BeyondTrust

Network Security Scanner
PowerShell Integration Guide

Powered By Retina

TC:5/15/2019

Table of Contents

BeyondTrust Network Security Scanner PowerShell Integration

This guide provides the instructions and software requirements for the BeyondTrust Network Security Scanner PowerShell Integration.

For more information about its features, benefits, functionality, and basic procedures, please see the Network Security Scanner User Guide.

Get Started

Prerequisites

- Microsoft .Net Framework 4.7 or later
- Windows Management Framework 4.0 or later

Import the Module

Before cmdlets will be available, the **.PowerShell module** must be imported into PowerShell.

The.PowerShell dll is located at **<Retina Install>\API\PowerShell**.

Open a PowerShell session, and use the `Import-Module` cmdlet:

```
Import-Module .\Retina.PowerShell.dll
```

If you are not in the **<Retina Install>\API\PowerShell** directory, you will need to specify the full path. Place this code into your PowerShell profile to automatically import the module.

Samples

The **<Retina Install>\API\PowerShell\Samples** directory has a number of scripts that use many of the available cmdlets. These scripts are helpful examples of cmdlet usage and can serve as a starting point for your own scripts.

Get Help

Use the PowerShell `Get-Help` to get an overview of each cmdlet's functionality

```
Get-Help Start-RetinaScan
```

```
NAME
    Start-RetinaScan
SYNOPSIS
    Start a scan.
SYNTAX
    Start-RetinaScan [-RetinaScan <RetinaScanDecorator>] [-Id <Guid>] [-Name <String>] [-
DatabaseFileName <String>] [<CommonParameters>]

    Start-RetinaScan [-RetinaScan <RetinaScanDecorator>] [-Id <Guid>] [-Name <String>] [-
DatabaseFileName <String>] -ScheduleMode <ScheduleMode> -StartDateTime
    <DateTime> [<CommonParameters>]

    Start-RetinaScan [-RetinaScan <RetinaScanDecorator>] [-Id <Guid>] [-Name <String>] [-
DatabaseFileName <String>] -CustomSchedule <String> [<CommonParameters>]

    Start-RetinaScan [-Id <Guid>] [-Name <String>] [-DatabaseFileName <String>] -XCCDFProfile
<String> -XCCDFFilePath <String> [<CommonParameters>]
```

```
    Start-RetinaScan [-Id <Guid>] [-Name <String>] [-DatabaseFileName <String>] -RequestFile
<String> -JobFile <String> [<CommonParameters>]

DESCRIPTION
    Starts a specific scan that is passed in, or creates and outputs a new scan with default
options set.

RELATED LINKS

REMARKS
    To see the examples, type: "get-help Start-RetinaScan -examples".
    For more information, type: "get-help Start-RetinaScan -detailed".
    For technical information, type: "get-help Start-RetinaScan -full".
```

List available cmdlets

PowerShell's `Get-Command` cmdlet lists all available cmdlets.

```
Get-Command -Module Retina.PowerShell
```

CommandType	Name	ModuleName
cmdlet	Add-RetinaScanAddressGroup	Retina.PowerShell
cmdlet	Add-RetinaScanAuditGroup	Retina.PowerShell
cmdlet	Add-RetinaScanPort	Retina.PowerShell
cmdlet	Add-RetinaScanPortGroup	Retina.PowerShell
cmdlet	Add-RetinaScanStoredCredential	Retina.PowerShell
cmdlet	Add-RetinaScanTargetNamedHost	Retina.PowerShell
cmdlet	Add-RetinaTargetNamedHost	Retina.PowerShell
cmdlet	Get-RetinaAddressGroup	Retina.PowerShell
cmdlet	Get-RetinaAuditGroup	Retina.PowerShell
cmdlet	Get-RetinaCentralPolicySettings	Retina.PowerShell
cmdlet	Get-RetinaConfiguration	Retina.PowerShell
cmdlet	Get-RetinaDebugDump	Retina.PowerShell
cmdlet	Get-RetinaEngineStatus	Retina.PowerShell
cmdlet	Get-RetinaLicenseInfo	Retina.PowerShell
cmdlet	Get-RetinaPaths	Retina.PowerShell
cmdlet	Get-RetinaPortGroup	Retina.PowerShell
cmdlet	Get-RetinaRegistrationInfo	Retina.PowerShell
cmdlet	Get-RetinaScan	Retina.PowerShell
cmdlet	Get-RetinaScanResults	Retina.PowerShell
cmdlet	Get-RetinaScanStatus	Retina.PowerShell
cmdlet	Get-RetinaSCAPScanResults	Retina.PowerShell

cmdlet	Get-RetinaSchedule	Retina.PowerShell
cmdlet	Get-RetinaSerialNumberInformation	Retina.PowerShell
cmdlet	Get-RetinaStoredCredentials	Retina.PowerShell
cmdlet	Get-RetinaUpdateServer	Retina.PowerShell
cmdlet	Get-RetinaVersion	Retina.PowerShell
cmdlet	New-RetinaAddressGroup	Retina.PowerShell
cmdlet	New-RetinaAddressGroupAddress	Retina.PowerShell
cmdlet	New-RetinaAuditGroup	Retina.PowerShell
cmdlet	New-RetinaAuditGroupAudit	Retina.PowerShell
cmdlet	New-RetinaPortGroup	Retina.PowerShell
cmdlet	New-RetinaPortGroupPort	Retina.PowerShell
cmdlet	New-RetinaRegistrationInfo	Retina.PowerShell
cmdlet	New-RetinaScan	Retina.PowerShell
cmdlet	New-RetinaSCAPScan	Retina.PowerShell
cmdlet	New-RetinaSCAPScanBenchmark	Retina.PowerShell
cmdlet	New-RetinaStoredCredential	Retina.PowerShell
cmdlet	Remove-RetinaAddressGroup	Retina.PowerShell
cmdlet	Remove-RetinaAuditGroup	Retina.PowerShell
cmdlet	Remove-RetinaData	Retina.PowerShell
cmdlet	Remove-RetinaPortGroup	Retina.PowerShell
cmdlet	Remove-RetinaScan	Retina.PowerShell
cmdlet	Remove-RetinaScanAddressGroup	Retina.PowerShell
cmdlet	Remove-RetinaScanAuditGroup	Retina.PowerShell
cmdlet	Remove-RetinaScanPort	Retina.PowerShell
cmdlet	Remove-RetinaScanPortGroup	Retina.PowerShell
cmdlet	Remove-RetinaScanStoredCredential	Retina.PowerShell
cmdlet	Remove-RetinaStoredCredential	Retina.PowerShell
cmdlet	Restart-RetinaEngineService	Retina.PowerShell
cmdlet	Resume-RetinaScan	Retina.PowerShell
cmdlet	Set-RetinaCentralPolicy	Retina.PowerShell
cmdlet	Set-RetinaConfiguration	Retina.PowerShell
cmdlet	Set-RetinaRegistrationInfo	Retina.PowerShell
cmdlet	Set-RetinaSerialNumber	Retina.PowerShell
cmdlet	Set-RetinaUpdateServer	Retina.PowerShell
cmdlet	Start-RetinaEngineService	Retina.PowerShell
cmdlet	Start-RetinaScan	Retina.PowerShell
cmdlet	Start-RetinaSCAPScan	Retina.PowerShell
cmdlet	Stop-RetinaEngineService	Retina.PowerShell

cmdlet	`Stop-RetinaScan`	`Retina.PowerShell`
cmdlet	`Stop-RetinaScheduledScan`	`Retina.PowerShell`
cmdlet	`Suspend-RetinaScan`	`Retina.PowerShell`
cmdlet	`Test-RetinaCentralPolicy`	`Retina.PowerShell`
cmdlet	`Test-RetinaUpdateServer`	`Retina.PowerShell`
cmdlet	`Update-Retina`	`Retina.PowerShell`
cmdlet	`Update-RetinaConfig`	`Retina.PowerShell`
cmdlet	`Update-RetinaScheduledScan`	`Retina.PowerShell`

Environment

Network Security Scanner Version

The `Get-RetinaVersion` cmdlet reports the current versions for:

- The scanner application
- Audits database
- Retina Scripting Engine

```
Get-RetinaVersion
```

```
Audits    Product    ScriptingEngine
------    -------    ---------------
3096      5.24.0     1.0.0
```

Engine Status

Your scripts can check the status of the engine with the `Get-RetinaEngineStatus` cmdlet.

```
Get-RetinaEngineStatus
Running
```

Engine Service

The engine service can be started, stopped, and restarted with the following commands.

```
Start-RetinaEngineService
Stop-RetinaEngineService
Restart-RetinaEngineService
```

Engine Service Memory Dump

If needed, you can use the `Get-RetinaDebugDump` command to generate a .dmp file from the engine service. By default, the file is placed in the **Logs/Exception** folder, or a path can be specified via the `-Path` parameter.

```
Get-RetinaDebugDump
```

Remove Data

You can remove job, queue, schedule and log data with the `Remove-RetinaData` command and the appropriate switches.

```
Remove-RetinaData
  - Removes Queue, Schedule, Log and Job data

Remove-RetinaData -Queue
  - Remove only Queue data

Remove-RetinaData -Schedule
  - Remove only Schedule data

Remove-RetinaData -Logs
  - Remove only Log data

Remove-RetinaData -Jobs
  - Remove only Job data
```

Paths

Important paths can be retrieved with the `Get-RetinaPaths` cmdlet:

```
Get-RetinaPaths
```

```
Config:          C:\Program Files (x86)\BeyondTrust\Retina 5\
Data:            C:\Program Files (x86)\BeyondTrust\Retina 5\Database
Groups:          C:\Program Files (x86)\BeyondTrust\Retina 5\Groups
Jobs:            C:\Program Files (x86)\BeyondTrust\Retina 5\Scans\Jobs\
Log:             C:\Program Files (x86)\BeyondTrust\Retina 5\Logs\
Output:          C:\Program Files (x86)\BeyondTrust\Retina 5\Scans\
Reports:         C:\Program Files (x86)\BeyondTrust\Retina 5\\Reports
Requests:        C:\Program Files (x86)\BeyondTrust\Retina 5\Scans\ScanRequests\
Root:            C:\Program Files (x86)\BeyondTrust\Retina 5\
Scans:           C:\Program Files (x86)\BeyondTrust\Retina 5\Scans\
```

Licensing

`Get-RetinaLicenseInfo` returns an object containing:

- `RegistrationInformation` : Name and company contact details
- `SerialNumber`
- `IsProductLicensed` : `True` or `False`

```
RegistrationInformation : Retina.Service.License.RegistrationInfo
```

```
SerialNumber            : 55555-44444-33333-22222-11111-00000
IsProductLicensed       : True
```

Serial Number

```
$LicenseInfo = Get-RetinaLicenseInfo

Write-Output $LicenseInfo.SerialNumber

00000-11111-22222-33333-44444-55555
```

`Set-RetinaSerialNumber` will register a new or alternate serial number with Network Security Scanner.

```
Set-RetinaSerialNumber -SerialNumber 0000-11111-22222-33333-44444-55555
```

Configuration

You can set various configuration settings using the `Get-RetinaConfiguration`, `Set-RetinaConfiguration`, and `Update-RetinaConfig` cmdlets:

```
<#
-------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script
Manage Retina's settings
-------------------------------------------------------------------------
#>
# Get the current settings/configuration and assign values to the properties
$Config = Get-RetinaConfiguration

# View the configuration categories
Write-Output $Config

# Set the logging settings
$Config.Logging.MaxLogDiskSpace = 100 # Set maximum log size to 100 GB

# Set the health and recovery settings
$Config.HealthAndRecovery.ThreadScanAttempts = 3 # Set the number of scan attempts

# Set the settings related to exception and snapshot files
$Config.ExceptionAndSnapshot.MaxExceptionDumps = 15 # Limit the number of exception files to 15

# Write the settings to the configuration file
Set-RetinaConfiguration -Configuration $Config

# Refresh the application's settings from the configuration file
Update-RetinaConfig
```

Registration Information

The registration information contains contact information for the person associated with the current license.

```
Get-RetinaRegistrationInfo
```

```
Address   : 2 Elm
City      : Winchester
Company   : SecurityCompany
Country   : USA
Email     : iancognito@securitycompany.com
Fax       :
FirstName : Ian
LastName  : Cognito
Phone     : 949-555-1212
>State     : CA
Title     : Software Architect
ZipCode   : 92596
FullName  : Ian Cognito
```

New registration information can be set by creating a new `RegistrationInfo` object, setting the appropriate properties, and passing this object to the `Set-RetinaRegistrationInfo` cmdlet. The following script snippet illustrates this process.

```
$registrationInformation = New-RetinaRegistrationInfo
$registrationInformation.Address = "2 Elm"
$registrationInformation.City = "Winchester"
$registrationInformation.Company = "SecurityCompany"
$registrationInformation.Country = "USA"
$registrationInformation.Email = "iancognito@securitycompany.com"
$registrationInformation.Fax = ""
$registrationInformation.FirstName = "Ian"
$registrationInformation.LastName = "Cognito"
$registrationInformation.Phone = "949-555-1212"
$registrationInformation.State = "CA"
$registrationInformation.Title = "Software Architect"
$registrationInformation.ZipCode = "92596"
Set-RetinaRegistrationInfo -RegistrationInformation $registrationInformation
```

Central Policy

The `Get-RetinaCentralPolicySettings` cmdlet reports the current registration information for a Central Policy Server.

```
Get-RetinaCentralPolicySettings

MachineName : IANCOGNITO856A
Server      : 127.0.0.1
Type        : REM 3
```

```
Enabled     : True
```

The following sample script illustrates the use of `Set-RetinaCentralPolicy` for configuring support for a Central Policy server.

```
<#
-------------------------------------------------------------------------------
 Retina Network Security Scanner
 Sample PowerShell Script


Manage Central Policy
-------------------------------------------------------------------------------
#>


# Retrieve and review the current Central Policy settings
Get-RetinaCentralPolicySettings

# Add the current machine to central policy
# The Agent Name of the current host as it would appear in the Central Policy server
$MachineName = "Demo"

# The IP address or hostname of the Central Policy server
$ServerName = Read-Host -Prompt "Enter the IP Address of the Central Policy server"

# Central Policy version
$Type = "v2"

# Central policy password
$SecurePassword = Read-Host -Prompt "Enter password" -AsSecureString

# Set up Retina for Central Policy
Set-RetinaCentralPolicy -MachineName $MachineName -Server $ServerName -Type $Type -
SecurePassword $SecurePassword -Enabled

# Test the current Central Policy settings
Test-RetinaCentralPolicy

# Verify the central policy settings
Get-RetinaCentralPolicySettings
```

Update Server

`Get-RetinaUpdateServer` returns information about the server used to check for and retrieve updates.

```
Get-RetinaUpdateServer
```

```
Server       : update.eeye.com
UpdateScript : /UpdateServer/
Protocol     : https
```

`Set-RetinaUpdateServer` sets the server the scanner uses to retrieve updates. You can verify successful communication between the scanner and the update server with the `Test-RetinaUpdateServer` cmdlet. The following sample script demonstrates the process of managing the update server:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script
Retrieve and set the update server
-------------------------------------------------------------------------------
#>

# Get the current update server settings
Get-RetinaUpdateServer

# Set the new update server
$URI = "https://update.eeye.com/UpdateServer/"
Set-RetinaUpdateServer -Uri $URI

# Verify the connectivity to the server
Test-RetinaUpdateServer
```

You can update the application and its components using the `Update-Retina` cmdlet:

```
# Start an update
Update-Retina
```

Create, Manage, and Schedule Scans

Create a Scan

You can customize a scan by creating a new reference to the scan using the `New-RetinaScan` cmdlet:

```
$Scan = New-RetinaScan -Name "DailyScan" -DatabaseFilename "DailyScanDatabase"
Write-Output $Scan
```

```
Id                : 945caccf-454d-4ddc-a938-a3c9adcbaf54
Name              : DailyScan
DataBaseName      : DailyScanDatabase
StoredCredentials : {}
JobOptions        : Retina.PowerShell.Scan.OptionsViewModel
AddressGroups     : {}
AuditGroups       : {}
PortGroups        : {}
RetinaJob         : 945caccf454d4ddca938a3c9adcbaf54_RetinaJob.xml
```

This creates an empty scan that can be configured.

> ***Note:*** *There are no* `PortGroups` *or* `AuditGroups` *assigned.*

Add or Remove Single Port

The `Add-RetinaScanPort` cmdlet associates a single port with the scan. Ports can be added to multiple scans using the pipeline. The example script demonstrates the procedure:

```
<#
-------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Adding a single port to a scan
-------------------------------------------------------------------------
#>

# Create a scan to work with
$Scan = New-RetinaScan

# Add a single port to the scan by passing the scan to the -RetinaScan parameter
Add-RetinaScanPort -RetinaScan $Scan -Protocol both -Number 80
```

```
# Add a single port to multiple scans using the pipeline
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Add-RetinaScanPort -Protocol both -Number 80
```

The ports can be removed from a scan using `Remove-RetinaScanPort`. The example script below demonstrates the process:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Removing a single port from a scan
-------------------------------------------------------------------------------
#>

# Create a scan to work with
$Scan = New-RetinaScan

# Remove a single port from the scan by passing the scan to the -RetinaScan parameter
Remove-RetinaScanPort -RetinaScan $Scan -Protocol both -Number 80

# Remove a single port from multiple scans using the pipeline
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Remove-RetinaScanPort -Protocol both -Number 80
```

Add or Remove Port Groups

Port Groups are pre-configured lists of ports the scanner will look for when performing a scan.

The `Add-RetinaScanPortGroup` cmdlet associates a `PortGroup` with a scan. Port Groups can be added to multiple scans using the pipeline. The example script demonstrates the procedure:

```
# Create a scan to work with
$Scan = New-RetinaScan

# Add an existing Port Group to the scan by passing it to the -RetinaScan parameter
Add-RetinaScanPortGroup -PortGroupName 'Common Ports' -RetinaScan $Scan

# Add a Port Group to multiple scans by piping the scans to Add-RetinaScanPortGroup
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Add-RetinaScanPortGroup -PortGroupName 'Common Ports'
```

The Port Groups can be removed from a scan identically using `Remove-RetinaScanPortGroup`. The example script below demonstrates the process:

```
# Remove a Port Group from the scan by passing the scan to the -RetinaScan parameter
Remove-RetinaScanPortGroup -PortGroupName 'Discovery Ports' -RetinaScan $Scan

<# Remove a Port Group from multiple scans by piping the scans to Remove-RetinaScanPortGroup#>
$Scan1,$Scan2 | Remove-RetinaScanPortGroup -PortGroupName 'Discovery Ports'
```

Add or Remove Audit Groups

Audit Groups are pre-configured audit collections that the engine uses when performing a scan.

The `Add-RetinaScanAuditGroup` cmdlet associates an `AuditGroup` with a scan reference:

```
# Create a scan to work with
$Scan = New-RetinaScan

<# Add an existing Audit Group to the scan by passing the scan to the -RetinaScan parameter#>
Add-RetinaScanAuditGroup -AuditGroupName 'All Audits' -RetinaScan $Scan

# Add an audit group to more than 1 scan
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Add-RetinaScanAuditGroup -AuditGroupName 'All Audits'
```

Similarly, `Remove-RetinaScanAuditGroup` removes an association with a specific `AuditGroup` from a scan:

```
<# Remove an existing Address Group from a scan by passing the scan to the -RetinaScan parameter#>
Remove-RetinaScanAddressGroup -AddressGroupName 'All Audits' -RetinaScan $Scan

# Remove an Address Group from multiple scans using the pipeline
$Scan1,$Scan2 | Remove-RetinaScanAddressGroup -AddressGroupName 'All Audits'
```

Add or Remove Address Groups

Address Groups are pre-configured collections of IP addresses and ranges that the engine targets when performing a scan.

The `Add-RetinaScanAddressGroup` cmdlet associates an `AddressGroup` with a scan reference:

```
<# Add an existing Address Group to a scan by passing the scan to the -RetinaScan paramete#>r
$Scan = New-RetinaScan
Add-RetinaScanAddressGroup -RetinaScan $Scan -AddressGroupName Localhost

# Add an existing Address Group to multiple scans using the pipeline
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Add-RetinaScanAddressGroup -AddressGroupName Localhost
```

Similarly, `Remove-RetinaScanAddressGroup` removes an association with a specific `AddressGroup` from a scan:

```
<# Remove an existing Address Group from a scan by passing the scan to the -RetinaScan parameter#>
Remove-RetinaScanAddressGroup -AddressGroupName Localhost -RetinaScan $Scan

# Remove an Address Group from multiple scans using the pipeline
$Scan1,$Scan2 | Remove-RetinaScanAddressGroup -AddressGroupName Localhost
```

Add or Remove Stored Credentials

The `Add-RetinaScanAddressGroup` cmdlet associates a `StoredCredential` with a scan reference:

```
<#


-------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Create and add a credential to a scan
-------------------------------------------------------------------------
#>

# Create a scan to work with
$Scan = New-RetinaScan

# Create a general credential
$User = "User1"
$Description = "MyCredential"
$SecurePWD = Read-Host -Prompt "Enter password" -AsSecureString
$StoredCred = New-RetinaStoredCredential -StoredCredentialType Any -Description $Description -
UserName $User -SecurePassword $SecurePWD

# Create an SSH credential
$SSHDescription = "SSH Credential"
$SSHUser = "User2"
$SSHStoredCred = New-RetinaStoredCredential -StoredCredentialType SSH-PlainText -Description
$SSHDescription -UserName $SSHUser -SecurePassword $SecurePWD -ElevationType None

# Add the credentials to the scan
Add-RetinaScanStoredCredential -RetinaScan $Scan -CredentialName $StoredCred.Description
Add-RetinaScanStoredCredential -RetinaScan $Scan -CredentialName $SSHDescription
```

Similarly, `Remove-RetinaScanStoredCredential` removes an association with a specific `StoredCredential` from a scan:

```
# Remove a single credential from a scan
$Scan | Remove-RetinaScanStoredCredential -CredentialName MyCredential

# Remove all credentials from a scan
$Scan.Credentials = $null
```

Add Named Host

The `Add-RetinaScanTargetNamedHost` cmdlet associates a named host with a scan reference:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Adding a NETBIOS target to a scan
-------------------------------------------------------------------------------
#>

# Create a scan to work with
$Scan = New-RetinaScan

# Add an existing NETBIOS target to the scan
Add-RetinaScanTargetNamedHost -RetinaScan $Scan -Name "Target 1"

# Add a NETBIOS target to more than 1 scan
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Add-RetinaScanTargetNamedHost -Name "Target 1"
```

Run a Scan

`Start-RetinaScan` is used to perform scans with several options job customization.

```
Start-RetinaScan
```

The `Start-RetinaScan` cmdlet is versatile and can handle different sets of options depending on your intentions.

In its simplest form, it can be used to start a scan with no options.

```
Id                : 55ce6e93-5b16-4ab6-9934-05585666f8dc
Name              : 55ce6e935b164ab6993405585666f8dc
DataBaseName      : 55ce6e935b164ab6993405585666f8dcDatabase
StoredCredentials : {}
JobOptions        : Retina.PowerShell.Scan.OptionsViewModel
AddressGroups     : {}
AuditGroups       : {All Audits}
PortGroups        : {Common Ports}
RetinaJob         : 5a49fb76317a4cadb2cb44074df4e1dc_RetinaJob.xml
```

In this scenario, a unique ID will be generated and used in the `Id`, `Name` and `DatabaseName` file properties. The scan is created and started immediately. The output lists the basic properties of the scan for reference.

Using a couple of the name properties, we can start to configure and name this scan:

```
Start-RetinaScan -Name "DailyScan" -DatabaseFileName "DailyScanDatabase"
```

```
Id                : 88afb8a7-93ad-4df1-a7b5-e1c587e9d31a
Name              : DailyScan
DataBaseName      : DailyScanDatabase
StoredCredentials : {}
JobOptions        : Retina.PowerShell.Scan.OptionsViewModel
AddressGroups     : {}
AuditGroups       : {All Audits}
PortGroups        : {Common Ports}
```

You can store a reference to the scan, and then use this reference throughout your script.

In the following example, the scan object is stored in the `$Scan` variable. This scan is configured with friendly names for the scan and its database file.

```
$Scan = Start-RetinaScan -Name "DailyScan" -DatabaseFileName "DailyScanDatabase"
```

After we have stored a reference to our scan, we can use other cmdlets to perform actions on it.

Manage Scans

The Network Security Scanner includes the ability to control a scan's lifetime. Scans can be started, suspended, resumed, and aborted.

Start

Once you have created a new scan and configured it to your needs, you can start the scan using the `Start-RetinaScan` cmdlet. Simply pass the scan into `Start-RetinaScan`'s `-RetinaScan` parameter. You can also pipe the scan or multiple scans into `Start-RetinaScan`:

```
Start-RetinaScan
```

Suspend

An active scan can be suspended for any reason with the `Suspend-RetinaScan` cmdlet.

```
Suspend-RetinaScan
```

Resume

A suspended scan can be resumed with the `Resume-RetinaScan` cmdlet.

```
Resume-RetinaScan
```

Stop

An active or queued scan can be aborted with the `Stop-RetinaScan` cmdlet.

```
Stop-RetinaScan
```

The example script below demonstrates the usage of these cmdlets:

```
<#
-----------------------------------------------------------------------------
 Retina Network Security Scanner
 Sample PowerShell Script

 Manage scans
-----------------------------------------------------------------------------
#>

# Create and start a new scan
$Scan = Start-RetinaScan

# Pause the current scan
Suspend-RetinaScan -Name $Scan.Name
```

```
# Resume the scan
Resume-RetinaScan -Name $Scan.Name

# Stop the scan
Stop-RetinaScan -RetinaScan $Scan

# Prompt to remove the scan data
if ( (Read-Host -Prompt "Would you like to remove the scan data? (y,n)") -eq "y")
{
       Remove-RetinaData


}
```

Get a Scan

You can retrieve a previously created scan using the `Get-RetinaScan` cmdlet:

```
# When called with no parameters, the most recent scan job is returned.
$latestScan = Get-RetinaScan

# Append "-all" to get a list of all historical scan jobs.
$allSScans = Get-RetinaScan -All

# Use the "-Name" option to get the status of a specific scan by name.
$mySampleScanName = Get-RetinaScan -Name 'MySampleScanName'

# Use the "-Id" option to get the status of a specific scan by ID.
$myScanById = Get-RetinaScan -Id 'D80B2F4195DE40CBB6C8E6EE8440769E'
```

This retrieves a scan that can be modified.

Remove a scan

You can remove a previously ran scan using the `Remove-RetinaScan` cmdlet:

```
# Remove a scan by Job Id
Remove-RetinaScan -Id 'D80B2F4195DE40CBB6C8E6EE8440769E'

# Remove a scan by name
Remove-RetinaScan -Name 'MySampleScanName'

# Remove a scan by instance
$mySampleScanName = Get-RetinaScan -Name 'MySampleScanName'
Remove-RetinaScan -RetinaScan 'MySampleScanName'
```

Get Scan Status

`Get-RetinaScanStatus` returns status information for one or more scans.

When called with no options, the cmdlet returns a status for the most recent scan.

```
Get-RetinaScanStatus

JobId     : 88afb8a793ad4df1a7b5e1c587e9d31a
JobName   : DailyScan
JobStatus : Completed
```

Adding the `-All` switch returns the status of all scans.

```
Get-RetinaScanStatus -All
```

```
JobId               : bdb704d053674fe394aae78316dd2aae
JobName             : bdb704d053674fe394aae78316dd2aae
JobStatus           : Completed
EndTime             : 4/25/2016 9:18:06 AM
StartTime           : 4/25/2016 9:17:18 AM
ScheduledStartTime  : 4/25/2016 9:15:00 AM
JobProgress         : 1 of 1 completed
GroupId             : 8EE499B4AAA84D289C4FEE6534A4F3E4
JobId               : 8fcaa710dcb545b88d33c2fbb7805737
JobName             : 8fcaa710dcb545b88d33c2fbb7805737
JobStatus           : Completed
StartTime           : 4/25/2016 9:12:53 AM
EndTime             : 4/25/2016 9:13:41 AM
ScheduledStartTime  : 4/25/2016 9:00:00 AM
JobProgress         : 1 of 1 completed
GroupId             : 0D155D44DBE247D4B5F9947A731F4BD4
JobId               : 45ac0428285f403e92cdee09adb70514
JobName             : 45ac0428285f403e92cdee09adb70514
JobStatus           : Completed
StartTime           : 4/25/2016 9:13:40 AM
EndTime             : 4/25/2016 9:14:28 AM
ScheduledStartTime  : 4/25/2016 9:00:00 AM
JobProgress         : 1 of 1 completed
GroupId             : 839C6D15722D49D9A6CAB647B05F555E
```

Get the status of a scan or group of scans by passing the scan into the `-RetinaScan` parameter of the cmdlet or by piping scan objects into the cmdlet.

```
# Start a quick scan of the local machine
$Scan = Start-RetinaScan

# Retrieve the status of the current scan
Get-RetinaScanStatus -RetinaScan $Scan

Start multiple scans
$Scan1 = New-RetinaScan
$Scan2 = New-RetinaScan
$Scan1,$Scan2 | Start-RetinaScan

# Retrieve the status  both scans by piping the scans into the cmdlet
$Scan1,$Scan2 | Get-RetinaScanStatus
```

Get Scan Results

`Get-RetinaScanResults` returns result data for one or more scans.

When called with no options, the cmdlet returns result data for the most recent scan. Call the `Save` method and pass a file path to save the report to a specific location.

```
$Path = "C:\ReportData\DailyScan.xml"
$ScanResults = Get-RetinaScanResults
$ScanResults.Save($Path)
```

Get Results for Specific Scans

Results for a specific scan can be obtained using one or more of the command properties.

Passing a scan reference to the `-RetinaScan` parameter can uniquely identify the scan to process.

```
$Results = Get-RetinaScanResults -RetinaScan $Scan
```

Passing the name of a scan to the `-Name` parameter returns the scan results of the specified scan name.

```
$Results = Get-RetinaScanResults -Name "MyScan"
```

Use the `$Results` variable to get the scan results in .xml format.

The level of detail in data returned from `Get-RetinaScanResults` can be tailored for specific needs.

- `IncludePorts`: Include specific information about the ports detected and the applications utilizing them.
- `IncludeProcesses`: Include detailed information about which processes were detected on scanned machines.
- `IncludeServices`: Include detailed information about which services were found on scanned machines.
- `IncludeHardware`: Include specific information about what hardware was detected on scanned machines.
- `IncludeShares`: Include details about the shares present on scanned machines.
- `IncludeUsers`: Include specific information about the users and groups found on the scanned machines.
- `IncludeSoftware`: Include details about which software applications were detected on the scanned machines.
- `IncludeCertificates`: Include details about certificates installed on the target system.
- `IncludeProtocols`: Include details about protocols the target system supports.
- `IncludeAll`: Include all detailed information. Ports, Processes, Services, Hardware, Shares, Users and Software.
- `IncludeAlerts`: Include alerts generated during a scan.
- `IncludeAuditsNotVerified`: Include audits...
- `IncludeAuditsNotVulnerable`: Include audit information about vulnerabilities that were not found in the scan.
- `IncludeAuditsVulnerable`: Include audit information about vulnerabilities the were found in the scan.
- `IncludeGeneral`: Include general information about each machine in the scan.
- `IncludeUserGroups`: Include information about user groups for each machine in the scan.

```
Get-RetinaScanResults -RetinaScan $Scan -Name "DailyScan" -IncludeAll
```

Schedule Scans

Create a Scheduled Scan

You can create a reocurring scan using the `Start-RetinaScan` cmdlet and passing the cmdlet a scan as well as a schedule string:

```
<#
-------------------------------------------------------------------------------
 Retina Network Security Scanner
 Sample PowerShell Script

Add and remove scheduled scans
-------------------------------------------------------------------------------
#>

# Generate GUID for to be used as the Job ID.
$JobId = [GUID]::NewGuid().ToString("N")

<#
 Set the schedule frequency.  Valid values are based on the below information
  SYNTAX:
   ONCE   :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
  MEANING:
   Run once on given date and time.
  SAMPLE:
   ONCE:01:2015:12:26:11:15 :*
  SAMPLE DESCRIPTION:
   Run once on Dec 26, 2015 @ 11:15AM

  SYNTAX:
   DAILY  :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
  MEANING:
   Every day at a given time, starting on a specified date.
  SAMPLE:
   DAILY:01:2015:12:26:02:33:*
  SAMPLE DESCRIPTION:
   Daily scan starting on Dec 26, 2015 @ 2:33AM

  SYNTAX:
   DAILY  :02: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
  MEANING:
   Weekdays only at a given time, starting on a specified date.
  SAMPLE:
   DAILY:02:2015:12:26:02:33:*
  SAMPLE DESCRIPTION:
   Weekdays starting on Dec 26, 2015 @ 2:33AM

  SYNTAX:
   DAILY  :03: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : <INTERVAL>
  MEANING:
   Every <INTERVAL> days (1-365) at a given time, starting on a specified date.
  SAMPLE:
```

```
   DAILY:03:2015:12:26:02:33:*:5
  SAMPLE DESCRIPTION:
   Every 5 days starting on Dec 26, 2015 @ 2:33AM

  SYNTAX:
   WEEKLY :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<DAY 1>,<DAY 2>,<DAY N>]: <INTERVAL>
  MEANING:
   Every <INTERVAL> weeks (1-52) at a given time, on specific days of the week, starting on a
specified date.
  SAMPLE:
   WEEKLY:01:2015:12:26:11:15:*:[4,5]:2
  SAMPLE DESCRIPTION:
   Thursday (4) and Friday (5) of every 2 weeks starting on Dec 26, 2015 @ 11:15AM

  SYNTAX:
   MONTHLY:01: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<MONTH 1>,<MONTH 2>,<MONTH N>]: <DAY>
  MEANING:
   Specific <DAY> (1-31) of specific months, starting on a specified date.
  SAMPLE:
   MONTHLY:01:2015:12:26:15:48:*:[4,7]:26
  SAMPLE DESCRIPTION:
   Day 26 of April (4) and July (7) starting on Dec 26, 2015 @ 3:48PM.

  SYNTAX:
   MONTHLY:02: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<MONTH 1>,<MONTH 2>,<MONTH N>]: <OCCURANCE> :
<DAYOFWEEK>
  MEANING:<OCCURANCE> (1-5) of <DAYOOFWEEK> (1-7) <DAY> of specific months, starting on a
specified date.
  SAMPLE:
   MONTHLY:02:2015:12:26:15:48:*:[4,7]:2:5
  SAMPLE DESCRIPTION:
   Second (2) Thursday (5) of April (4) and July (7) starting on Dec 26, 2015 @ 3:48PM.

  SYNTAX:
   now
  MEANING:
   Run the scan immediately.
  SAMPLE:
   now
  SAMPLE DESCRIPTION:
   Run the scan immediately
#>

# Set the actual value, in this example the scan will run daily staring on Feb 01, 2016 @ 10:00PM
$Schedule = "DAILY:01:2016:02:01:22:00:*"

# Create an instance of the new scan job with a scan name
$Scan = New-RetinaScan -Name "Sample Job"

# Configure any custom options
$Scan.JobOptions.RemoteAgent.PerformLocalScanning = $false

# Configure other job options
```

```
$Scan.JobOptions.PerformOsDetection = $true
$Scan.JobOptions.GetReverseDns = $true
$Scan.JobOptions.GetNetBiosName = $true
$Scan.JobOptions.GetMacAddress = $true
$Scan.JobOptions.PerformTraceroute = $true
$Scan.JobOptions.EnumerateRegistry = $true
$Scan.JobOptions.EnumerateUsers = $true
$Scan.JobOptions.EnumerateShares = $true
$Scan.JobOptions.EnumerateFiles = $true
$Scan.JobOptions.EnumerateHotFixes = $true
$Scan.JobOptions.EnumerateNamedPipes = $true
$Scan.JobOptions.EnumerateMachineInformation = $true
$Scan.JobOptions.EnumerateAuditPolicy = $true
$Scan.JobOptions.EnumeratePerUserRegistrySettings = $true
$Scan.JobOptions.EnumerateGroups = $true
$Scan.JobOptions.EnumerateProcesses = $true
$Scan.JobOptions.EnumerateServices = $true
$Scan.JobOptions.EnumerateUserAndGroupPrivileges = $true
$Scan.JobOptions.EnumerateHardware = $true
$Scan.JobOptions.EnumerateSoftware = $true
$Scan.JobOptions.EnumerateCertificates = $true
$Scan.JobOptions.EnumerateDatabases = $true
$Scan.JobOptions.PerformIPProtocolScanning = $false
$Scan.JobOptions.RandomizePortList = $false
$Scan.JobOptions.RandomizePortList = $false

# Create and set a credential to use.  Repeat this step for each credential to add.
$SecurePassword = Read-Host -Prompt "Enter password" -AsSecureString
$StoredCredential = New-RetinaStoredCredential -StoredCredentialType Any -Description
"MyCredential" -UserName "User" -SecurePassword $SecurePassword
$Scan | Add-RetinaScanStoredCredential -CredentialName $StoredCredential.Description

# Schedule the new scan
Start-RetinaScan -RetinaScan $Scan -CustomSchedule $Schedule
```

Update a Scheduled Scan

You can update the settings of a scheduled scan using the `Update-RetinaScheduledScan` cmdlet:

```
<#
------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Update a scan job
------------------------------------------------------------------------------
#>

# Get a list of the current retina schedule
Get-RetinaSchedule
```

```
# Specify the Job ID of the scan you want to modify.
$JobId = $null

# Make sure that a JobId was specified.
$JobId = Read-Host -Prompt 'Input a Job ID'

# Set the schedule frequency.  Valid values are based on the below information
#  SYNTAX:
#    ONCE   :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
#  MEANING:


#    Run once on given date and time.
#  SAMPLE:
#    ONCE:01:2015:12:26:11:15 :*
#  SAMPLE DESCRIPTION:
#    Run once on Dec 26, 2015 @ 11:15AM#
#  SYNTAX:
#    DAILY  :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
#  MEANING:
#    Every day at a given time, starting on a specified date.
#  SAMPLE:
#    DAILY:01:2015:12:26:02:33:*
#  SAMPLE DESCRIPTION:
#    Daily scan starting on Dec 26, 2015 @ 2:33AM#
#  SYNTAX:
#    DAILY  :02: <YYYY>:<MM>:<DD>: <HH>:<MM> :*
#  MEANING:
#    Weekdays only at a given time, starting on a specified date.
#  SAMPLE:
#    DAILY:02:2015:12:26:02:33:*
#  SAMPLE DESCRIPTION:
#    Weekdays starting on Dec 26, 2015 @ 2:33AM#
#  SYNTAX:
#    DAILY  :03: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : <INTERVAL>
#  MEANING:
#    Every <INTERVAL> days (1-365) at a given time, starting on a specified date.
#  SAMPLE:
#    DAILY:03:2015:12:26:02:33:*:5
#  SAMPLE DESCRIPTION:
#    Every 5 days starting on Dec 26, 2015 @ 2:33AM#
#  SYNTAX:
#    WEEKLY :01: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<DAY 1>,<DAY 2>,<DAY N>]: <INTERVAL>
#  MEANING:
#    Every <INTERVAL> weeks (1-52) at a given time, on specific days of the week, starting on a
specified date.
#  SAMPLE:
#    WEEKLY:01:2015:12:26:11:15:*:[4,5]:2
#  SAMPLE DESCRIPTION:
#    Thursday (4) and Friday (5) of every 2 weeks starting on Dec 26, 2015 @ 11:15AM#
#  SYNTAX:
#    MONTHLY:01: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<MONTH 1>,<MONTH 2>,<MONTH N>]: <DAY>
#  MEANING:
```

```
#    Specific <DAY> (1-31) of specific months, starting on a specified date.
#  SAMPLE:
#    MONTHLY:01:2015:12:26:15:48:*:[4,7]:26
#  SAMPLE DESCRIPTION:
#    Day 26 of April (4) and July (7) starting on Dec 26, 2015 @ 3:48PM.
#
#  SYNTAX:
#    MONTHLY:02: <YYYY>:<MM>:<DD>: <HH>:<MM> :* : [<MONTH 1>,<MONTH 2>,<MONTH N>]: <OCCURANCE> :
<DAYOFWEEK>
#  MEANING:
#    <OCCURANCE> (1-5) of <DAYOOFWEEK> (1-7) <DAY> of specific months, starting on a specified
date.
#  SAMPLE:
#     MONTHLY:02:2015:12:26:15:48:*:[4,7]:2:5
#  SAMPLE DESCRIPTION:
#    Second (2) Thursday (5) of April (4) and July (7) starting on Dec 26, 2015 @ 3:48PM.
#
#  SYNTAX:
#    now
#  MEANING:
#    Run the scan immediately.
#  SAMPLE:
#     now
#  SAMPLE DESCRIPTION:
#    Run the scan immediately

$ScheduleString = "WEEKLY:01:2015:12:26:11:15:*:[4,5]:2"
# Modify the schedule of an existing retina job
Update-RetinaScheduledScan -Id $JobId -CustomSchedule $ScheduleString

# Get a list of the current retina schedule
Get-RetinaSchedule -Id $JobId
```

Remove a Scheduled Scan

You can stop a scheduled scan using the `Stop-RetinaScheduledScan` cmdlet:

```
# You can get a list of active Job IDs by using "Get-RetinaSchedule"
# Get the first scheduled scan
$ScheduledJob = Get-RetinaSchedule | Select-Object -First 1

# If there were scheduled jobs, then remove the first one that was returned
if ($ScheduledJob)
{
    Stop-RetinaScheduledScan -Id $ScheduledJob.JobId
}
```

SCAP Scanning

Create a SCAP Scan

You can customize a SCAP scan by creating a new reference to a scan using the `New-RetinaSCAPScan` cmdlet:

```
# Set the profile
$XCCDFProfile = "Level 1 - Member Server"

# Get the XCCDF file
$XCCDFFolderPath = "$((Get-RetinaPaths).Root)Database\XCCDF\benchmarks\CIS\Windows_10"

# Get the latest xccdf.xml file associated with the SCAP scan
$XCCDFFilePath = Get-ChildItem -Path $XCCDFFolderPath -Filter "*-xccdf.xml" | Select-Object -
ExpandProperty FullName -First 1

# Create a scan to work with
$ScapScan = New-RetinaScapScan -Name "MyScapScan" -XCCDFProfile $XCCDFProfile -XCCDFFilePath
$XCCDFFilePath
```

```
Id                : 945caccf-454d-4ddc-a938-a3c9adcbaf54
Name              : MyScapScan
DataBaseName      : MyScapScan
StoredCredentials : {}
JobOptions        : Retina.PowerShell.Scan.OptionsViewModel
AddressGroups     : {}
AuditGroups       : {}
PortGroups        : {}
RetinaJob         : 945caccf454d4ddca938a3c9adcbaf54_RetinaJob.xml
```

This creates an empty scan that can be configured.

***Note:** There are no `AuditGroups` assigned.*

***Tip:** By using the commands described in "Create a Scan" on page 15, you can configure SCAP scans.*

```
# Add an existing Address Group to the scan by passing it to the -RetinaScan parameter
Add-RetinaScanAddressGroup -AddressGroupName Localhost -RetinaScan $ScapScan

# Create benchmark and assign its properties by passing the values of the properties to the
corresponding parameters
$Benchmark1 = New-RetinaSCAPScanBenchmark -XCCDFProfile $XCCDFProfile -XCCDFFilePath
$XCCDFFilePath
```

```
# Intialize the scan, passing in the Benchmark
$ScapScan = Start-RetinaScapScan -RetinaScan $ScapScan -Benchmarks $Benchmark1 -Name $Name
```

Run a SCAP Scan

`Start-RetinaSCAPScan` is used to perform scans with several options for job customization.

```
# Specify the Profile, ProfileFolderPath for constructing a Benchmark
# Set the profile
$XCCDFProfile = "xccdf_org.cisecurity.benchmarks_profile_Level_1"
# Get the XCCDF file


$XCCDFFolderPath = "$((Get-RetinaPaths).Root)Database\XCCDF\benchmarks\CIS\Windows_10"
# Get the latest xccdf.xml file associated with the SCAP scan
$XCCDFFilePath = Get-ChildItem -Path $XCCDFFolderPath -Filter "*-xccdf.xml" | Select-Object -
ExpandProperty FullName -First 1

# Create benchmark and assign its properties by passing the values of the properties to the
corresponding parameters
$Benchmark1 = New-RetinaSCAPScanBenchmark -XCCDFProfile $XCCDFProfile -XCCDFFilePath
$XCCDFFilePath

# Start the SCAP scan
$ScapScan = Start-RetinaSCAPScan -Benchmarks $Benchmark1
```

***Note:** When running a SCAP scan, you can also limit user enumeration To enable this feature, use the parameter `-LimitedDCUserEnumeration`.*

Single Benchmark

To run an SCAP scan with a single benchmark, pass the file path of the `XCCDFFile` to the `XCCDFFilePath` parameter and the profile name to the `XCCDFProfile` parameter of the `Start-RetinaSCAPScan` cmdlet:

```
# Set the name of the scan (optional).
$Name = "SCAPSampleScan"

# Set the name of the database (optional).
$DatabaseName = "SCAPSampleScanDatabase"

# Set the profile
$XCCDFProfile = "Level 1 - Member Server"

# Get the XCCDF file
```

```
$XCCDFFolderPath = "$((Get-RetinaPaths).Root)Database\XCCDF\benchmarks\CIS\Windows_2012_R2"
$XCCDFFilePath = Get-ChildItem -Path $XCCDFFolderPath -Filter "*-xccdf.xml" | Select-Object -
ExpandProperty FullName -First 1

# Start an SCAP scan, passing valid values to the XCCDFProfile and XCCDFFilePath parameters of
Start-RetinaSCAPScan
$Scan1 = Start-RetinaSCAPScan -Name $Name -DatabaseFileName $DatabaseName -XCCDFProfile
$XCCDFProfile -XCCDFFilePath $XCCDFFilePath
```

Multiple Benchmarks

To run an SCAP scan with multiple benchmarks:

- Create the benchmark objects
- Assign the profile name to the `XCCDFProfile` property for each benchmark
- Assign the file path of the benchmark to the `XCCDFFile` property for each benchmark

The values of the benchmark object's properties can also be assigned by passing the values to the `XCCDFProfile` and `XCCDFFile` parameters:

```
# Set the name of the scan (optional).
$Name = "SCAPSampleScan"

# Set the name of the database (optional).
$DatabaseName = "SCAPSampleScanDatabase"

# Set the profile
$XCCDFProfile = "Level 1 - Member Server"

# Get the XCCDF file
$XCCDFFolderPath = "$((Get-
RetinaPaths).Root)Database\XCCDF\benchmarks\CIS\Windows_2012_R2"
$XCCDFFilePath = Get-ChildItem -Path $XCCDFFolderPath -Filter "*-xccdf.xml" | Select-Object -
ExpandProperty FullName -First 1

# Create benchmark and assign its properties by passing the values of the properties to the
corresponding parameters
$Benchmark1 = New-RetinaSCAPScanBenchmark -XCCDFProfile $XCCDFProfile -XCCDFFilePath
$XCCDFFilePath

# Create an empty SCAP Benchmark, the assign the profile and file you want to use to the
corresponding properties
$Benchmark2 = New-RetinaSCAPScanBenchmark
$Benchmark2.XccdfFile = $XCCDFFilePath
$Benchmark2.XccdfProfile = $XCCDFProfile -replace "Level 1","Level 2"

# Intialize the scan, passing in the 2 Benchmarks just created

$Scan2 = Start-RetinaSCAPScan -Name $Name -DatabaseFileName $DatabaseName -Benchmarks
```

```
$Benchmark1,$Benchmark2
```

Both procedures are demonstrated in the sample script:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Start a SCAP scan and generate a report
-------------------------------------------------------------------------------
#>

# Set the name of the scan (optional).
$Name = "SCAPSampleScan"

# Set the name of the database (optional).
$DatabaseName = "SCAPSampleScanDatabase"

# Set the profile
$XCCDFProfile = "Level 1 - Member Server"

# Get the XCCDF file
$XCCDFFolderPath = "$((Get-
RetinaPaths).Root)Database\XCCDF\benchmarks\CIS\Windows_2012_R2"

if (Test-Path -Path "$XCCDFFolderPath\*-xccdf.xml")
{
       # Get the latest xccdf.xml file associated with the SCAP scan
       $XCCDFFilePath = Get-ChildItem -Path $XCCDFFolderPath -Filter "*-xccdf.xml" |
Select-Object -ExpandProperty FullName -First 1

       # Start an SCAP scan, passing valid values to the XCCDFProfile and
XCCDFFilePath parameters of Start-RetinaSCAPScan
       $Scan1 = Start-RetinaSCAPScan -Name $Name -DatabaseFileName $DatabaseName -XCCDFProfile
$XCCDFProfile -XCCDFFilePath $XCCDFFilePath


       # Start an SCAP scan, passing 1 or more Benchmarks to the Benchmarks parameter

       # Create benchmark and assign its properties by passing the values of the
properties to the corresponding parameters
       $Benchmark1 = New-RetinaSCAPScanBenchmark -XCCDFProfile $XCCDFProfile -
XCCDFFilePath $XCCDFFilePath

       # Create an empty SCAP Benchmark, then assign the profile and file you want to use to the
corresponding properties
       $Benchmark2 = New-RetinaSCAPScanBenchmark
       $Benchmark2.XccdfFile = $XCCDFFilePath
       $Benchmark2.XccdfProfile = $XCCDFProfile -replace "Level 1","Level 2"
```

```
        # Intialize the scan, passing in the 2 Benchmarks just created
        $Scan2 = Start-RetinaSCAPScan -Name $Name -DatabaseFileName $DatabaseName -Benchmarks
$Benchmark1,$Benchmark2

        # Scan results

        # Wait for the scans to complete
        while ( ( (Get-RetinaScanStatus -RetinaScan $Scan1).JobStatus -ne "Completed") `

                -or  ( (Get-RetinaScanStatus -RetinaScan $Scan2).JobStatus -ne "Completed"))
        {
                Start-Sleep -Seconds 5
        }
        # Output the file paths
        $Scan1,$Scan2 | ForEach-Object `
        {
                $ResultFilePath = Get-RetinaSCAPScanResults -Id $_.Id
                Write-Output "Scan results saved to: $($ResultFilePath.FullName)"
        }
}
else
{
                Write-Warning "The profile path $XCCDFFolderPath was not found"
                pause
}
```

Address Groups

Create Address Groups

Address Groups can be created by passing the name of the new Address Group to the `New-RetinaAddressGroup` cmdlet. Single addresses, multiple addresses along with `CIDR`, `Range`, and `Named` address types can be added to the new Address Group's **Addresses** property:

```
#-------------------------------------------------------------------------------
# Retina Network Security Scanner
# Sample PowerShell Script
#
# Create an Address Group
#-------------------------------------------------------------------------------

# Create a new address group and give it a name
$AddressGroup = New-RetinaAddressGroup -Name "SampleAddressGroup"

# Assign a single IP to the list of addresses
$AddressGroup.Addresses = New-RetinaAddressGroupAddress -Type Single -Value 127.0.0.1

# Add an additional single IP Address to the Address Group
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type Single -Value 127.0.0.1

# Remove all items from an Address Group
$AddressGroup.Addresses = $null

# Add a range of IP addresses to the existing single IP can be specified with a start and end
address seperated by a '-'
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type Range -Value 10.100.50.10-
10.100.50.100

# The Omit parameter determines whether the targets described by the element are to be included or
omitted from the scan.
# Omit an single IP from a scan
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type Single -Value 10.100.50.50 -Omit

# A CIDR block denoting a bitmasked range of ip addresses can be specified with cidr notation
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type CIDR -Value 10.100.1.1/24

# Individual targets can also be specified by hostname.
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type Name -Value "MyHostName"

# Add an IPv6 address
$AddressGroup.Addresses += New-RetinaAddressGroupAddress -Type Single -Value
"2001:0db8:85a3:0000:0000:8a2e:0370:7334"

# when finished configuring the address group, call Save() to write the file to disk.
$AddressGroup.Save()
```

The Address Group will be available to add to a scan.

Remove Address Groups

Address groups can be removed by passing the name of the Address Group to the `Remove-RetinaAddressGroup` cmdlet:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Delete an Address Group from Retina
-------------------------------------------------------------------------------
#>
# The name of the Address Group to remove
$AddressGroupName = "Test Address Group"

# Create the Address Group
$AddressGroup = New-RetinaAddressGroup -Name $AddressGroupName

# Save it
$AddressGroup.Save()

# Remove the Address Group
Remove-RetinaAddressGroup -Name $AddressGroupName
```

The Address Group will no longer be available to add to a scan.

Audit Groups

Create Audit Groups

Audit Groups can be created by passing the name of the new Audit Group to the `New-RetinaAuditGroup` cmdlet. Single and multiple audits can be added to the new Audit Group's **AuditList** property:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Create an Audit Group
-------------------------------------------------------------------------------
#>
# Create an audit group
$AuditGroupName = "My Audit Group"
$AuditGroup = New-RetinaAuditGroup -Name $AuditGroupName

# Add a single audit to the new audit group
$Audit = New-RetinaAuditGroupAudit -Type single -AuditId 44
$AuditGroup.AuditList += $Audit

# Add multiple single audits to the list of audits in the audit group
$Audit1 = New-RetinaAuditGroupAudit -Type single -AuditId 33
$Audit2 = New-RetinaAuditGroupAudit -Type single -AuditId 22
$Audit3 = New-RetinaAuditGroupAudit -Type single -AuditId 11
$AuditGroup.AuditList += $Audit1,$Audit2,$Audit3

# Add a range of audits to the Audit Group
$AuditGroup.AuditList += New-RetinaAuditGroupAudit -Type range -AuditId 100-150

# Save the new audit group to add to a scan later
$AuditGroup.Save()
```

The Audit Group will be available to add to a scan.

Remove Audit Groups

Audit groups can be removed by passing the name of the Audit Group to the `Remove-RetinaAuditGroup` cmdlet:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Delete an Audit Group from Retina
-------------------------------------------------------------------------------
#>
```

```
# The name of the Audit Group to remove
$AuditGroupName = "Test Audit Group"

# Create the Audit Group
$AuditGroup = New-RetinaAuditGroup -Name $AuditGroupName

# Save it
$AuditGroup.Save()

# Remove the Audit Group
Remove-RetinaAuditGroup -Name $AuditGroupName
```

The Audit Group will no longer be available to add to a scan.

Port Groups

Create Port Groups

Port Groups can be created by passing the name of the new Port Group to the `New-RetinaPortGroup` cmdlet. UDP and TCP ports as well as port ranges can be added to the new Port Group's **PortList** property:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Create a Port Group
-------------------------------------------------------------------------------
#>

# Create the port group
$Name = "My Port Group"
$PortGroup = New-RetinaPortGroup -Name $Name

# Add ports to the port group
# Add single a TCP/UDP port
$PortGroup.PortList += New-RetinaPortGroupPort -Protocol both -Type single -PortNumber 443

# Add a range of TCP ports to a port group
$PortGroup.PortList += New-RetinaPortGroupPort -Protocol tcp -Type range -PortNumber 1-1000

# Save the Port Group to add to scans later
$PortGroup.Save()
```

The Port Group will then be available to add to a scan.

Remove Port Groups

Port groups can be removed by passing the name of the Port Group to the `Remove-RetinaPortGroup` cmdlet:

```
<#
-------------------------------------------------------------------------------
Retina Network Security Scanner
Sample PowerShell Script

Delete a Port Group from Retina
-------------------------------------------------------------------------------
#>

# The name of the Port Group to remove
$PortGroupName = "Test Port Group"

# Create the Port Group
```

```
$PortGroup = New-RetinaPortGroup -Name $PortGroupName

# Save it
$PortGroup.Save()

# Remove the Port Group
Remove-RetinaPortGroup -Name $PortGroupName
```

The Port Group will no longer be available to add to a scan.

Manage Stored Credentials

Stored Credentials can be created by passing the name of the new Stored Credential to the `New-RetinaStoredCredential` cmdlet. Credential types `Any`, `Windows`, `SSH-PlainText`, `SSH-PublicKey`, `MySQL`, `MSSqlServer`, `Oracle`, `SNMP`, and `VMware` are available to be created. The `StoredCredentialType` determines the properties required to create the Stored Credential.

> **Note:** *You can view a list of the current stored credentials with the* `Get-RetinaStoredCredentials` *cmdlet.*

```
# View a list of stored credentials
Get-RetinaStoredCredentials
```

Any Credential

The `Any` credential type includes the following parameters:

- `Description`: A unique name for the credential.
- `Username` : The user name to use for authentication.
- `Password`: A SecureString password for authentication.
- `IsDefault`: Specifies if the credential will always be included in scans, regardless of selection state.

```
# Any
$User      = "MyUserName"
$SecurePWD = Read-Host -Prompt "Enter password" -AsSecureString
$AnyCred   = New-RetinaStoredCredential `
             -StoredCredentialType Any  `
             -Description "Any Sample"  `
             -UserName $User            `
             -SecurePassword $SecurePWD `
             -IsDefault true
```

Windows Credential

`Windows` credentials include the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `SecurePassword`: A SecureString password for authentication.

```
# Windows
$User        = "MyUserName"
$SecurePWD   = Read-Host -Prompt "Enter password" -AsSecureString
```

```
$WindowsCred = New-RetinaStoredCredential      `
               -StoredCredentialType Windows `
               -Description "WindowsCred"    `
               -UserName $User               `
               -SecurePassword $SecurePWD
```

SSH-PlainText Credential

`SSH-PlainText` credentials include the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `SecurePassword`: A SecureString password for authentication.
- `ElevationType`: Valid values for elevation type are: `None`, `Enable`, `sudo`, and `pbrun`.
- `ElevationUserName`: Optional user name for `sudo` and `pbrun` elevation types.
- `ElevationPassword`: Optional password for `Enable` elevation type.

```
# SSH-PlainText
$User             = "MyUserName"
$SecurePWD        = Read-Host -Prompt "Enter password" -AsSecureString
$SSHPlainTextCred = New-RetinaStoredCredential          `
                   -StoredCredentialType SSH-PlainText `
                       -Description "SSH-PlainText Sample" `
                       -UserName $User                     `
                       -SecurePassword $SecurePWD          `
                       -ElevationType None
```

SSH-PublicKey Credential

`SSH-PublicKey` credentials include the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `PrivateKeyPath`: A path to the private key to be used for authentication.
- `PassPhrase`: A SecureString pass phrase for authentication.
- `ElevationType`: Valid values for elevation type are: `None`, `Enable`, `sudo`, and `pbrun`.
- `ElevationUserName`: Optional user name for `sudo` and `pbrun` elevation types.
- `ElevationPassword`: Required password for `sudo` elevation type. Optional password for `Enable` elevation type.

To create an `SSH-PublicKey` credential with an `ElevationType` of `sudo`, use the following:

```
# SSH-PublicKey
```

```
$User             = "Administrator"
$PrivateKeyPath   = "~/.ssh/id_rsa"
$SecurePassphrase = Read-Host -Prompt "Enter passphrase" -AsSecureString

$SSHPublicKeyCred = New-RetinaStoredCredential          `
                    -StoredCredentialType SSH-PublicKey `
                    -Description "SSH-PublicKey"        `
                    -UserName $User                     `
                    -PrivateKeyPath $PrivateKeyPath     `
                    -PassPhrase $SecurePassphrase       `
                    -ElevationType sudo                 `
                    -ElevationPassword $SecurePassphrase
```

MySQL Credential

The `MySQL` credential type includes the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `SecurePassword`: A SecureString password for authentication.

```
# MySQL
$MySqlCred = New-RetinaStoredCredential  `
             -StoredCredentialType MySQL `
             -Description "MySQL Cred"   `
             -UserName $User             `
             -SecurePassword $SecurePWD
```

MSSqlServer Credential

The `MSSqlServer` credential type includes the following parameters:

- `Description`: A unique name for the credential
- `Username`: The user name to use for authentication
- `SecurePassword`: A SecureString password for authentication
- `AuthenticationType`: Valid values for authentication type are `Any`, `Windows`, or `Sql`.

```
# MSSqlServer
$MSSqlServerCred = New-RetinaStoredCredential          `
                   -StoredCredentialType MSSqlServer `
                   -Description "MSSqlServer Cred"   `
                   -UserName $User                   `
                   -SqlAuthenticationType Any        `
                   -SecurePassword $SecurePWD
```

Oracle Credential

The `Oracle` credential type includes the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `SecurePassword`: A SecureString password for authentication.
- `AccessLevel`: This parameter can be set to `Standard`, `SYSDBA`, or `SYSOPER`.
- `ConnectTo`: Valid values for `-ConnectTo` are `Database SID` or `Named Service`.
- `DatabaseSID`: Available when `-ConnectTo` is set `Database SID`. This parameter accepts a string representing an SID that uniquely identifies the database instance.
- `ServiceName`: Available when `-ConnectTo` is set `Named Service`. This parameter accepts a string representing the TNS alias representing the database instance.
- `Protocol`: Can be set to `TCP`, `TCPS`, or `NMP`.
- `Hosts`: Available when the `-Protocol` is set to `TCP` or `TCPS`. This parameter accepts a string array of hosts.
- `PortNumber`: Available when the `-Protocol` is set to `TCP` or `TCPS`. This parameter accepts an integer representing the target TCP port.
- `NMPServer`: Available when `-Protocol` is set to `NMP`. This parameter accepts a string representing the IP address of the server.
- `NMPPipe`: Available when `-Protocol` is set to `NMP`. This parameter accepts a string representing the name of the pipe.

```
# Oracle Samples
$TCPHosts       = @{"<TCP Host1>", "<TCP Host2>"}
$TCPPortNumber  = "<TCP Port>"
$NmpServer      = "<NMP Server>"
$NmpPipe        = "<Pipe Server>"
$TCPSHosts      = @{"<TCPS Host1>", "<TCPS Host2>"}
$TCPSPortNumber = "<TCPS Port>"

# Oracle Protocol: TCP
$cred = New-RetinaStoredCredential  `
       -StoredCredentialType Oracle `
       -Description "Oracle TCP"    `
       -UserName $User              `
       -SecurePassword $SecurePWD   `
       -AccessLevel Standard        `
       -ConnectTo DatabaseSID       `
       -DatabaseSID $myDatabaseSID  `
       -Protocol TCP                `
       -Host $TCPHost               `
       -PortNumber $TCPPortNumber

# Oracle Protocol: NMP
$cred = New-RetinaStoredCredential  `
       -StoredCredentialType Oracle `
       -Description "Oracle NMP"    `
       -UserName $User              `
       -SecurePassword $SecurePWD   `
```

```
        -AccessLevel Standard         `
        -ConnectTo DatabaseSID        `
        -DatabaseSID $myDatabaseSID   `
        -Protocol NMP                 `
        -NmpServer $NmpServer         `
        -NmpPipe $NmpPipe

# Oracle Protocol: TCPS
$cred = New-RetinaStoredCredential   `
        -StoredCredentialType Oracle `
        -Description "Oracle TCPS"   `
        -UserName $User              `
        -SecurePassword $SecurePWD   `
        -AccessLevel Standard        `
        -ConnectTo DatabaseSID       `
        -DatabaseSID $myDatabaseSID  `
        -Protocol TCPS               `
        -Host $TCPSHost              `
        -PortNumber $TCPSPortNumber
```

SNMP Credential

The `SNMP` credential type includes the following parameters:

- `Description`: A unique name for the credential.
- `Community String`: SNMP community string name used to authenticate.

```
# SNMP
$CommunityString = ""
$SnmpCred = New-RetinaStoredCredential `
            -StoredCredentialType SNMP `
            -Description "SNMP Cred"   `
            -CommunityString $CommunityString
```

VMware Credential

The `VMware` credential type includes the following parameters:

- `Description`: A unique name for the credential.
- `Username`: The user name to use for authentication.
- `SecurePassword`: A SecureString password for authentication.

```
# VMware
$VMWareCred = New-RetinaStoredCredential   `
              -StoredCredentialType VmWare `
```

```
                -Description "VMWare Cred"    `
                -UserName $User               `
                -SecurePassword $SecurePWD
```

Remove Stored Credentials

Stored Credentials can be removed by passing the name of the Stored Credential to the `Remove-RetinaStoredCredential` cmdlet:

```
# Remove-RetinaStoredCredential takes one command line argument: the Description of the credential
# This will remove the Windows credential created previously
Remove-RetinaStoredCredential -Description "Windows Cred"

# The description can also be retrieved from the credential created previously
# This will remove the SSH credential created previously
Remove-RetinaStoredCredential -Description $SSHCred.Description
```

The Stored Credential will no longer be available to add to a scan.