AUG 13 2016

PowerShell Security: PowerShell Attack Tools, Mitigation, & Detection

By Sean Metcalf in Microsoft Security, PowerShell, Technical Reference

This post is a follow-up of sorts from my earlier posts on PowerShell, my PowerShell presentation at BSides Baltimore, and my presentation at DEF CON 24.
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
- PowerShell Version 5 is Available for Download (again)
- Detecting Offensive PowerShell Attack Tools
- PowerShell Version 5 Security Enhancements

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.

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

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

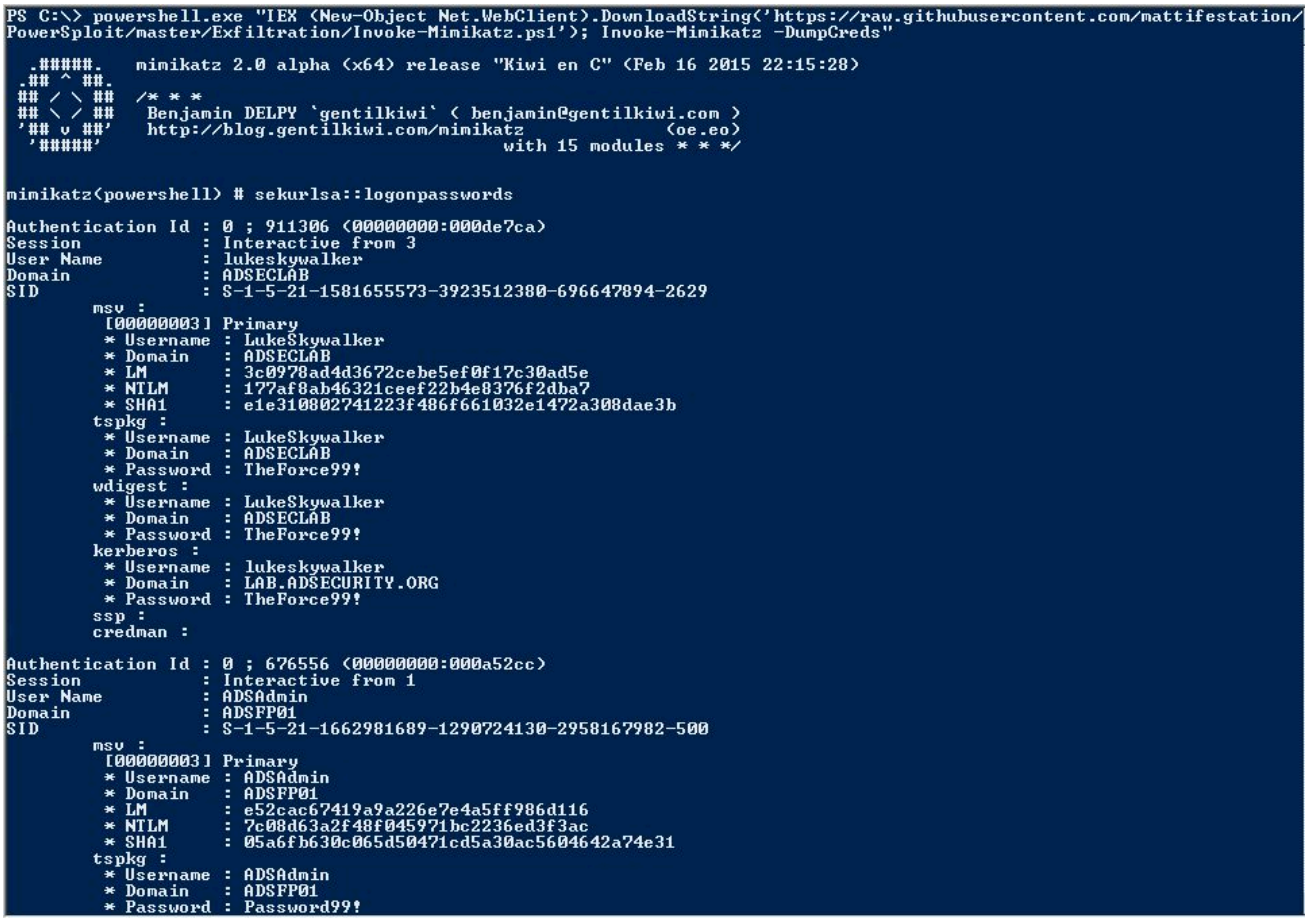
- Invoke-CredentialInjection.ps1
- Invoke-Mimikatz.ps1
- Invoke-NinjaCopy.ps1
- Invoke-TokenManipulation.ps1
- Out-Minidump.ps1
- VolumeShadowCopyTools.ps1
- Invoke-ReflectivePEInjection.ps1

Invoke-Mimikatz

Capabilities: Mimikatz execution from PowerShell, Credential theft & injection, Forged Kerberos ticket creation, Much more!

Use: Credential theft & reuse, Persistence

Author: Joseph Bialek (@clymb3r)



PowerView

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-ADObject
- 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  : {S-1-5-32-544}
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
```

PowerUp

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

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

Current Version: 1.5 (3/31/2016)

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 & extract. Once extracted, review PS1 files in data\module_source.

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) 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 (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

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".

- 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 by Lee Christensen (@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.

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. 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.

- **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.

- **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. 🙂

- 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.

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

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.

The second detection shows a detected threat in "PowerShell.exe_10.0.1058.0000000000010". Hmmm, that's odd.
It detected a threat in memory that was downloaded from the internet and executed in memory. 🙂

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.
- Matt Graeber described how to use reflection to bypass AMSI
  [Ref].Assembly.GetType('System.Management
  .Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)

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.

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

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

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.

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.

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

PS>Attack

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.

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

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

- Powersploit
  - Invoke-Mimikatz
  - Get-GPPPassword
  - Invoke-NinjaCopy
  - Invoke-Shellcode
  - Invoke-WMICommand

    - 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.

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. Event if PowerShell v5 is installed with system-wide transcript or script block logging.

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

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

Detecting custom EXEs calling PowerShell

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

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

- 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:

Note the addition of “KerberosRequestorSecurityToken” which is the PowerShell method to request Kerberos tickets (typically used for “Kerberoasting“).

Bypass PowerShell ExecutionPolicy, bypass PowerShell security, constrained language mode, Detect Invoke-Mimikatz, Detect offensive PowerShell, detect PowerShell attack tools, Detect PowerShell attacks, ExecutionPolicyBypass, Invoke-Expression, Invoke-Mimikatz, InvokeMimikatz, New-Object Net.WebClient DownloadString, Offensive PowerShell, Offensive PowerShell indicators, OffensivePowerShell, PowerShell Attack Tool, PowerShell attack tools, PowerShell Attacks, PowerShell constrained language, PowerShell Constrained Language Mode, PowerShell Execution Policy, PowerShell GPO, PowerShell Logging Group Policy, PowerShell Mimikatz, PowerShell.exe, PowerShellAttack, PowershellLogging, PowerShellMafia, PowerShellSecurity, PowerShellv5, PowerSploit, PowerUp, PowerView, script block logging, System-wide transcript, System.Management.Automation.dll, 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

2 comments

Avi Ron on *August 15, 2016 at 10:57 am* #

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 know.
Do you have a theory about his?

Sean Metcalf on *August 15, 2016 at 7:54 pm* #

Author

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

BSides Dublin – The Current State of Microsoft Identity Security: Common Security Issues and Misconfigurations – Sean Metcalf

DEFCON 2017: Transcript – Hacking the Cloud

Detecting the Elusive: Active Directory Threat Hunting

Detecting Kerberoasting Activity

Detecting Password Spraying with Security Event Auditing

POPULAR POSTS

PowerShell Encoding & Decoding (Base64)

Attack Methods for Gaining Domain Admin Rights in…

Kerberos & KRBTGT: Active Directory's…

Finding Passwords in SYSVOL & Exploiting Group…

Securing Domain Controllers to Improve Active…

Securing Windows Workstations: Developing a Secure Baseline

Detecting Kerberoasting Activity

Mimikatz DCSync Usage, Exploitation, and Detection

Scanning for Active Directory Privileges &…

Microsoft LAPS Security & Active Directory LAPS…

CATEGORIES

ActiveDirectorySecurity

Apple Security

Cloud Security

Continuing Education

Entertainment

Exploit

Hacking

Hardware Security

Hypervisor Security

Linux/Unix Security

Malware

Microsoft Security

Mitigation

Network/System Security

PowerShell

RealWorld

Security

Security Conference Presentation/Video

Security Recommendation

Technical Article

Technical Reading

Technical Reference

TheCloud

Vulnerability

TAGS

ActiveDirectory Active Directory Active Directory Security ActiveDirectorySecurity ADReading AD Security ADSecurity Azure AzureAD DCSync DomainController GoldenTicket GroupPolicy HyperV Invoke-Mimikatz KB3011780 KDC Kerberos KerberosHacking KRBTGT LAPS LSASS MCM MicrosoftEMET MicrosoftWindows mimikatz MS14068 PassTheHash PowerShell PowerShellCode PowerShellHacking PowerShellv5 PowerSploit Presentation Security SilverTicket SneakyADPersistence SPN TGS TGT Windows7 Windows10 WindowsServer2008R2 WindowsServer2012 WindowsServer2012R2

Search

RECENT POSTS

BSides Dublin – The Current State of Microsoft Identity Security: Common Security Issues and Misconfigurations – Sean Metcalf

DEFCON 2017: Transcript – Hacking the Cloud

Detecting the Elusive: Active Directory Threat Hunting

Detecting Kerberoasting Activity

Detecting Password Spraying with Security Event Auditing

RECENT COMMENTS

Derek on Attacking Read-Only Domain Controllers (RODCs) to Own Active Directory

Sean Metcalf on Securing Microsoft Active Directory Federation Server (ADFS)

Brad on Securing Microsoft Active Directory Federation Server (ADFS)

Joonas on Gathering AD Data with the Active Directory PowerShell Module

Sean Metcalf on Gathering AD Data with the Active Directory PowerShell Module

ARCHIVES

June 2024

May 2024

May 2020

January 2020

August 2019

March 2019

February 2019

October 2018

August 2018

May 2018

January 2018

November 2017

August 2017

June 2017

May 2017

February 2017

January 2017

November 2016

October 2016

September 2016

August 2016

July 2016

June 2016

April 2016

March 2016

February 2016

January 2016

December 2015

November 2015

October 2015

September 2015

August 2015

July 2015

June 2015

May 2015

April 2015

March 2015

February 2015

January 2015

December 2014

November 2014

October 2014

September 2014

August 2014

July 2014

June 2014

May 2014

April 2014

March 2014

February 2014

July 2013

November 2012

March 2012

February 2012

CATEGORIES

ActiveDirectorySecurity

Apple Security

Cloud Security

Continuing Education

Entertainment

Exploit

Hacking

Hardware Security

Hypervisor Security

Linux/Unix Security

Malware

Microsoft Security

Mitigation

Network/System Security

PowerShell

RealWorld

Security

Security Conference Presentation/Video

Security Recommendation

Technical Article

Technical Reading

Technical Reference

TheCloud

Vulnerability

META

Log in

Entries feed

Comments feed

WordPress.org

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.