Detecting CONTI CobaltStrike Lateral Movement Techniques - Part 1 | Cyb3rSn0rlax

Published: 2021-11-08 · Archived: 2026-04-05 19:57:51 UTC

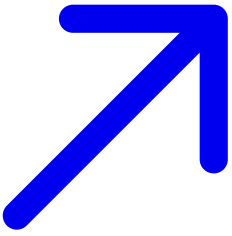
1. ☢️ DEATH : Detection Engineering And Threat Hunting

2. 🦘TA0008 : Lateral Movement

Detecting CONTI CobaltStrike Lateral Movement Techniques - Part 1

Detection opportunities on lateral movement techniques used by CONTI ransomware group using CobaltStrike.

In an attempt to contribute to the defensive capabilities of security teams regarding the increase of CobaltStrike usage by threat actors (TA) and in a joined effort with @MichalKoczwara



, a series of articles will be released on CobaltStrike's TTP detections related to the CONTI leak.

For the first part of this blog post, I will cover detection opportunities for lateral movement (LM) techniques used by the TA **CONTI** via CobaltStrike. Keep in mind that I tried to boil it down to analytics that can be used for other lateral movements variation and not just specific to CONTI Group or CobaltStrike (CS).

MITRE ATT&CK defines lateral movement as :

> *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.*

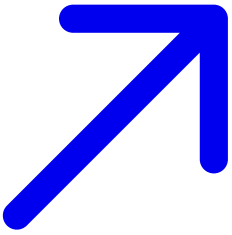
Looking in the CobaltStrike documentation we can find some built-in modules for Lateral Movement defined in the table bellow which were included in the leaked documentation:

Other capabilities are used by the group like `Remote-Exec` command, PTH module, RDP and `SHELL` command to remotely execute commands using `WMIC.EXE` utility. I will go through these TTPs in the second part.

-
-
- Elastic Stack (Winlogbeat + Filebeat)
-
- VICTIM Windows 10 user machine (Initial Access)
- DC_ATLAS Domain Controller Windows Server 2016 (Lateral Movement Target)

T1021.006 Remote Services: Windows Remote Management

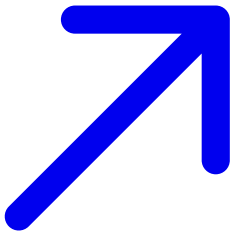
WinRM is the Microsoft implementation of **WS-Management** protocol which is an open source standard for constructing XML messages following the standards of Simple Object Access Protocol (SOAP) messages.



This great blog explain in simple steps a typical WinRM based conversation for invoking commands:

1.
2.
3.
4. Keep requesting output until the command state is done and examine the exit code.
5.
6.

I will go more in depth about WinRM from a defensive perspective during lateral movement in a separate blog but for more details I recommend checking the official documentation [MS-WSMV



]. However, a couple of things we should keep in mind when it come to the limitations of WinRM and why PowerShell Remoting Protocol (PSRP) is much better choice to go with.

The default value of a SOAP message size **512KB** and a maximum of **8192KB.** This attribute can be modified with the following command : `winrm set winrm/config/winrs '@{<Quota>="<Value>"}'` .

WinRM also doesn't have a built-in functionality for file transfer. We will learn in the next section that PowerShell Remoting Protocol (PSRP) is much better alternative.

Windows Built-in WinRM tools

In order to understand CobaltStrike WinRM beacon capabilities, first, I tried to see normal behavior of some of the tools that can be used in a legitimate way. There are 3 main ways to execute command remotely using WinRM:

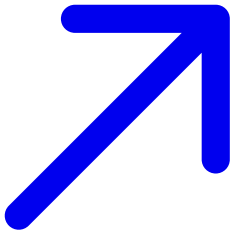
Windows Remote Shell built-in tool is a pure implementation of remote command execution via WinRM. Upon executing a command using winrs.exe utility via the command `winrs -r:dc_atlas "ipconfig"` the following telemetry was recorded on the destination:

- `svchost.exe` spawns `winrshost.exe` with the parent command line `C:\\Windows\\system32\\svchost.exe -k DcomLaunch`
- The `winrshost.exe` then invokes `cmd.exe` instance and execute the command within its context.

After finishing the execution of the command these processes are terminated because `winrs.exe` doesn't support persistent sessions so every time you execute a command remotely this behavior repeats itself.

Invoke-Command & Enter-PSSession :

These PowerShell cmdlets use the PowerShell Remoting Protocol [MS-PSRP



] which is a separate protocol that runs over WinRM. PSRP supports many message types to execute commands and retrieve their outputs and its main difference from

WSMV specs is its message fragmentation handling process which makes it more reliable vis-à-vis WinRM message size limitations.

While testing these cmdlets, the following telemetry was recorded on the destination:

- `svchost.exe` spawns wsmprovhost.exe with the parent command line `C:\Windows\system32\svchost.exe -k DcomLaunch`
- `Invoke-Command` & `Enter-PSSession` both run commands within the context of `wsmprovhost.exe`

The difference between these two cmdlets is that `Invoke-Command` will terminate `wsmprovhost.exe` process after receiving the output while the `Enter-PSSession` will establish a persistent session.

Now that we have established what telemetry can be left behind by using Windows built-in tools we can distinguish suspicious process behavior. lets see in the following section how CS default configurations for lateral movement behave.

First, lets discover the telemetry that will be generated from source and destination for every attempt to use WinRM remotely:

-
-

Other events are generated on the destination side but these in the previous table are the most relevant to remote WinRM activity. You can use them according to your collection and correlation strategy. Obviously, `EID 1`, `EID 91` and `EID 4656` have much higher event decisiveness than the rest. I will be releasing a Mindmap that groups all this telemetry in one place at the end of this blog post series.

Now jumping to `jump winrm` command and some first differences in process tree behavior were observed at execution time:

- `jump winrm` command generated the same telemetry as in previous observations except that the beacon runs under the context of a PowerShell instance invoked by `wsmprovhost.exe`. This is not something we can normally observe by using `winrs`, `Invoke-Command` or `Enter-PSSession` except if the command invoked `powershell.exe` itself then PowerShell cmdlets would produce this behavior.
- By default the powershell.exe instance run via the command line : `"c:\windows\syswow64\windowspowershell\v1.0\powershell.exe" -Version 5.1 -s -NoLogo -NoProfile`
- CobalStrike provides a `shell` command to interact with the beacon and execute command. The `shell` command spawns a `cmd.exe` instance from the invoked `powershell.exe` process for every executed command

A general diagram of process tree observed during the execution of this CS module is illustrated bellow:

CobaltStrike jump winrm64

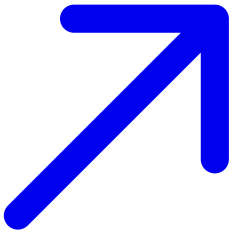
Here are the main differences from `jump winrm` command :

- Like `Enter-PSSession` , `jump winrm64` executes commands within the context of a `wsmprovhost.exe` instance. The session is persistent no termination of the `wsmprovhost.exe` process was observed.

In the previous sections we have established some key observations regarding remote command execution via WinRM. However, during the demo, I used a stageless beacon. The script first decodes the Base64 encoded payload then it uses the `.Net API` to call Windows API function in memory using assemblies. The script then allocates some memory and copies the payload in the allocated memory space. The payload was a 64-bits DLL and technique used was **DLL Reflective Loading**.

The payload strings contained by default:

-
-
-



This yara rule can be effective in detecting default usage of CS stageless beacons.

The following PowerShell events were observed on the target:

- `EID 4104` Script Block Logging:
    - This event can be considered noisy, so be careful during you detection engineering process and consider its verbosity.
    - Script blocks exceeding the maximum length of an event log message are fragmented into multiple entries.
    - Unlike `EID 4103` , this event doesn't record the output of the script
- `EID 4103` Module Logging:
    - Generates a large volume of events
    - Records the output of the executed commands

Keep in mind that these event are not enabled by default.

In order to validate your detection rules against WinRM being used for remote command execution, Atomic Red Team provides a great guide bellow:

https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1021.006/T1021.006.md

In DFIR engagements these events can be good source of information to get the right attack attributions:

- **EID 142** WSMan operation CreateShell failed (Helpful when WinRM is not enabled on the target host)
- **EID 169** User Authenticated Successfully (The user who was connected remotely)
- **EID 81** Processing Client Request for Operation CreateShell (Start of remoting activity)
- **EID 134** Sending Response for Operation DeleteShell (End of remoting activity)
- **EID 403** Engine state is changed from Available to Stopped (This event records the completion of a PowerShell activity)

WinRM event logs lack simple attribution and traceability meaning you need multiple correlation layers in order to identify the user, source IP and the ID of the infected process.

The command `Get-WSManInstance -ComputerName localhost -ResourceURI Shell -Enumerate` lists all currently active remote WinRM sessions and provides useful information :

- **Owner** : Username that opened the remote session
- **ClientIP**: Source IP from where the attacker attempted to move laterally.
- **ProcessID**: In this case it is wsmprovhost.exe where the executed commands will be invoked from.
- **ChildPocesses**: Number of child processes it opened.
- **MemoryUsed**: Can be good indicator since `winrm64` CS module used more than twice the memory used by `Enter-PSSession` for the same command.

```
 PS C:\\Users\\Administrator> Get-WSManInstance -ComputerName localhost -ResourceURI Shell -Enumerate

rsp             : <http://schemas.microsoft.com/wbem/wsman/1/windows/shell>
lang            : en-US
```

```
ShellId         : 04E49AF8-1CA8-4ACC-9135-6A3269115F3E
Name            : WinRM1
ResourceUri     : <http://schemas.microsoft.com/powershell/Microsoft.PowerShell>
Owner           : ATLAS\\Administrator
ClientIP        : 10.10.10.30
ProcessId       : 2844
IdleTimeOut     : PT7200.000S
InputStreams    : stdin pr
OutputStreams   : stdout
MaxIdleTimeOut  : PT2147483.647S
Locale          : en-US
DataLocale      : en-US
CompressionMode : XpressCompression
ProfileLoaded   : Yes
Encoding        : UTF8
BufferMode      : Block
State           : Connected
ShellRunTime    : P0DT0H4M32S
ShellInactivity : P0DT0H1M28S
MemoryUsed      : 134MB
ChildProcesses  : 2
```

A good idea would be to generate an event with the output of this command every time the process wsmprovhost.exe is created using scheduled tasks.

CobaltStrike jump psexec & psexec64

I love going through ZEEK logs first and look for network related telemtery specially for lateral movement techniques. When using CS psexec or psexec64 modules for lateral movement I observed remote service creation.

These modules use named pipes (RPC/NP) method to interact with the service control manager (SCM) RPC server. The server interface is identified by UUID `367ABB81-9844-35F1-AD32-98F038001003` and uses RPC endpoint `\\PIPE\\svcctl`.

The following ZEEK event logs were recorded :

- ZEEK DCE-RPC event was generated with DCE-RPC endpoint `SVCCTL` and operation `CreateServiceWoW64A`
- On the target `EID 5145` A network share object was checked to see whether client can be granted desired access will be generated with `Relative Target Name` defined as `SVCCTL` and Share Name `\*\IPC$`
- A service is then created with a random name and Image Path calling the process via the command `\\127.0.0.1\ADMIN$\[SERVICE_RANDOM_NAME].exe`. This will generate `EID 7045 New Service Was Installed` and `EID 4697 A Service Was Installed in the System`

- Then `\\127.0.0.1\ADMIN$\[SERVICE_RANDOM_NAME].exe` is executed and it invokes a `rundll32.exe` instance with no arguments which is very suspicious.
- Interacting with the beacon via `SHELL` command invokes a `CMD` instance

The following table is a summary of the observed telemetry relevant to this lateral movement technique.

CobaltStrike jump psexec_psh

CobaltStrike can laverage a PowerShell version of PsExec using the built-in module psexec_psh with everything being executed in memory via a one-liner.

- As previously noticed an interaction with SCM RPC server in order to create a service remotely was observed. Bellow are the ZEEK DCE-RPC event logs with the same operation as `psexec & psexec64` `CreateServiceWOW64A`
- Followed by creation of a new service which generated EID 7045/4697 with `%COMSPEC%` and `powershell` in the `Service File Name` field.
- PowerShell's EID 400 can be used as a detection opportunity where `HostApplication` contains `powershell -nop -w hidden -encodedcommand`.
- Pipe creation with regex pattern `status_[0-9a-f]{2}` was also observed. I provided bellow a gist with several regex pattern to detect hard coded named pipes in CobaltStrike modules. Bellow is a EID 5145 that can be used for this purpose but I encourage you to sysmon instead for it high event traceability quality.

Cobalt Strike Named Pipe Regex.csv

- Interacting with the beacon via the CS `shell` command would invoke a `cmd.exe` instance.

This pattern alone is very suspicious and can be a good detection opportunity for default usage of `psexec_psh` command.

The following are the event logs I observed during the demos:

Atomic Red Team provides a good start to validate your detection against some of these attack techniques:

https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1569.002/T1569.002.md

- You can use the following CyberChef recipe to decode and extract shellcode information executed by psexec_psh command.

https://github.com/SophosRapidResponse/CyberChef/blob/main/Cobalt%20Strike%20recipe%20for%20JABz.txt

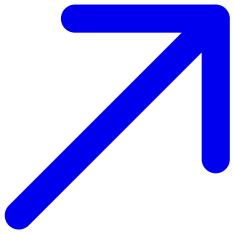
- You can list created pipes using `Get-ChilIt em` PowerShell cmdlets

```
Get-ChildItem \\\\.\\pipe\\
```

- Systinternal has a dedicated tool that also can be leveraged for the same purpose.

This blog post series of **Detecting CONTI CobaltStrike Lateral Movement Techniques** is focused on default usage of CS built-in capabilities meaning that sophisticated attacker will be able to change these settings and evade detections based on them thanks to CobalStrike modularity. My hope is to increase awareness at least about the telemetry that needs to be audited and qualified, how to correlate it and how to respond to relevant attacks in order to increase the time, effort and skills an APT has to invest in order to compromise your assets.

You can read my previous post on Detection Engineering Dimensions Analytics



part where I discuss analytic resilience.
