



8장

PowerShell을 통한 Remote
Computer Management의 모든 것

전체 내용

Remoting 개요

Remoting
Security

Enabling
Remoting

One-to-One
Remoting

One-to-Many
Remoting

CredSSP를
이용한 Second-
Hop 허용하기

Reusable
sessions of
Remoting

PowerShell Web
Access

쉬어가는 코너

1 – Remoting 개요

• Remoting 개요

- Remoting은 Web Service for Management(Ws-MAN)이라는 개방형 표준 프로토콜을 사용한다
- Windows OS는 Windows Remote Management(WinRM) 서비스에서 이 프로토콜을 구현한다
- WinRM은 Windows PowerShell Remoting용으로 사용된다
- Remoting은 반드시 원격 접속을 허용하는 쪽에서 설정되어야 한다 (기본으로는 http(5985 포트)가 열리며, 옵션으로 https(5986 포트)를 사용할 수 있다)
- Windows Server 2012 부터는 기본적으로 WinRM을 사용하도록 설정되어 있으며, Windows 8 부터는 수동으로 WinRM을 사용하도록 설정해야 한다
- PowerShell 2.0 이상에서는 수동으로 설정하면 언제든지 사용 가능하다 (**Enable-PSRemoting -Force** 또는 **WinRM qc**)

1 – Remoting 개요

- **Remoting 및 Remote Connectivity**

- Remoting은 명령어에 대한 구문 분석은 로컬 컴퓨터에서 하고, 명령어 실행은 원격 컴퓨터에서 진행한다. (로컬 컴퓨터 부하 경감)
- PowerShell 및 CIM을 관리하는 **Get-CimInstance**는 WinRM 서비스를 이용하므로 Remoting이라고 한다 (5985, 5986)
- 하지만 서버 관리자(**Server Manager**)는 WinRM 서비스로 원격컴퓨터를 관리한다

2 – Remoting Security

- **Remoting Security**

- 원격 컴퓨터의 Local Administrators 그룹의 구성원, Windows 8, Windows Server 2012부터는 **Remote Management Users** 그룹의 구성원도 Remoting 작업을 수행할 수 있다
- Remoting할 때 인증은 Mutual Authentication으로서 접속하는 컴퓨터랑 접속을 허용하는 컴퓨터들간에 서로 인증을 한다
- Active Directory Domain 환경에서는 기본적으로 도메인 구성원 컴퓨터간에는 기본적으로 상호 인증을 하도록 설계되어 있다
- Non-Domain computer에 접속할 때는 SSL 접속을 하거나 (이를 위해서는 사전에 인증서가 설치되어 있어야 하며), 아니면 로컬 컴퓨터의 TrustedHosts 목록에 원격 컴퓨터 이름을 추가하여 Mutual Authentication 기능을 끈다
- Remoting 할 때 Active Directory에 있는 컴퓨터 이름을 사용해야 한다. 만약 IP Address나 DNS 이름을 사용할 때는 SSL 접속을 하거나 아니면 로컬 컴퓨터의 TrustedHosts 목록에 IP Address나 DNS 이름을 등록한다

2 – Remoting Security

• TrustedHosts 목록

- AD 구성원 컴퓨터가 아닌 컴퓨터에 접속하기 위해서는 안전한 접속을 위해서는 SSL을 사용하기를 권장한다
- 단순히 쉽게 접속하기 위해서는 로컬 컴퓨터의 TrustedHosts list에 원격 컴퓨터 이름, IP Address, DNS 이름 또는 *를 사용하면 된다
- NonDomain 컴퓨터에 접속할 때 Mutual Authentication을 사용하지 않게 하려면 TrustedHosts 목록에 해당 컴퓨터를 등록시키면 된다
- TrustedHosts에 원격 컴퓨터 정보가 있는지 확인하기 위해
 - **Dir WSMAN:\localhost\Client\TrustedHosts**
 - **Get-Item WSMAN:\localhost\client\TrustedHosts**
- Mutual Authentication을 하지 않기 위해 TrustedHosts에 컴퓨터 이름을 다음과 같이 입력한다 (강추)
 - **Set-Item -Path WSMAN:\localhost\client\TrustedHosts -Value DC1**
- TrustedHosts의 내용을 삭제(초기화)하기 위해
 - **Clear-Item WSMAN:\localhost\client\TrustedHosts**

2 – Remoting Security

- **Remoting Encryption(또는 Privacy)**

- Remoting할 때는 반드시 **먼저 인증 단계를 거친다**
- Active Domain 환경인 경우에는, 인증 프로토콜을 Kerberos를 사용하는데, Credential은 password가 포함되지 않은 암호화된 Kerberos ticket으로 보내지기 때문에 안전하다
- Remoting에서는 데이터 전송할 때는 http를 사용한다. Domain 환경일 때는 데이터 전송전의 인증은 암호화된 상호인증이고, 데이터 전송은 Http를 사용한다 (사내에서는 이것만으로도 충분하다)
- 데이터 전송을 할 때도 안전하게 하고자 한다면 https를 사용하면 된다
- **Non-domain 컴퓨터랑 통신할 때는 기본적으로 인증은 Basic, 데이터 전송은 http**이다. Basic 인증은 Credential이 암호화 되지 않는다
- 만약 Non-domain 컴퓨터랑 통신할 때, 안전하게 통신을 하고자 하면 상호 인증을 하지 않은 상태에서 SSL 통신을 하도록 추가적으로 구성하여 Basic 인증 및 데이터 전송시에도 모두 암호화 통신하도록 한다
- 가장 보안이 약한 것은 상호 인증을 하지 않도록 설정하는 것이다. 물론 데이터 전송도 기본 설정인 http로 하는 것이다.

3 - Enabling Remoting

• Enable-PSRemoting

- Remoting 설정은 원격 작업을 받아 들이는 쪽에서만 한다. (원격 컴퓨터에 접속하는 쪽에서는 설정하지 않는다)
- Remoting 설정은 수동 및 개별적으로 Administrator 계정으로 로그인하여 **Enable-PSRemoting**을 입력하거나 아니면 중앙에서 Group Policy로 할 수 있다 (반대 설정은 Disable-PSRemoting)
- Enable-PSRemoting을 실행하면
 - 1) 방화벽에서 5985 포트를 예외 처리,
2) 모든 로컬 IP Address에 대하여 5985 포트에 HTTP listener를 생성,
3) WinRM 서비스를 자동으로 시작하도록 설정하고, 지금 restart한다,
4) 최대 4개의 Default endpoint를 등록하여 Windows PowerShell이 사용하도록 한다
- 클라이언트 컴퓨터에서 방화벽 설정이 **Public**로 되어 있는 경우에는 Enable-PSRemoting을 하더라도 방화벽에서 예외 처리를 해주지 않는다
 - 원격 관리를 하기 위해서는 Ethernet Adapter의 Network Profile이 반드시 Domain이나 Private로 되어 있어야 한다
 - Get-NetConnectionProfile | Set-NetConnectionProfile -NetworkCategory Private

3 - Enabling Remoting

- **Enable-PSRemoting하는 다양한 방법**

- 모든 컴퓨터에서 수동으로 **Enable-PSRemoting -Force** 실행하기
- **Group Policy**를 사용하여 이 명령어를 스크립트(EnablePSRemoting.ps1) 로 다운로드하여 설치하기
 - 1) GPMC.msc에서 Group Policy Object를 생성한다
 - 2) 다음과 같이 설정하여 Set-ExecutionPolicy를 RemoteSigned로 설정하여 PowerShell Script 파일을 실행할 수 있도록 허용한다
[Computer Configuration] - [Administrative Template] - [Windows Components] - [Windows PowerShell] 에서 "Turn on Script Execution" 항목을 Enable하여 "Allow local scripts and remote signed scripts"로 설정한다
 - 3) 로그인 스크립트를 생성한다
- **배치 파일을 사용하여** EnablePSRemoting.ps1 파일 실행하기
 - @echo off
powershell.exe -ExecutionPolicy RemoteSigned -File
\\DC1\Mod6\EnablePSRemoting.ps1

4 - One-to-One Remoting

- **1:1 연결**

- **Enter-PSSession -ComputerName DC1**
- 이렇게 접속을 하면 Prompt가 원격 컴퓨터가 된다
- Netstat -ano을 하면 5985 포트에 ESTABLISHED되어 있음을 알 수 있다
- 연결을 끊고 다시 되돌아 오려면 **Exit-PSSession**을 한다

5 - One-to-Many Remoting

- 1:N 연결

- **Invoke-Command -cn DC1, Server2 -ScriptBlock {명령어}**

- 하나의 명령어를 여러 컴퓨터에 병렬로 보내어 명령을 실행하여 결과를 로컬 컴퓨터로 가져온다
- 명령어 구문 분석은 로컬 컴퓨터에서, 실행은 원격 컴퓨터에서 진행한다

- **Invoke-Command -cn DC1, Server2 -FilePath c:\script.ps1**

- 이 스크립트 파일은 로컬 컴퓨터에 저장된 것이다
- 기본적으로 동시 처리 수량은 32개이고, 더 많은 컴퓨터인 경우에는 Queue에 남아 있다가, 다른 것들이 처리되면 그 다음에 처리된다

- **-ThrottleLimit**를 조정하면 32개 이상의 처리할 수 있지만, 그러면 로컬 컴퓨터의 메모리와 CPU에 부하가 생긴다 (32개 동시 세션 연결에서 64개 동시 세션 연결이 생기면 당연히 로컬 컴퓨터에 메모리 사용량이 증가한다)

5 - One-to-Many Remoting

- **-Scriptblock의 구문 분석은 로컬에서, 실행은 원격에서**
 - PowerShell v4:
 - Invoke-Command -ComputerName **Server2** -Scriptblock {**Install-WindowsFeature** -Name DNS}
 - 이렇게 입력하면서 Tab을 누르면 문장 자동 완성이 된다. 하지만 명령어 실행은 안 된다
 - 그 이유는 Server2가 Windows Server 2008이면 Install-WindowsFeature가 없기 때문이다
 - PowerShell v2:
 - Invoke-Command -ComputerName **DC1** {**Get-ADUser -Filter ***}
 - 이렇게 할 때 Tab을 눌러도 자동 완성이 안 된다. 그 이유는 Server2에는 ActiveDirectory 모듈이 설치되어 있지 않기 때문이다
 - 하지만 명령어는 제대로 실행되어 정상적인 결과가 나타난다

5 - One-to-Many Remoting

- **Multiple Computer Name 사용하는 방법**
 - 수동으로 입력하기
 - -ComputerName **DC1, Server1, Server2**
 - Text 파일에서 컴퓨터 이름 불러 와서 사용하기
 - -ComputerName (**Get-Content C:\Lab\Computers.txt**)
 - CSV 파일 불러 와서 사용하기
 - -ComputerName (**Import-CSV C:\Lab\Computers.csv | Select-Object -ExpandProperty name**)
 - Active Directory에서 컴퓨터 이름 가져 오기
 - -ComputerName (**Get-ADComputer -Filter * | Select -Expand Name**)

5 - One-to-Many Remoting

- **Local variable를 Remote Computer로 보내기**

- 로컬 컴퓨터에서 사용한 변수를 원격 컴퓨터로 보내는 방법은 **-ArgumentList**를 사용하는 것이다.
- -ArgumentList의 값을 Script Block에 있는 **Param(\$var)**에 넣는다
- 다음 2개의 예제를 보자
 - \$quantity = Read-Host "몇 개의 로그를 조회하고 싶은가요?"
 - Invoke-Command **-ArgumentList \$quantity -cn** DC1 -ScriptBlock { **Param(\$x)** Get-EventLog -LogName Security **-Newest \$x** }
 - \$Log = 'Security'
 - \$Quantity = 10
 - Invoke-Command **-ArgumentList \$Log,\$Quantity -cn** DC1, Server2 -ScriptBlock {**Param(\$x,\$y)** Get-EventLog -LogName \$x -Newest \$y}

5 - One-to-Many Remoting

Invoke-Command

```
-ScriptBlock {  
    Param($c,$r)  
    New-PSDrive -Name Z  
                -Credential $c  
                -PSProvider FileSystem  
                -Root $r  
}  
-ComputerName SERVER1,SERVER2,SERVER3  
-ArgumentList (Get-Credential), 'Path'
```

This command runs on the remote computer

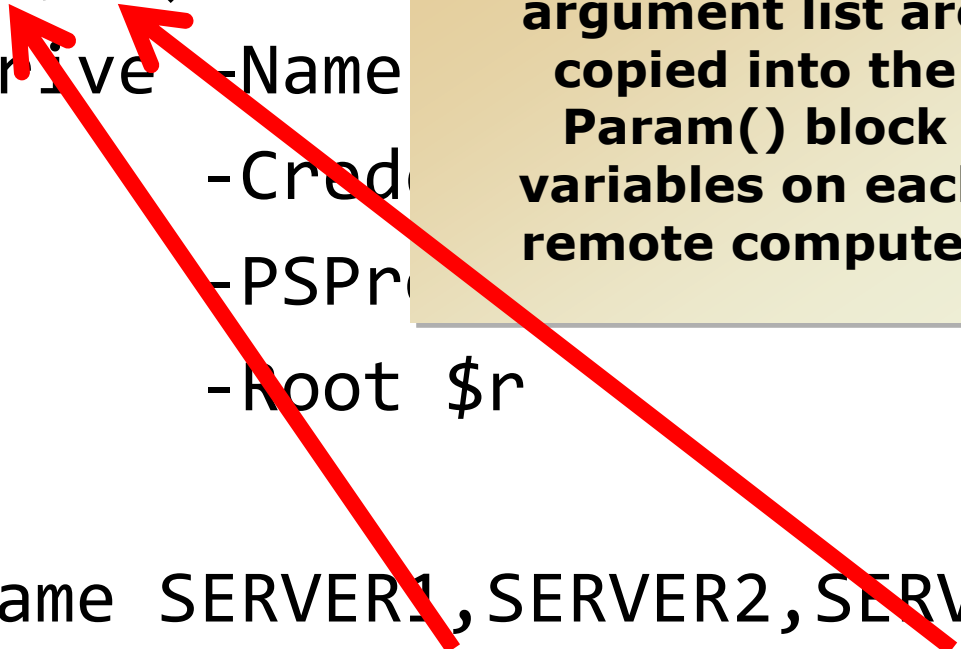
These computers run the command

This command runs on the local computer

5 - One-to-Many Remoting

Invoke-Command

```
-ScriptBlock {  
    Param($c,$r)  
    New-PSDrive -Name  
                -Cred  
                -PSPr  
                -Root $r  
}  
-ComputerName SERVER1,SERVER2,SERVER3  
-ArgumentList (Get-Credential),'Path'
```



The objects in the argument list are copied into the Param() block variables on each remote computer

5 - One-to-Many Remoting

Invoke-Command

```
-ScriptBlock {  
    Param($c,$r)
```

```
    New-PSDrive -Name Z
```

```
    -Credential $c
```

```
    -PSProvider FileSystem
```

```
    -Root $r
```

**The parameters are
used like variables in
the remote command**

```
-ComputerName SERVER1,SERVER2,SERVER3
```

```
-ArgumentList (Get-Credential), 'Path'
```

5 - One-to-Many Remoting

- **-ArgumentList 매개변수(parameter) 활용하기**

- -ArgumentList(=Args)를 Invoke-Command와 Invoke-WmiMethod에서 주로 사용한다
- -Args는 로컬 컴퓨터에서 입력한 값을 원격 컴퓨터의 변수 값으로 보낼 때 사용하는 것이다
- -ArgumentList를 사용하여 cmdlet 알아보기
 - Get-Command **-ParameterName argumentlist**
 - Export-PSSession, ForEach-Object, Get-Command, Import-Module, Import-PSSession, **Invoke-Command**, **Invoke-WmiMethod**, New-Module, New-Object, Start-Job, Start-Process, Trace-Command

5 - One-to-Many Remoting

- **-ArgumentList** 파라미터 활용하기

- 원격 컴퓨터의 인터넷 연결 여부를 확인하기 위해 Invoke-Command의 { } 내용에 입력할 값을 -Args로 처리하기
 - Invoke-Command -ComputerName server1 -Args "powershell.kr"
 { Param (\$x) Test-Connection -Destination \$x -Quiet -Count 1 }
- 로컬 컴퓨터에 저장된 Script를 원격 컴퓨터에서 실행하기
 - Internet.ps1 내용 (인터넷 연결 여부 확인하기)
 Test-Connection -Destination www.google.com -Quiet -Count 1
 - 스크립트를 불러와서 원격 컴퓨터에서 작업(실행)하기
 Invoke-Command -ComputerName server1 -FilePath internet.ps1
- Script를 원격 컴퓨터에서 실행할 때 Script에 포함된 Parameter 값을 로컬에서 입력하여 보내기
 - Internet.ps1 내용 (인터넷 연결 여부 확인시 사용자가 다른 주소 입력하기)
 Param (\$x) Test-Connection -Destination \$x -Quiet
 - icm -cn server1 -Args "powershell.kr" -FilePath internet.ps1
 ** 여기서 -Args의 value인 powershell.kr을 internet.ps1의 변수 값(-Destination의 value)으로 입력하게 하는 것이다

5 - One-to-Many Remoting

- **-ArgumentList** 파라미터 활용하기

- 원격 컴퓨터에 정해진 Message 띄우기

- **Invoke-WmiMethod** -ComputerName server1 -Class Win32_Process
-Name Create -**ArgumentList** "C:\Windows\System32\msg.exe * This is
a test message."

** 이렇게 하면 server1에 갑자기 메시지 창이 나타난다

** 이것을 script 파일로 저장하여 사용하면 된다

- 원격 컴퓨터에 원하는 Message 띄우기 (**##강추**)

- **\$msg = Read-Host** "어떤 메시지를 보내고 싶은가요? 입력하세요."
- **Invoke-WmiMethod** -ComputerName server1 -Class Win32_Process
-Name Create -**ArgumentList** "C:\Windows\System32\msg.exe * \$msg"

** 이 내용을 script 파일로 저장하여 실행하면 관리자가 입력한 메시지가
원격 컴퓨터에 Popup되게 된다

** 여러 대의 컴퓨터에 메시지를 보내고 싶으면 미리 컴퓨터 이름이 포함된
파일(pclist.txt)을 만들어 두고 -ComputerName (Get-Content c:\pclist.txt)로
사용하면 된다

6 - CredSSP를 이용한 Second-Hop 허용하기

- **Remoting할 때 컴퓨터 인증 문제로 연결 실패하는 일**

- 원격 컴퓨터1에 접속하여 또 다른 원격 컴퓨터2로 작업을 하고자 하면 작업을 할 수 없는 문제가 발생한다
- 다음과 같이 DC1에서 Server1에 접속하여 Server2에 작업을 해본다
 - **Enter-PSSession** -Comp Server1
net use x: \\server2\Data
Get-Service **-ComputerName Server2** -Name bits
 - **Invoke-Command** -Comp Server1 {**Test-Path** \\Server2\Data}
 - **icm** -cn Server1 {**Add-Content** **\\Server2\Data\1.txt** -Value "God is Love"}
 - \$remote = **New-PSSession** -ComputerName Server1
Invoke-Command **-Session \$remote** {Get-Process **-cn Server2**}
- DC1에서 로컬에 저장된 스크립트를 실행하여 Server1, Server2가 DC1에 네트워크 드라이브로 연결해본다
 - **icm** -ComputerName Server1, Server2 **-FilePath c:\ConnectToDC1.ps1**
*if (!(Test-Path -Path N: -IsValid))
{ New-PSDrive -Name N -PSProvider FileSystem -Root \\DC1\LabFiles -Persist }*

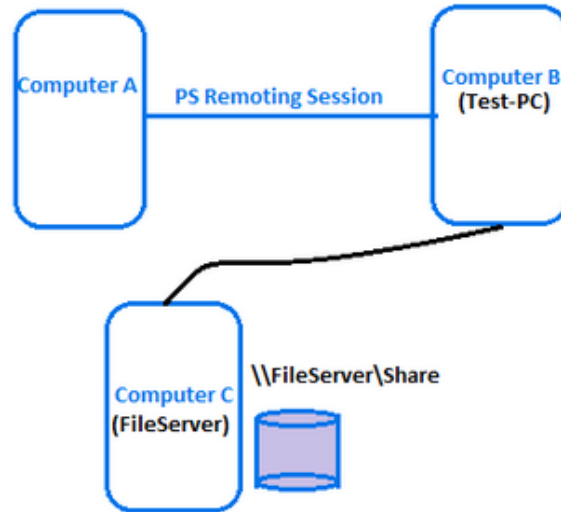
**** 모든 작업이 실패한다**

6 - CredSSP를 이용한 Second-Hop 허용하기

- **Remoting할 때 컴퓨터 인증 문제로 연결 실패하는 일**
 - **DC1에서 WinRM이 실행중인 Server1에 연결할 때 먼저 인증문제가 해결되어야 한다.** 인증은 Computer 및 User 모두 통과해야 한다
 - DC1이 Server1의 WinRM에 원격 접속할 때 컴퓨터에 대한 상호 인증을 한 후 그 다음에 사용자 인증을 처리한다
 - AD 환경에서 Domain 구성원 컴퓨터간에는 기본적으로 서로 [상호 인증]하기 때문에 DC1에서 Server1에 접속할 수 있다. 그런 다음 DC1에서 사용한 사용자 계정으로 Server1에 사용자 인증을 한다
 - 다시 Server1에서 Server2로 원격 접속을 할 때도 이와 같은 절차를 거친다
 - 그런데 [사용자 인증은 기본적으로 위임이 되지만] [컴퓨터 계정 인증은 기본적으로 위임이 되지 않는다]
 - 컴퓨터 계정도 위임을 하기 위해서 Credential Security Service Provider (CredSSP) 기능을 컴퓨터에 설정을 해야 한다
 - **Invoke-Command, Enter-PSSession, New-PSSession**을 사용한 후(원격 컴퓨터에 WinRM에 접속) 원격 컴퓨터에 액세스할 때 조심한다 (-ComputerName, net use, 명령어 \\server\data 등등)

6 - CredSSP를 이용한 Second-Hop 허용하기

- CredSSP 설정하기

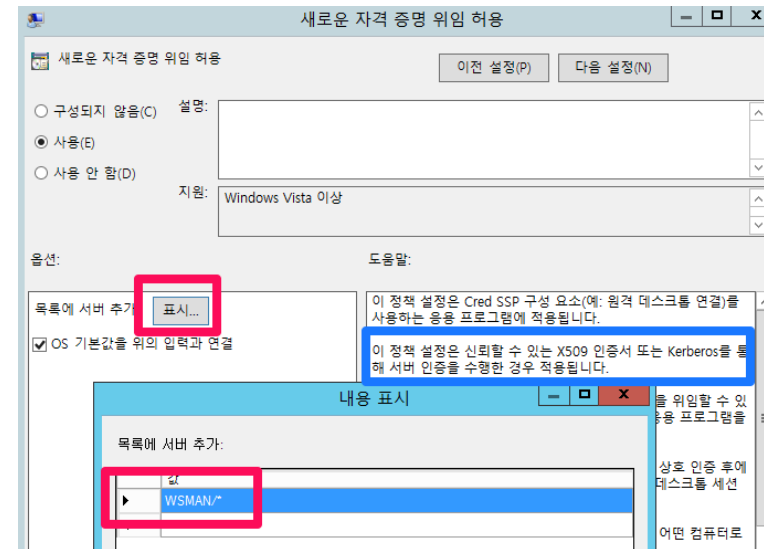


- **Get-WSManCredSSP**을 사용하여 현재 설정 확인하기
- [최초 접속을 시도하는 **Computer A**]에서 설정하기
Enable-WSManCredSSP -Role Client -DelegateComputer *
- [중계하는 **Computer B**]에서 설정하기
Enable-WSManCredSSP -Role Server
- Computer A에서 Computer B로 접속할 때는 반드시
"-Credential 도메인\사용자"와 "-Authentication CredSSP"를
사용해야 한다

6 - CredSSP를 이용한 Second-Hop 허용하기

• GPO를 사용한 CredSSP 설정하기

- [Computer A에서의 설정]을 Group Policy에서도 할 수 있다. 즉, Credentials Delegation을 어느 컴퓨터로 할 것인지를 지정한다
- [컴퓨터 구성] - [관리 템플릿] - [시스템] - [자격 증명 위임]에서 “새로운 자격 증명 위임 허용”을 클릭하여 “표시”를 클릭하여 **WSMAN/***을 입력한다
- 특정한 컴퓨터로만 위임할 때는 다음과 같이 할 수 있다
WSMAN/Server1
WSMAN/Server1.MCTconnected.com
WSMAN/*.MCTconnected.com



6 - CredSSP를 이용한 Second-Hop 허용하기

- **GPO를 사용한 CredSSP 설정 확인하기**

- 설정 여부를 확인하기 위해서 다음과 같이 하여 value가 True로 보인다
<Computer A>

Get-Item WSMAN:\localhost**Client**\Auth\CredSSP

<Computer B>

Get-Item WSMAN:\localhost**Service**\Auth\CredSSP

- 설정을 해제하기 위해서

Disable-WSManCredSSP -Role Client

Disable-WSManCredSSP -Role Server

```
PS C:\> hostname
dc1
PS C:\> Get-WSManCredSSP
새 자격 증명을 다음 대상에 위임할 수 있도록 컴퓨터가 구성되어 있습니다. wsman/*
이 컴퓨터는 원격 클라이언트 컴퓨터에서 자격 증명을 받도록 구성되어 있지 않습니다.
```

```
PS C:\> hostname
Server1
PS C:\> Get-WSManCredSSP
새 자격 증명을 위임할 수 있도록 컴퓨터가 구성되어 있지 않습니다.
이 컴퓨터는 원격 클라이언트 컴퓨터에서 자격 증명을 받도록 구성되어 있습니다.
PS C:\>
```

6 - CredSSP를 이용한 Second-Hop 허용하기

- **CredSSP를 사용한 예제**

- **Enter-PSSession** -ComputerName Server1 **-Credential** MCTconnected\administrator **-Authentication** CredSSP
net use x: \\server2\c\$
- **lcm** -Computer Server1 **-Credential** MCTconnected\administrator **-Authentication** CredSSP {**Test-Path** \\Server2\C\$}
- **icm** -Comp Server1 **-Credential** MCTconnected\administrator **-Authentication** CredSSP {**Add-Content** \\Server2\c\$\1.txt -Value "God is Love"}
- **\$session = New-PSSession -cn Server1 -Credential** MCTconnected\administrator **-Authentication** CredSSP
Invoke-Command -Session \$session {Test-Path \\Server2\C\$}

6 - CredSSP를 이용한 Second-Hop 허용하기

- **PowerShell에서 Remoting을 위한 최선의 설정**

- **Client 측**

- Domain GPO를 구성하면 잘 안 되는 경우가 있다
- **Enable-WSManCredSSP -Role Client -DelegateComputer ***

- **Server 측**

- 각 서버 측 컴퓨터에 일일이 명령어를 입력하여 설정한다
- **Enable-WSManCredSSP -Role Server**

- **Second-Hop Remoting을 위한 사용법**

- **Client 측**

- **Get-WSManCredSSP**를 사용하여 현재 설정 확인
- 명령어에 -Credential 도메인\사용자"와 "-Authentication CredSSP"를 사용
- 명령어에 아래와 같이 변수로 Credential을 사용하고 인증은 직접 입력
\$cred = Get-Credential
명령어 **-Credential** \$cred **-Authentication CredSSP**

- **Server 측**

- Get-WSManCredSSP를 사용하여 현재 설정 확인

7 – Reusable Sessions of Remoting

• Session 관리

- 지금까지 Enter-PSSession, Invoke-Command를 실행할 때는 이 명령어를 실행할 때 세션을 만든 후에 접속하여 사용했다
- 세션을 일단 만들어 놓고(**New-PSSession**) -**Session** Parameter로 접속하여 명령어를 실행하면 기존 세션을 계속 사용할 수 있다
 - \$dc = **New-PSSession** -ComputerName dc1
 - Invoke-Command -**Session \$dc** {hostname}
- 항상 세션 관리는 접속을 하는 쪽에서 하는 것이다
 - Get-PSSession
- 세션을 완전히 끊을 때도 접속하는 쪽에서 끊어야 한다
 - Get-PSSession | Remove-PSSession
- Disconnect-PSSession을 하면 세션 관리는 원격 컴퓨터에서 한다
 - \$server = New-PSSession -ComputerName server1
 - Invoke-Command -Session \$server {hostname}
 - **Disconnect-PSSession -Session** \$server
다른 PowerShell 콘솔을 하나 더 실행한다. 그런 다음 다시 연결한다
Connect-PSSession -ComputerName server1

7 – Reusable Sessions of Remoting

- **Session 생성 및 재사용하는 방법**

- Session을 생성할 때는 New-PSSession을 사용한다
 - \$dc = New-PSSession -ComputerName DC1
 - \$servers = New-PSSession -ComputerName server1, server2

- **생성된 Session에 연결하기**

- 생성된 세션을 확인하는 방법
 - **Get-PSSession**
- DC1에 1:1 연결한 후 작업을 하고 끊고 나오기
 - **Enter-PSSession -Session \$dc**
hostname
Exit-PSSession
Get-PSSession (세션 수량 변화 없음)
- Server1에 1:1로 연결한 후 작업을 하고 끊고 나오기
 - **Enter-PSSession -Session (Get-PSSession -ComputerName server1)**
ipconfig
Exit-PSSession
Get-PSSession (세션 수량 변화 없음)

7 – Reusable Sessions of Remoting

- **생성된 Session에 연결하기**

- Server1, Server2에 1:N으로 연결하여 작업한다
 - **Invoke-Command -Session \$servers -Scriptblock {hostname}**
 - Get-PSSession (세션 수량에 변화가 없다)
- DC1, Server1에 1:N으로 연결하여 작업한다
 - **Invoke-Command -Session (Get-PSSession -ComputerName DC1, Server1) -Scriptblock {ipconfig}**
 - Get-PSSession (세션 수량에 변화가 없다)

- **생성된 Session을 영구적으로 제거하기**

- Get-PSSession -ComputerName DC1 | Remove-PSSession
Get-PSSession (하나의 세션이 제거되었다)
- Remove-Pssession -ComputerName Server1
- 모든 세션을 한꺼번에 제거하기
 - Get-PSSession | Remove-PSSession

7 – Reusable Sessions of Remoting

- 연결된 **Session**을 끊어 버리기

- 원격 데스크톱 연결을 하여 작업하다가 창을 닫고 연결을 끊는 것을 Disconnect라고 한다. 조금 후에 다시 연결하면 이전에 작업하던 것이 Memory에서 그대로 남아 있어서 계속 작업할 수 있다
- 이것과 동일한 것이 **Disconnect-PSSession**이다
- 세션이 끊기 원격 데스크톱에 다시 연결하는 것과 같은 것이 **Connect-PSSession**이다
- 원격 데스크톱으로 접속하여 작업을 끝내고 로그오프하면 그동안 실행하여 사용중인 모든 프로그램이 종료되고 메모리도 확보하는 것이 Logoff하는 것이다
- 원격 데스크톱 연결에서 Logoff하는 것과 동일한 것이 **Remove-PSSession**이다.

7 – Reusable Sessions of Remoting

- 연결된 Session을 끊어 버리기

- `$dc = New-PSSession -ComputerName DC1`

Get-PSSession

Invoke-Command -Session \$dc {Start-Job -Name FindingLog {dir c:\ -Filter *.log -Recurse}}

Disconnect-PSSession -Session \$dc

- PowerShell 콘솔을 하나 더 실행하여
Get-PSSession
(아무 세션도 없다)

- 끊어진 세션에 다시 연결하기

Connect-PSSession -ComputerName DC1

```
PS C:\> $dc = New-PSSession -ComputerName dc1
PS C:\> Get-PSSession
```

Id	Name	ComputerName	State
21	Session21	dc1	Opened

```
PS C:\> Disconnect-PSSession -Session $dc
```

Id	Name	ComputerName	State
21	Session21	dc1	Disconnected

7 – Reusable Sessions of Remoting

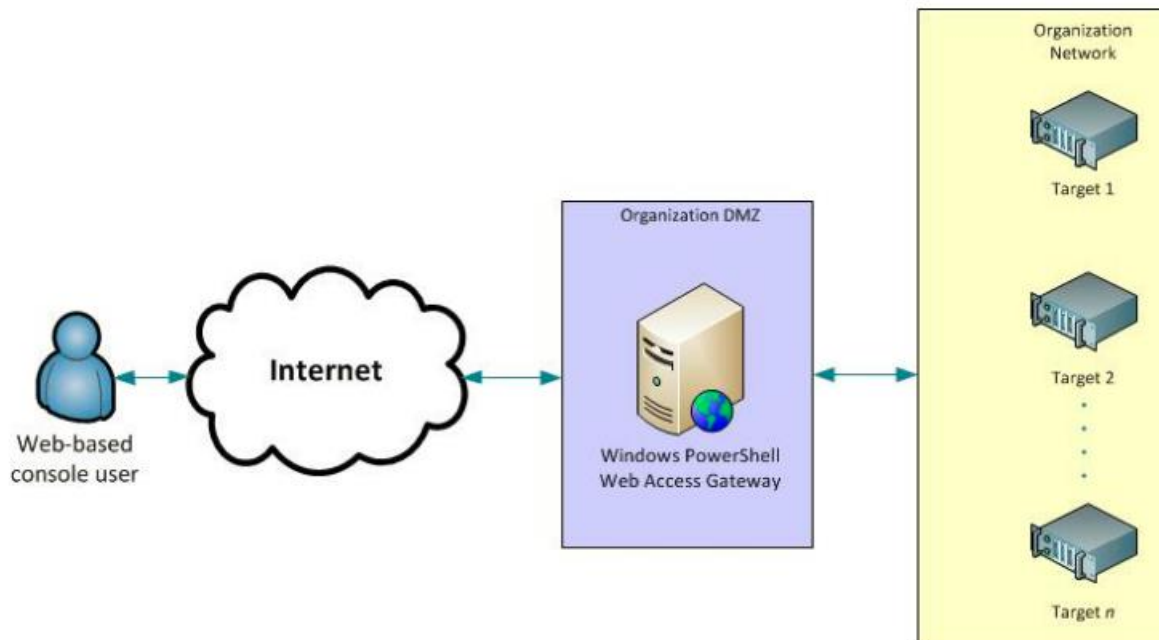
- 원격 컴퓨터랑 세션이 연결된 상태에서 -PSSession을 사용하여 작업하기

- Domain Member 컴퓨터에서 Active Directory를 관리하고자 한다. 그렇게 하려면 **Get-Module -Name ActiveDirectory**이 있어야 한다
- 세션을 생성하여 원격 컴퓨터랑 연결한다
 - \$dc = **New-PSSession** -ComputerName DC1
- 세션이 연결된 상태에서는 **-PSSession \$dc**를 사용하여 작업을 한다. DC1에 어떤 모듈이 저장되어 있는지 확인한다
 - **Get-Module -ListAvailable -PSSession \$dc**
- 이제 그 저장된 모듈을 로컬 컴퓨터로 복사 및 설치한다(Import-Module)
 - **Import-Module -Name ActiveDirectory -PSSession \$dc**
- 로컬 컴퓨터에서 해당 모듈을 이용할 수 있는지 확인한다
 - **Get-Module**
Get-Module -ListAvailable
Get-Command -Module ActiveDirectory
Get-Aduser -Filter * | ft name
- -PSSession은 Get-Module과 Import-Module에서만 사용된다

8 - PowerShell Web Access

• PowerShell Web Access의 필요성

- 사외에서 사내 서버를 PowerShell로 관리하고자 할 때 사외에서 DMZ에 위치한 Web Server에 <https://>로 접속한 후 PowerShell 명령어를 실행하면 Web Server가 Proxy 역할을 한다
- 최종 접속하는 컴퓨터(DC1)에 여러 개의 PowerShell Script를 준비해두면 원격 컴퓨터 관리가 쉽다
- Invoke-Command는 **사용 가능**, Enter-PSSession은 **사용 불가능**



8 - PowerShell Web Access

- **PowerShell Web Access 구현하기**

- 원격 관리를 허용하는 사내 서버들에서는 Enable-PSRemoting -Force
- PowerShell Web Access Gateway 구성하기
 - 1) 필요한 Feature 설치하기

Install-WindowsFeature -Name WindowsPowerShellWebAccess -IncludeManagementTools -Restart

2) PSWA라는 Web Application 생성, Self-Signed 인증서, <https://로> Binding을 하도록 설정하기

Install-PSWAWebApplication -UseTestCertificate

3) Proxy 해줄 컴퓨터 및 접속할 권한을 가진 User를 지정한다(Rule 생성하기)

Add-PSWAAuthorizationRule -UserName * -ComputerName * -ConfigurationName * -RuleName WebServers
(줄여서 Add-PSWAAuthorizationRule * * *)

- 사용자 접속하기: **https://Server1/pswa**
- 참고: PSWA를 사용하여 Web Server를 PowerShell로 관리한다

8 - PowerShell Web Access

• PowerShell Web Access 구현하기

- Rule을 생성할 때 -UserName 부분에는 정확하게 사용자 계정을 입력한다
- 사용자 계정을 입력하지 않는 경우에는 사용자 계정이 접속을 할 때 다음과 같은 User를 입력한다
 - 1) 최종 접속하여 컴퓨터의 SAM의 Administrators 구성원
 - 2) Remote Management Users 그룹의 구성원
- -ComputerName에도 Proxy해 줄 컴퓨터 이름을 정확히 입력한다
- -ConfigurationName에서 이름을 정확히 입력한다
Add-PSWAAuthorizationRule -UserName MCTconnected\suser1 -ComputerName DC1 -ConfigurationName AdminsOnly -RuleName webservers
- 제대로 구성되었는지 확인한다
 - Get-PSWAAuthorizationRule
 - Get-PSWAAuthorizationRule -UserName MCTconnected\suser1 -ComputerName DC1
 - Get-PSWAAuthorizationRule * * *

9 – 쉬어가는 코너

Session과 관련된 명령어 찾기

9 – 쉬어가는 코너

• Session과 관련된 명령어 찾기

- Remoting을 할 때는 ComputerName, Session, PSSession과 같은 Parameter를 사용하는데 이것들이 포함된 cmdlet을 찾아보자
- 특정한 Parameter가 포함된 cmdlet을 찾을 때는 **-ParameterName**을 사용한다
- 매개변수 **ComputerName**이 포함된 cmdlet
 - Get-Command **-ParameterName ComputerName**
- 매개변수 **PSSession**이 포함된 cmdlet
 - Get-Command **-ParameterName PSSession**
- 매개변수 **Session**이 포함된 cmdlet
 - Get-Command **-ParameterName Session**

```
PS C:\#> Get-Command -ParameterName Session
```

CommandType	Name
Cmdlet	Connect-PSSession
Cmdlet	Disconnect-PSSession
Cmdlet	Enter-PSSession
Cmdlet	Export-PSSession
Cmdlet	Import-PSSession
Cmdlet	Invoke-Command
Cmdlet	New-PSSession
Cmdlet	Receive-Job
Cmdlet	Receive-PSSession
Cmdlet	Remove-PSSession

9 – 쉬어가는 코너

- Parameter 2개가 [동시에 포함된 cmdlet]을 찾고자 한다
- 매개변수 **ComputerName**과 **Session**은 포함된 cmdlet 찾기
 - Gcm | Where { \$_.Parameters.Keys **-contains** '**ComputerName**' -and \$_.Parameters.Keys **-contains** '**Session**' }

```
PS C:\> Gcm | Where { $_.Parameters.Keys -contains 'ComputerName' -and $_.Parameters.Keys -contains 'Session' }
```

CommandType	Name	ModuleName
Cmdlet	Connect-PSSession	Microsoft.PowerShell.Core
Cmdlet	Enter-PSSession	Microsoft.PowerShell.Core
Cmdlet	Invoke-Command	Microsoft.PowerShell.Core
Cmdlet	New-PSSession	Microsoft.PowerShell.Core
Cmdlet	Receive-Job	Microsoft.PowerShell.Core
Cmdlet	Receive-PSSession	Microsoft.PowerShell.Core
Cmdlet	Remove-PSSession	Microsoft.PowerShell.Core

- 매개변수 **ComputerName**은 포함, **Session**은 포함되지 않은 cmdlet 찾기
 - Gcm | Where { \$_.Parameters.Keys **-contains** '**ComputerName**' -and \$_.Parameters.Keys **-notcontains** '**Session**' }

- Remoting을 위한 사전 설정하기
- Remoting 구현하기
- Second-Hop 구현하기
- PowerShell Web Access 구현하기

- Remoting 개요
- Remoting Security
- Enabling Remoting
- One-to-One Remoting
- One-to-Many Remoting
- CredSSP를 이용한 Second-Hop 허용하기
- Reusable sessions of Remoting
- PowerShell Web Access