Lab 02 - PowerShell Policies in Core

Objective

Apply the concepts you learned in the first lab to implement and use logging in PowerShell Core.

Background

The following policies also work on Windows, MacOS, and Linux in PowerShell Core. This lab will specifically cover PowerShell Core on Linux. Learn to implement policies and identify PowerShell fingerprints in the following locations:

- PSReadLine command history
- Script block logging
- Transcription

For more information see about_Logging_Non-Windows and about_Logging_Windows. Bookmark these to read completely later.

Overview

This lab contains a Linux host with PowerShell Core already installed. You will use SSH to connect. In the online lab web interface you must click the `View VMs` button under the **Virtual Clients** heading. Use the **RDP/SSH IP** of the `client-05.training.com` machine to connect.

If you are are using a Windows machine, you will need an SSH client like the free utility PuTTY or the Windows Subsystem for Linux (WSL) or enabled the Windows Optional Feature *OpenSSH Client* on Windows 10 1809 and above. If you prefer you can RDP into the `client01.training.com` Windows 10 machine where the PuTTY client is already installed. MacOS or Linux clients can use SSH natively from the terminal.

Exercise 2.1 - Implement logging settings

Begin by reviewing and implementing the logging settings available in PowerShell Core.

2.1.1 Get connected

1. Use the **RDP/SSH IP** of the `client-05.training.com` lab machine to connect via SSH using your tool of choice (PuTTY, terminal, etc.).
    - Using terminal:
        - `ssh user@1.2.3.4` (use the **RDP/SSH IP** from your lab web page)
        - Enter the password from the lab guide.
    - Using PuTTY on Windows:
        - Install PuTTY from putty.org.
        - Launch PuTTY.
        - Paste the IP in the appropriate box. Click the `Open` button.

        - If using PuTTY from your local machine, then use the **RDP/SSH IP**.
        - If using PuTTY from the Windows 10 lab VM, then use the hostname **client-05**.
    - If prompted to trust the host click `Yes`.
    - Login as `user` with the password from the lab guide.

2. Launch PowerShell Core:

   ```
   pwsh
   ```

3. Check the version and edition:

   ```
   $PSVersionTable
   ```

2.1.2 PSReadLine

PSReadLine command history is a free source of logging information that does not require any configuration. It will only record commands physically typed at the console. It is easily bypassed.

1. Notice that the `PSReadLine` module is also installed by default here:

   ```
   Get-Module
   ```

2. It behaves very similarly as on Windows PowerShell:

   ```
   Get-Command -Module PSReadLine
   ```

   Notice the properties `MaximumHistoryCount` and `HistorySavePath`:

   ```
   Get-PSReadLineOption
   ```

> **NOTE** - As on Windows PowerShell it is easy to evade this by unloading the module: `Remove-Module PSReadLine`.

2.1.3 PowerShell policies JSON

1. Open this help topic in your local web browser about_Logging_Non-Windows and skim the section `Configuring Logging on non-Windows system`.

2. Review the sample JSON settings file in the documentation.

3. In your PowerShell Core session type the following commands:

   ```
   cd $pshome
   ```

   ```
   dir *.json
   ```

4. In the interest of lab time a sample PowerShell policy file has been provided for you. Review it with this command:

   ```
   cat powershell.config.lab.json
   ```

```
{
    "Microsoft.PowerShell:ExecutionPolicy": "RemoteSigned",
    "PowerShellPolicies": {
        "ScriptBlockLogging": {
        "EnableScriptBlockInvocationLogging": false,
        "EnableScriptBlockLogging": true
        },
        "ModuleLogging": {
        "EnableModuleLogging": false,
        "ModuleNames": [
            "*"
        ]
        },
        "Transcription": {
        "EnableTranscripting": true,
        "EnableInvocationHeader": true,
        "OutputDirectory": "/var/tmp/pstranscripts/"
        }
    },
    "LogLevel": "verbose"
}
```

5. Notice the following settings implemented the same as on Windows:

    - ScriptBlockLogging
    - ModuleLogging
    - Transcription

    Notice the transcription path is `/var/temp`. This `tmp` directory will persist between reboots, where as `/tmp` root directory does not persist between reboots.

6. Implement the policy by making a copy of the JSON file. Use TAB completion just like in Windows to complete commands and paths. You will need to perform this command as root:

    ```
    sudo cp ./powershell.config.lab.json ./powershell.config.json
    ```

    Enter the `root` password from the lab setup guide if prompted.

    Double check your typing to make sure the name of the new file is correct.

    ```
    dir *.json
    ```

Exercise 2.2 - Generate PowerShell activity

2.2.1 Generate PowerShell activity

1. Remember that PowerShell sessions have a cached copy of policies. In order to use the new policy you must exit the current PowerShell session and launch a new one:

    ```
    exit
    ```

    ```
    pwsh
    ```

2. Now run some commands in the session. Try whatever you like. Here are some samples:

   ```
   "Hello, world."
   ```

   ```
   $env:PSModulePath
   ```

   ```
   Get-Process | Sort-Object CPU -Descending | Select-Object -First 5
   ```

   ```
   Test-Connection ts1
   ```

3. Close this session to complete the transcript file.

   ```
   exit
   ```

> **NOTE** - Do not type `exit` too many times, or you will leave your SSH session. If you did this by accident, then follow the instructions at the beginning of the lab to reconnect.

Exercise 2.3 - Find PowerShell activity in the logs

2.3.1 PSReadLine

1. Open a fresh PowerShell Core session:

   ```
   pwsh
   ```

2. For starters, let's view the PSReadLine history:

   ```
   (Get-PSReadLineOption).HistorySavePath
   ```

   ```
   cat (Get-PSReadLineOption).HistorySavePath
   ```

3. Copy the path from the first command showing you where the history file is located:

   ```
   /home/user/.local/share/powershell/PSReadLine/ConsoleHost_history.txt
   ```

4. ***In order to avoid further polluting the PowerShell logs with commands and output hunting for activity, we will use native Linux commands to view the PowerShell activity.*** Exit the PowerShell Core session and return to the Linux shell:

   ```
   exit
   ```

5. Now view the PSReadLine history file with native Linux commands:

   ```
   cat /home/user/.local/share/powershell/PSReadLine/ConsoleHost_history.txt
   ```

2.3.2 Script block logging

PowerShell logs to syslog on Linux and any of the tools commonly used to view syslog contents may be used. See docs.

1. To view basic syslog output for PowerShell Core on Linux use these commands:

   ```
   cd /var/log
   ```

```
sudo grep powershell messages
```

(`messages` is the name of the syslog file.)

2. According to the documentation logging follows this format:

```
TIMESTAMP MACHINENAME powershell[PID]: (COMMITID:TID:CID)
[EVENTID:TASK.OPCODE.LEVEL] MESSAGE
```

Carefully study the output to find the PowerShell commands delimited by the characters `#012` in the data.

3. Now let's have some fun. In this SSH session run the following command:

```
sudo tail -f messages
```

4. Follow the instructions at the beginning of this lab to open a second SSH session into the remote Linux host. Position the two SSH windows side-by-side.

5. In the second session run PowerShell Core and try some commands:

```
pwsh
```

```
Get-Process
```

```
dir
```

6. Watch the commands drop into the syslog messages stream in the other window. Find the commands you just typed from the PowerShell session.

7. Notice just like on windows that Script Block Logging drops `prompt` into the logs every time you come back to the prompt. This is annoying noise you will most definitely want to filter out in your log hunting automation.

8. In the PowerShell session type `exit` twice to leave PowerShell and leave the remote SSH session.

9. In the `tail` command window press `CTRL-C` to quit the `tail` command. Leave this SSH session open for the next exercise.

10. Bonus: Follow the steps here to create a dedicated rsyslog for PowerShell events.

2.3.3 Transcripts

1. Next, we will look at the transcripts. Explore the directory we configured for transcription:

```
ls -lR /var/tmp/pstranscripts
```

```
cd /var/tmp/pstranscripts
```

```
ls
```

```
cd USE_DATE_DIRECTORY_HERE
```

```
ls -l
```

2. View the files. Copy and paste one of the transcript file names into one of the following commands:

   `nano PASTED_TXT_FILE_NAME` (Use `PgUp`/`PgDn` to view the file and `CTRL-X` to exit.)

   `cat PASTED_TXT_FILE_NAME`

3. Use `grep` to recursively search for a case-insensitive keyword in all transcripts. Use a keyword from one of the commands you ran in an earlier PowerShell session:

   ```
   cd /var/tmp/pstranscripts
   ```

   ```
   grep -iR KEYWORD_HERE
   ```

4. Based on header data you have observed in transcript files, what `grep` keyword would you use to find the user name of anyone who has executed a script on this box?

5. Experiment with your own search techniques and keywords.

6. Type `exit` to leave the remote SSH session.

2.4 Considerations for PowerShell Core logging

Consider the following points as you deploy PowerShell Core in the enterprise:

1. Throughly review the unique requirement for each operating system here:

   - about_Logging_Windows
   - about_Logging_Non-Windows

   Note that there is an additional step required on the Windows machines beyond simply creating the JSON file.

2. Using the techniques from this lab how would you automate hunting for badness in PowerShell Core?

3. Do you plan to deploy PowerShell Core in your enterprise?

4. How will you manage policies in multiple side-by-side installs of PowerShell Core on the same machine?

5. How will you configure both *Windows* PowerShell and PowerShell *Core* logging on the same machine?

6. How will you filter this data at scale for malicious activity?

7. How will you forward relevant entries to your SIEM?

8. How will you trim the transcripts directory on a schedule to keep from filling the drive? (ex. `cron`)

End of line.