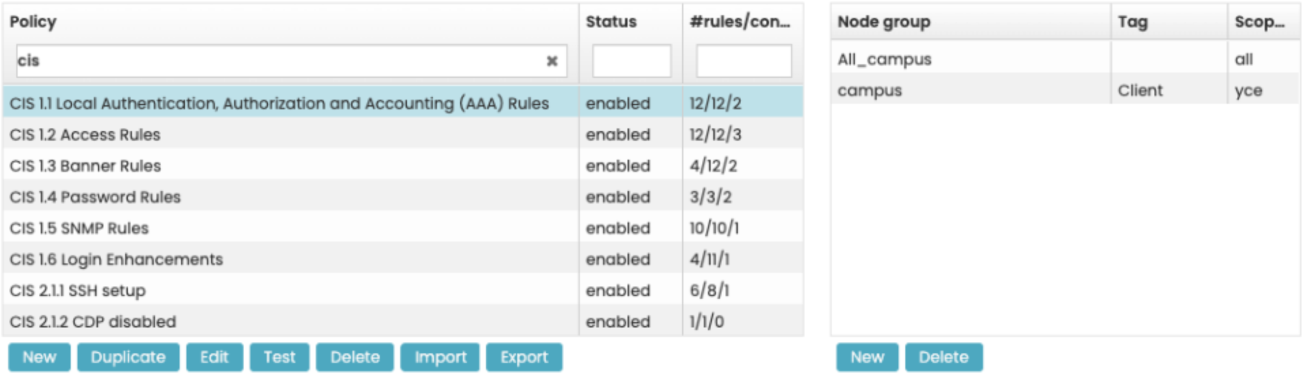
Table of Contents

Policies

In this form you can create, test and manage compliance policies, rules and conditions. A policy is tested on a node, to return either compliant or non compliant. A policy contains various vendor type specific rules, each of these rules contains a string of conditions that together form a logic the rule tests at. Only if all rules for the node's vendor type are compliant, is the policy compliant for this node.

Policies

The policy grid:



Policy	Status	#rules/con...
cis		
CIS 1.1 Local Authentication, Authorization and Accounting (AAA) Rules	enabled	12/12/2
CIS 1.2 Access Rules	enabled	12/12/3
CIS 1.3 Banner Rules	enabled	4/12/2
CIS 1.4 Password Rules	enabled	3/3/2
CIS 1.5 SNMP Rules	enabled	10/10/1
CIS 1.6 Login Enhancements	enabled	4/11/1
CIS 2.1.1 SSH setup	enabled	6/8/1
CIS 2.1.2 CDP disabled	enabled	1/1/0

Node group	Tag	Scop...
All_campus		all
campus	Client	yce

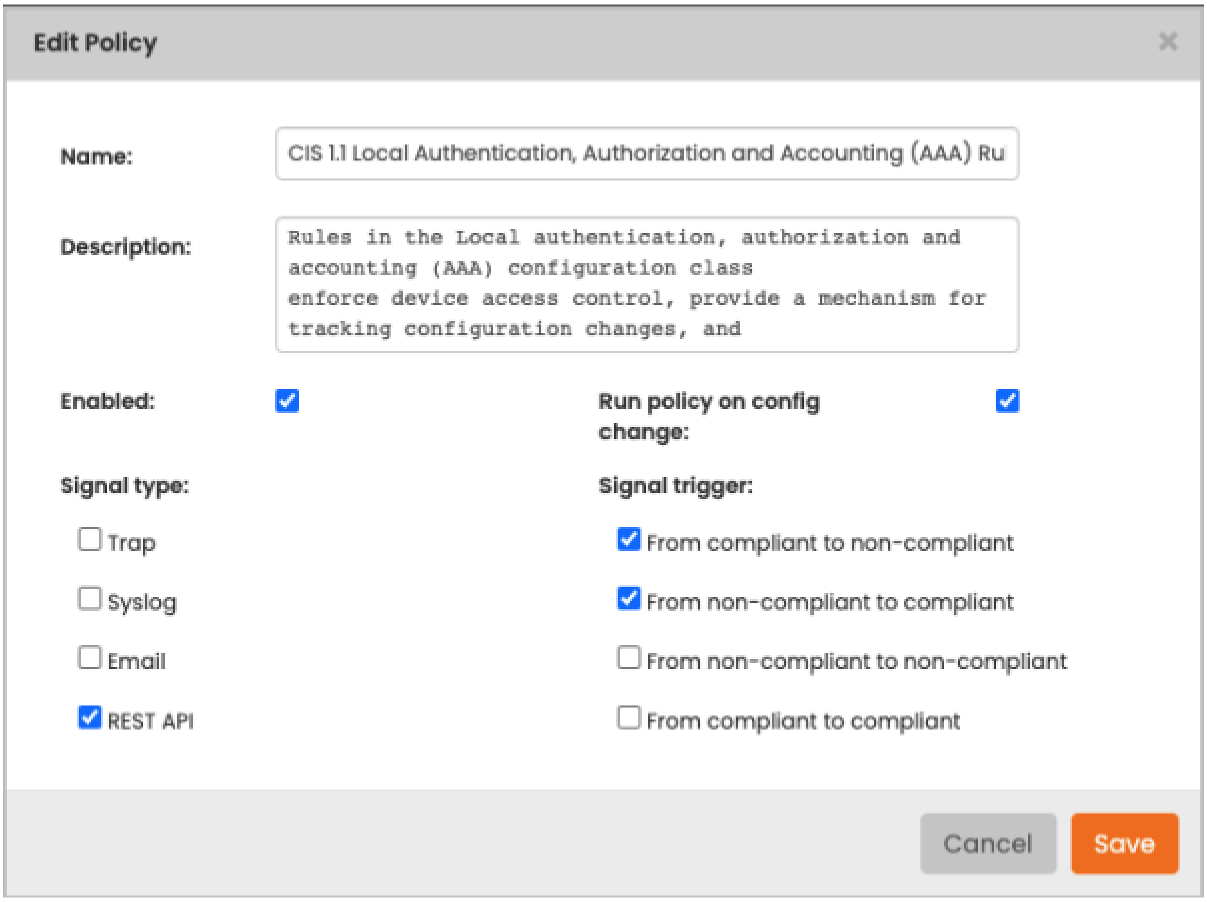
You can create a new policy, edit an existing one and you can duplicate it. The copy will contain all of the policy's rules, conditions and node groups. You can test a policy on a node to see whether the logic works and search through all policies you have. You can delete, export and import them. You can export single or multiple policies. The import function works both with exported files from netYCE and HPNA.

A few caveats about importing from HPNA:

- a rule with a vendor type not supported by NetYCE will not work, a list of supported vendors can be found here
- diagnostics rules are not supported. NetYCE's version of this are command rules, but they will not support a conversion from diagnostic rules.

The grid on the right contains all node groups linked to this policy. All nodes belonging to these node groups will be validated on compliance, unless the policy in question is disabled (you can see this on the status-column of the policy grid). The scope can be either 'cmdb', 'yce' or 'all', meaning where the node groups should be evaluated from. For more details on node groups, see Node Groups.

Edit Policy



Edit Policy

Name: CIS 1.1 Local Authentication, Authorization and Accounting (AAA) Ru

Description:

```
Rules in the Local authentication, authorization and
accounting (AAA) configuration class
enforce device access control, provide a mechanism for
tracking configuration changes, and
```

Enabled: ☑

Run policy 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

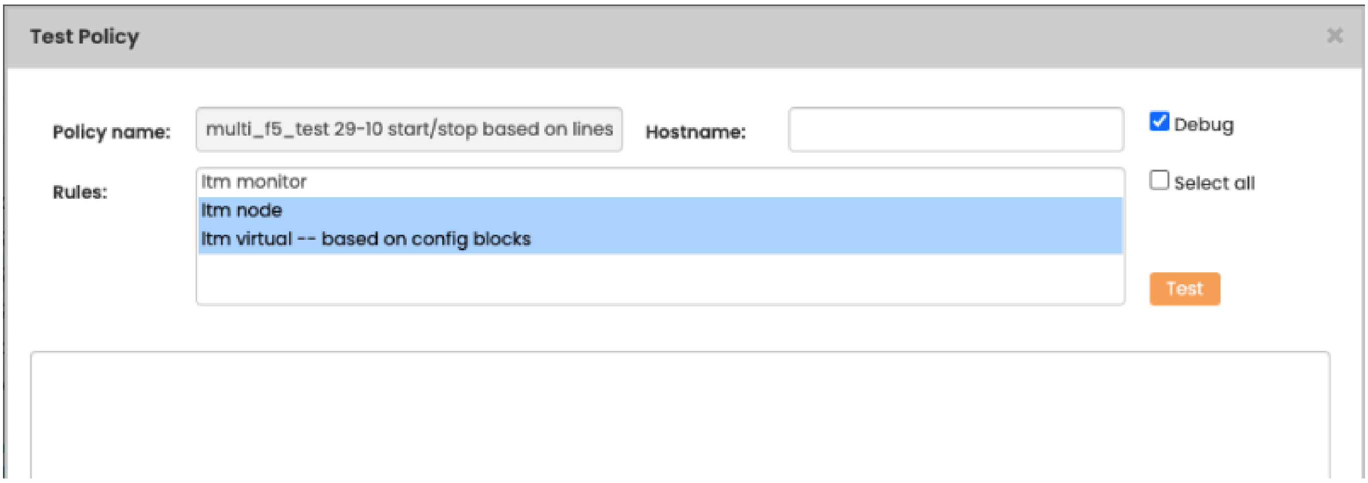
The options (Trap, Syslog, Email and REST API) presented under Signal type represent the available actions to be undertaken upon the compliance result. This can be triggered in four different ways:

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

The most important option is whenever a node changes from compliant to non-compliant.

You can enable or disable a policy here with its checkbox *Enabled*. “Run compliance on config change” means that whenever we detect a config change, the node will be scheduled for a compliance check. This is useful for nodes whose configs change once in a while. For nodes whose config change constantly it is better to use scheduled policies. This is a work in progress and will be released in future versions of netYCE.

Test Policy



Policy name: multi_f5_test 29-10 start/stop based on lines

Hostname:

☑ Debug

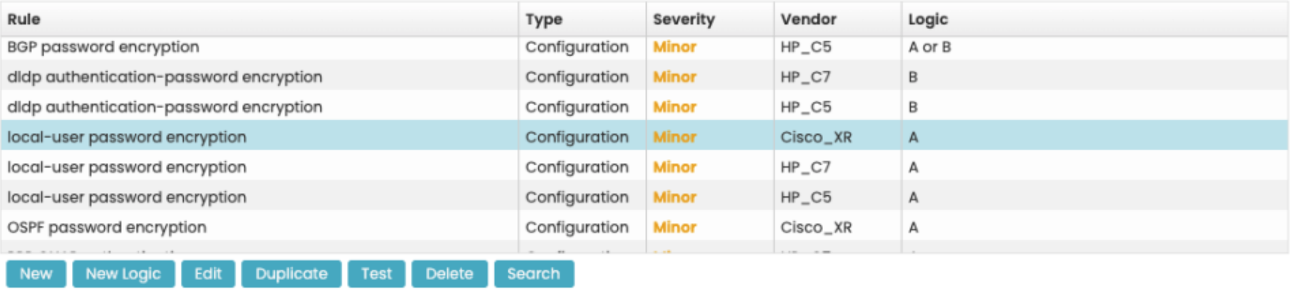
☐ Select all

Rules:
- ltm monitor
- ltm node
- ltm virtual -- based on config blocks

Policies can get quite complex, and testing them is difficult when you're working with live nodes and a daemon that only runs periodically. This form is meant to give you a quick option to test whether your policies do what they're supposed to do. You can select multiple rules (by not selecting any it just takes all of them) belonging to this policy, enter a hostname and they will be evaluated. The debug-check shows more detailed information that might help you in case there is a problem.

An important caveat is this: the form checks the policy on the config of the node that's saved in the NCCM, it will not poll the node whatsoever. This means that for this to work, you do need to have an NCCM config for the node in the database, otherwise this function will not work.

Rules



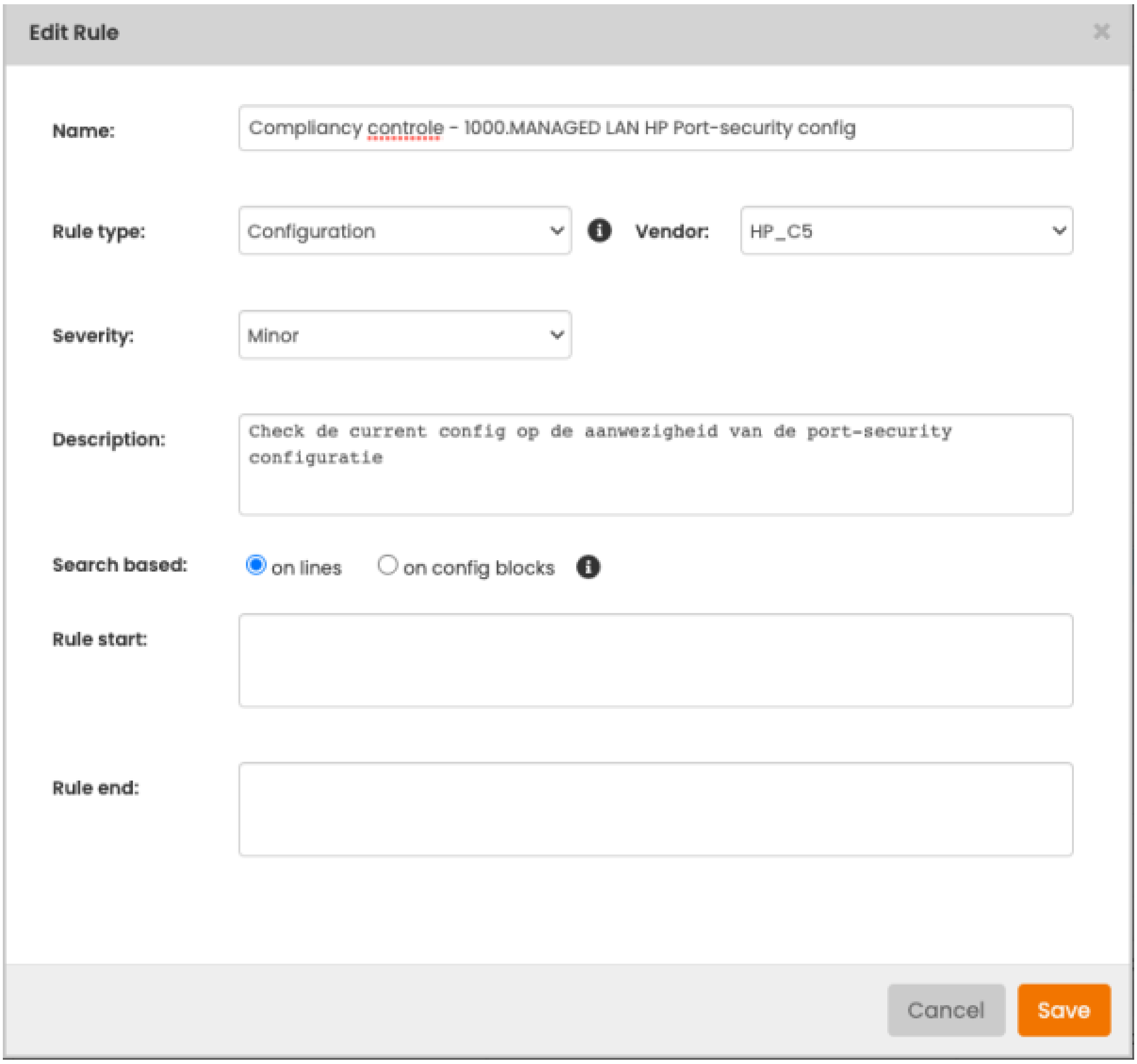
Rule	Type	Severity	Vendor	Logic
BGP password encryption	Configuration	Minor	HP_C5	A or B
dldp authentication-password encryption	Configuration	Minor	HP_C7	B
dldp authentication-password encryption	Configuration	Minor	HP_C5	B
local-user password encryption	Configuration	Minor	Cisco_XR	A
local-user password encryption	Configuration	Minor	HP_C7	A
local-user password encryption	Configuration	Minor	HP_C5	A
OSPF password encryption	Configuration	Minor	Cisco_XR	A

Rules are vendor type bound: only for nodes with that vendor type will they be evaluated. There are three types of rules: Configuration, Command and Multi-config.

- Configuration rules take a config or a part of the config (as signified by Rule_start and Rule_end) and runs a number of conditions on it.
- Command rules take the output of a command on the node, and compares it to a number of conditions. Future plans are also to parse these into variables using command parsing. This feature is not yet available and will be in future netYCE releases.
- Multi-config rules compare the config to the config of one to three other nodes, to see if they are equal. For more information, refer to the Compliance user guide

You can create, edit, duplicate and delete rules and search through them. A rule's conditions combine together with a 'logic'. You can test whether this logic has a valid syntax and create a bunch of conditions at the same time with the 'new logic' button, but more on that below, in the 'Conditions' section.

Edit Rule



Name: Compliancy controle - 1000.MANAGED LAN HP Port-security config

Rule type: Configuration

Vendor: HP_C5

Severity: Minor

Description: Check de current config op de aanwezigheid van de port-security configuratie

Search based: ● on lines ○ on config blocks

Rule start:

Rule end:

In the rule edit form you define the part of the config you will check against. There are two ways to do this: **Search based on lines** and **Search based on config blocks**.

When you search configs based on lines, the compliance will look for lines in the config that match 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 where they 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
```

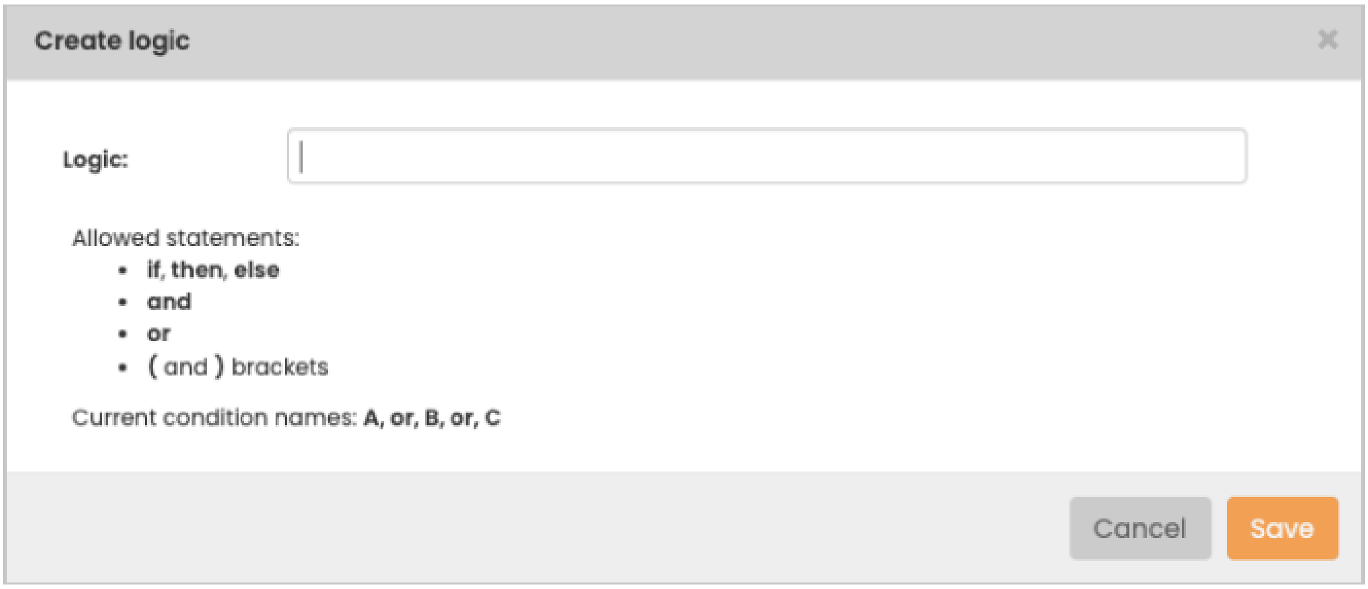
For each parent block, you need one line. They need to be in order, and can also contain regular expressions.

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

Also note that some vendors do not support splitting up the config in blocks. These will be searched by lines by default.

Create Logic

Sometimes, you want to create complex rules that consist out of a lot of conditions, it might be annoying to keep creating new conditions one at a time. The form create logic is a bit of a shortcut for this use case: you can type in a valid string of condition logic and it will automatically create those conditions for you when you submit. For more information on how condition logic works, refer to this section.



Create logic

Logic:

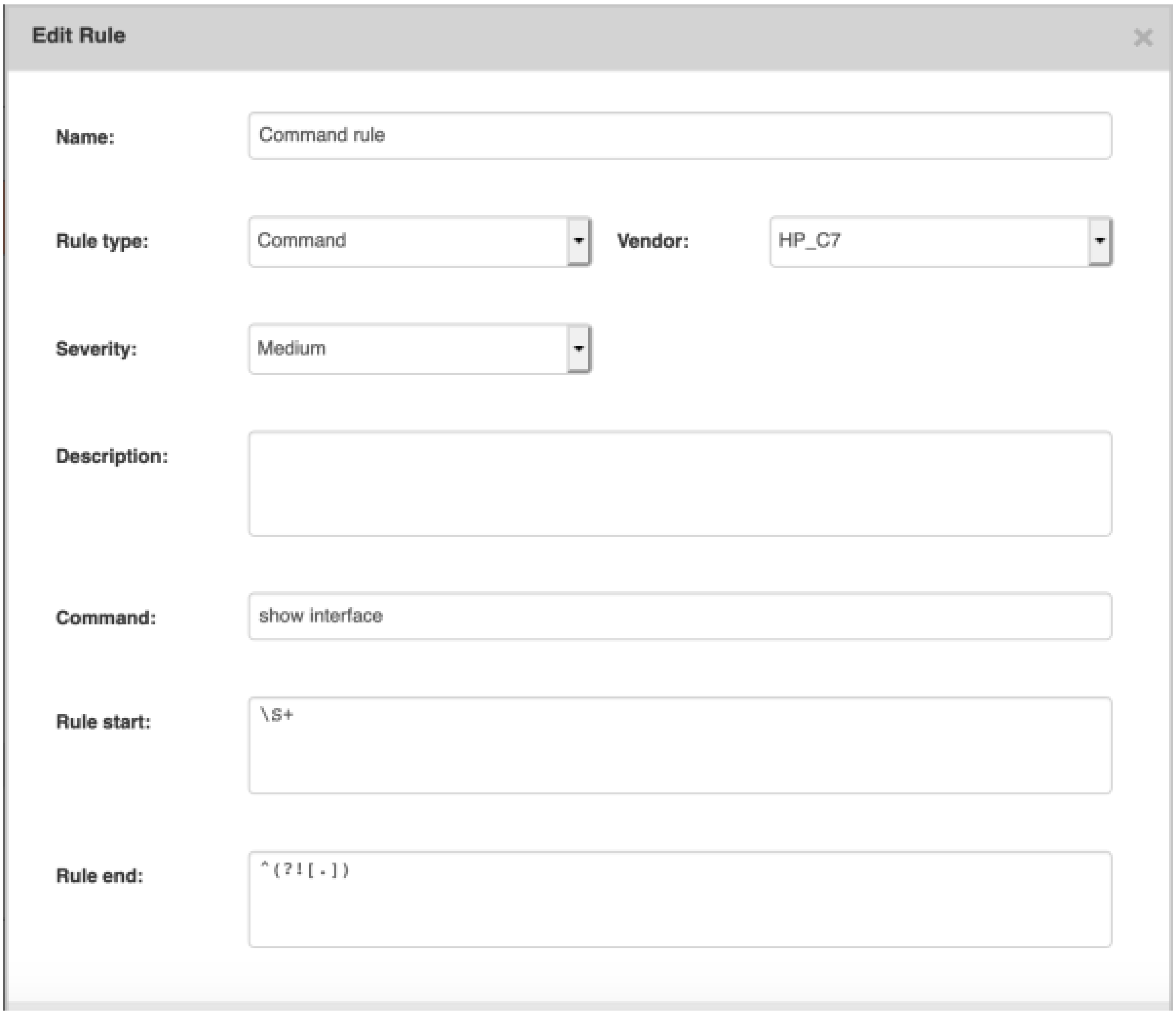
Allowed statements:

- **if**, **then**, **else**
- **and**
- **or**
- **(** and **)** brackets

Current condition names: **A, or, B, or, C**

Command Rules

Command rules allow you to check the result of a command for compliance. The form is nearly identical for normal rules, with the addition of a Command-field:



Edit Rule

Name: Command rule

Rule type: Command

Vendor: HP_C7

Severity: Medium

Description:

Command: show interface

Rule start: \S+

Rule end: ^(?![.])

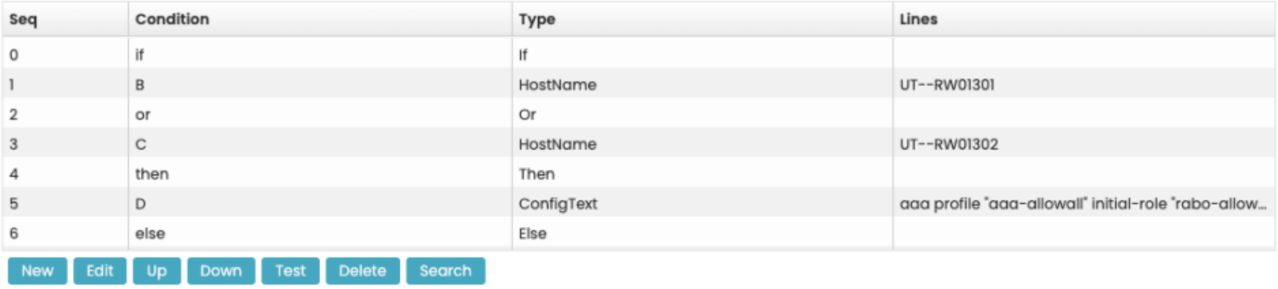
This is the command that will be run on the server.

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.

When you finish editing a command rule, the nodes that are linked to it are automatically scheduled for an nccm poll, following a compliance check. It will take the nccmd daemon a few minutes to get to it, but its update process is automated.

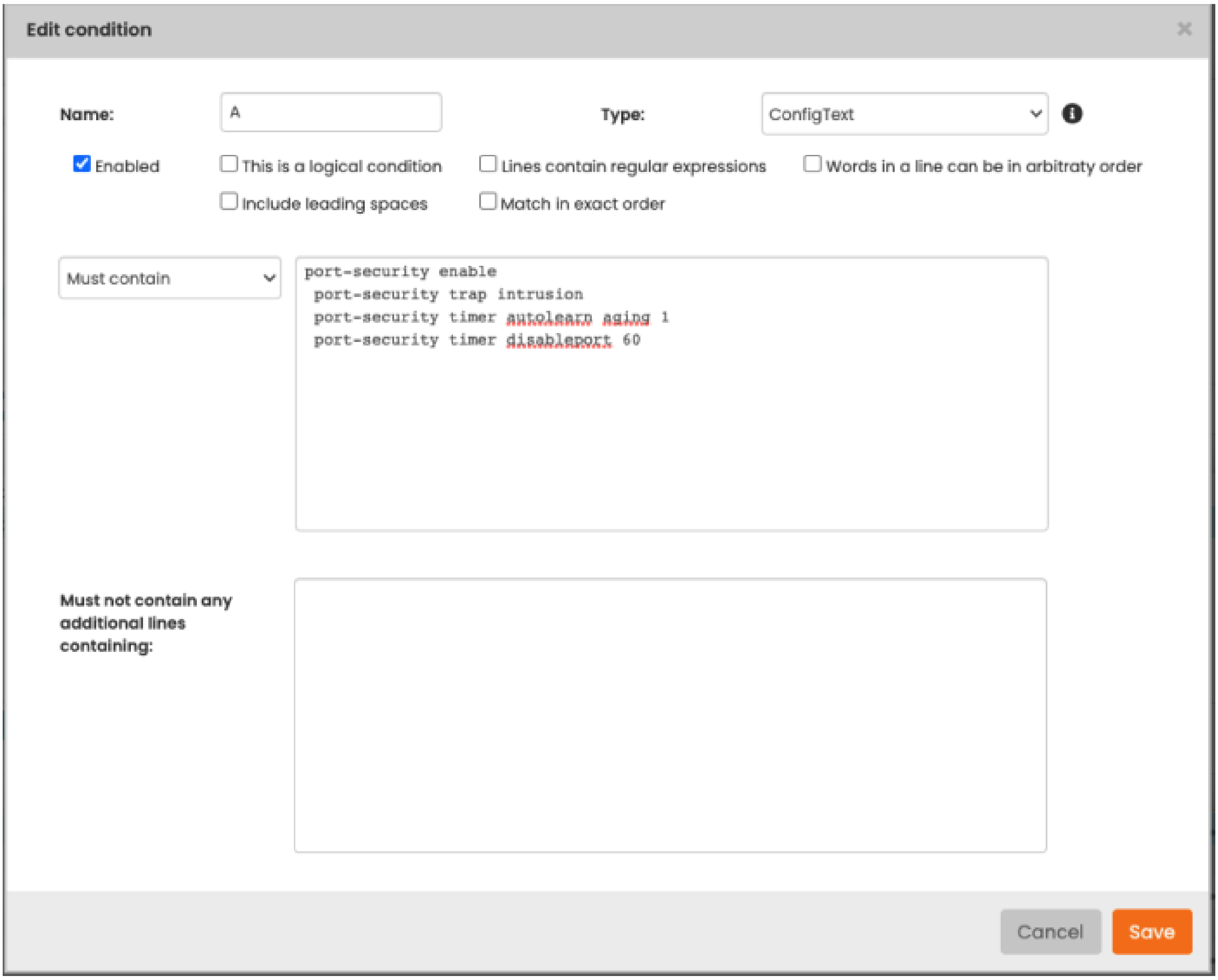
Conditions



Seq	Condition	Type	Lines
0	if	If	
1	B	HostName	UT--RW01301
2	or	Or	
3	C	HostName	UT--RW01302
4	then	Then	
5	D	ConfigText	aaa profile "aaa-allowall" initial-role "rabo-allow...
6	else	Else	

There are two types of conditions: Conditions that test a config block, and logic conditions that tie everything together. Logic conditions can be either 'if' ('then', 'else'), 'and', 'or' and '(' and ')' to group things together. Together they form a logic string that needs to be valid for a rule to be compliant. A rule won't work if the syntax of its conditions is invalid, you can use the 'test'-button at the rules-grid to verify this.

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:

You can switch between condition types using the “This is a logical condition”-checkbox. Like policies, conditions can also be enabled or disabled. A disabled condition will not be evaluated from a rule's logic and compliance checks. Disabled conditions will also be greyed out in the conditions grid in the main compliance form.

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

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.

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.

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 }
```

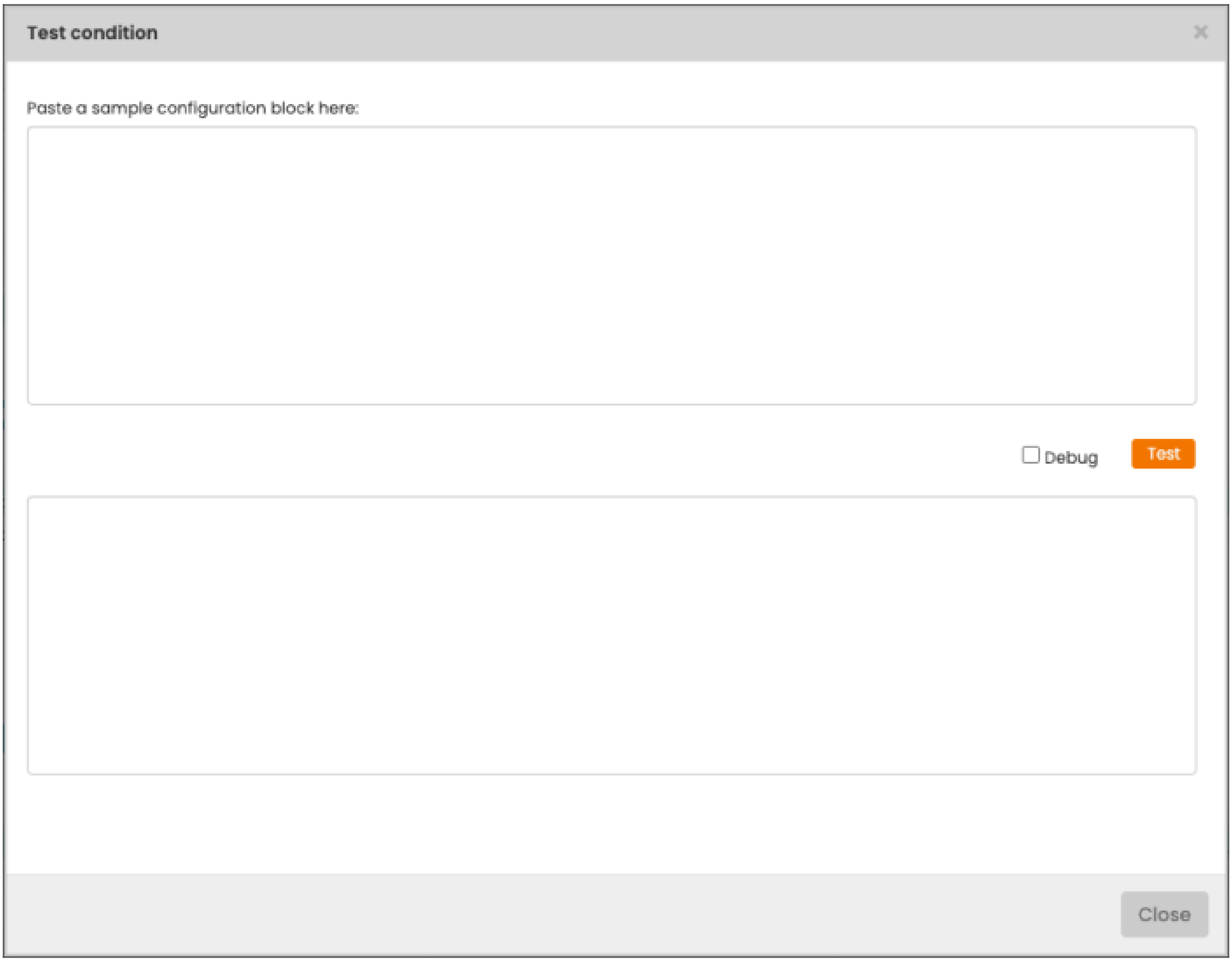
Where the ip addresses don't have a set order. F5 is an example of a vendor that can do this. For those cases, you can check the “Words can be in arbitrary order”-checkbox. This will evaluate a line against a condition, and it will not care about the order the different words (words here is defined by 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.

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.

Test condition



In order to help you test whether your condition is valid, there is the test form. You can access it by the “test” button under the conditions grid.

The test form validates your condition against a config block that you provide, so it doesn't retrieve any node data from the database. It's meant as a quick tool to check for typos. For a more extensive test that uses actual node data, you can use the policy test form.

If the debug-checkbox is checked, the output will be a bit more detailed.

From:
https://wiki.netyce.com/ - **Technical documentation**

Permanent link:
https://wiki.netyce.com/doku.php/menu:cmpl:policies

Last update: **2022/06/02 11:05**