



N2OS

Software Development Kit

N2OS v25.2.0 – 2025-07-03

Legal notices

Information about the Nozomi Networks copyright and use of third-party software in the Nozomi Networks product suite.

Copyright

Copyright © 2013-2025, Nozomi Networks. All rights reserved. Nozomi Networks believes the information it furnishes to be accurate and reliable. However, Nozomi Networks assumes no responsibility for the use of this information, nor any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, copyright, or other intellectual property right of Nozomi Networks except as specifically described by applicable user licenses. Nozomi Networks reserves the right to change specifications at any time without notice.

Third Party Software

Nozomi Networks uses third-party software, the usage of which is governed by the applicable license agreements from each of the software vendors. Additional details about used third-party software can be found at <https://security.nozominetworks.com/licenses>.

Contents

Chapter 1. Scriptable protocols.....	5
Architecture.....	7
Setup.....	9
Script parameters.....	10
Writing a standalone scriptable protocol.....	13
Writing an extension scriptable protocol.....	19
API reference.....	24
Chapter 2. Scriptable variables.....	47
Setup.....	49
Writing a variables correlation script.....	50
API reference.....	53
Chapter 3. OpenAPI.....	55
OpenAPI.....	57
Setup.....	58
Errors.....	60
CLI endpoint.....	61
Import CSV endpoint.....	62
Import JSON endpoint.....	64
Alerts endpoint.....	66
Trace endpoint.....	72
Users endpoint.....	75
PCAPs endpoint.....	81
Reports endpoint.....	87
Report templates endpoint.....	91
Quarantine endpoint.....	93
Threat intelligence.....	94
Sensors endpoint.....	96
Throttling policy.....	98
Chapter 4. Data model reference.....	99
alerts.....	101
appliances.....	104
assertions.....	106
assets.....	107
asset_cves.....	109
captured_logs.....	111
captured_urls.....	112
function_codes.....	113
health_log.....	114

- link_events.....115
- links.....116
- node_cpe_changes.....119
- node_cpes.....121
- node_cves.....123
- node_points.....125
- nodes.....126
- report_files.....131
- sessions_history.....132
- sessions.....134
- variable_history.....136
- variables.....137
- Chapter 5. Data integration best practices.....139**
 - OpenAPI data.....141
 - Nozomi Networks certification.....145
 - Technical requirements.....145
 - Certify your integration.....146
- Glossary.....147

Chapter 1. Scriptable protocols



Architecture

With scriptable protocols, it is possible to add new detection capabilities in Guardian. These protocols are implemented using the [Lua](#) scripting language and have access to most of the hardcoded traffic inspection and detection functionality via a dedicated [application programming interface \(API\)](#).

There are two different kinds of scriptable protocols, differing on the way they are invoked in the traffic processing pipeline:

- Standalone protocols
- Extensions

Standalone protocols

Standalone protocols handle [transmission control protocol \(TCP\)](#) / UDP traffic in an exclusive way: when a session is deemed to be handled by a standalone protocol then all packets belonging to it will be handled by this protocol and no other standalone protocol (hardcoded or not) will be involved. With standalone scriptable protocols, it is possible to implement a completely new protocol that currently doesn't have a dedicated handler in Guardian.

Standalone protocols are identified by their name and when instantiated they are inserted to the list of possible packet handlers in the packet pipeline. When offered a packet they have the opportunity to declare themselves (via the `can_handle` function) as handlers of the session. If a positive response is provided, then this packet as well as all subsequent packets in the session are classified as being handled by the specific protocol and will all be offered for processing to the `update_status` callback.

Another possibility with standalone scriptable protocols is to override existing hardcoded implementations of a protocol. In such a case, the name of the scriptable protocol should be the same as the one to be overridden. This possibility should be used only in extraordinary circumstances: hardcoded protocols offer rich handling and it is going to be challenging to offer equivalent handling via [Lua](#) scripting.

Standalone protocols get to handle raw packets as they are captured by the network interface. For this reason, any kind of message delineation / defragmentation will need to be done by the scriptable protocol itself.

Extensions

Extensions are scriptable protocols that add detection capabilities on top of existing (hardcoded or not) standalone protocols. There is no need to completely handle the traffic of a specific protocol (this is done by the standalone protocol being extended), but instead to implement the additional detection capabilities requested.

Extension protocols are identified by the protocol being extended and their name. Their identification string is constructed by concatenating the name of the standalone protocol being extended, with the name of the extension protocol using the hash character as separator. For example, the extension protocol `token_detector` that is extending the `http` protocol will use `http#token_detector` for its identification.

Extension scriptable protocols are offered packets that have been accepted and handled by the base standalone protocol. Accordingly, their `can_handle` function needs to check only if the extension should handle a specific packet of the base protocol (and not do thorough checking if the packet is indeed a base protocol one). Also, if there is any message delineation / defragmentation done for the protocol, this will already been done by the base protocol: the extensions will get to handle complete, reconstructed messages.

Safe restarts

The activation of a scriptable protocol is an operation that carries some risk: since new traffic handling components are being added in the packet processing pipeline, it is possible that the characteristics of the system are adversely affected, or even worse that restarts are triggered. Care has been taken to protect the system from errors happening during script execution, but it is still possible to eventually have system disturbances.

As a safety net against such occurrences, the safe restarts detection feature has been implemented. According to it, if Guardian is doing non graceful restarts too frequently (by default if more than 3 times in the last 15 minutes), then the system switches into safe restart mode. In this mode, no scriptable protocol gets activated (the user gets informed about it in the log file).

Setup

Do this procedure to add a new scriptable protocol.

To add a new scriptable protocol there are two options:

1. Explicit configuration
2. Handle as custom user contents

Explicit configuration

1. Copy the [Lua](#) script in `/data/scriptable_protocols/`
2. Configure Guardian with this rule `conf.user configure probe scriptable-protocol <protocol_name> <script_name>` in [command-line interface \(CLI\)](#) (<script_name> is the name of the file including the extension)
3. There is no need to restart the [intrusion detection system \(IDS\)](#) after the scriptable protocol configuration: the system will automatically activate it (alongside any other scriptable protocols already configured)

Custom user contents

1. Make sure that the scriptable protocol parameters (most importantly, the name) have been set via comments in the script body.
2. Add the `.content` extension to the script file name.
3. Copy the [Lua](#) script in `/data/contents/scriptable_protocols/`
4. Notify [IDS](#) that the scriptable protocols contents have changed by issuing in the [CLI](#): `ids contents_reload {"content_type": "scriptable_protocols"}`
5. There is no need to restart the [IDS](#), the system will automatically activate all scriptable protocols delivered as contents, together with all those that have been explicitly configured.

After these steps the new protocol is loaded in Guardian and will analyze the network traffic.

Script parameters

Apart from the [Lua](#) script that provides the implementation of a scriptable protocol, there are a number of parameters that are configurable via different means. These parameters affect diverse aspects of a scriptable protocol such as which protocol it extends and which Guardian versions it is applicable to. Scriptable protocol parameters can be configured in two ways:

1. [JavaScript Object Notation \(JSON\)](#) object at the end of the `conf.user` `configure probe scriptable-protocol` configuration line
2. Special [Lua](#) comments (`--nn-`) embedded within the [Lua](#) script

Configuration line

After the mandatory `<protocol_name>` and `<script_name>` arguments in the `probe scriptable-protocol` configuration line, a [JSON](#) object may optionally be provided. This object shall hold the keys / values for all the parameters that need to be configured. For example, if a scriptable protocol is to be allowed to be executed concurrently, then the `multithreaded` parameter needs to be set to `true`:

```
conf.user configure probe scriptable-protocol fast_protocol script.lua
{ "multithreaded": true }
```

Embedded as script comments

When a scriptable protocol is not to be loaded via explicit configuration, but as custom user contents (i.e. by storing as `.content` file in `/data/contents/scriptable_protocols`), all parameters are provided as special [Lua](#) comments at the top of the file. These comments have the `--nn-` prefix followed by the parameter key and value, separated by the colon (`:`) character. Using the same example as above, if a scriptable protocol is to be allowed to be executed concurrently, then the `multithreaded` parameter shall be set to `true` by adding as comment in the script file the line:

```
--nn- multithreaded: true
```

Note that when a scriptable protocol is loaded via an explicit configuration line, then the parameters provided as comments within the file are ignored: only those explicitly provided in the configuration line (as [JSON](#) object) are going to be used.

Supported parameters

Name (`name`)

The `name` parameter provides the string which is going to be used for identifying the scriptable protocol within the system. If the scriptable protocol is a standalone one, then the Protocol [identifier \(ID\)](#) shall consist of the name string only. If the protocol is an extension, then the Protocol [ID](#) shall be created by combining the `name` and `extends_protocol` parameters.

The `name` parameter must be provided (as a script comment) when the scriptable protocol is to be loaded as custom user content.

Example:

```
--nn- name: some_scada_prot
```

Extends protocol (**extends_protocol**)

The `extends_protocol` parameter provides the name of the standalone protocol to be extended, effectively marking the scriptable protocol as an extension. In such a scenario, the scriptable protocol shall be identified in the system by concatenating the `extends_protocol` and `name` parameters, using the hash (#) as separator.

For example, if the scriptable protocol is an [hypertext transfer protocol \(HTTP\)](#) extension that e.g. detects invalid tokens then the following script comments could be set:

```
--nn- name: invalid_token_detector
--nn- extends_protocol: http
```

If this scriptable protocol is to be explicitly configured, then the following configuration line should be used:

```
conf.user configure probe scriptable-protocol invalid_token_detector
script.lua { "extends_protocol": "http" }
```

Multithreaded (**multithreaded**)

The `multithreaded` parameter dictates whether the scriptable protocol should be allowed to be executed concurrently. If set to `true`, then the script will be instantiated multiple times and each packet processing thread shall always use one of them. In such a scenario, there is no shared [Lua](#) state between the different instances and thus persistent variables written by one instance will not have their values visible by other instances.

The default value for this parameter is `false`

Examples:

```
--nn- multithreaded: true
```

```
conf.user configure probe scriptable-protocol prot_name script.lua
{ "multithreaded": true }
```

N2OS Version (**n2os_version**)

The `n2os_version` parameter allow scriptable protocols to be conditionally loaded, based on the version of the currently running [IDS](#). This is useful when the same script is to be distributed to Guardians on different software levels and the script is applicable only to some of them.

The value of this parameter consists of comma separated version specifiers. Each version specifier consists of an operator and a version literal. For example, the version specifier `>= 20.0.0` would be satisfied by all versions that are greater or equal to 20.0.0. If multiple version specifiers are provided, then they must be all satisfied for the scriptable protocol to be loaded. For example, if the `n2os_version` parameter is set to `>= 20.0.0, != 21.0.0`, then the scriptable protocol would be loaded for versions 20.0.0 and 22.0.0, but not for 19.0.0 or 21.0.0.

The operators that are supported are:

1. ==: Equal
2. !=: Not equal
3. >: Greater
4. >=: Greater or equal
5. <=: Less or equal
6. <: Less

For example, if this scriptable protocol is to be loaded for versions that are greater or equal to 20.0.0, but not equal to 21.0.0, then the following comment should be added to the script:

```
--nn- n2os_version: >= 20.0.0, != 21.0.0
```

If the configuration is to be explicit, then the following [JSON](#) object should be provided:

```
conf.user configure probe scriptable-protocol prot_name script.lua
{ "n2os_version": ">= 20.0.0, != 21.0.0" }
```

This parameter can be provided multiple times, if there are multiple disjoint version sets that could be satisfied. In such a case, the scriptable protocol will be loaded if any of the provided parameter values is satisfied.

For example, if a scriptable protocol should be loaded on any version belonging to the releases with major version 20 or 22, then the following comments should be set in the script:

```
--nn- n2os_version: >= 20.0.0, < 21.0.0
--nn- n2os_version: >= 22.0.0, < 23.0.0
```

If the script is going to be loaded via explicit configuration line, then the following [JSON](#) object should be provided:

```
conf.user configure probe scriptable-protocol prot_name script.lua
{ "n2os_version": [">= 20.0.0, < 21.0.0", ">= 22.0.0, < 23.0.0"] }
```


Writing a standalone scriptable protocol

The language used to write a scriptable protocol is [Lua](https://www.lua.org/start.html), please refer to the official [Lua](https://www.lua.org/start.html) documentation (<https://www.lua.org/start.html>) to learn more.

This is a minimal protocol implementation:

```
function can_handle()  
    return true  
end
```

From the example we can see that the only mandatory thing to do is to define a function called `can_handle` which returns `true` if it recognize the target protocol.

Of course this implementation is not very useful and it will try to handle every packet so let's write something more complex to detect and analyze some modbus traffic:

```
function can_handle()  
    return packet.source_port() == 502 or packet.destination_port() == 502  
end
```

Here we can see a usage of the [API](#) to retrieve the packet ports. In this way the check is a bit more accurate but it's still insufficient to detect a modbus packet in the real world.

Let's start to do some deep packet inspection:

```
function can_handle()  
    if data_size() < 8 then  
        return false  
    end  
  
    local has_right_port = packet.source_port() == 502 or  
    packet.destination_port() == 502  
  
    fwd(2)  
    local has_right_protocol_id = consume_n_uint16() == 0  
    local expected_length = consume_n_uint16()  
  
    return has_right_port and  
        has_right_protocol_id and  
        remaining_size() == expected_length  
end
```

WARNING: don't use global variables. Variables defined outside of the `can_handle` and `update_status` functions are global and their status is shared across every session of the same protocol.

NOTE: the `fwd` and `consume_*` functions will move forward the payload pointer.

NOTE: the result of the `remaining_size` function depends on the position of the payload pointer.

In this example we use the [API](#) to inspect the content of the payload. First we check that there are enough bytes, a modbus packet is at least 8 bytes long. Then we check the port in the same way we did in the previous example, then we skip two bytes with the function `fwd` and we read the next two 16 bit integers. We check that the protocol id is zero and that the length written in the packet matches the remaining bytes count in our payload. If every check succeeds `true` is returned, informing Guardian that the next packets in this session should be analyzed by this protocol decoder.

A protocol with just the `can_handle` function implemented will only create the node and the session in the Network but the link is still missing from the graph, no additional information will be displayed in the Process information.

To extract more information from the modbus packets we are going to implement the `update_status` function:

```
function get_protocol_type()
    return ProtocolType.SCADA
end

function can_handle()
    return is_modbus()
end

function update_status()
    if not is_modbus() then
        return
    end

    local is_request = packet.destination_port() == 502
    local rtu_id = consume_uint8()
    local fc = consume_uint8() & 0x7f

    if is_request then
        is_packet_from_src_to_dst(true)
        set_roles("consumer", "producer")

        if fc == 6 then
            local address = consume_n_uint16()

            local value = DataValue.new()
            value.value = read_n_uint16()
```

```

    value.cause = DataCause.WRITE
    value.type = DataType.ANALOG
    value.time = packet.time()

    execute_update_with_variable(FunctionCode.new(fc),
    RtuId.new(rtu_id), "r"..tostring(address), value)
    return
  end
end

execute_update()
end

```

**Note:**

To avoid duplication we created a `is_modbus` function from the content of the previous `can_handle` function.

**Note:**

The `is_modbus` function has the effect to advance the payload pointer by 6 bytes, so we can directly read the `rtu_id` without further payload pointer manipulations.

**Note:**

We defined the `get_protocol_type` function to define the protocol type.

In this example of `update_status` we read more data from the payload and we decode the write single register request. We can understand the direction of the communication so we call `is_packet_from_src_to_dst` with `true` to notify Guardian and create a link and we call `set_roles` to set the roles on the involved nodes.

To insert a variable in Guardian there is the `execute_update_with_variable` function, it takes 4 arguments: the function code, the rtu id, the variable name and the value. The `FunctionCode` and `RtuId` objects can be constructed from a string or a number, the `DataValue` object can be constructed with the empty constructor and then filled with the available information.

With the next example we cover a more complex case and we store some data in the session to handle a request and a response:

```

local PENDING_FC = 1
local PENDING_START_ADDR = 2
local PENDING_REG_COUNT = 3

```

```
function update_status()
  if not is_modbus() then
    return
  end

  rwd()

  local is_request = packet.destination_port() == 502
  local transaction_id = consume_n_uint16()
  fwd(4)

  local rtu_id = consume_uint8()
  local fc = consume_uint8() & 0x7f

  if is_request then
    is_packet_from_src_to_dst(true)
    set_roles("consumer", "producer")
    session.set_pending_request_number(transaction_id, PENDING_FC, fc)

    if fc == 3 then
      if remaining_size() < 4 then
        return
      end

      local start_addr = consume_n_uint16()
      local registers_count = consume_n_uint16()

      session.set_pending_request_number(transaction_id,
PENDING_START_ADDR, start_addr)
      session.set_pending_request_number(transaction_id,
PENDING_REG_COUNT, registers_count)
    end
  else
    is_packet_from_src_to_dst(false)
    local req_fc = session.read_pending_request_number(transaction_id,
PENDING_FC)
```

```
    if fc == req_fc then
        if fc == 3 then
            local start_addr =
session.read_pending_request_number(transaction_id, PENDING_START_ADDR)
            local reg_count =
session.read_pending_request_number(transaction_id, PENDING_REG_COUNT)
            session.close_pending_request(transaction_id)

            if remaining_size() < 1 then
                return
            end

            local byte_count = consume_uint8()

            if remaining_size() ~= byte_count or
                reg_count * 2 ~= remaining_size() then
                send_alert_malformed_packet("Packet is too small")
                return
            end

            for i = 0, reg_count - 1, 1 do
                local value = DataValue.new()
                value.value = consume_n_uint16()
                value.cause = DataCause.READ_SCAN
                value.type = DataType.ANALOG
                value.time = packet.time()

                execute_update_with_variable(FunctionCode.new(fc),
                    RtuId.new(rtu_id),
                    "r"..tostring(start_addr+i),
                    value)
            end

            return
        end
    end
end
```

```
execute_update()  
end
```

This time we are focusing on the read holding register function code, to understand the communication and create a variable we need to analyze both the request and the response and we need to keep some data from the request and use it in the response. To achieve this we can use the functions provided by the `session` object.

Writing an extension scriptable protocol

In this section, an extension scriptable protocol is to be developed, which will add some additional detection capability for [HTTP](#) traffic. Specifically, the `script` tag contents in POST requests will be correlated to some text in the response. If data are found in both request and response, then a property combining these two will be set in the client node.

Since we are authoring an [HTTP](#) extension, it is known beforehand that only [HTTP](#) traffic will be offered for processing. The `can_handle` can be trivial since it is desired that all [HTTP](#) traffic passes through the extension:

```
function can_handle()  
  return true  
end
```

In the `update_status` function, the extension functionality is to be implemented:

```
function update_status()  
  msg = read_string()  
  if is_post(msg) then  
    handle_post(msg)  
  elseif is_response(msg) then  
    handle_response(msg)  
  end  
end
```

From all the functions that are invoked in it, only `read_string` is provided by the Guardian [API](#), the other ones are defined within the scriptable protocol script and will be provided later on.

In this function, the complete [HTTP](#) message is read (reminder that extensions get to handle defragmented data) and then based on checks if this is a POST or a response message the appropriate functions are invoked.

The functions that support the handling of the POST messages are:

```
PENDING_SCRIPT_KEY = 10001  
  
function is_post(msg)  
  req_pattern = "^POST"  
  index, _ = string.find(msg, req_pattern)  
  return index ~= nil  
end
```

```

function parse_script(msg)
    pattern = "<script>(.*?)</script>"
    _, _, script = string.find(msg, pattern)
    return script
end

function handle_post(msg)
    script = parse_script(msg)
    if script then
        session.set_pending_request_string(PENDING_SCRIPT_KEY, 0, script)
    end
end

```

The `is_post` function checks whether the [HTTP](#) message starts with the `POST` string. If the index returned by the `string.find` [Lua](#) function is not `nil`, then the `true` is returned (the `^` anchor makes sure that a match will only be made at the beginning of the string).

The `parse_script` function again uses the [Lua](#) `string.find` function to check for the `script` contents, this time returning the captured text in parentheses. If no match could be found, then `nil` will be returned.

The `handle_post` function again searches for the `script` contents and if found it stores it in the session, under the `PENDING_SCRIPT_KEY` key. The `session.set_pending_request_string` function is part of the Guardian specific [API](#) and stores an arbitrary string in the session. The first argument is the `request_id` and since the session data store is common between the base protocol + all extensions, the developer needs to be careful when selecting this number, so that there is no interference between the session data writers. It is recommended that user developed scripts use consecutive numbers starting from 10001 for extensions to the same protocol, to avoid such conflicts. The second argument is used to discriminate between different data values stored for the same request, it is set here to 0.

Thus at the end of the `POST` handling functions, a pending request string may be set on the session which will hold the contents of the `script` tag.

The response handling functions are:

```

function is_response(msg)
    res_pattern = "^HTTP"
    index, _ = string.find(msg, res_pattern)
    return index ~= nil
end

function parse_p(msg)
    pattern = "<p>(.*?)</p>"
    _, _, p_body = string.find(msg, pattern)

```



```

    return p_body
end

function handle_response(msg)
    if session.has_pending_request_value(PENDING_SCRIPT_KEY, 0) then
        p_body = parse_p(msg)
        if p_body then
            pending_script =
session.read_pending_request_string(PENDING_SCRIPT_KEY, 0)
            restored_value = pending_script .. " -- " .. p_body
            packet.destination_node():set_property("restored_value",
restored_value)
        end
        session.close_pending_request(PENDING_SCRIPT_KEY)
    end
end
end

```

The `is_response` and `parse_p` functions are very similar to those that were described for the `POST` side handling, so they will not be commented further. Instead, the focus will be in the more interesting `handle_response` one.

For the response handling, the script first checks if the script key has already been stored in the session (when a `POST` message was handled). Note how in order to access the correct placeholder in the session, the `session.has_pending_request_value` function has been invoked with the same arguments as when the value was stored in the session. If the script value has been found in the session then the body of the `p` tag is searched for and if that one is found as well, then the script value is fetched from the session (`session.read_pending_request_string`) and gets concatenated with the body of the `p` tag. The resulting string is stored as a property in the destination node (`packet.destination_node():set_property`). At the end, the data stored in the session are cleared `session.close_pending_request` since they are no longer needed and would otherwise consume memory unnecessarily.

In the code block below, the complete script is provided:

```

function can_handle()
    return true
end

PENDING_SCRIPT_KEY = 10001

function parse_script(msg)
    pattern = "<script>(.*?)</script>"
    _, _, script = string.find(msg, pattern)
    return script
end

```

```
end

function parse_p(msg)
    pattern = "<p>(.*?)</p>"
    _, _, p_body = string.find(msg, pattern)
    return p_body
end

function is_post(msg)
    req_pattern = "^POST"
    index, _ = string.find(msg, req_pattern)
    return index ~= nil
end

function is_response(msg)
    res_pattern = "^HTTP"
    index, _ = string.find(msg, res_pattern)
    return index ~= nil
end

function handle_post(msg)
    script = parse_script(msg)
    if script then
        session.set_pending_request_string(PENDING_SCRIPT_KEY, 0, script)
    end
end

function handle_response(msg)
    if session.has_pending_request_value(PENDING_SCRIPT_KEY, 0) then
        p_body = parse_p(msg)
        if p_body then
            pending_script =
session.read_pending_request_string(PENDING_SCRIPT_KEY, 0)
            restored_value = pending_script .. " -- " .. p_body
            packet.destination_node():set_property("restored_value",
restored_value)
        end
    end
end
```

```
        session.close_pending_request(PENDING_SCRIPT_KEY)
    end
end

function update_status()
    msg = read_string()
    if is_post(msg) then
        handle_post(msg)
    elseif is_response(msg) then
        handle_response(msg)
    end
end
end
```

API reference

Available Lua libraries

- base
- string
- table
- math
- debug
- utf8

Data types

Class	FieldInfoAttributes
Constructors	<ul style="list-style-type: none"> • <code>FieldInfoAttributes.new()</code>
Read/write properties	<ul style="list-style-type: none"> • <code>FieldInfoAttributes.source_code</code> (<code>FieldSourceCode</code>) • <code>FieldInfoAttributes.granularity</code> (<code>FieldGranularity</code>) • <code>FieldInfoAttributes.confidence</code> (<code>FieldConfidence</code>)


Class	FunctionCode
Constructors	<ul style="list-style-type: none"> • <code>FunctionCode.new(<string>)</code> • <code>FunctionCode.new(<number>)</code>

Class	RtuId
Constructors	<ul style="list-style-type: none"> • <code>RtuId.new(<string>)</code> • <code>RtuId.new(<number>)</code>

Class	DataValue
Constructors	<ul style="list-style-type: none"> • <code>DataValue.new()</code>
Read/write properties	<ul style="list-style-type: none"> • <code>DataValue.value</code> (<code>number</code>) • <code>DataValue.str_value</code> (<code>string</code>) • <code>DataValue.cause</code> (<code>DataCause</code>) • <code>DataValue.time</code> (<code>number</code>, milliseconds since epoch) • <code>DataValue.type</code> (<code>DataType</code>)

Class	Variable
--------------	----------

Methods	<ul style="list-style-type: none"> • <code>set_label(<string>)</code>
Class	Node
Methods	<ul style="list-style-type: none"> • <code>set_property(<key>, <value>)</code> • <code>get_property(<key>)</code> • <code>delete_property(<key>)</code> • <code>set_label(<label>)</code> • <code>set_vendor(<string>, <FieldInfoAttributes>)</code> • <code>get_vendor()</code> • <code>set_product_name(<string>, <FieldInfoAttributes>)</code> • <code>get_product_name()</code> • <code>set_firmware_version(<string>, <FieldInfoAttributes>)</code> • <code>get_firmware_version()</code> • <code>set_serial_number(<string>, <FieldInfoAttributes>)</code> • <code>get_serial_number()</code> • <code>notify_vulnass()</code> <ul style="list-style-type: none"> ◦ notifies the vulnass process to create a Common Platform Enumeration (CPE)
Enum	DataCause
Values	<ul style="list-style-type: none"> • <code>DataCause.READ_SCAN</code> • <code>DataCause.READ_CYCLIC</code> • <code>DataCause.READ_EVENT</code> • <code>DataCause.WRITE</code>
Enum	DataType

Values	<ul style="list-style-type: none"> • <code>DataType.ANALOG</code> <ul style="list-style-type: none"> ◦ the Analog type represents a floating point number • <code>DataType.DIGITAL</code> <ul style="list-style-type: none"> ◦ the Digital type represents a boolean type and can be either 0 or 1 • <code>DataType.BITSTRING</code> <ul style="list-style-type: none"> ◦ the Bitstring type represents a raw value in the form of a sequence of 0 and 1, e.g. "00101110" • <code>DataType.STRING</code> <ul style="list-style-type: none"> ◦ the String type represents a value in the form of a sequence of printable characters • <code>DataType.DOUBLEPOINT</code> <ul style="list-style-type: none"> ◦ the Double Point type represents a boolean value with an additional degree of redundancy. It is commonly used in protocols such as DNP3, IEC 104 or IEC 61850 • <code>DataType.TIMESTAMP</code> <ul style="list-style-type: none"> ◦ the Timestamp type represents a point in time in the format of milliseconds from the epoch <div>  Note: Only ANALOG, DIGITAL and DOUBLEPOINT types are kept in consideration by the Process Learning Engine when detecting deviations from the baseline. </div>
---------------	--

Enum	<code>FieldConfidence</code>
Values	<ul style="list-style-type: none"> • <code>FieldConfidence.MANUAL_OR_IMPORT</code> • <code>FieldConfidence.HIGH</code> • <code>FieldConfidence.GOOD</code> • <code>FieldConfidence.LOW</code> • <code>FieldConfidence.UNKNOWN</code> • <code>FieldConfidence.UNSUPPORTED</code>

Enum	<code>FieldGranularity</code>
Values	<ul style="list-style-type: none"> • <code>FieldGranularity.MANUAL_OR_IMPORT</code> • <code>FieldGranularity.COMPLETE</code> • <code>FieldGranularity.PARTIAL</code> • <code>FieldGranularity.GENERIC</code> • <code>FieldGranularity.UNKNOWN</code> • <code>FieldGranularity.UNSUPPORTED</code>

Enum	FieldSourceCode
Values	<ul style="list-style-type: none"> • FieldSourceCode.NONE • FieldSourceCode.ENRICHMENT • FieldSourceCode.PASSIVE • FieldSourceCode.DATA_INTEGRATION • FieldSourceCode.SMART_POLLING • FieldSourceCode.ARC • FieldSourceCode.ASSET_KB • FieldSourceCode.IMPORT • FieldSourceCode.MANUAL • FieldSourceCode.OVERWRITE

Enum	ProtocolType
Values	<ul style="list-style-type: none"> • ProtocolType.SCADA • ProtocolType.NETWORK • ProtocolType.IoT

Functions

Syntax	data(<index>)
Parameters	<ul style="list-style-type: none"> • index: the position of the byte to read, starting from 0
Description	Return the value of the byte from the specified position, return 0 if index is out of bounds

Syntax	data_size()
Description	Return the total size of the payload

Syntax	remaining_size()
Description	Return the size of the payload from the pointer to the end. The result depends on the usage of functions fwd(), rwd() and consume_*().

Syntax	fwd(<amount>)
Parameters	<ul style="list-style-type: none"> • amount: the number of bytes to skip
Description	Move the payload pointer by the specified number of bytes.

Syntax	<code>rwd()</code>
Description	Move the payload pointer to the beginning of the payload.

Syntax	<code>read_uint8()</code>
Description	Read an unsigned 8bit integer at the payload pointer position.

Syntax	<code>read_int8()</code>
Description	Read an signed 8bit integer at the payload pointer position.

Syntax	<code>read_n_uint16()</code>
Description	Read a network order unsigned 16bit integer at the payload pointer position.

Syntax	<code>read_h_uint16()</code>
Description	Read a host order unsigned 16bit integer at the payload pointer position.

Syntax	<code>read_n_int16()</code>
Description	Read a network order signed 16bit integer at the payload pointer position.

Syntax	<code>read_h_int16()</code>
Description	Read a host order signed 16bit integer at the payload pointer position.

Syntax	<code>read_n_uint32()</code>
Description	Read a network order unsigned 32bit integer at the payload pointer position.

Syntax	<code>read_h_uint32()</code>
Description	Read a host order unsigned 32bit integer at the payload pointer position.

Syntax	<code>read_n_int32()</code>
Description	Read a network order signed 32bit integer at the payload pointer position.

Syntax	<code>read_h_int32()</code>
---------------	-----------------------------

Description	Read a host order signed 32bit integer at the payload pointer position.
--------------------	---

Syntax	<code>read_n_uint64()</code>
---------------	------------------------------

Description	Read a network order unsigned 64bit integer at the payload pointer position.
--------------------	--

Syntax	<code>read_h_uint64()</code>
---------------	------------------------------

Description	Read a host order unsigned 64bit integer at the payload pointer position.
--------------------	---

Syntax	<code>read_n_int64()</code>
---------------	-----------------------------

Description	Read a network order signed 64bit integer at the payload pointer position.
--------------------	--

Syntax	<code>read_h_int64()</code>
---------------	-----------------------------

Description	Read a host order signed 64bit integer at the payload pointer position.
--------------------	---

Syntax	<code>read_n_float()</code>
---------------	-----------------------------

Description	Read a network order float at the payload pointer position.
--------------------	---

Syntax	<code>read_h_float()</code>
---------------	-----------------------------

Description	Read a host order float at the payload pointer position.
--------------------	--

Syntax	<code>read_n_double()</code>
---------------	------------------------------

Description	Read a network order double at the payload pointer position.
--------------------	--

Syntax	<code>read_h_double()</code>
---------------	------------------------------

Description	Read a host order double at the payload pointer position.
--------------------	---

Syntax	<code>read_string()</code>
---------------	----------------------------

Description	Read a string at the payload pointer position until the null terminator.
--------------------	--

Syntax	<code>read_string_with_len(str_len)</code>
---------------	--

Description	Read a string at the payload pointer position for <code>str_len</code> bytes.
--------------------	---

Syntax	<code>consume_uint8()</code>
Description	Read an unsigned 8bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_int8()</code>
Description	Read a signed 8bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_uint16()</code>
Description	Read a network order unsigned 16bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_uint16()</code>
Description	Read a host order unsigned 16bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_int16()</code>
Description	Read a network order signed 16bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_int16()</code>
Description	Read a host order signed 16bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_uint32()</code>
Description	Read a network order unsigned 32bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_uint32()</code>
Description	Read a host order unsigned 32bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_int32()</code>
---------------	--------------------------------

Description	Read a network order signed 32bit integer at the payload pointer position and move the pointer after the data.
--------------------	--

Syntax	<code>consume_h_int32()</code>
Description	Read a host order signed 32bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_uint64()</code>
Description	Read a network order unsigned 64bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_uint64()</code>
Description	Read a host order unsigned 64bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_int64()</code>
Description	Read a network order signed 64bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_int64()</code>
Description	Read a host order signed 64bit integer at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_float()</code>
Description	Read a network order float at the payload pointer position and move the pointer after the data.


Syntax	<code>consume_h_float()</code>
Description	Read a host order float at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_n_double()</code>
Description	Read a network order double at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_h_double()</code>
Description	Read a host order double at the payload pointer position and move the pointer after the data.

Syntax	<code>consume_string()</code>
Description	Read a string at the payload pointer position until the null terminator and move the pointer after the data.

Syntax	<code>consume_string_with_len(str_len)</code>
Description	Read a string at the payload pointer position for <code>str_len</code> bytes and move the pointer after the data.

Syntax	<code>consume_xor_data(bytes_len, key, callback_function)</code>
Description	<p>Read <code>bytes_len</code> bytes at the payload pointer position and apply the XOR function with the byte in <code>key</code> at the same index. <code>callback_function</code> is then invoked with the payload pointer changed to the trasformed payload. When exiting from the callback function, the previous context is restored and the pointer is moved after the data.</p> <div style="border: 1px solid #007bff; border-radius: 10px; padding: 10px; margin-top: 10px;">  Note: <code>key</code>: must be an array of hex integers with a length greater or equal than <code>bytes_len</code>. </div>

Syntax	<code>consume_gzip_data(bytes_len, callback_function)</code>
Description	Read <code>bytes_len</code> bytes at the payload pointer position and decompress it with gzip. <code>callback_function</code> is then invoked with the payload pointer changed to the decompressed payload. When exiting from the callback function, the previous context is restored and the pointer is moved after the data.

Syntax	<code>consume_zlib_data(bytes_len, callback_function)</code>
Description	Read <code>bytes_len</code> bytes at the payload pointer position and decompress it with zlib. <code>callback_function</code> is then invoked with the payload pointer changed to the decompressed payload. When exiting from the callback function, the previous context is restored and the pointer is moved after the data.

Syntax	<code>compute_crc16(size, poly, init, xor_out, ref_in, ref_out)</code>
Parameters	<ul style="list-style-type: none"> • size: the amount of bytes on which the CRC is computed • poly, init, xor_out, ref_in, ref_out: the common CRC input parameters
Description	Compute the CRC16 of the remaining payload according to the input parameters. The input parameters for CRC functions can be easily found online. For example, to get a CRC16/DNP the parameters are: 0x3D65, 0x0000, 0xFFFF, true, true

Syntax	<code>compute_crc32(size, poly, init, xor_out, ref_in, ref_out)</code>
Parameters	<ul style="list-style-type: none"> • size: the amount of bytes on which the CRC is computed • poly, init, xor_out, ref_in, ref_out: the common CRC input parameters
Description	Compute the CRC32 of the remaining payload according to the input parameters. The input parameters for CRC functions can be easily found online. For example, to get a plain CRC32 the parameters are: 0x04C11DB7, 0xFFFFFFFF, 0xFFFFFFFF, true, true

Syntax	<code>set_roles(<client_role>, <server_role>)</code>
Parameters	<ul style="list-style-type: none"> • client_role: the role of the client • server_role: the role of the server
Description	Set the roles of the involved nodes, valid values are: "consumer", "producer", "historian", "terminal", "web_server", "dns_server", "db_server", "time_server", "other"

Syntax	<code>set_source_type(<node_type>)</code>
Parameters	<ul style="list-style-type: none"> • node_type: the type of the source node
Description	Set the type of the source node, valid values are: "switch", "router", "printer", "group", "OT_device", "broadcast", "computer"

Syntax	<code>variables_are_on_client()</code>
Parameters	
Description	Notify to Guardian that the variables should be added to the client node

syntax	<code>is_packet_from_src_to_dst(<is_from_src>)</code>
parameters	<ul style="list-style-type: none"> • <code>is_from_src</code>: true is the direction is from src to dst, false otherwise
description	notify Guardian about the direction of the packet, this function must be called to obtain a link creation

syntax	<code>execute_update()</code>
parameters	
description	notify Guardian about the a packet, at least one variant of <code>execute_update</code> should be called for every packet (available for standalone protocols only)

syntax	<code>execute_update_with_function_code(<function_code>, <rtu id>)</code>
parameters	<ul style="list-style-type: none"> • <code>function_code</code>: an object of type <code>functioncode</code> • <code>rtuid</code>: an object of type <code>rtuid</code>
description	notify Guardian about the a packet with a function code and a rtu id (available for standalone protocols only)

syntax	<code>execute_update_with_variable(<function_code>, <rtu id>, <var_name>, <value>)</code>
parameters	<ul style="list-style-type: none"> • <code>function_code</code>: an object of type <code>functioncode</code> • <code>rtu_id</code>: an object of type <code>rtuid</code> • <code>var_name</code>: the name of the variable • <code>value</code>: an object of type <code>datavalue</code> containing the value of the variable and some information about the data
description	notify Guardian about the a packet with a function code, a rtu id, a variable name and a variable value (available for standalone protocols only)

syntax	<code>execute_update_with_function(<function_code>, <rtu id>, <var_name>, <value>, <function>)</code>
---------------	---

parameters	<ul style="list-style-type: none"> • <code>function_code</code>: an object of type <code>functioncode</code> • <code>rtu_id</code>: an object of type <code>rtuid</code> • <code>var_name</code>: the name of the variable • <code>value</code>: an object of type <code>datavalue</code> containing the value of the variable and some information about the data • <code>function</code>: the function will be called passing <code>variable</code> as an argument
description	notify Guardian about the a packet with a function code, a rtu id, a variable name, a variable value and a function that give the possibility to directly access the variable (available for standalone protocols only)

syntax	<code>set_fc_info(<fc_num>, <fc_descr>)</code>
parameters	<ul style="list-style-type: none"> • <code>fc_num</code>: the function code as a number • <code>fc_descr</code>: description of the function code
description	establish a correspondence between a function code number and its description. When Guardian is later notified about this function code numerically, then the description will automatically be recalled and used

syntax	<code>AlertFactory.new_net_device()</code>
description	raise an alert of type VI:NEW-NET-DEV

syntax	<code>AlertFactory.firmware_transfer()</code>
description	raise an alert of type SIGN:FIRMWARE-TRANSFER

syntax	<code>AlertFactory.protocol_error(<reason>)</code>
parameters	<ul style="list-style-type: none"> • <code>reason</code>: a message to be displayed in the alert
description	raise an alert of type SIGN:PROTOCOL-ERROR

syntax	<code>AlertFactory.wrong_time(<reason>)</code>
parameters	<ul style="list-style-type: none"> • <code>reason</code>: a message to be displayed in the alert
description	raise an alert of type PROC:WRONG-TIME

syntax	<code>AlertFactory.sync_asked_again(<reason>)</code>
---------------	--

parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type PROC:SYNC-ASKED-AGAIN

syntax	<code>AlertFactory.protocol_flow_anomaly(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type VI:PROC:PROTOCOL-FLOW-ANOMALY

syntax	<code>AlertFactory.variable_flow_anomaly(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type VI:PROC:VARIABLE-FLOW-ANOMALY

syntax	<code>AlertFactory.dhcp_request(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:DHCP-OPERATION

syntax	<code>AlertFactory.invalid_ip(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:INVALID-IP

syntax	<code>AlertFactory.new_arp(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type VI:NEW-ARP

syntax	<code>AlertFactory.duplicated_ip(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:ARP:DUP

syntax	<code>AlertFactory.link_reconnection(<reason>)</code>
---------------	---

parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type NET:LINK-RECONNECTION

syntax	<code>AlertFactory.rst_from_producer(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type NET:RST-FROM-PRODUCER

syntax	<code>AlertFactory.tcp_syn(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type NET:TCP-SYN

syntax	<code>AlertFactory.tcp_flood(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:TCP-FLOOD

syntax	<code>AlertFactory.protocol_flood(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:PROTOCOL-FLOOD

syntax	<code>AlertFactory.mac_flood(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MAC-FLOOD

syntax	<code>AlertFactory.network_scan(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:NETWORK-SCAN

syntax	<code>AlertFactory.cleartext_password(<reason>)</code>
---------------	--

parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:CLEARTEXT-PASSWORD

syntax	<code>AlertFactory.ddos_attack(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:DDOS

syntax	<code>AlertFactory.unsupported_func(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:UNSUPPORTED-FUNC

syntax	<code>AlertFactory.illegal_parameters(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:ILLEGAL-PARAMETERS

syntax	<code>AlertFactory.weak_password(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:PASSWORD:WEAK

syntax	<code>AlertFactory.malware_detected()</code>
description	raise an alert of type SIGN:MALWARE-DETECTED

syntax	<code>AlertFactory.unknown_rtu(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:PROC:UNKNOWN-RTU

syntax	<code>AlertFactory.missing_variable(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert

description	raise an alert of type SIGN:PROC:MISSING-VAR
--------------------	--

syntax	<code>AlertFactory.protocol_injection(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:PROTOCOL-INJECTION

syntax	<code>AlertFactory.new_variable(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type VI:PROC:NEW-VAR

syntax	<code>AlertFactory.new_variable_value(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type VI:PROC:NEW-VALUE

syntax	<code>AlertFactory.device_state_change(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:DEV-STATE-CHANGE

syntax	<code>AlertFactory.configuration_change(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:CONFIGURATION-CHANGE

syntax	<code>AlertFactory.malicious_protocol(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MALICIOUS-PROTOCOL

syntax	<code>AlertFactory.weak_encryption(<reason>)</code>
parameters	<ul style="list-style-type: none"> • reason: a message to be displayed in the alert

description	raise an alert of type SIGN:WEAK-ENCRYPTION
--------------------	---

syntax	AlertFactory.malformed_traffic(<triggerId>, <reason>)
parameters	<ul style="list-style-type: none"> triggerId: identifier of the triggering engine entity reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MALFORMED-TRAFFIC

syntax	AlertFactory.suspicious_time(<triggerId>, <reason>)
parameters	<ul style="list-style-type: none"> triggerId: identifier of the triggering engine entity reason: a message to be displayed in the alert
description	raise an alert of type SIGN:SUSP-TIME

syntax	AlertFactory.new_node(<nodeId>)
parameters	<ul style="list-style-type: none"> nodeId: identifier of the node
description	raise an alert of type VI:NEW-NODE

syntax	AlertFactory.new_target_node(<nodeId>)
parameters	<ul style="list-style-type: none"> nodeId: identifier of the node
description	raise an alert of type VI:NEW-NODE:TARGET

syntax	AlertFactory.new_node_malicious_ip(<nodeId>, <threatName>)
parameters	<ul style="list-style-type: none"> nodeId: identifier of the node threatName: the name of the threat
description	raise an alert of type VI:NEW-NODE:MALICIOUS-IP

syntax	AlertFactory.new_mac_vendor(<nodeId>, <macAddress>, <reason>)
parameters	<ul style="list-style-type: none"> nodeId: identifier of the node macAddress: <i>media access control (MAC)</i> Address reason: a message to be displayed in the alert
description	raise an alert of type VI:GLOBAL:NEW-MAC-VENDOR

syntax	<code>AlertFactory.new_mac(<nodeId>, <macAddress>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • nodeId: identifier of the node • macAddress: MAC Address • reason: a message to be displayed in the alert
description	raise an alert of type VI:NEW-MAC

syntax	<code>AlertFactory.malicious_domain(<domain>, <threatName>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • domain: the malicious domain • threatName: the name of the threat • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MALICIOUS-DOMAIN

syntax	<code>AlertFactory.malicious_url(<url>, <threatName>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • url: the malicious uniform resource locator (URL) • threatName: the name of the threat • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MALICIOUS-URL

syntax	<code>AlertFactory.configuration_mismatch(<nodeId>, <triggerId>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • nodeId: identifier of the node • triggerId: identifier of the triggering engine entity • reason: a message to be displayed in the alert
description	raise an alert of type VI:CONF-MISMATCH

syntax	<code>AlertFactory.multiple_ot_device_reservations(<sNodeId>, <dNodeId>, <protocolId>, <bpfFilter>, <protocolType>, <reason>)</code>
---------------	--

parameters	<ul style="list-style-type: none"> • sNodeId: identifier of the source node • dNodeId: identifier of the destination node • protocolId: identifier of the protocol • bpfFilter: Berkeley Packet Filter (BPF) filter • protocolType: type of the protocol according to the ProtocolType type • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MULTIPLE-OT_DEVICE-RESERVATIONS

syntax	<code>AlertFactory.multiple_unsuccessful_logins(<sNodeId>, <dNodeId>, <protocolId>, <bpfFilter>, <protocolType>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • sNodeId: identifier of the source node • dstNodeId: identifier of the destination node • protocolId: identifier of the protocol • bpfFilter: BPF filter • protocolType: type of the protocol according to the ProtocolType type • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MULTIPLE-UNSUCCESSFUL-LOGINS

syntax	<code>AlertFactory.generic_event(<triggerId>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • triggerId: identifier of the triggering engine entity • reason: a message to be displayed in the alert
description	raise an alert of type GENERIC:EVENT

syntax	<code>AlertFactory.multiple_access_denied(<sNodeId>, <dNodeId>, <protocolId>, <bpfFilter>, <protocolType>, <reason>)</code>
parameters	<ul style="list-style-type: none"> • sNodeId: identifier of the source node • dNodeId: identifier of the destination node • protocolId: identifier of the protocol • bpfFilter: BPF filter • protocolType: type of the protocol according to the ProtocolType type • reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MULTIPLE-ACCESS-DENIED

syntax	<code>AlertFactory.protocol_injection(<reason>)</code>
---------------	--

parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:PROTOCOL-INJECTION

syntax	<code>send_alert_malformed_packet(<reason>)</code>
parameters	<ul style="list-style-type: none"> reason: a message to be displayed in the alert
description	raise an alert of type SIGN:MALFORMED-TRAFFIC

syntax	<code>notify_captured_url(<clientId>, <serverNodeId>, <url>, <user>, <operation>, <size>, <properties>)</code>
parameters	<ul style="list-style-type: none"> clientId: node <i>ID</i> of the client serverNodeId: node <i>ID</i> of the server url: the captured <i>URL</i> to notify user: optional string parameter for the user related to the captured <i>URL</i> operation: optional string parameter describing the operation size: optional integer parameter reporting the size in bytes transferred when accessing the <i>URL</i> properties: optional parameter in <i>JSON</i> format for properties
description	notify a captured <i>URL</i> to the system. Note that captured <i>URLs</i> need to be explicitly enabled by specifying the <code>vi captured_urls enabled</code> configuration setting.

syntax	<code>notify_link_events(<event>, <parameters>)</code>
parameters	<ul style="list-style-type: none"> event: event to notify parameters: <i>JSON</i> dictionary reporting the parameters associated with the event
description	notify a link event to the system. Note that link events need to be explicitly enabled by specifying the <code>vi link_events enabled</code> configuration setting.

syntax	<code>packet.source_id()</code>
description	return the source node id

syntax	<code>packet.destination_id()</code>
description	return the destination node id

syntax	<code>packet.source_ip()</code>
description	return the source node ip

syntax	<code>packet.destination_ip()</code>
description	return the destination node ip

syntax	<code>packet.source_mac()</code>
description	return the source node mac

syntax	<code>packet.destination_mac()</code>
description	return the destination node mac

syntax	<code>packet.source_port()</code>
description	return the source node port

syntax	<code>packet.destination_port()</code>
description	return the destination node port

syntax	<code>packet.is_ip()</code>
description	return true if the packet is an ip packet

syntax	<code>packet.transport_type()</code>
description	return the transport layer type, can be "tcp", "udp", "ethernet", "icmp" or "unknown"

syntax	<code>packet.source_node()</code>
description	returns the source node

syntax	<code>packet.destination_node()</code>
description	returns the destination node

syntax	<code>packet.time()</code>
---------------	----------------------------

description	return the packet time
--------------------	------------------------

syntax	<code>session.set_pending_request_number(<request_id>, <key>, <value>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request key: a number used to separate different values in the same request value: the number to store
description	store a number on the session

syntax	<code>session.read_pending_request_number(<request_id>, <key>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request key: a number used to separate different values in the same request
description	read a number from the session

syntax	<code>session.set_pending_request_string(<request_id>, <key>, <value>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request key: a number used to separate different values in the same request value: the string to store
description	store a string on the session

syntax	<code>session.read_pending_request_string(<request_id>, <key>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request key: a number used to separate different values in the same request
description	read a string from the session

syntax	<code>session.has_pending_request(<request_id>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request
description	return true if there are values stored with the request_id

syntax	<code>session.has_pending_request_value(<request_id>, <key>)</code>
---------------	---

parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request key: a number used to separate different values in the same request
description	return true if there are values stored with the request_id and key

syntax	<code>session.close_pending_request(<request_id>)</code>
parameters	<ul style="list-style-type: none"> request_id: a number used to uniquely identify the request
description	close the pending request and delete the associated data

syntax	<code>log_d(<msg>)</code>
parameters	<ul style="list-style-type: none"> msg: the message to log
description	log a debug message

syntax	<code>log_i(<msg>)</code>
parameters	<ul style="list-style-type: none"> msg: the message to log
description	log an info message

syntax	<code>log_e(<msg>)</code>
parameters	<ul style="list-style-type: none"> msg: the message to log
description	log an error message

Chapter 2. Scriptable variables



Setup

Do this procedure to add a script for custom variables correlation.

Procedure

1. Copy the [Lua](#) script in `/data/scriptable_variables`
2. Configure Guardian with this rule `conf.user configure vi scriptable-variable script <script_name>` in [CLI](#) (<script_name> is the name of the file including the extension)
3. Execute `service n2osids stop`, the [IDS](#) process will be restarted automatically.

What to do next

It is advised that after the [IDS](#) process gets restarted, the corresponding log file (`n2os_ids.log`) is checked:

- If the script was loaded successfully, an **INFO** log line like the example below will have been output:

```
INFO: ScriptableVariablesScript: Successfully loaded script
(script.lua)
```

- If the script loading has failed, one or more **ERROR** log lines should be present in the log file, providing details on what the problem was.

After the above steps, the new scriptable variables correlation script will be loaded in Guardian and will be offered all variable updates.

Writing a variables correlation script

As is the case for scriptable protocols, variable correlation scripts are written in [Lua](#). For more information on the language, please refer to the official [Lua](#) documentation (<https://www.lua.org/start.html>).

The only requirement for variable correlation scripts is that a global function with name `on_receive_variable` exists. Thus, the basic structure of a variable correlation script is:

```
function on_receive_variable(node_id, namespace, name, data_value,
    is_from_config)
end
```

Function `on_receive_variable` will be invoked on every variable update that takes place within Guardian. On every invocation, this function will receive 5 arguments which will provide information on the new variable value and its context:

- `node_id`: The identity of the node, to which the variable belongs.
- `namespace`: Identifier of the variable container, also known as [remote terminal unit \(RTU\) ID](#). When the variable update is coming from a protocol that does not support it, it will be empty or hold a fixed, hardcoded value.
- `name`: The name of the variable being updated.
- `data_value`: The data value is a table value describing the updated variable value. Consult the [API reference \(on page 53\)](#) for the fields exposed by this value.
- `is_from_config`: Variable updates may arrive over traffic or due to configuration commands. This argument will be `true` in case the variable update has been administered via command.

By adding logic in the `on_receive_variable` function, it is possible to make the system respond to variable update events in user specific ways. For example, the script below will raise a Variable Flow Anomaly alert, if the variable with name `ioa-515` raises above a threshold of 200:

```
function on_receive_variable(node_id, namespace, name, data_value,
    is_from_config)
    if name == "ioa-515" and data_value.value > 200 then
        AlertFactory.variable_flow_anomaly("Unexpected variable value!")
    end
end
```

The above example shows how a simple variables correlation script may look like. In practice though such usages are not expected, since Guardian provides other, easier ways for implementing such checks (i.e. assertions).

The scriptable variables correlation becomes a much more interesting mechanism when scripts become stateful: by using top level scope variables, it is possible to maintain state that is consulted and updated across different script invocations. For example, if we want to introduce a check on the first derivative of variable `ioa-515`, the script below may serve as a basis:

```
local values_per_node = {}

function on_receive_variable(node_id, namespace, name, data_value,
is_from_config)
    if name == "ioa-515" then
        if values_per_node[node_id] == nil then
            values_per_node[node_id] = {}
        end
        work_table = values_per_node[node_id]

        work_table.previous = work_table.current
        work_table.current = { time=data_value.time, value=data_value.value }

        if work_table.previous ~= nil then
            delta_position = work_table.current.value -
work_table.previous.value

            -- Time is reported in msec
            delta_time = (work_table.current.time -
work_table.previous.time) / 1000.0

            computed_rate = delta_position / delta_time

            if (computed_rate > 1.0) then
                AlertFactory.variable_flow_anomaly("Derivative increased
above threshold!")
            end
        end
    end
end
```

Notes on above script:

- The `local values_per_node = {}` statement initializes a table on top level script scope. This table is maintained across script invocations and is used to keep the two last values of variable `ioa-515` per node.
- In [Lua](#), accessing fields that don't exist does not raise an error, but simply returns value `nil`. Thus the statement `work_table.previous = work_table.current` can be invoked regardless if there exists a current value or not.
- In production level scripts, it would be necessary to check whether the current and previous times are identical, in order to avoid dividing by zero.

API reference

The capabilities available for scriptable variables correlation is a subset of those available for scriptable protocols. This section enumerates all available data types or functions; for more details on them consult the [Scriptable Protocols API Reference \(on page 24\)](#).

Available Lua libraries

Same as for scriptable protocols.

Data types

- `DataValue`
- `DataCause`
- `DataType`

Functions

- All `AlertFactory` functions
- All `log_*` functions



Chapter 3. OpenAPI



OpenAPI

This section describes OpenAPI implementation, which consists of a Hypertext Transfer Protocol (HTTP) endpoint for executing custom queries.

**Important:**

Queries and exports permission is needed to query all OpenAPI endpoints. In addition, there are further Role-Based Access Control (RBAC) permissions that restrict [API](#) queries to specific tables.

OpenAPI methods that change sensor data produce audit logs. For example, this happens when a new user is added through OpenAPI or when an alert is acknowledged. By default, read-only operations do not produce audit logs. It is possible to change this behavior and have GET OpenAPI methods produce audit logs by specifying the following [CLI](#) command:

```
conf.user configure open_api audit get enabled true
```

Setup

To perform a call to the endpoint you need to pass authentication credentials as headers, the examples provided use [Postman](#), an [HTTP](#) client.

Remember to use your Nozomi Networks Solution's web interface [internet protocol \(IP\)](#) address instead of the example one.

Basic authentication

Nozomi Networks suggests to create dedicated users for OpenAPI usage, with minimal permissions necessary to access the required data sources.

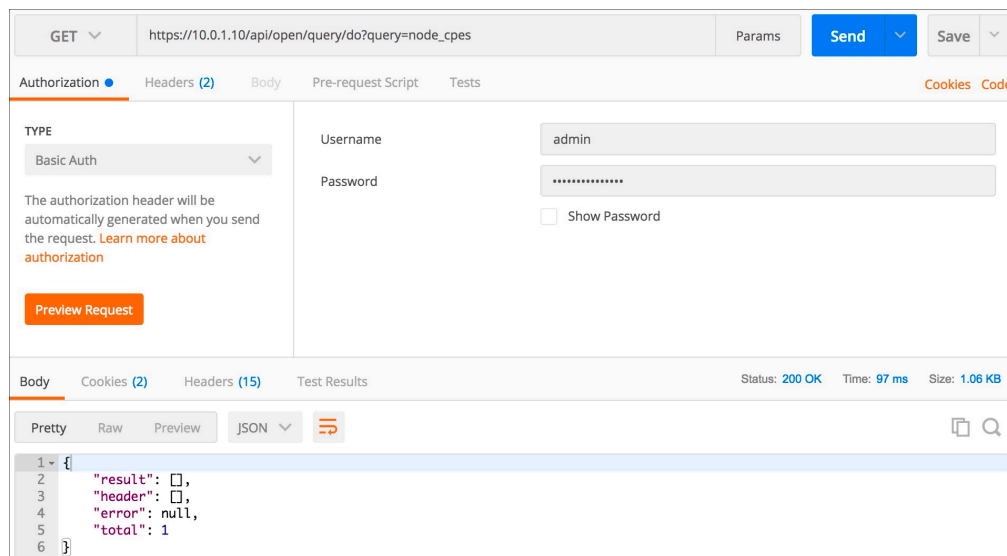


Figure 1. How to perform an authenticated call

Token authentication

As an alternative to basic authentication, use OpenAPI keys created from the Web [user interface \(UI\)](#) to sign in. See Chapters 3 and 5 in the N2OS User Manual for instructions on creating an OpenAPI key.



Note:

Only local users can have OpenAPI keys.

Using token authentication is a two step process. First, use the `/api/open/sign_in` endpoint with a valid key to obtain a [JSON web token \(JWT\)](#) token.

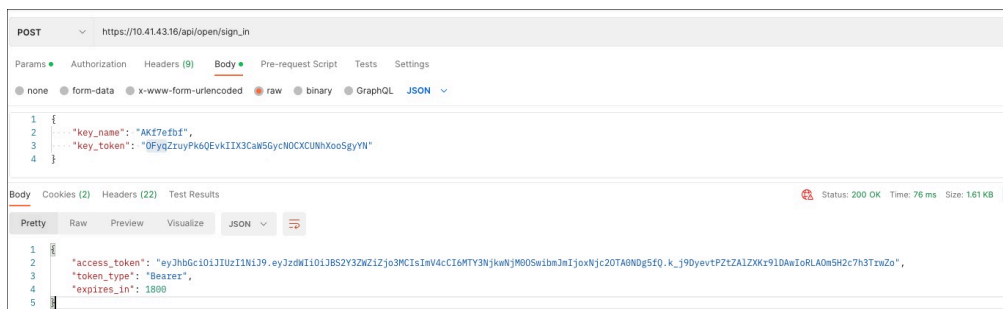


Figure 2. Obtaining a JWT token

Then, use the *JWT* token as bearer token for any successive call to the *API*.

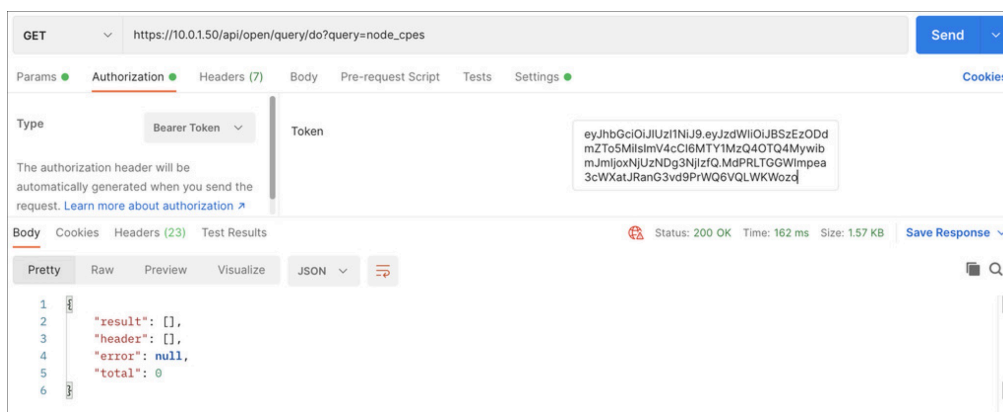


Figure 3. Authentication with bearer token

Remarks

1. The `JWT` token expires 30 minutes after being created. To use `API` for a longer time, request a new token by calling `sign_in` again.
2. Any number of `JWT` tokens can be created.

Errors

With basic authentication, if you fail to provide valid authentication credentials the expected error will be 401 `Unauthorized`, as shown below.

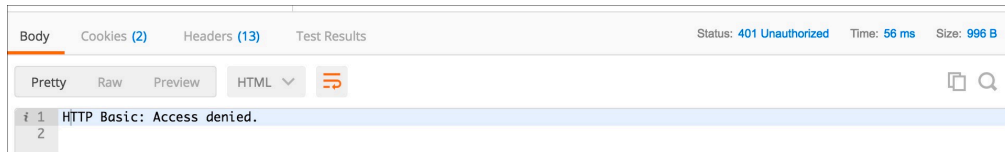


Figure 4. Example of a failed basic authentication call

With token authentication, when an invalid or expired token is used, the expected error will be 401 `Unauthorized`. The body of the response will include a description of the problem, as shown below for the case of an expired token:

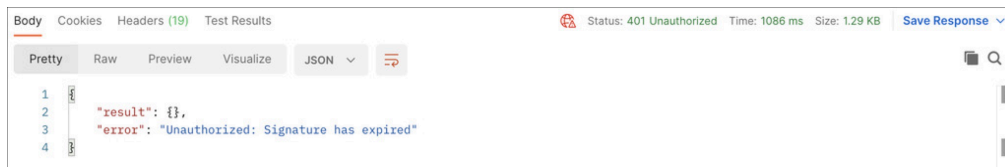


Figure 5. Example of a failed token authentication call

If you ask for a data source that does not exist you will receive a proper message in the error field.

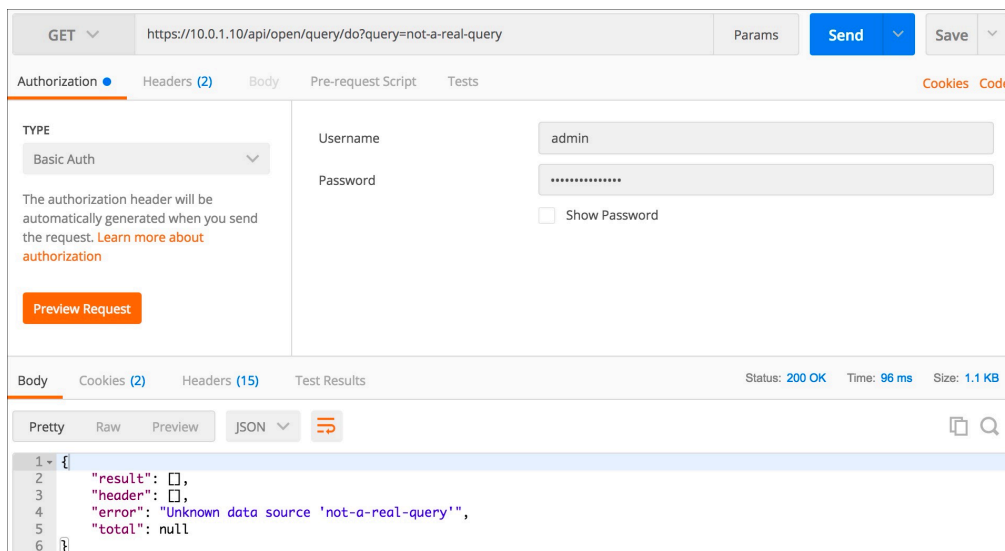
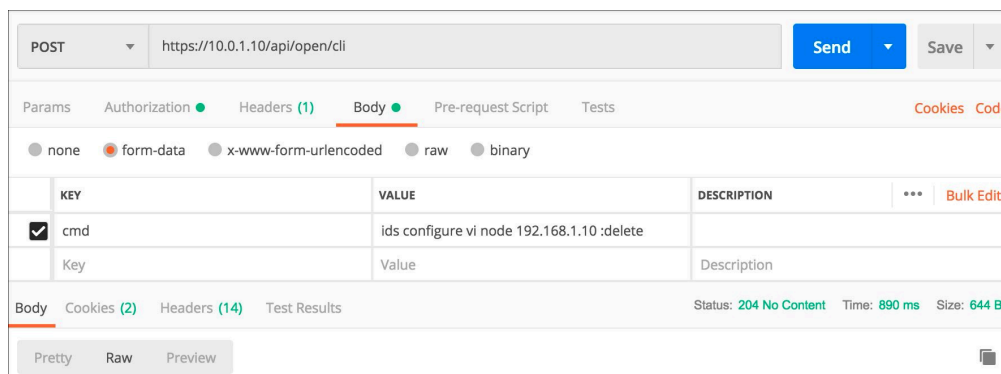


Figure 6. Wrong data source

CLI endpoint

You can apply changes to the system by issuing *CLI* commands over this endpoint. The endpoint is located at `/api/open/cli` and requires to be invoked with the `cmd` parameter with a POST.



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** https://10.0.1.10/api/open/cli
- Buttons:** Send, Save
- Tabs:** Params, Authorization, Headers (1), Body (selected), Pre-request Script, Tests, Cookies, Code
- Body Type:** form-data (selected), none, x-www-form-urlencoded, raw, binary
- Body Content:**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> cmd	ids configure vi node 192.168.1.10 :delete	
Key	Value	Description
- Status:** 204 No Content
- Time:** 890 ms
- Size:** 644 B
- Footer:** Pretty, Raw, Preview

Figure 7. Example of a CLI command

CLI commands allow to change virtually anything inside the system, please refer to the Configuration section of the User Manual for a more complete reference.

Import CSV endpoint

`/api/open/nodes/import` allows you to enrich the information associated to nodes by uploading a *comma-separated value (CSV)* file. Each row affects the nodes matching the specified `ip` field value. When there are no matches, new nodes are created.

Requirements and Restrictions

1. The authenticated user must be in a group with **admin role**
2. Only *CSV* files with a **header** are accepted
3. There must be an `ip` column
4. In addition to `ip`, only the fields listed below and custom fields are considered. Every other provided field will be ignored. If you need to provide values for custom fields, please make sure that the names of these custom fields have been already created.

label
firmware_version
vendor
product_name
serial_number
os
mac_address
type

Example of *CSV* file

```
ip,label,firmware_version,vendor,product_name,serial_number,os,mac_address,type
192.168.1.57,node 57,1.2.2,ACME,ACME Product 0,abcdefg,Windows XP
SP3,00:00:00:11:11:11,computer
```

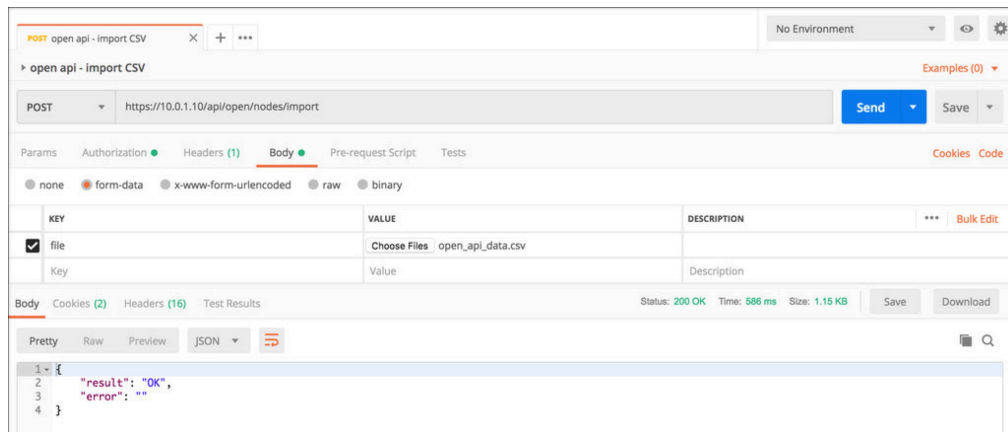


Figure 8. Example of the request

Import JSON endpoint

`/api/open/nodes/import_from_json` allows you to enrich the information associated to nodes. The provided information affects the nodes matching the specified `ip` field value. When there are no matches, new nodes are created.

Requirements and Restrictions

1. The authenticated user must be in a group with **admin role**
2. The input must be a [JSON](#) dictionary containing a `nodes` key whose value is an array of nodes information
3. Nodes must have a value for the `ip` field
4. In addition to `ip`, only the fields listed below and custom fields are considered. Every other provided field will be ignored. If you need to provide values for custom fields, please make sure that the names of these custom fields have been already created.

label
firmware_version
vendor
product_name
serial_number
os
mac_address
type

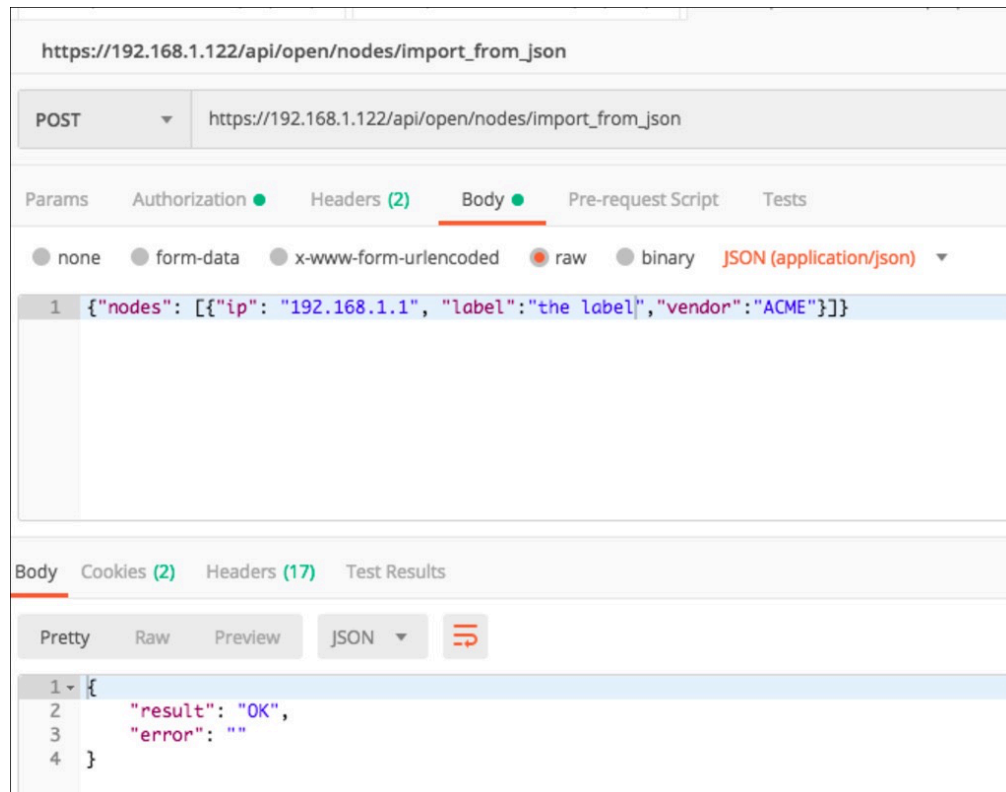


Figure 9. Example of the request

Alerts endpoint

A POST to `/api/open/alerts/close` request lets you to close a group of alerts passed as a json list of ids in the body of the request. You must also pass as parameter the `close_action` field containing `delete_rules` or `learn_rules` in case you want to close alerts as security or as change.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The input data must be a [JSON](#) dictionary containing a `ids` key whose value must be an array of alert 'id' and a 'close_action' constant field.
3. In case the request body does not adhere to the format described above the call returns a 422 error.
4. In case the request is well formed, the result will contain the id of the job in charge of the task. You can monitor the status of the job via the `alerts/close/status/:id` [API](#).

```
{
  "ids": ["uuid"],
  "close_action": "learn_rules"
}
```

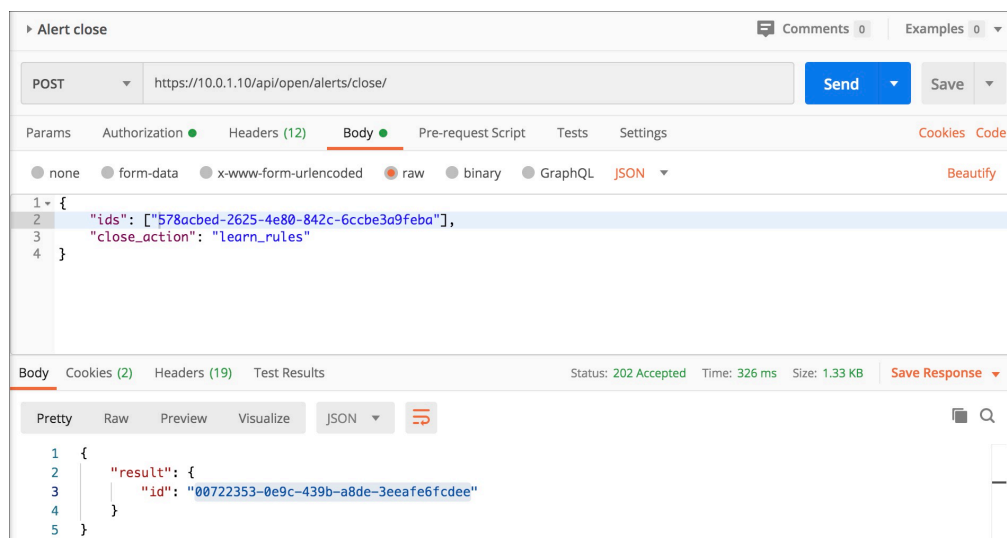


Figure 10. Example of alerts close request

A GET to `/api/open/alerts/close/status/:id` request lets you to get the status of a job in charge of a close alerts task.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. As last parameter of the path you need to specify the id of the job returned by the `alerts/close` [API](#).
3. The result will contain the status of the job, which can have one of the following values: `SUCCESS`, `PENDING` or `FAIL`
4. In case of `FAIL` status, the error field will report the error reason.

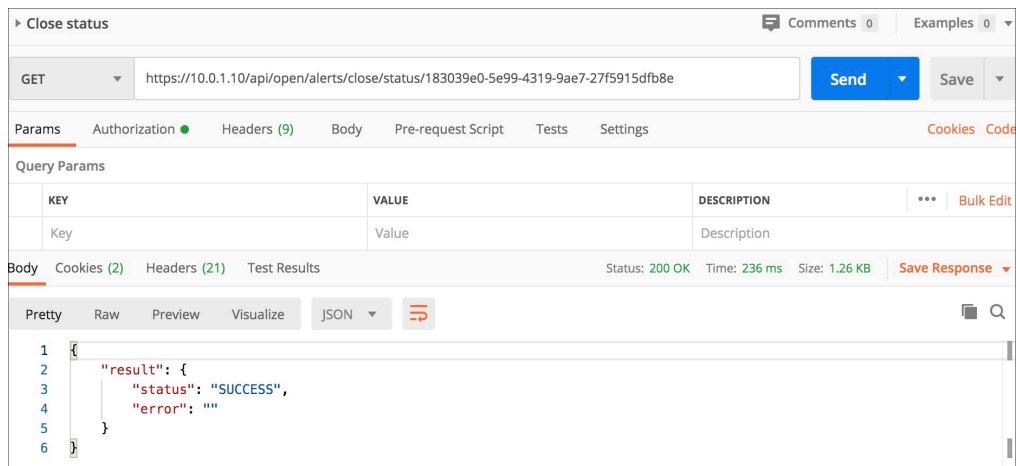


Figure 11. Example of `alerts/close/status/:id` request

A POST to `/api/open/alerts/ack` request lets you to ack/un-ack a group of alerts passed as a json list of `id/ack_status` pairs in the body of the request.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The input data must be a [JSON](#) dictionary containing a `data` key whose value should be an array of pairs with an alert 'id' and an 'ack' field. Ack can be true or false.
3. In case the request body does not adhere to the format described above the call returns a 422 error.
4. In case the request is well formed, the result will contain the id of the job in charge of the task. You can monitor the status of the job via the `alerts/ack/status/:id` [API](#).

```
{
  "data": [
    {
      "id": "uuid",
      "ack": true
    }
  ]
}
```

```
}

```

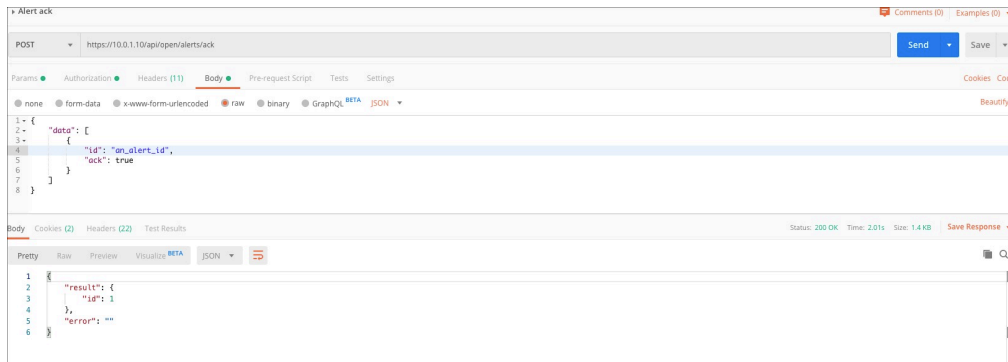


Figure 12. Example of alerts/ack request

A GET to `/api/open/alerts/ack/status/:id` request lets you to get the status of a job in charge of a ack/un-ack alerts task.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. As last parameter of the path you need to specify the id of the job returned by the `alerts/ack` [API](#).
3. The result will contain the status of the job, which can have one of the following values: **SUCCESS**, **PENDING** or **FAIL**
4. In case of **FAIL** status, the error field will report the error reason.

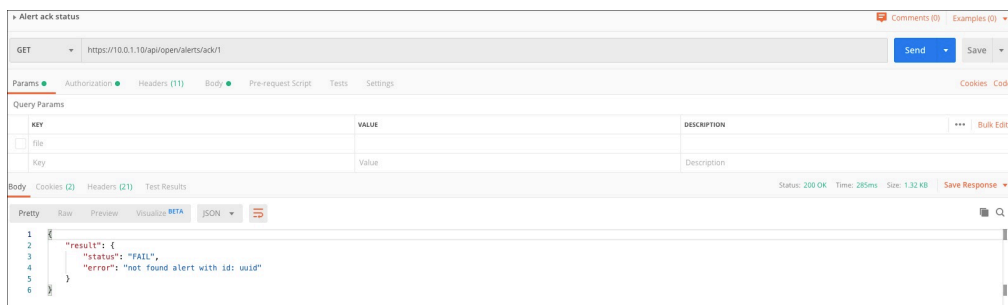


Figure 13. Example of alerts/ack/status/:id request

A GET to `/api/open/alerts/all` request lets you to get the IDs of alerts matching a condition. You can specify a filter query in the `query` parameter and an additional parameter named `has_trace` to get the status of the corresponding trace.

Requirements and Restrictions

1. The authenticated user has to belong to a group having **admin role** or with **Alerts section** enabled.
2. The `query` parameter should be in Nozomi Networks Query Language format, where the table name is implicit, i.e. `alerts`.

3. The `has_trace` parameter type is boolean.
4. If no alert matches the specified conditions, a 404 error will be returned.

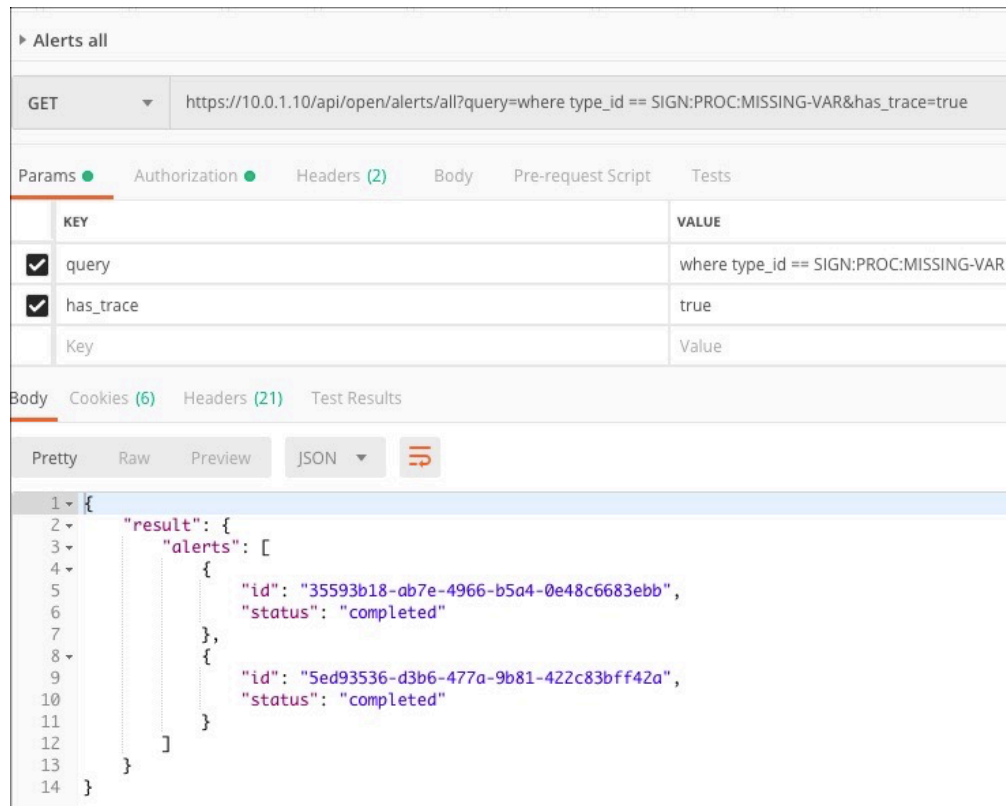


Figure 14. Example of alerts/all request

A GET to `/api/open/alerts/:id/trace` request lets you to get a file containing the trace of the alert, whose id is specified as a parameter.

Requirements and Restrictions

1. The authenticated user has to belong to a group having **admin role** or with **Alerts section** enabled.
2. The alert id should be passed in the path.
3. If the alert does not exist, a **422** error is returned.
4. In case there is no trace for the specified alert, a **404** error will be returned.



Figure 15. Example of alerts/:id/trace request

A POST to `/api/open/alerts/import` request lets you to import alerts by passing attributes in [JSON](#) format in the body of the request.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Alerts section** enabled.
2. The alerts information should be provided in a [JSON](#) array named `alerts`.
3. In case the request body does not adhere to the format described above the call returns a 422 error.
4. In case the request is well formed, the result can contain the validation outcome for errors regarding mandatory fields and warnings for fields that are potentially missing.
5. If one or more alerts are passing the validation, the result will also contain the id of the job in charge of importing the alerts. You can monitor the status of the job via the `alerts/import-status` [API](#).

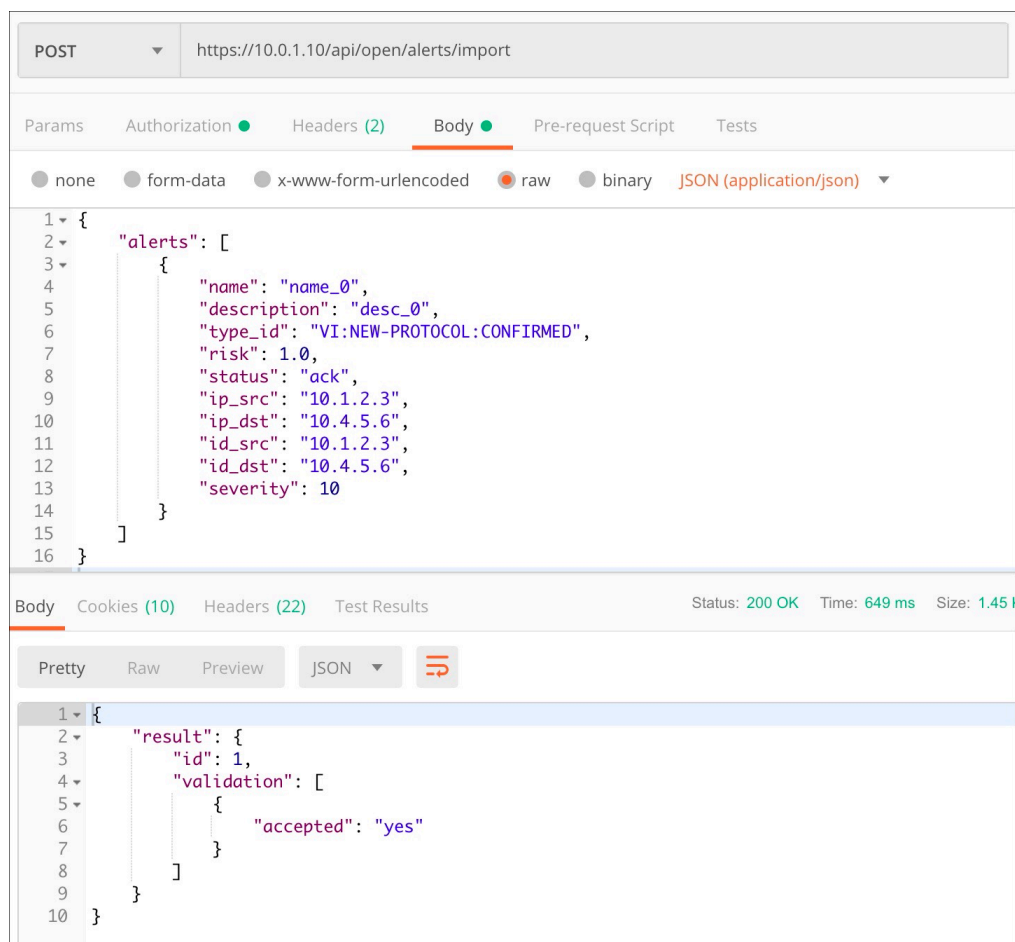


Figure 16. Example of alerts/import request

A GET to `/api/open/alerts/import-status` request lets you to get the status of a job in charge of importing alerts.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. You need to specify the id of the job returned by the `alerts/import` [API](#) in the `id` parameter.
3. The result will contain the status of the job, which can have one of the following values: `SUCCESS`, `PENDING` or `FAIL`
4. In case of `FAIL` status, the error field will report the error reason.

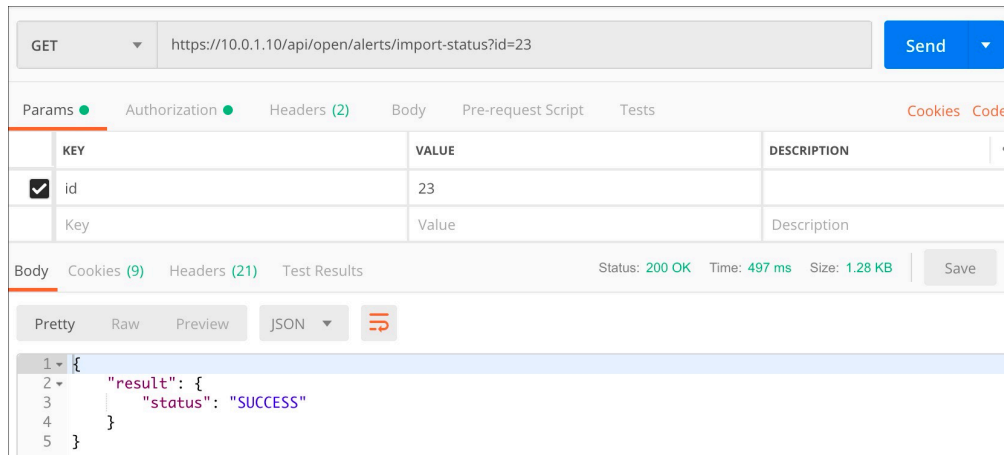


Figure 17. Example of `alerts/import-status` request

A PATCH to `/api/open/alerts/:id` lets you to update the alert's note.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Allow alert ack and edit** enabled.
2. The `note` must be passed as a query parameter or in the [JSON](#) body.
3. The result will contain the [JSON](#) representation of the Alert updated.

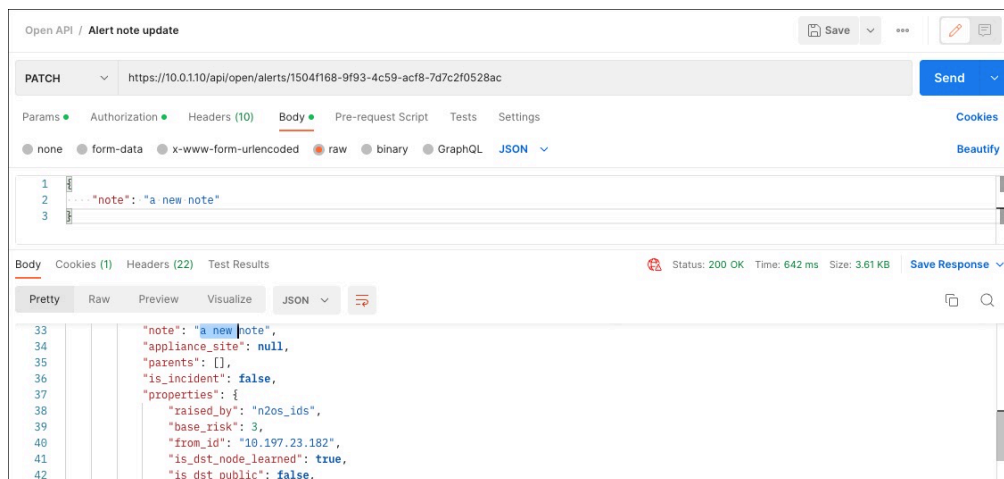


Figure 18. Example of alert's note update request

Trace endpoint

Filter traces

A GET to `/api/open/traces/all` request allows you to get traces matching a condition. You can specify a filter query in the **query** parameter, which is a standard N2OS query condition, applied to the `trace_requests` data source. You have to specify the **operation** parameter defining the requested operation. So far the only allowed value for the **operation** parameter is **download**.

Requirements and Restrictions

1. The authenticated user must be in a group having admin role permission.
2. As a result you will get a file containing the trace or the traces filtered according to the specified condition.
3. If the trace is still in progress or it is not found, a 422 error with a proper reason string will be returned.

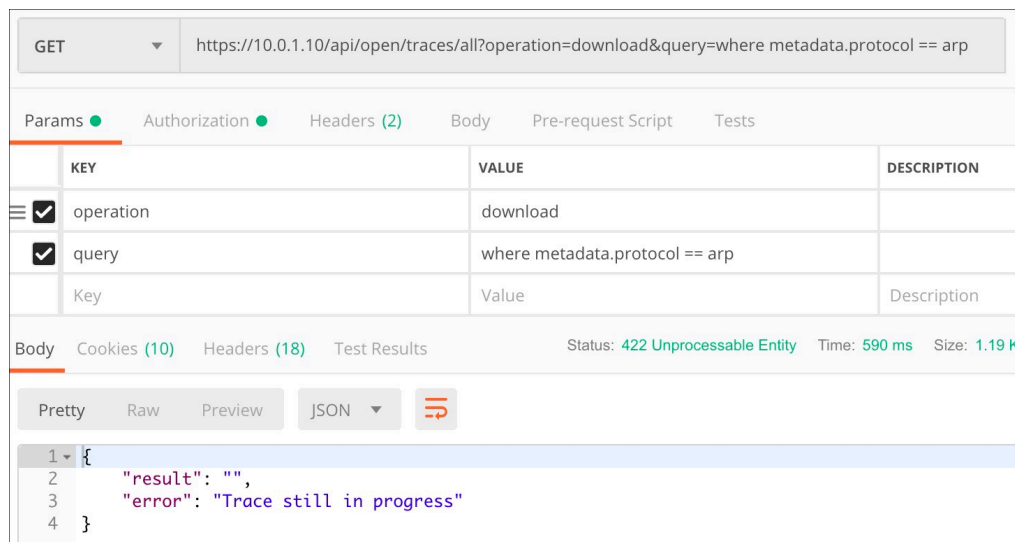


Figure 19. Example of traces/all request

BPF filter

A GET to `/api/open/traces/bpf-filter` request allows you to select traces using a **BPF** filter. This call returns a `job_id`, while the actual disk search is performed asynchronously. The search will return a list of the first **packet capture (pcap)** traces that match the filter. The maximum number of **pcap** traces is 50 by default and can be configured with the `open_api bpf_filter traces_limit` setting. There can't be more than a limited number of concurrent **BPF** trace searches at a time. This number is 2 by default and can be configured with the `open_api bpf_filter max_concurrent_searches` setting.

Requirements and Restrictions

1. The authenticated user must be in a group having admin role permission.

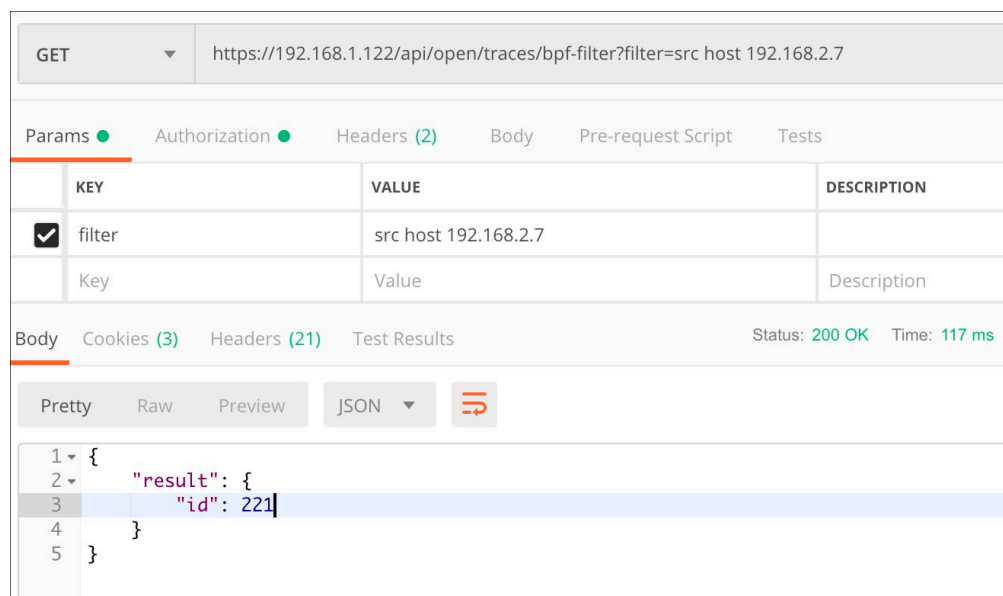


Figure 20. Example of a BPF filter request

A GET to `/api/open/traces/bpf-filter-status` request allows you to get the status of a job in charge of looking for traces given a [BPF](#) filter.

Requirements and Restrictions

1. The authenticated user must be in a group having admin role permission.
2. You need to specify the id of the job returned by the [traces/bpf_filter API](#) in the id parameter.
3. The result will contain the status of the job, which can have one of the following values: SUCCESS, PENDING or FAIL.
4. In case of FAIL status, the error field will report the error reason.

The screenshot displays a REST client interface for a GET request to `https://192.168.1.122/api/open/traces/bpf-filter-status?id=221`. The 'Params' tab is active, showing a single parameter `id` with the value `221`. The 'Body' tab is selected, showing the response in JSON format. The response status is `200 OK`, with a time of `126 ms` and a size of `2.92 KB`.

The response body is a JSON object:

```
{
  "result": {
    "status": "SUCCESS",
    "traces": [
      {
        "pathname": "/data/traces/05775_46be89e6-65e2-495c-a500-9b270d41776c.pcap",
        "trace_id": "46be89e6-65e2-495c-a500-9b270d41776c",
        "packets": "5"
      },
      {
        "pathname": "/data/traces/05772_14b7bfbd-5e7b-400f-8369-9d625c6e0e34.pcap",
        "trace_id": "14b7bfbd-5e7b-400f-8369-9d625c6e0e34",
        "packets": "5"
      }
    ]
  }
}
```

Figure 21. Example of traces/bpf-filter-status request

Users endpoint

A GET to `/api/open/users` allows you to get a list of all the users.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The result contains the list of all users.
3. It's possible to use pagination adding `page` and `count` params
4. The `page` param is the number of the page to return, the `count` is the dimension of the page.
5. If `count` is nil or 0 the default value will be 100, if `page` is nil or 0 the request will not be paginated.
6. This api is disabled by default; to enable it add `conf.user configure api users enabled true` in [CLI](#).

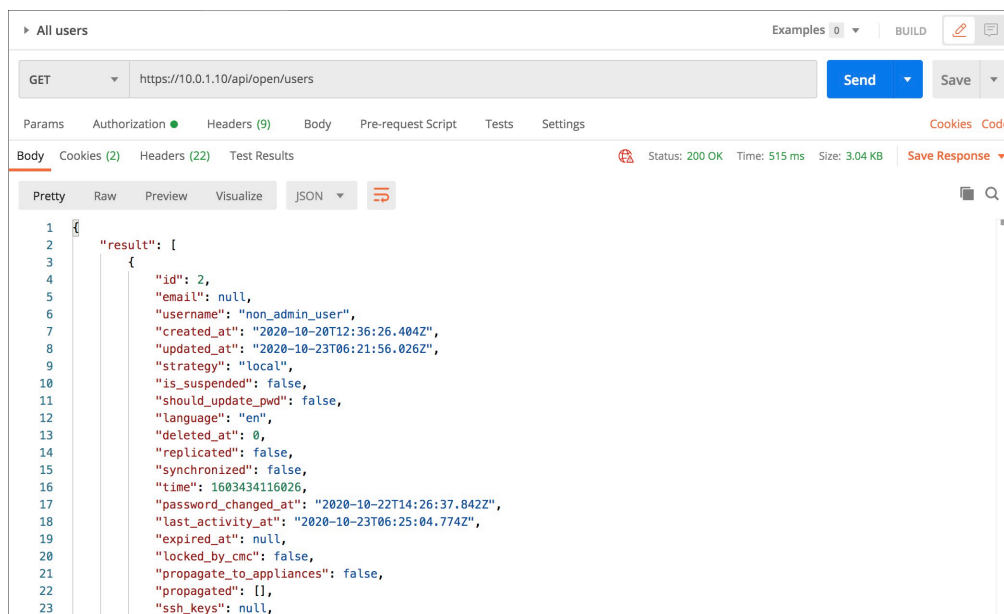


Figure 22. Example of users all request

A GET to `/api/open/user_groups` allows you to get a list of all the user groups.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The result contains the list of all user groups.
3. It's possible to use pagination adding `page` and `count` params
4. The `page` param is the number of the page to return, the `count` is the dimension of the page.
5. If `count` is nil or 0 the default value will be 100, if `page` is nil or 0 the request will not be paginated.

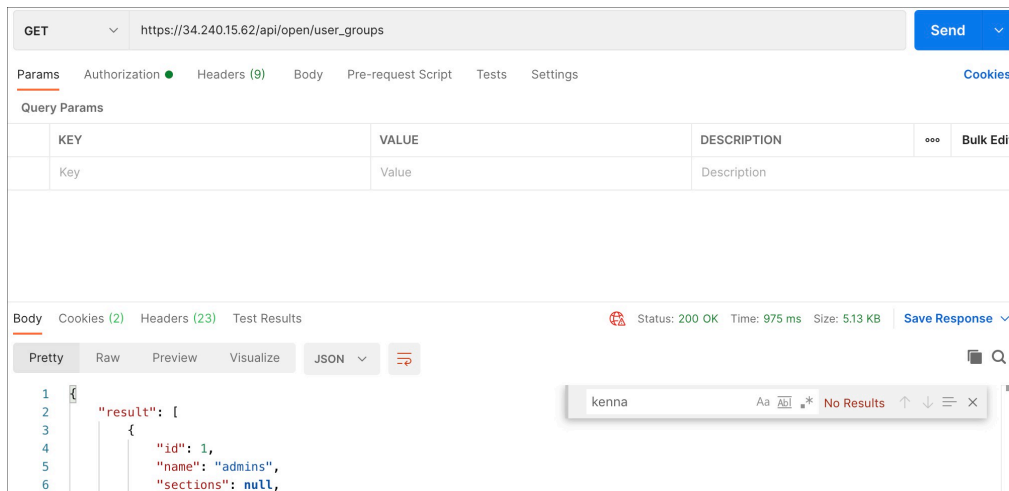


Figure 23. Example of user groups all request

A GET to `/api/open/users/:id` allows you to get the user having the id passed as path parameter.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. As last parameter of the path you need to specify the id of the user.
3. The result will contain the user
4. In case the user with that id is not found you'll get a 404.

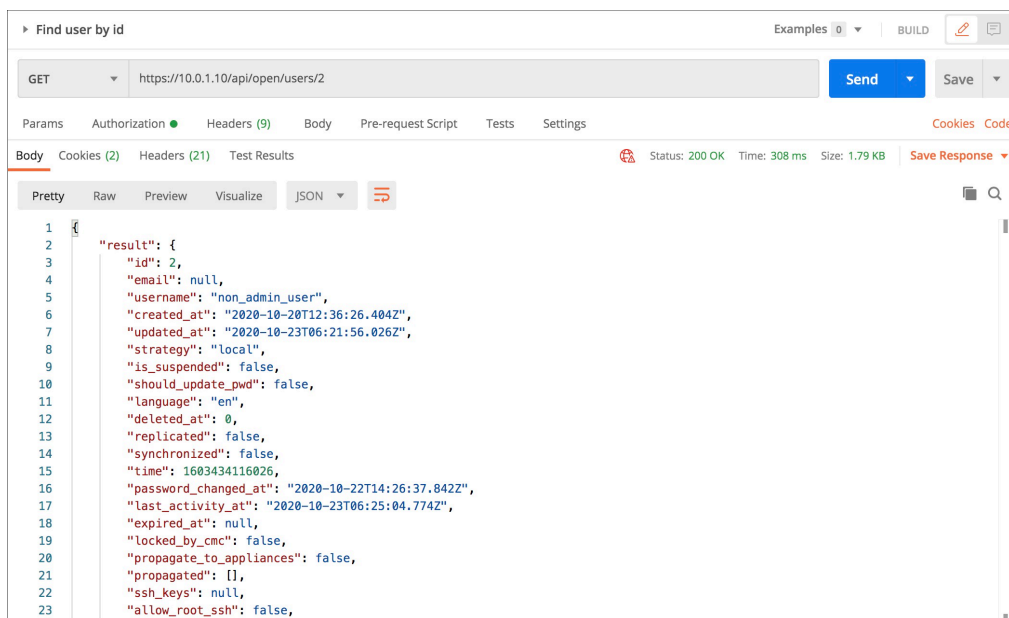


Figure 24. Example of users/:id request

A DELETE to `/api/open/users/:id` allows you to delete the user having the id passed as path parameter.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. As last parameter of the path you need to specify the id of the user.
3. The result will contain the status code **204** for success else the error code
4. In case the user with that id is not found you'll get a 404.

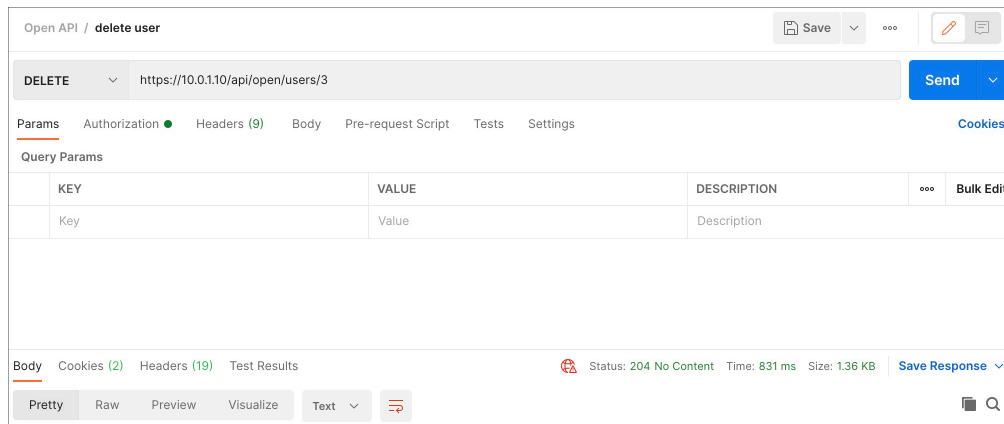


Figure 25. Example of delete users/:id request

A POST to `/api/open/users` allows you to create a new user.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The input must be a **JSON** dictionary containing the user fields properly populated
3. `username` is mandatory and unique.
4. `password` is mandatory and have to respect the password strength rules.
5. `user_group_ids` is mandatory, must contain at least an id of an existing user-group.
6. `strategy` can contain the value "local" or "saml".
7. `is_suspended` is a boolean.
8. `should_update_pwd` true if the user must update the password when log-in.
9. `ssh_keys` is the user **secure shell (SSH)** key if wants to connect via ssh to the instance.
10. `allow_root_ssh` true to allow the user having the ssh_key above to connect via **SSH** to the instance.
11. In case the request is well formed return a 201 response with the id of the user created inside the result.

```
{
  "username": "user_under_test22",
  "password": "aValidP4ss!",
  "user_group_ids": [2],
```

```

        "strategy": "local",
        "is_suspended": false,
        "should_update_pwd": false,
        "ssh_keys": "an_ssh_key",
        "allow_root_ssh": true
    }

```

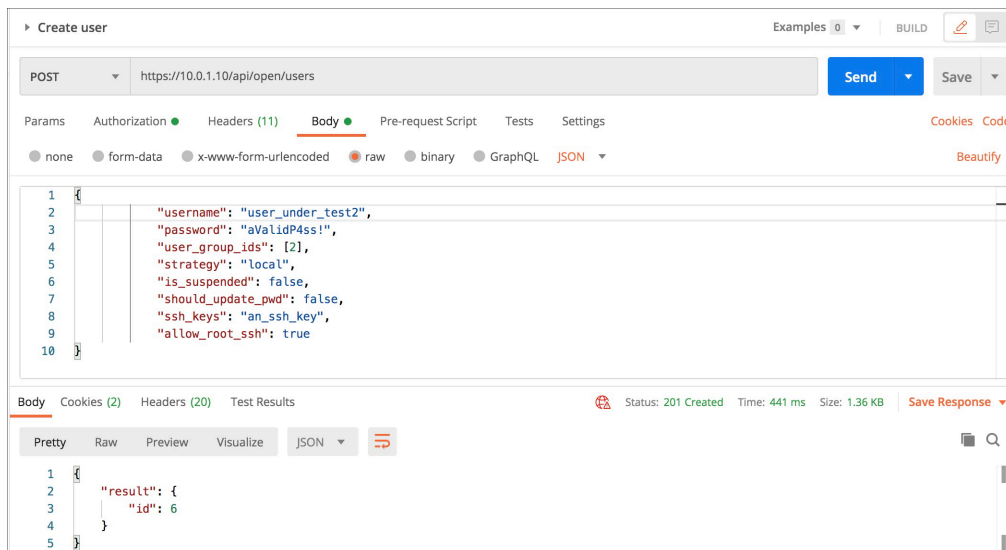


Figure 26. Example of users/ack request

A PUT to `/api/open/users/:id` allows you to update the user with the id passed as path param.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. As last parameter of the path you need to specify the id of the user you want to update.
3. The input must be a [JSON](#) dictionary containing the user field properly populated
4. If the update goes well the call return 204 (No content) response
5. You can't update the password here because updating password is not idempotent so you can't do via PUT.
6. The fields you can update are listed below.
7. `user_group_ids` must contain at least one valid id.

```

{
    "username": "user_under_test22",
    "strategy": "local",

```

```

        "user_group_ids": [1,2],
        "is_suspended": false,
        "should_update_pwd": false,
        "ssh_keys": "a_new_key",
        "allow_root_ssh": true
    }

```

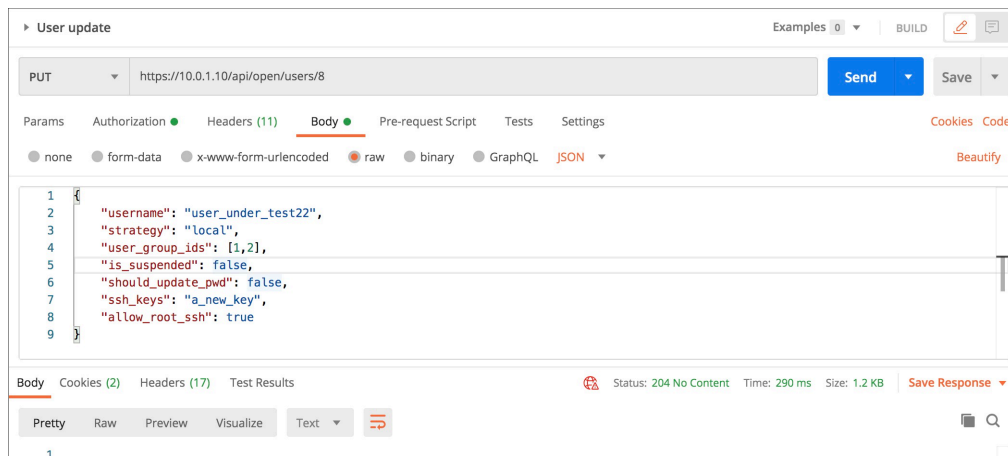


Figure 27. Example of update users/:id request

A PATCH to `/api/open/users/:id/password` allows you to change the password of the user having the id passed as path param.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The user id should be passed in the path.
3. You need to pass the new password in the body.
4. New password must respect the password strength rules.
5. In case the password is valid will be return an empty response with status code 204.

```

{
    "password": "4ValidP4ssw0rd!"
}

```

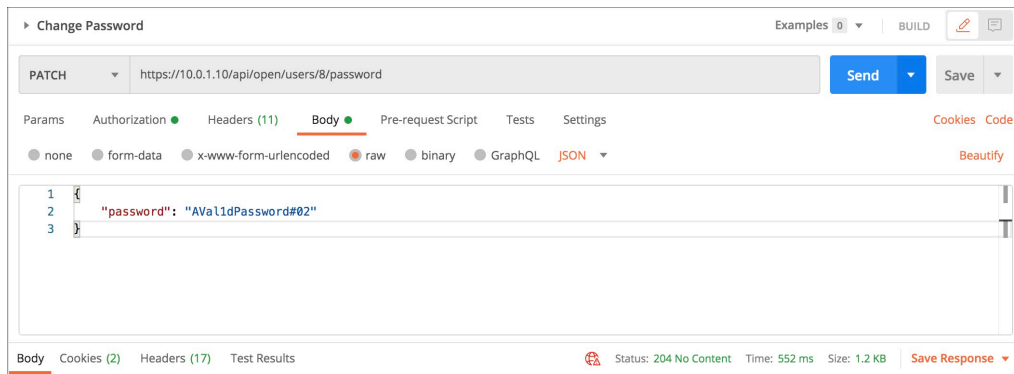


Figure 28. Example of users/:id/password request

PCAPs endpoint

A GET request to `/api/open/pcaps` allows you to get the list of all traces available on the machine.

This endpoint lets you to interact with *pcaps* that have been uploaded to Guardian from the **Upload traces** page of the **System** section.

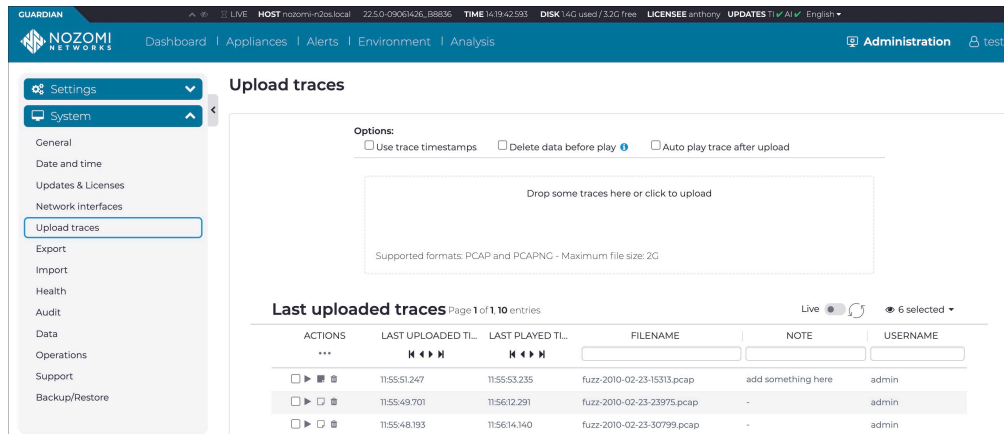


Figure 29. Upload traces page

A GET request to `/api/open/pcaps/:id` allows you to retrieve a given trace.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. In case the request body does not adhere to the format described above, the call returns a 422 error.
3. If you specify an *ID* of a trace that does not exist, the call returns a 404 error.
4. If the request is accepted, the result will contain information on the retrieved trace.

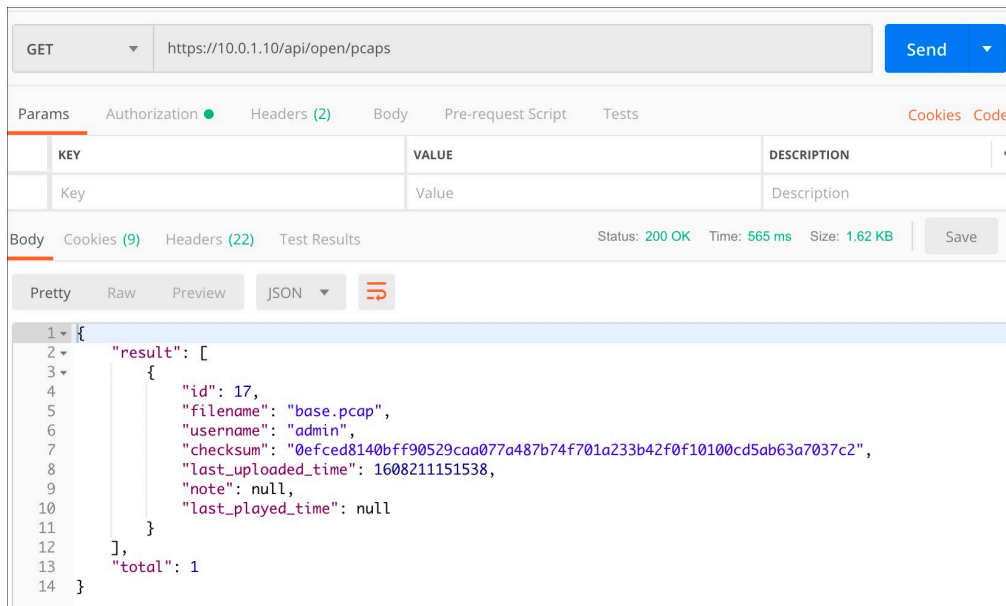


Figure 30. Example of traces get all list request

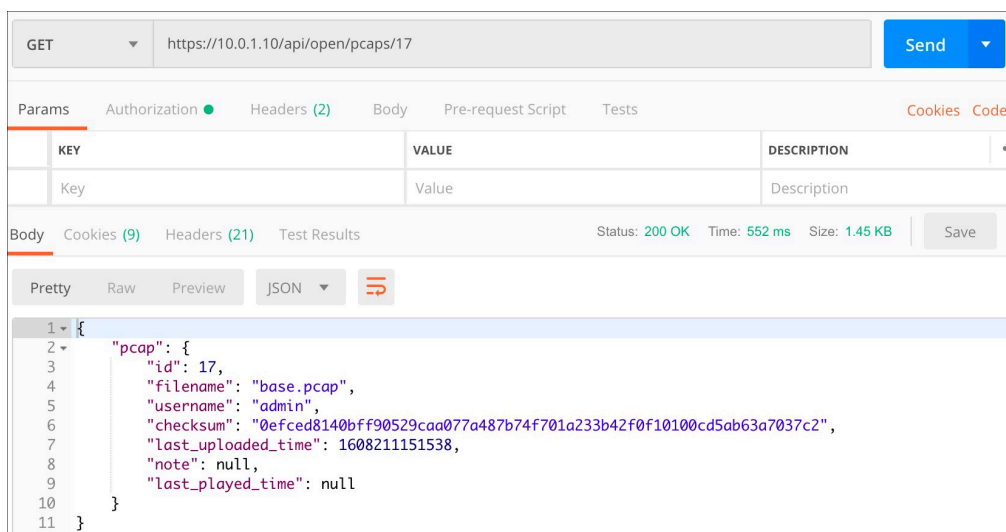


Figure 31. Example of traces get by ID request

A DELETE request to `/api/open/pcaps/:id` allows you to delete a given trace.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. In case the request body does not adhere to the format described above, the call returns a 422 error.
3. If you specify an *ID* of a trace that does not exist, the call returns a 404 error.
4. If the request is accepted, the trace will be deleted.

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (9) Headers (20) Test Results Status: 200 OK Time: 554 ms Size: 1.16 KB

Pretty Raw Preview JSON

Figure 32. Example of trace delete request

A POST request to `/api/open/pcaps/upload` allows you to upload a trace passed as a file in the body of the request.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. The trace should be passed in the `form-data` section of the request body.
3. In case the request body does not adhere to the format described above, the call returns a 422 error.
4. If the file sent in the request is not a valid trace, the call returns a 422 error along with an error reason describing the cause of the validation failure.
5. If the request is accepted, the trace will be uploaded.

POST https://10.0.1.10/api/open/pcaps/upload Send

Params Authorization Headers (2) Body Pre-request Script Tests Cookies Code

none form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> file	base.pcap X	
Key	Value	Description

Body Cookies (9) Headers (21) Test Results Status: 200 OK Time: 1044 ms Size: 1.47 KB Save

Pretty Raw Preview JSON

```
1 {
2   "result": "OK",
3   "error": "",
4   "pcap": {
5     "id": 18,
6     "filename": "base.pcap",
7     "username": "admin",
8     "checksum": "68077ff2a694d3f39b3c520cce7dbe68812bb94e3e596d597f323a6448d1a41a",
9     "last_uploaded_time": 1608211606563,
10    "note": null,
11    "last_played_time": null
12  }
13 }
```

Figure 33. Example of trace upload request

A POST request to `/api/open/pcaps/import` allows you to import a trace file that is already present in the machine.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. The trace file should be present in the `/data/tmp` directory of the machine.
3. The `filename` parameter of the request should contain the name of the trace file.
4. In case the request body does not adhere to the format described above, the call returns a 422 error.
5. If the trace file is not a valid trace, the call returns a 422 error along with an error reason describing the cause of the validation failure.
6. If the request is accepted, the trace will be uploaded.

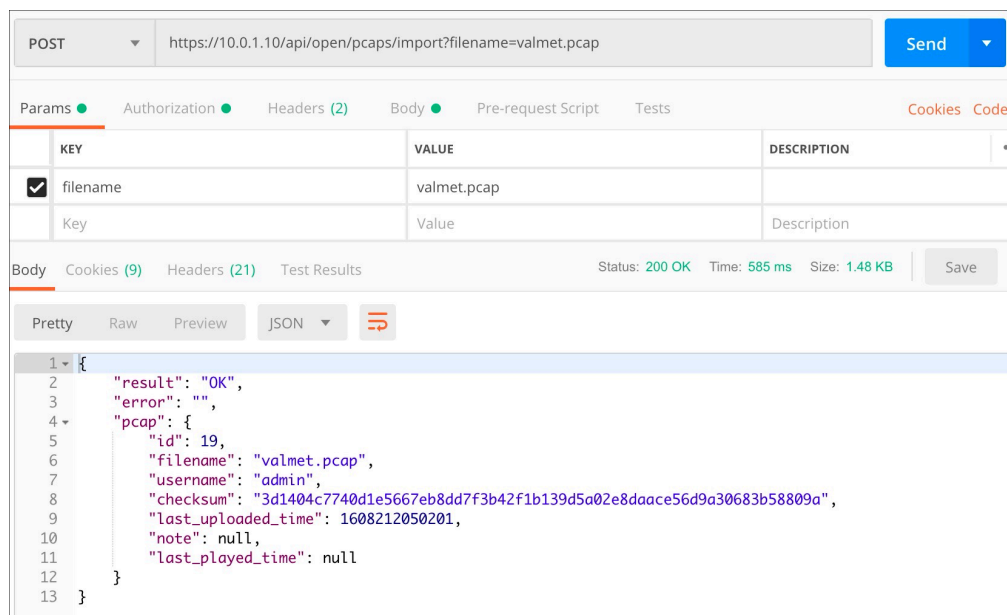


Figure 34. Example of trace import request

A PATCH request to `/api/open/pcaps` allows you to replay a trace that has been previously uploaded or imported.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. The trace should be present in the list of the available traces returned by the GET request to `/api/open/pcaps`.
3. The `id` parameter of the request should contain the *ID* of the trace.
4. The `use_packet_time` boolean parameter should be set to `true` if you want to use the time of the packets; `false` otherwise.

5. The `data_to_reset_before_play` parameter should be set to `{}` if you do not want to reset data before playing the trace. Otherwise, you need to specify a [JSON](#) dictionary with the sections you want to reset, for example `{"alerts": true, "vi": true}`. The list of all available sections is the following:
- `alerts_data`
 - `assertions`
 - `learning`
 - `network_data`
 - `process_data`
 - `queries`
 - `smart_polling_data`
 - `timemachine_data`
 - `traces_data`
 - `vi_data`
 - `vulnerability_data`

The list above reflects the options available for Data reset in the [UI](#).

6. In case the request body does not adhere to the format described above, the call returns a 422 error.
7. If you specify an [ID](#) of a trace that does not exist, the call returns a 404 error.
8. If the request is accepted, the trace will be replayed.

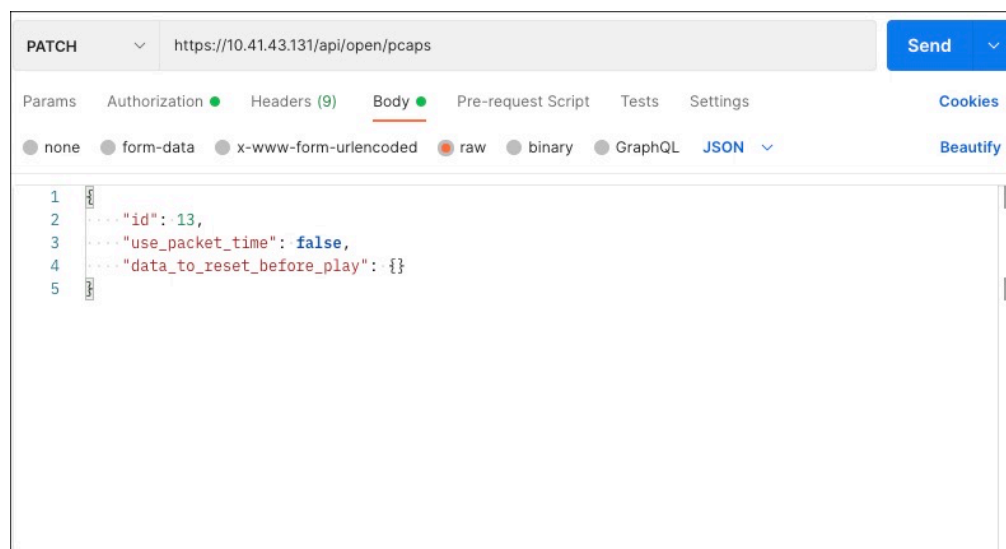


Figure 35. Example of trace replay request

A PATCH request to `/api/open/pcaps/note` allows you to change the note field of a trace.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or with **Upload traces** section enabled.
2. The trace should be present in the list of the available traces returned by the GET request to `/api/open/pcaps`.

3. The `id` parameter of the request should contain the `ID` of the trace.
4. The `note` parameter of the request should contain the text you want to change.
5. In case the request body does not adhere to the format described above, the call returns a 422 error.
6. If the request is accepted, the note will be changed.

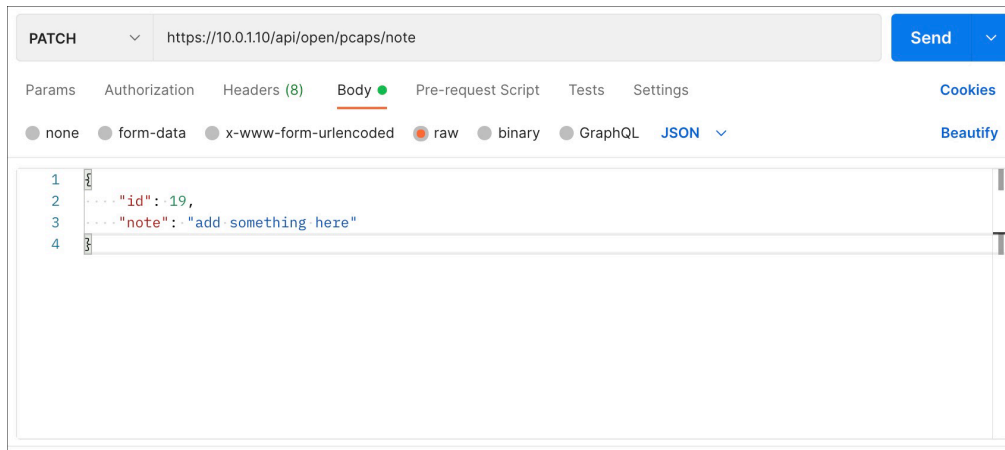


Figure 36. Example of trace note request

Reports endpoint

A GET to `/api/open/reports` allows you to get a list of all the reports generated.

Requirements and Restrictions

1. A user having the permission to execute api.
2. The result contains the list of all the reports.
3. It's possible to use pagination adding `page` and `count` params
4. The `page` param is the number of the page to return, the `count` is the dimension of the page.
5. If count is nil or 0 the default value will be 100, if page is nil or 0 the request will not be paginated.
6. You can filter the result passing a `template_name` as query param having value the report templates name you want filtering on.

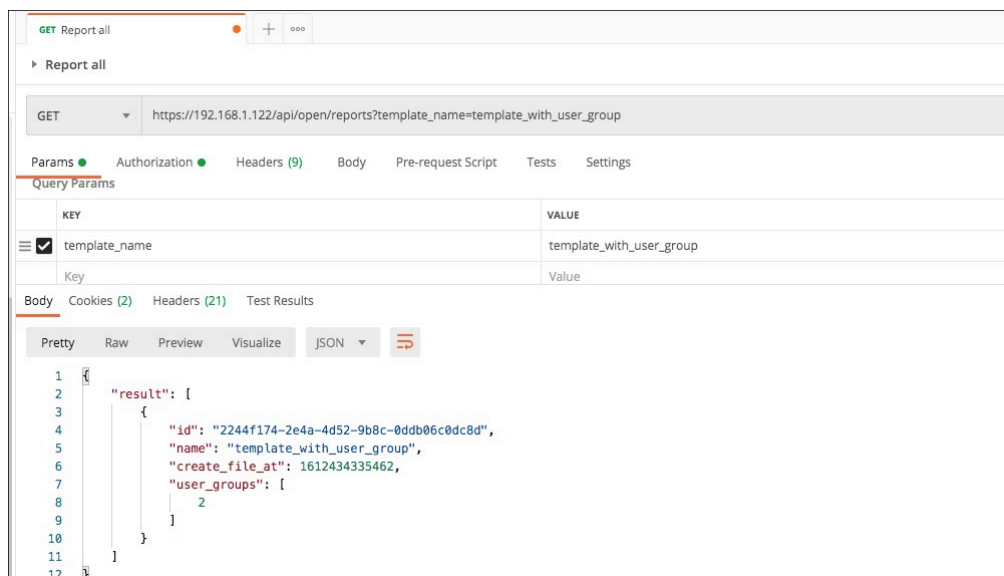


Figure 37. Example of reports all request

A GET to `/api/open/reports/:id` allows you to get the report having the id passed as path parameter.

Requirements and Restrictions

1. A user having the permission to execute api.
2. As last parameter of the path you need to specify the id of the report.
3. The result will contain the report.
4. In case the report with that id is not found you'll get a 404.

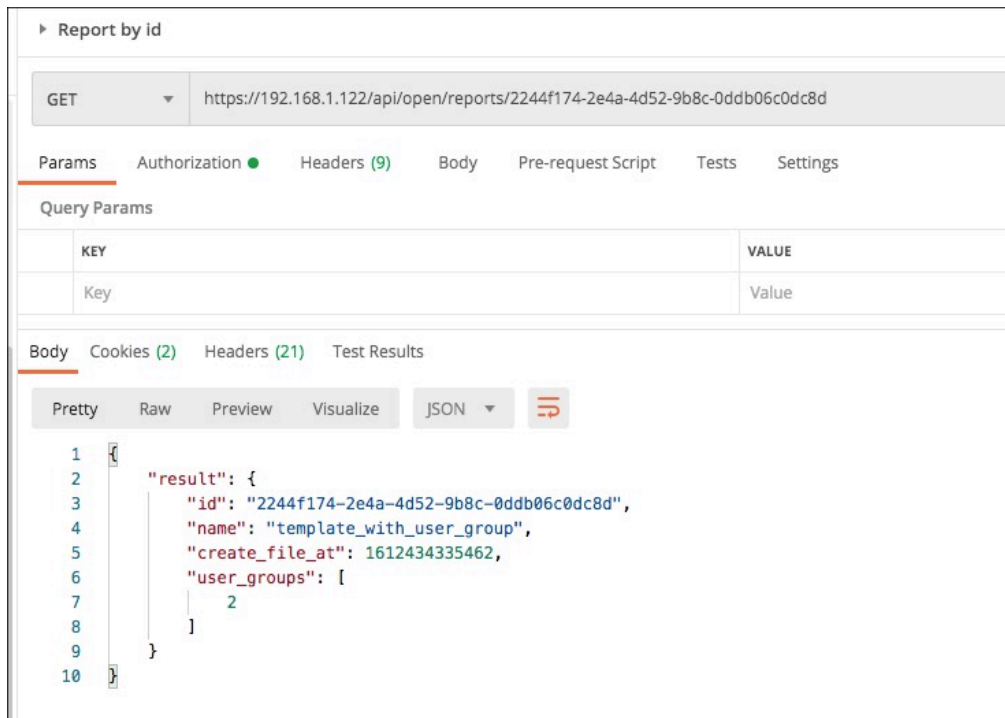


Figure 38. Example of reports/:id request

A GET to `/api/open/reports/:id/files` allows you to download the report having the id passed as path parameter.

Requirements and Restrictions

1. A user having the permission to execute api.
2. As middle parameter of the path you need to specify the id of the report.
3. The report download will be triggered.
4. In case the report with that id is not found you'll get a 404.

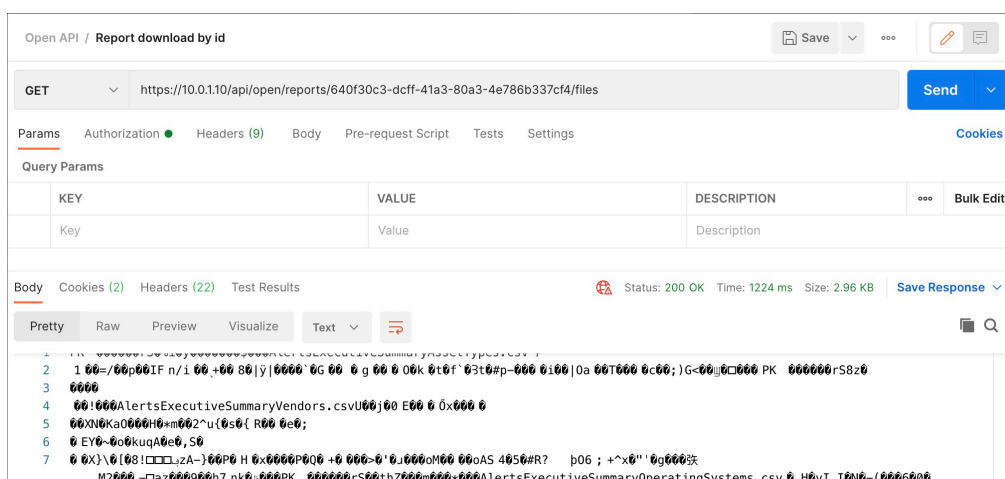


Figure 39. Example of reports/:id request

A POST to `/api/open/reports` allows you to create a new report.

Requirements and Restrictions

1. A user having the permission to execute api.
2. You need to pass as query param the `report_template_id` you want to create.
3. In case the request is well formed return a 202 response with the id of the job is taking care of the request.

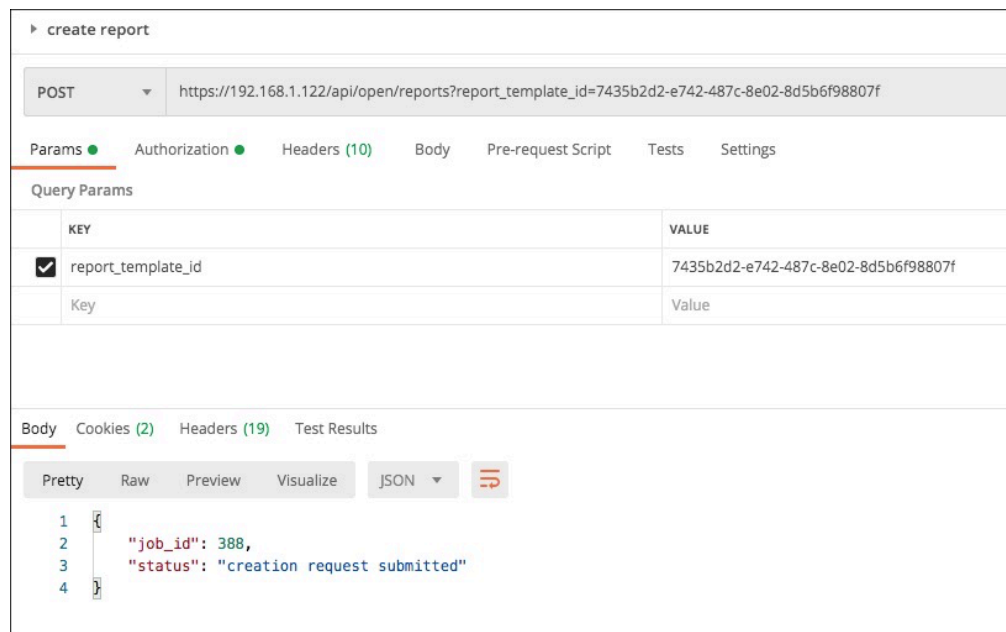


Figure 40. Example of create report request

A GET to `api/open/reports/jobs/1/status` allows you to get the create report job result.

Requirements and Restrictions

1. A user having the permission to execute api.
2. As parameter of the path you need to specify the id of the job.
3. The result will contain the job status

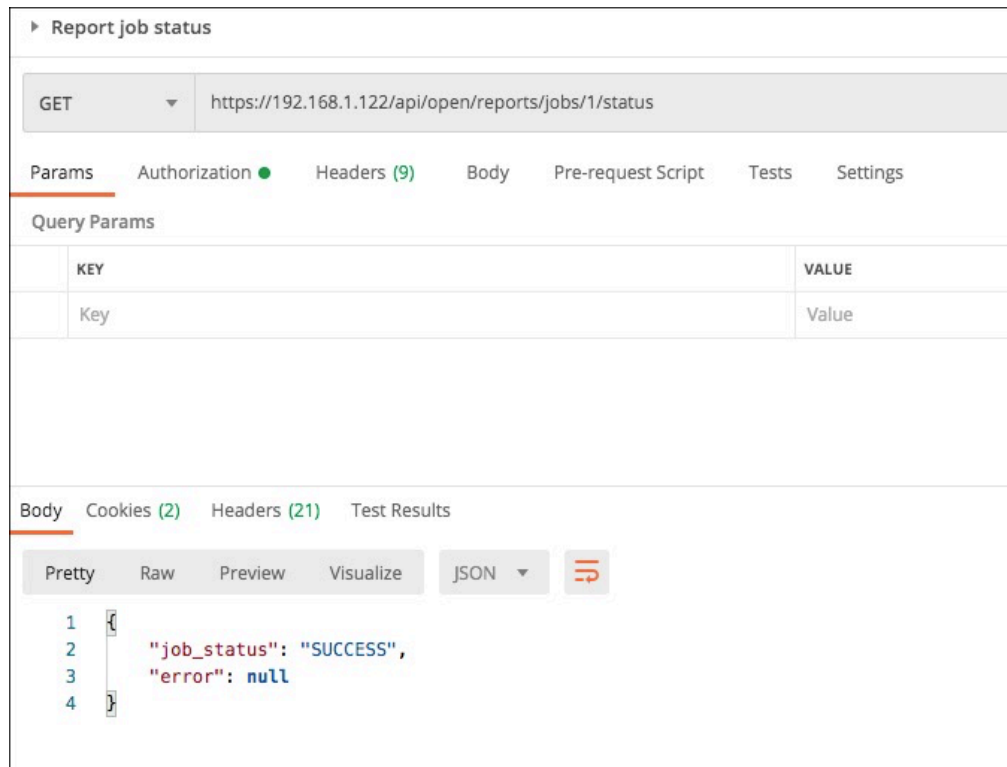


Figure 41. Example of reports/:id request

Report templates endpoint

A GET to `/api/open/report_templates` allows you to get a list of all the report templates.

Requirements and Restrictions

1. A user having the permission to execute api.
2. The result contains the list of all the report templates.
3. It's possible to use pagination adding `page` and `count` params
4. The `page` param is the number of the page to return, the `count` is the dimension of the page.
5. If `count` is nil or 0 the default value will be 100, if `page` is nil or 0 the request will not be paginated.

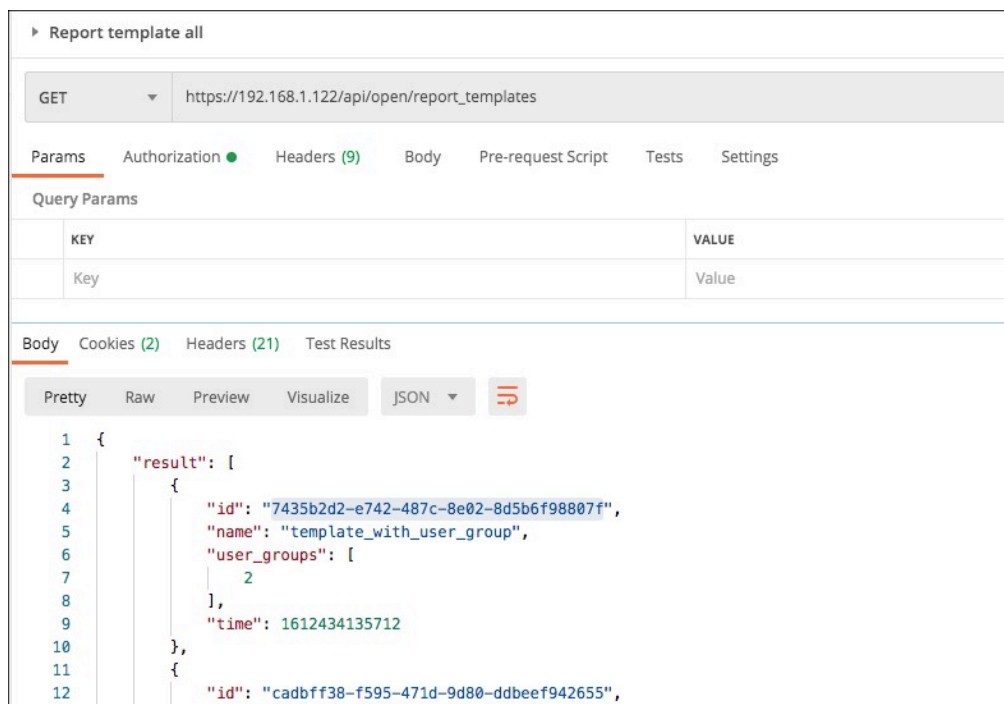


Figure 42. Example of reports all request

A GET to `/api/open/report_templates/:id` allows you to get the report template having the `id` passed as path parameter.

Requirements and Restrictions

1. A user having the permission to execute api.
2. As last parameter of the path you need to specify the `id` of the report template.
3. The result will contain the report template.
4. In case the report with that `id` is not found you'll get a 404.

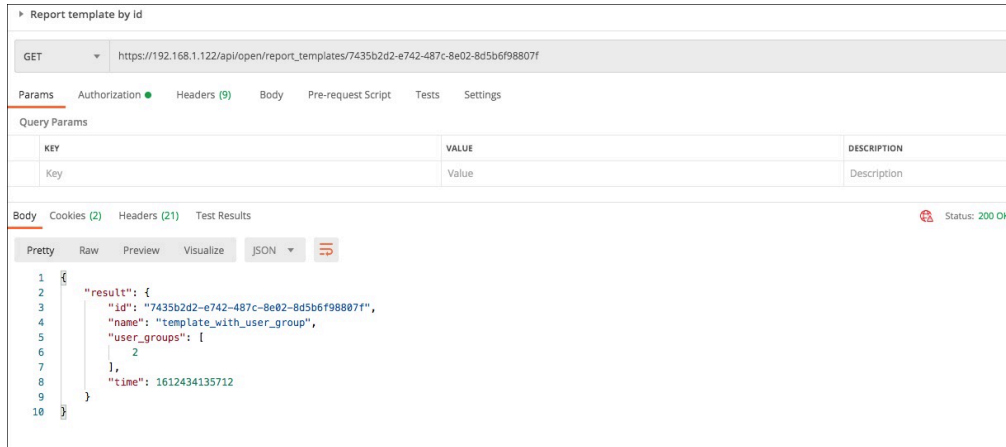


Figure 43. Example of reports/:id request

Quarantine endpoint

A GET request to `/api/open/quarantine` allows you to get a file from the quarantine directory.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.
2. The full path of the file must be specified in the `file` parameter and the format should be `/data/quarantine/<NAME>`.
3. If you specify a path that does not exist, the call returns a 404 error.
4. If the request is accepted, the result will contain the actual file that Guardian extracted from traffic and that the Sandbox classified as malicious.

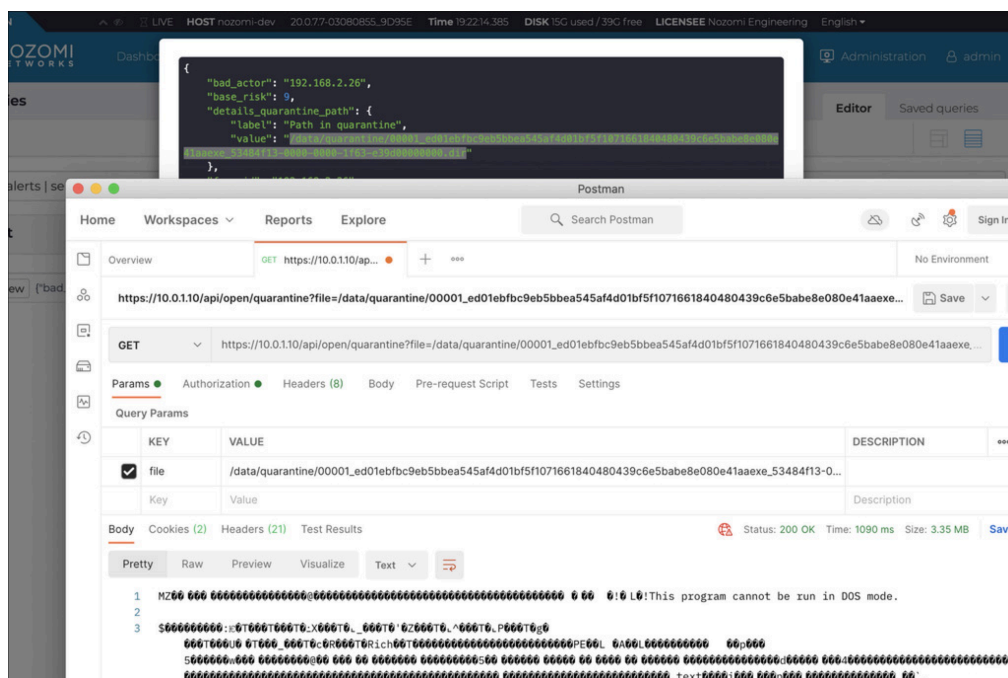


Figure 44. Example of request

Hint: as shown in the top part of the previous screenshot, the `file` parameter to be used with the request can be found in the `properties` field of `SIGN:MALWARE-DETECTED` alerts.

Threat intelligence

A POST request to `/api/open/threat_intelligence` allows you to create indicators.

Requirements and restrictions

1. An authenticated user must be in an **admin role** group or belong to a **Threat Intelligence** group with the **Allow configuration** option switched to ON in the group settings.
2. **JSON** content is represented as a array of contents that allows you to insert more than one (1) indicator at a time.
3. Type of content must be specified in the `type` parameter and the value must be: `packet_rules`, `yara_rules` or `stix_indicators`.
4. Content name must be specified in the `name` parameter.
5. The content must be specified in the `content` parameter.
6. If the request is accepted, the result contains the `result` with an **ID** as value.
7. The request is rejected if the sensor is connected to a [Central Management Console \(CMC\)](#).

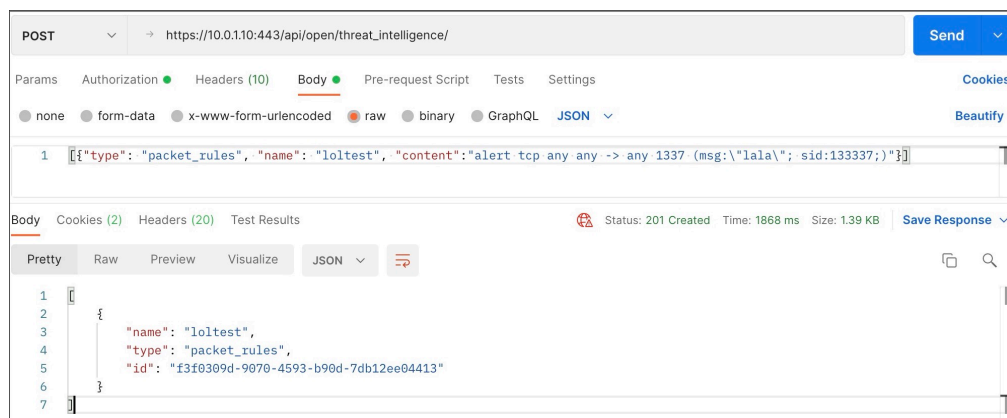


Figure 45. Example of request

A GET request to `/api/open/threat_intelligence` allows you to list indicators created by the user.

Requirements and Restrictions

1. An authenticated user must be in an **admin role** group or belong to a **Threat Intelligence** group.
2. If the request is accepted, the result contains a Json array of contents with `id`, `name` and `type`.

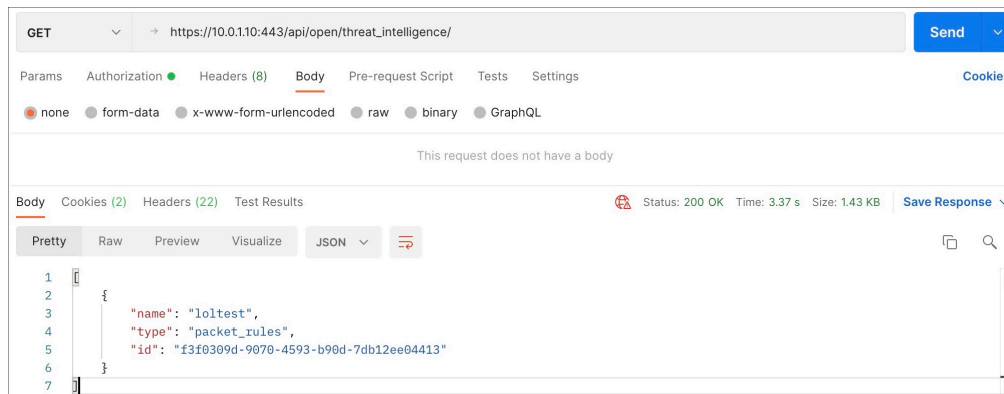


Figure 46. Example of request

A DELETE request to `/api/open/threat_intelligence` allows you to remove indicators.

Requirements and restrictions

1. An authenticated user must be in an **admin role** group or belong to a **Threat Intelligence** group with the **Allow configuration** option switched to ON in the group settings.
2. The Json content is represented as an array of contents that allows you to remove more than one (1) indicator at a time
3. The type of content must be specified in the `type` parameter and the value must be: `packet_rules`, `yara_rules` or `stix_indicators`.
4. The content id must be specified in the `id` parameter.
5. If the request is accepted, the result contains contents with `id` and `type`.
6. The request is rejected if the sensor is connected to a [CMC](#).

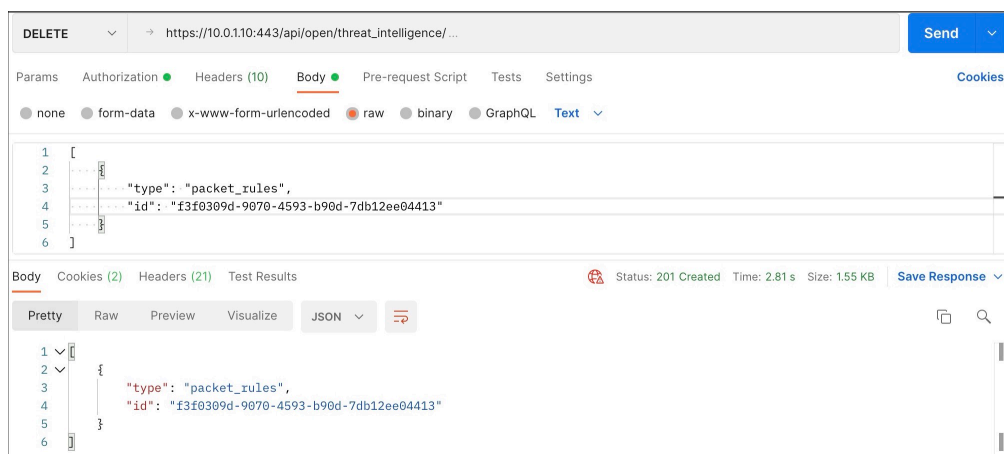


Figure 47. Example of request

Sensors endpoint

A GET to `/api/open/sensors/resources` allows you to get `cpu_perc`, `mem_used_perc` and `disk_usage_perc` resources.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role** or be part of a group with the **health** permission.

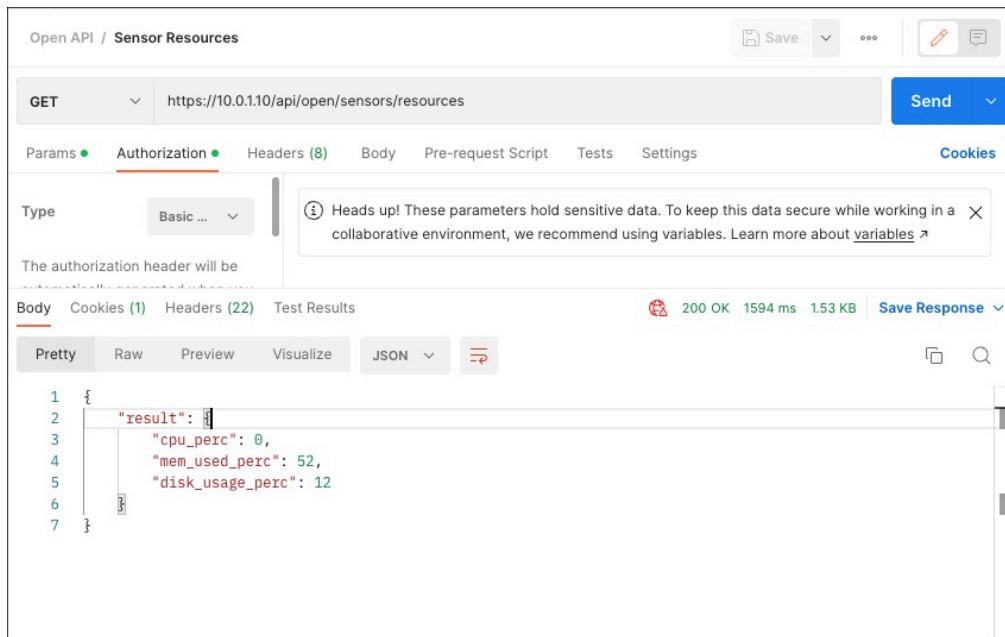


Figure 48. Example request

A GET to `/api/open/sensors/license` allows you to get the license information in the sensor.

Requirements and Restrictions

1. The authenticated user must be in a group having **admin role**.

Open API / Sensor License

GET https://10.0.1.10/api/open/sensors/license

Params Authorization Headers (8) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
file		

Body Cookies (1) Headers (22) Test Results

Status: 200 OK Time: 660 ms Size: 2.03 KB

Pretty Raw Preview Visualize JSON

```
1  {
2    "result": {
3      "base": {
4        "licensee": "Nozomi Engineering",
5        "bundle_name": "full",
6        "status": "ok",
7        "expire_date": "6403422937000",
8        "supported_nodes": "9999"
9      },
10     "threat_intelligence": {
11       "licensee": "company",
12       "status": "ok",
13       "expire_date": "6402994375000"
14     },
15     "asset_intelligence": {
16       "licensee": "company",
17       "status": "ok",
18       "expire_date": "6402994375000"
19     },
20     "arc": {
21       "licensee": "Unregistered",
22       "status": "UNLICENSED",
23       "expire_date": "0",
24       "supported_sensors": "-1"
25     },
26     "smart_polling": {
27       "licensee": "Unregistered",
28       "status": "UNLICENSED",
29       "expire_date": "0"
30     },
31     "fips": {
32       "licensee": "Unregistered",
33       "status": "UNLICENSED",
34       "expire_date": "0"
35     }
36   }
}
```

Figure 49. Example of user groups all request

Throttling policy

Overview

Throttling has been implemented in the **HTTP OpenAPI** to ensure fair usage and to maintain system stability. When the rate limit is exceeded, the [API](#) responds with a `429 Too Many Requests` status code along with the `retry-after` header indicating the number of seconds the client should wait before retrying the request.

Rate limit

The rate limit for [API](#) requests is set to 60 requests per minute. If this limit is exceeded, further requests within the same minute will be rejected with a `429 Too Many Requests` response.



Note:

The computation of the number of requests is not precise as the requests are counted based on a 1 minute time window starting from the next minute. That means that more than 60 requests might be needed to get the `429 - Too Many Requests` error.

Retry-after header

After receiving a `429 Too Many Requests` response, clients should parse the `retry-after` header to determine the waiting period before making another request. This header indicates the number of seconds the client must wait before retrying the request. Clients should respect this waiting period to avoid further throttling.

Impact

Throttling affects all endpoints exposed under the `/api/open` path. Clients making requests to any of these endpoints should be aware of the throttling policy and handle a `429 Too Many Requests` response appropriately.

Example

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
retry-after: 30
```

Chapter 4. Data model reference



alerts

A list of alerts that Guardian raises.

id	Primary key of this query source
type_id	The Type <i>ID</i> represents a unique "class" of the Alert, that characterizes what the Alert is about in a unique way
name	Name of the type <i>ID</i> . It can be updated dynamically by the correlation engine.
description	More details about the alert
severity	Syslog-like severity
mac_src	Source <i>MAC</i> address
mac_dst	Destination <i>MAC</i> address
ip_src	Source <i>IP</i> address
ip_dst	Destination <i>IP</i> address
risk	Risk, between 0 and 10
protocol	The protocol in which this entity has been observed
src_roles	Roles of the source node
dst_roles	Roles of the target node
time	Time when the first packet triggers the alert; for incidents, it is the time of the last correlated alert, which updates over time
ack	True if the Alert has been acknowledged
id_src	<i>ID</i> of the source node
id_dst	<i>ID</i> of the destination node
synchronized	True if this entity has been synchronized with the upper <i>CMC</i> or Vantage
zone_src	Source zone
zone_dst	Destination zone
appliance_id	The id of the sensor where this entity has been observed
port_src	Source port

port_dst	Destination port
label_src	Label of the source node
label_dst	Label of the destination node
trigger_id	<i>ID</i> of the triggering engine entity
trigger_type	Name of the trigger/engine
appliance_host	The hostname of the sensor where this entity has been observed
appliance_ip	The <i>IP</i> address of the sensor where this entity has been observed
transport_protocol	Name of the transport protocol (e.g. tcp/udp/icmp...)
is_security	True if the alert is a Cybersecurity alert. False otherwise (e.g. a network monitoring one)
note	User-defined note about the Alert
appliance_site	Site name of the sensor where this alert has been generated
parents	<i>ID</i> of parent incidents.
is_incident	True if this Alert is an incident grouping more alerts
properties	<i>JSON</i> with additional information for this alert
created_time	Time when the alert record was created
incident_keys	(Internal use)
bpf_filter	<i>BPF</i> filter for the entity, used when performing traces for this entity
closed_time	Time in epoch milliseconds when the alert has been closed. 0 if still open.
status	Status of the alert
session_id	<i>ID</i> of the Session during which this alert was raised
replicated	This is true if the record has been replicated on the replica machine
capture_device	Name of the interface from which this entity has been detected
threat_name	In case of known threat, this holds the threat name
type_name	Name of the type <i>ID</i> . It is immutable.

sec_profile_visible	True if the alert is visible according to the Security Profile. For alerts that are part of incidents, the field value is set to True when at least one child alert has the field value equal to True.
---------------------	--

appliances

This query source contains information about the sensors connected to the current CMC or Guardian.

ip	Last <i>IP</i> address of the sensor
last_sync	Timestamp in epoch milliseconds when the last full sync occurred
id	Primary key of this query source
info	<i>JSON</i> with miscellaneous information about the sensor
allowed	True if the sensor is in allowed state, meaning that all its data will be pushed its upstream sensor
sync_throughput	Amount of throughput used for synchronization purposes
is Updating	True if the sensor is currently applying a software update
map_position	(Internal use)
previous_alerts_count_last_5m	(Internal use)
version_locked	True if the sensor has been version locked
site	Site name this sensor belongs to
host	Host name of the sensor
time	Timestamp in epoch milliseconds when this entity was created or updated
synchronized	True if this entity has been synchronized with the upper <i>CMC</i> or Vantage
replicated	This is true if the record has been replicated on the replica machine
deleted_at	Time the entity was cancelled
health	(Internal use)
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The <i>IP</i> address of the sensor where this entity has been observed

appliance_host	The hostname of the sensor where this entity has been observed
force_update	True if a force update has been issued to this sensor
model	Model of the sensor
last_seen_packet	Point in time in epoch milliseconds when a packet has been captured by the sensor
has_same_version_of_cmc	(Internal use)
is_cmc	True if the sensor is a CMC
is_guardian	True if the sensor is a Guardian
is_remote_collector	True if the sensor is a Remote Collector
has_smart_polling	True if the Smart Polling is available

assertions

An assertion represents an automatic check against other query sources.

query	The query that is run as basis of the assertion
result	True if the assertion is satisfied, false if it is failing
name	Name of the assertion
failed_since	Time of since failure, in epoch milliseconds
id	Primary key of this query source
can_send_alert	True if the assertion will raise alerts
has_sent_alert	True if the assertion has sent alerts in the past
bpf_filter	BPF filter used to capture traffic on failure
failures_count	Number of failures
time	Timestamp in epoch milliseconds when this entity was created or updated
alert_delay	Delay in seconds before an alert is raised. Can be used as soft limit to handle flipping-states situations.
can_request_trace	True if a trace will be requested on failure
alert_risk	Risk of raised alerts
is_security	True if the assertion is a Cybersecurity assertion. False otherwise (e.g. a network monitoring one)
group_id	(Internal use)
note	Note about the assertion
deleted_at	Time the entity was cancelled
replicated	This is true if the record has been replicated on the replica machine
synchronized	True if this entity has been synchronized with the upper CMC or Vantage or Vantage
propagate_to_appliances	(Internal use)
propagated	(Internal use)

assets

Assets represent a local, physical system, and can be composed of one or more nodes.

name	Name of the node (Note: This field is automatically assigned by Guardian based on the most reliable available information, such as: address, network qualified names, nodes' assigned labels, etc.)
level	The purdue-model level of the asset
appliance_hosts	The hostname(s) of the sensor(s) where this entity has been observed
capture_device	Name of the interface from which this entity has been detected
ip	<i>IP</i> address(es) of the asset. It can be either IPv4, IPv6 or empty (in case of L2 node)
mac_address	<i>MAC</i> address(es) of the asset. It can be missing in some situations (serial nodes)
mac_address_level	(for internal use)
vlan_id	The <i>virtual local area network (VLAN) ID</i> (s) of the asset. It can be absent if the traffic to/from the node is not <i>VLAN</i> -tagged
mac_vendor	<i>MAC</i> address vendor(s). Is not empty when the <i>MAC</i> address is present and the corresponding Vendor name is known
os	Operating System of the asset, if available. This field is not present when the firmware_version is available
roles	The set of application-level roles of the asset. Differently from the type, these are behaviors
vendor	Vendor of the asset
vendor:info	This is a metadata field about the vendor field
firmware_version	The firmware version of the asset. The field is not present when the os field is available
firmware_version:info	This is a metadata field about the firmware_version field
os_or_firmware	Since os and firmware cannot be present at the same time, this field allow to get either of the two in a coalesce-like manner
serial_number	The serial number of the asset
serial_number:info	This is a metadata field about the serial_number field

product_name	The product name of the asset
product_name:info	This is a metadata field about the product_name field
type	The type of the asset
type:info	This is a metadata field about the type field
protocols	The unique protocols used from and to this asset
nodes	The set of node id(s) that compose this asset
custom_fields	Any additional custom field defined in the Custom fields
device_id	(Internal use)
is_ai_enriched	This field is true if this asset has been enriched by Asset Intelligence

asset_cves

View of Node Common Vulnerability and Exposures (CVE) grouped by CVE and asset.

asset_id	The ID of the vulnerable asset
cve	Common Vulnerabilities and Exposures (CVE) ID
cwe_id	Vulnerability category ID
cwe_name	Vulnerability category name
creation_time	Timestamp for creation of the vulnerability
epss_score	The EPSS score assigned to the CVE
id	Primary key for this query source
is_kev	Status of Known Exploited Vulnerability
latest_hotfix	Latest and most complete hotfix to install to solve the related CVE (only relevant for Microsoft Windows assets)
likelihood	Value between 0.1 and 1.0, where 1.0 represents the maximum likelihood that the CVE is present
matching_cpes	List of CPEs that lead to assigning the vulnerability to this node
minimum_hotfix	Minimum hotfix to install to solve the related CVE (only relevant for Microsoft Windows assets)
name	Labels of the vulnerable nodes
nodes	List of vulnerable nodes belonging to the same asset
references	List of references to external websites providing extra information about the vulnerability
resolved	Whether or not the vulnerability has been resolved by an installed patch (only relevant for Microsoft Windows assets)
score	CVSS (Common Vulnerability Scoring System) score assigned to this CVE
source	Entity that provided the original information about the vulnerability
summary	Description of the vulnerability
time	Timestamp (in epoch milliseconds) at which the vulnerability has been found on the network node in the user's environment

update_time	Timestamp for when this vulnerability was last updated
-------------	--

captured_logs

Logs captured passively over the network.

id	Primary key of this query source
time	Timestamp in epoch milliseconds when this entity was created or updated
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The <i>IP</i> address of the sensor where this entity has been observed
appliance_host	The hostname of the sensor where this entity has been observed
synchronized	True if this entity has been synchronized with the upper <i>CMC</i> or Vantage
id_src	Source id of the packet where the log was captured
id_dst	Destination id of the packet where the log was captured
protocol	The protocol in which this entity has been observed
log	Log contents
replicated	This is true if the record has been replicated on the replica machine
sync_time	Timestamp in epoch milliseconds when the event was synchronized

captured_urls

URLs and other protocol calls found in the network. Access to files, requests to DNS, requested URLs and others are available in this query source.

id	Primary key of this query source
id_src	Source id of the packet where the URL was captured
id_dst	Destination id of the packet where the URL was captured
protocol	The protocol in which this entity has been observed
time	Timestamp in epoch milliseconds when this entity was created or updated
url	Captured URL
operation	Operation performed to access the URL
username	Username that performed the activity
size_bytes	Size in bytes transferred when accessing the URL
session_id	ID of the Session during which this URL was captured
properties	JSON with additional information captured with this event

function_codes

Function codes used in the environment.

id	Primary key of this query source
protocol	The protocol in which this entity has been observed
fc	The symbolic function code
count	How many times this function code has been used since restart of the system
description	The description of the function code

health_log

Health-related events about the system - like high resource utilization or hardware-related issues or events.

id	Primary key of this query source
time	Timestamp in epoch milliseconds when this entity was created or updated
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The <i>IP</i> address of the sensor where this entity has been observed
appliance_host	The hostname of the sensor where this entity has been observed
synchronized	True if this entity has been synchronized with the upper <i>CMC</i> or Vantage
info	<i>JSON</i> with the information captured with about the event
replicated	This is true if the record has been replicated on the replica machine

link_events

Events that can occur on a link, including being available or not.

id_src	Source node id
id_dst	Destination node id
protocol	The protocol in which this entity has been observed
event	Payload of the event
id	Primary key of this query source
port_src	Source port
port_dst	Destination port
time	Timestamp in epoch milliseconds when the event was created
session_id	<i>ID</i> of the Session during which this <i>URL</i> was captured
transport_protocol	Transport protocol used by the traffic generating this event
params	<i>JSON</i> with additional information captured with this event

links

Links are protocol relations between two nodes with a specific protocol. They model the interaction between nodes.

from	Client node of the link
to	Server node of the link
is_from_public	True if client node is not a local node but an outside, public <i>IP</i> .
is_to_public	True if server node is not a local but an outside, public <i>IP</i> .
from_zone	Zone of the client node of the link
to_zone	Zone of the server node of the link
protocol	The protocol in which this entity has been observed
first_activity_time	Timestamp in epoch milliseconds when this a packet was sent on this link for the first time
last_activity_time	Timestamp in epoch milliseconds when this a packet was sent on this link for the last time
last_handshake_time	Timestamp in epoch milliseconds when the last <i>TCP</i> handshake has occurred on this link
transport_protocols	Set of transport protocols observed for this link
tcp_handshaked_connections.total	Total amount of <i>TCP</i> handshaked connections
tcp_handshaked_connections.last_5m	Amount of <i>TCP</i> handshaked connections in the last 5 minutes
tcp_handshaked_connections.last_15m	Amount of <i>TCP</i> handshaked connections in the last 15 minutes
tcp_handshaked_connections.last_30m	Amount of <i>TCP</i> handshaked connections in the last 30 minutes
tcp_connection_attempts.total	Total amount of bytes for <i>TCP</i> SYN packets
tcp_connection_attempts.last_5m	Amount of <i>TCP</i> SYN packets in the last 5 minutes
tcp_connection_attempts.last_15m	Amount of <i>TCP</i> SYN packets in the last 15 minutes

tcp_connection_attempts.last_30m	Amount of <i>TCP</i> SYN packets in the last 30 minutes
transferred.packets	Total number of packets transmitted
transferred.bytes	Total number of bytes transmitted
transferred.last_5m_bytes	Number of bytes transmitted in the last 5 minutes
transferred.last_15m_bytes	Number of bytes transmitted in the last 15 minutes
transferred.last_30m_bytes	Number of bytes transmitted in the last 30 minutes
transferred.smallest_packet_bytes	Smallest packet size in bytes observed
transferred.biggest_packet_bytes	Biggest packet size in bytes observed
transferred.avg_packet_bytes	Average packet size in bytes observed
tcp_retransmission.percent	Percentage of <i>TCP</i> packets that have been retransmitted
tcp_retransmission.packets	Total number of <i>TCP</i> packets that have been retransmitted
tcp_retransmission.bytes	Total amount of bytes for <i>TCP</i> packets that have been retransmitted
tcp_retransmission.last_5m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 5 minutes
tcp_retransmission.last_15m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 15 minutes
tcp_retransmission.last_30m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 30 minutes
throughput_speed	Live throughput for the entity
is_learned	This is true for links that were observed during the learning phase
is_fully_learned	This is true for links that were observed also during the learning phase and which properties are not changed since then
is_broadcast	True if this is not a real node but a broadcast or multicast entry

has_confirmed_data	True if data has been exchanged in both directions, or more genererically if the data is really flowing and is not a likely scan or alike
alerts	The number of alerts being created around this link
last_trace_request_time	Last time in epoch milliseconds that a trace has been asked on the link
active_checks	List of active real-time checks on the entity
function_codes	Set of function codes seen on this link
bpf_filter	BPF filter for the entity, used when performing traces for this entity

node_cpe_changes

When a Common Platform Enumeration (CPE) updates, it creates an entry in this query source to track software updates or to detect software.

id	Primary key of this query source
node_id	The id of the node this CPE refers to
cpe	The old full CPE
cpe_part	The old part piece of the CPE
cpe_vendor	The old vendor piece of the CPE
cpe_product	The old product piece of the CPE
cpe_version	The old version piece of the CPE
cpe_update	The old update piece of the CPE
new_cpe	The CPE that has replaced the old one
new_cpe_vendor	The CPE vendor that has replaced the old one
new_cpe_product	The CPE product that has replaced the old one
new_cpe_version	The CPE version that has replaced the old one
new_cpe_update	The CPE update that has replaced the old one
node_cpe_id	The ID of the Node CPE id (node_cpe query source) entity to which this change event relates to
time	Timestamp in epoch milliseconds when this entity was created or updated
synchronized	True if this entity has been synchronized with the upper CMC or Vantage
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The IP address of the sensor where this entity has been observed
appliance_host	The hostname of the sensor where this entity has been observed
human_cpe_vendor	The old human-readable version of the CPE vendor
human_cpe_product	The old human-readable version of the CPE product

new_human_cpe_vendor	The new human-readable version of the <i>CPE</i> vendor
new_human_cpe_product	The new human-readable version of the <i>CPE</i> product
human_cpe_version	The old human-readable version of the <i>CPE</i> version
human_cpe_update	The old human-readable version of the <i>CPE</i> update
new_human_cpe_version	The new human-readable version of the <i>CPE</i> version
new_human_cpe_update	The new human-readable version of the <i>CPE</i> update
likelihood	A value between 0.1 and 1.0 where 1.0 represents the maximum likelihood of the <i>CPE</i> to be real. This is the old value.
new_likelihood	A value between 0.1 and 1.0 where 1.0 represents the maximum likelihood of the <i>CPE</i> to be real. This is the new value.
replicated	This is true if the record has been replicated on the replica machine
cpe_edition	The old edition piece of the <i>CPE</i>
new_cpe_edition	The new edition piece of the <i>CPE</i>
human_cpe_edition	The old human-readable version of the <i>CPE</i> edition
new_human_cpe_edition	The new human-readable version of the <i>CPE</i> edition

node_cpes

Lists Common Platform Enumerations (CPEs), that is software or components connected to a specific node in the system.

id	Primary key of this query source
node_id	The id of the node this CPE refers to
cpe	The full CPE
cpe_part	The part piece of the CPE
cpe_vendor	The vendor piece of the CPE
cpe_product	The product piece of the CPE
cpe_version	The version piece of the CPE
cpe_update	The update piece of the CPE
time	Timestamp in epoch milliseconds when this entity was created or updated
synchronized	True if this entity has been synchronized with the upper CMC or Vantage
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The IP address of the sensor where this entity has been observed
appliance_host	The hostname of the sensor where this entity has been observed
updated	This is true if the record has been processed. When false, the value of the record must not be used.
cpe_translator	Name of the CPE translator that produced this CPE . For diagnostic purposes only.
human_cpe_vendor	The human-readable version of the CPE vendor
human_cpe_product	The human-readable version of the CPE product
human_cpe_version	The human-readable version of the CPE version
human_cpe_update	The human-readable version of the CPE update
likelihood	A value between 0.1 and 1.0 where 1.0 represents the maximum likelihood of the CPE to be real

replicated	This is true if the record has been replicated on the replica machine
cpe_edition	The edition piece of the <i>CPE</i>
human_cpe_edition	The human-readable version of the <i>CPE</i> edition

node_cves

Vulnerabilities are matched against current CVEs.

id	Primary key for this query source
time	Timestamp (in epoch milliseconds) at which the vulnerability has been found on the network node in the user's environment
appliance_host	Host name of the Nozomi Networks sensor where the CVE entry is hosted
appliance_id	ID of the Nozomi Networks sensor where the CVE entry is hosted
appliance_ip	IP address of the Nozomi Networks sensor where the CVE entry is hosted
cve	CVE ID
cve_creation_time	Timestamp for creation of the vulnerability
cve_references	List of references to external websites providing extra information about the vulnerability
cve_score	CVSS (Common Vulnerability Scoring System) score assigned to this CVE
cve_source	Entity that provided the original information about the vulnerability
cve_summary	Description of the vulnerability
cve_update_time	Timestamp for when this vulnerability was last updated
cwe_id	Vulnerability category ID
cwe_name	Vulnerability category name
installed_on	(For internal use)
node_label	Label of the vulnerable node
node_type	Type of the vulnerable node
node_vendor	Vendor of the vulnerable node
node_product_name	Product name of the vulnerable node
node_os	Operating system of the vulnerable node

node_firmware_version	Firmware version of the vulnerable node
node_id	Node <i>ID</i> for the referenced <i>CVE</i>
asset_id	<i>ID</i> of the vulnerable asset
zone	Network zone to which the vulnerable node belongs
likelihood	Value between 0.1 and 1.0, where 1.0 represents the maximum likelihood that the <i>CVE</i> is present
matching_cpes	List of CPEs that lead to assigning the vulnerability to this node
resolved	Whether or not the vulnerability has been resolved by an installed patch (only relevant for Microsoft Windows assets)
resolution_reason	Specifies the possible resolution reason for a vulnerability
resolved_source	Specifies the data source from which the resolution status' related information could be retrieved (only relevant for Microsoft Windows assets)
latest_hotfix	Latest and most complete hotfix to install to solve the related <i>CVE</i> (only relevant for Microsoft Windows assets)
minimum_hotfix	Minimum hotfix to install to solve the related <i>CVE</i> (only relevant for Microsoft Windows assets)

node_points

Data points are polled via Smart Polling from monitored nodes.

id	Primary key of this query source
node_id	The id of the node this point refers to
strategy	The strategy used to retrieve this point
time	Timestamp in epoch milliseconds when this entity was created or updated
name	The name of the point
value	(Deprecated) See content below
value_type	The type of the point
human_name	The human name of the point
content	The actual content of the polled information

nodes

A list of nodes, where a node is an L2 or L3 or other entity able to speak some protocol.

appliance_host	The hostname of the sensor where this entity has been observed
label	Name of the node
id	Primary key of this query source
ip	<i>IP</i> address of the node. It can be either IPv4, IPv6 or empty (in case of L2 node)
mac_address	<i>MAC</i> address of the node. It can be missing in some situations (serial nodes)

mac_address:info	<p>This is a metadata field about the mac_address field.</p> <ul style="list-style-type: none"> • protocol_source - is the cause of the latest mac_address:info change • likelihood - a value between 0.1 and 1.0 where 1.0 represents the maximum likelihood that the MAC address is the native one from the node. • likelihood_level - level of confidence regarding whether the MAC address is the native one from the node, or it is one routed/substituted by the network. Values: <ul style="list-style-type: none"> ◦ unconfirmed (no information is available) ◦ likely (some information indicates it can be native) ◦ confirmed (it is definitely native) • source - indicates where the information comes from: <ul style="list-style-type: none"> ◦ manual: information that is manually added from the configuration ◦ import: imported information ◦ passive: information from Deep Packet Inspection ◦ asset-kb: information from Asset Intelligence ◦ smart-polling: information from Smart Polling • granularity - is the level of detail of the information. Values: <ul style="list-style-type: none"> ◦ manual-or-import: information manually added or imported ◦ complete: detailed information has been extracted. ◦ partial: detailed, but still not complete. ◦ generic: a family/generic value has been found, but is not detailed. ◦ unknown • confidence - measures the confidence that the information is the one published. Values: <ul style="list-style-type: none"> ◦ manual-or-import: information manually added or imported, therefore the highest confidence ◦ high ◦ good ◦ low ◦ unknown
mac_vendor	<p>MAC address vendor. Is not empty when the MAC address is present and the corresponding Vendor name is known.</p>

subnet	The subnet to which this node belongs, if any.
vlan_id	The VLAN ID of the node. It can be absent if the traffic to/from the node is not VLAN -tagged.
vlan_id:info	This is a metadata field about the vlan_id field.
zone	The zone name to which this node belongs to
level	The purdue-model level of the node
type	The type of the node
type:info	This is a metadata field about the type field.
os	Operating System of the node, if available. This field is not present when the firmware_version is available.
vendor	Vendor of the node
vendor:info	This is a metadata field about the vendor field.
product_name	The product name of the node
product_name:info	This is a metadata field about the product_name field.
firmware_version	The firmware version of the node. The field is not present when the os field is available.
firmware_version:info	This is a metadata field about the firmware_version field.
serial_number	The serial number of the node
serial_number:info	This is a metadata field about the serial_number field.
is_broadcast	True if this is not a real node but a broadcast or multicast entry
is_public	True if this not a local node but an outside, public IP address.
reputation	This can be good or bad depending on information coming from STIX indicators
is_confirmed	This is true for nodes that are confirmed to exist. Non-existing targets of port scans for instance are not confirmed

is_compromised	This is true for nodes that have been recognised as compromised according to threat indicators
is_learned	This is true for nodes that were observed during the learning phase
is_fully_learned	This is true for nodes that were observed also during the learning phase and which properties are not changed since then
is_disabled	This is true for nodes that are hidden from graphs because too noisy
roles	The set of application-level roles of the node. Differently from the type, these are behaviors.
links	The set of links to which this node is related
links_count	The total number of links from and to this node
protocols	The unique protocols used from and to this node
created_at	Timestamp in epoch milliseconds when this node was first observed
first_activity_time	Timestamp in epoch milliseconds when this node send a packet for the first time
last_activity_time	Timestamp in epoch milliseconds when this node send a packet for the last time
received.packets	Total number of packets received
received.bytes	Total number of bytes received
received.last_5m_bytes	Number of bytes received in the last 5 minutes
received.last_15m_bytes	Number of bytes received in the last 15 minutes
received.last_30m_bytes	Number of bytes received in the last 30 minutes
sent.packets	Total number of packets sent
sent.bytes	Total number of bytes sent
sent.last_5m_bytes	Number of bytes sent in the last 5 minutes
sent.last_15m_bytes	Number of bytes sent in the last 15 minutes

sent.last_30m_bytes	Number of bytes sent in the last 30 minutes
tcp_retransmission.percent	Percentage of <i>TCP</i> packets that have been retransmitted
tcp_retransmission.packets	Total number of <i>TCP</i> packets that have been retransmitted
tcp_retransmission.bytes	Total amount of bytes for <i>TCP</i> packets that have been retransmitted
tcp_retransmission.last_5m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 5 minutes
tcp_retransmission.last_15m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 15 minutes
tcp_retransmission.last_30m_bytes	Amount of bytes of <i>TCP</i> packets that have been retransmitted in the last 30 minutes
variables_count	Amount of variables attached to the node
device_id	(Internal use)
properties	Additional properties found by several protocols attached to the node
custom_fields	Any additional custom field defined in the Custom fields
bpf_filter	<i>BPF</i> filter for the node, used when performing traces for this node and as building block for link traces too
device_modules	Set of modules of this devices, if any
capture_device	Name of the interface from which this entity has been detected

report_files

Generated reports available for consultation.

id	Primary key of this query source
name	Name of the report file
create_file_at	Time the report was created
deleted_at	Time the entity was cancelled
time	Timestamp in epoch milliseconds when this entity was created or updated
appliance_id	The id of the sensor where this entity has been observed
appliance_ip	The <i>IP</i> address of the sensor where this entity has been observed
appliance_host	The hostname of the sensor where this entity has been observed
synchronized	True if this entity has been synchronized with the upper <i>CMC</i> or Vantage
replicated	This is true if the record has been replicated on the replica machine
created_by	User that generated the report
user_groups	User groups allowed to see the report
file_type	Type of file generated

sessions_history

Archived sessions.

For more details, see the sessions query source.

id	Primary key of this query source
status	Tells if the session is ACTIVE, CLOSED, SYN, SYN-ACK
direction_is_known	True if the session direction has been discovered. If false, from and to may be swapped.
from	Client node id
to	Server node id
from_zone	Client zone
to_zone	Server zone
transport_protocol	Transport protocol of the session
from_port	Port on the client side
to_port	Port on the server side
protocol	The protocol in which this entity has been observed
vlan_id	The VLAN ID of the session. It can be absent if the traffic of the session is not VLAN -tagged.
transferred.packets	Total number of packets transmitted
transferred.bytes	Total number of bytes transmitted
transferred.last_5m_bytes	Number of bytes transmitted in the last 5 minutes
transferred.last_15m_bytes	Number of bytes transmitted in the last 15 minutes
transferred.last_30m_bytes	Number of bytes transmitted in the last 30 minutes
transferred.smallest_packet_bytes	Smallest packet size in bytes observed
transferred.biggest_packet_bytes	Biggest packet size in bytes observed
transferred.avg_packet_bytes	Average packet size in bytes observed
throughput_speed	Live throughput for the session

first_activity_time	Timestamp in epoch milliseconds when this session was found for the first time
last_activity_time	Timestamp in epoch milliseconds when this session was detected for the last time
key	(Internal use)
bpf_filter	<i>BPF</i> filter for the entity, used when performing traces for this entity

sessions

Live, mostly open, sessions between nodes. A session is a specific application-level connection between nodes. A link can hold one or more sessions at a given time.

id	Primary key of this query source
status	Tells if the session is ACTIVE, CLOSED, SYN, SYN-ACK
direction_is_known	True if the session direction has been discovered. If false, from and to may be swapped.
from	Client node id
to	Server node id
from_zone	Client zone
to_zone	Server zone
transport_protocol	Transport protocol of the session
from_port	Port on the client side
to_port	Port on the server side
protocol	The protocol in which this entity has been observed
vlan_id	The VLAN ID of the session. It can be absent if the traffic of the session is not VLAN -tagged.
transferred.packets	Total number of packets transmitted
transferred.bytes	Total number of bytes transmitted
transferred.last_5m_bytes	Number of bytes transmitted in the last 5 minutes
transferred.last_15m_bytes	Number of bytes transmitted in the last 15 minutes
transferred.last_30m_bytes	Number of bytes transmitted in the last 30 minutes
transferred.smallest_packet_bytes	Smallest packet size in bytes observed
transferred.biggest_packet_bytes	Biggest packet size in bytes observed
transferred.avg_packet_bytes	Average packet size in bytes observed
throughput_speed	Live throughput for the entity

first_activity_time	Timestamp in epoch milliseconds when this session was found for the first time
last_activity_time	Timestamp in epoch milliseconds when this session was detected for the last time
key	(Internal use)
bpf_filter	<i>BPF</i> filter for the entity, used when performing traces for this entity

variable_history

History of values for variables, where history has been enabled.

id	Primary key of this query source
var_key	Variable identifier this historic value belongs to
value	The captured value of the variable
datatype	The type of the variable value
time	Timestamp in epoch milliseconds when this entity was created or updated
quality_enum	The quality values attached to the variable value
client_node	The client node involved in the communication when observing the variable
function_code	Function code used to access the variable

variables

Variables extracted via DPI from the monitored system.

var_key	The primary key of this data source
host	The node to which this variable belongs to
host_label	The label of the node to which this variable belongs to
namespace	It is the identifier of the subsystem in the producer to which the variable belongs. Also known as the RTU ID of the variable.
name	The name of the variable, likely an identifier of the memory area
label	The human-readable name of the variable
unit	The unit for the value of the variable
scale	The scale of the variable. By default it is 1.0, and can be configured/changed with external information.
offset	The offset of the variable. By default it is 0.0, and can be configured/changed with external information.
type	The type of the value of the variable.
is_numeric	True if it represents a number
min_value	The minimum observed value
max_value	The maximum observed value
value	The live, last observed value of the variable. Upon restart, this value is unknown because it needs to reflect the real time status.
bit_value	The live, last observed value of the variable, expressed in bits. Upon restart, this value is unknown because it needs to reflect the real time status.
last_value	This is the last observed value, and is persisted on reboots
last_value_is_valid	True if the last value is valid (has valid quality)
last_value_quality	The quality of the last value
last_cause	The cause of the last value
protocol	The protocol in which this entity has been observed

last_function_code_info	The last value function code information
last_function_code	The last value function code
first_activity_time	Timestamp in epoch milliseconds when this variable was found for the first time
last_range_change_time	Timestamp in epoch milliseconds when this variable's range changed
last_activity_time	Timestamp in epoch milliseconds when this variable was detected for the last time
last_update_time	Timestamp in epoch milliseconds of the last valid quality
last_valid_quality_time	Timestamp in epoch milliseconds of the last time quality was valid
request_count	The number of times this variable has been accessed
changes_count	The number of times this variable has changed
latest_bit_change	Indices of the flipped bits during the latest variable change
last_client	The last node that accessed this variable (in read or write mode)
history_status	Tells if the history is enabled or not on this variable
active_checks	List of active real-time checks on the entity
flow_status	Tells the status of the flow, that is if the variable has a cyclic behavior or not
flow_anomalies	Reports anomalies in the flow, if any
flow_anomaly_in_progress	Reports a flow anomaly is in progress or not
flow_hiccups_percent	Shows the amount of hiccups in the flow
flow_stats.avg	Shows the average access time
flow_stats.var	Shows the variance of the access time

Chapter 5. Data integration best practices



OpenAPI data

API users

Nozomi Networks recommends the practice of creating a user specifically for the purpose of OpenAPI access. This provides a straightforward demarcation of responsibilities for auditing and tracing.

Best Practice: Create a user specifically to access OpenAPI.

Authentication

Each call to an OpenAPI method requires authentication. OpenAPI currently supports basic authentication. For example, when using CURL, if you have a username and password for your OpenAPI user, you would use the following header along with your query:

```
-H "Authorization: Basic <AUTH_TOKEN>"
```

Where **<AUTH_TOKEN>** is the base64 encoding of Username:Password.

Note that the language and method of implementation (e.g. CURL vs. Java) dictates how basic authentication is performed.



Note:

When you query OpenAPI for data, the `-k -user Username:Password` option may be used for basic authentication.

Querying Nozomi Networks sensors

Data retrieved from the OpenAPI is done by calling the OpenAPI HTTP interface on either a Guardian or [CMC](#) sensor.

The query endpoint is powerful and allows the integrator to manipulate data through the use of queries. A full list of the available query data sources, commands, and functions is available in the N2OS User Manual.

Simple query example

This query retrieves the nodes in the Nozomi Networks sensor:

```
curl -k -H "Authorization: Basic <AUTH_TOKEN>"  
https://<YourHost>/api/open/query/do?query=nodes
```

If there are two nodes; the results will be similar to this:

```
{  
  "header": [  
    All of the headers...  
  ],  
  "result": [  
    { First Node data },  
    { Second Node data }  
  ]  
}
```

```
{ Second Node data }
],
"total": 2
}
```

Complex query example

Ensure that the complex query commands are properly [uniform resource identifier \(URI\)](#) encoded. The following query retrieves the node count in the Nozomi Networks sensor:

```
curl -k -H "Authorization: Basic <AUTH_TOKEN>"
https://<YourHost>/api/open/query/do?query=nodes%20%7C%20count
```

Note that the original query text “**nodes | count**” has been [URI](#) encoded to **nodes %20%7C%20count**.

Note that the language and method of implementation dictate how the [URI](#) encoding is accomplished.

Uploading asset information to the Nozomi Networks sensor

Data can also be uploaded into Guardian or [CMC](#) via an upload or enhanced asset information. This is referred to as **importing** in the OpenAPI.

The import endpoint is simple and allows the integrator to upload node data through the use of import statements. The command list is available in the N2OS User Manual.



Note:

The credentials of the user performing the OpenAPI call to import data must be in the admin group to upload information into a Nozomi Networks sensor.

Import example using CURL with a CSV file

Using a sample [CSV](#) file, **assets.csv** looks like this:

```
ip,label,firmware_version,vendor,product_name,serial_number,mac_address
192.168.1.60,CSV Uploaded Asset 1,1.2.2,ACME,ACME Product
1,abcdefge,00:01:02:03:04:06
192.168.1.61,CSV Uploaded Asset 2,1.2.2,ACME,ACME Product
2,abcdefge,00:11:12:13:14:16
```

The following command will upload these assets into the Guardian or [CMC](#):

```
curl -k -X POST https://<YourHost>/api/open/nodes/import -H
"Authorization: Basic <AUTH_TOKEN>" -F file=@<PathTo>/assets.csv
```

Import example using CURL with JSON file

Using a sample [JSON](#) file, assets.json looks like this:

```
{
  "nodes": [
    {
      "ip": "1.2.3.8",
      "label": "JSON_Uploaded_Asset_1",
      "mac_address": "00:00:00:11:11:11",
      "firmware_version": "1.2.3",
      "product_name": "ACME_PLC_2",
      "serial_number": "1-789A10-2",
      "vendor": "ACME"
    },
    {
      "ip": "1.2.3.3",
      "label": "JSON_Uploaded_Asset_2",
      "mac_address": "00:00:00:11:11:15",
      "firmware_version": "1.2.2",
      "product_name": "ACME_PLC_1",
      "serial_number": "1-789A10-6",
      "vendor": "ACME"
    }
  ]
}
```

Depending on your CURL implementation, the file may have to be submitted using `-d` as in the example below.

The following command uploads these assets into the Guardian or [CMC](#):

```
curl -k -X POST https://<YourHost>/api/open/nodes/import_from_json -H
"Authorization: Basic <AUTH_TOKEN>" -H "Content-Type: application/json" -d
{"nodes":[{"ip":"1.2.3.8","label":"JSON_Uploaded_Asset_1","mac_address":"00:00:00:11:11:11",
"firmware_version":"1.2.3","product_name":"ACME_PLC_2","serial_number":"1-789A10-2",
"vendor":"ACME"}, {"ip":"1.2.3.3","label":"JSON_Uploaded_Asset_2",
"mac_address":"00:00:00:11:11:15","firmware_version":"1.2.2",
```

```
"product_name": "ACME_PLC_1", "serial_number": "1-789A10-6", "vendor": "ACME" } ] }
```

Import commands

Command	HTTP Parameters	Description
import_from_csv_file	-F file=@</path/to/CSV_FILE>	This command allows the import of asset information from a CSV file. The CSV file must have the appropriate column headers present in the first line.
import_from_json	-H 'Content-Type: application/json' -d <JSON_DATA>	This command allows the import of asset information from JSON data. Note that the JSON data is specified in the HTTP headers directly.

Downloading traces

Traces associated with an alert can be downloaded via the [API](#) as well. You need the alert ID in order to accomplish this. The following command downloads a trace associated with an alert [ID](#) <YourAlertID> to the file specified by <YourTraceFile>:

```
curl -k -X GET https://<YourHost>/api/open/alerts/<YourAlertID>/trace -H "Authorization: Basic <AUTH_TOKEN>" -H "Content-Type: application/json" --output <YourTraceFile>
```

Nozomi Networks certification

Technical requirements

This reference outlines the technical requirements for integrating with Nozomi Networks, including API authentication, pagination rules, and logging format compliance.

API integrations

The integration must use [API](#)-key based authentication.

Pagination must be used as follows:

- For [API](#)-based integrations, pagination must be used
- For [API](#)-based integrations, no more than 10,000 records can be pulled in a single call, and no more than 1,000 records in a single page

Make sure that a maximum parallel requests limit is configured so requests sent to the [API](#) is throttled.

An `nn-app` value-key pair is included in the header:

```
{
  'nn-app': 'your-app-name',
  'nn-app-version': '1.0.0'
}
```

CEF Syslog integrations

Make sure that the custom fields are mapped as necessary for your integration. `flexString4` in the [Syslog](#) can contain multiple field-value pairs.

The [Syslog](#) connection can be received with encryption enabled.

For more details, see **Data Integrations > Common Event Format (CEF)** in the related **Administrator Guide**.

Custom JSON integrations

There are no technical requirements for certification.

Certify your integration

Follow this process to certify your integration with Nozomi Networks. Learn how to comply with technical requirements, submit necessary documentation, and complete a demonstration for final approval.

Before you begin

Make sure that your integration satisfies the [Technical requirements \(on page 145\)](#).

About this task

**Note:**

If you need help during the certification process, please send an email to:

partners@nozominetworks.com

Procedure

1. Make sure that the integration has been tested with all products it is intended to work with:
 - Guardian, and/or
 - [CMC](#) (All-in-One mode), and/or
 - [CMC](#) (Multicontext mode), and/or
 - Vantage
2. Go to the [Nozomi Networks Sandbox](#) to download the integration fact sheet.
3. Complete the integration fact sheet.
4. Submit all these items, as applicable to: sandbox@nozominetworks.com
 - Your integration fact sheet
 - Your end-user configuration guide
 - All sales and marketing materials
 - A video demonstration of the integration

The video should be 5 to 10 minutes long and include a complete walk-through of the integration, from initial configuration to events flowing from the Nozomi Networks platform into your environment.

5. Nozomi Networks will review and provide all the technical feedback necessary to certify your integration. If necessary, a telephone/video call will be scheduled to do a live review of the integration.

Results

Once this procedure has been successfully completed, Nozomi Networks will certify your integration.

Glossary



Application Programming Interface

An API is a software interface that lets two or more computer programs communicate with each other.

Berkeley Packet Filter

The BPF is a technology that is used in some computer operating systems for programs that need to analyze network traffic. A BPF provides a raw interface to data link layers, permitting raw link-layer packets to be sent and received.

Central Management Console

The Central Management Console (CMC) is a Nozomi Networks product that has been designed to support complex deployments that cannot be addressed with a single sensor. A central design principle behind the CMC is the unified experience, that lets you access information in a similar way as on the sensor.

Command-line interface

A command-line processor uses a command-line interface (CLI) as text input commands. It lets you invoke executables and provide information for the actions that you want them to do. It also lets you set parameters for the environment.

Comma-separated Value

A CSV file is a text file that uses a comma to separate values.

Common Platform Enumeration

CPE is a structured naming scheme for information technology (IT) systems, software, and packages. CPE is based on the generic syntax for Uniform Resource Identifiers (URI) and includes a formal name format, a method for checking names against a system, and a description format for binding text and tests to a name.

Common Vulnerabilities and Exposures

CVEs give a reference method information-security vulnerabilities and exposures that are known to the public. The United States' National Cybersecurity FFRDC maintains the system.

Hypertext Transfer Protocol

HTTP is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Identifier

A label that identifies the related item.

Internet Protocol

An Internet Protocol address, or IP address, identifies a node in a computer network that uses the Internet Protocol to communicate. The IP label is numerical.

Intrusion Detection System

An intrusion detection system (IDS), which can also be known as an intrusion prevention system (IPS) is a software application, or a device, that monitors a computer network, or system, for malicious activity or policy violations. Such intrusion activities, or violations, are typically reported either to a system administrator, or collected centrally by a security information and event management (SIEM) system.

JavaScript Object Notation

JSON is an open standard file format for data interchange. It uses human-readable text to store and transmit data objects, which consist of attribute-value pairs and arrays.

JSON web token

A JWT is an internet standard to create data with optional encryption and/or optional signature whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a private/public key.

Lua

Lua is a lightweight, high-level programming language designed for embedded use in applications, known for its simple syntax, efficiency, and its extensive use in game development.

Media Access Control

A MAC address is a unique identifier for a network interface controller (NIC). It is used as a network address in network segment communications. A common use is in most IEEE 802 networking technologies, such as Bluetooth, Ethernet, and Wi-Fi. MAC addresses are most commonly assigned by device manufacturers and are also referred to as a hardware address, or physical address. A MAC address normally includes a manufacturer's organizationally unique identifier (OUI). It can be stored in hardware, such as the card's read-only memory, or by a firmware mechanism.

Packet Capture

A pcap is an application programming interface (API) that captures live network packet data from the OSI model (layers 2-7).

Remote Terminal Unit

An RTU is a microprocessor-controlled electronic device that acts as an interface between a SCADA (supervisory control and data acquisition) system, or distributed control system, to a physical object. It transmits telemetry data to a master system, and uses messages from the master supervisory system to control connected objects.

Secure Shell

A cryptographic network protocol that let you operate network services securely over an unsecured network. It is commonly used for command-line execution and remote login applications.

Syslog

Syslog (System Logging Protocol) is a standard protocol used for message logging in computer networks and operating systems. It enables devices such as servers, routers, firewalls, and applications to send log messages to a central logging server, making it useful for monitoring, debugging, and security analysis.

Transmission Control Protocol

One of the main protocols of the Internet protocol suite.

Uniform Resource Identifier

A URI is a unique string of characters used to identify a logical or physical resource on the internet or local network.

Uniform Resource Locator

An URL is a reference to a resource on the web that gives its location on a computer network and a mechanism to retrieving it.

User Interface

An interface that lets humans interact with machines.

Virtual Local Area Network

A VLAN is a broadcast domain that is isolated and partitioned in a computer network at the data link layer (OSI layer 2).