Reference Guide

# Expert Rules Syntax for McAfee Endpoint Security Threat Prevention 10.5.3

For use with McAfee ePolicy Orchestrator

# Contents

# 1 Expert Rules overview

**Contents**



## Overview of Expert Rules

Expert Rules are text-based custom rules that you create in the Exploit Prevention policy in Threat Prevention.

Expert Rules provide additional parameters and allow much more flexibility than the custom rules you create in the Access Protection policy. But, to create Expert Rules, you must understand the McAfee proprietary syntaxes.

McAfee Endpoint Security includes two McAfee technologies and rule engines for Expert Rules: Arbitrary Access Control (AAC) and legacy McAfee Host IPS Core.

Each Expert Rule supports only one rule engine type. You can't mix different rule engine types in the same rule. For example, you can't combine a McAfee Host IPS-based rule, such as an Illegal API Use rule, with an AAC-based rule, such as a Files rule. Endpoint Security doesn't support signatures with multiple rules.

**Best practice:** Before writing Expert Rules, we recommend that you familiarize yourself with the Tcl programming language.

### AAC-based Expert Rules

AAC is a McAfee proprietary technology that Threat Prevention uses to protect key resources. You can extend this protection by creating rules to protect specific files, processes, and registry items. AAC-based Expert Rules use a new syntax used with the Tool Command Language (Tcl) interpreter version 7.6.

- **Files** — Protects files.
- **Processes** — Protects processes.
- **Registry** — Protects registry keys and registry values.

You can also create custom Files, Processes, and Registry rules in the Access Protection policy in Threat Prevention. But, these rules don't provide the complete functionality available with Expert Rules.

### Legacy McAfee Host IPS-based Expert Rules

These Expert Rules follow the same syntax as rules created using the Expert method in McAfee Host IPS. Endpoint Security supports the following legacy class types:

- **Buffer Overflow** — Prevents buffer overflow exploits for applications in the Application Protection list.
- **Illegal API Use** — Prevents illegal use of the Exploit Prevention API. The Expert Rules can only extend the functionality of the Illegal API Use signatures provided by Exploit Prevention content. Expert Rules can't refer to APIs that aren't already covered in an Illegal API Use signature available in content.
- **Services** — Protects Windows Services (Windows versions 8.0 and earlier only).

  You can also create custom Services rules in the Access Protection policy in Threat Prevention. But, these rules don't provide the complete functionality available with Expert Rules.

## Rule types and supported syntaxes

Endpoint Security provides two syntaxes for creating the different Expert Rule types.

| Rule type | AAC-based syntax | Legacy McAfee Host IPS-based syntax |
|---|---|---|
| Files | ✓ | ✗ |
| Registry | ✓ | ✗ |
| Processes | ✓ | ✗ |
| Buffer Overflow | ✗ | ✓ |
| Illegal API Use | ✗ | ✓ |
| Services | ✗ | ✓ |
| Program (McAfee Host IPS only) | ✗ | ✗ |

The new AAC **Processes** rule type replaces the McAfee Host IPS **Program** rule type, which is not supported in Endpoint Security.

# How Expert Rules work

Threat Prevention enforces Expert Rules on the client system the same as any other rule.

The signatures in the Exploit Prevention content provide default protection from McAfee Labs. If you need to protect additional resources, you can create custom rules in the Access Protection policy. For even further customization, create Expert Rules in the Exploit Prevention policy.

When writing an Expert Rule for any class type, Exploit Prevention automatically generates the rule ID and lets you configure:

- Name
- Severity (for documentation purposes only)
- Reaction (report, block, or both)
- Rule type
- Notes

You can enter or paste the rule syntax into the **Rule content** pane or use the template to define the rule syntax.

# Create Expert Rules

Use Expert Rules when you need to create Buffer Overflow, Illegal API Use, or Services rules, or more complicated Files, Processes, or Registry rules than Access Protection custom rules allow.

**Best practice** To isolate any potential issues, every time you create a rule, verify that it was successfully enforced on the client system. Check the EndpointSecurityPlatform_Errors.log file for compilation errors.

## Task

1. Select **Menu** | **Policy** | **Policy Catalog**, then select **Endpoint Security Threat Prevention** from the **Product** list.
2. From the **Category** list, select **Exploit Prevention**.
3. Click the name of an editable policy.
4. Click **Show Advanced**.
5. In the **Signatures** section, click **Add Expert Rule**.
6. In the **Rules** section, complete the fields.
   a. Select the severity and action for the rule.

      The severity provides information only; it has no effect on the rule action.

   b. Select the type of rule to create.

      The **Rule content** field is populated with the template for the selected type.

   c. Change the template code to specify the behavior of the rule.

      When you select a new class type, the code in the **Rule content** field is replaced with the corresponding template code.

   Endpoint Security assigns the ID number automatically starting with 20000. Endpoint Security doesn't limit the number of Expert Rules you can create.
7. Save the rule, then save the settings.
8. Enforce the policy to a client system.
9. Validate the new Expert Rule on the client system.

### See also

*Validate an Expert Rule on a client system* on page 8

# Create Expert Rules on a client system

Use Expert Rules when you need to create Buffer Overflow, Illegal API Use, or Services rules, or more complicated Files, Processes, or Registry rules than Access Protection custom rules allow.

### Before you begin

The interface mode for the Endpoint Security Client is set to **Full access** or you are logged on as administrator.

**Best practice** To isolate any potential issues, every time you create a rule, verify that it was successfully enforced on the client system.

### Task

1 Open the Endpoint Security Client.

2 Click **Threat Prevention** on the main **Status** page.

Or, from the **Action** menu, select **Settings**, then click **Threat Prevention** on the **Settings** page.

3 Click **Show Advanced**.

4 In the **Signatures** section, click **Add Expert Rule**.

5 In the **Rules** section, complete the fields.

a Select the severity and action for the rule.

The severity provides information only; it has no effect on the rule action.

b Select the type of rule to create.

The **Rule content** field is populated with the template for the selected type.

c Change the template code to specify the behavior of the rule.

When you select a new class type, the code in the **Rule content** field is replaced with the corresponding template code.

Endpoint Security assigns the ID number automatically starting with 20000. Endpoint Security doesn't limit the number of Expert Rules you can create.

6 Save the rule, then save the settings.

7 Validate the new Expert Rule on the client system.

### See also

*Validate an Expert Rule on a client system* on page 8

# Validate an Expert Rule on a client system

Once you deploy a new Expert Rule to a client test system, validate that the syntax is correct and that it is working properly before deploying more widely.

### Before you begin

The interface mode for the Endpoint Security Client is set to **Full access** or you are logged on as administrator.

Syntax checking is available for Files, Registry, and Processes rules only.

### Task

1 Open the Endpoint Security Client.

2 Click **Threat Prevention** on the main **Status** page.

Or, from the **Action** menu, select **Settings**, then click **Threat Prevention** on the **Settings** page.

3 Click **Show Advanced**.

4 Click **Exploit Prevention**.

5 In the **Signatures** section, double-click on a user-defined Expert Rule.

6 In the **Edit Rules** window, click **Check**.

If the syntax checker finds any errors:

a Review the EndpointSecurityPlatform_errors.log file for information about the syntax error.

b In Endpoint Security Client, correct the error.

c Click **Check**.

When all errors are fixed, the **Enforce** button is available.

7 Click **Enforce** to save and enforce the rule or **Close** to cancel any changes and close the window.

# 2 AAC-based Expert Rules

**Contents**



## AAC rule structure

Rules define the boundaries of acceptable behavior and tell AAC how to react when the filtered action matches the rule specifications.

The `Rule` command at the root level defines the rule. Each Expert Rule identifier can contain only one rule definition and multiple subrules. The `Match` command defines subrules, each of which has an assigned role: `Initiator` or `Target`.

Because `Initiator` subrules always apply to PROCESS objects, the `Process` command provides a shortcut method for defining `Initiator` sections.

Commands for building AAC rules are case sensitive.

Here is the basic structure of AAC-based rules:

```
Rule {
    Initiator {
        Match … {
            Include … { … }
            Exclude … { … }
        }
    }
    Target {
        Match … {
            Include … { … }
            Exclude … { … }
        }
    }
}
```

Endpoint Security doesn't support signatures with multiple rules.

## Valid parent-child relationships between AAC commands

The AAC syntax defines which commands can be the parent or children of other commands.

| Command | Parent | Children |
|---|---|---|
| Rule | Not applicable | Initiator |
| | | Process |
| | | Target |
| Initiator | Rule | Match |
| Process | Rule | Include |
| | | Exclude |
| Target | Rule | Match |
| Match | Initiator | Include |
| | Target | Exclude |
| Include | Process | Not applicable |
| | Match | |
| Exclude | Process | Not applicable |
| | Match | |

## How match criteria in AAC-based subrules are evaluated

The match criteria in each subrule specifies either the `Include` or `Exclude` directive. The rule engine evaluates the filtered event against the match criteria in the subrule.

The subrule matches the filtered event if both of the following are true:

- At least one `Initiator` subrule matches the process that initiated the action described by the event.
- At least one `Target` subrule matches the object type that is the subject of the action.

When evaluating a filtered event against a subrule, the rule engine performs logical OR between matching criteria of the same type and logical AND between matches of different type. The rule engine first evaluates the matches with the `Exclude` directive, and then evaluates the matches with the `Include` directive.

The subrule evaluates to TRUE if both of the following are true:

- Exclude matches evaluate to FALSE.
- Include matches evaluate to TRUE.

### Example

```
Rule {
    Initiator {
        Match PROCESS {
                Include OBJECT_TYPE_A      { … }
                Include OBJECT_TYPE_B      { condition 1 }
                Include OBJECT_TYPE_B      { condition 2 }
                Exclude OBJECT_TYPE_C      { … }
            }
        Target {
            Include OBJECT_TYPE_D     { condition 1 }
            Include OBJECT_TYPE_D     { condition 2 }
        }
}
```

This rule evaluates to TRUE if both the following are TRUE:

- One of the following `Initiator` conditions is TRUE:
  - OBJECT_TYPE_A and OBJECT_TYPE_B condition 1 are TRUE.
  - OBJECT_TYPE_A and OBJECT_TYPE_B condition 2 are TRUE.
  - OBJECT_TYPE_A is TRUE and OBJECT_TYPE_C is FALSE.
- One of the following `Target` conditions is TRUE:
  - OBJECT_TYPE_D condition 1 is TRUE.
  - OBJECT_TYPE_D condition 2 is TRUE.

# AAC commands for building Files, Processes, and Registry rules

### Contents



## Rule command

The `Rule` command defines an AAC rule. Each Expert Rule identifier can contain only one rule definition.

### Description

This command takes no arguments and can contain one or more `Initiator`, `Process`, and `Target` commands. Only the `Target` command is required.

### Syntax

```
Rule    {
    Initiator …
    Process …
    Target …
}
```

**See also**

*Initiator command* on page 13
*Process command* on page 14
*Target command* on page 14

## Initiator command

The `Initiator` command in a `Rule` command defines the AAC initiator matches. Only processes can be initiators.

### Description

This command takes no arguments and can contain only `Match` commands.

A `Rule` command must contain at least one Initiator command and can contain multiple `Initiator` commands. If the value isn't specified, the rule uses **** to indicate all processes.

### Syntax

```
Rule {
    …
    Initiator    {
        Match …
    }
    …
}
```

## Process command

The `Process` command provides a shortcut method for defining `Initiator` `Match` sections.

### Description

This command takes no arguments and can contain multiple `Include` and `Exclude` commands.

A `Rule` command can contain multiple `Process` commands. The `Process` command is optional. If not specified, the rule uses the value **** to indicate all processes.

### Syntax

```
Rule {    ...
    Process {
    }
    ...
}
```

This syntax is a shortcut for:

```
Rule    { ...
    Initiator {
        Match PROCESS {
        }
    }
    ...
}
```

## Target command

The `Target` command defines the AAC target matches for the rule.

### Description

This command takes no arguments and can contain only `Match` commands.

A `Rule` must contain at least one `Target` command and can contain multiple `Target` commands.

### Syntax

```
Rule {
    Target    {
        Match …
    }
 ...
}
```

# Match command

The `Match` command defines the criteria that AAC uses to match an event.

## Description

This command takes one required argument, object_type_value, which specifies the case-sensitive AAC object type to match, and can contain multiple `Include` and `Exclude` commands.

The `Match` command can be used in `Initiator` and `Target` commands only.

## Syntax

```
Rule    {
    Initiator
            Match object_type_value {
                Include …
                Exclude …
            }
    }
    Target
            Match object_type_value {
                Include …
                Exclude …
            }
    }
}
```

**See also**

*Include and Exclude commands* on page 16
*Match object type values* on page 15

## Match object type values

The `Match` command takes one required argument, object_type_value, which is the case-sensitive AAC object type to match.

This table lists the valid values of object_type_value.

| Match object_type_ value | Description | Valid match object type | Notes |
|---|---|---|---|
| FILE | Controls access to a file. | Target | |
| KEY | Controls access to both key and value data in a key object. | Target | |
| PROCESS | Controls access to a process handle. | • Initiator<br>• Target | If PROCESS is not used in the Initiator match, you must use THREAD.<br>If the access to be blocked is CREATE, the object type must be SECTION rather than PROCESS. |
| SECTION | Controls access to creating a section object. | Target | |
| THREAD | Controls access to a thread handle. | • Initiator<br>• Target | If THREAD is not used in the Initiator match, you must use PROCESS. |
| VALUE | Controls access to value data in a key object. | Target | |

**See also**

*Match command* on page 15

# Include and Exclude commands

The `Include` and `Exclude` commands specify the data used for matching.

## Description

The `Include` and `Exclude` commands take two required arguments:

- MATCH_type, which determines the entries in an `Include` or `Exclude` that are ORed or ANDed
- The actual data to match

  The body of the command can contain multiple data entries. Each data entry must begin with either `-v` or `-l`.

## Syntax

```
Rule    {
    Initiator
        Match    {
            Include MATCH_type < -type PATH > {
                -v data | -l data
                ...
            }
        }
    }
}
```

```
Rule    {
    Initiator
        Match    {
            Exclude MATCH_type < -type PATH > {
                -v data | -l data
                ...
            }
        }
    }
}
```

## Arguments

| Argument | Description |
|---|---|
| -v | Specifies to interpret the following entry as a single value. |
| -l | Specifies to interpret the following entry as a Tcl list — each entry in the list is automatically broken out into its own match entry. |
| -type PATH | Treats all entries in the body as paths and automatically removes any trailing directory separators: **/** or ****.<br>This is useful to avoid double separators when you are appending strings to the values with the `-sfx` option. |

## Shortcuts for MATCH_type

You can use the following shortcuts instead of building the entire MATCH_type entry.

### Syntax

```
Include/Exclude -processor_mode  user|kernel
```

```
Include/Exclude -vtp_trust  true|false
```

```
Include/Exclude -access access_types
```

The access_types value is a list of access tokens separated by a delimiter and is case insensitive. The valid delimiters are a space, tab, comma, or pipe **|**.

The valid access tokens are:

- CLEANUP
- CLOSE
- CONNECT_NAMED_PIPE
- CREATE
- DELETE
- ENUM
- EXECUTE
- LOAD_IMAGE
- LOAD_KEY
- OPEN_NAMEDSECTION
- POST
- QUERY
- READ
- REPLACE_KEY
- RESTORE_KEY
- SET_REPARSE
- SET_SECURITY
- START_DEVICE
- TERMINATING
- WRITE
- WRITE_ATTRIBUTE

### See also

*Match type values* on page 17

## Match type values

The MATCH_type value determines which entries in an `Include` or `Exclude` are ORed or ANDed. Commands with the same MATCH_type value evaluate to either value (OR). Commands with different MATCH_type values evaluate to both values (AND).

MATCH_types values are case sensitive.

| Match type value | Description | Valid in object types |
|---|---|---|
| ACCESS_MASK | Specifies the access type. | All |
| AUTHENTICATION_ID | Matches a textual account SDDL SID identifier. This match can be used to identify a specific user-account in policy enforcement.<br>For information about SDDL strings, see http://msdn.microsoft.com/en-us/library/windows/desktop/aa379602(v=vs.85).aspx. | All |
| CACHE_ATTRIBUTE | Matches a cache attribute for the given object.<br>Because it is a bitmask match type, any matching bits are considered a match. | • PROCESS<br>• FILE |

| Match type value | Description | Valid in object types |
|---|---|---|
| CERT_NAME | Matches the object's signing certificate name, but doesn't check whether the certificate is chained to the root.<br>If the object is of type PROCESS or THREAD, the certificate is obtained from the main entry module. This match never evaluates to true if the object is not signed. | • PROCESS<br>• THREAD<br>• SECTION |
| CERT_NAME_CHAINED | Matches the object's signing certificate name, and the signing certificate must be chained to the root of the certificate store.<br>If the object is of type PROCESS or THREAD, the certificate is obtained from the main entry module. This match never evaluates to true if the object is not signed. | • PROCESS<br>• THREAD<br>• SECTION |
| DESCRIPTION | Matches the “FileDescription” resource extracted from the resource section for the PE. | • PROCESS<br>• FILE<br>• SECTION |
| DLL_LOADED | Matches a loaded DLL in a specified PROCESS object.<br>This is primarily useful for narrowing `Initiator` matches, such as svchost.exe service exclusions. The DLL name generally is the base name of the DLL without a path or file extension. That is, “MFEVTPA” matches, whereas “MFEVTPA.DLL” or “c:\program files\common files\mcafee \systemcore\mfevtpa.dll”. The match data is pulled directly from the process structures where the DLL is known by its base name and the associated image file name is not present.<br>To match when the DLL is loaded, set the value part of the name-value bitmask to 1. To match when the DLL is not loaded, set it to 0. | PROCESS |
| ENV_VAR | Specifies an environment variable name and its value. This criteria matches only if both name and value match the environment variables extracted from the PEB. | • PROCESS<br>• THREAD |
| FILE_ATIME | Matches against the file last accessed time. | • PROCESS<br>• FILE |
| FILE_ATTRIBUTES | Matches against the file attribute bits. | • PROCESS<br>• FILE |
| FILE_CTIME | Matches against the file create time. | • PROCESS<br>• FILE |
| FILE_MTIME | Matches against the file last changed time. | • PROCESS<br>• FILE |

| Match type value | Description | Valid in object types |
|---|---|---|
| FILE_PROPERTIES | Matches the bitmask against file properties reported by the `Target`. The defined bits are:<br>• NETWORK (0x1) — File is in a network path.<br>• REMOVABLE (0x2) — File is on a removable drive.<br>• FLOPPY (0x4) — File is on a floppy drive.<br>• CD (0x8) — File is on a CD drive.<br>• DFS (0x10) — File is over on DFS.<br>• REDIRECTOR (0x20) — File is opened using a redirector. | FILE |
| GROUP_SID | Matches the provided textual SID (that is, S-1-5-18) against the groups that the user token belongs to.<br>The criteria evaluates to true if at least one matching group is found. | • PROCESS<br>• THREAD |
| IMAGE_BASE_ADDRESS | Specifies the virtual base address for an image.<br>This is useful for retrieving the base address for an image during an image load notification. | SECTION<br>Available only during load image callbacks, access mask set to LOAD_IMAGE. |
| IMAGE_ENTRY_POINT | Specifies the entry point offset (in bytes) for an image.<br>This is useful for retrieving the entry point address for an image during an image load notification. | SECTION<br>Available only during load image callbacks, access mask set to LOAD_IMAGE. |
| IMAGE_PROPERTIES | Specifies different image properties, as available during an image load notification.<br>The defined bits are:<br>• 64-bit — 64-bit image.<br>• SYSTEM_MODE — System mode image.<br>• MAPPED_TO_ALL_PROCESSES — The image is mapped to all processes. | SECTION<br>Available only during load image callbacks, access mask set to LOAD_IMAGE. |
| MD5 | Indicates the MD5 digest of the backing file. If object is of type PROCESS or THREAD, MD5 is calculated against its main executable module. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |

| Match type value | Description | Valid in object types |
|---|---|---|
| NT_ACCESS_MASK | Matches against the native NT access mask of the I/O operation for file, registry, process, and thread access attempts. Make sure to use access masks appropriate for the object type as described in Microsoft MSDN.<br>• **FILE** — https://msdn.microsoft.com/en-us/library/windows/desktop/aa364399(v=vs.85).aspx<br>• **PROCESS** — https://msdn.microsoft.com/en-us/library/windows/desktop/ms684880(v=vs.85).aspx<br>• **THREAD** — https://msdn.microsoft.com/en-us/library/windows/desktop/ms686769(v=vs.85).aspx<br>• **REGISTRY** — https://msdn.microsoft.com/en-us/library/windows/desktop/ms724878(v=vs.85).aspx<br>For example, to use NT_ACCESS_MASK to block calls to CreateFile() with GENERIC_WRITE, the bit mask must be FILE_GENERIC_WRITE.<br>Due to operating system limitations, you can't block PROCESS_QUERY_LIMITED_INFORMATION but you can use it in ALLOW rules for reporting purposes. | • PROCESS<br>• THREAD<br>• FILE<br>• REGISTRY |
| OBJECT_NAME | Specifies the object name. Any combination of wildcards is accepted. | All |
| OBJECT_SIZE | Matches against the size of the file or, for a section, the image size during load. | • FILE<br>• SECTION |
| OPERATION_STATUS | Matches the operation status for a post-event. Not useful with non-post events. | FILE |
| OS_VERSION | Compares the specified OS version to the actual version. The OS version must be specified in the format:<br>`OS_Version = Major_Version * 1000 + Minor_Version * 10 + ServicePack. By way of example: VistaRtm = 6000; VistaSp1=6001; Win7=6010; Win7Sp1=6011; Win8=6020` | All |
| PE | Matches a data value of "1" if the target file is a PE (Portable Executable, Windows executable binary) file.<br>`Initiator` PROCESS/THREAD matches are not supported because, by definition, they are PE files. | FILE |
| PE_MD5 | Compares MD5 digest calculated across PE against the match criteria.<br>The digest is calculated according to Microsoft's Authenticode PE hash value calculations – 4-byte PE header check sum is omitted as well as the Certificate Table Entry, which is part of Optional Header Directories. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| PE_SHA1 | Compares the match data with the SHA-1 hash sum calculated across the PE. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |

| Match type value | Description | Valid in object types |
|---|---|---|
| PE_SHA2_256 | Compares the match data with the SHA2-256 hash sum calculated across the PE. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| PE_SHA2_384 | Compares the match data with the SHA2-384 hash sum calculated across the PE. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| PE_SHA2_512 | Compares the match data with the SHA2-512 hash sum calculated across the PE. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| PROCESSOR_MODE | Matches if the match is evaluated in the context of an I/O operation originating from user-mode or kernel-mode.<br>This is most useful for excluding processes from matching a rule if the process is executing in user-mode.<br>Do not use this type with registry operations. | • PROCESS<br>• THREAD |
| PROCESS_CMD_LINE | Compares the command-line parameters received by the process. | • PROCESS<br>• THREAD |
| PROCESS_ID/THREAD_ID | Matches a specified thread ID.<br>Remember when using this match type that thread IDs and process IDs are rapidly recycled in the Windows environment. | • PROCESS<br>• THREAD |
| PROCESS_STATE_BITS | Compares the specified name/bitmask with the stateID/stateBits carried by the `Initiator` or `Target` ProcessInfo object. The comparison evaluates to true if stateBits with stateID are present in ProcessInfo and the “bitwise and” between the stateBits and the bitmask carried by the match object yields a non-zero result. | • PROCESS<br>• THREAD |
| PRODUCT_NAME | Matches the "ProductName" resource extracted from the resource section of the PE. | • PROCESS<br>• FILE<br>• SECTION |

| Match type value | Description | Valid in object types |
|---|---|---|
| REMOTE_MACHINE_ADDRESS | This type is for reporting only.<br>If used for matching, matches the specified type against file I/O initiated by a specific SMB client IP address in either IPv4 or IPv6 format.<br>In other words, this type does not match for file I/O initiated on the local system going to an SMB server. It only matches for client I/O going to the local SMB server. This match type is mostly useful for generating event details. | This match type is valid in PROCESS `Initiator` (requires OBJECT_NAME to match SYSTEM:REMOTE) or FILE `Target` match. |
| SESSION_ID | Compares the specified match criteria against the session ID that the process/thread belongs to and can apply to both `Initiator` and `Target` objects. | • PROCESS<br>• THREAD |
| SHA1 | Compares the SHA-1 hash sum of the backing file with the match data.<br>If the object is of type PROCESS or THREAD, the hash sum is calculated against its main executable module. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| SHA2_256 | Compares the SHA2-256 hash sum of the backing file with the match data.<br>If the object is of type PROCESS or THREAD, the hash sum is calculated against its main executable module. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| SHA2_384 | Compares the SHA2-384 hash sum of the backing file with the match data.<br>If the object is of type PROCESS or THREAD, the hash sum is calculated against its main executable module. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| SHA2_512 | Compares the SHA2-512 hash sum of the backing file with the match data.<br>If the object is of type PROCESS or THREAD, the hash sum is calculated against its main executable module. | • PROCESS<br>• THREAD<br>• FILE<br>• SECTION |
| TARGET_OBJECT_NAME | Specifies the object name. Any combination of wildcards is accepted.<br>Names follow the same conventions as OBJECT_NAME. But, they only match against the target of a file rename operation. This enables rules to be written that only apply to renames based on both source (OBJECT_NAME) and target (TARGET_OBJECT_NAME) name.<br>• OBJECT_NAME is not required. If it is not specified, any source matches.<br>• ACCESS_MASK for a rename is DELETE, because it's from the perspective of the source file, even if the OBJECT_NAME is not specified. | FILE |
| USER_SID | Matches the text representation of the user account SID (that is, S-1-5-21-22-23-24-1168). | • PROCESS<br>• THREAD |

<table>
<tr><th>Match type value</th><th>Description</th><th>Valid in object types</th></tr>
<tr><td>VERSION_RESOURCE</td><td>Matches the “FileVersion” resource extracted from the resource section for the PE.</td><td>• PROCESS<br>• FILE<br>• SECTION</td></tr>
<tr><td>VERSION</td><td>Matches the version extracted from the resource section for the file.</td><td>• PROCESS<br>• THREAD<br>• SECTION</td></tr>
<tr><td>VTP_PRIVILEGES</td><td>Matches the bitmask against the VTP privileges of the target.<br>The defined bits are:<br>• PRIVILEGE_IOCTL (0x1) — Signed by a VTP-trusted certificate.<br>• PRIVILEGE_ISG (0x8) — Signed by a McAfee certificate specifically.<br><table><tr><th>Files signed by</th><th>VTP_TRUST</th><th>VTP_PRIVILEGES</th><th>=0x08</th><th>=0x09</th></tr><tr><td>Microsoft</td><td>Yes</td><td>Yes</td><td>No</td><td>Yes</td></tr><tr><td>McAfee</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr><tr><td>3rd party</td><td>No</td><td>No</td><td>No</td><td>No</td></tr></table></td><td>• FILE<br>• PROCESS<br>• THREAD</td></tr>
<tr><td>VTP_TRUST</td><td>Checks if VTP trusts the process or file.<br>The value is treated as Boolean. That is, a value of 1 in the match type matches only processes trusted by VTP. A value of 0 matches non-trusted processes.</td><td>• PROCESS<br>• THREAD<br>• SECTION</td></tr>
<tr><td>WOW64</td><td>Matches a data value of "1" if the process/thread is a WOW64 process.<br>This can only be true on 64-bit platforms and always matches a "0" on 32-bit platforms.<br>This match can apply to both <code>Initiator</code> and <code>Target</code> objects.</td><td>• PROCESS<br>• THREAD</td></tr>
</table>

**See also**

*Include and Exclude commands* on page 16
*OBJECT_NAME guidelines* on page 24
*ACCESS_MASK flags* on page 26

## OBJECT_NAME guidelines

Use these guidelines when specifying the OBJECT_NAME match value in a Match_type value. You can use any combination of wildcards.

| OBJECT_NAME value | Notes |
|---|---|
| Disk name | Accepted formats are:<br>• HardDiskXX — HardDisk0<br>• $(SystemDrive) — The disk that contains the system volume. |
| Fully qualified file path | AAC doesn't support short paths.<br>• System — Specifies the system process name.<br>To match based on the thread running in the system process context, the rule must set an `Initiator` command to "System".<br>• System:Remote — Specifies the system process name for remote systems.<br>To match file operations for a remote system, the rule must set an `Initiator` command to "System:Remote".<br>To match based on both “System” and “System:Remote”, configure the rule to specify 2 matches or specify “System*”. |

<table>
<tr><th>OBJECT_NAME value</th><th>Notes</th></tr>
<tr><td>Fully qualified registry key/value path</td><td>These root keys are recognized:<br>Key — Matches<br>HKLM — HKLM is equivalent to HKEY_LOCAL_MACHINE.<br>HKCU — All user registry keys (not just the current user) and the .default user key. HKCU is equivalent to:<br>• HKEY_CURRENT_USER<br>• HKEY_USERS<br>Matching against specific user SIDs is not supported.<br>HKCUC — All user classes (HKCU/*_CLASSES).<br>HKCR — System classes and all user classes (HKCU/*_CLASSES). HKCR is equivalent to HKEY_CLASSES_ROOT.<br>HKCCS — • HKLM/SYSTEM/CurrentControlSet • HKLM/SYSTEM/ControlSet00X<br>HKLMS — • HKLM/Software on 32-bit and 64-bit systems • HKLM/Software/Wow6432Node on 64-bit systems only<br>HKCUS — • HKCU/Software on 32-bit and 64-bit systems • HKCU/Software/Wow6432Node on 64-bit systems only<br>HKULM — • HKLM • HKCU<br>HKULMS — • HKLMS • HKCUS<br>HKALL — • HKLM • HKU<br>If the rule specifies a name where the root starts or contains a wild character, the AAC code performs no name normalization and that name might never match correctly. For example, **\mcshield\start is a valid name, but H*L*\mcshield\start is not.<br>HKEY_CURRENT_CONFIG is not supported.</td></tr>
<tr><td>Fully qualified section name</td><td></td></tr>
<tr><td>Process name or fully qualified process path</td><td>Process name must also be specified for thread objects.</td></tr>
<tr><td>Volume name</td><td>• Must be specified in the format:<br><code>Volume{35FC9B67-54AC-49ff-AB99-33FFA2999670}</code><br>• $(SystemDrive) — Immutable and always applies to the system volume.</td></tr>
</table>

**See also**

*Match type values* on page 17

## ACCESS_MASK flags

Use these flags with the ACCESS_MATCH Match_type value.

| Flag | Applies to object types | Applies when |
|---|---|---|
| CONNECT_NAMED_PIPE | FILE (representing a named pipe) | Attempt to connect to a named pipe. |
| CREATE | • FILE<br>• KEY<br>• PROCESS<br>• THREAD<br>• SECTION | • File, Key, Process, or Thread is created.<br>If the Target to be blocked is a process, specify the object type as SECTION rather than PROCESS.<br>• File is open for execute (SECTION object).<br>This doesn't mean that the SECTION object itself is created, rather that a SECTION object can be created. The SECTION object might not be created for execute. |
| DELETE | • FILE<br>• KEY<br>• PROCESS<br>• THREAD | • File or Key (not registry values) is deleted or set security is called.<br>• Process is opened with PROCESS_TERMINATE.<br>• Thread is opened with THREAD_TERMINATE. |
| ENUM | • KEY<br>• VALUE | • Key is opened with KEY_ENUMERATE_SUB_KEYS.<br>• Values are enumerated with RegEnumValue. |
| EXECUTE | FILE | • File is opened with FILE_EXECUTE access.<br>• SECTION object is created with SECTION_MAP_EXECUTE.<br>  **Best practice** Blocking SECTION objects might cause Windows to call a NtRaiseHardError(). To block loading unwanted code without this side-effect, use CREATE with SECTION.<br>• Directory is opened with traverse access. |
| LOAD_IMAGE | SECTION | Notification only (cannot block the image load). |
| LOAD_KEY | KEY | Registry hive is loaded into a key with ZwLoadKey or RegLoadKey. |
| LOCK_RANGE | | Attempt to lock or unlock a byte-range lock on a file.<br>Use this access mask to protect a log file. You don't need to use this access mask for files that you aren't going to WRITE to at runtime, but byte-range locks don't stop reading and executing files. |
| OPEN_FOR_DELETE | FILE | Create/open event that requested delete access. |
| POST | FILE | Post-operation event.<br>Events that carry this bit only match against rules that have this bit set. Also, if the access mask contains other bits set (not including POST), the rule evaluates to true only if at least one other bit matches the event. |
| QUERY | • KEY<br>• VALUE | Attempt to query a registry key/value occurs. |

| Flag | Applies to object types | Applies when |
|---|---|---|
| READ | • FILE<br>• KEY<br>• VALUE | Existing file/key is being opened for read access.<br>This does not match with registry key/value enum/query operations. See ENUM and QUERY for matching against registry query/enum operations. |
| READ_DATA | FILE | An actual read file I/O occurs (ReadFile executed from user-space). |
| RENAME | • FILE<br>• KEY<br>• VALUE | Registry key or file rename operation occurs. |
| REPLACE_KEY | KEY | Registry key is replaced (RegReplaceKey). |
| RESTORE_KEY | KEY | Registry key is restored (RegRestoreKey). |
| SET_FILE_LENGTH | FILE | Any operation that changes the file length (ZwSetInformationFile), where class is one of:<br>• FileEndOfFileInformation<br>• FileAllocationInformation<br>• FileValidDataLengthInformation<br>This access bit helps with file-copy detection, when the destination file is extended and then written to. |
| SET_REPARSE | FILE | Attempt to set the reparse data on a file or directory object.<br>Do not use this access mask with IS_DIRECTORY. Attempts to set a reparse point on an alternate data stream don't match correctly. This is because the file system always considers alternate data streams as "file" objects, even if the base file object is a directory. But, reparse data is configurable from an alternate data stream file handle on a directory, which causes STATUS_REPARSE to be returned for all streams of a directory or file object. |
| TERMINATING | • PROCESS<br>• THREAD | Notification only (cannot block a terminate action). |

| Flag | Applies to object types | Applies when |
|---|---|---|
| WRITE | • FILE<br>• KEY<br>• VALUE<br>• PROCESS | • Existing file is opened for write (FILE_GENERIC_WRITE and disposition TRUNCATE_EXISTING).<br>File rules, using this flag, and specifying the file name as a fully qualified path including drive letter, also matches rename operations for any of the upper-level directories. For example, if the rule specifies "c:\program files\mcafee\systemcore**", this rule matches rename operations against:<br>• c:\program files\mcafee\systemcore<br>• c:\program files\mcafee<br>• c:\program files\<br>But the rule doesn't match:<br>• c:\program files\microsoft<br>• c:\program files\mcafee\VSE<br>• Existing key is opened for write (KEY_WRITE).<br>• Process is opened for write access:<br>• PROCESS_CREATE_PROCESS<br>• PROCESS_CREATE_THREAD<br>• PROCESS_DUP_HANDLE<br>• PROCESS_SET_QUOTA<br>• PROCESS_SET_INFORMATION<br>• PROCESS_SUSPEND_RESUME<br>• PROCESS_VM_OPERATIONS<br>• PROCESS_VM_WRITE<br>• Handle to the thread is opened with write access:<br>• THREAD_DIRECT_IMPERSONATION<br>• THREAD_IMPERSONATE<br>• THREAD_SET_CONTEXT<br>• THREAD_SET_INFORMATION<br>• THREAD_SET_LIMITED_INFORMATION<br>• THREAD_SET_THREAD_TOKEN<br>• THREAD_SUSPEND_RESUME<br>• Registry value is created, written, or deleted.<br>Values are considered the data of a key. |
| WRITE_ATTRIBUTE | FILE | File or directory's attributes are written to. |
| WRITE_DATA | FILE | Actual write file I/O (WriteFile executing from user-space). |

**See also**

*Match type values* on page 17

# AAC commands for querying system state

**Contents**



## iDump command

The `iDump` command dumps global variables defined in the rule to the log file if debug logging is enabled.

### Syntax

```
iDump filter
```

If filter is not specified, this command dumps all variables.

### Parameter

| Parameter | Description |
|---|---|
| filter | String that represents the names of the global variables to dump. The filter parameter can contain wildcards. |

## iEnv command

The `iEnv` command returns the specified environment variable value or an empty string if the variable does not exist.

### Syntax

```
iEnv name
```

### Parameter

| Parameter | Returns |
|---|---|
| name | Value of the specified environment variable. |

### Example

```
set PingExe [iEnv SystemRoot]\\system32\\ping.exe
```

## iList command

The `iList` command sorts the values in the list in ascending order and removes duplicate values.

### Syntax

```
iList -d list
```

### Parameter

| Parameter | Description |
|---|---|
| -d | Indicates that the list contains directory names and converts all directory characters to the proper format for the operating system.<br>Any duplicate separators are combined into one before the comparisons are done, any trailing directory characters are removed before the list is returned.<br>If an entry is a subdirectory of another element, only the parent directory is returned. |

### Example

```
set alist {{c:/tmp\\ } {c:\tmp\a} {c:\tmp/b/c} {d:\debug}}
set blist [iList -d $alist]
```

The "blist" list now contains:

```
{{c:\tmp} {d:\debug}}
```

## iReg command

The `iReg` command reads information from the local registry.

### Syntax

```
iReg [-32] param
```

### Parameters

To read the 32-bit hive on a 64-bit operating system, specify `-32` as the first argument.

| Parameter | Description |
|---|---|
| open keyname | Opens a registry key named keyname and returns "1" if successful or "0" otherwise. Closes the key when the scanning session is over. |
| exist keyname | Tests to see if a registry key named keyname exists and returns "1" if it exists or "0" otherwise. |
| value keyname valuename | Reads information from the registry key keyname with the value name of valuename.<br>If the value is type:<br>• `string` — Returns the string value.<br>• `int` — Returns the string value.<br>• `MULTI_SZ` — Returns a Tcl list. |
| keys keyname | Returns a list of subkeys that exist under the key specified by keyname. |
| v_exists keyname valuename | Tests to see if the valuename item exists under the key specified by keyname and returns "1" if exists or "0" otherwise. |

You can use the following shortcuts for the registry keyname.

| Keyname | Shortcut |
|---|---|
| HKEY_LOCAL_MACHINE | HKLM |
| HKEY_CLASSES_ROOT | HKCR |
| HKEY_CURRENT_CONFIG | HKCC |

| Keyname | Shortcut |
|---|---|
| HKEY_CURRENT_USER | HKCU |
| HKEY_USERS | HKUS |

For example, to specify the software hive on the local system, use HKLM\\Software.

## iSystem command

The `iSystem` command returns information about the client system where the rule is executed.

### Syntax

```
iSystem param
```

### Parameters

| Parameter | Returns |
|---|---|
| version | Version of the operating system in the format *major.minor.build*. |
| major | Major version of the operating system. |
| minor | Minor version of the operating system. |
| build | Build number of the operating system. |
| csd | CSD value. Usually, this is the Service Pack in the form of a string, such as "Service Pack 1". |
| platform | String with the platform name, for example, "Windows 7". |
| type | System type:<br>• Workstation<br>• Server<br>• Unknown |
| cpu_arch | CPU architecture:<br>• 320 for 32-bit CPU<br>• 640 for 64-bit AMD type CPU<br>• 641 for 4-bit Itanium type CPU |
| os_arch | Operating system architecture:<br>• 320 for 32-bit operating system<br>• 640 for 64-bit operating system |
| install_dir | Location of the Windows installation directory. |

| Parameter | Returns |
|---|---|
| sys32_dir | Location of the System32 folder. |
| users_folders *folder_types* | List of folder locations for all users created on the system.<br>You can specify the types of folders to return.<br>The valid folder types are listed in HKEY_USERS\<user sid>\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders<br>In addition, you can specify these special folder types:<br>• Temp — All temp folders on the system<br>• Profile — All users' profile root folder<br>• Downloads — All users' download locations |

## iTerminate command

The `iTerminate` command stops building the rules and adds the specified message text to the error log.

### Syntax

```
iTerminate "msg"
```

### Parameter

| Parameter | Description |
|---|---|
| msg | The message to add to the error log. |

## iUser command

The `iUser` command returns information about users on a system.

### Syntax

```
iUser param
```

### Parameters

| Parameter | Returns |
|---|---|
| username | "1" if the user exists on the system, otherwise "0". |
| list | List of all users on the system. |
| groups username | List of the groups a user belongs to. |

# AAC rule examples

### Contents

## Preventing file creation

This example rule prevents users from creating files with the name "*test.txt" from cmd.exe in the C:\temp folder.

```
Rule {
    Process {
        Include OBJECT_NAME { -v cmd.exe  }
    }
    Target {
        Match FILE {
            Include OBJECT_NAME {
                -v "c:\\temp\\*test.txt"
            }
   Include -access "CREATE"
        }
    }
}
```

## Preventing users from changing a registry value

This example rule prevents all users from changing the registry value under HKLMS\test.

```
Rule {
    Process {
        Include OBJECT_NAME {
            -v regedit.exe
        }
    }
Target {
    Match KEY {
        Include OBJECT_NAME {
            -v "HKLMS\\test**"
        }
        Include -access "CREATE WRITE DELETE REPLACE_KEY RESTORE_KEY"
        }
    }
}
```

## Blocking specified PowerShell parameters

This example rule prevents PowerShell from executing with specific command-line parameters, except for the encoded command, which is "dir c:\program files".

```
Rule {
    Process {
        Include OBJECT_NAME      { -v "*PowerShell*" }
        Include PROCESS_CMD_LINE { -v "*-NoLogo*" }
        Include PROCESS_CMD_LINE { -v "*-File*" }
        Include PROCESS_CMD_LINE { -v "*-EncodedCommand*" }
        Include PROCESS_CMD_LINE { -v "*-Command*" }
        Exclude PROCESS_CMD_LINE { -v "*-EncodedCommand
                ZABpAHIAIAAnAGMAOgBcAHAAcgBvAGcAcgBhAG0AIABmAGkAbABlAHMAJwAgAA==" }
    }
    Target {
        Match SECTION { Include -access "CREATE" }
    }
}
```

## Allowing a file to be created only from an excluded folder

This rule prevents the test.dat file from being created or read from any folder other than Program Files folder.

```
Rule {
    Process {
        Include OBJECT_NAME { -v cmd.exe }
    }
    Target {
        Match FILE {
            Include OBJECT_NAME { -v **\\test.dat }
            Exclude OBJECT_NAME -type PATH {
                -v "c:\\program files\\test.dat"
            }
            Include -access "CREATE READ EXECUTE"
        }
    }
}
```

## Logging environment variables

This example rule dumps environment variables to the debug log, and prevents cmd.exe from creating .exe and .dll files in the Program Files folder. You can check the values in the Access Protection debug log file.

```
Rule {
    set test_var1 [iSystem major]
    set test_var2 [iSystem os_arch]

    set os_major_version [iSystem major]
    set os_arch [iSystem os_arch]
        if { $os_arch == 320 } {
            set test_var3 {%windir%\\System32}
        } else {
            set test_var4 {%windir%\\System32}
            set test_var5 {%windir%\\Syswow64}
        }

        if { $os_major_version >= 6 } {
            set test_var6 "%programdata%\\McAfee\\Endpoint Security"
        } else {
            set test_var7 "%allusersprofile%\\Application Data\\McAfee\\Endpoint Security"
        }

        set test_var8 "HKLM\\SOFTWARE\\McAfee\\Endpoint"
        lappend test_var9 [iReg value $test_var8 szInstallDir32]
        lappend test_var9 [iReg value $test_var8 szInstallDir64]
        set test_var9 [iList -d $test_var9]

        set test_var10 [iReg value HKCR\\http\\shell\\open\\command ""]
        set test_var10 [iUtil cvt2args $test_var10 ]
        set test_var10 [lindex $test_var10 0]

        iDump test_

        Process {
            Include OBJECT_NAME { -v cmd.exe  }
        }
        Target {
            Match FILE {
                Include OBJECT_NAME {
                    -v "%programfiles%\\**.exe"
                    -v "%programfiles%\\**.dll"
                    -v "%programfiles(x86)%\\**.exe"
                    -v "%programfiles(x86)%\\**.dll"
                }
                Include -access "CREATE"
            }
        }
}
```

# Troubleshooting AAC-based rules

McAfee Endpoint Security provides information in the EndpointSecurityPlatform_Errors.log file about rules that didn't successfully compile and so were not enforced.

Because all Expert Rules are compiled into a single group, when an Expert Rule generates an error, no Expert Rules are enforced.

**Best practice** To isolate any potential issues, every time you create a rule, verify that it was successfully enforced on the client system.

The EndpointSecurityPlatform_Errors.log file includes detailed information, such as the content of the rule and the parameter that caused the error. For example, this log error shows the Expert Rules error, which is an extra `Include` command:

```
08/11/2017 11:57:34.403 AM   mfeesp(4016.4412) <SYSTEM> ApBl.AP.Error: Syntax error:
Include: Invalid number of arguments
    while executing
"Include Include OBJECT_NAME  { -v "*PowerShell*" }"
        Include Include OBJECT_NAME  { -v "*PowerShell*" }
        Include PROCESS_CMD_LINE { -v "*-extoff* script.scp" }
        Include ..."
    invoked from within
"Process {
        Include OBJECT_NAME      { -v "*PowerShell*" }
        Include PROCESS_CMD_LINE { -v "*-extoff*" }
        Include PROCE ..."
    invoked from within
"Rule -id "4100" {
    Reaction BLOCK
    Group "ExPExpertRules"
    Description "testrule"
    Process {
         Include AggregateMatch {
         Include OBJECT_NAME      {  ..."
    invoked from within
"Policy {
Rule -id "4100" {
    Reaction BLOCK
    Group "ExPExpertRules"
    Description "testrule"
    Process {
         Include AggregateMatch {
         Include OBJECT_NA ..."LastErr 0x000010dd The operation identifier is not valid.
08/11/2017 11:57:34.403 AM   mfeesp(4016.4412) <SYSTEM> ApBl.AP.Error: ERR: BLError
0xc0380102, Could not process content file
```

# 3 Legacy McAfee Host IPS-based Expert Rules

**Contents**



## Legacy McAfee Host IPS rule structure

Rules contain both required and optional sections, one section per line. Each section defines a rule category and its value. One section always identifies the class of the rule, which defines the rule's overall behavior. Optional sections vary according to the class of the rule.

Here is the basic structure of a McAfee Host IPS rule:

```
Rule {
    SectionA value
    SectionB value
    SectionC value
    ...
}
```

Because the structure and class types for legacy Expert Rules are identical to those in McAfee Host IPS, you can copy existing McAfee Host IPS rules into Endpoint Security Expert Rules.

Endpoint Security doesn't support signatures with multiple rules.

## Legacy syntax

**Contents**

## Wildcards

You can use wildcards for section values in Expert Rules.

| Wildcard character | Represents |
|---|---|
| **?** (question mark) | A single character. |
| ***** (one asterisk) | Multiple characters, including **/** and ****.<br>For paths and addresses, use ****** (2 asterisks) to include **/** and ****. Use ***** (one asterisk) to exclude **/** and ****. |
| **&** (ampersand) | Multiple characters except **/** and ****.<br>Use **&** to match the root-level contents of a folder, but no subfolders.<br>For example:<br>`Include`<br>`"C:\test\\&.txt"` |
| **!** (exclamation point) | Wildcard escape.<br>For example:<br>`Include`<br>`"C:\test\\yahoo!.txt"` |

## Environment variables

Use environment variables to specify file and directory path names.

The `iEnv` command takes one parameter (the variable name) in square brackets **[ ]**.

| Environment variable | Represents |
|---|---|
| iEnv SystemRoot | C:\winnt\, where C is the drive that contains the Windows System folder.<br>For example:<br>`Include [iEnv SystemRoot]\\system32\\abc.txt` |
| iEnv SystemDrive | C:\, where C is the drive that contains the Windows System folder.<br>For example:<br>`Include [iEnv SystemDrive]\\system32\\abc.txt` |

## Using the Include and Exclude keywords

When you select a section value as `Include`, the section works on the value indicated. When you select a section value as `Exclude`, the section works on all values except the one indicated.

The keywords `Include` and `Exclude` are supported for all sections except `directives` and `attributes`.

Enclose the `Include` and `Exclude` keywords in brackets { ... }.

For a standard subrule, use a single backslash in file paths. The standard subrule translates the single slashes to required double slashes. For a subrule in an Expert Rule, use double backslashes in file paths. The expert subrule performs no translation.

For example, to monitor all text files in C:\test\:

```
files { Include C:\\test\\*.txt }
```

To monitor all files except the text files in C:\test\:

```
files { Exclude C:\\test\\*.txt }
```

Combine keywords to exclude values from a set of included values.

For example, to monitor all text files in folder C:\test\ except file abc.txt:

```
files { Include C:\\test\\*.txt }
files { Exclude C:\\test\\abc.txt }
```

Each time you add the same section with the same keyword, you add an operation.

For example, to monitor any text file in folder C:\test\ whose name starts with the string “abc”:

```
files { Include C:\\test\\*.txt }
files { Include C:\\test\\abc* }
```

`Exclude` takes precedence over `Include`. For example:

- If a single subrule includes a particular user marketing\jjohns and excludes the same user marketing\jjohns, the signature doesn't trigger even when the user marketing\jjohns performs an action that triggers the signature.
- If a subrule includes all users but excludes the particular user marketing\jjohns, the signature triggers if the user isn't marketing\jjohns.
- If a subrule includes user marketing* but excludes marketing\jjohns, the signature triggers only when the user is marketing\anyone, unless the user is marketing\jjohns, in which case it doesn't trigger.

## Sections that are common to all class types

Use these sections when defining rules of all class types.

All section names are case sensitive. Section values are not case sensitive.

For sections that apply to a specific class type only, see the section lists for that class type.

| Section | Value | Description | Required? |
|---|---|---|---|
| user_name | {Include/Exclude user's name or system account} | Specifies the users that rule applies to. Specify particular users or all users.<br>• **Local users**: machine name/local user name<br>• **Domain users**: domain name/domain user name<br>• **Local system**: Local/System<br>Some remotely initiated actions don't report the ID of the remote user, but use the local service and its user context instead. You must plan accordingly when developing rules.<br>When a process occurs in the context of a Null Session, the user and domain are "Anonymous".<br>If a rule applies to all users, use the ***** wildcard. | Yes |
| Executable | {Include/Exclude file path name, fingerprint, signer, or description} | Specifies the executables that the rule applies to.<br>Specify each executable inside brackets using:<br>• -path — File path name<br>• -hash — MD5 hash<br>• -sdn — Signer<br>• -desc — Description<br>Each section can have multiple brackets and, inside the brackets, one or more options.<br>The `-path`, `-sdn`, and `-desc` values are strings and must be Tcl-escaped if they contain spaces or other Tcl-reserved characters. The `-hash` value is a 32-character hexbin string.<br>For example:<br>`Executable { Include -path "C:\\Program Files (x86)\\McAfee Endpoint Security\\ Threat Prevention\\mfetp.exe" -sdn "CN=\"McAfee, Inc.\", OU=Engineering, O=\"McAfee, Inc.\", L=Santa Clara, ST=California, C=US" -desc "on-access scanner service" }`<br>If a rule applies to all executables, use the ***** wildcard. | Yes |
| directives | operation type | Specifies the class-dependent operation types.<br>For the operation type, see the directives in each class type description. | Yes |
| dependencies | {Include/Exclude “ID of a rule”} | Defines dependencies between rules and prevents triggering dependent rules.<br>Add the dependencies section to prevent a more general rule from being triggering with a more specific rule. For example, use ID 428 for Buffer Overflow signatures. | No |
| attributes | -no_log | | No |
| | -not_auditable | Generates no exceptions for the signature when Adaptive mode is enabled. | |

| Section | Value | Description | Required? |
|---|---|---|---|
| | -no_trusted_apps | Specifies that the trusted application list doesn't apply to this signature. | |
| | -inactive | Disables the signature. | |

**See also**

*Buffer Overflow class type* on page 41
*Illegal API Use class type* on page 42
*Services class type* on page 43

# Class types

**Contents**



## Buffer Overflow class type

The `Buffer Overflow` class type prevents buffer overflow exploits for applications in the application protection list.

| Section | Value | Notes |
|---|---|---|
| user_name | | |
| Executable | | |
| dependencies | 428 | Specifies Signature 428, Generic Buffer Overflow, a generic buffer overflow rule. (Optional)<br>We recommend including section "dependencies 428" to avoid triggering the generic signature. |
| caller module | Path to a module (for example, a DLL) loaded by an executable that calls and causes a buffer overflow | |
| directives | bo:stack | Examines the memory location that is executing and detects if it is running from writable memory that is part of the current thread's stack. |
| | bo:heap | Examines the memory location that is executing and detects if it is running from writable memory that is part of a heap. |
| | bo:writeable_memory | Examines the memory location that is executing and detects if it is running from writable memory that is not part of the current thread's stack or a heap. |
| | bo:invalid_call | Checks that an API is called from a proper call instruction. |
| | bo:target_bytes | A hexadecimal string representing 32 bytes of instructions that can be used to create a targeted exception for a false positive without disabling buffer overflow for the entire process. |
| | bo:call_not_found | Checks that the code sequence before the return address isn't a call. |

| Section | Value | Notes |
|---|---|---|
| | bo:call_return_unreadable | Checks that the return address isn't readable memory. |
| | bo:call_different_target_address | Checks that the call target doesn't match the hooked target. |
| | bo:call_return_to_api | Checks that the return address is an API entry point. |

**See also**

*Sections that are common to all class types* on page 39

## Illegal API Use class type

The `Illegal API Use` class type prevents illegal use of the Exploit Prevention API.

| Section | Value | Notes |
|---|---|---|
| user_name | | |
| Executable | | |
| vulnerability_name | Name of the vulnerability | |
| detailed_event_info | One or more CLSIDs. | This value is a 128-bit number that represents a unique ID for a software component, such as:<br>`"{FAC7A6FB-0127-4F06-9892-8D2FC56E3F76}"` |
| directives | illegal_api_use:bad_parameter | |
| | illegal_api_use:invalid_call | |

Use this class to create a custom killbit signature. The killbit is a security feature in web browsers and other applications that use ActiveX. A killbit specifies the object class identifier (CLSID) for ActiveX software controls that are identified as security vulnerability threats. Applications that use ActiveX don't load specified ActiveX software with a corresponding killbit in place.

The primary purpose of a killbit is to close security holes. Killbit updates are typically deployed to Microsoft Windows operating systems using Windows security updates.

Here is an example of a killbit signature:

```
Rule {
tag "Sample4"
Class Illegal_API_Use
Id 4001
level 4
Executable { Include "*"}
user_name { Include "*"}
vulnerability_name {Include "Vulnerable ActiveX Control Loading ?"}
detailed_event_info { Include
"0002E533-0000-0000-C000-000000000046"\"0002E511-0000-0000-C000-000000000046"}
directives files:illegal_api_use:bad_parameter illegal_api_use:invalid_call
attributes -not_auditable
}
```

**See also**

*Sections that are common to all class types* on page 39

## Services class type

The `Services` class type protects Windows Services operations.

| Section | Values | Notes |
|---|---|---|
| user_name | | |
| Executable | | |
| service | Name of the service to protect. | (Required)<br>The name of the service is in the corresponding registry key under HKLM_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\. |
| display_names | Display name of the service. | Required.<br>This name appears in the Services manager and in the registry value HKLM_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<name-of-service>\ |
| directives | services:delete | Deletes a service. |
| | services:create | Creates a service. |
| | services:start | Starts a service. |
| | services:stop | Stops a service. |
| | services:pause | Pauses a service. |
| | services:continue | Continues a service after a pause. |
| | services:startup | Changes the startup mode of a service. |
| | services:profile_enable | Enables a hardware profile. |
| | services:profile_disable | Disables a hardware profile. |
| | services:logon | Changes the logon information of a service. |

### Example

The following rule prevents deactivation of the Alerter service.

```
Rule {
service { Include "Alerter" }
application { Include "*"}
user_name { Include "*" }
directives service:stop
}
```

| Section | Description |
|---|---|
| service { Include “Alerter” } | Indicates that the rule applies to the service with name "Alerter”.<br>If the rule applies to multiple services, add them in this section in different lines. |
| application { Include “*”} | Indicates that this rule is valid for all processes.<br>To limit the rule to specific processes, list the pathname to each process. |
| user_name { Include “*” } | Indicates that this rule is valid for all users (or more precisely, the security context in which a process runs).<br>To limit the rule to specific user contexts, list them using the form Local/user or Domain/user. |
| directives service:stop | Indicates that this rule applies to deactivation of a service. |

**See also**

*Sections that are common to all class types* on page 39