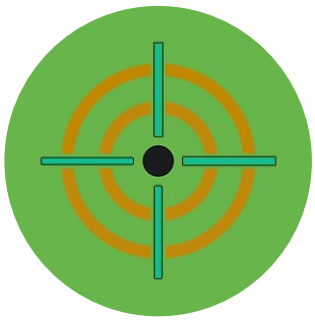
Share:



(/en_us/blog/author/secmrkt-research.html) By Splunk Threat Research Team (/en_us/blog/author/secmrkt-research.html) September 17, 2021

The Splunk Threat Research Team recently evaluated ways to generate security content using native Windows event logging regarding PowerShell Script Block Logging to assist enterprise defenders in finding malicious PowerShell scripts. This method provides greater depth of visibility as it provides the raw (entire) PowerShell script output. There are three sources that may enhance any defender's perspective: module, script block and transcript logging. We focused our security content on script block logging (4104) as it provides the most granular visibility of PowerShell scripts that execute on an endpoint. However, we also provided a way to gather all three for testing validation, production or curiosity.

Summary of Logging Types

- **Transcript Logging:** As described by Microsoft, Transcript Logging provides a summary of what's happening in a PowerShell session as if you were looking over the shoulder of the person typing. It will provide the commands and the output of those commands. This is fairly verbose in most organizations and would require filtering (via the Splunk UF, in Splunk) or logging only on critical assets. What does it look like?

```
Start time: 20210810204526
Username: ATTACKRANGE\administrator
RunAs User: ATTACKRANGE\administrator
Machine: WIN-DC-26 (Microsoft Windows NT 10.0.14393.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -nop -Command Write-Host 54797c2f-4d5b-44f6-8e0f-0048c8896ec2; Start-Sleep -Seconds 2; exit
Process ID: 6312
PSVersion: 5.1.14393.4530
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.4530
BuildVersion: 10.0.14393.4530
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
**********************
**********************
```

- **Module Logging:** module logging is a bit different in that it includes the command invocations and portions of the script. It's possible it will not have the entire details of the execution and the results. EventCode = 4103. What does it look like?

```
08/10/2021 08:04:35 PM
LogName=Microsoft-Windows-PowerShell/Operational
SourceName=Microsoft-Windows-PowerShell
EventCode=4103
EventType=4
Type=Information
ComputerName=win-dc-26.attackrange.local
User=NOT_TRANSLATED
Sid=S-1-5-21-4085333548-3215320843-1534856649-500
SidType=0
TaskCategory=Executing Pipeline
OpCode=To be used when operation is just executing a method
RecordNumber=81465
Keywords=None
Message=CommandInvocation(Write-Host): "Write-Host"
ParameterBinding(Write-Host): name="Object"; value="65540fb3-05d7-49a4-8565-7cfdd103bea7"


Context:
        Severity = Informational
        Host Name = ConsoleHost
        Host Version = 5.1.14393.4530
        Host ID = cf76ef16-4808-41ba-b83f-41f7c39a8ba7
        Host Application = C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -nop -Command Write-Host 65540fb3-05d7-49a4-8565-7cfdd103bea7; Start-Sleep -Seconds 2; exit
        Engine Version = 5.1.14393.4530
        Runspace ID = f7e926f9-726d-40a2-a82b-1ef736caa011
        Pipeline ID = 1
        Command Name = Write-Host
        Command Type = Cmdlet
        Script Name =
        Command Path =
        Sequence Number = 16
        User = ATTACKRANGE\administrator
        Connected User =
        Shell ID = Microsoft.PowerShell
```

- **Script Block Logging:** This is the raw, deobfuscated script supplied through the command line or wrapped in a function, script, workflow or similar. Think of everytime an adversary executes an encoded PowerShell script or command, script block logging provides that data in its raw form. EventCode = 4104. What does it look like?

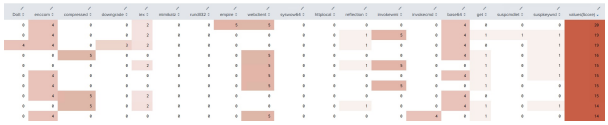
```
Event

08/10/2021 08:04:34 PM
LogName=Microsoft-Windows-PowerShell/Operational
SourceName=Microsoft-Windows-PowerShell
EventCode=4104
EventType=5
Type=Verbose
ComputerName=win-dc-26.attackrange.local
User=NOT_TRANSLATED
Sid=S-1-5-21-4085333548-3215320843-1534856649-500
SidType=0
TaskCategory=Execute a Remote Command
OpCode=On create calls
RecordNumber=81443
Keywords=None
Message=Creating Scriptblock text (1 of 1):
{
            # Only signal success if the parent process is mshta or rundll32
            $ParentProcessID = $EventArgs.NewEvent.TargetInstance.ParentProcessId

            $ParentProcess = Get-CimInstance -ClassName 'Win32_Process' -Filter "ProcessId = $ParentProcessID"
            $ExecutableFileInfo = Get-Item -Path $ParentProcess.ExecutablePath
            $ParentProcessCommandLine = $ParentProcess.CommandLine
            $ParentProcessPath = $ParentProcess.Path

            $SpawnedProcInfo = [PSCustomObject] @{
                ProcessId = $EventArgs.NewEvent.TargetInstance.ProcessId
                ProcessCommandLine = $EventArgs.NewEvent.TargetInstance.CommandLine
                ParentProcessId = $ParentProcessID
                ParentProcessCommandLine = $ParentProcessCommandLine
                ParentPath = $ParentProcessPath
            }

            if (@{'MSHTA.EXE', 'rundll'} -contains $ExecutableFileInfo.VersionInfo.InternalName) {
                # Signal that the child proc was spawned and surface the relevant into to Wait-Event
                New-Event -SourceIdentifier 'ChildProcSpawned' -MessageData $SpawnedProcInfo
            }
        }

ScriptBlock ID: 1e8344af-ec29-4edd-b109-6f1cd1d2bc0a
Path: C:\Users\Administrator\Documents\WindowsPowerShell\Modules\AtomicTestHarnesses\1.7.0.0\TestHarnesses\T1218.005_Mshta\InvokeHTMLApplication.ps1
```

Hunting Analytic

As we began generating content, we wanted a way to evaluate the dataset created to identify keywords that would ultimately convert to new analytics. We utilized all the standard frameworks in use; Empire, Cobalt Strike, Metasploit, Atomic Red Team and AtomicTestHarnesses. Script block provides a voluminous amount of data and we didn't want to be too selective on our keywords for the analytics we wanted to produce. With this release, we're publishing a hunting analytic that will assist with combing through 4104 event data. This detection is powered by the merging of two queries (Thank you Alex Teixeira (https://twitter.com/ateixei)) to assist with maximizing the identification of suspicious PowerShell usage. The detection may be found on our Security Content repository here (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_4104_hunting.yml).

```
`powershell` EventCode=4104
  | eval DoIt = if(match(Message,"(?i)(\$doit)"), "4", 0)
  | eval enccom=if(match(Message,"[A-Za-z0-9+\/]{44,}([A-Za-z0-9+\/]{4}|[A-Za-z0-9+\/]{3}=|[A-Za-z0-9+\/]{2}==)") OR
match(Message, "(?i)[-]e(nc*o*d*e*d*c*o*m*m*a*n*d*)*\s+[^-]"),4,0)
  | eval suspcmdlet=if(match(Message, "(?i)Add-Exfiltration|Add-Persistence|Add-RegBackdoor|Add-
ScrnSaveBackdoor|Check-VM|Do-Exfiltration|Enabled-DuplicateToken|Exploit-Jboss|Find-Fruit|Find-GPOLocation|Find-
TrustedDocuments|Get-ApplicationHost|Get-ChromeDump|Get-ClipboardContents|Get-FoxDump|Get-GPPPassword|Get-
IndexedItem|Get-Keystrokes|LSASecret|Get-PassHash|Get-RegAlwaysInstallElevated|Get-RegAutoLogon|Get-RickAstley|Get-
Screenshot|Get-SecurityPackages|Get-ServiceFilePermission|Get-ServicePermission|Get-ServiceUnquoted|Get-
SiteListPassword|Get-System|Get-TimedScreenshot|Get-UnattendedInstallFile|Get-Unconstrained|Get-VaultCredential|Get-
VulnAutoRun|Get-VulnSchTask|Gupt-Backdoor|HTTP-Login|Install-SSP|Install-ServiceBinary|Invoke-ACLScanner|Invoke-
ADSBackdoor|Invoke-ARPScan|Invoke-AllChecks|Invoke-BackdoorLNK|Invoke-BypassUAC|Invoke-CredentialInjection|Invoke-
DCSync|Invoke-DllInjection|Invoke-DowngradeAccount|Invoke-EgressCheck|Invoke-Inveigh|Invoke-InveighRelay|Invoke-
Mimikittenz|Invoke-NetRipper|Invoke-NinjaCopy|Invoke-PSInject|Invoke-Paranoia|Invoke-PortScan|Invoke-PoshRat|Invoke-
PostExfil|Invoke-PowerDump|Invoke-PowerShellTCP|Invoke-PsExec|Invoke-PsUaCme|Invoke-ReflectivePEInjection|Invoke-
ReverseDNSLookup|Invoke-RunAs|Invoke-SMBScanner|Invoke-SSHCommand|Invoke-Service|Invoke-Shellcode|Invoke-
Tater|Invoke-ThunderStruck|Invoke-Token|Invoke-UserHunter|Invoke-VoiceTroll|Invoke-WScriptBypassUAC|Invoke-
WinEnum|WailRaider|New-HoneyHash|Out-Minidump|Port-Scan|PowerBreach|PowerUp|PowerView|Remove-Update|Set-
MacAttribute|Set-Wallpaper|Show-TargetScreen|Start-CaptureServer|VolumeShadowCopyTools|NEEEEWWW|
(Computer|User)Property|CachedRDPConnection|get-net\S+|invoke-\S+hunter|Install-Service|get-\S+
(credent|password)|remoteps|Kerberos.*(policy|ticket)|netfirewall|Uninstall-
Windows|Verb\s+Runas|AmsiBypass|nishang|Invoke-
Interceptor|EXEonRemote|NetworkRelay|PowerShellUdp|PowerShellIcmp|CreateShortcut|copy-vss|invoke-dll|invoke-
mass|out-shortcut|Invoke-ShellCommand"),1,0)
  | eval base64 = if(match(lower(Message),"frombase64"), "4", 0)
  | eval empire=if(match(lower(Message),"system.net.webclient") AND match(lower(Message), "frombase64string") ,5,0)
  | eval mimikatz=if(match(lower(Message),"mimikatz") OR match(lower(Message), "-dumpcr") OR match(lower(Message),
"SEKURLSA::Pth") OR match(lower(Message), "kerberos::ptt") OR match(lower(Message), "kerberos::golden") ,5,0)
  | eval iex = if(match(lower(Message),"iex"), "2", 0)
  | eval webclient=if(match(lower(Message),"http") OR match(lower(Message),"web(client|request)") OR
match(lower(Message),"socket") OR match(lower(Message),"download(file|string)") OR
match(lower(Message),"bitstransfer") OR match(lower(Message),"internetexplorer.application") OR
match(lower(Message),"xmlhttp"),5,0)
  | eval get = if(match(lower(Message),"get-"), "1", 0)
  | eval rundll32 = if(match(lower(Message),"rundll32"), "4", 0)
  | eval suspkeywrd=if(match(Message, "(?i)
(bitstransfer|mimik|metasp|AssemblyBuilderAccess|Reflection\.Assembly|shellcode|injection|cnvert|shell\.application|
start-
process|Rc4ByteStream|System\.Security\.Cryptography|lsass\.exe|localadmin|LastLoggedOn|hijack|BackupPrivilege|ngrok
|comsvcs|backdoor|brute.?force|Port.?Scan|Exfiltration|exploit|DisableRealtimeMonitoring|beacon)"),1,0)
  | eval syswow64 = if(match(lower(Message),"syswow64"), "3", 0)
  | eval httplocal = if(match(lower(Message),"http://127.0.0.1"), "4", 0)
  | eval reflection = if(match(lower(Message),"reflection"), "1", 0)
  | eval invokewmi=if(match(lower(Message), "(?i)(wmiobject|WMIMethod|RemoteWMI|PowerShellWmi|wmicommand)"),5,0)
  | eval downgrade=if(match(Message, "(?i)([-]ve*r*s*i*o*n*\s+2)") OR match(lower(Message),"powershell -
version"),3,0)
  | eval compressed=if(match(Message, "(?i)GZipStream|::Decompress|IO.Compression|write-zip|(expand|compress)-
Archive"),5,0)
  | eval invokecmd = if(match(lower(Message),"invoke-command"), "4", 0)
  | addtotals fieldname=Score DoIt, enccom, suspcmdlet, suspkeywrd, compressed, downgrade, mimikatz, iex, empire,
rundll32, webclient, syswow64, httplocal, reflection, invokewmi, invokecmd, base64, get
  | stats values(Score) by DoIt, enccom, compressed, downgrade, iex, mimikatz, rundll32, empire, webclient,
syswow64, httplocal, reflection, invokewmi, invokecmd, base64, get, suspcmdlet, suspkeywrd
  | `powershell_4104_hunting_filter`
```

As we played with the data more and more, we have found that it's even more useful by adding scores to each keyword. Keywords in this case are each `eval`. In this instance, the scoring is based on fidelity. $DoIt is a function that Cobalt Strike uses, therefore the score is set to 4. Keywords like IEX are more commonly used and I've set the score to 2. An example of the scoring used in the following capture showcases how the scores can help bring up interesting PowerShell scripts. It is also easy enough to copy and paste an eval statement and add new keywords. Our example is not exhaustive, but a starting point for defenders to begin digging deeper.



Detections

Following our research effort, we were able to compile a good amount of new analytics. We hope this inspires others to contribute (via GitHub Issues (https://github.com/splunk/security_content/issues) or PR (https://github.com/splunk/security_content/pulls)) to continue to enhance coverage for the community.

Analytic	Technique	Tactic
Detect Empire with PowerShell Script Block Logging (https://github.com/splunk/security_content/blob/develop/detections/endpoint/detect_empire_with_powershell_script_block_logging.yml)	T1059.001	Execution (https://attack.mitre.org/techniques/T105
Detect Mimikatz With PowerShell Script Block Logging (https://github.com/splunk/security_content/blob/develop/detections/endpoint/detect_mimikatz_with_powershell_script_block_logging.yml)	T1059.001	Execution (https://attack.mitre.org/techniques/T105
Powershell Fileless Process Injection via GetProcAddress (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_fileless_process_injection_via_getprocaddress.yml)	T1059.001, T1055	Execution (https://attack.mitre.org/techniques/T105 Defense Evasion, Privilege Escalation (https://attack.mitre.org/techniques/T105
Powershell Fileless Script Contains Base64 Encoded Content (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_fileless_script_contains_base64_encoded_content.yml)	T1059.001, T1027	Execution (https://attack.mitre.org/techniques/T105
Unloading AMSI via Reflection (https://github.com/splunk/security_content/blob/develop/detections/endpoint/unloading_amsi_via_reflection.yml)	T1562	Defense Evasion (https://attack.mitre.org/techniques/T156
PowerShell Domain Enumeration (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_domain_enumeration.yml)	T1059.001	Execution (https://attack.mitre.org/techniques/T105
PowerShell Loading .NET into Memory via System.Reflection.Assembly (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_loading_dotnet_into_memory_via_system_reflection_assembly.yml)	T1059.001	Execution (https://attack.mitre.org/techniques/T105
Powershell Creating Thread Mutex (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_creating_thread_mutex.yml)	T1027.005	Defense Evasion (https://attack.mitre.org/techniques/T102
Powershell Processing Stream Of Data (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_processing_stream_of_data.yml)	T1059.001	Execution (https://attack.mitre.org/techniques/T105
Powershell Using memory As Backing Store (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_using_memory_as_backing_store.yml)	T1140	Defense Evasion (https://attack.mitre.org/techniques/T114
Recon AVProduct Through Pwh or WMI (https://github.com/splunk/security_content/blob/develop/detections/endpoint/recon_avproduct_through_pwh_or_wmi.yml)	T1592	Reconnaissance (https://attack.mitre.org/techniques/T159
Recon Using WMI Class (https://github.com/splunk/security_content/blob/develop/detections/endpoint/recon_using_wmi_class.yml)	T1592	Reconnaissance (https://attack.mitre.org/techniques/T159

WMI Recon Running Process or Services (https://github.com/splunk/security_content/blob/develop/detections/endpoint/wmi_recon_running_process_or_services.yml)	T1592	Reconnaissance (https://attack.mitre.org/techniques/T159
Allow Inbound Traffic In Firewall Rule (https://github.com/splunk/security_content/blob/develop/detections/endpoint/allow_inbound_traffic_in_firewall_rule.yml)	T1021.001	Lateral Movement (https://attack.mitre.org/techniques/T102
Mailsniper Invoke functions (https://github.com/splunk/security_content/blob/develop/detections/endpoint/mailsniper_invoke_functions.yml)	T1114.001	Collection (https://attack.mitre.org/techniques/T111
Delete ShadowCopy With PowerShell (https://github.com/splunk/security_content/blob/develop/detections/endpoint/delete_shadowcopy_with_powershell.yml)	T1490	Impact (https://attack.mitre.org/techniques/T149
Powershell Enable SMB1Protocol Feature (https://github.com/splunk/security_content/blob/develop/detections/endpoint/powershell_enable_smb1protocol_feature.yml)	T1027.005	Defense Evasion (https://attack.mitre.org/techniques/T102
Detect WMI Event Subscription Persistence (https://github.com/splunk/security_content/blob/develop/detections/endpoint/detect_wmi_event_subscription_persistence.yml)	T1546.003	Privilege Escalation, Persistence (https://attack.mitre.org/techniques/T154

How to Enable It?

There are three effective ways to enable PowerShell Logging. Depending upon the deployment method or if needing to deploy across a large fleet, the registry or Group Policy will be the best bet. If testing in a lab setting, all three methods following will help.

Registry

This method may be useful if using a deployment or logon script.

- Enable ScriptBlock Logging
  - HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
    - EnableScriptBlockLogging = 1
- Enable Module Logging
  - HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging
- EnableModuleLogging = 1
  - HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging \ModuleNames
    - * = *
- Transcription
  - HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription\
    - EnableInvocationHeader = 1
    - EnableTranscripting = 1
    - OutputDirectory = <path_to_directory>

The PowerShell Operational Log may be found here:

%SystemRoot%\system32\winevt\logs\Microsoft-Windows-PowerShell%4Operational.evtx

PowerShell

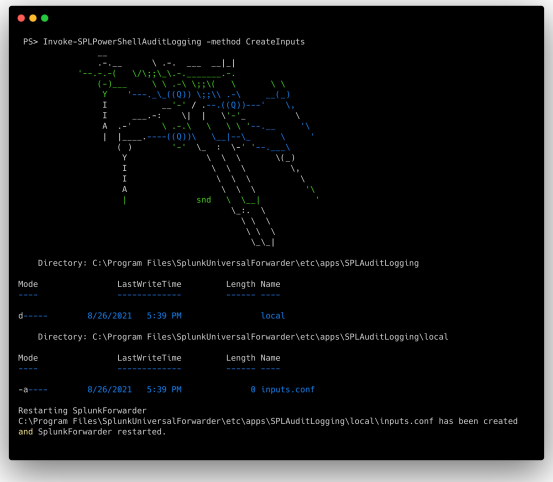
In any case, Hurricane Labs also provided a handy script (https://github.com/timip/splunk/blob/master/powershell_logging.ps1) that we have borrowed to help even further. We enhanced it with the following abilities:

- Enable one or all PowerShell logging methods
- Create a new inputs.conf to capture transcript logs and PowerShell Operational logs
- Disable all logging
- Enables Process Creation with Command-Line (4688)

Get Invoke-SPLPowerShellAuditLogging here (https://github.com/MHaggis/notes/blob/master/utilities/Invoke-SPLPowerShellAuditLogging.ps1).

Update a currently used Windows inputs.conf on the Splunk Universal Forwarder or use Invoke-SPLPowerShellAuditLogging to create the inputs.

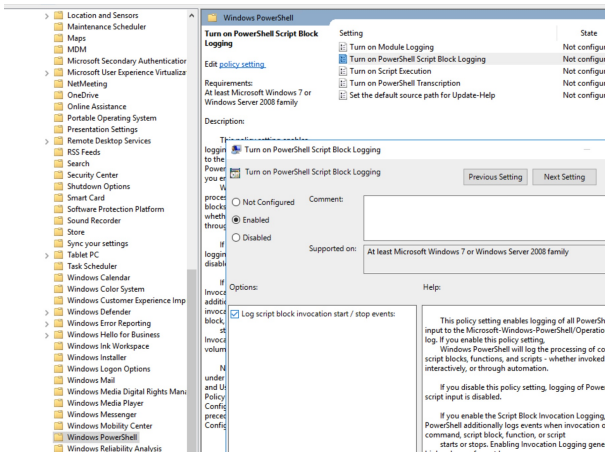
```
[WinEventLog://Microsoft-Windows-PowerShell/Operational]
source = XmlWinEventLog:Microsoft-Windows-PowerShell/Operational
renderXml = 0
disabled = false
index = win
[monitor://c:\pstransactions\]
sourcetype = powershell:transcript
disabled = false
multiline_event_extra_waittime = true
time_before_close = 300
index = win
`Invoke-SPLPowerShellAuditLogging -method CreateInputs`
```



Enable Logging via Group Policy Objects

For a more enterprise and granular policy deployment approach, within the Group Policy Management Console, create a new or modify an existing object, browse to Computer Configuration > Policies > Administrative Templates > Windows Components > Windows PowerShell

From here, enable the policies of interest and begin logging. Deploy to critical assets or all as needed.



This work was inspired by many others who have written about PowerShell Logging, but not limited to:

- Get Data into Splunk User Behavior Analytics (https://docs.splunk.com/Documentation/UBA/5.0.4.1/GetDataIn/AddPowerShell) - PowerShell logging
- Blueteam Powershell Recommendations (https://github.com/marcurdy/dfir-toolset/blob/master/Powershell%20Blueteam.txt)
- PowerShell ♥ the Blue Team (https://devblogs.microsoft.com/powershell/powershell-the-blue-team/) - Microsoft
- about_Logging (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logging?view=powershell-5.1) - Microsoft Docs
- Greater Visibility Through PowerShell Logging - FireEye (https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html)
- How to Use PowerShell Transcription Logs in Splunk (https://hurricanelabs.com/splunk-tutorials/how-to-use-powershell-transcription-logs-in-splunk/) - Hurricane Labs
- Hurricane Labs Add-on for Windows PowerShell Transcript (https://splunkbase.splunk.com/app/4984/)
- How to detect suspicious PowerShell activity with Splunk? (https://github.com/inodee/threathunting-spl/blob/master/hunt-queries/powershell_qualifiers.md) **-** Alex Teixeira (https://twitter.com/ateixei)

Test Yourself

Atomic Red Team: Using Atomic Red Team, we can simulate PowerShell commands simply using Invoke-AtomicRedTeam (https://github.com/redcanaryco/invoke-atomicredteam/). To begin, check out the Wiki (https://github.com/redcanaryco/invoke-atomicredteam/wiki/Installing-Atomic-Red-Team#install-execution-framework-and-atomics-folder) and follow along.

In a lab setting, or authorized device, run the following commands to get started:

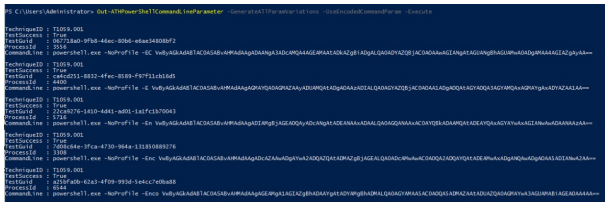
```
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/install-atomicredteam.ps1' -UseBasicParsing);
    Install-AtomicRedTeam -getAtomics -force
```

This will install Invoke-AtomicRedTeam. From here, we may now run T1059.001 (https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1059.001/T1059.001.md) from Atomic Red Team.

```
invoke-AtomicTest T1059.001
```

Want some more data? Check out AtomicTestHarnesses (https://github.com/redcanaryco/AtomicTestHarnesses).

```
Out-ATHPowerShellCommandLineParameter -GenerateAllParamVariations -UseEncodedCommandParam -Execute
```



Learn More

To learn more, watch the on-demand Splunk Tech Talk "Hunting for Malicious PowerShell using Script Block Logging (https://events.splunk.com/malicious-powershell)" now.

POSTED BY

Splunk Threat Research Team (/en_us/blog/author/secmrkt-research.html)

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository (https://github.com/splunk/attack_data/).

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content (https://github.com/splunk/security_content).