

Analyse de x86.exe

Etape 0: récupérer x86.exe

On peut utiliser volatility pour dumper tous les fichiers qui correspondent au PID de x86.exe dans la mémoire.

```
$ vol -f memory/data.lime windows.dumpfiles --pid 7616
```

Le fichier “*.x86.exe.img” correspond à l’image (l’exécutable) de x86.exe.

Étape 1: qu’est-ce que c’est ?

Si on reprend le cours, ne commencez pas par ouvrir le fichier dans IDA ! Le reverse statique manuel doit être la dernière étape, car elle est la plus coûteuse en temps.

On commence par chercher le hash du fichier sur VirusTotal: on peut récupérer le hash de deux manières

- Dans les logs Sysmon: on cherche le ProcessCreate (EventID=1) de x86.exe, et on lit le MD5
- On calcule le MD5 du fichier “IMG” extrait précédemment.

Attention ! Les deux hashes sont différents: c’est normal, la version qui tourne en mémoire est forcément différente: elle a subi des modifications du loader Windows pour la préparer à être exécutée.

Que vous cherchiez l’un ou l’autre des hashes sur VirusTotal, le résultat est le même: on a à faire à un beacon [CobaltStrike](#). CobaltStrike est un framework de post-exploitation commercial, un peu comme Metasploit mais en plus clé-en-main. Ce logiciel est vendu à destination des Red Teams, pour un usage légal, mais de nombreuses versions crackées tournent sur Internet et on le retrouve régulièrement utilisé par des acteurs malveillants.

VirusTotal vous extrait aussi la configuration du beacon, mais on va voir comment retrouver ça nous-même.

Etape 2: extraction de la configuration

Si vous ouvrez ce fichier dans Detect-It-Easy et que vous observez l’entropie des différentes sections, il devient évident que le programme est packé: l’entropie est proche du maximum. Certaines versions de DiE vous informent même du packer utilisé: [UPX](#), mais modifié.

[illegible]

Etape 3: comment x86.exe est-il arrivé ici ?

Dans les deux cas, on trouve que x86.exe a été lancé par [HWMonitor.exe](#), un logiciel qui surveille la température des composants d'un ordinateur.

Vous pouvez vérifier le hash de HWMonitor.exe, c'est un exécutable légitime ! Mais alors, pourquoi démarre-t-il un beacon CobaltStrike ?

- Créer un fichier "cpuz152_x64.sys" (sys = driver)
- Lancer la commande suivante: WMIC /
Namespace:\\root\\Microsoft\\Windows\\Defender class
MSFT_MpPreference call Add ExclusionPath = "C:\\Program
Files\\CPUID\\HWMonitor"

- Cette commande demande à Windows Defender d'exclure le chemin indiqué de ses analyses (c'est suspect)
- Ecrire x86.exe sur le disque
- Lancer x86.exe

De toute évidence, ce processus a été détourné pour lui faire faire des choses malveillantes. Si vous analysez "ToolsInstaller.ps1", vous verrez que HWMonitor est un des programmes qui est géré par le script: il est associé au chemin

"\$HOME\AppData\Local\Microsoft\WindowsApps\nvapi64.dll". Même si vous n'avez pas entièrement compris le script Powershell, ça peut valoir le coup de jeter un œil à cette DLL.

On la récupère depuis la RAM:

```
$ cat filescan.txt | fgrep nvapi64.dll
0xde88146c0bc0 \Users\Thomas\AppData\Local\Microsoft\WindowsApps\nvapi64.dll
```

```
$ ~/env/bin/vol -f memory/data.lime windows.dupfiles --virtaddr 0xde88146c0bc0
```

Un petit tour dans IDA nous montre que cette DLL récupère une ressource embarquée, ajoute le dossier courant aux exclusions Defender, déchiffre la ressource et écrit le résultat dans x86.exe. Enfin, elle exécute x86.exe...jackpot !

Une petite recherche sur Internet pour "nvapi64.dll" nous indique que cette DLL est normalement dans System32, elle n'a rien à faire ici...ceci est un exemple de [DLL sideloading](#): au moment de résoudre les imports DLL en chargeant un PE, le loader Windows va regarder dans une liste de dossier: dès qu'il trouve une DLL dont le nom correspond, il la charge et arrête de chercher. Il se trouve que "AppData\Local\Microsoft\WindowsApps" est plus haut dans la liste que System32, et est donc regardé avant.