BeyondTrust

Privilege Management
Core Scripting Guide 5.3.217.0 SR1

TC:2/6/2019

Table of Contents

Power Rules **4**
Get Started with Power Rules **5**
Example Script 5
Creating the Power Rule Script 5
Applying the Rule Script in Policy 5
Running the Power Rule Script at the Endpoint 7
Script Examples 8
Example One - Variables, Enabling and Catching Logs, Auditing Events, Requesting User Input, and Messages 8
Example Two - Variables, Enabling and Catching Logs, Changing Rule Behavior, and Messages 9
Example Three - Enabling and Catching Logs, Try-Catch Blocks, Set-PRScript Property -CancelOutcome, Displaying a Progress Dialog, and Messages 10
Example Four - Using the PRInterface and PRTestHarness Modules as well as a Settings file to get the Challenge Code 10
Example Five - Using the Set-PRRunAsProperty and a Settings file 11
Script Environment **12**
Import the PRInterface Module 12
Import the PRTestHarness Module 12
Stub Defendpoint with PRTestHarness 13
Test the Configuration 13
PRInterface Cmdlets **14**
Use a Settings File **16**
Test a Settings File 16
Additional Guidance **17**
Compatibility 17
Third Party Integration Security 17
Supported Application Types 17
Validation 17
Script Restrictions 18
Block Comments 18
#Requires 18

Script Audit Failure Event 18
PowerShell Scripting Execution Policy 19
Encodings 19
Windows Parameters 20

Power Rules

A Power Rule is a PowerShell based framework that lets you change the outcome of an Application Rule, based on the outcome of a PowerShell script.

So, instead of a fixed Default Rule that can either 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 within a Workstyle can be updated to a Power Rule simply by setting the action to a Power Rule script, and importing the PowerShell script you want to use. Defendpoint 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 Defendpoint client to apply. For a full list of cmdlets that are available in the `PRInterface` PowerShell module, see "PRInterface Cmdlets" on page 14.

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 messages that are supported include:

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

Full details of the cmdlets can be found in the "PRInterface Cmdlets" on page 14 section.

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 – Collecting 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 submitting tickets to IT Service Management solutions, and determining the outcome of a service ticket.
- File Reputation – performing additional checks on an application by looking up the file hash in an application store, reputation service, or a vulnerability database.
- Privileged Access Management – Checking out credentials from a password safe or vault, and passing them back to Defendpoint to run the application in that context.

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

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 Defendpoint.

You need a Defendpoint 5.3 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`.

Creating the Power Rule Script

1. On the machine where you have installed the Defendpoint Policy Editor, open Windows Notepad and paste the following code in.

```
$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-defendpoint-rulescript.ps1`, ensuring you specify the `ps1` extension.

You can use the Defendpoint Management Console or the Defendpoint ePO Extension to apply the rule script in policy.

Applying the Rule Script in Policy

> ***Note:*** *See the Administration Guide for your policy editor for details on any of these steps if required. This summary is intended for those who are familiar with editing policy in Defendpoint.*

Defendpoint Management Console

These instructions apply to the Defendpoint Management Console. Please see the next section for instructions for the Defendpoint ePO Extension.

In your policy editor:

1. In your policy editor:
    a. Create a **Block Message** called `Test Power Rule Block Message`. This message will be displayed if the Rule Script doesn't run.
    b. Create an **Application Group** called `Test Power Rule Applications` and add both `mspaint.exe` and `cmd.exe` as the `File/Folder Name` in the matching criteria.
    c. Create a Workstyle called `Test Power Rules Applications` and add an Application Rule.
2. In the Application Rule:
    a. Set the Application Group to `Test Power Rule Applications` from the **Target Application Group** drop-down.
    b. From the **Run a Rule Script** drop-down, select **Manage Scripts**.
    c. From the **Rule Scripts** node, click **Import Script**.
    d. Navigate to your `test-defendpoint-rulescript.ps1` and click **Open**.
    e. Click **Close** on the **Script Manager** dialog.
    f. 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 that you choose should always be more restrictive than your Power Rule script.
    g. Set **Raise an Event** to `On` and click **OK** to finish configuring your Application Rule.

You have now configured a Workstyle that contains an Application Rule, which in turn targets an Application Group, that contains both `cmd.exe` and `mspaint.exe` as matching criteria on the **File/Folder Name**. The Application Rule is configured to run your Power Rule which will block the application if the `File/Folder Name` contains `cmd.exe`, otherwise it will allow it.

If the Power Rule doesn't run, the default action is **Block**, and the Defendpoint Block message that you configured will be displayed.

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

Defendpoint ePO Extension

These instructions apply to the Defendpoint ePO Extension. Please see the previous section for instructions for the Defendpoint Management Console.

In your policy editor:

a. Create a **Block Message** called `Test Power Rule Block Message`. This message will be displayed if the Rule Script doesn't run.
b. Create an **Application Group** called `Test Power Rule Applications` and add both `mspaint.exe` and `cmd.exe` as the `File/Folder Name` in the matching criteria.
c. Create a Workstyle called `Test Power Rules Applications` and add an Application Rule.

In the Application Rule:

a. Set the Application Group to `Test Power Rule Applications` from the **Target Application Group** drop-down.
b. From the **Run a Rule Script** drop-down, select `test-defendpoint-rulescript.ps1`.
c. 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 that you choose should always be more restrictive than your Power Rule script.
d. Set **Raise an Event** to `On` and click **OK** to finish configuring your Application Rule.

You have now configured a Workstyle that contains an Application Rule, which in turn targets an Application Group, that contains both `cmd.exe` and `mspaint.exe` as matching criteria on the **File/Folder Name**. The Application Rule is configured to run your Power Rule which will block the application if the `File/Folder Name` contains `cmd.exe`, otherwise it will allow it.

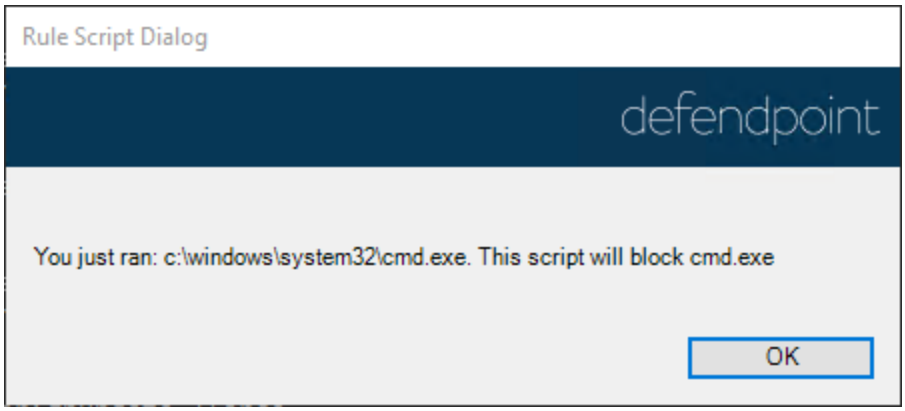
If the Power Rule doesn't run, the default action is **Block**, and the Defendpoint Block message that you configured will be displayed.

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

Running the Power Rule Script at the Endpoint

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

1. Ensure the policy has been applied, and that you have a valid license. From the **Start** menu, type in `cmd.exe` and press **Return**. You should see the following dialog:



2. When you click **OK** `cmd.exe` will not be run because the script stopped it. The default rule properties that you set in the Script Editor are not used because you do not see the Defendpoint Block message.
3. Now run `mspaint.exe`. You will see the message from the script, but it will run successfully.

If you see your Defendpoint Block message, ensure that your Workstyle is enabled, has a valid license and has been configured correctly.

Script Examples

These scripting examples show you how to use some of the more common cmdlets available to you. You will need to change the path in the examples to point to your instance of PowerRules.

- "Example One - Variables, Enabling and Catching Logs, Auditing Events, Requesting User Input, and Messages" on page 8
- "Example Two - Variables, Enabling and Catching Logs, Changing Rule Behavior, and Messages" on page 9
- "Example Three - Enabling and Catching Logs, Try-Catch Blocks, Set-PRScript Property -CancelOutcome, Displaying a Progress Dialog, and Messages" on page 10
- "Example Four - Using the PRInterface and PRTestHarness Modules as well as a Settings file to get the Challenge Code " on page 10
- "Example Five - Using the Set-PRRunAsProperty and a Settings file" on page 11

Example One - Variables, Enabling and Catching Logs, Auditing Events, Requesting 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"
```

Example Two - Variables, Enabling and Catching Logs, Changing 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 script into Defendpoint 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'
}
```

Example Three - Enabling and Catching Logs, Try-Catch Blocks, Set-PRScript Property - CancelOutcome, Displaying a Progress Dialog, and Messages

```
#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-PRLogSettings -LogToFile $true -LogFilePath "C:\Temp\examplescriptlog.log"
Set-PRLogSettings -LogToConsole $true

$script:progressDialogHandler = Show-PRProgressDialog
start-sleep -seconds 5
if ($script:progressDialogHandler.UserHasCanceled)
{
    Set-PRScriptProperty -CancelOutcome
    Exit
}
try
{
    #Interact with your 3rd party supplier
}
catch
{
    Write-PRLog -Message "Error communicating with 3rd party"
    throw "Error communicating with 3rd party"
}
if ($script:progressDialogHandler.UserHasCanceled)
{
    Set-PRScriptProperty -CancelOutcome
}
$script:progressDialogHandler.Close()
```

Example Four - Using 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
```

Example Five - Using the Set-PRRunAsProperty and a Settings file

Note: *Ensure the Settings file is present in the location you specify in the -TestSettings paramter of the Defendpoint Accessor.*

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"
```

Script Environment

In order to create your own Power Rules script outside of the Defendpoint environment, you need to 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 Defendpoint environment.

The first steps are to import the `PRInterface` module and `PRTestHarness` module. You can then create an instance of the `TestHarness` and provide it with test data if required.

1. Import the `PRInterface` module. It contains the cmdlets you can use for your script. It is installed with the Defendpoint Client but you can import it to a different environment if you need to work on a script in a separate environment. See "Import the PRInterface Module" on page 12 for more information.
2. Import the `PRTestHarness` module. It allows you to mimic the behavior of Defendpoint so the Rule Script behaves as it would if Defendpoint was installed. See "Import the PRTestHarness Module" on page 12 for more information.
3. Create an instance of the `PRTestHarness` module . This allows you to work with, and if you choose, stub the `PRTestHarness` with test data. See "Stub Defendpoint with PRTestHarness" on page 13 for more information.

Import the PRInterface 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 Defendpoint.

- Manipulate the behavior of a rule
- Provide information about a Defendpoint event
- Show customizable dialogs 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 will need to alter the paths to point to your **PRInterface** folder.*

You can import the `PRInterface` modules, so your script mimics the behavior of the Defendpoint PowerShell execution environment. The `AddAdmin_ExamplesApp.json` file in the **PRTestHarness** folder contains example data you can use to get started. A full list of variables can be found in the Defendpoint Administration Guide.

Import the `PRInterface` module:

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

Import the PRTestHarness Module

`PRTestHarness` is a PowerShell module that provides a mechanism of testing Power Rules scripts outside of the Defendpoint execution environment. This allows you to simulate a Defendpoint 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 of the Defendpoint Power Rules environment.

> ***Note:*** *You will need to alter the paths to point to your* ***PRTestHarness*** *folder.*

> ***Note:*** *This test harness mimics the behavior of Defendpoint. This is known as stubbing. It allows you to write scripts that use the* `PRInterface` *module in other PowerShell environments.*

To import the `PRTestHarness`:

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

Stub Defendpoint 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, you can provide example data to it. This example references the `AddAdmin_ExampleApp.json` file which contains example data.

> ***Note:*** *Stub the Defendpoint interface with some test data.*

```
$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
```

`-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. See "Use a Settings File" on page 16.

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>
```

For example:

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

Name	Description
`ConvertTo-PRHashTable`	Converts a JSON object created using ConvertFrom-Json into a Hashtable.
`Get-PRChallengeCode`	Displays the Defendpoint Challenge Code valid for the currently executing application.
`Get-PREnvironmentVariable`	Displays an environment variable valid for the currently executing application.
`Get-PRFileHash`	Displays the file hash information for the currently executing application.
`Get-PRScriptSettings`	Displays the settings information for the currently executing script.
`Get-PRVariable`	Displays a Defendpoint variable valid for the currently executing application.
`Set-PRLogSettings`	Sets the defaults for Power Rules script logging.
`Set-PRRuleProperty`	Sets the Defendpoint Rule Properties to action when the script finishes executing.
`Set-PRRunAsProperty`	Sets the Defendpoint Run As Properties to action when the script finishes executing.
`Set-PRScriptProperty`	Sets the Defendpoint Script Properties which are recorded in audit events. Executes the application under another user's credentials
`Show-PRAuthenticationDialog`	Displays an authentication message dialog.
`Show-PRBusinessJustificationDialog`	Displays a business justification message dialog.
`Show-PRMessageDialog`	Displays an informational message dialog.
`Show-PRPasscodeDialog`	Displays a passcode message dialog.

Name	Description
`Show-PRProgressDialog`	Displays an asynchronous progress dialog for long running tasks.
`Show-PRVaultedCredentialDialog`	Displays a vaulted credential message dialog.
`Test-PRAgentVersion`	Tests to see if the Defendpoint Agent meets the minimum version required.
`Test-PRModuleVersion`	Tests to see if the Defendpoint Power Rules module (`PRInterface`) meets the minimum version required.
`Test-PRNetworkConnectivity`	Tests to see if a network connection can be established to a specific hostname URL.
`Write-PRLog`	Outputs script logging information to a number of sources.

Use a Settings File

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

Once you have associated a Settings (*.json) file with a Rule Script (*.ps1) that Settings file will be used wherever the Rule Script is used. If you make any changes to the Settings file, the updated Settings file will be applied to all rules that reference the Rule Script. In Defendpoint the two files are intrinsically linked once you have associated a Settings file with a Rule Script.

Test a Settings File

***Note:** Instantiate the `PRTestHarness` with a Settings file using `defendpointAccessor`.*

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

Additional Guidance

You can use the PowerShell `get-help` command to get help on any cmdlet in PowerShell. You can also the following arguments to get additional guidance on the cmdlet `-examples`, `-detailed`, `-full`, `-online`.

Compatibility

Power Rules requires PowerShell 3.0 or higher. You can check the version of PowerShell you're running with the following command:

```
$PSVersionTable.PSVersion
```

If you attempt to edit an Application Rule that contains a Power Rule in a Defendpoint Management console older than 5.3.x, the "PowerRuleScript" attribute (that is linked to the Power Rule), is removed from the application rule.

See the Release Notes for each version for detailed compatibility with other Defendpoint clients.

Third Party Integration Security

When you are integrating with a third party, you should ensure that you use the most secure mechanism possible. For example, if a vendor offers both `http` and `https`, you should 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 try and use these Application Types with a Power Rule, the Rule Script will not be executed and the event will state `Script execution skipped: Application Type not supported`. This is an 801 event.

Validation

Some restrictions are enforced by the Defendpoint policy editor but cannot be enforced in a scripting environment. The following is guidance for creating your Power Rule. If Defendpoint 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.

- 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 be **Allow**, we assume a passive **Token** but you can add a different token such a **Custom Token** that you have 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, see Script Audit Failure Event for more information.

If you use `Set-PRRunAsProperty` you need to use `Set-PRRuleProperty` and set the `-Action` argument to **Allow**. You can optionally set the `-Token` argument. If you don't define a Token then a **Passive** Token is applied.

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

Script Restrictions

There are some scripting restrictions that 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, the Defendpoint 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 Defendpoint rule.
RuleScriptStatus	Timeout, Exception
RuleScriptResult	Script timeout exceeded: X seconds, Set Rule Properties failed validation
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 that you create. If you create a Power Rule script that in turn calls an additional PowerShell script, you will 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 need to ensure they are encoded in UTF-16 LE prior to importing them into Defendpoint. Rule Script files that are exported from Defendpoint are always encoded UTF-16 LE.

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

Windows Parameters

The Defendpoint Settings include a number of features that allow customization of text and strings that are used for end user messaging and auditing. If you want to include properties that relate to the settings applied, the application being used, the user or the installation of the Defendpoint Client, then parameters may be used that expand when the text is used.

Parameters are identified as any string surrounded by [square parentheses], and if detected, the agent will attempt to expand the parameter. If successful, the parameter will be replaced with the expanded property. If unsuccessful, the parameter will remain part of the string. The table below shows a summary of all available parameters and where they are supported.

Parameter	Description
[PG_ACTION]	The action which the user performed from an end user message
[PG_AGENT_VERSION]	The version of the Defendpoint Client
[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_AUTH_USER_DOMAIN]	The domain of the designated user who authorized the application
[PG_AUTH_USER_NAME]	The account name of the designated user who authorized the application
[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_CONTENT_DEF]	The definition name of the matching content
[PG_CONTENT_FILE_DRIVE_TYPE]	The drive type of a matching content
[PG_CONTENT_FILE_HASH]	The SHA-1 hash of a matching content
[PG_CONTENT_FILE_IE_ZONE]	The Internet Zone of a matching content
[PG_CONTENT_FILE_NAME]	The file name of a matching content
[PG_CONTENT_FILE_OWNER]	The owner of a matching content
[PG_CONTENT_FILE_PATH]	The full path of a matching content
[PG_CONTENT_GROUP]	The group name of a matching content definition
[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 / 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_MSG_CHALLENGE]	The 8 digit challenge code presented to the user

Parameter	Description
[PG_MSG_RESPONSE]	The 8 digit response code entered by the user
[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_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
[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_SERVICE_ACTION]	The action performed on the matching service
[PG_SERVICE_DISPLAY_NAME]	The display name of the Windows service
[PG_SERVICE_NAME]	The name of the Windows service
[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_URL_ADDRESS]	The full address of the matching URL
[PG_URL_DEF]	The definition name of the matching URL
[PG_URL_GROUP]	The URL group name of the matching URL
[PG_URL_HOST]	The hostname of the matching URL
[PG_URL_IE_ZONE]	The Internet Zone of the matching URL
[PG_URL_PROTOCOL]	The protocol of the matching URL
[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_USER_REASON]	The reason entered by the user
[PG_USER_SID]	The SID of the user
[PG_WORKSTYLE_NAME]	The name of the workstyle