

Maximilien CHAUX
Madigan LEBRETON
Bertrand MARTIN
Alan PICARD

Développement d'une attaque informatique à but de sensibilisation basée sur un rootkit

Dossier de démonstration

Version 1

Sommaire

1. Introduction	3
1. Vocabulaire et terminologie	3
2. Rappels	4
1. Contexte	4
2. Les grandes lignes du scénario	4
3. Scénario d'attaque	4
3. Présentation globale de la solution	5
1. Cible de notre attaque	5
2. Architecture de la solution	5
4. Les composants de la solution	6
1. Package Livraison	6
2. Package Exploitation – Installation du package rootkit	7
Exécutable Userland - SpoolFool	7
Librairie Useland – Dropper	7
3. Package Rootkit	8
Cacher des fichiers	8
Application Userland	11
4. Package Command & Control	13
5. Taux de couverture de la conception	14
6. Sensibilisation	14
7. Conclusion	15

1. Introduction

Le dossier démonstrateur a pour objectif de décrire la solution développée, notamment sa conception et son fonctionnement. Pour ce faire, nous rappellerons brièvement les grandes lignes de notre projet, puis nous présenterons la solution proposée dans sa globalité. Ensuite, nous détaillerons les différents composants de cette solution, en présentant les concepts de leur fonctionnement ainsi que les limites de ces composants. Enfin, nous présenterons et justifierons les différents écarts de la solution que nous proposons par rapport au dossier de définition v0 réalisé au Jalon 3.

Il est à noter que des vidéos de démonstration sont fournies dans l'archive contenant notre démonstrateur V1. Puisque Visual Studio a été utilisé durant nos développements et pour faciliter le rendu technique, nous avons décidé de rendre les solutions Visual Studio.

1. Vocabulaire et terminologie

Termes	Définition
C2	Command & Control / Commande et Contrôle
Composant	Partie logicielle appartenant à un package, répondant à un objectif
Cyber Kill Chain	Méthode de modélisation des procédés d'intrusion des systèmes d'information, développée par Lockheed Martin
Kernel	Noyau en français. Ce terme sera utilisé pour qualifier des éléments (code, drivers, ...) s'exécutant au niveau ring 0 du système d'exploitation.
Minifilter	Driver kernel de type filtre de système de fichier sur Windows
Package	Ensemble de logiciels munis d'une documentation, conçus pour répondre à des besoins spécifiques et permettre une utilisation autonome
Userland / User mode	Ces termes seront utilisés pour qualifier des éléments (code, applications, ...) s'exécutant au niveau ring 3 du système d'exploitation.

2. Rappels

1. Contexte

- Dans notre projet, nous endossons le rôle d'une entreprise qui fait à la fois du pentest offensif et de la sensibilisation auprès des personnels techniques.
- Afin de maximiser l'impact du volet sensibilisation de notre projet, nous avons opté pour le secteur de la santé en créant un scénario d'attaque visant un laboratoire de biologie médicale afin d'extraire les données personnelles des patients de ce laboratoire.
- Inspiré d'une fuite de donnée réelle ayant touché des laboratoires français en 2021 + contexte de crise sanitaire globale + accélération de la numérisation des données médicales avec des impacts locaux (*un laboratoire de ma ville a été hacké*) et personnels (*mon dossier médical est accessible publiquement sur internet*)
- Responsabilisation des acteurs pour sécuriser les SI contenant des données personnelles
- Sensibilisation sur les attaques furtives, plus *low profile* que les attaques par ransomware.

2. Les grandes lignes du scénario

Le laboratoire de biologie médicale Vannalyz' d'une ville moyenne se fait acheter par le grand groupe HealthSIBS dans le secteur la santé en raison du succès de ses avancées en matière d'analyse des prélèvements et de sa croissance commerciale. Ce marché de l'analyse est très concurrentiel et plusieurs acteurs majeurs sont en compétition pour gagner des parts de marché.

HealthSIBS a missionné notre organisation pour réaliser une attaque sur son SI et les SI des laboratoires achetés par le groupe. Nous avons identifié le laboratoire Vannalyz' récemment acheté comme une potentielle porte d'entrée pour notre attaque.

3. Scénario d'attaque

Nous présentons notre scénario d'attaque en nous appuyant sur le Framework **Cyber Kill Chain**. Cela nous permet de présenter les informations ainsi que le déroulé de notre attaque de manière claire et globale.

Ce framework sera utilisé pour caractériser les différents composants de la solution que nous proposons. Ainsi, il sera plus simple de comprendre la chronologie de l'utilisation des composants.

- **Reconnaissance**
 - Etat de l'art sur les rootkits Windows et les vulnérabilités touchant Windows
 - OSINT / sources de risque et SPOF
- **Armement**
 - Développement du rootkit
- **Livraison**
 - Spearphishing : recherche sur les techniques de mail spoofing
 - Rubber Ducky
- **Exploitation**

- Exécution du programme seringue
- **Installation**
 - Installation et exécution du rootkit
- **Commande & Contrôle**
 - Command & Control HTTPS

3. Présentation globale de la solution

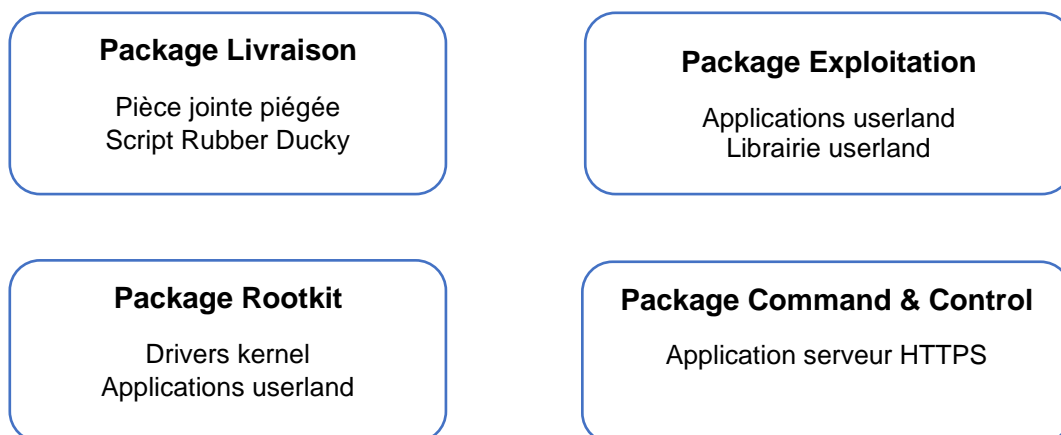
1. Cible de notre attaque

Pour développer notre attaque, nous avons dû sélectionner une cible à attaquer. Le système d'exploitation sélectionné est Windows 10, dans sa version 1909. Cette version de Windows est sortie environ deux ans et six mois avant l'écriture de ce document. Elle est donc relativement récente.

Cette version de Windows a été choisie car elle nous permet de charger des drivers kernel (noyau). Windows 10 requiert que les drivers noyau soient signés. Nous avons pu récupérer une paire clé/certificat ayant fuité, nous permettant de signer nos drivers et charger ces derniers. Or, le certificat ayant été révoqué par Microsoft, il est détecté sur les versions plus récentes que Windows 1909 et nous ne pouvons donc pas charger nos drivers sur ces versions.

2. Architecture de la solution

La solution que nous proposons se divise en quatre principaux packages, composés de multiples composants. La figure suivante présente les différents packages de notre solution.



Chaque package proposé s'inscrit dans une des étapes de la **Cyber Kill Chain**. Ainsi, nous pouvons présenter l'association suivante entre package et étape.

Etape de la Cyber Kill Chain	Package proposé
Livraison	Package Livraison
Exploitation	Package Exploitation
Installation	Package Rootkit
Commande & Contrôle	Package Command & Control

4. Les composants de la solution

Les composants des différents packages de la solution sont présentées dans la suite de ce document.

1. Package Livraison

Pour la livraison, nous avons choisi d'employer deux méthodes : le spearphishing ainsi qu'un rubber ducky.

Pour le spearphishing, nous avons choisi d'envoyer un fichier excel de la suite microsoft office avec une macro malveillante. Evidemment, cela signifie que nous avons besoin d'excel installé sur la machine. Cette macro se compose de deux méthodes, DownloadFile qui nous permet de télécharger un fichier à un chemin précis, et Payload qui utilise la fonction DownloadFile pour télécharger trois fichiers : SpoolFool.exe, dropper.exe et AddUser.dll qui seront détaillés plus loin. La fonction exécute ensuite la commande « SpoolFool.exe -dll AddUser.dll » qui exécutera ensuite notre dropper (voir package exploitation). Le téléchargement d'un fichier avec une extension .exe est considéré comme une action malveillante. Plusieurs solutions peuvent être employé pour passer sous les radars. Nous avons choisi de compresser les fichiers dans un .zip, de le télécharger sur la machine victime puis le décompresser, ce qui suffit à éviter la détection.

Évidemment, ces attaques sont de moins en moins vraisemblantes grâce à la sensibilisation sur les pièces jointes malicieuses. Cependant, cela reste un moyen relativement efficace pour entrer dans un système d'information pour le moment. Le niveau technique requis pour écrire une macro malveillante passant à travers AMSI n'est pas très élevé et un script kiddie pourrait réussir.

Pour la méthode utilisant le rubber ducky, nous avons choisi de récupérer les fichiers nécessaires sur la clé USB directement. Pour cela, nous avons choisi de flasher le firmware « Twin Duck » disponible sur le Github du projet. Ce firmware nous permet d'utiliser le rubber ducky en tant que clavier dissimulé mais aussi en tant que périphérique de stockage. Pour cela, nous avons renommé nos fichiers et les avons définis comme fichiers cachés pour tenter de les dissimuler du mieux possible. Une fois la clé branchée, il lance un powershell grâce à la commande WIN + R. Ce powershell récupère le chemin de la clé et exécute le script powershell de la clé qui va récupérer notre dropper et l'exécutable SpoolFool sur le stockage. La documentation disponible sur leur site est telle que le temps de programmation du rubber ducky est moindre et tout se fait rapidement avec des outils open sources. De même

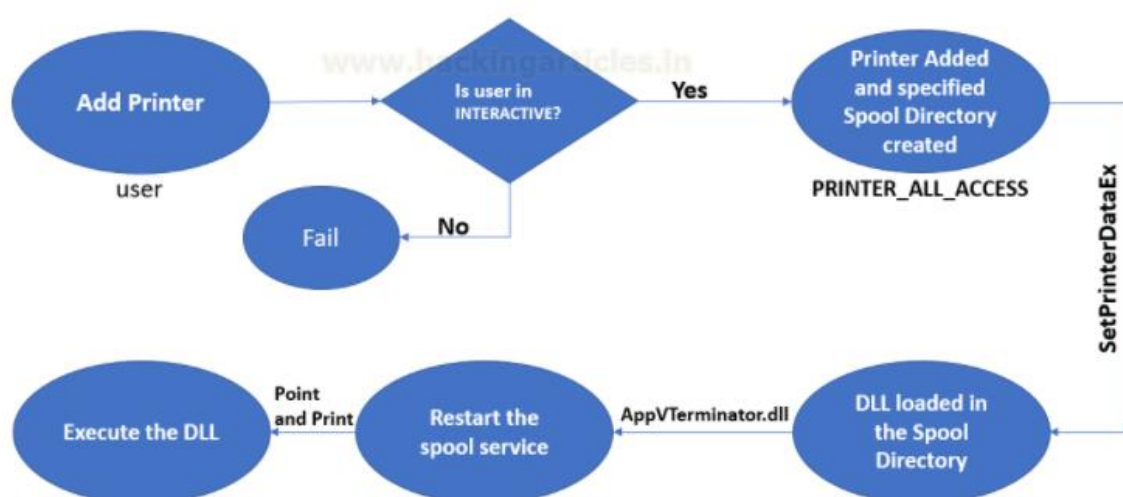
que la méthode par macro, la sensibilisation grandissante fait que les gens sont moins enclins à entrer une clé USB dans leur machine si celle-ci est trouvée.

2. Package Exploitation – Installation du package rootkit

Le package exploitation se décompose en 2 fichiers : l'un permettant de réaliser une élévation de privilège, le second, exécuté par le premier, s'attèle à télécharger tous les composants du package rootkit afin de les installer correctement.

Exécutable Userland - SpoolFool

C'est le premier exécutable qui est exécuté. Il utilise la vulnérabilité SpoolFool afin d'élever ses privilèges. Cette vulnérabilité exploite le service spouleur d'impression sur les machines Windows. Elle est très récente (février 2022) et se situe dans la lignée de PrintNightmare, vulnérabilité que nous avons envisagée dans le dossier de définition. Voici un schéma résumant le fonctionnement de la vulnérabilité :



Comme nous pouvons le voir, n'importe quel utilisateur appartenant au groupe INTERACTIVE (fréquent) peut ajouter une imprimante et obtenir les droits PRINTER_ALL_ACCESS. Il est alors possible de modifier le répertoire de spouleur en spécifiant un nouveau répertoire, ce qui aura pour effet de le créer avec les droits d'écriture pour tout le monde. On peut alors charger la DLL malicieuse dans ce répertoire et l'exécuter en tant que NT Autorite/Systems

Librairie Useland – Dropper

Cette DLL est une simple librairie possédant une fonction permettant de lancer l'exécutable dropper en tant que NT Autorite/Systems (Administrateur). Le dropper lui va permettre de télécharger tous les composants du package Rootkit depuis le serveur C&C, les installer correctement, et supprimer tous fichiers inutiles par la suite. Afin que nos driver fonctionnent, il nous faut amener sur la machine un certificat que le dropper se chargera d'enregistrer dans le store Windows. C'est avec ce certificat que nous avons signé nos driver. Une fois cela fait, nous pouvons installer correctement les drivers.

Nous avons décidé d'installer tous nos fichiers dans le dossier C:\Windows\System32\ServiceUtilities. Ainsi dans ce dossier apparaîtra le fichier de service du driver kernel permettant de cacher le processus, l'exécutable permettant de communiquer

avec le C2 et réalisé les tâches nécessaires sur la machine, ainsi que le fichier de service du minifilter driver permettant de cacher l'entièreté de ce dossier.

Nous utilisons le Service Control (sc) afin de configurer les services nécessaires pour nos driver.

Les 3 fichiers utilisés sont donc les suivants : Spoolfool.exe (exécutable permettant d'exploiter la faille), AddUser.dll (qui permet simplement de lancer le dropper) et dropper.exe (qui télécharge le package rootkit et l'installe)

3. Package Rootkit

Le package rootkit est la charge utile de notre exemple d'attaque utilisée pour la sensibilisation. Ce package se résume à 3 composants :

- Un driver kernel permettant de cacher des processus (abandonné)
- Un driver kernel de type minifilter permettant de cacher des fichiers
- Une application userland effectuant les communications avec le serveur Command&Control

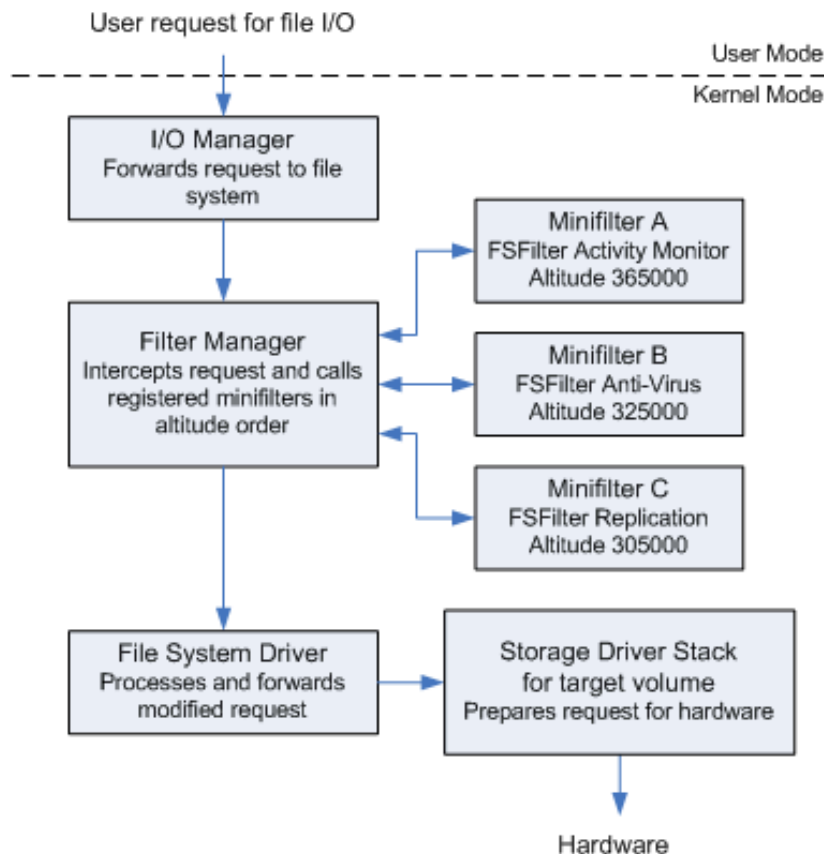
Cacher des fichiers

Windows propose un nouveau type de "file system filter driver", les "minifilter". Ceux-ci viennent remplacer l'ancien type "legacy file system filter model", apportant une simplicité du développement ainsi qu'une meilleure qualité et robustesse aux file system filter driver.

Fltmgr, le "filter manager" s'occupe de gérer les minifilters qui s'enregistrent sur le système de fichiers. Les minifilters s'enregistrent indirectement, et enregistrent des fonctions de callback sur des requêtes d'entrée sortie envoyées par le système d'exploitation. Ils peuvent enregistrer deux types de fonction :

- Preoperation callback function
- PostOperation callback function

Voici un schéma plus parlant pour détailler le système de requête d'entrée sortie :



Le schéma démarre par une requête d'entrée sortie. La requête arrive ensuite au filter manager qui va s'occuper d'appeler les fonctions de callback preoperation des différents minifilters qui en auront enregistré sur le type de requête demandé. L'ordre dans lequel ces fonctions seront appelées dépend de l'altitude du minifilter. Lors de son enregistrement, le minifilter déclare une altitude qui déterminera l'ordre de son appel par rapport aux autres minifilters. Deux minifilters ne peuvent avoir la même altitude.

Une fois toutes les fonctions de preoperations callback appelées, le filter manager envoie la requête au "file system driver" qui s'occupera de réaliser la requête sur le hardware directement.

Lors de la remontée, le filter manager appellera, dans l'ordre décroissant cette fois-ci, les fonctions de postoperation callback des différents minifilters.

Les requêtes sont divisées en deux "function code" : le "major function code" qui définit le type de requête demandé et le "minor function code" qui définit un sous type de cette même requête. Pour notre minifilter, le type de requête qui nous intéresse est "IRP_MJ_DIRECTORY_CONTROL" avec le "minor function code" "IRP_MN_QUERY_DIRECTORY". Cette requête est envoyée pour demander la liste des fichiers et sous dossiers d'un répertoire dans Windows, par exemple lors de la navigation dans le système de fichiers via l'explorateur de fichiers ou encore un appel à la fonction dir via cmd sur un dossier.

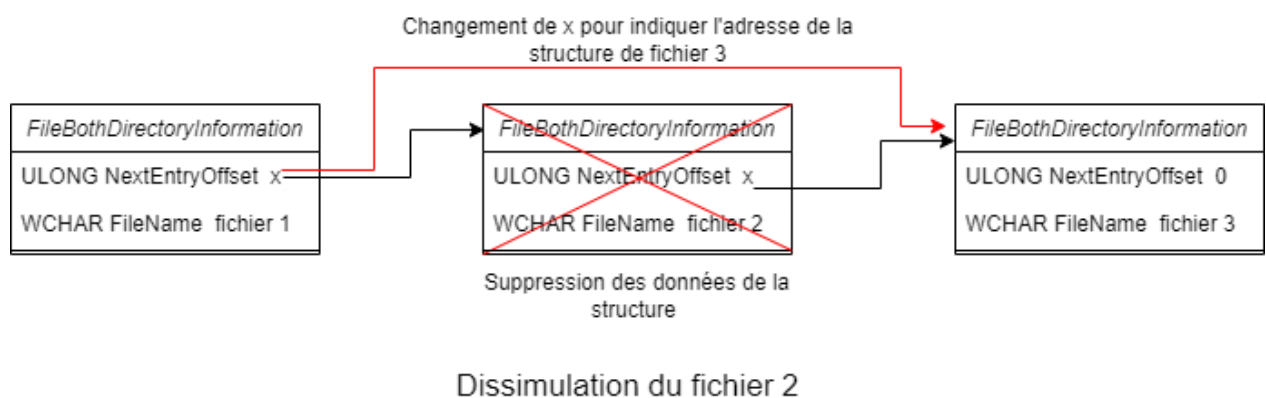
Cette requête retournera une liste de structure. Il y a en tout 9 types de structures retournés, une structure représentant un fichier à l'intérieur du dossier :

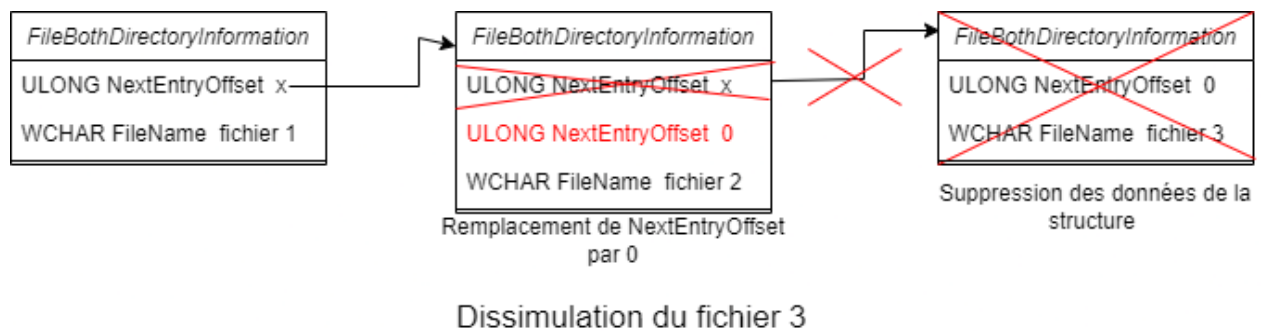
- FileBothDirectoryInformation
- FileDirectoryInformation
- FileFullDirectoryInformation
- FileIdBothDirectoryInformation
- FileIdFullDirectoryInformation
- FileNamesInformation
- FileObjectIdInformation
- FileQuotaInformation
- FileReparsePointInformation

Certaines sont plus complètes que d'autres, permettant de renvoyer plus d'informations sur les fichiers en question. Nous nous intéresserons seulement aux 6 premières car elles représentent les requêtes d'information sur les fichiers à l'intérieur d'un dossier, les 3 dernières étant différents et répondant à un autre besoin.

Lors de la remontée de la requête, un buffer est renvoyé contenant la liste des structures représentant les différents fichiers contenus dans le dossier. Le parcours de la liste, contrairement à la liste des structures EPROCESS expliquée plus haut, se fait au moyen d'un offset indiquant le début de la structure suivante.

Ainsi pour notre minifilter, nous avons enregistré une fonction de postoperation callback sur le major function code "IRP_MJ_DIRECTORY_CONTROL" et le minor function code "IRP_MN_QUERY_DIRECTORY". Nous vérifions le type de structure demandé pour ne traiter que les 6 premières présentées plus haut. Dans notre fonction de postoperation callback, le buffer contenant la liste des structures nous est envoyé. Il nous suffit de parcourir cette liste à la recherche des fichiers que l'on souhaite cacher et de supprimer les données à cette adresse dans le buffer. Deux cas de figure spécifiques se présentent à nous : lorsque le fichier se trouve au milieu de la structure et lorsqu'il se trouve en fin de structure. Des schémas seront plus parlants. Nous indiquerons seulement les champs des structures qui nous intéressent :





Il est à noter que le fichier ne peut jamais se retrouver au début de la liste, car il y aura toujours les dossiers symboliques "." et ".." qui occuperont les deux premières structures de la liste. C'est pour la même raison que le buffer renvoyé contenant la liste de structure ne sera jamais vide, même pour un dossier vide.

Ce système fonctionne, il permet de bien cacher les fichiers à tous les processus du système. Seulement, cela pourrait nous poser problème, notamment lorsque notre application Userland ou tout autre processus système aurait besoin d'accéder aux fichiers dissimulés pour fonctionner. Cela pourrait créer des crashes du noyau système. Nous avons donc pris la décision de dissimuler seulement les fichiers lorsque le processus réalisant la requête est soit l'explorateur de fichiers, soit le cmd. Cela est rendu possible à l'aide de la fonction `PsGetCurrentProcess()` qui nous renvoie la structure `EPROCESS` du processus courant, donc celui qui a initié la requête. Il nous suffit alors de récupérer le nom du processus dans un des champs de cette structure afin de vérifier quel est le processus ayant initié la requête. Cela nous a permis d'éviter de nombreux problèmes et de rendre notre minifilter robuste et stable, au prix d'une dissimulation amoindrie. Néanmoins, nous avons jugé que cette dernière serait suffisante et efficace.

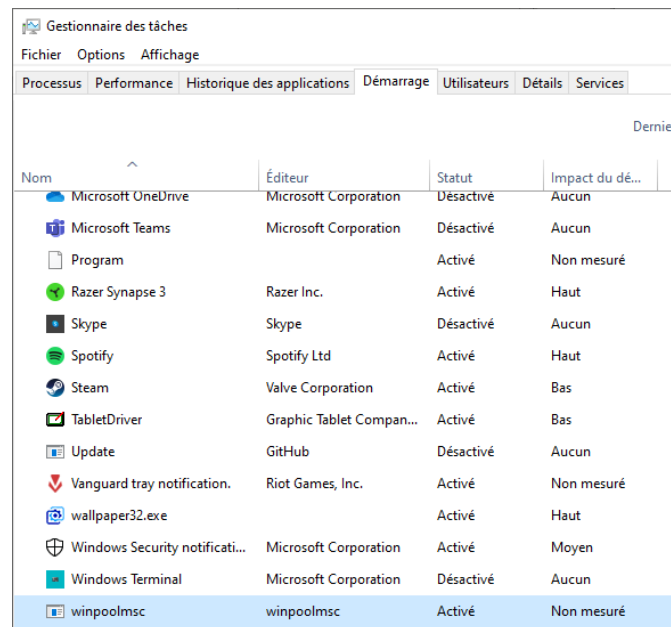
Dans le rendu technique, le driver sera nommé Winflt. Comme c'est un minifilter, nous utilisons le fichier Winflt.inf pour l'installer. Le fichier Winflt.sys sera le fichier utilisé pour le service, qui sera installé ici : `C:\Windows\System32\drivers\Winflt.sys`.

L'écriture d'un driver kernel prend du temps et nécessite des connaissances assez poussées ce qui demande un niveau technique assez élevé. Les informations de la documentation Windows sont cependant bien fournies mais difficiles à naviguer lorsque l'on travaille à un aussi bas niveau.

Application Userland

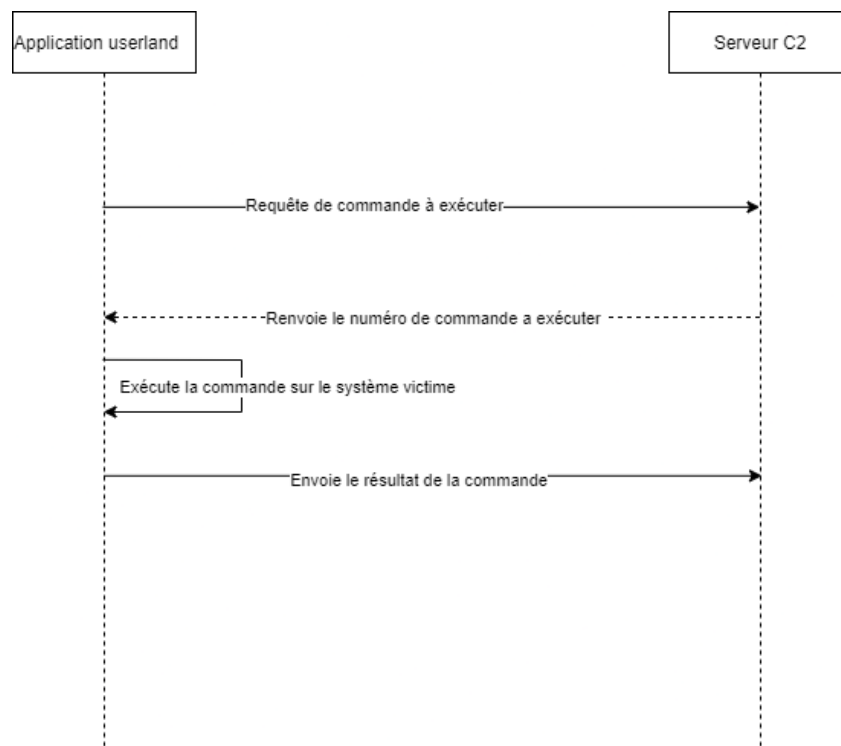
Pour faire le lien entre la machine infectée et notre serveur command and control, nous avons développé une application userland. Cette application nommée « winpoolmsc.exe » est une application .Net permettant d'interagir avec le driver « wintbk.sys » ainsi que le server C2. Nous avons choisi de développer ce dernier en C# pour plusieurs raisons, la première étant la facilité à écrire le code car étant familier avec ce dernier. De plus, le langage étant développé par Microsoft, nous pensions qu'il serait facile d'interagir avec l'OS et de faire des appels systèmes. Enfin, le langage nous permet de n'avoir qu'un seul exécutable portable une fois compilé.

Le fonctionnement de cette application est assez simple malgré quelques problèmes que nous avons rencontrés et qui seront expliqués dans la suite de ce document. Il se base sur une boucle infinie qui ponctuellement envoie une requête C2 pour savoir s'il doit exécuter une commande sur la machine victime (voir les commandes dans le package command & control). Avant de commencer cette boucle, le programme crée une clé de registre au chemin « Ordinateur\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run » permettant à l'application d'être lancée au démarrage. Il vérifie aussi que cette clé est à la valeur « true ».



Nom	Éditeur	Statut	Impact du démarrage
Microsoft OneDrive	Microsoft Corporation	Désactivé	Aucun
Microsoft Teams	Microsoft Corporation	Désactivé	Aucun
Program		Activé	Non mesuré
Razer Synapse 3	Razer Inc.	Activé	Haut
Skype	Skype	Désactivé	Aucun
Spotify	Spotify Ltd	Activé	Haut
Steam	Valve Corporation	Activé	Bas
TabletDriver	Graphic Tablet Compan...	Activé	Bas
Update	GitHub	Désactivé	Aucun
Vanguard tray notification.	Riot Games, Inc.	Activé	Non mesuré
wallpaper32.exe		Activé	Haut
Windows Security notificati...	Microsoft Corporation	Activé	Moyen
Windows Terminal	Microsoft Corporation	Désactivé	Aucun
winpoolmsc	winpoolmsc	Activé	Non mesuré

Nôtre processus se présente sous la forme d'une boucle principale. Le programme suivra le schéma suivant :



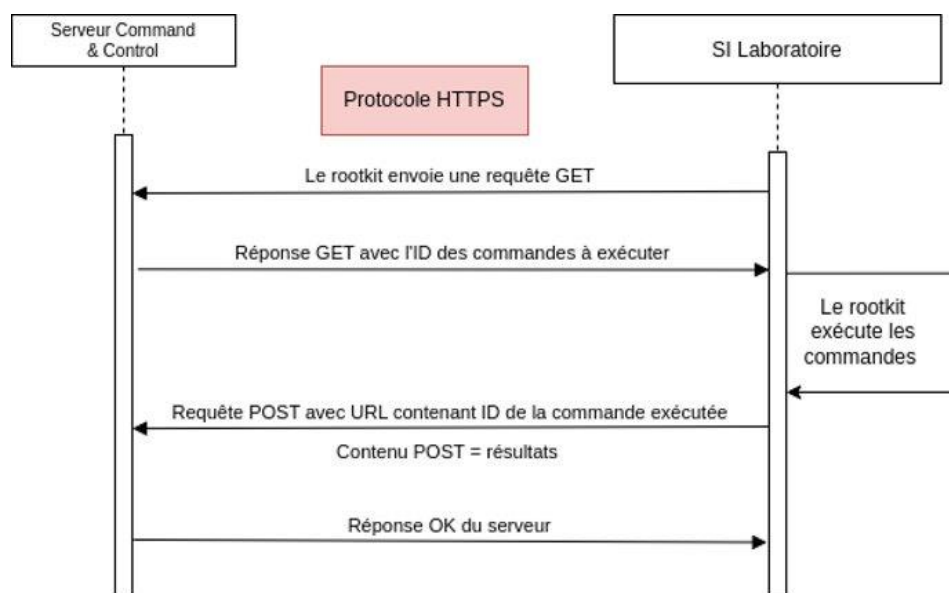
Cela se représente dans le code comme un gros switch exécutant la fonction correspondant au numéro de commande.

4. Package Command & Control

Le package Command & Control correspond à l'application serveur HTTPS avec laquelle l'application userland du package rootkit communiquera.

Notre choix s'est porté sur la plateforme NodeJS utilisant le langage de programmation Javascript. NodeJS nous permet de créer une application rapidement et facilement. De plus, ayant déjà réalisé des projets avec cette plateforme, nous avons pu gagner du temps sur cette application pour travailler sur le reste. Pour le rendu technique, le code n'étant pas compilé, celui-ci ne sera disponible que dans le dossier "Sources" du rendu.

Comme nous l'avons défini dans le dossier de définition, la communication entre le client et le serveur C2 fonctionnera de la manière suivante :



Notre application serveur HTTPS devra donc fournir les informations de commande au rootkit, puis récupérer et stocker les résultats de ces commandes reçus depuis le client (rootkit).

Aujourd'hui, nous avons implémenté 4 commandes côté serveur :

- Commande 1 : Informations génériques (Nom de la machine, Adresse MAC, Version de Windows, Nom de domaine, Adresse IP, état du driver, taille occupée du disque)
- Commande 2 : Chercher tous les fichiers contenant un certain mot-clé passé en paramètre
- Commande 3 : Transférer le fichier spécifié en paramètre du client vers le serveur
- Commande 4 : Scan de la plage réseau de la machine

Ces commandes sont passées en réponse à la requête GET. Du code correspondant à ces commandes est exécuté par le rootkit. Le résultat produit par le code précédemment cité sera ensuite envoyé au serveur dans une requête POST. Les résultats sont envoyés au format texte pour les commandes 1, 2 et 4. En revanche, pour les résultats de la commande 3, le contenu est encodé en Base64. En effet, les fichiers peuvent contenir des données binaires,

il est préférable de les encoder pour ne gérer que le transfert de données de type texte. Les résultats de cette commande seront donc décodés par notre serveur C2.

Enfin, les différents résultats reçus par notre application serveur seront stockés dans des fichiers situés dans des dossiers correspondant à la commande. Le nom des fichiers indiquera la date et l'heure de création de ces fichiers. Cela nous permet de garder un historique de toutes les communications entre notre serveur C2 et le rootkit. Toutes ces données récoltées pourront faire l'objet d'une analyse future.

5. Taux de couverture de la conception

Tâches	Phase 1 A faire	Phase 2 Début	Phase 3 En cours	Phase 4 Achevée
Sensibilisation			X	
Construction de l'architecture				X
Développement du package Livraison				X
Développement du package Exploitation				X
Développement du package Rootkit				X
Développement du package Command & Control				X
Déroulement de l'attaque / Mise en commun				X

A ce stade du projet, notre taux de couverture de la conception est de 96 %

Taux de couverture de la conception (%) = $[(\text{Tâches_achevées} \times 1 + \text{Tâches_en_cours} \times 0.75 + \text{Tâches_débutées} \times 0.25) / \text{Tâches_totales}] \times 100$

6. Sensibilisation

Le volet sensibilisation repose sur différents documents :

- Un poster à destination du public non-initié qui décrit notre attaque en particulier et comment s'en prémunir.
- Un rapport technique à destination d'un public informatique

Ces supports visent à atteindre ces trois objectifs principaux :

1. Présenter les différents types de rootkit
2. Mettre en avant les rootkits Windows en mode kernel
3. Mettre en évidence la comparaison entre les techniques d'attaques utilisées et les mesures de protection à mettre en œuvre, dans un contexte Red Team / Blue Team.

7. Conclusion

Pour la V1 de notre dossier démonstrateur, nous nous sommes concentrés sur la partie sensibilisation de notre projet. Nous avons pu corriger les derniers bugs de notre partie technique et améliorer les quelques points restants. Nous avons pu réaliser notre poster à destination d'un public non-initié, ainsi que commencer la réalisation de notre rapport technique à destination d'un public plus averti.