BeyondTrust

Privilege Management for Windows Core Scripting Guide

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

TC:12/7/2020

Table of Contents

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Introduction to Privilege Management Power Rules

A Power Rule lets you change the outcome of an application rule, based on the outcome of a PowerShell script.

Rather than a fixed Default rule that can be set to **Allow**, **Elevate**, **Audit**, or **Block** for the applications in the targeted application group, a Power Rule lets you determine your own outcome based on any scenario you can build into a Power Shell script.

Any existing Default rule in a Workstyle can be updated to a Power Rule by setting the **Run a Rule Script** option to a Power Rule script, and importing the PowerShell script you want to use. Privilege Management provides a PowerShell module with an interface to collect information about the user, application, and policy. The module can then send a resulting action back to the Privilege Management client to apply.

The Power Rules module also provides a variety of message options that allow you to collect additional information to support your PowerShell script logic and provide updates to the user as to the status, progress, or outcome of your rule. The supported messages include:

- Authentication message
- Business Justification message
- Information message
- Pass code message
- Vaulted credential message
- Asynchronous progress dialog for long running tasks

Power Rules is a highly flexible feature with unlimited potential. If you can do it in PowerShell, you can do it in a Power Rule. Here are some example use cases for Power Rules:

- **Environmental Factors**: Collect additional information about the application, user, computer, or network status to influence whether an application should be allowed to run, or run with elevated privileges.
- **Service Management**: Automatically submit tickets to IT Service Management solutions, and determine the outcome of a service ticket.
- **File Reputation**: Perform additional checks on an application by looking up the file hash in an application store, reputation service, or a vulnerability database.
- **Privileged Access Management**: Check out credentials from a password safe or vault, and pass them back to Privilege Management to run the application in that context.

Note: *Power Rules are best used for exception handling and with static policy.*

For a list of cmdlets available in the PRInterface PowerShell module, please see "PowerShell PRInterface Cmdlets" on page 14.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Get Started with Power Rules

This section takes you through creating an example Power Rule, adding it to an application rule and seeing it work in Privilege Management for Windows.

You need a Privilege Management for Windows 5.6 or later environment to run this script.

Example Script

This script will override a Default rule block for all applications where the **PG_PROG_PATH** variable does not contain **cmd.exe**.

Create the Power Rule Script

1. On the machine where you installed the Privilege Management Policy Editor, open Windows Notepad and paste the following code.

```
$ExecutingProgramPath = Get-PRVariable -Name "PG_PROG_PATH"
$ProgramNameToMatch = 'cmd.exe'
Show-PRMessageDialog -Title 'Rule Script Dialog' -LabelHeader "You just ran:
$ExecutingProgramPath. This script will block $ProgramNameToMatch" -ButtonOK 'OK'
if($ExecutingProgramPath.Contains($ProgramNameToMatch))
{
  Set-PRRuleProperty -Action 'Block'
}
  else
{
  Set-PRRuleProperty -Action 'Allow'
}
```

2. Save the file as **test-rulescript.ps1**, ensuring you specify the **ps1** extension.

You can use the Privilege Management Policy Editor or the Privilege Management ePO Extension to apply the rule script in policy.

Apply the Rule Script in Policy

Note: *This summary is intended for those who are familiar with editing policy in Privilege Management Policy Editor. If you need more information, please see the Administration Guide at https://www.beyondtrust.com/docs/privilege-management/windows/index.htm.*

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Privilege Management Policy Editor

These instructions apply to the Privilege Management Policy Editor.

In the Policy Editor:

1. Create a **Block Message** called **Test Power Rule Block Message**. This message will be displayed if the rule script doesn't run.
2. Create an **Application Group** called **Test Power Rule Applications** and add both **mspaint.exe** and **cmd.exe** as the **File or Folder Name** in the matching criteria.
3. Create a Workstyle called **Test Power Rules Applications** and add an application rule.

In the application rule:

1. Set the application group to **Test Power Rule Applications** from the **Target Application Group** dropdown list.
2. From the **Run a Rule Script** dropdown list, select **Manage Scripts**.
3. From the **Rule Scripts** node, click **Import Script**.
4. Navigate to **test-rulescript.ps1** and click **Open**.
5. Click **Close** on the **Script Manager** dialog box.
6. Set the **Default Action** to **Block Execution** and set the **Default End User Message** to **Test Power Rule Block Message** that you created earlier. The default action you choose should always be more restrictive than your Power Rule script.
7. Set **Raise an Event** to **On**.
8. Click **OK** to finish configuring the application rule.

You have now configured a Workstyle containing an application rule, which in turn targets an application group, containing both **cmd.exe** and **mspaint.exe** as matching criteria on the **File or Folder Name**. The application rule is configured to run the Power Rule, which blocks the application if the file or folder name contains **cmd.exe**; otherwise it allows it.

If the Power Rule does not run, the default action is **Block**, and the Privilege Management block message you configured will be displayed.

The next section includes the core integration script on the endpoint.

Privilege Management ePO Extension

These instructions apply to the Privilege Management ePO Extension.

In the Policy Editor:

1. Create a **Block Message** called **Test Power Rule Block Message**. This message is displayed if the rule script does not run.
2. Create an **Application Group** called **Test Power Rule Applications** and add both **mspaint.exe** and **cmd.exe** as the **File or Folder Name** in the matching criteria.
3. Create a Workstyle called **Test Power Rules Applications** and add an application rule.

In the application rule:

1. Set the **Application Group** to **Test Power Rule Applications** from the **Target Application Group** dropdown list.
2. From the **Run a Rule Script** dropdown list, select **test-rulescript.ps1**.
3. Set the **Default Action** to **Block Execution** and set the **Default End User Message** to the **Test Power Rule Block Message** that you created earlier. The default action you choose should always be more restrictive than the Power Rule script.
4. Set **Raise an Event** to **On**.
5. Click **OK** to finish configuring the application rule.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

You have now configured a Workstyle containing an application rule, which in turn targets an application group, containing both **cmd.exe** and **mspaint.exe** as matching criteria on the **File or Folder Name**. The application rule is configured to run the Power Rule, which blocks the application if the file or folder name contains **cmd.exe**; otherwise it will allow it.

If the Power Rule does not run, the default action is **Block**, and the Privilege Management block message you configured is displayed.

The next section takes you through testing the core integration script on the endpoint.

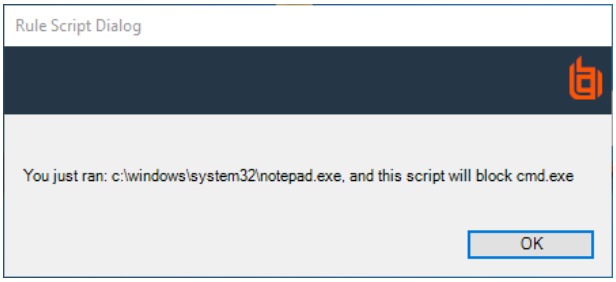
For more information, please see the following:

- For ePO Extension instructions, "Privilege Management ePO Extension" on page 5
- For Policy Editor instructions, "Privilege Management Policy Editor" on page 5

Run the Power Rule Script at the Endpoint

You can now use your endpoint Privilege Management environment to see the results of your Power Rule.

1. Ensure the policy is applied, and you have a valid license.
2. From the **Start** menu, type **cmd.exe** and press **Return**. The **Rule Script Dialog** is displayed.



3. When you click **OK**, **cmd.exe** does not run because the script stops it. The Default rule properties you set in the script editor are not used because you do not see the Privilege Management block message.
4. Run **mspaint.exe**. You will see the message from the script, but it will run successfully.

If you see the Privilege Management block message, ensure the Workstyle is enabled, has a valid license, and is configured correctly.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Script Examples for Privilege Management Power Rules

The scripting examples show you how to use some of the more common cmdlets available to you. Change the path in the examples to point to your instance of Power Rules.

Example:

Variables, Enable and Catch Logs, Audit Events, Request User Input, and Messages

```
#Enables logging to file
#Shows business justification dialog and outputs result to log file
#Utilizes Set-ScriptProperty to show name, version and output on event

#Import the PRInterface
Import-Module 'C:\PowerRules\Output\PRInterface\PRInterface.psd1'

#Import the PRTestHarness
Import-Module 'C:\PowerRules\Output\PRTestHarness\PRTestHarness.psd1'

#Instantiate PRTestHarness
$global:DefendpointAccessor = New-PRTestHarness -TestConfig
'C:\PowerRules\Output\PRTestHarness\AddAdmin_ExampleApp.json'

#Set the logging file and location
Set-PRLogSettings -LogToFile $true -LogFilePath "C:\Temp\examplescriptlog.log"

#Declare two variables for the prgram name an program path
$ProgramName = Get-PRVariable -Name "PG_PROG_NAME"
$ProgramPath = Get-PRVariable -Name "PG_PROG_PATH"

#Declare a new variable for the result of the business justification and dialog result
$businessJustificationDialogResult = Show-PRBusinessJustificationDialog -LabelHeader "Please
enter a business justification for why you need to run $ProgramName" -Title "Business
justification for launching application"

#If the user clicked 'OK', write the business justification they entered to the log file
If ($businessJustificationDialogResult.DialogResult -eq 'OK')
{
    Write-PRLog -Message ("Business Justification: {0}" -f
$businessJustificationDialogResult.BusinessJustification)
}

#If the user clicked 'Cancel', write the message and $ProgramName out to the log file
ElseIf ($businessJustificationDialogResult.DialogResult -eq 'Cancel')
{
    Write-PRLog -Message ("User chose to cancel the launch of $ProgramName")
}

#Sets the script properties, program name and program path to show on events
Set-PRScriptProperty -ScriptName "Example Power Rules Script" -ScriptVersion "1.0.0" -
ScriptOutput "User attempted to launch $ProgramName from $ProgramPath"
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Example:

Variables, Enable and Catch Logs, Chang Rule Behavior, and Messages

This example uses the message and token names in the QuickStart policy for Windows version 5.3. Please ensure you import this template into Privilege Management prior to running this script.

```
#Import the PRInterface
Import-Module 'C:\PowerRules\Output\PRInterface\PRInterface.psd1'

#Import the PRTestHarness
Import-Module 'C:\PowerRules\Output\PRTestHarness\PRTestHarness.psd1'

#Instantiate PRTestHarness
$global:DefendpointAccessor = New-PRTestHarness -TestConfig
'C:\PowerRules\Output\PRTestHarness\AddAdmin_ExampleApp.json'

#Sets the logging to file and to the console
Set-PRLogSettings -LogToFile $true -LogFilePath "C:\Temp\examplescriptlog.log"
Set-PRLogSettings -LogToConsole $true

#Declare a new variable for the program path
$ExecutingProgramPath = Get-PRVariable -Name "PG_PROG_PATH"

#Declare a new variable for the string 'cmd.exe'
$ProgramNameToMatch = 'cmd.exe'

#Display a message to the user
Show-PRMessageDialog -Title 'Rule Script Dialog' -LabelHeader "You just ran:
$ExecutingProgramPath. This script will block $ProgramNameToMatch" -ButtonOK 'OK'

#Check to see if the variable $ExecutingProgramPath contains 'cmd.exe'
if($ExecutingProgramPath.Contains($ProgramNameToMatch))
{
    #Set the action to block and the message to the Block Message
    Set-PRRuleProperty -Action 'Block' -Message 'Block Message'
    Write-PRLog -Message 'This application was blocked'
}
else
{
    #Set the action to allow, the message to the Allow Message (Yes / No) and the Token
    #to the Avecto Support Token
    Set-PRRuleProperty -Action 'Allow' -Message 'Allow Message (Yes / No)' `
    -Token 'Custom' -TokenName 'Avecto Support Token'
    Write-PRLog -Message 'This application was allowed to run'
}
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Example:

Use the PRInterface and PRTestHarness Modules as well as a Settings file to get the Challenge Code

```
#Import the PRInterface
Import-Module 'C:\PowerRules\Output\PRInterface\PRInterface.psd1'

#Import the PRTestHarness
Import-Module 'C:\PowerRules\Output\PRTestHarness\PRTestHarness.psd1'

#Instantiate PRTestHarness
$global:DefendpointAccessor = New-PRTestHarness -TestConfig
'C:\PowerRules\Output\PRTestHarness\AddAdmin_ExampleApp.json'

Get-PRChallengeCode
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Example:

Use the Set-PRRunAsProperty and a Settings file

> ***Note:*** *Ensure the Settings file is present in the location you specify in the* ***-TestSettings*** *parameter for DefendpointAccessor.*

Settings File

```
{
   "Account": {
      "UserName": "Stan",
      "Password": "Stan"
    }
}
```

```
#Import the PRInterface
Import-Module 'C:\PowerRules\Output\PRInterface\PRInterface.psd1'

#Import the PRTestHarness
Import-Module 'C:\PowerRules\Output\PRTestHarness\PRTestHarness.psd1'

#Instantiate PRTestHarness
$global:DefendpointAccessor = New-PRTestHarness -TestConfig
'C:\PowerRules\Output\PRTestHarness\AddAdmin_ExampleApp.json' -TestSettings
"C:\PowerRules\Settings.json"

# Get Account details from settings file (which is encrypted on endpoint)
$Settings = Get-PRScriptSettings
$AccountName = $Settings.Account.UserName
$Password = $Settings.Account.Password

# Set script properties to appear in audit eventsSet-PRScriptProperty -ScriptName "Run As
Demo" -ScriptVersion "1.0.0" -ScriptOutput "Running as $AccountName"

#Set RunAs account properties
Set-PRRunAsProperty -Username "$AccountName" -Password "$Password"

#Set Rule Properties to run
Set-PRRuleProperty -Action "Allow" -Token "Passive"
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Set up a Script Environment to Access PowerShell

To create a Power Rules script outside the Privilege Management for Windows environment, you must set up a scripting environment. This allows you to access the PowerShell cmdlets you can use to write your script and test it in a mock Privilege Management for Windows environment.

The first steps are to import the following modules:

- **PRInterface module**: Contains the cmdlets you can use for your script. It is installed with Privilege Management for Windows.
- **PRTestHarness module**: Allows you to mimic the behavior of Privilege Management for Windows so the rule script behaves as if Privilege Management for Windows was installed.

You can then create an instance of the **TestHarness** module and provide test data if required.

1. Import the **PRInterface** module. You can import the module to a different environment if you need to work on a script in a separate environment.
2. Import the **PRTestHarness** module.
3. Create an instance of the **PRTestHarness** module. This allows you to work with, and if you choose, stub the **PRTestHarness** module with test data.

For more information, please see the following:

- "Import the PRInterface PowerShell Module" on page 11
- "Import the PRTestHarness PowerShell Module" on page 12
- "Stub Privilege Management for Windows with PRTestHarness" on page 12

Import the PRInterface PowerShell Module

PRInterface is the name of a PowerShell module that provides cmdlets for the Power Rules script to use to pass information to and from Privilege Management for Windows. Using it you can:

- Manipulate the behavior of a rule
- Provide information about a Privilege Management for Windows event
- Show customizable dialog boxes that display information to, and gather information from, an end user
- Run an application as another user
- Add detail to audit events

You can use the PowerShell cmdlet **get-help <cmdletname>** to view the full help documentation, including examples, in PowerShell.

Note: *You must change the paths to point to your* ***PRInterface*** *folder.*

You can import the **PRInterface** modules, so your script mimics the behavior of the Privilege Management for Windows PowerShell execution environment. The **AddAdmin_ExamplesApp.json** file in the **PRTestHarness** folder contains example data you can use to get started.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

For a list of variables, please see the Privilege Management for Windows Administration Guide at https://www.beyondtrust.com/docs/privilege-management/windows/index.htm.

To import the **PRInterface** module:

```
Import-Module 'C:\PowerRules\PRInterface\PRInterface.psd1'
```

Import the PRTestHarness PowerShell Module

PRTestHarness is a PowerShell module that provides a mechanism for testing Power Rules scripts outside the Privilege Management for Windows execution environment. This allows you to simulate a Privilege Management event and run a script in PowerShell ISE to see the resulting behavior. This means you can speed up script development by testing and debugging outside the Privilege Management for Windows Power Rules environment.

The test harness mimics the behavior of Privilege Management for Windows. This process is known as *stubbing* and it allows you to write scripts using the **PRInterface** module in other PowerShell environments.

***Note:** You must change the paths to point to your **PRTestHarness** folder.*

To import the **PRTestHarness** module:

```
Import-Module 'C:\PowerRules\PRTestHarness\PRTestHarness.psd1'
```

Stub Privilege Management for Windows with PRTestHarness

Each time you want to edit your integration script in a new instance of Windows PowerShell ISE, you need to create a **Defendpoint Accessor**, and optionally, provide example data to it. This example references the **AddAdmin_ExampleApp.json** file, which contains example data.

Stub the Privilege Management for Windows interface with some test data:

Example:

```
$global:DefendpointAccessor = New-PRTestHarness -TestConfig
'C:\PowerRules\PRTestHarness\AddAdmin_ExampleApp.json'
```

Test the Configuration

You can provide the following arguments to the PRTestHarness cmdlet:

```
[-TestConfig <string>] [-TestSettings <string>] [<CommonParameters>]
```

Test the data is coming through from the **AddAdmin_ExampleApp.json** file:

```
Get-PRChallengeCode
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

-TestSettings allows you to also specify a path with file name to a **Settings** file. Settings files are encrypted on the endpoint so any credentials or sensitive information for your integration are protected.

For more information, please see "Use a Settings File with a Rule Script File" on page 15.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

PowerShell PRInterface Cmdlets

The following cmdlets are available from the **PRInterface** module. You can run the **get-help** command in PowerShell to obtain detailed information on each cmdlet, including the syntax and description:

```
get-help <cmdletname>
```

Example:

```
get-help Get-PRVariable
```

Name	Description
ConvertTo-PRHashTable	Converts a JSON object created using **ConvertFrom-Json** into a hash table.
Get-PRChallengeCode	Displays the challenge code valid for the currently running application.
Get-PREnvironmentVariable	Displays an environment variable valid for the currently running application.
Get-PRFileHash	Displays the file hash information for the currently running application.
Get-PRScriptSettings	Displays the settings information for the currently running script.
Get-PRVariable	Displays a Privilege Management for Windows variable valid for the currently running application.
Set-PRLogSettings	Sets the defaults for Power Rules script logging.
Set-PRRuleProperty	Sets the Rule Properties to action when the script finishes running.
Set-PRRunAsProperty	Sets the Run As Properties to action when the script finishes running.
Set-PRScriptProperty	Sets the Script Properties which are recorded in audit events. Runs the application under another user's credentials.
Show-PRAuthenticationDialog	Displays an authentication message dialog box.
Show-PRBusinessJustificationDialog	Displays a business justification message dialog box.
Show-PRMessageDialog	Displays an informational message dialog box.
Show-PRPasscodeDialog	Displays a passcode message dialog box.
Show-PRProgressDialog	Displays an asynchronous progress dialog box for long running tasks.
Show-PRVaultedCredentialDialog	Displays a vaulted credential message dialog box.
Test-PRAgentVersion	Tests to see if the Privilege Management Agent meets the minimum version required.
Test-PRModuleVersion	Tests to see if the Privilege Management for Windows Power Rules module (**PRInterface**) meets the minimum version required.
Test-PRNetworkConnectivity	Tests to see if a network connection can be established to a specific host name URL.
Write-PRLog	Outputs script logging information to a number of sources.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Use a Settings File with a Rule Script File

A **Settings (*.json)** file is an optional file you can use in your integration. Settings files are encrypted at the endpoint so they are useful for storing credentials and other sensitive information.

After you associate a Settings file with a rule script (*.ps1), that Settings file is used wherever the rule script is used. If you change the Settings file, the updated Settings file is applied to all rules referencing the rule script.

In Privilege Management for Windows, the two files are intrinsically linked after you associate a Settings file with a rule script.

Test a Settings File

Instantiate the **PRTestHarness** with a Settings file using **DefendpointAccessor**.

```
$Global:defendpointAccessor = New-PRTestHarness -TestConfig "C:\PowerRules\PRTestHarness\AddAdmin_
ExampleApp.json" -TestSettings "\PowerRules\Settings.json"
```


©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Additional Guidance on Using PowerShell

You can use the PowerShell **get-help** command to get help on any cmdlet in PowerShell. You can also use the following arguments for additional guidance on the cmdlet:

- **-examples**
- **-detailed**
- **-full**
- **-online**

Compatibility

Power Rules requires PowerShell 3.0 or later. Run the following command to check the version of PowerShell you are running:

```
$PSVersionTable.PSVersion
```

If you attempt to edit an application rule containing a Power Rule in a Privilege Management Policy Editor older than 5.3.x, the **PowerRuleScript** attribute (that is linked to the Power Rule) is removed from the application rule.

For more information about compatibility with other Privilege Management for Windows versions, please see the Release Notes for each version, at www.beyondtrust.com/support/changelog.

Third Party Integration Security

When you integrate with a third party, you should ensure you use the most secure mechanism possible. For example, if a vendor offers both HTTP and HTTPS, use HTTPS.

Supported Application Types

All Application Types are supported, with the exception of:

- Remote PowerShell Script
- Remote PowerShell Command
- Windows Service
- Windows Store Application

If you use these application types with a Power Rule, the rule script will not run and the event will state, *Script execution skipped: Application Type not supported*. This is an 801 event.

Validation

Some restrictions are enforced by the Privilege Management Policy Editor but cannot be enforced in a scripting environment. The following is guidance for creating your Power Rule. If Privilege Management cannot determine the correct course of action, the Default rule is applied.

All messages and tokens must exist in your policy configuration prior to referencing them in a Power Rule script.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

- The **Action** must match the **Message**. For example, if the Action is **Allow**, the message must be of type **Allow**.
- If you set the Action to **Allow**, we assume a passive **Token** but you can add a different token such as a **Custom Token** you created.
- Tokens cannot be used when the action is **Block**.
- If you specify an *account to run as*, your action must be **Allow**.

If the script fails, a local audit event 801 will be triggered.

If you use **Set-PRRunAsProperty**, then you need to use **Set-PRRuleProperty** and set the **-Action** argument to **Allow**. You can optionally set the **-Token** argument. If you do not define a token, then a **Passive** token is applied.

The values for **-Action** and **-Token** are case sensitive.

For more information, please see "Script Audit Failure Event" on page 17.

Script Restrictions

There are some scripting restrictions you need to be aware of when you are creating your own integrations.

Block Comments

Single line comments are supported, but block comments are not. Block comments take the form:

```
<# block comment #>
```

PowerShell single line comments are supported.

```
# comment
```

#Requires

The **#Requires** notation is not supported.

Script Audit Failure Event

If a Rule Script fails, then a local Windows event is created, and the Privilege Management for Windows event number is 801. This event is always created even when auditing is turned off. The following fields are shown in the event:

Variable Name	Description
RuleScriptFileName	Name attribute of the script in the config
RuleScriptName	Set by script properties
RuleScriptVersion	Set by script properties
RuleScriptPublisher	The publisher of the script
RuleScriptRuleAffected	Whether a rule script changed a Privilege Management for Windows rule
RuleScriptStatus	Timeout, Exception
RuleScriptResult	Script timeout exceeded: X seconds, Set Rule Properties failed validation

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Variable Name	Description
ExceptionType	Any valid .NET exception type
ExceptionMessage	The short exception message
ProcessId	PID of the process matching the rule
ParentProcessId	PID of the parent process matching the rule
ProcessStartTime	Time the process started
Event Time	Time the script started
UniqueProcessId	GUID of process to link this data to associated audit process event

PowerShell Scripting Execution Policy

We recommend using one PowerShell script for each integration you create. If you create a Power Rule script that in turn calls an additional PowerShell script, you need to distribute that PowerShell script independently and may need to change your PowerShell execution policy to ensure it can run.

Encodings

If you want to maintain signed scripts, you must ensure they are encoded in UTF-16 LE prior to importing them into Privilege Management for Windows. Rule Script files exported from the Privilege Management Policy Editor are always encoded in UTF-16 LE.

Settings files are encrypted at the endpoint. Settings files must be encoded in UTF-8.

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Use Windows Parameters with Power Rules

The Privilege Management for Windows settings include a number of features allowing customization of text and strings used for end user messaging and auditing. If you want to include properties relating to the settings applied, the application being used, the user, or the installation of Privilege Management for Windows, then parameters may be used which are replaced with the value of the variable at runtime.

Parameters are identified as any string surrounded by brackets (**[ ]**), and if detected, the agent attempts to expand the parameter. If successful, the parameter is replaced with the expanded property. If unsuccessful, the parameter remains part of the string. The table below shows a summary of all available parameters and where they are supported.

Parameter	Description
[PG_AGENT_VERSION]	The version of Privilege Management for Windows
[PG_APP_DEF]	The name of the application rule that matched the application
[PG_APP_GROUP]	The name of the application group that contained a matching application rule
[PG_COM_APPID]	The APPID of the COM component being run
[PG_COM_CLSID]	The CLSID of the COM component being run
[PG_COM_NAME]	The name of the COM component being run
[PG_COMPUTER_DOMAIN]	The name of the domain that the host computer is a member of
[PG_COMPUTER_NAME]	The NetBIOS name of the host computer
[PG_DOWNLOAD_URL]	The full URL from which an application was downloaded
[PG_DOWNLOAD_URL_DOMAIN]	The domain from which an application was downloaded
[PG_EVENT_TIME]	The date and time that the policy matched
[PG_EXEC_TYPE]	The type of execution method: application rule or shell rule
[PG_GPO_DISPLAY_NAME]	The display name of the GPO (Group Policy Object)
[PG_GPO_NAME]	The name of the GPO that contained the matching policy
[PG_GPO_VERSION]	The version number of the GPO that contained the matching policy
[PG_MESSAGE_NAME]	The name of the custom message that was applied
[PG_POLICY_NAME]	The name of the policy
[PG_PROG_CLASSID]	The ClassID of the ActiveX control
[PG_PROG_CMD_LINE]	The command line of the application being run
[PG_PROG_DRIVE_TYPE]	The type of drive where application is being executed
[PG_PROG_FILE_VERSION]	The file version of the application being run
[PG_PROG_HASH]	The SHA-1 hash of the application being run
[PG_PROG_HASH_MD5]	The MD5 hash of the application being run
[PG_PROG_NAME]	The program name of the application
[PG_PROG_PARENT_NAME]	The file name of the parent application
[PG_PROG_PARENT_PID]	The process identifier of the parent of the application
[PG_PROG_PATH]	The full path of the application file
[PG_PROG_PID]	The process identifier of the application

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.

Parameter	Description
[PG_PROG_PROD_VERSION]	The product version of the application being run
[PG_PROG_PUBLISHER]	The publisher of the application
[PG_PROG_TYPE]	The type of application being run
[PG_PROG_URL]	The URL of the ActiveX control
[PG_STORE_PACKAGE_NAME]	The package name of the Windows Store App
[PG_STORE_PUBLISHER]	The package publisher of the Windows Store app
[PG_STORE_VERSION]	The package version of the Windows Store app
[PG_TOKEN_NAME]	The name of the built-in token or custom token that was applied
[PG_USER_DISPLAY_NAME]	The display name of the user
[PG_USER_DOMAIN]	The name of the domain that the user is a member of
[PG_USER_NAME]	The account name of the user
[PG_WORKSTYLE_NAME]	The name of the Workstyle

©2003-2020 BeyondTrust Corporation. All Rights Reserved. Other trademarks identified on this page are owned by their respective owners. BeyondTrust is not a chartered bank or trust company, or depository institution. It is not authorized to accept deposits or trust accounts and is not licensed or regulated by any state or federal banking authority.