ?

Les solutions EDR sont devenues un élément essentiel de la sécurité des entreprises. Elles permettent de surveiller les activités des points finaux (postes de travail, serveurs, etc.) en temps réel et de détecter les comportements anormaux grâce à des algorithmes avancés. Elles sont particulièrement efficaces pour contrer les menaces avancées qui échappent aux antivirus traditionnels.

Comment fonctionne un EDR ?

Les EDR visent à détecter et à répondre aux menaces avancées qui ciblent les endpoints, c'est-à-dire les terminaux (ordinateurs, téléphones, tablettes, etc.) qui sont utilisés pour accéder aux systèmes d'information d'une entreprise.

Voici le principe de fonctionnement d'un EDR

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Collecte de données

L'EDR collecte des données à partir des points finaux, telles que les événements système, les fichiers, les journaux d'application, les connexions réseau, les processus en cours d'exécution, etc. Ces données sont collectées en temps réel pour fournir une vue en temps réel de l'état des points finaux.

Analyse des données

L'EDR utilise des algorithmes d'analyse pour détecter les comportements malveillants. Ces algorithmes sont basés sur des modèles de comportement normaux, de sorte que toute activité qui s'écarte de ces modèles est considérée comme suspecte. Ils utilisent également des bases de données de signatures de virus et de logiciels malveillants pour identifier les menaces connues.

Alertes

Lorsqu'une activité suspecte est détectée, l'EDR génère une alerte pour informer les équipes de sécurité. Les alertes peuvent être déclenchées par des événements tels que l'ouverture d'un fichier malveillant, la tentative d'exécution d'une commande dangereuse, la modification des paramètres système, etc. Les alertes sont généralement classées en fonction de leur niveau de gravité.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Investigation

Les équipes de sécurité enquêtent sur les alertes pour déterminer si elles sont réellement malveillantes ou non. Les EDRs fournissent des outils d'investigation pour aider les équipes de sécurité à recueillir des preuves, à examiner les fichiers, à analyser les connexions réseau, à suivre les processus en cours d'exécution, etc.

Réponse à incident

Si une menace est confirmée, l'EDR déclenche une réponse automatique ou manuelle pour neutraliser la menace. Les réponses peuvent inclure l'isolation de l'Endpoint, la suppression du fichier malveillant, la fermeture des connexions réseau suspectes, la restauration du système à un état antérieur, etc.

Rapports

L'EDR génère des rapports pour fournir une vue d'ensemble de l'activité de sécurité, y compris le nombre d'alertes générées, le temps de réponse moyen, le nombre de menaces confirmées, etc. Ces rapports peuvent être utilisés pour surveiller l'efficacité de la sécurité et pour identifier les zones à risque.

3. FONCTIONNEMENT DES ANTIVIRUS ET EDR

Quel est le meilleur EDR ?

La sélection du meilleur EDR dépend des besoins spécifiques de chaque organisation. Parmi les solutions EDR les plus populaires, on trouve SentinelOne, Cortex, CrowdStrike et bien d'autres. Ces produits de sécurité se distinguent par leurs fonctionnalités avancées, leur capacité à détecter les programmes malveillants, et leur efficacité à répondre aux incidents en temps réel.

Dans les sections à venir, nous fournirons une explication détaillée sur les techniques avancées employées par les EDRs.

EN RÉSUMÉ

En résumé, l'EDR est une solution de sécurité qui collecte et analyse des données à partir des points finaux pour détecter les menaces avancées et y répondre de manière rapide et efficace.

Les EDRs sont devenus un élément essentiel de la sécurité des entreprises, en particulier dans un contexte où les attaques informatiques sont de plus en plus sophistiquées et ciblées. Pour les professionnels de la sécurité informatique, il est crucial de comprendre les mécanismes de fonctionnement des EDR et des antivirus afin de développer des stratégies de défense et des stratégies de contournement efficaces.

NBS System
Groupe CELESTE

PARTIE 2

1. TECHNIQUES D'ÉVASION DES EDRS ET ANTIVIRUS

Pour réaliser notre travail de recherche sur les techniques d'évasion des antivirus et EDR, nous avons commencé par effectuer une analyse approfondie de la documentation existante, en utilisant des sources telles que des publications de recherche, des blogs de sécurité, des présentations de conférences, des forums de sécurité en ligne, etc.

Nous avons également consulté des outils de sécurité **open source** pour comprendre comment ces outils utilisent des techniques d'évasion pour contourner les antivirus et EDR.

Après avoir identifié les techniques d'évasion les plus courantes, nous avons commencé à les implémenter en développant des templates en **C++**. Les templates sont des modèles de code qui peuvent être réutilisés pour implémenter différentes techniques d'évasion.

Nous avons développé une base de templates centralisée à partir de chaque technique d'évasion identifiée, ce qui nous a permis de gagner du temps et d'efficacité dans le développement des **payloads indétectables** pour différents scénarios.

1. TECHNIQUES D'ÉVASION DES EDRS ET ANTIVIRUS

Les étapes que nous avons suivies pour implémenter ces techniques d'évasion en développant des templates comprenaient :

Analyse de la technique d'évasion choisie

Comprendre son fonctionnement et les conditions dans lesquelles elle est efficace.

Développement d'un template de code

Utilisation du langage C++ pour la technique d'évasion en question.

Test de la technique d'évasion

Vérification de son efficacité sur une machine virtuelle avec différents antivirus et EDR.

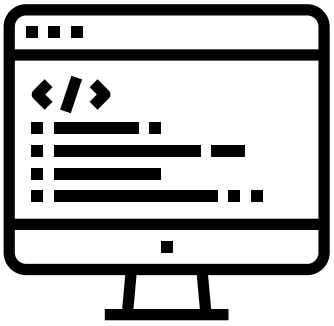
Ajout du template à la base centrale

Pour une utilisation ultérieure.

Répétition de ces étapes pour chaque technique d'évasion identifiée.

2. INJECTION DE CODE EN MÉMOIRE

Dans le cadre de notre projet, nous nous sommes intéressés aux techniques **d'injection de code** (ou shellcode) en mémoire dans un processus distant pour effectuer différentes attaques durant les tests d'intrusion chez les clients, telles que le vol d'informations, le contrôle à distance, le chiffrement de données, la suppression de fichiers et la propagation de logiciels malveillants sur le système cible.



Après avoir examiné différentes techniques d'évasion, nous avons constaté que l'injection en mémoire est une méthode efficace pour surmonter les difficultés liées à l'exécution du shellcode à l'intérieur d'un processus unique, qui pourrait être terminé par l'utilisateur du système cible.

En effet, l'injection en mémoire dans un processus distant nous fournit une stabilité accrue sur le système cible, notre code pouvant être injecté dans un processus qui fonctionne en permanence, tel que **« explorer.exe »**, assurant ainsi une persistance.

2. INJECTION DE CODE EN MÉMOIRE

Cette méthode est également discrète, notamment lorsque notre shellcode doit effectuer des appels réseaux sortants vers internet. Par exemple, nous pouvons choisir d'injecter notre code dans un processus légitime tel que **« OneDrive.exe »**, qui a l'habitude de faire des appels réseaux sortants, minimisant ainsi les risques de détection par les EDR et antivirus.

C'est devenue l'une des méthodes les plus populaires et efficaces pour les attaquants cherchant à échapper à la détection et à mener des attaques malveillantes sur des systèmes cibles.

> L'injection de shellcode peut se produire de différentes manières, mais elle implique généralement la création d'un code malveillant, qui est ensuite inséré dans la mémoire d'un processus en cours d'exécution à l'aide d'un programme exécutable nommé payload dans ce rapport.

Il existe diverses variantes d'injection de code en mémoire, mais nous présenterons plus en détail les deux plus intéressantes dans les parties suivantes.

3. INJECTION DE DLL DEPUIS LE DISQUE (DLL INJECTION)

L'injection de DLL en mémoire depuis le disque est une technique utilisée par certains programmes malveillants pour charger une bibliothèque de **liens dynamiques (DLL)** malveillante dans l'espace mémoire d'un processus en cours d'exécution. Cela permet au programme malveillant de contourner les mesures de sécurité de l'ordinateur et de masquer sa présence.

L'injection de DLL en mémoire se déroule en plusieurs étapes :

- Tout d'abord, le programme malveillant doit localiser un processus en cours d'exécution qui peut être utilisé pour l'injection de la DLL. Ce processus doit être choisi avec soin car il doit posséder des privilèges suffisants pour charger la DLL.

- Pour injecter une DLL depuis le disque, nous devons d'abord déposer la DLL sur l'ordinateur cible. Il existe plusieurs méthodes pour le faire depuis notre payload. Dans notre cas, nous avons téléchargé via **HTTP** la DLL depuis un serveur distant et nous l'avons déposé sur le dossier **TEMP** du l'utilisateur cible.

3. INJECTION DE DLL DEPUIS LE DISQUE (DLL INJECTION)

- Ensuite, le programme malveillant (payload) doit allouer de l'espace mémoire dans le processus cible pour y charger la DLL. Cela peut être réalisé en utilisant des appels système spécifiques dans **l'API Win32** tels que **VirtualAllocEx()** pour allouer de l'espace dans l'espace mémoire du processus cible puis utiliser **WriteProcessMemory()** pour y copier le chemin et le nom de la DLL.

- Une fois que l'espace mémoire a été alloué, nous avons résolu l'adresse mémoire de la fonction **LoadLibraryA()** dans le processus distant. Heureusement pour nous, la plupart des DLLs natives de Windows sont allouées à la même adresse de base à travers les processus, donc l'adresse de LoadLibraryA() dans notre processus actuel sera la même que dans le processus distant. Nous avons donc réussi à résoudre l'adresse en faisant appel aux fonctions **GetModuleHandle()** et **GetProcAddress()**

- Pour forcer le processus cible à charger de **[DM3]** notre DLL malveillante en mémoire, nous avons invoqué la fonction **CreateRemoteThread()** qui va créer un thread dans le processus distant. Ce thread aura pour tâche de charger notre DLL malveillante avec la fonction LoadLibraryA().

3. INJECTION DE DLL DEPUIS LE DISQUE (DLL INJECTION)

- Enfin, le processus cible peut utiliser la DLL chargée en mémoire pour exécuter notre code malveillant.

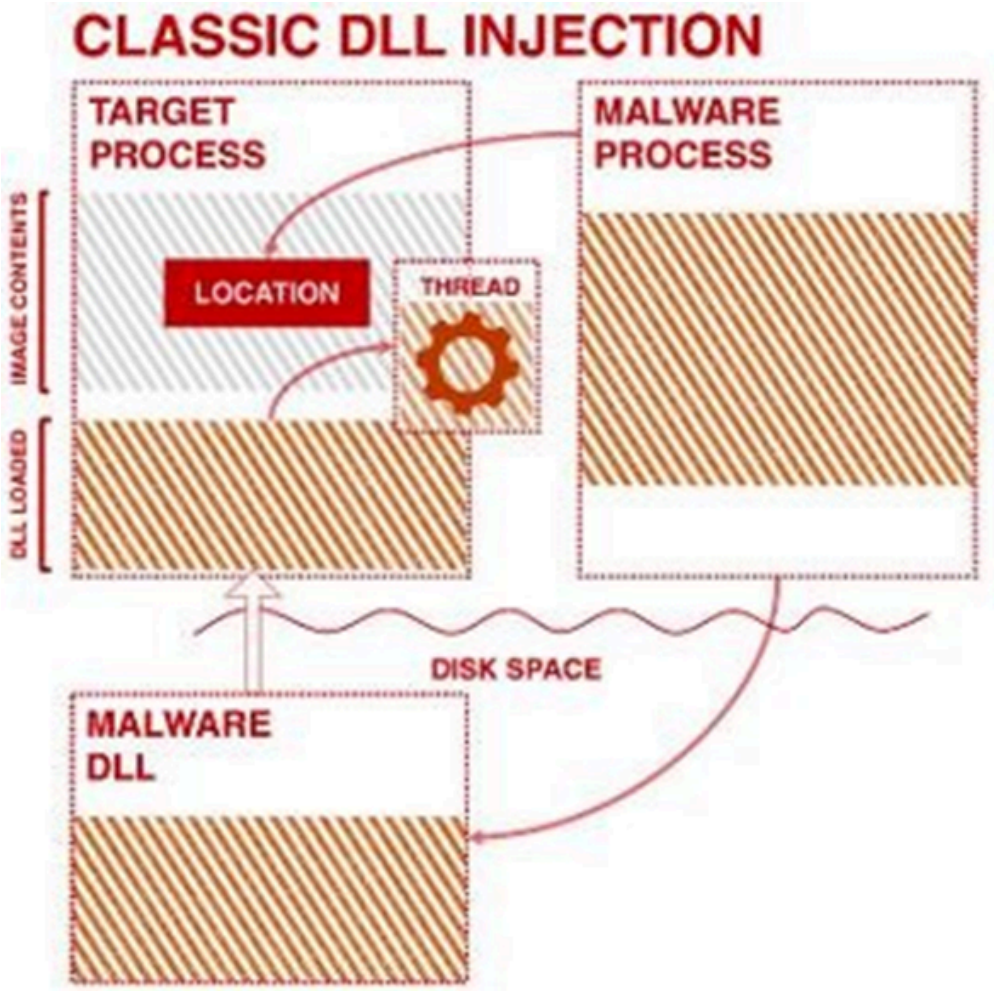


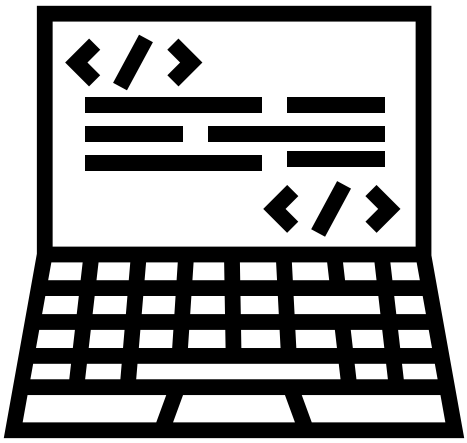
Figure 5 - Exemple d'injection DLL

3. INJECTION DE DLL DEPUIS LE DISQUE (DLL INJECTION)

La technique actuelle présente un inconvénient car la DLL malveillante doit être déposée dans le système cible. Toutefois, dans la section suivante, nous examinerons une technique plus discrète qui permettra de charger la DLL ou n'importe quel **PE** sans le stocker sur le disque dur.

Cette méthode est plus complexe car elle nécessite une compréhension approfondie du mécanisme utilisé le **loader de l'OS** (parsing de l'IAT, EAT, etc.) pour charger les PE et plus particulièrement les DLLs en mémoire.

Grâce à cette compréhension, nous serons en mesure de développer notre propre loader de PE et d'intégrer la DLL directement dans le code source au lieu de la déposer dans le disque.



4. INJECTION DE PE DEPUIS LE CODE (REFLECTIVE PE INJECTION)

L'injection d'un PE (Portable Executable) depuis le code est une technique utilisée par les programmes malveillants pour injecter un fichier exécutable (PE) notamment les DLLs dans la mémoire d'un processus en cours d'exécution, sans avoir à l'écrire sur le disque.

Cette technique est souvent utilisée pour contourner les mesures de sécurité du système d'exploitation, car elle évite de laisser des traces sur le disque et peut être difficile à détecter.

L'injection de DLL en mémoire se déroule en plusieurs étapes :

1. Tout d'abord, le programme malveillant doit localiser un processus en cours d'exécution qui peut être utilisé pour l'injection du PE. Ce processus doit être choisi avec soin car il doit posséder des privilèges suffisants pour charger le PE.

2. Ensuite, le programme malveillant alloue de l'espace mémoire dans le processus cible pour y charger le PE. Cette étape est similaire à l'étape 2 de l'injection de DLL depuis le disque.

4. INJECTION DE PE DEPUIS LE CODE (REFLECTIVE PE INJECTION)

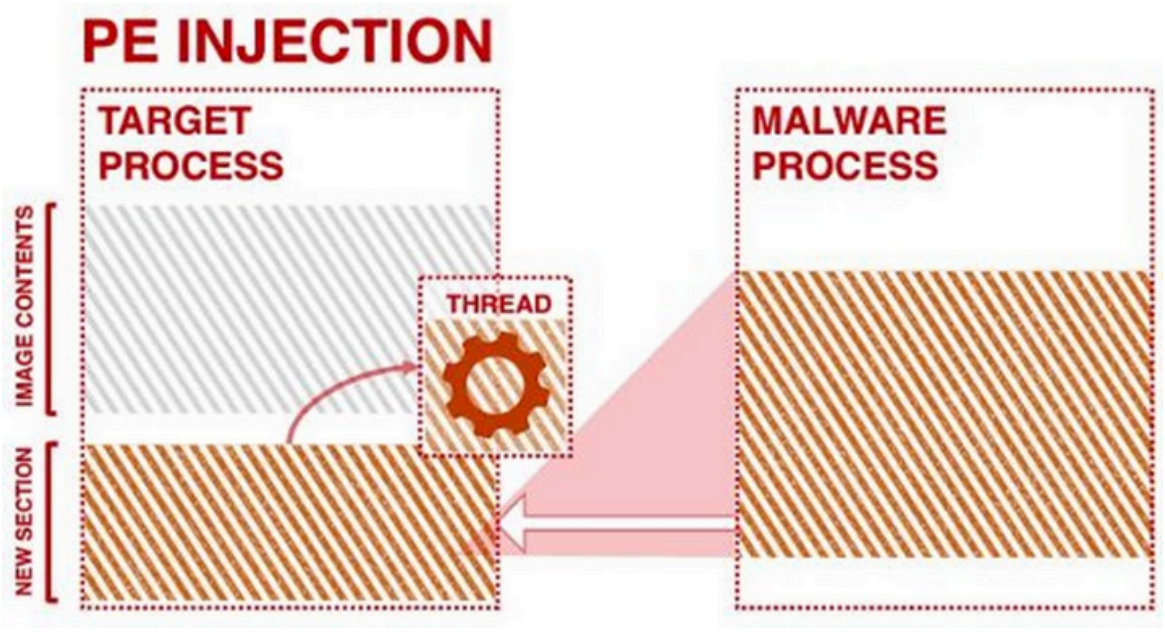
Note : Dans le scénario de « Reflective PE injection » : Le PE (dans notre cas une DLL) est stocké dans la mémoire de l'ordinateur cible. Plus précisément, la DLL est chargée directement en mémoire à partir d'un bloc de données dans notre processus malveillant en cours d'exécution sans avoir à toucher le disque. Cela rend le processus d'injection plus furtif et difficile à détecter.

3. Le programme malveillant doit ensuite copier le contenu du PE dans l'espace mémoire alloué. Cependant, le PE doit être adapté pour une exécution en mémoire, car les adresses de sauts et les adresses de chargement peuvent différer de celles prévues pour une exécution depuis le disque.

   Pour ce faire, le programme malveillant doit parser les différentes sections du PE, les charger en mémoire et modifier le PE en conséquence, en remplaçant les adresses absolues par des adresses relatives, en modifiant les points d'entrée, etc. Cela permettra au PE de s'exécuter correctement en mémoire.

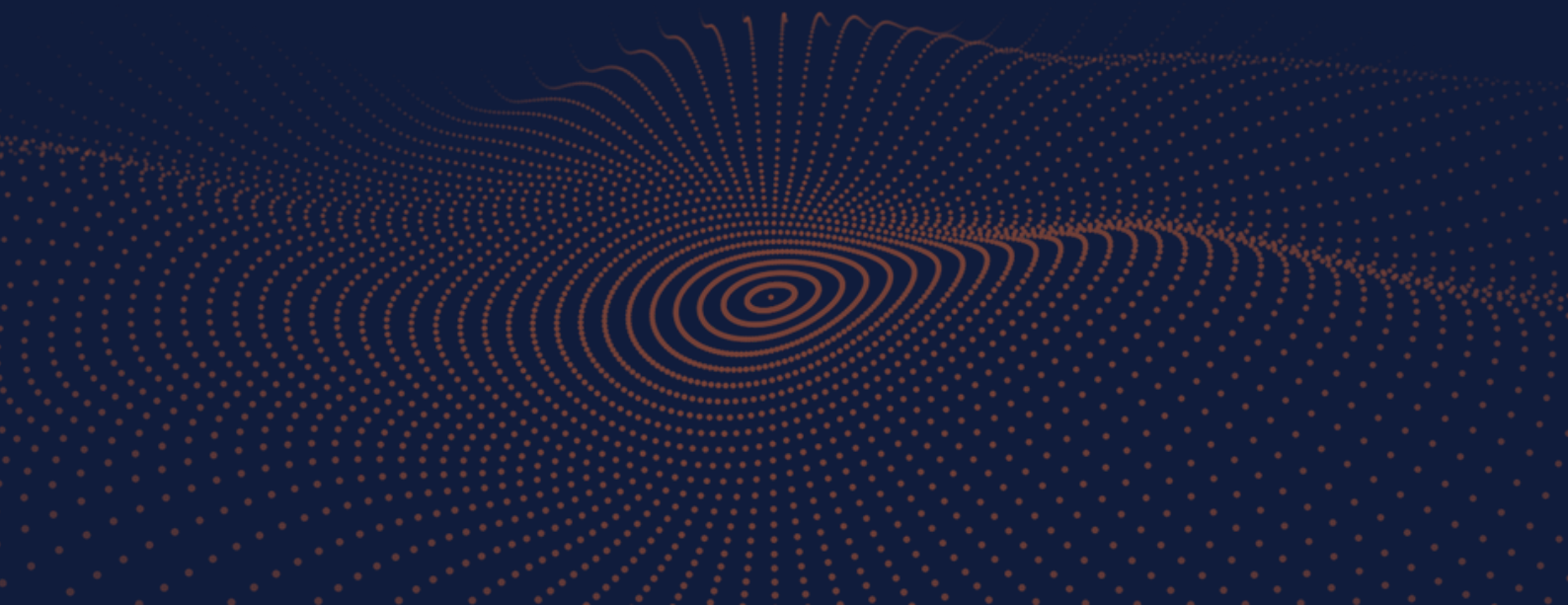
4. INJECTION DE PE DEPUIS LE CODE (REFLECTIVE PE INJECTION)

4. Enfin, le programme malveillant peut utiliser la fonction d'entrée du PE (point d'entrée) pour exécuter son code malveillant dans le contexte du processus cible.



NBS System
Groupe CELESTE

PARTIE 3



1. PRINCIPE DE FONCTIONNEMENT DU HOOKING

Après avoir effectué des tests en utilisant les techniques d'injection mentionnées précédemment, il est apparu qu'elles étaient efficaces contre les antivirus mais pas contre les EDRs.

Nous avons donc dû approfondir notre compréhension du fonctionnement des EDRs en effectuant des recherches.

Après plusieurs investigations, nous avons compris que ces derniers utilisaient des mécanismes plus avancés que les antivirus, dont l'un des plus importants est le **Hooking**.

> Le Hooking est une technique utilisée par les EDRs pour surveiller les activités des processus en cours d'exécution sur un **Endpoint**. L'objectif principal du Hooking est de fournir une visibilité approfondie sur les actions effectuées par les programmes, ce qui permet de détecter les comportements malveillants et de répondre à ces menaces de manière **proactive**.

Voici le principe de fonctionnement du Hooking

1. PRINCIPE DE FONCTIONNEMENT DU HOOKING

Injection de code

Les EDR utilisent des techniques d'injection de code pour ajouter du code supplémentaire aux processus en cours d'exécution dans leur espace mémoire exécutable. Ce code supplémentaire est appelé "Hook" et permet de surveiller les activités du processus.

Surveillance des activités

Une fois le Hook injecté, l'EDR peut surveiller les activités du processus, telles que la création de fichiers, l'ouverture de connexions réseau, la modification des paramètres système, etc. L'EDR peut également surveiller les appels de fonctions effectués par le processus en question.

Analyse comportementale

Les activités surveillées sont ensuite analysées pour détecter les comportements malveillants. Les EDRs utilisent des algorithmes d'analyse comportementale pour détecter les activités suspectes, telles que la modification de fichiers système, l'installation de logiciels malveillants, etc.

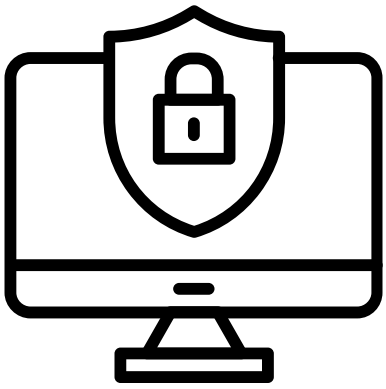
1. PRINCIPE DE FONCTIONNEMENT DU HOOKING

Déclenchement d'alertes

Si une activité suspecte est détectée, l'EDR génère une alerte pour informer les équipes de sécurité. Les alertes peuvent être déclenchées par des événements tels que la création d'un fichier malveillant, la modification des paramètres système, la tentative d'exécution d'une commande dangereuse, etc.

Réponse

Si une menace est confirmée, l'EDR déclenche une réponse automatique ou manuelle pour neutraliser la menace. Les réponses peuvent inclure l'isolation de l'Endpoint, la suppression du fichier malveillant, la fermeture des connexions réseau suspectes, la restauration du système à un état antérieur, etc.



1. PRINCIPE DE FONCTIONNEMENT DU HOOKING

L'intérêt du Hooking pour les EDRs réside dans le fait qu'il permet de **surveiller** les activités des processus en cours d'exécution de manière **fine** et **précise**. Cette surveillance approfondie permet de détecter les comportements malveillants qui pourraient autrement passer inaperçus pour la **BlueTeam**.

En outre, le Hooking peut être utilisé pour surveiller des processus spécifiques, ce qui permet de cibler des menaces potentielles et de détecter les comportements malveillants avant qu'ils ne causent des dommages.

Cependant, il convient de noter que le Hooking est une technique avancée qui nécessite une **connaissance approfondie** des systèmes d'exploitation et des processus en cours d'exécution. Une mauvaise utilisation du Hooking peut causer des **dysfonctionnements** sur les endpoints surveillés, ce qui peut entraîner des conséquences négatives sur la sécurité et la performance du système

2. TECHNIQUES DE HOOKING

Les EDRs utilisent plusieurs techniques de Hooking pour surveiller les activités des processus et détecter les menaces.

Dans notre projet, nous nous sommes concentrés spécifiquement sur les techniques de Hooking en **User land** plutôt qu'en **Kernel Land**, car elles sont les plus couramment utilisées par les EDRs pour plusieurs raisons :

Sécurité

Les **Hooks** en Kernel Land ont un accès privilégié au système d'exploitation et peuvent potentiellement causer des instabilités ou des plantages s'ils ne sont pas bien conçus ou bien implémentés. De plus, les Hooks en Kernel Land sont souvent la cible privilégiée des attaques de rootkits, qui tentent de masquer leur présence en modifiant le comportement du système d'exploitation.

Performance

Les Hooks en Kernel Land peuvent également impacter les performances du système en raison de la surcharge de traitement. En revanche, les Hooks en Userland ont moins d'impact sur les performances du système et sont plus faciles à déployer.

2. TECHNIQUES DE HOOKING

Flexibilité

Les Hooks en Userland sont plus flexibles que les Hooks en Kernel Land, car ils peuvent être placés sur des processus spécifiques plutôt que sur l'ensemble du système d'exploitation. Cette flexibilité permet aux EDR de surveiller les activités des processus spécifiques qui sont les plus susceptibles d'être ciblés par les attaques.

Cependant, il convient de noter que les Hooks en Userland ont également des limites, car ils ne peuvent surveiller que les activités des processus exécutés dans l'espace utilisateur. Les attaques qui visent directement le système d'exploitation et qui ne passent pas par l'espace utilisateur peuvent échapper à la surveillance des EDR utilisant uniquement des Hooks en Userland.

Ainsi, certains EDR peuvent utiliser une combinaison de Hooks en Userland et en Kernel Land pour offrir une surveillance plus complète et une protection plus efficace contre les menaces avancées.

Parmi les techniques de Hooking les plus utilisées en Userland par les EDRs nous trouvons

2. TECHNIQUES DE HOOKING

A. Principe de fonctionnement des antivirus

Cette technique consiste à remplacer les premières instructions d'une fonction existante par un code Hook qui intercepte les appels de fonctions. Cette technique est souvent utilisée pour surveiller les activités des processus en cours d'exécution. Voici un exemple de la technique :

« Inline Hooking » utilisée par un EDR : Supposons qu'un EDR souhaite surveiller les activités du processus malveillant **"evil.exe"** en interceptant l'appel à la fonction **CreateFileA**. Pour ce faire, l'EDR peut utiliser l'Inline Hooking pour remplacer les premières instructions de cette fonction par des instructions personnalisées qui redirigent l'exécution vers une fonction personnalisée.

1. Tout d'abord, l'EDR recherche l'emplacement de la fonction CreateFileA dans le code de "evil.exe".

2. Il modifie les premières instructions de cette fonction pour qu'elles redirigent l'exécution vers une fonction personnalisée, appelée par exemple **MyCreateFileA.**

2. TECHNIQUES DE HOOKING

3. La fonction MyCreateFileA est conçue pour enregistrer les informations sur la création de fichiers, telles que le nom du fichier, l'heure de création, l'emplacement, etc.

4. Lorsque "evil.exe" utilise la fonction CreateFileA, l'Inline Hooking redirige l'appel vers la fonction MyCreateFileA, qui enregistre les informations sur le fichier créé.

5. Après l'exécution de la fonction MyCreateFileA, l'EDR analyse les informations enregistrées pour détecter les comportements malveillants, tels que la création de fichiers suspects.

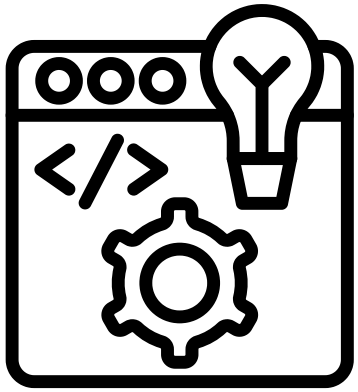
Il est important de noter que cette stratégie de Hooking peut être détectée et contournée par des **attaquants expérimentés**. La technologie des EDRs utilise donc souvent plusieurs techniques de Hooking Userland pour offrir une surveillance plus complète et une protection plus efficace du système d'exploitation contre les **menaces avancées**.

2. TECHNIQUES DE HOOKING

La technique actuelle présente un inconvénient car la **DLL** malveillante doit être déposée dans le système cible. Toutefois, dans la section suivante, nous examinerons une technique plus discrète qui permettra de charger la DLL ou n'importe quel **PE** sans le stocker sur le disque dur.

Cette méthode est plus complexe car elle nécessite une compréhension approfondie du mécanisme utilisé le loader de l'**OS** (parsing de l'**IAT**, **EAT**, etc.) pour charger les PE et plus particulièrement les DLLs en mémoire.

Grâce à cette compréhension, nous serons en mesure de développer notre propre loader de PE et d'intégrer la DLL directement dans le code source au lieu de la déposer dans le disque.



2. TECHNIQUES DE HOOKING

B. IAT Hooking

Le **IAT Hooking (Import Address Table Hooking)** est une autre technique de Hooking en Userland couramment utilisée par les EDRs pour intercepter les appels de fonctions dans les modules DLL chargés dans un processus. Elle consiste à modifier la table des adresses d'importation (IAT) d'un processus pour rediriger les appels de fonctions vers des fonctions personnalisées.

Cette technique permet à l'attaquant d'intercepter les appels de fonctions sans avoir besoin de modifier directement les instructions de la fonction. Il peut simplement modifier la table d'adresses d'importation pour rediriger les appels de fonctions vers une fonction personnalisée. Voici un exemple d'utilisation de la technique d'IAT Hooking par un EDR :

> Supposons qu'un EDR souhaite surveiller les appels à la fonction LoadLibraryA dans le processus "evil.exe". Il peut utiliser la technique d'IAT Hooking pour modifier l'adresse d'importation de cette fonction dans la table des adresses d'importation de "evil.exe" et rediriger les appels vers une fonction personnalisée.

2. TECHNIQUES DE HOOKING

1. L'EDR recherche l'emplacement de la table des adresses d'importation de "evil.exe" pour la DLL contenant la fonction LoadLibraryA.

2. Il modifie l'adresse d'importation de la fonction LoadLibraryA dans la table des adresses d'importation de "evil.exe" pour rediriger les appels vers une fonction personnalisée, appelée par exemple MyLoadLibraryA.

3. La fonction MyLoadLibraryA est conçue pour enregistrer les informations sur les DLL chargées par "evil.exe", telles que le nom de la DLL, l'emplacement, etc.

4. Lorsque "evil.exe" utilise la fonction LoadLibraryA, la table des adresses d'importation redirige l'appel vers la fonction MyLoadLibraryA, qui enregistre les informations sur la DLL chargée.

5. Après l'exécution de la fonction MyLoadLibraryA, l'EDR analyse les informations enregistrées pour détecter les comportements malveillants, tels que le chargement de DLL suspectes.

2. TECHNIQUES DE HOOKING

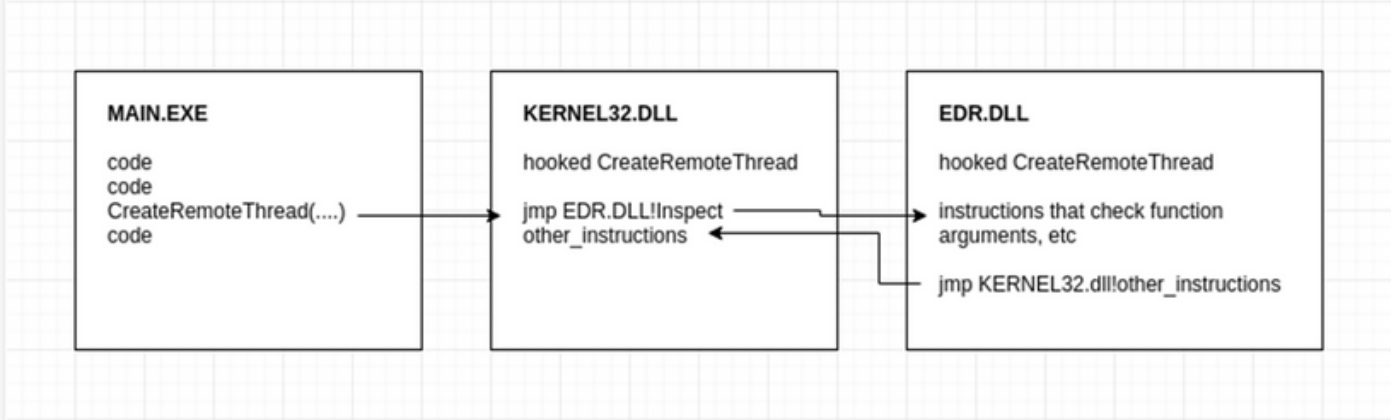


Figure 7 - Exemple d'InlineHooking utilisé par les EDRs

2. TECHNIQUES DE HOOKING

B. IAT Hooking

Le **IAT Hooking (Import Address Table Hooking)** est une autre technique de Hooking en Userland couramment utilisée par les EDRs pour intercepter les **appels de fonctions** dans les modules DLL chargés dans un processus. Elle consiste à modifier la table des adresses d'importation (IAT) d'un processus pour rediriger les appels de fonctions vers des fonctions personnalisées.

Cette technique permet à l'attaquant d'intercepter les appels de fonctions sans avoir besoin de modifier directement les instructions de la fonction. Il peut simplement modifier la **table d'adresses d'importation** pour rediriger les appels de fonctions vers une fonction personnalisée. Voici un exemple d'utilisation de la technique d'IAT Hooking par un EDR :

> Supposons qu'un EDR souhaite surveiller les appels à la fonction LoadLibraryA dans le processus "evil.exe". Il peut utiliser la technique d'IAT Hooking pour modifier l'adresse d'importation de cette fonction dans la table des adresses d'importation de "evil.exe" et rediriger les appels vers une fonction personnalisée.

2. TECHNIQUES DE HOOKING

1. L'EDR recherche l'emplacement de la table des adresses d'importation de "evil.exe" pour la DLL contenant la fonction **LoadLibraryA**.

2. Il modifie l'adresse d'importation de la fonction LoadLibraryA dans la table des adresses d'importation de "evil.exe" pour rediriger les appels vers une fonction personnalisée, appelée par exemple **MyLoadLibraryA**.

3. La fonction MyLoadLibraryA est conçue pour enregistrer les informations sur les DLL chargées par "evil.exe", telles que le nom de la DLL, l'emplacement, etc.

4. Lorsque "evil.exe" utilise la fonction LoadLibraryA, la table des adresses d'importation redirige l'appel vers la fonction MyLoadLibraryA, qui enregistre les informations sur la DLL chargée.

5. Après l'exécution de la fonction MyLoadLibraryA, l'EDR analyse les informations enregistrées pour détecter les comportements malveillants, tels que le chargement de DLL suspectes.

2. TECHNIQUES DE HOOKING

Figure 8 - Exemple d'IAT Hooking sur la fonction GetCurrentProcessId

3. EDR ET HOOKING SUR LA NTDLL

NTDLL (NT Dynamic Link Library) est une bibliothèque de liens dynamiques qui contient des fonctions importantes pour l'exécution de **Windows NT**. Les fonctions de la NTDLL sont utilisées par de nombreux processus et applications système de Windows, et sont donc une cible importante pour les attaquants qui souhaitent intercepter et modifier le comportement de ces processus.

Les EDRs utilisent souvent des techniques de Hooking pour surveiller les appels de fonctions de la NTDLL et détecter les comportements malveillants.

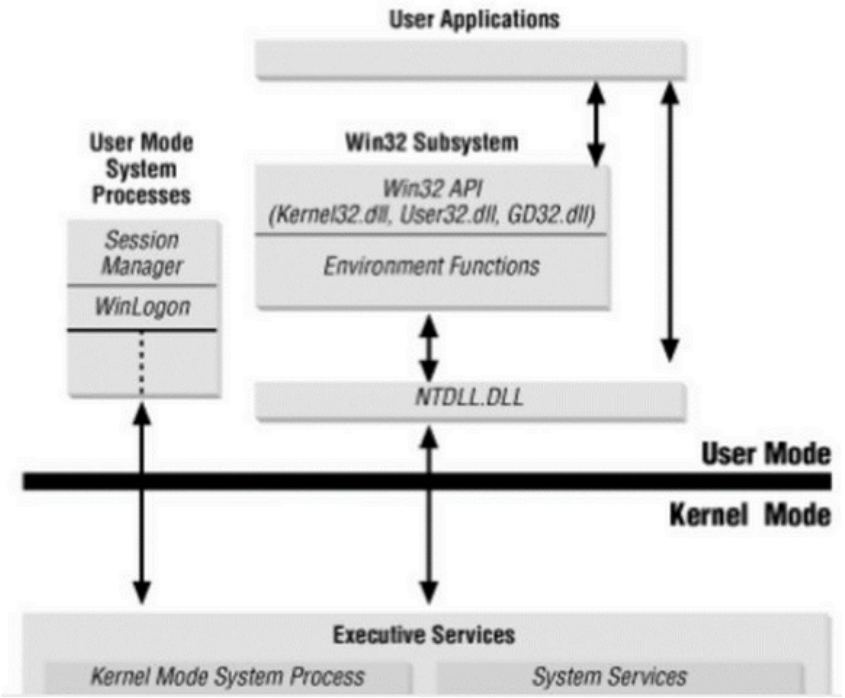


Figure 9 - NTDLL et l'architecture de l'OS Windows (Image credit : TechNet)

3. EDR ET HOOKING SUR LA NTDLL

Les appels aux fonctions de la NTDLL sont utilisés par de nombreux processus système et applications, et l'interception de ces appels peut fournir des informations précieuses sur l'activité d'un processus.

Les EDRs utilisent des techniques de Hooking pour intercepter les appels de fonctions de la NTDLL, telles que **NtCreateFile**, **NtOpenFile**, **NtReadFile**, **NtWriteFile**, **NtAllocateVirtualMemory**, **NtCreateSection**, **NtMapViewOfSection**, etc. Ils peuvent également utiliser différentes techniques de Hooking pour intercepter les appels de fonctions de la NTDLL.

Par exemple, l'Inline Hooking peut être utilisé pour modifier les instructions de la fonction de la NTDLL et rediriger les appels vers une fonction personnalisée, tandis que le IAT Hooking peut être utilisé pour modifier la table d'adresses d'importation de la NTDLL et rediriger les appels vers une fonction personnalisée.

3. EDR ET HOOKING SUR LA NTDLL

Les appels aux fonctions de la NTDLL sont utilisés par de nombreux processus système et applications, et l'interception de ces appels peut fournir des informations précieuses sur l'activité d'un processus.

Les EDRs utilisent des techniques de Hooking pour intercepter les appels de fonctions de la NTDLL, telles que NtCreateFile, NtOpenFile, NtReadFile, NtWriteFile, NtAllocateVirtualMemory, NtCreateSection, NtMapViewOfSection, etc. Ils peuvent également utiliser différentes techniques de Hooking pour intercepter les appels de fonctions de la NTDLL.

Par exemple, l'Inline Hooking peut être utilisé pour modifier les instructions de la fonction de la NTDLL et rediriger les appels vers une fonction personnalisée, tandis que le IAT Hooking peut être utilisé pour modifier la table d'adresses d'importation de la NTDLL et rediriger les appels vers une fonction personnalisée.

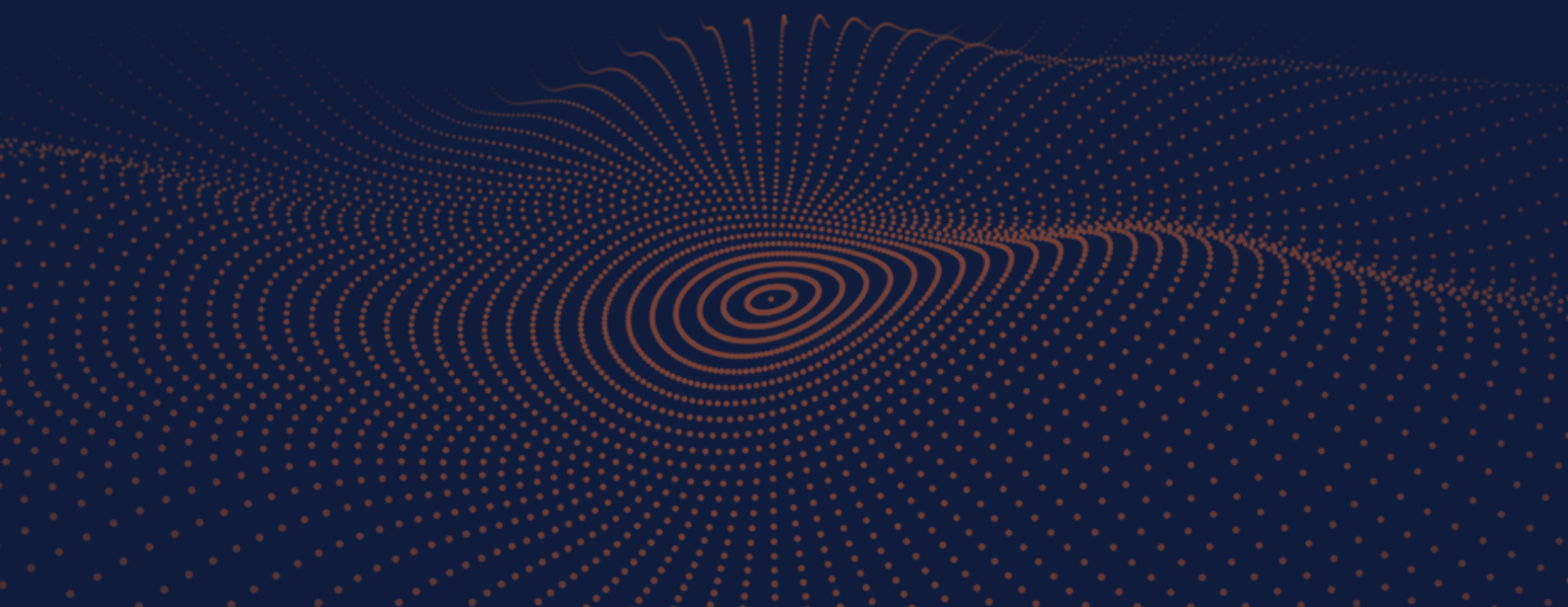
3. EDR ET HOOKING SUR LA NTDLL

En résumé, les EDR utilisent des techniques de Hooking pour intercepter les appels de fonctions de la NTDLL et surveiller l'activité des processus système et des applications.

L'interception des appels de fonctions de la NTDLL est une technique clé utilisée par les EDRs pour détecter les comportements malveillants, tels que la création de fichiers ou de processus suspects, le chargement de bibliothèques DLL malveillantes, etc.

NBS System
Groupe CELESTE

PARTIE 4



TECHNIQUES DE CONTOURNEMENT DE HOOKING SUR LA NTDLL

Nous pourrons contourner ces mesures de sécurité en utilisant des techniques de « **Unhooking** » afin de masquer nos activités malveillantes et éviter la détection.

L'intérêt du Unhooking pour les attaquants est de **masquer leurs activités malveillantes** en retirant les Hooks (ou crochets) mis en place par les EDRs pour les surveiller.

> Les Hooks sont des points d'entrée dans les processus qui permettent aux EDR d'intercepter les appels de fonctions et de surveiller les activités suspectes. En retirant ces hooks, les attaquants peuvent masquer leur présence et éviter la détection par les EDRs.

Les attaquants utilisent différentes techniques pour effectuer le « Unhooking ».

1. UNHOOKING NTDLL DEPUIS LE DISQUE

Après avoir compris le fonctionnement de la technique de Hooking utilisée par les EDRs sur la NTDLL, nous avons décidé d'implémenter une technique de contournement en chargeant une copie fraîche de la NTDLL depuis le disque.

> Cette technique consiste à charger une copie non modifiée de la NTDLL dans l'espace d'adressage de notre processus, évitant ainsi les Hooks installés par les antivirus et EDR sur la version chargée en mémoire

Pour mettre en œuvre cette technique, nous avons commencé par écrire un code en C++ pour charger la NTDLL depuis le disque dur, en utilisant les fonctions **CreateFileMapping()** et **MapViewOfFile()**.

Nous avons ensuite modifié les appels système dans notre code pour utiliser les fonctions de la NTDLL chargée depuis le disque plutôt que celles de la version potentiellement modifiée en mémoire.

1. UNHOOKING NTDLL DEPUIS LE DISQUE

```
hFile = CreateFile((LPCSTR) sNtdllPath, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);

hFileMapping = CreateFileMappingA_p(hFile, NULL, PAGE_READONLY | SEC_IMAGE, 0, 0, NULL);

pMapping = MapViewOfFile_p(hFileMapping, FILE_MAP_READ, 0, 0, 0);

result = UnhookNtdll(GetModuleHandle((LPCSTR) sNtdll), pMapping);
```

Figure 10 - Unhooking de la NTDLL depuisle disque (CreateFileMapping)

1. UNHOOKING NTDLL DEPUIS LE DISQUE

Nous avons ensuite réalisé des tests afin d'évaluer l'efficacité de cette technique. Nous avons comparé les résultats obtenus avec cette technique à ceux issus d'autres méthodes de contournement , ainsi qu'à une exécution normale du code sans aucun contournement.

Les résultats ont montré que cette technique de contournement était **efficace pour éviter les Hooks** de la NTDLL installés par les solutions antivirus et EDR.

Cependant, nous avons également rencontré des difficultés lors de l'implémentation de cette technique, notamment en raison de la complexité de l'architecture de la NTDLL et de la nécessité de prendre en compte les différentes versions du système d'exploitation ainsi que les différences entre les processeurs 32 bits et 64 bits.

Nous avons également dû faire face à des défis liés à l'intégration et à la compatibilité de notre solution avec divers environnements clients.

1. UNHOOKING NTDLL DEPUIS LE DISQUE

En conclusion, l'implémentation dela technique de contournement de Hooking de laNTDLL en chargeant une fraîche copie de la NTDLL depuisle disque a été une étape importante de notre projet de développement de techniques d'évasion. Bien que cette technique ait montré son efficacitédans certains cas, elle nécessite une compréhension approfondie du fonctionnement de la NTDLL et peut être bloquée par certaines solutions antivirus et EDR.

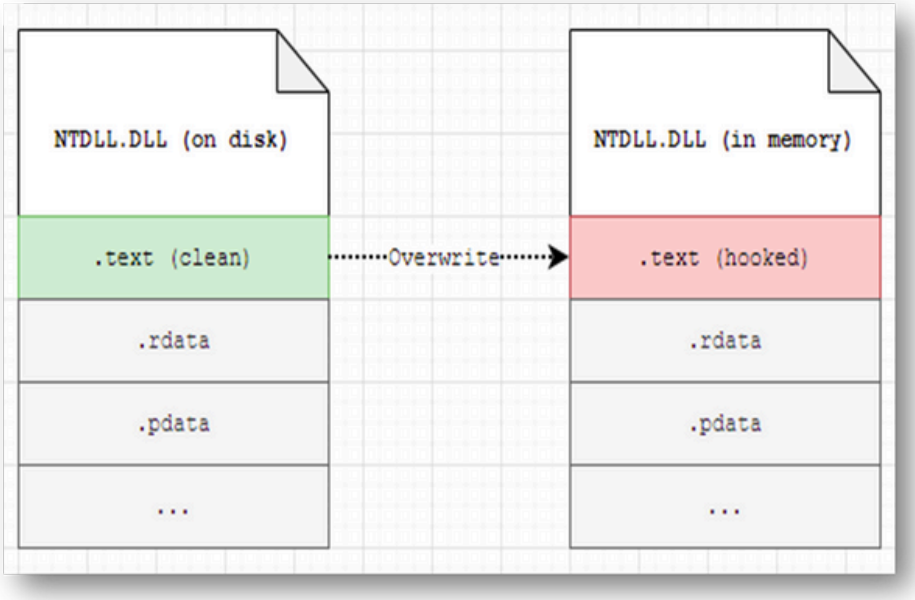


Figure 11 - Unhooking de la NTDLL depuisle disque

2. UNHOOKING NTDLL DEPUIS UN PROCESSUS SUSPENDU

La technique précédente qui consiste à charger en mémoire une NTDLL depuis le disque présente une limite majeure, car chaque processus a déjà une NTDLL dans son espace d'adressage, ce qui peut donc susciter la suspicion des antivirus et EDRs.

> Pour contourner cette limite, nous avons exploré d'autres méthodes pour charger une copie fraîchede la NTDLL en mémoire sans toucher l'originale sur le disque. Après des recherches sur le chargement des processus en mémoire, nous avons découvert qu'il était possible d'obtenir une NTDLL en mémoire sans Hooks en lançant un processus avec l'état suspendu.

Cette approche permet de court-circuiter le processus de Hooking des EDRs sur la NTDLL. Nous pouvons alors lancer un processus quelconque en mode suspendu, copier sa NTDLL en mémoire et remplacer la version hookée de la NTDLL dans le processus cible pour contourner les Hooks de l'antivirus ou de l'EDR.

2. UNHOOKING NTDLL DEPUIS UN PROCESSUS SUSPENDU

Pour mettre en œuvre cette technique, nous avons utilisé la fonction **CreateProcess** de Windows **en spécifiant le flag CREATE_SUSPENDED** dans les options de création du processus. Cette fonction renvoie un Handle du processus créé.

```
BOOL result = CreateProcessA(
    NULL,
    (LPSTR)"cmd.exe",
    NULL,
    NULL,
    FALSE,
    CREATE_SUSPENDED | CREATE_NEW_CONSOLE,
    NULL,
    "C:\\Windows\\System32\\",
    &si,
    &pi);
```

Figure 12 - Création de processus suspendu (Unhooking)

2. UNHOOKING NTDLL DEPUIS UN PROCESSUS SUSPENDU

Une fois que le processus est suspendu, nous pouvons accéder à sa mémoire et extraire la NTDLL en utilisant la fonction **GetModuleHandle** avec le nom de module **"ntdll.dll"**. Cette fonction renvoie un Handle du module en mémoire.

Ensuite, nous copions la NTDLL en mémoire dans un buffer en utilisant la fonction **memcpy**. Comme le processus est suspendu, il n'y a pas suffisamment de temps pour que les EDRs puissent injecter leurs Hooks dans la NTDLL. Nous obtenons ainsi une copie fraîche de la NTDLL en mémoire, sans Hooks. Nous pouvons alors utiliser cette copie pour remplacer la version hookée de la NTDLL dans notre processus malveillant.

Enfin, en utilisant cette copie non-hookée de la NTDLL, nous sommes alors en mesure **d'exécuter des actions malveillantes sans être détecté par les EDRs ou les antivirus**.

Il est important de noter que cette technique peut être détectée par les EDRs si le processus suspendu reste en état suspendu pendant une période prolongée, ce qui peut indiquer une activité suspecte. Il est donc recommandé de relancer rapidement le processus cible après avoir effectué les modifications nécessaires

2. UNHOOKING NTDLL DEPUIS UN PROCESSUS SUSPENDU

En résumé, l'implémentation de cette technique implique le lancement d'un processus en mode suspendu, l'extraction de sa NTDLL en mémoire, la copie et l'utilisation de cette NTDLL non hookéedans notre processus malveillant. Cette technique peut contourner les hooks de l'antivirus ou de l'EDR,mais elle peut être détectéesi elle est utilisée de manière prolongée.

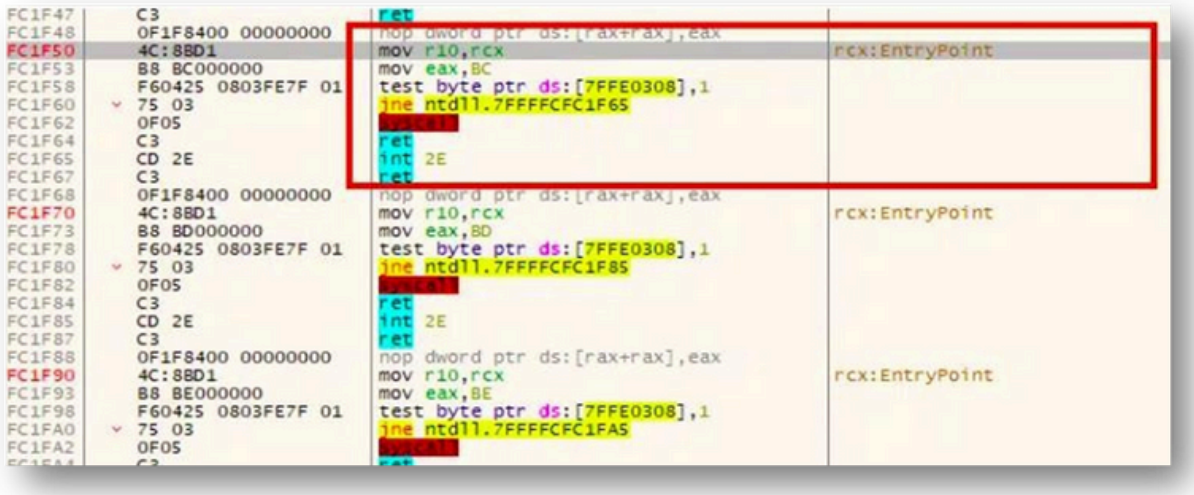


Figure 12 - Absence de hooks dans les fonctions de la NTDLL d'un process suspendu via xdbg

2. UNHOOKING NTDLL DEPUIS UN PROCESSUS SUSPENDU

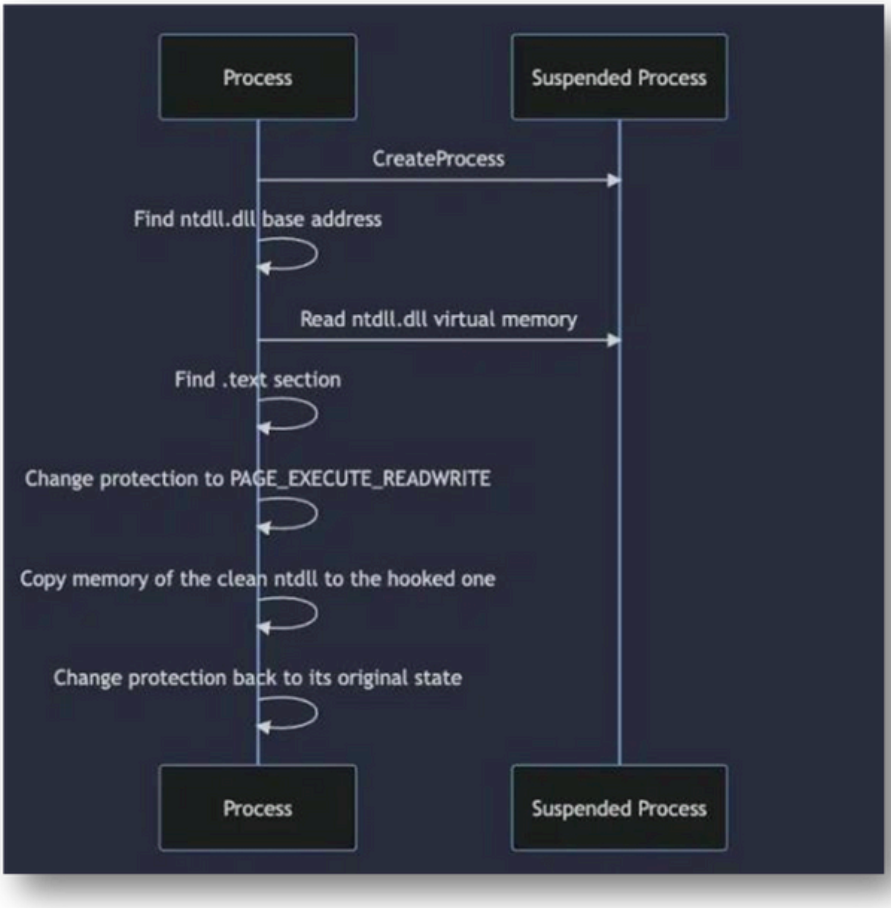


Figure 12 - Unhooking de la NTDLL depuisun processus suspendu

NBS System
Groupe CELESTE

CONTACTS

+33 1 58 56 60 80

www.nbs-system.com

contact@nbs-system.com