Livre blanc

Partie 2

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

NBS System
Groupe CELESTE

SOMMAIRE

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.

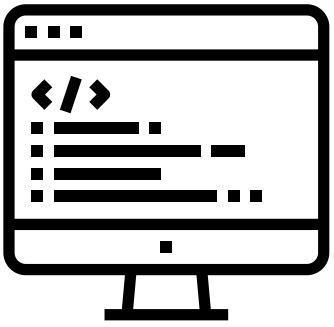
Ajout du template à la base centrale

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.

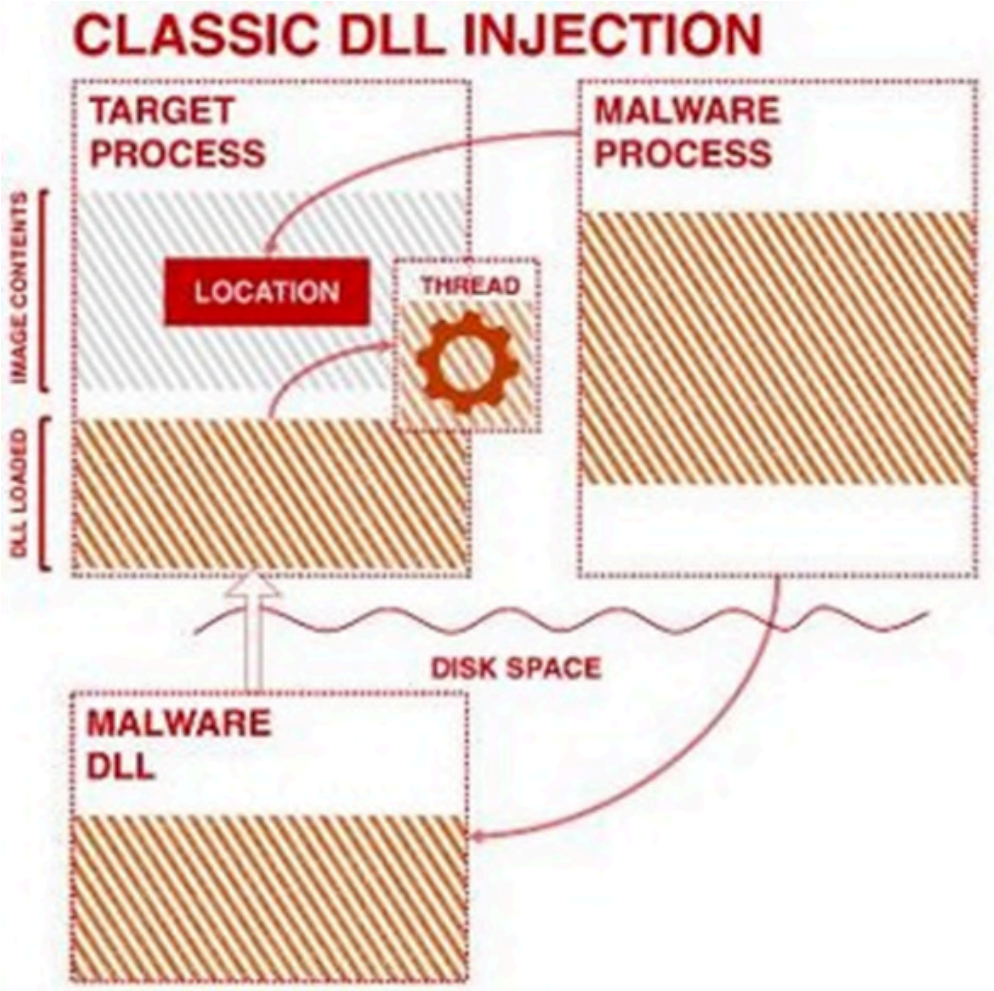


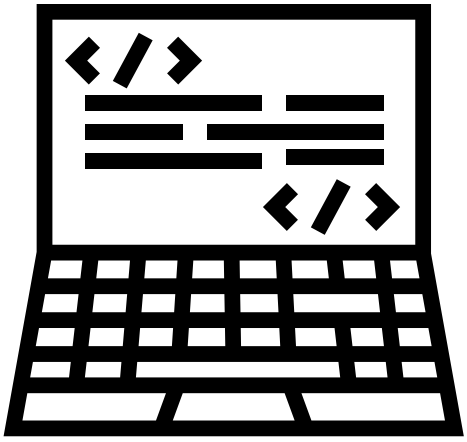
Figure 5 - Exemple d'injection DLL

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.