13

What Else? – Further Mitigations and Resources

In this book, we have looked at many topics and techniques that help you mitigate risks in your environment when it comes to PowerShell. But of course, there are many more things that you can do to secure your environment – many directly related to PowerShell, but also others that are not directly related but help you secure PowerShell.

In this chapter, we won't deep dive into every mitigation; instead, I will provide an overview of what other mitigations exist so that you can explore each on your own. We will cover the following topics:

- Secure scripting
- Exploring Desired State Configuration
- Hardening systems and environment
- Attack detection – Endpoint Detection and Response

Technical requirements

To make the most out of this chapter, ensure that you have the following:

- PowerShell 7.3 and above
- Installed Visual Studio Code
- Access to the GitHub repository for this chapter: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/tree/master/Chapter13**

Secure scripting

If you are leveraging self-written scripts in your environment, secure scripting is indispensable. If your scripts can be manipulated, it doesn't matter (most of the time) what other security mechanisms you have implemented.

Be aware that your scripts can be hacked, and malicious code can be injected. In these cases, you must do the following:

- Always validate input
- Have your code reviewed when developing scripts
- Secure the script's location and access
- Adopt a secure coding standard, such as the *OWASP Secure Coding Practices – Quick Reference Guide*: **https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/**

Additionally, two neat PowerShell modules come in handy when developing your own PowerShell scripts that you should know about – **PSScriptAnalyzer** and **InjectionHunter**.

PSScriptAnalyzer

PSScriptAnalyzer is a tool that statically checks code for PowerShell scripts and modules. It checks against predefined rules and returns all findings, along with recommendations on how to improve your potential code defects.

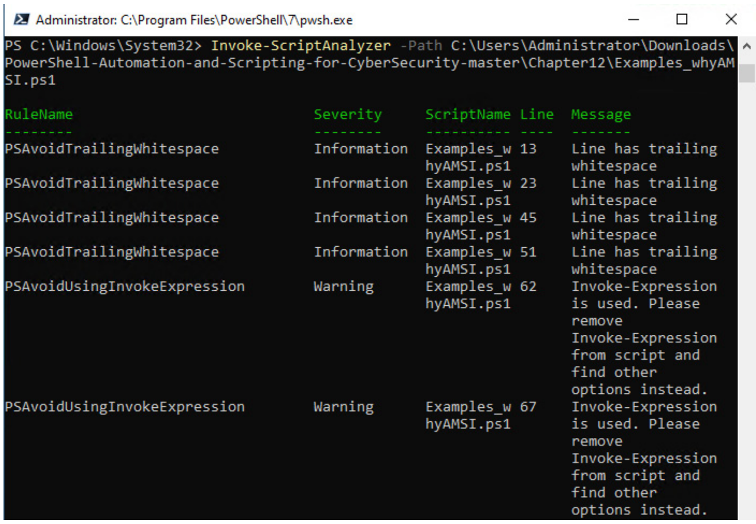
Using **PSScriptAnalyzer** to verify your code helps you to maintain higher code quality and avoid common issues. It is not necessarily a tool to check the security of your code (although it provides security checks such as **Avoid using Invoke-Expression**), but a tool to check whether you applied PowerShell best practices.

It can be installed from PowerShell Gallery using **Install-Module PSScriptAnalyzer**.

Once installed, it provides the **Get-ScriptAnalyzerRule**, **Invoke-Formatter**, and **Invoke-ScriptAnalyzer** cmdlets.

For our use case, we will only look into **Invoke-ScriptAnalyzer**, but make sure you check out the entire module on your own to improve your PowerShell scripts and modules.

Use **Invoke-ScriptAnalyzer**, followed by **-Path** and the path to the script, to have your code checked, as shown in the following screenshot:



```
PS C:\Windows\System32> Invoke-ScriptAnalyzer -Path C:\Users\Administrator\Downloads\
PowerShell-Automation-and-Scripting-for-CyberSecurity-master\Chapter12\Examples_whyAM
SI.ps1

RuleName                            Severity     ScriptName Line  Message
--------                            --------     ---------- ----  -------
PSAvoidTrailingWhitespace           Information  Examples_w 13    Line has trailing
                                                 hyAMSI.ps1       whitespace
PSAvoidTrailingWhitespace           Information  Examples_w 23    Line has trailing
                                                 hyAMSI.ps1       whitespace
PSAvoidTrailingWhitespace           Information  Examples_w 45    Line has trailing
                                                 hyAMSI.ps1       whitespace
PSAvoidTrailingWhitespace           Information  Examples_w 51    Line has trailing
                                                 hyAMSI.ps1       whitespace
PSAvoidUsingInvokeExpression        Warning      Examples_w 62    Invoke-Expression
                                                 hyAMSI.ps1       is used. Please
                                                                  remove
                                                                  Invoke-Expression
                                                                  from script and
                                                                  find other
                                                                  options instead.
PSAvoidUsingInvokeExpression        Warning      Examples_w 67    Invoke-Expression
                                                 hyAMSI.ps1       is used. Please
                                                                  remove
                                                                  Invoke-Expression
                                                                  from script and
                                                                  find other
                                                                  options instead.
```

Figure 13.1 – Invoking ScriptAnalyzer

When nothing else is specified, **PSScriptAnalyzer** checks against its own set of rules. But you can also specify your own custom rules by using the **-CustomRulePath** and **-RecurseCustomRulePath** parameters.

If you're using Visual Studio Code with the *PowerShell* extension to write PowerShell scripts, **PSScriptAnalyzer** is enabled by default. Here, your code will be automatically checked and you will be provided with warnings for any potential issues while writing your code.

InjectionHunter

InjectionHunter is a module, written by Lee Holmes, that helps you detect ways to inject code into your very own PowerShell script. It can be downloaded from **PowerShell Gallery**: **https://www.powershellgallery.com/packages/InjectionHunter/1.0.0**

Install it by using **Install-Module InjectionHunter**.

InjectionHunter relies on **ScriptAnalyzer.Generic.DiagnosticRecord** as its output type and uses custom detection rules, so **PSScriptAnalyzer** also needs to be installed.

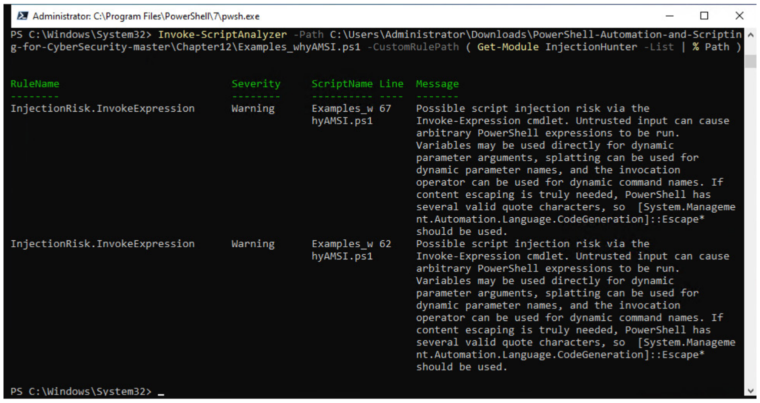
InjectionHunter comes with eight different functions, all of which can help you find out whether your code is vulnerable to various scenarios. These are **Measure-AddType**, **Measure-CommandInjection**, **Measure-DangerousMethod**, **Measure-ForeachObjectInjection**, **Measure-InvokeExpression**, **Measure-MethodInjection**, **Measure-PropertyInjection**, and **Measure-UnsafeEscaping**.

The **InjectionHunter** functions are used to create a new **PSScriptAnalyzer** plugin that can detect potential injection attacks in PowerShell scripts. These functions are designed to accept **-ScriptBlockAst** as a parameter, which represents the **Abstract Syntax Tree** (**AST**) of the script. The AST groups tokens into structures and is a deliberate way to parse and analyze data with PowerShell.

The following example demonstrates how to use **PSScriptAnalyzer** to call the **InjectionHunter** rules:

```
> Invoke-ScriptAnalyzer -Path C:\Users\Administrator\Downloads\PowerShell-Automation-and-Scripting-for-C
```

The following screenshot shows what it looks like to call **InjectionHunter** rules from **PSScriptAnalyzer**:



```
PS C:\Windows\System32> Invoke-ScriptAnalyzer -Path C:\Users\Administrator\Downloads\PowerShell-Automation-and-Scriptin
g-for-CyberSecurity-master\Chapter12\Examples_whyAMSI.ps1 -CustomRulePath ( Get-Module InjectionHunter -List | % Path )


RuleName                           Severity     ScriptName Line  Message
--------                           --------     ---------- ----  -------
InjectionRisk.InvokeExpression     Warning      Examples_w 67    Possible script injection risk via the
                                                hyAMSI.ps1       Invoke-Expression cmdlet. Untrusted input can cause
                                                                 arbitrary PowerShell expressions to be run.
                                                                 Variables may be used directly for dynamic
                                                                 parameter arguments, splatting can be used for
                                                                 dynamic parameter names, and the invocation
                                                                 operator can be used for dynamic command names. If
                                                                 content escaping is truly needed, PowerShell has
                                                                 several valid quote characters, so  [System.Manageme
                                                                 nt.Automation.Language.CodeGeneration]::Escape*
                                                                 should be used.
InjectionRisk.InvokeExpression     Warning      Examples_w 62    Possible script injection risk via the
                                                hyAMSI.ps1       Invoke-Expression cmdlet. Untrusted input can cause
                                                                 arbitrary PowerShell expressions to be run.
                                                                 Variables may be used directly for dynamic
                                                                 parameter arguments, splatting can be used for
                                                                 dynamic parameter names, and the invocation
                                                                 operator can be used for dynamic command names. If
                                                                 content escaping is truly needed, PowerShell has
                                                                 several valid quote characters, so  [System.Manageme
                                                                 nt.Automation.Language.CodeGeneration]::Escape*
                                                                 should be used.

PS C:\Windows\System32> _
```

Figure 13.2 – Calling the InjectionHunter rules from PSScriptAnalyzer

InjectionHunter was not intended for direct use in analyzing scripts. However, you can use its functions to develop a custom

PSScriptAnalyzer plugin that can detect injection attacks in your PowerShell scripts.

But wouldn't it be cool to immediately know whether you were implementing a potential injection risk while writing your scripts? Lee Holmes and the PowerShell team have you covered. The following blog article explains how this can be achieved when using Visual Studio Code to edit scripts: **https://devblogs.microsoft.com/powershell/powershell-injection-hunter-security-auditing-for-powershell-scripts/**.

Exploring Desired State Configuration

PowerShell **Desired State Configuration** (**DSC**) is a feature that enables you to manage your servers using PowerShell configuration as code.

At the time of writing, the following versions of DSC are available that you can use for deployment: **DSC 1.1**, **DSC 2.0**, and **DSC 3.0**.

While DSC 1.1 was included in Windows PowerShell 5.1, in DSC 2.0, which must run DSC on PowerShell 7.2 and above, **PSDesiredStateConfiguration** is no longer included in the PowerShell package. This enables the DSC creators to develop DSC independently of PowerShell and enables users to upgrade DSC without the need to upgrade PowerShell as well.

DSC 1.1

DSC 1.1 is included in Windows and updated through Windows Management Framework. It runs in Windows PowerShell 5.1. This is the go-to version if Azure Automanage Machine Configuration is not in use.

Remediation

DSC 1.1 has two configuration modes:

- **Push**: The configuration is pushed manually
- **Pull**: The nodes are configured to pull their configuration frequently from the pull server

One huge advance of DSC in pull mode is that your configuration, once specified, is self-healing. This means you configure your nodes using code and set up your configuration. Once activated, you can configure your configuration so that it's frequently pulled from your nodes. This means that if someone were to change the local configuration of a server or endpoint configured with DSC, the configuration would be changed back after the next pull.

Pull mode is a more complex configuration, but in the end, it is easier to maintain and helps you keep your devices more secure than using push mode. When using this mode, systems remediate themselves.

If you're interested in using DSC for central administration, it's worth noting that signed configurations make DSC an even more secure form of remote policy management. Signed configurations ensure that only authorized changes are applied to a system. Without a valid signature, a configuration cannot be applied.

This can be particularly valuable in protecting against attacks that compromise central management channels, such as GPO. With signed configurations in DSC and tight control over your signing infrastructure, attackers cannot use compromised channels to deliver ransomware companywide, for example.

You can learn more about the DSC module and configuration signing by visiting the following documentation page: **https://learn.microsoft.com/en-us/powershell/scripting/windows-powershell/wmf/whats-new/dsc-improvements?#dsc-module-and-configuration-signing-validations**.

DSC is quite extensive, but there's a lot of documentation, including quick starts and tutorials, that can help you get started: **https://learn.microsoft.com/en-us/powershell/dsc/overview?view=dsc-1.1**.

DSC 2.0

DSC 2.0 is supported for PowerShell 7.2 and above. While the original DSC platform was built on top of WMI for Windows, newer versions were decoupled from that model.

It can be deployed using PSGallery by running the following command:

```
Install-Module -Name PSDesiredStateConfiguration -Repository PSGallery -MaximumVersion 2.99
```

DSC version 2.0 should only be used if Azure Automanage Machine Configuration is in use. Although the **Invoke-DscResource** cmdlet is still available with this version, you should only use it for testing purposes and rely on Azure Automanage Machine Configuration instead.

Remediation

Thanks to Azure Automanage Machine Configuration, you don't need to set up a pull server as you must with DSC 1.1 since Azure Automanage Machine Configuration deals with this responsibility for you.

There are three different machine configuration assignment types that you can choose from:

- **Audit**: Only report; don't change anything.
- **ApplyAndMonitor**: Apply the configuration once, but if the configuration is changed, only report and don't remediate until it's triggered manually.
- **ApplyAndAutoCorrect**: Apply the configuration permanently. Once a change is made, the machine remediates at the next evaluation.

ApplyAndAutoCorrect is a great option that is similar to the pull configuration mode in DSC 1.1; it helps your systems become more secure as they remediate changes by themselves.

Check out the following link to learn more about DSC 2.0: **https://learn.microsoft.com/en-us/powershell/dsc/overview?view=dsc-2.0**.

DSC 3.0

DSC 3.0 is a preview release that is still under development as of April 2023.

This version supports cross-platform features and is supported by Azure Automanage Machine Configuration in Azure Policy. It can be installed with PSGallery by using the following command:

```
Install-Module -Name PSDesiredStateConfiguration -AllowPrerelease
```

For DSC 3.0, the remediation options are the same as for DSC 2.0.

You can find out more about DSC 3.0 by reading the official documentation: **https://learn.microsoft.com/en-us/powershell/dsc/overview?view=dsc-3.0**.

Configuration

To get started with DSC, you need a DSC configuration, which you can compile into a **.mof** file. Often, you will want to cover a scenario that has already been predefined as a resource and tweak it to your use case; in this case, you also want to include a predefined resource in your configuration.

DSC resources

Before creating your own DSC resources, always check whether there is already a resource that fits your use case; there's a multitude of existing resources that you can find on GitHub or PowerShell Gallery. Once you have found the right DSC resource for your use case, you can install it using **PowerShellGet**:

> **Install-Module -Name AuditPolicyDSC**

In this example, the **AuditPolicyDSC** resource would be installed, which helps you configure and manage the advanced audit policy on Windows machines.

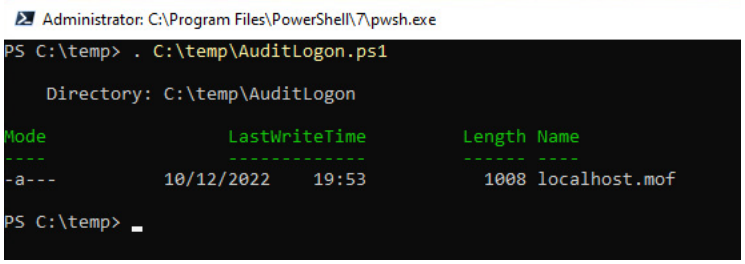
The following example shows a configuration that imports the **AuditPolicyDsc** resource and then uses it to ensure that all successful logons are being audited on the host, on which this configuration will be applied, via the equivalent advanced audit policy setting:

```
Configuration AuditLogon

{

    Import-DscResource -ModuleName AuditPolicyDsc

    Node 'localhost'

    {

        AuditPolicySubcategory LogonSuccess

        {

            Name      = 'Logon'

            AuditFlag = 'Success'

            Ensure    = 'Present'

        }

    }

}

AuditLogon
```

We must save this code in a file named **AuditLogon.ps1** under **C:\temp\** to dot source it:

```
> . C:\temp\AuditLogon.ps1
```

The following screenshot shows how the file is being compiled into a **.mof** file:



Administrator: C:\Program Files\PowerShell\7\pwsh.exe

```
PS C:\temp> . C:\temp\AuditLogon.ps1

    Directory: C:\temp\AuditLogon

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          10/12/2022    19:53           1008 localhost.mof

PS C:\temp> _
```

Figure 13.3 – Compiling your DSC configuration into a .mof file

Depending on the setup and the DSC version that you are running, you can now use this file to apply your DSC configuration to the system of your choice. Please refer to the official documentation for more information:

- DSC 1.1: **https://learn.microsoft.com/en-us/powershell/dsc/configurations/write-compile-apply-configuration?view=dsc-1.1**
- DSC 2.0: **https://learn.microsoft.com/en-us/powershell/dsc/concepts/configurations?view=dsc-2.0**
- DSC 3.0: **https://learn.microsoft.com/en-us/powershell/dsc/concepts/configurations?view=dsc-3.0**

Hardening systems and environments

In the end, you can harden PowerShell as much as you like; if the systems on which PowerShell is running are not protected, adversaries will make use of that if they have the chance. Therefore, it is important to also look at how you can harden the security of your infrastructure.

Security baselines

A great start to hardening your Windows systems – regardless of the server, domain controller, or client – are the so-called security baselines provided by Microsoft. These security baselines are part of Microsoft's **Security Compliance Toolkit** (**SCT**) 1.0, which can be downloaded from here: https://www.microsoft.com/en-us/download/details.aspx?id=55319.

Please be careful when applying security baselines!

You should never just apply a security baseline to a running production system. Before applying it, carefully audit your settings and evaluate them. Then, work on a plan to enroll your changes. Many settings are included that could break the functioning of your systems if they are not carefully planned for and enrolled.

When you download SCT, you will see that there are many files within it that you can download. Most of the files are the actual baselines (most baseline packages end with **Security Baseline.zip**).

But helpful tools are also included, including **LGPO**, **SetObjectSecurity**, and **Policy Analyzer**.

- **LGPO**: This tool can be used to perform local Group Policy Object (GPO) operations. You can use this tool to import settings into a local Group Policy, export a local Group Policy, parse a **registry.pol** file in **LGPO text** format, build a **registry.pol** file from **LGPO text**, and enable Group Policy client-side extensions for local policy processing. Since it's a command-line tool, LGPO can be used to automate local GPO operations.
- **SetObjectSecurity**: Using **SetObjectSecurity**, you can set the security descriptor for any type of Windows securable object – be it files, registry hives, event logs, and many more.
- **Policy Analyzer**: Policy Analyzer is a tool for comparing baselines and GPOs, but not only exported GPOs – you can also compare a GPO with your local policy. It can highlight differences between the policies, as well as help you spot redundancies.

All three tools are standalone, which means that you don't need to install them to use them.

You can use **PolicyAnalyzer** to check the current state of your machines. Download **PolicyAnalyzer** and the security baseline that you want to use to check your systems against. In our example, I used the *Windows Server 2022 Security Baseline* as my example baseline.

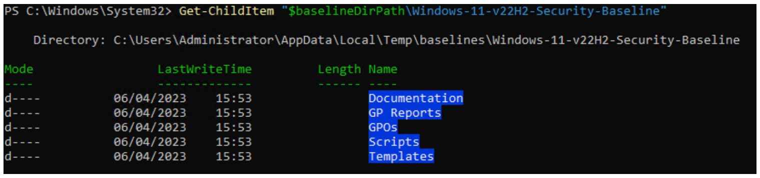
We looked into the **SCT** in ***Chapter 4***, *Detection – Auditing and Monitoring*, when we talked about auditing recommendations and EventList. There, we learned that security baselines contain auditing recommendations. But they also contain some system settings recommendations, such as the Lan Manager authentication level (**LmCompatibilityLevel**), which you can use to deny insecure authentication mechanisms in your domain. Please be extremely careful and audit which authentication protocols are used before applying this setting to the recommended one.

Before you can work with baselines, you will need to extract them. The following code snippet shows how you can use PowerShell to extract them:

```
$baselineZipPath = $env:TEMP + "\baselines\Windows 11 version 22H2 Security Baseline.zip"

$baselineDirPath = $env:TEMP + "\baselines\"

if ( !( Test-Path -Path $baselineDirPath ) ) {

    New-Item -ItemType Directory -Path $baselineDirPath

}

Expand-Archive -Path $baselineZipPath -DestinationPath $baselineDirPath
```

While the **$baselineZipPath** variable leads to the path where the baseline ZIP file is located, the **$baselineDirPath** variable points to the folder into which the baselines should be extracted. If the **$baselineDirPath** folder is not available yet, the folder will be created. The archive can be extracted using the **Expand-Archive** cmdlet.

After extracting a security baseline, you will find the five following folders in the ZIP file, as shown in the following screenshot:



```
PS C:\Windows\System32> Get-ChildItem "$baselineDirPath\Windows-11-v22H2-Security-Baseline"

    Directory: C:\Users\Administrator\AppData\Local\Temp\baselines\Windows-11-v22H2-Security-Baseline

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          06/04/2023    15:53                Documentation
d----          06/04/2023    15:53                GP Reports
d----          06/04/2023    15:53                GPOs
d----          06/04/2023    15:53                Scripts
d----          06/04/2023    15:53                Templates
```

Figure 13.4 – Contents of a security baseline

The actual baselines reside in the **GPOs** folder. You can use the files in there to import the baselines for testing purposes on a test system or to add them to Policy Analyzer.

When initially executing Policy Analyzer, you will see its starting interface, which looks as follows:

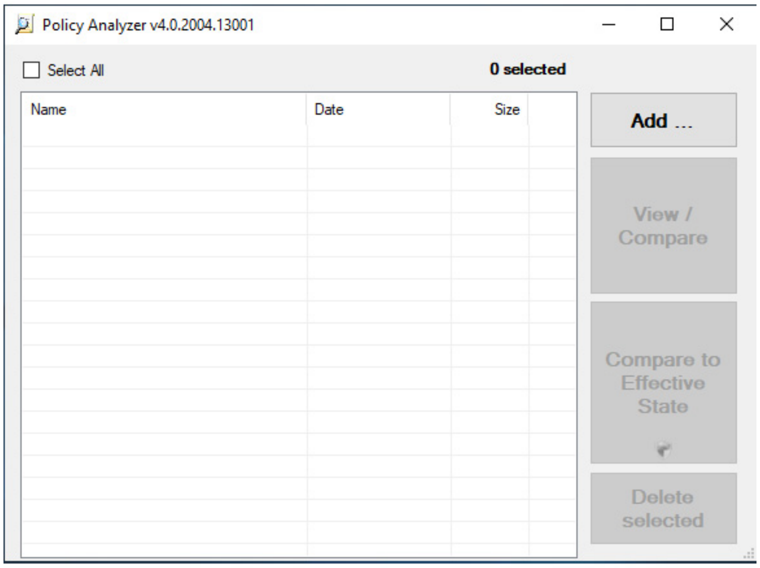
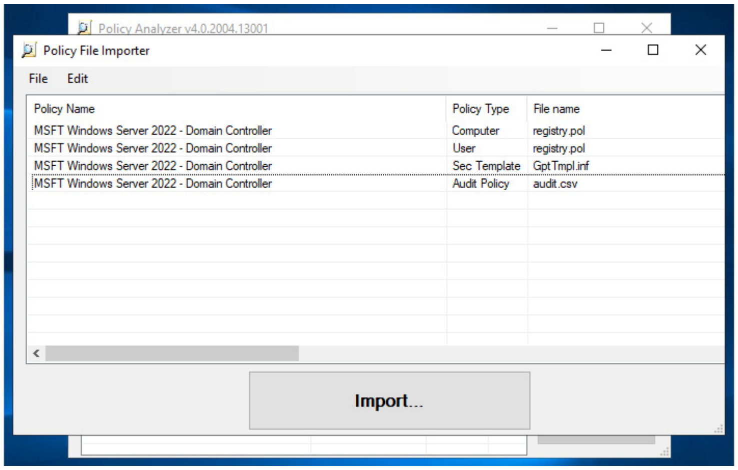


Figure 13.5 – Policy Analyzer

To get started, click on **Add ...** to add a new baseline to compare. Navigate to the GPOs folder within the selected baseline and select it. Since many baseline files are included that you won't want to add, you need to remove all the unnecessary ones by selecting them in the **Policy File Importer** view and removing them by using the *Delete* key on your keyboard.

In this example, I want to investigate a domain controller, so I deleted every other baseline except for the domain controller ones, as shown in the following screenshot:

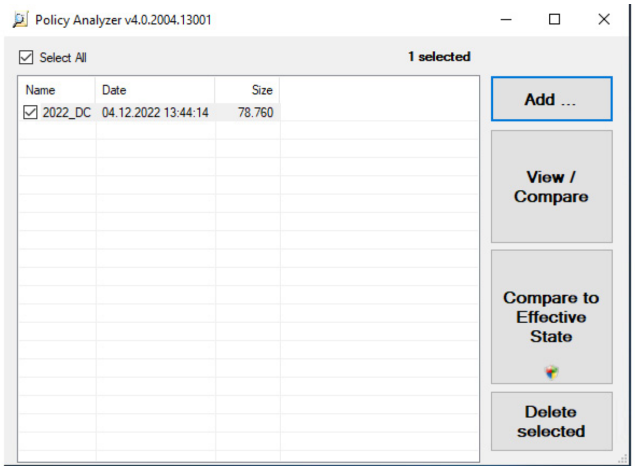


Policy Name	Policy Type	File name
MSFT Windows Server 2022 - Domain Controller	Computer	registry.pol
MSFT Windows Server 2022 - Domain Controller	User	registry.pol
MSFT Windows Server 2022 - Domain Controller	Sec Template	GptTmpl.inf
MSFT Windows Server 2022 - Domain Controller	Audit Policy	audit.csv

Figure 13.6 – Importing domain controller security baselines

Once all the necessary baselines are in the **Policy File Importer** view, click on **Import...** to import them. Before they are imported, you will be prompted to enter a name and save the policy. In this example, I have called the policy **2022_DC**.

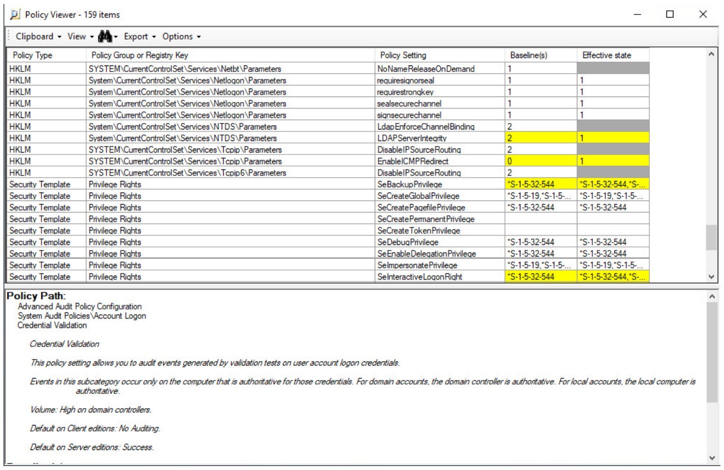
Once the baselines have been imported, you can either add another baseline or exported GPO to compare their settings (using **View / Compare**). Alternatively, you can also compare a baseline with the effective state of the current system (using **Compare to Effective State**):



Name	Date	Size
☑ 2022_DC	04.12.2022 13:44:14	78.760

Figure 13.7 – The imported 2022_DC policy within Policy Analyzer

In our example, I have selected the **2022_DC** policy and compared the **DC01** demo environment's domain controller with the effective state. A new window will appear so that you can investigate all the recommended and effective settings: if a setting remains white, then it matches, while if a setting is marked in gray, it's not been configured or has been left empty. Finally, if a setting is marked in yellow, that means that there's a conflict and there's a setting mismatch:



Policy Type	Policy Group or Registry Key	Policy Setting	Baseline(s)	Effective state
HKLM	SYSTEM\CurrentControlSet\Services\Netbt\Parameters	NoNameReleaseOnDemand	1	
HKLM	System\CurrentControlSet\Services\Netlogon\Parameters	requiresignorseal	1	1
HKLM	System\CurrentControlSet\Services\Netlogon\Parameters	requirestrongkey	1	1
HKLM	System\CurrentControlSet\Services\Netlogon\Parameters	sealsecurechannel	1	1
HKLM	System\CurrentControlSet\Services\Netlogon\Parameters	signsecurechannel	1	1
HKLM	System\CurrentControlSet\Services\NTDS\Parameters	LdapEnforceChannelBinding	2	
HKLM	System\CurrentControlSet\Services\NTDS\Parameters	LDAPServerIntegrity	2	1
HKLM	SYSTEM\CurrentControlSet\Services\Tcpip\Parameters	DisableIPSourceRouting	2	
HKLM	SYSTEM\CurrentControlSet\Services\Tcpip\Parameters	EnableICMPRedirect	0	1
HKLM	SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters	DisableIPSourceRouting	2	
Security Template	Privilege Rights	SeBackupPrivilege	*S-1-5-32-544	*S-1-5-32-544,*S-...
Security Template	Privilege Rights	SeCreateGlobalPrivilege	*S-1-5-19,*S-1-5-...	*S-1-5-19,*S-1-5-...
Security Template	Privilege Rights	SeCreatePagefilePrivilege	*S-1-5-32-544	*S-1-5-32-544
Security Template	Privilege Rights	SeCreatePermanentPrivilege		
Security Template	Privilege Rights	SeCreateTokenPrivilege		
Security Template	Privilege Rights	SeDebugPrivilege	*S-1-5-32-544	*S-1-5-32-544
Security Template	Privilege Rights	SeEnableDelegationPrivilege	*S-1-5-32-544	*S-1-5-32-544
Security Template	Privilege Rights	SeImpersonatePrivilege	*S-1-5-19,*S-1-5-...	*S-1-5-19,*S-1-5-...
Security Template	Privilege Rights	SeInteractiveLogonRight	*S-1-5-32-544	*S-1-5-32-544,*S-...

Policy Path:
Advanced Audit Policy Configuration
System Audit Policies\Account Logon
Credential Validation

Credential Validation

This policy setting allows you to audit events generated by validation tests on user account logon credentials.

Events in this subcategory occur only on the computer that is authoritative for those credentials. For domain accounts, the domain controller is authoritative. For local accounts, the local computer is authoritative.

Volume: High on domain controllers.

Default on Client editions: No Auditing.

Default on Server editions: Success.

Figure 13.8 – Comparing settings with Policy Analyzer

By doing this, you can check whether the recommendation reflects the current state of your configuration and what you need to configure if it doesn't match yet. Again – please do not just apply the recommendations without evaluating what these changes mean for your environment.

There are not only security baselines for domain controllers but also for member servers and clients, as well as for settings for other areas.

It is also possible to use PowerShell to interact with those baselines. Every baseline is an exported GPO that you can parse. The **gpreport.xml** file contains every setting that was configured in this GPO. So, if we import the **gpreport.xml** file of a security baseline as a PowerShell object, we can query all the settings available while referring to the XML syntax.

The following **Import-Baseline** function helps you with this task:

```
function Import-Baseline {

    [cmdletbinding()]

    param (

        [Parameter(Mandatory)]

        [string]$Path

    )

    $Item  = Join-Path -Path (Get-ChildItem -Path $Path -Filter "gpreport.xml" -Recurse | Select-Object

    if (Test-Path -Path $Item) {
        [xml]$Settings = Get-Content $Item

    }

    return $Settings.GPO

}
```

It looks for the first **gpreport.xml** file in the specified folder recursively and returns its settings as an XML object.

For example, if you want to access the recommended audit settings of the *Windows 10 22H2 – Computer*, baseline, we would first import it into the **$Baseline** variable, as shown in this code snippet:

```
> $Baseline = Import-Baseline -Path "C:\baselines\Windows-10-v22H2-Security-Baseline\GPOs\{AA94F467-FC14
```

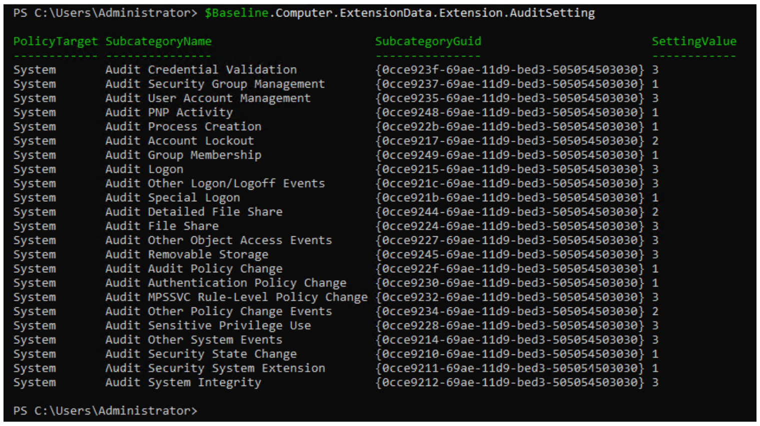
Now, all XML nodes are available and can be queried using the **$Baseline** variable. First, let's check the name of the baseline to make sure that we imported the right one:

```
> $Baseline.Name

MSFT Windows 10 22H2 - Computer
```

Next, we want to access the audit settings, which are located under the **Computer.ExtensionData.Extension.AuditSetting** node:

```
> $Baseline.Computer.ExtensionData.Extension.AuditSetting
```

As shown in the following screenshot, you can see every recommended audit setting and its value – that is, the output of the command:



```
PS C:\Users\Administrator> $Baseline.Computer.ExtensionData.Extension.AuditSetting

PolicyTarget SubcategoryName                        SubcategoryGuid                        SettingValue
------------ ---------------                        ---------------                        ------------
System       Audit Credential Validation            {0cce923f-69ae-11d9-bed3-505054503030} 3
System       Audit Security Group Management        {0cce9237-69ae-11d9-bed3-505054503030} 1
System       Audit User Account Management          {0cce9235-69ae-11d9-bed3-505054503030} 3
System       Audit PNP Activity                     {0cce9248-69ae-11d9-bed3-505054503030} 1
System       Audit Process Creation                 {0cce922b-69ae-11d9-bed3-505054503030} 1
System       Audit Account Lockout                  {0cce9217-69ae-11d9-bed3-505054503030} 2
System       Audit Group Membership                 {0cce9249-69ae-11d9-bed3-505054503030} 1
System       Audit Logon                            {0cce9215-69ae-11d9-bed3-505054503030} 3
System       Audit Other Logon/Logoff Events        {0cce921c-69ae-11d9-bed3-505054503030} 3
System       Audit Special Logon                    {0cce921b-69ae-11d9-bed3-505054503030} 1
System       Audit Detailed File Share              {0cce9244-69ae-11d9-bed3-505054503030} 2
System       Audit File Share                       {0cce9224-69ae-11d9-bed3-505054503030} 3
System       Audit Other Object Access Events       {0cce9227-69ae-11d9-bed3-505054503030} 3
System       Audit Removable Storage                {0cce9245-69ae-11d9-bed3-505054503030} 3
System       Audit Audit Policy Change              {0cce922f-69ae-11d9-bed3-505054503030} 1
System       Audit Authentication Policy Change     {0cce9230-69ae-11d9-bed3-505054503030} 1
System       Audit MPSSVC Rule-Level Policy Change {0cce9232-69ae-11d9-bed3-505054503030} 3
System       Audit Other Policy Change Events       {0cce9234-69ae-11d9-bed3-505054503030} 2
System       Audit Sensitive Privilege Use          {0cce9228-69ae-11d9-bed3-505054503030} 3
System       Audit Other System Events              {0cce9214-69ae-11d9-bed3-505054503030} 3
System       Audit Security State Change            {0cce9210-69ae-11d9-bed3-505054503030} 1
System       Audit Security System Extension        {0cce9211-69ae-11d9-bed3-505054503030} 1
System       Audit System Integrity                 {0cce9212-69ae-11d9-bed3-505054503030} 3

PS C:\Users\Administrator>
```

Figure 13.9 – Querying the audit setting XML nodes of the baseline

Here, you can see **SettingValue**, which indicates whether it is recommended to audit for **Success** (**1**), **Failure** (**2**), or for both **Success and Failure** (**3**). **0** would indicate that there it is explicitly not recommended to audit this setting (that is, *audit setting disabled*) – a value that you will never find in the security baselines distributed by Microsoft.

With this, you can now query all imported XML nodes that were configured in this GPO.

Another great tool that can help you monitor your security settings for compliance using **DSC** is the **BaselineManagement** module. With its help, you can convert baselines as well as Group Policies into DSC configuration scripts (**.ps1**) and **.mof** files, which you can use to monitor the compliance of your systems.

You can find more information on how to set this up in the GPO DSC quick start documentation: **https://learn.microsoft.com/en-us/powershell/dsc/quickstarts/gpo-quickstart**.

Applying security updates and patch compliance monitoring

During my work as Premier Field Engineer at Microsoft, I performed a lot of security assessments for companies and organizations of all sizes, all around the world. One of the most critical, but also most common, findings in those security assessments was missing updates. Believe it or not, but of all the organizations I assessed, in perhaps 2% of the assessments, I found that all updates were installed. For all other assessments, at least one critical update was missing.

In addition to other attack vectors, such as social engineering and abusing legitimate admin capabilities, missing updates are a common reason for systems being breached: if a security update was released, this means that a vulnerability was fixed and that knowledge about this vulnerability exists publicly. Adversaries can even reverse-engineer the released patch to find out what exactly was fixed.

This means that as soon as an update is released, it is only a race against time before adversaries will have an exploit ready. And if a system is missing a patch, it will be vulnerable in no time.

So, apply security updates as soon as possible. Establish a plan to test and install your updates as soon as possible after a release and prioritize this properly.

It is not enough to just install updates – you also need to verify whether all needed updates are installed regularly.

Checking for updates

Many organizations use WSUS and/or SCCM to deploy and monitor security updates. Although it's a great method to deploy them, it is not enough for checking that all required updates were installed. Therefore, if you have only relied on WSUS or SSCM so far, you need to set up another mechanism to check whether all the relevant updates have been installed.

Often, organizations only deploy Windows Security updates and forget about other products. But there are so many tools that are installed on servers worldwide that come with Microsoft Visual C++ or other programs. Once installed, they are never updated, even though critical vulnerabilities exist, which leaves a hole in the infrastructure for adversaries to exploit.

For earlier Windows versions, checking whether all relevant updates were installed could be achieved by using **Microsoft Baseline Security Analyzer** (**MBSA**) and the **WSUS offline catalog** known as **wsusscn2.cab**. But since MBSA got deprecated and is no longer developed, there are new ways to scan for patch compliance.

One option is to use the PowerShell **Scan-UpdatesOffline.ps1** script, which is available in PowerShell Gallery: https://www.powershellgallery.com/packages/Scan-UpdatesOffline/1.0.

You can install the script using **Install-Script**:

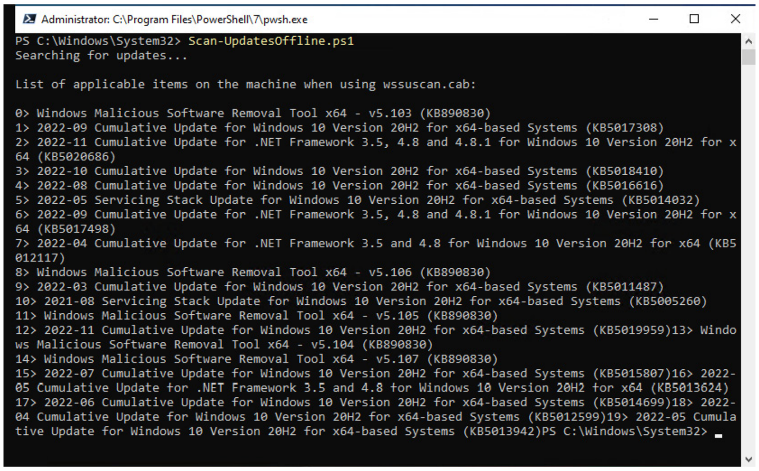
```
> Install-Script -Name Scan-UpdatesOffline
```

Before running the script, download the latest **wsusscn2.cab** file from http://go.microsoft.com/fwlink/?linkid=74689 and save it under **C:\temp\wsusscn2.cab**:

```
> Invoke-WebRequest http://go.microsoft.com/fwlink/?linkid=74689 -OutFile c:\temp\wsusscn2.cab
```

It is important to note that this specific path is hardcoded into the **Scan-UpdatesOffline** script, so make sure that the **wsusscn2.cab** file is in the right location before running this script.

Once everything is in place, you can start the scan using **Scan-UpdatesOffline.ps1**, as shown in the following screenshot:



```
PS C:\Windows\System32> Scan-UpdatesOffline.ps1
Searching for updates...

List of applicable items on the machine when using wssuscan.cab:

0> Windows Malicious Software Removal Tool x64 - v5.103 (KB890830)
1> 2022-09 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5017308)
2> 2022-11 Cumulative Update for .NET Framework 3.5, 4.8 and 4.8.1 for Windows 10 Version 20H2 for x
64 (KB5020686)
3> 2022-10 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5018410)
4> 2022-08 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5016616)
5> 2022-05 Servicing Stack Update for Windows 10 Version 20H2 for x64-based Systems (KB5014032)
6> 2022-09 Cumulative Update for .NET Framework 3.5, 4.8 and 4.8.1 for Windows 10 Version 20H2 for x
64 (KB5017498)
7> 2022-04 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 20H2 for x64 (KB5
012117)
8> Windows Malicious Software Removal Tool x64 - v5.106 (KB890830)
9> 2022-03 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5011487)
10> 2021-08 Servicing Stack Update for Windows 10 Version 20H2 for x64-based Systems (KB5005260)
11> Windows Malicious Software Removal Tool x64 - v5.105 (KB890830)
12> 2022-11 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5019959)13> Windo
ws Malicious Software Removal Tool x64 - v5.104 (KB890830)
14> Windows Malicious Software Removal Tool x64 - v5.107 (KB890830)
15> 2022-07 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5015807)16> 2022-
05 Cumulative Update for .NET Framework 3.5 and 4.8 for Windows 10 Version 20H2 for x64 (KB5013624)
17> 2022-06 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5014699)18> 2022-
04 Cumulative Update for Windows 10 Version 20H2 for x64-based Systems (KB5012599)19> 2022-05 Cumula
tive Update for Windows 10 Version 20H2 for x64-based Systems (KB5013942)PS C:\Windows\System32>
```

Figure 13.10 – Scanning for missing updates

Now, you can use this script to create regular checks to ensure the latest updates are installed on your servers and clients. Make sure you always download the latest **wsusscn2.cab** file before scanning.

Since you can only use this method to check for Windows and Microsoft product updates, make sure you also keep an inventory of all available software in your organization and monitor patch compliance.

Avoiding lateral movement

Lateral movement is a technique that attackers use to dive deeper into a network to compromise endpoints, servers, and identities.

Once an adversary has managed to compromise a device within an organization, they try to gather more credentials and identities to use them to move laterally and compromise the entire network.

To detect lateral movement, organizations can use PowerShell to monitor remote logon event logs, specifically event ID *4624*. This event ID provides information on successful logons, including the logon's type, process, and authentication package. For example, to get all events with event ID *4624* that have a logon type of 3 (network logon) from the last 7 days, you can use the following code snippet:

```
> Get-WinEvent -FilterHashtable @{LogName='Security'; ID=4624; StartTime=(Get-Date).AddDays(-7)} | Where
```

Logon type 3 indicates that the logon attempt was made over the network. This can happen, for example, when a user connects to a network share or when a process running on one computer accesses resources on another computer.

By monitoring logon-type-3 events, organizations can detect attempts by an attacker to access network resources from a compromised system, which can be an early sign of lateral movement within the network. Depending on your network, it makes sense to fine-grain this example and adjust it to your needs.

Please refer to ***Chapter 4***, *Detection – Auditing and Monitoring*, to learn more about how to leverage the different event logs for detecting malicious activities.

You should abide by the following guidelines to avoid lateral movement as much as possible:

- Enforce unique passwords for workstations and servers by using **Local Administrator Password Solution (LAPS)**
- Implement a **Red Forest** for Active Directory administrators, also called **Enhanced Security Administrative Environment (ESAE)**
- Implement a tiering model and have your administrators use **Privileged Access Workstations (PAWs)** for their administrative tasks

- Restrict logins and maintain proper credential hygiene
- Have updates installed as soon as possible
- Audit your identity relations by using tools such as BloodHound or SharpHound

Of course, this is not a 100% guarantee that attackers will not be able to move laterally, but it already covers a lot and will keep attackers busy for some time.

Multi-factor authentication for elevation

Multi-Factor Authentication (**MFA**) always adds another layer of security to your administrative accounts. Of course, people can be tricked into allowing authentication, but with MFA, it is so much harder for adversaries to steal and abuse identities.

There are many options that you can use for MFA. Depending on your scenario, you can leverage the following:

- Smartcard authentication
- Windows Hello
- OAuth hardware tokens
- OAuth software tokens
- Fido2 security keys
- Biometrics
- SMS or voice calls
- An authenticator application (for example, Microsoft Authenticator)

Time-bound privileges (Just-in-Time administration)

A great option for following the principles of least privilege is to implement time-bound privileges, also known as **Just-in-Time administration**. Using this approach, no administrators have any rights by default.

Once they request privilege elevation, a timestamp is bound to their privileges. Once the specified time has run out, the privileges don't apply any longer.

If an account is compromised, the adversary can't do any harm since the rights of the account were not requested by the administrator. Usually, the elevation request comes with MFA.

Moreover, **privileged identity management** (**PIM**) and **privileged access management** (**PAM**) solutions can be used to automate the process of granting and revoking time-bound privileges. These solutions provide a centralized platform for managing and monitoring privileged access across an organization.

They can also offer additional security measures, such as approval workflows, audit trails, and session recordings to ensure accountability and compliance. Implementing PIM and PAM solutions can greatly enhance the security of time-bound privileges and reduce the risk of unauthorized access to critical systems and data.

Attack detection – Endpoint Detection and Response

Another really important point is to have a product in place to detect attacks and react to them. There are many great products out there that can help you with this task. Make sure that the product of your choice also supports PowerShell and helps you detect suspicious commands that were launched via PowerShell and other command-line tools.

Microsoft's solution, for example, is called Microsoft Defender for Endpoint. But other vendors provide similar solutions.

Enabling free features from Microsoft Defender for Endpoint

Even if you do not use Microsoft Defender for Endpoint, various features are free to use without any subscription:

- Hardware-based isolation/Application Guard
- Attack surface reduction rules
- Controlled folder access
- Removable storage protection
- Network protection
- Exploit Guard
- Windows Defender Firewall with advanced security

Many of these features can even be used while Microsoft Defender is disabled. Check out the ASR capabilities to learn more about these features: **https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/overview-attack-surface-reduction?view=o365-worldwide#configure-attack-surface-reduction-capabilities**.

Summary

This chapter sums up this book on PowerShell security. It was not meant to provide deep technical information, but rather an outlook of what else can be done to improve the security of your network. With this, you have a good overview of what to do next and what to look up.

You got some insights into secure scripting and what tools you can use to improve your scripting security. You also learned what DSC is and how to get started. And last but not least, you also got insights into hardening your systems.

I hope you enjoyed this book and could make the most of it. Happy scripting!

Further reading

If you want to explore some of the topics that were mentioned in this chapter, take a look at these resources:

LAPS

- LAPS: **https://www.microsoft.com/en-us/download/details.aspx?id=46899**

PSScriptAnalyzer

- PSScriptAnalyzer on GitHub: **https://github.com/PowerShell/PSScriptAnalyzer**
- PSScriptAnalyzer reference: **https://learn.microsoft.com/en-us/powershell/module/psscriptanalyzer/?view=ps-modules**
- PSScriptAnalyzer module overview: **https://learn.microsoft.com/en-us/powershell/utility-modules/psscriptanalyzer/overview?view=ps-modules**

Security baselines and SCT

- Microsoft SCT 1.0 – How to use it: **https://learn.microsoft.com/en-us/windows/security/operating-system-security/device-management/windows-security-configuration-framework/security-compliance-toolkit-10**
- **LGPO.exe** – Local Group Policy Object Utility, v1.0: **https://techcommunity.microsoft.com/t5/microsoft-security-baselines/lgpo-exe-local-group-policy-object-utility-v1-0/ba-p/701045**
- New and Updated Security Tools: **https://techcommunity.microsoft.com/t5/microsoft-security-baselines/new-amp-updated-security-tools/ba-p/1631613**

Security Updates

- A new version of the Windows Update offline scan file, **wsusscn2.cab**, is available for advanced users: **https://support.microsoft.com/en-us/topic/a-new-version-of-the-windows-update-offline-scan-file-wsusscn2-cab-is-available-for-advanced-users-fe433f4d-44f4-28e3-88c5-5b22329c0a08**
- Detailed information for developers who use the Windows Update offline scan file can be found here: **https://support.microsoft.com/en-us/topic/detailed-information-for-developers-who-use-the-windows-update-offline-scan-file-51db1d9e-038b-0b15-16e7-149aba45f295**
- What is Microsoft Baseline Security Analyzer and its uses?: **https://learn.microsoft.com/en-us/windows/security/threat-protection/mbsa-removal-and-guidance**

VBS

- Virtualization-based security: **https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-vbs**

You can also find all the links mentioned in this chapter in the GitHub repository for ***Chapter 13*** – there's no need to manually type in every link: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter13/Links.md**.