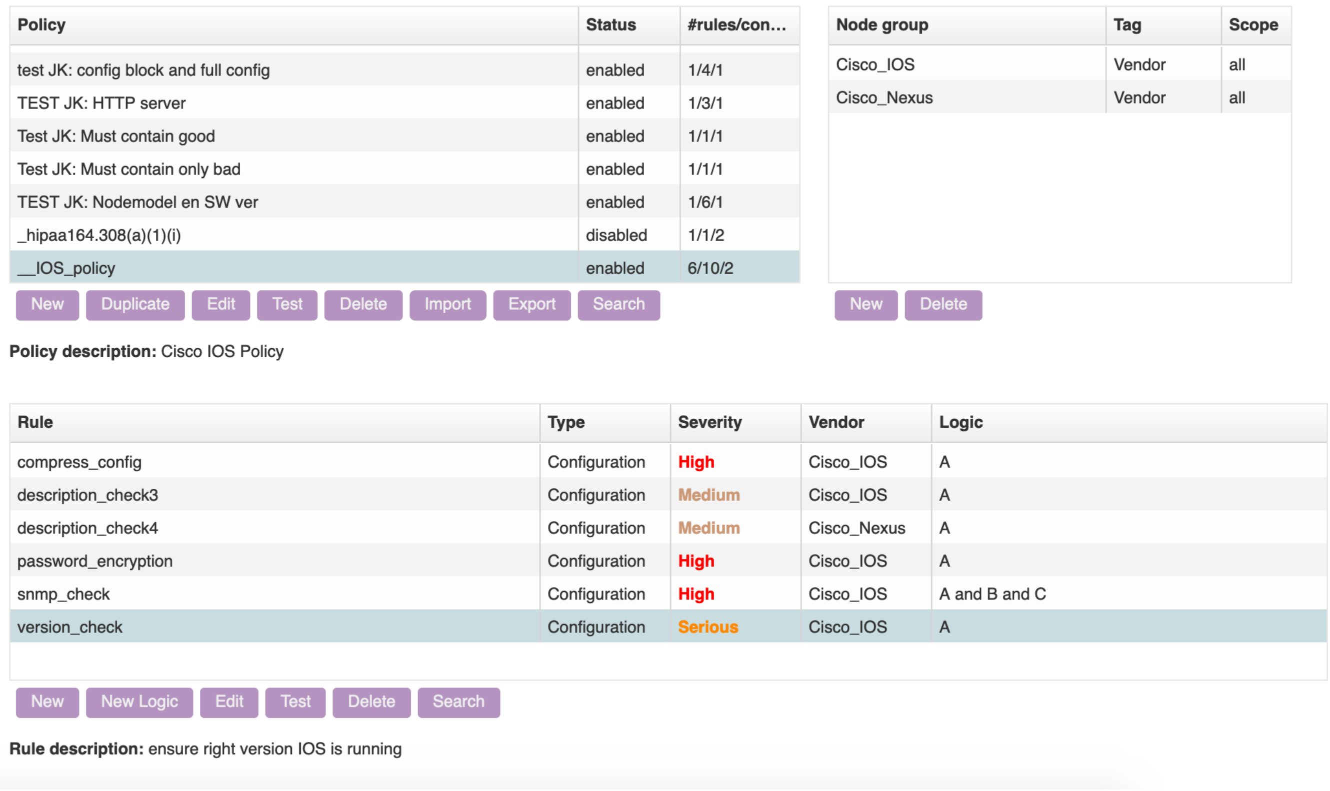
Table of Contents

Compliance Policies

Policies

Compliance policies

The compliance policies form can be found in the Operate→Compliance form. This is the form where all compliance policies, rules and conditions can be created and maintained.



Policy	Status	#rules/con...
test JK: config block and full config	enabled	1/4/1
TEST JK: HTTP server	enabled	1/3/1
Test JK: Must contain good	enabled	1/1/1
Test JK: Must contain only bad	enabled	1/1/1
TEST JK: Nodemodel en SW ver	enabled	1/6/1
_hipaa164.308(a)(1)(i)	disabled	1/1/2
__IOS_policy	enabled	6/10/2

Node group	Tag	Scope
Cisco_IOS	Vendor	all
Cisco_Nexus	Vendor	all

Policy description: Cisco IOS Policy

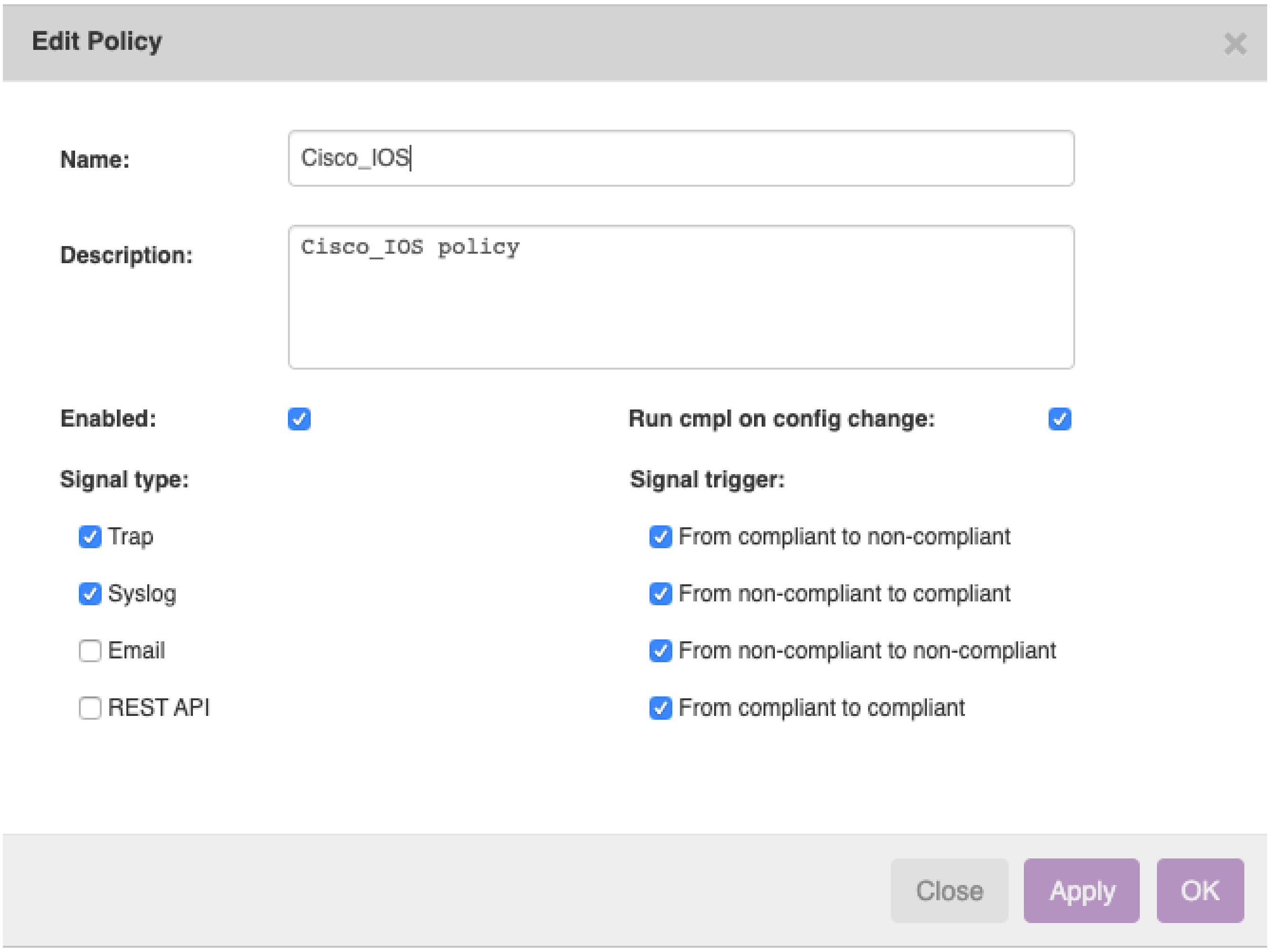
Rule	Type	Severity	Vendor	Logic
compress_config	Configuration	High	Cisco_IOS	A
description_check3	Configuration	Medium	Cisco_IOS	A
description_check4	Configuration	Medium	Cisco_Nexus	A
password_encryption	Configuration	High	Cisco_IOS	A
snmp_check	Configuration	High	Cisco_IOS	A and B and C
version_check	Configuration	Serious	Cisco_IOS	A

Rule description: ensure right version IOS is running

The top-left grid lists all policies, their descriptions and whether they are enabled. The last column gives an overview of the policy's rules, conditions, and the number of node groups assigned to it.

The new-button creates a new policy. Edit and delete can respectively edit and delete the policy's data. Duplicate duplicates a policy, including its rules and conditions (including its node groups). Import imports either a HPNA or netYCE policy export file, and export exports the policy. You can import and export multiple policies at the same time. If an import includes a policy with a name that already exists, the question whether or not to overwrite it is provided. With search you can search through the policies.

The top-right grid lists all those node groups and the new-button allows you to assign node groups to this policy. The delete-button allows you to remove node groups from this policy.



Edit Policy

Name: Cisco_IOS

Description: Cisco_IOS policy

Enabled: ☑

Run cmpl on config change: ☑

Signal type:

- ☑ Trap
- ☑ Syslog
- □ Email
- □ REST API

Signal trigger:

- ☑ From compliant to non-compliant
- ☑ From non-compliant to compliant
- ☑ From non-compliant to non-compliant
- ☑ From compliant to compliant

A policy's name is what you get to see in the reports. The description is purely what you get to see in the edit form and below the policies grid for extra clarification.

A policy that is not enabled will not be checked for compliance. If the “Run cmpl on config change”-checkbox is checked, whenever the nccmd daemon detects a change in config (through syslog), it checks for compliance. This is on by default, but recommended to be turned off for nodes whose configuration changes often. Policy schedules are an alternative in these cases.

Signalling

A compliance check can have one of these results:

- A compliant node becomes non compliant
- A non-compliant node becomes compliant
- A non-compliant node stays non-compliant
- A compliant node stays compliant

These settings allow for signals that just notify on compliance state changes or each time a compliance validation is made.

There are four ways we can send signals:

- A trap message
- A syslog message

- An email message
- A REST-API 'post' call using Json

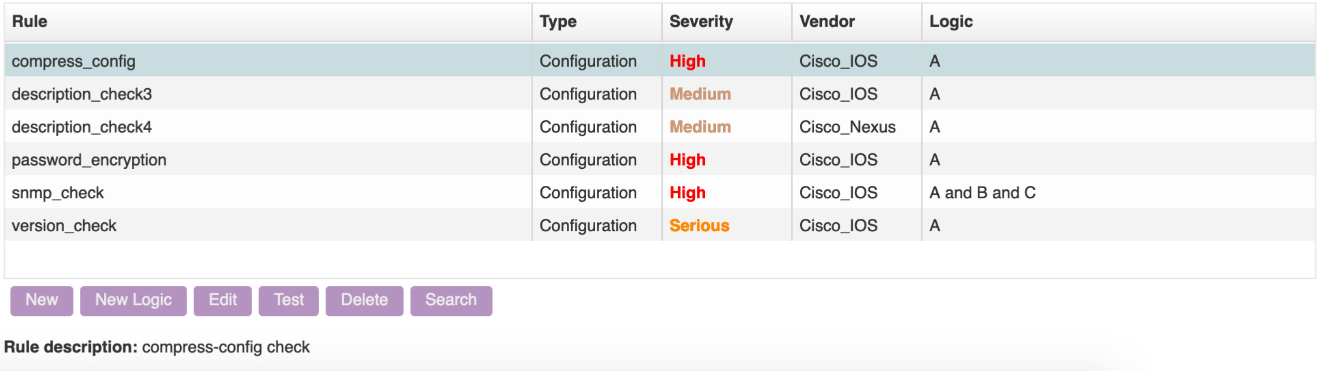
These settings are available in the front-end when editing a policy using four separate check-boxes allowing for multiple signal types for the same event.

These signal messages need setting-up for the environment and mostly requires the definition of the remote targets and priority. Configure these in the file **`etc/signal_cmpl.conf`**.

Details on the configuration of these signalling-types is available in the Reference guide on Compliance signalling.

Please note signals at the moment are not combined. So be careful when for example creating email signals: if a lot of triggers happen at the same time, a lot of emails will be sent and you could potentially set off a mini ddos attack.

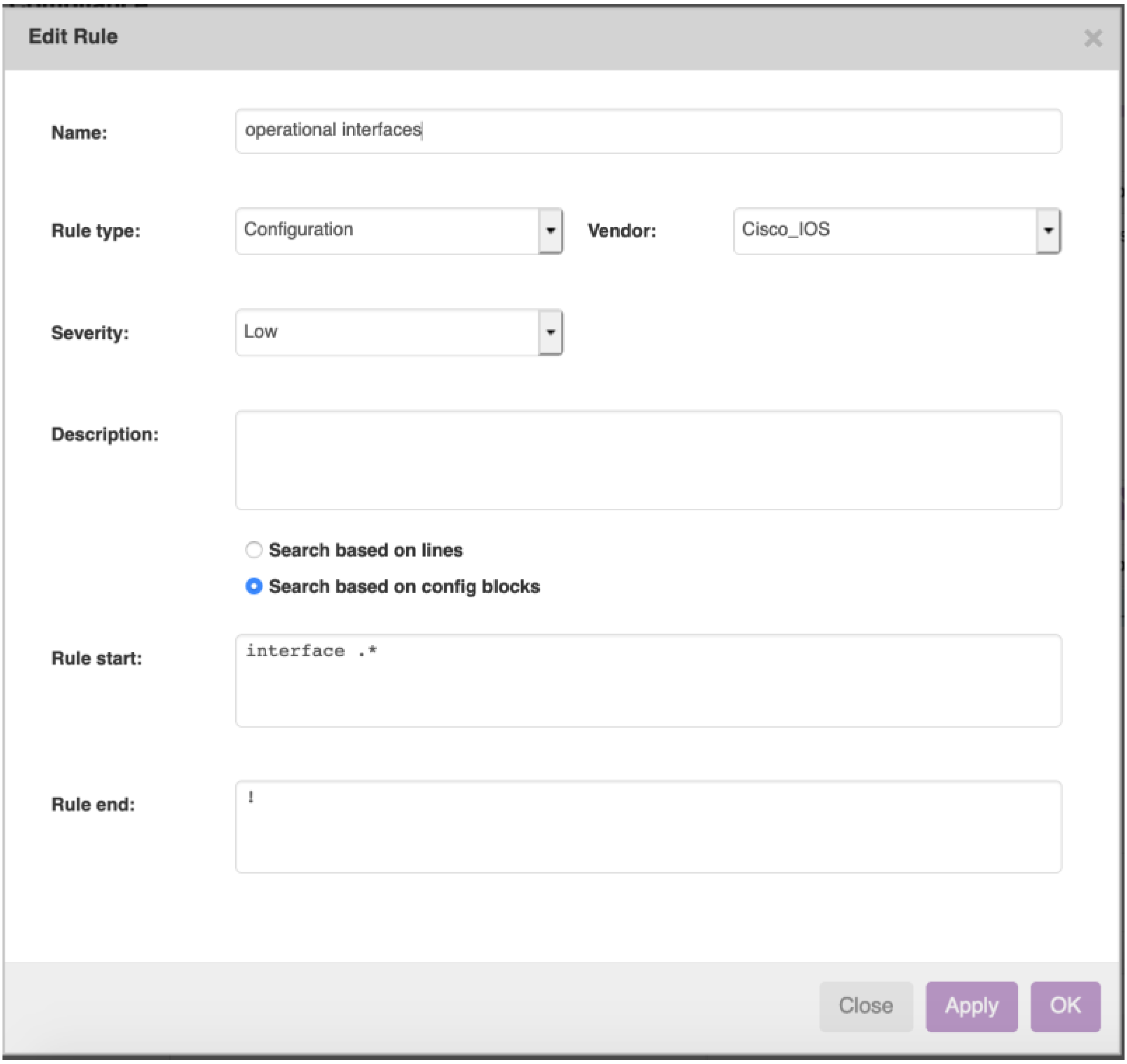
Rules



Rule	Type	Severity	Vendor	Logic
compress_config	Configuration	High	Cisco_IOS	A
description_check3	Configuration	Medium	Cisco_IOS	A
description_check4	Configuration	Medium	Cisco_Nexus	A
password_encryption	Configuration	High	Cisco_IOS	A
snmp_check	Configuration	High	Cisco_IOS	A and B and C
version_check	Configuration	Serious	Cisco_IOS	A

Rule description: compress-config check

The rules grid shows the currently selected policy's rules. The buttons here are also pretty straightforward, however the test-button tests if the logic of the rule's conditions is valid or not for our parser. If not, this rule cannot be used on compliance checks.



Name: operational interfaces

Rule type: Configuration

Vendor: Cisco_IOS

Severity: Low

Description:

○ Search based on lines

● Search based on config blocks

Rule start: interface .*

Rule end: !

There are two ways the nccmd daemon can parse configs for the content it needs: **Search based on lines** and **Search based on config blocks**.

When you search configs based on lines, the daemon will look for lines in the config that match its Rule start. When it finds one, it will mark the block starting with this line, until it encounters either a line that matches Rule end or the end of the config. If multiple blocks are found this way they will all be checked for compliance. Regular expressions are also supported.

Sometimes though, it is hard to know for sure how a block will end. Or in cases like Juniper, where blocks within blocks all contain the same characters. For this there is also the option to search based on blocks. To explain config blocks: a config consists out of a number of text blocks. Think of a block as follows:

```
block head
 block body
 block body
 block body
```

```
!
```

Or:

```
block a
block b
block c
block d
```

Or even hierarchical:

```
block head
 block body
 block body
 block body
 subblock head
  subblock body
  subblock body
  subblock body
 block body
!
```

Rule start looks at these block heads, and returns all blocks that match.

Some vendors, Juniper for instance, have complex hierarchical trees within their blocks. If you want to match sub blocks, you can put down the first lines all the parent blocks, followed by the first line of the sub block you want to match. For example a **Rule start** of:

```
block head
subblock head
```

For each parent block, you need one line. They need to be in order, and can also contain regular expressions.

If no Rule start is provided, the whole config is taken. This goes both for when you search based on blocks or on lines.

If no block is matched, the rule is automatically compliant.

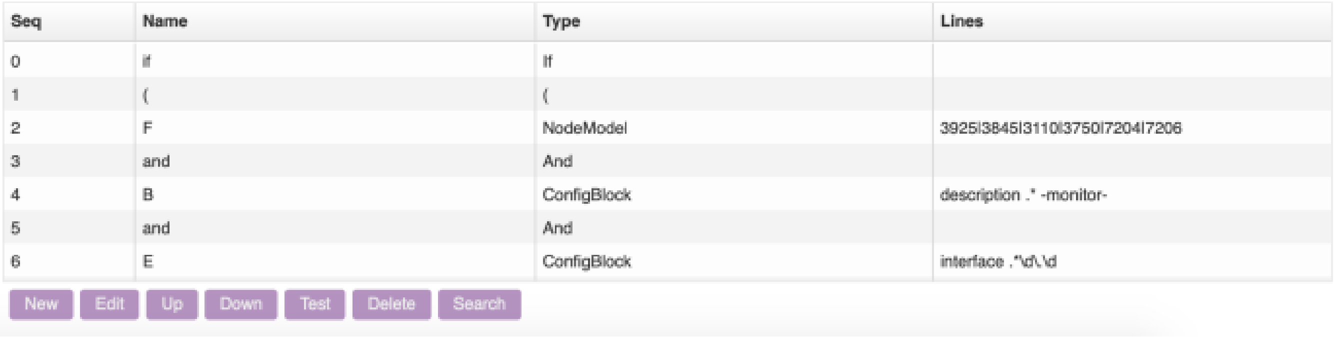
Configuration rules will match against a config. Command rules will execute a command on the node and match against its results. Multiconfig will compare the configs of multiple nodes with each other. For more information on multiconfig rules, refer to the Multiconfig compliance reference]

Sometimes you will want to run compliance on a string of text that does not fit well inside these blocks. In that case you can check the option to parse by lines. In that case the config is parsed, and all lines inbetween and including the rule start and the rule end will match. Multiple blocks that match these criterias will all match. If no rule end is defined, it takes the whole config starting from the first line that matches the rule start. If no rule start or end is defined, it still takes the whole config.

Banners

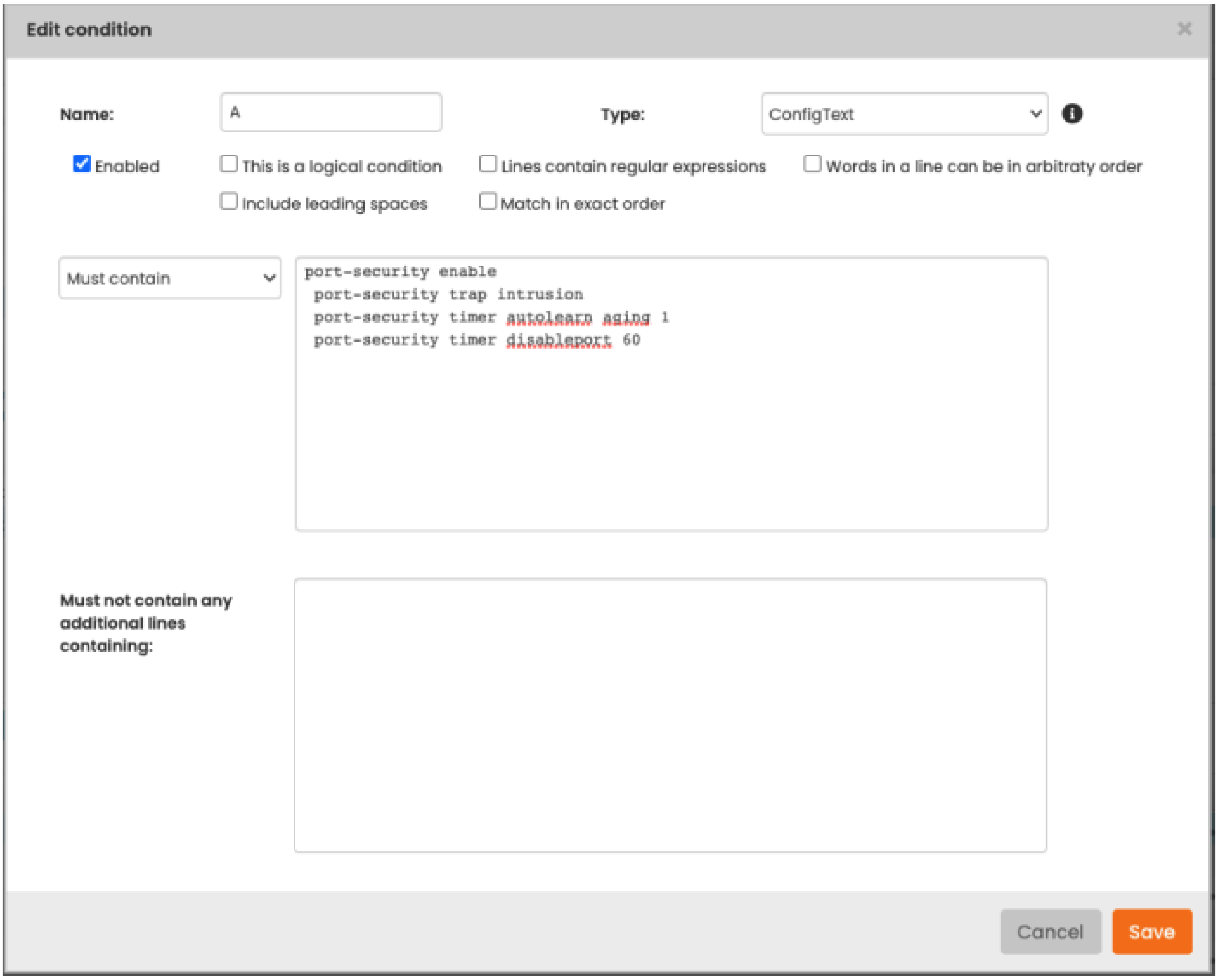
Banners are a special case. Because their indentation is often completely different from the rest of the structure of the config, we combine it into one line. We do this for the vendors Cisco IOS, Cisco Nexus, Cisco XE and Mrotek. To match them, you need to match for blocks, and match with a Rule start of “banner exec .*” No need for a Rule end.

Conditions



Seq	Name	Type	Lines
0	if	If	
1	(	(	
2	F	NodeModel	3925|3845|3110|3750|7204|7206
3	and	And	
4	B	ConfigBlock	description .* -monitor-
5	and	And	
6	E	ConfigBlock	interface .*\d\.\d

The bottom grid is for the rule's conditions. New creates, edit edits, delete deletes, search searches. And since conditions have a sequence, you can move them with the up and down buttons. Note that the edit form is different, depending on the condition's type and the rule's type: a logic condition does not need any lines, so the fields for that are not shown.



Edit condition

Name: A

Type: ConfigText

☑ Enabled ☐ This is a logical condition ☐ Lines contain regular expressions ☐ Words in a line can be in arbitraty order

☐ Include leading spaces ☐ Match in exact order

Must contain

```
port-security enable
 port-security trap intrusion
 port-security timer autolearn aging 1
 port-security timer disableport 60
```

Must not contain any additional lines containing:

Conditions take a piece of text, defined by the rule and match it against a number of lines, defined in the first big text area. They either need to match part of each line, a line fully, need to not be present or need to match the block exclusively with no other lines present. They can or cannot contain regex.

Logical conditions are there to create logic around the conditions with if-, and- and or-statements. A condition also can be disabled. In that case it won't be included in the rule's logic. You can see which conditions are disabled in the condition grid when they are greyed out. Names can be anything, but a good convention is to give them simple, single-letter names.

You can select different condition types, and this determines what text will be used for the condition:

- **ConfigBlock:** The config block as dictated by the Rule_start and Rule_end of the condition's rule
- **ConfigText:** The whole config
- **NodeModel:** The node's node model, as retrieved from cli command output
- **SoftwareVersion:** The node's software version, as retrieved from cli command output
- **Hostname:** The node's hostname

This string of text will be compared to the condition's lines. There are four ways to compare:

- **Must contain:** For each condition line, the text needs to contain a line that matches it. Lines don't have to match exactly, as long as the condition's line is part of it.
- **Must not contain:** There should be no instance of any condition line in the text.

- **Must contain lines:** For each condition line, there needs to be a line in the text that matches it exactly
- **Must contain exactly:** The text should match each condition line, and there should be no other lines present

Conditions will by default be parsed without leading spaces on each line. If you want to be exact in the amount of leading spaces that are required, you can check the “Include leading spaces” checkbox.

Additionally you can specify if the lines should contain regex or not. If the field “Must not contain any additional lines containing” is filled, the text will also be checked to see if there aren't any other additional matches, beyond the lines that have already matched. Note that this can match multiple lines, which will all be reported, up to 20 lines.

Note that for a rule that uses the whole config, the options “Must contain exactly” and “Match in exact order” don't make a lot of sense, since you're trying to match some very specific line orders against a whole configuration which can contain any random lines. Therefore they are omitted from the form in these cases.

Sometimes, you encounter lines in a config like this:

```
ip-addresses { 192.168.0.1 192.168.0.8 172.0.0.1 194.175.16.22 111.24.35.128 192.168.0.2 }
```

Where the ip addresses don't have a set order. F5 is an example of a vendor that can do this. For those cases, you can check the “Words can be in arbitrary order”-checkbox. This will evaluate a line against a condition, and it will not care about the order the different words (words here is defined by anything separated by a space), as long as they're all there. Must contain conditions will also allow additional words in a line. Must contain lines conditions will only match lines that contain no other words.

Logic

Conditions can be combined with each other using logic. In this way you can have conditionals within If, Else and Endif statements. In this case, a node will be checked against the conditions in the Then block if the conditions in the If-block are compliant, and if not it will be checked against all conditions in the Else block. It also supports And (both conditions have to be compliant) and Or (A minimum of one of the conditions has to be compliant). Conditions are evaluated sequentially, but if you want to force orders, then you can group them together with parentheses: all logic within the parentheses will be evaluated before anything outside of them.

Condition Variables

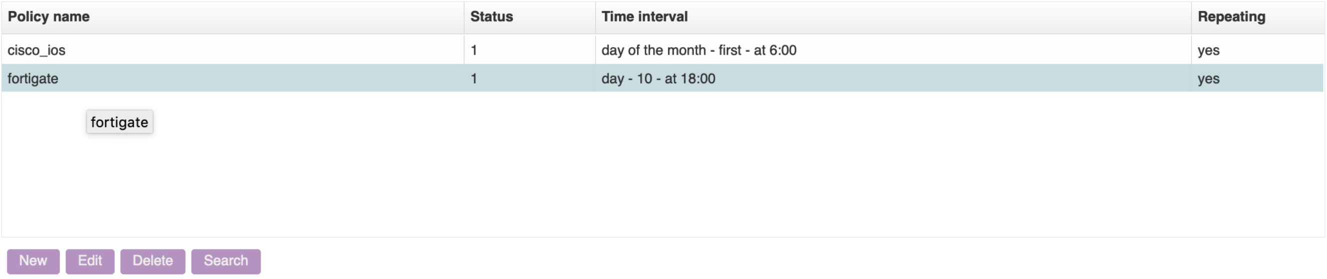
You can also parse relations and variables. For example, a line:

```
hostname <node>
```

Will parse <node> as the node's hostname. Relations will also be parsed. For more information, refer to the Relations reference.

Policy Schedules

Policy schedules are a way to control the time at which nodes will be checked for compliance. This is especially useful for big machines whose config changes very often and checking after every config change would flood the servers.



Policy name	Status	Time interval	Repeating
cisco_ios	1	day of the month - first - at 6:00	yes
fortigate	1	day - 10 - at 18:00	yes

A schedule consists out of a policy, and a time interval. A time interval determines the next interval for this schedule. Whenever a node is scheduled, it calculates the first date in the future that matches this interval. There are four different types:

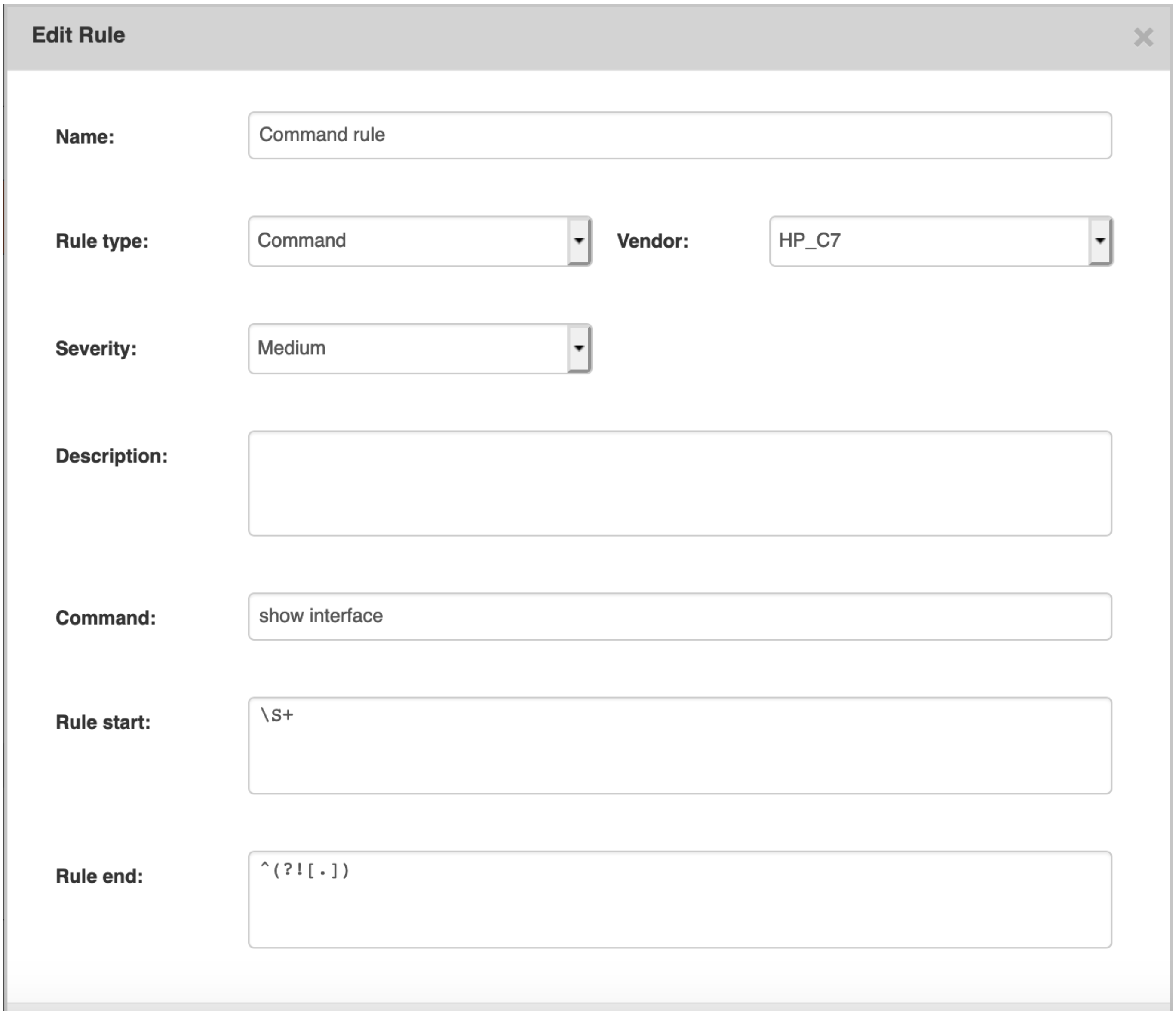
- **Hour of day**: This schedule will be for every day, at certain hours of the day. You can specify multiple hours (separated by a comma, for example 16,20), and even ranges of hours (separated by a dash, for example 4-6). The first and last checkbox are shortcuts for 0:00 and 23:00 respectively.
- **Every x days**: This schedule will take place every x days. Only one number is allowed, and you can set the schedule time for up to the whole hour.
- **Day of week**: This will schedule based on the day of the week, denoted by its two-letter abbreviation: mo, tu, we, th, fr, sa and su. Commas to denote multiple days and dashes to denote ranges are also supported. First and last are shortcuts for monday and sunday respectively. You can also set the schedule time for up to the whole hour.
- **Day of month**: This will schedule based on the day of the month, from 1 to 31, where non-existing days like February 30th will be ignored. Commas to denote multiple days and dashes to denote ranges are also supported. First is a shortcut for the first day of the month, and last for the last. You can also set the schedule time for up to the whole hour.

A schedule can be enabled or disabled. Only enabled schedules will be used when calculating the next schedule time. If none can be found, the node won't be scheduled and have to wait for another trigger for a compliance check (for example the API, or after a config change if it has been set so).

A schedule can be repeating. Repeating schedules will be scheduled over and over again: when a check finishes, the the next date matching the schedule's time interval will be set as the node's next schedule time. If the schedule is not repeating, it will be disabled after the schedule time for its nodes has been set.

Command rules

Command rules are a rule type that checks the reply of a console command on a node against a number of conditions.



Edit Rule

Name: Command rule

Rule type: Command

Vendor: HP_C7

Severity: Medium

Description:

Command: show interface

Rule start: \S+

Rule end: ^(?![.])

The Command field contains the command that will be run on the node. Like all rules, these are limited per vendor type, so in this example, this rule will only be checked for HP C7 nodes.

Rule start and Rule end can be used to parse part of the reply. Since commands are easier to parse than entire config, these values can be plain text, or contain regular expressions. When using both a rule start and rule end, multiple blocks of text can be matched. Compliance will then be run on all blocks separately, like with configuration rules. Note that whole lines will be matched, so lines that match, but contain some more text will not match.

For example, a Rule start of:

```
Interface
```

Will not match:

```
Interface interface_1/1/1
```

However:

```
Interface \S+
```

Will.

Another useful line is:

```
^(?![.])
```

This matches an empty line.

Command results are fetched along with the NCCM. The nccmd daemon first retrieves the full config of a node, and then evaluates if there are any command rules associated to it (so if it's in a node group that is coupled to a policy that contains command rules). If there are, it executes these, and stores the results in the database. Unlike for the NCCM, there is no history kept for these records.

When you save a command rule, the nodes that are linked to it are automatically scheduled for a nccm poll, followed by compliance check. It will take the nccmd daemon a few minutes to process this, but the followup process is automated. If the daemon manages to do a policy check on a command rule that hasn't been run on the node yet, it'll return compliant until the request completes.

Multiconfig compliance

> NOTE: This functionality was created for the F5 BIGIP vendor module and has been extended to function for other vendor modules as well. The F5 is still a special case however. For these BIGIP configurations some extensive pre-parsing takes place that will remove from the configuration tree all unreferenced segments before comparisons are made. This greatly improves the compliance results given that these configurations can be over 400,000 lines.

The Multiconfig compliance is intended for a group of two to four nodes, whose configurations, or parts of their configuration need to be equal at all times.

The Multiconfig uses it own Rule types as they will compare the node's new config to each of the current configs in its node group. A number of restrictions apply:

- The node group cannot contain more than four nodes. The rule will be automatically non-compliant when more than four nodes are detected in the node group.
- The node group must contain nodes of the same vendor-type
- You can provide multiple node groups for a policy, provided that the total number of nodes does not exceed four. All nodes will have their configs checked against each other
- It is not advisable for the rule's policy to be run at config change, because this would mean a non-compliance every time one of the nodes changes its config. Instead, we recommend Multiconfig compliance to be run periodically, for example daily after office hours.

The Rule type has to be set to “multi-config”, and its Vendor type to the vendor required. Rules of this type do not have any conditions. If its Rule_start and Rule_end are set, it will only compare the config blocks that match, otherwise it will take the complete configuration.

Like with regular rules, you can also select part of the configs to be compared to each other. You can use Rule start and Rule end to find either config blocks that start and end with the lines respectively (regex supported), or search the config for the lines in between and including the Rule start and Rule end lines that you supplied, depending on which selection option you checked in your rule (Search based on lines or Search based on config blocks).

If a rule start is specified, the daemon looks for blocks to match these conditions. Any it finds will be compared to the other configs in the policy's node group. If it can't find any, it's automatically compliant.

In the case of a F5 BIGIP node, all orphans (blocks that aren't referenced anywhere) will be logged in the rule's report, to serve as a guide for the operator to help clean them up.

NOTE that a number of blocks are meant to be orphaned. For now these are hard-coded, however if desired we can make this user-generated.

Nccmd daemon behavior

This section will give a rough overview of how the nccmd daemon acts. It is a daemon that is responsible for both running the nccm and the compliance.

It wakes up every five minutes. It then checks if there are any nodes that need to have their nccm checked, and grabs up a sensible number of them. If there are more, these need to wait until the next cycle. A nccm check is necessary if the node in question has sent out a syslog message due to a config change. It can happen because of periodical checks for nccm as dictated by its polling group. Also whenever a node is created or has its name changed, both in the regular yce environment as in the cmdb, it automatically gets scheduled for this as well.

Each node is polled for its configuration and compared with the most recent one in the nccm. In the case of a change they are updated (of course, cyphers and timestamps are filtered out to avoid false positives). If the node has any policies that are set to run compliance upon config changes, these are scheduled for a compliance check for the next cycle of the daemon.

The daemon also collects the number of nodes that need to be checked for compliance. These processes all run concurrently, so it doesn't have to wait for every single compliance check to finish before it can get on with actual compliance. A compliance check can be scheduled after nccm, due to a policy schedule, or when a policy gets modified in any way (to check whether its nodes still conform to the new settings).

Each combination of node to policy has its own compliance check, and therefore its own report. In the reports form these are bundled per policy or per node. There is no history for past compliance checks: all results you see are from the latest checks, and any previous results are thrown away. The only history is that we keep track of the last time a compliance result has changed for a node, and at what time its last policy check occurred. If a node is removed from a policy through its node group, its compliance results are deleted as well.

Also note that when you update a policy, a flag will automatically be set for the daemon to re-evaluate its compliance for all its nodes. This also happens when you edit (or create/delete) its rules and conditions, and change its assigned node groups.