How To Use The REST API In Nagios Network Analyzer 2026

Purpose

This document describes how to use the Nagios Network Analyzer (NNA) 2026 REST API to fetch data and programmatically manage the system. Most things that can be done in the UI can also be done via the API, so it provides a powerful method for interacting with the application, and for integrating Network Analyzer data and functions with other tools.

Basic Usage

To make a simple GET request to fetch data, you'd run the following from the command line of a machine with network access to your NNA server.

Replace `{NNA_server}` with the IP address or hostname of your Network Analyzer server, `{endpoint}` with the endpoint you wish to interact with, and `{your_API_key}` with your API key, which can be found at the bottom of the **My Profile** menu in the UI.

```
curl -X GET "http://{NNA_server}/api/v1/{endpoint}" \
  -H "Authorization: Bearer {your_API_key} \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Note that the trailing `| jq` is not required, but including it will nicely format the JSON data returned by GET requests.

A specific example:

To poll an NNA server with an IP address of 192.168.145.10 for details on all of the Sources configured on it, the call would be:

```
curl -X GET “http://192.168.145.10/api/v1/sources” \
  -H "Authorization: Bearer {your_API_key} \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Finding IDs

Many endpoints will call for an `{id}`, to pinpoint the exact object to target. IDs can be found by hitting an endpoint without passing additional fields. For example, in this return of a GET of `/sources`:

```
{
  "id": 1, ←-------------------------
  "port": 9001,
  "description": null,
  "name": "Cent 9 - fprobe",
  "flowtype": "netflow",
  "directory": "/usr/local/nagiosna/var/1",
  "lifetime": "24H",
  "is_active": 1,
  "status": true,
  "diskusage": "12M",
  "traffic": [ etc…
```

To fetch data only for this source, we'd append our call with it's ID, like this (note the trailing `1` after `/sources`):

```
curl -X GET "http://192.168.145.10/api/v1/sources/1" \
  -H "Authorization: Bearer {your_API_key} \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Passing Fields

In the case of POST, PUT, and PATCH calls, you'll need to pass various fields to define the parameters of the object you're creating or modifying. To see what fields are related to an object, you can run a GET request on an existing one. For example, to find the fields used by a Source, you could run the above GET command and review the output. This doesn't always provide exactly what you need, but is a good start.

A more exact approach you could take to determine what fields are used, and how to format them, is to use Wireshark. This method is detailed in full here in this document, though is not available if you've enabled TLS.

Details on how to format and pass fields in your requests are included in several of the examples in the upcoming pages.

Endpoints & Usage Examples

On the following pages you will find details on various NNA API endpoints, along with usage examples. Although every endpoint is not reflected, this directory covers the majority of them, including many useful options.

To see every available endpoint, you can review the api.php file on your NNA server:

```
cat /var/www/html/nagiosna/routes/api.php
```

Important: please do **not** edit this file in any way, doing so could cause serious issues.

Click any of the following links to go directly to sections of interest:

Sources and Groups | Nmap | Wireshark | Suricata | System | Users & Roles | Checks | Backups

Sources & Groups Endpoints

/api/v1/sources

- `GET sources/{id?}` – *get source data*
- `POST sources/` - *create a source*
- `PUT sources/{id}` – *update a source*
- `DELETE sources/{id?}` – *delete a source*
- `GET sources/{id}/summary/chart` – *get a summary chart for a specific source*
- `GET sources/summary/chart` – *get a summary chart for all sources for a timeframe*
- `GET sources/{id}/summary/talkers` – *get top talkers for a source*
- `POST sources/{id}/start` – *start a source*
- `POST sources/{id}/stop` – *stop a source*
- `POST sources/{id}/restart` – *restart a source*
- `POST sources/{id}/status` – *change source status*
- `POST sources/queries/run` – *run a query on a source*

/api/v1/source-groups

- `GET source-groups/{id?}` – *get source group data*
- `POST source-groups/` - *create a source group*
- `PUT source-groups/{id}` – *update a source group*
- `DELETE source-groups/{id?}` – *delete a source group*
- `GET source-groups/{id}/summary/chart` – *get summary chart for a source group*
- `GET source-groups/summary/chart` – *get summary chart for all source groups*
- `GET source-groups/{id}/summary/talkers` – *get top talkers for a specific source group*
- `POST source-groups/{id}/start` – *start all sources in a group*
- `POST source-groups/{id}/stop` – *stop all sources in a group*
- `POST source-groups/{id}/restart` – *restart all sources in a group*

Examples

Creating a Source

```
    curl -X POST "http://{NNA_server}/api/v1/sources" \
    -H "Authorization: Bearer {your_API_key}" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json" \
    -d '{
      "name": "New-Source",
      "lifetime": "24H",
      "port": 9999,
      "description": "description of source",
      "flowtype": "netflow"
    }'
```

Deleting a Source

```
  curl -X DELETE "http://{NNA_server}/api/v1/sources/{id}" \
    -H "Authorization: Bearer {your_API_key}" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json"
```

Starting, Stopping, and Restarting a Source

(Use `start`, `stop`, or `restart` at the end of the first line to choose your function)

```
curl -X POST "http://{NNA_server}/api/v1/sources/{id}/stop" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json"
```

Starting, Stopping, and Restarting All Sources in a Source Group

(Use `start`, `stop`, or `restart` at the end of the first line to choose your function)

```
curl -X POST "http://{NNA_server}/api/v1/source-groups/{id}/start" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Getting Source and Source Group Top Talkers

These examples will fetch Top Talkers for the last 24 hours. The timeframe can be customized with other options such as `-2%20hours`, `-1%20week`, and `-1%20month`.

Note that the range possible for this call is limited by the Raw Data Lifetime you chose for the target Source or Sources.

Source

```
curl -X GET "http://{NNA_Server}/api/v1/sources/1/summary/talkers\
?timeframe=-24%20hours" \
 -H "Authorization: Bearer {your_API_key}" \
 -H "Accept: application/json" | jq
```

Group

```
curl -X GET "http://{NNA_Server}/api/v1/source-groups/1/summary/talkers\
?timeframe=-24%20hours" \
 -H "Authorization: Bearer {your_API_key}" \
 -H "Accept: application/json" | jq
```

Getting Source Summary Chart Data

This example fetches data for the last 2 hours.

```
curl -X GET "http://{NNA_Server}/api/v1/sources/1/summary/chart?timeframe=-2+hours" \
 -H "Authorization: Bearer {your_API-key}" \
 -H "Content-Type: application/json" \
 -H "Accept: application/json" | jq
```

Nmap Endpoints

/api/v1/nmap/scans

- `GET nmap/scans/{id?}` – *get Nmap scan data*
- `POST nmap/scans/` - *create an Nmap scan*
- `DELETE nmap/scans/{id?}` – *delete an Nmap scan*
- `GET nmap/scans/{filename}/download` – *download an Nmap scan file*
- `POST nmap/scans/{id}/stop` – *stop an Nmap scan*
- `GET nmap/scans/{id}/statistics` – *get Nmap scan statistics*

/api/v1/nmap/scheduled-scans Endpoints

- `GET nmap/scheduled-scans/{id?}` – *get Nmap scheduled scan data*
- `POST nmap/scheduled-scans/` - *create Nmap scheduled scan*
- `PUT nmap/scheduled-scans/{id}` – *modify Nmap scheduled scan*
- `DELETE nmap/scheduled-scans/{id?}` – *delete Nmap scheduled scan*
- `GET nmap/scheduled-scans/{id}/chart` – *get Nmap scheduled scan chart*

/api/v1/nmap/profiles Endpoints

- `GET nmap/profiles/{id?}` – *get Nmap profile data*
- `POST nmap/profiles/` - *create Nmap profile*
- `PUT nmap/profiles /{id}` – *update Nmap profile*
- `DELETE nmap/profiles/{id?}` – *delete Nmap profile*

/api/v1/nmap/ndiffs Endpoints

- `GET nmap/ndiffs/{id?}` – *get Ndiff details*
- `POST nmap/ndiffs/` - *create an Ndiff*
- `DELETE nmap/ndiffs/{id?}` – *delete an Ndiff*
- `GET nmap/ndiffs/{filename}/download` – *download an Ndiff as a .txt file*

Examples

Start an Nmap Scan

This example will run a **Quick Scan +** using `nmap_profile_id` **7**.

To find other Profile IDs, `GET` the `/nmap/profiles` endpoint.

```
curl -X POST "http://{NNA_Server}/api/v1/nmap/scans" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
   "nmap_profile_id": "7",
    "title": "API-Scan-1",
    "scan_parameters": "{IP/Range/Network Segment} -sV -T4 -O -F -e ens18"
  }'
```

Download an Ndiff ast Text

You can `GET` the `/namp/ndiffs` endpoint to find the filenames.

```
curl -X GET
"http://{NNA_server}/api/v1/nmap/ndiffs/{ndiff_filename}.txt/download" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json"
```

Download Nmap Scan Results as Text

You can `GET` the `/namp/scans` endpoint to find the filenames.

```
curl -X GET "http://{NNA_server}/api/v1/nmap/scans/{filename}.txt/download" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json"
```

Wireshark Endpoints

/api/v1/wireshark

- `GET wireshark/interfaces` – *get interfaces available to Wireshark*
- `GET wireshark/version` – *get Wireshark version*
- `GET wireshark/latest-capture` – *get the latest Wireshark capture*
- `GET wireshark/overview` – *get data such as total captures and most recent capture*
- `GET wireshark/packet-details` – *get details of a specific packet*
- `GET wireshark/captures-over-time` – *get captures over time data*
- `GET wireshark/captures-metrics` - get capture metrics

/api/v1/wireshark/ring-buffers

- `GET wireshark/ring-buffers/` - *get ring buffer data*
- `POST wireshark/ring-buffers/` - *create a ring buffer*
- `PUT wireshark/ring-buffers/{id}` – *update a ring buffer*
- `POST wireshark/ring-buffers/{id}/start` – *start the ring buffer*
- `POST wireshark/ring-buffers/{id}/stop` – *stop the ring buffer*
- `GET wireshark/ring-buffers/{id}/packets` – *get ring buffer packets*

/api/v1/wireshark/captures

- `GET wireshark/captures/` - *get Wireshark captures*
- `POST wireshark/captures/` - *start a Wireshark capture*
- `POST wireshark/captures/import` – *import a PCAP*
- `DELETE wireshark/captures/{id?}` – *delete a Wireshark capture*
- `POST wireshark/captures/{id}/stop` – *stop a Wireshark capture*
- `GET wireshark/captures/{id}/download` – *download capture data*
- `POST wireshark/captures/{id}/scan-with-suricata` – *scan a capture with Suricata*
- `GET wireshark/captures/{id}/packets` – *get capture packets*
- `GET wireshark/captures/{id}/info` – *get capture info*
- `GET wireshark/captures/{id}/top-talkers` – *get capture top talkers*
- `GET wireshark/captures/{id}/protocol-usage` – *get capture protocol usage*
- `GET wireshark/captures/{id}/packet-size-distribution` – *get capture packet size distribution*

Examples

Start a Wireshark Capture

This example will run a 30 second capture on the defined interface.

```
curl -X POST "http://{NNA_Server}/api/v1/wireshark/captures" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
   "interface": "{interface}",
    "duration": "30s"
  }'
```

Get Latest Capture

```
  curl -X GET "http://{NNA_Server}/api/v1/wireshark/latest-capture" \
    -H "Authorization: Bearer {your_API_key}" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json" | jq
```

Get Capture Top Talkers

```
curl -X GET "http://{NNA_server}/api/v1/wireshark/captures/{id}/top-talkers" \
   -H "Authorization: Bearer {your_API_key}" \
   -H "Content-Type: application/json" \
   -H "Accept: application/json" | jq
```

Scan a Wireshark Capture With Suricata

```
curl -X POST "http://{NNA_server}/api/v1/wireshark/captures/{id}/scan-with-suricata" \
      -H "Authorization: Bearer {your_API_key}" \
      -H "Content-Type: application/json" \
      -H "Accept: application/json"
```

View Details of a Specific Packet

```
  curl -X GET "http://{NNA_Server}/api/v1/wireshark/packet-details\
  ?framenumber={frame_number}&filename=%2Fusr%2Flocal%2Fwireshark%{filename}.pcap" \
     -H "Authorization: Bearer {your_API_key}" \
     -H "Content-Type: application/json" \
     -H "Accept: application/json" | jq
```

Suricata Endpoints

/api/v1/suricata

- `GET suricata/` *- get basic Suricata status data*
- `POST suricata/` *- start a Suricata scan*
- `POST suricata/is-running` – *check running/stopped status*
- `GET suricata/interfaces` – *get Suricata interfaces*

/api/v1/suricata/data

- `GET suricata/data/` - *get data present in the eve.json*
- `GET suricata/data/full-log-json?id={id}` – *get a Suricata log entry as JSON*
- `POST suricata/data/import-pcap` – *import a PCAP into Suricata*
- `GET suricata/data/json-files` – *get Suricata log JSON files*
- `DELETE suricata/data/{filename}` - *delete a Suricata log file*

/api/v1/suricata/rules

- `GET suricata/rules/` - *get suricata rules*
- `GET suricata/rules/files` – *get names of rules files*
- `DELETE suricata/rules/files` – *delete a rules file*
- `GET suricata/rules/{sid}` – *get rule for a SID*
- `PUT suricata/rules/` - *create a rule*
- `DELETE suricata/rules/{sid?}` – *delete rule for a SID*
- `POST suricata/rules/import` – *import a rule*
- `PATCH suricata/rules/toggle` – *toggle rules on/off*
- `PATCH suricata/rules/{sid}/toggle` – *toggle a specific rule on/off*

/api/v1/suricata/config

- `GET suricata/config/` - *get Suricata config details*
- `PUT suricata/config/` - *update Suricata config*
- `GET suricata/config/version` – *get Suricata config version*

/api/v1/suricata/rulesets

- `GET suricata/rulesets/` *- get Suricata rulesets*
- `POST suricata/rulesets/` *- create a Suricata ruleset*
- `GET suricata/rulesets/{id}` *– view a specific ruleset*
- `PUT suricata/rulesets/{id}` *– update a ruleset*
- `PATCH suricata/rulesets/{id}` *– update a ruleset*
- `DELETE suricata/rulesets/{id?}` *– delete a ruleset*

/api/v1/suricata/alerts

- `GET suricata/alerts/` *- get data on all Suricata alerts*
- `GET suricata/alerts/total` *– get total Suricata alerts*
- `GET suricata/alerts/categories` *– get Suricata alert categories*
- `GET suricata/alerts/severities` *– get Suricata alert severities*
- `GET suricata/alerts/top` *– get top Suricata alerts*

Examples

Checking Suricata Status

```
  curl -X POST "http://{NNA_Server}/api/v1/suricata/is-running" \
    -H "Authorization: Bearer {your_API_key}" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json" | jq
```

Get Top Suricata Alerts

```
curl -X GET "http://{NNA_server}/api/v1/suricata/alerts/top" \
    -H "Authorization: Bearer {your_API_key}" \
    -H "Content-Type: application/json" \
    -H "Accept: application/json" | jq
```

Starting a Suricata Scan

```
curl -X POST "http://{NNA_Server}/api/v1/suricata" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
    "action": "start",
    "networkInterface": "{interface}"
  }'
```

Stopping a Suricata Scan

To stop Suricata, use the same call as above, but change the action field entry to “`stop`”, and delete the “`networkInterface`” field.

Fetching Specific Entries from the eve.json Log

This example grabs the 10 latest log entries that match the filter `192.168.1.53`

```
curl -X GET "http://{NNA_Server)/api/v1/suricata/data?\
page=1&per_page=10&sortBy=time&sortOrder=asc&filter=192.168.1.53+" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Turning a Suricata Rule On/Off

`toggleMode` options are `on`, `off`, and `toggle`. Using toggle will swap the state, while on and off will apply the chosen state regardless of current state.

```
curl -X PATCH "http://{NNA_Server}/api/v1/suricata/rules/{SID}/toggle" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
    "toggleMode": "toggle"
}'
```

Enable/Disable a Ruleset

You can poll `/suricata/rulesets` to find Ruleset IDs.

Change `true` to `false` in the `enabled` field to disable the ruleset.

```
curl -X PATCH "http://{NNA_Server}/api/v1/suricata/rulesets/{id}" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
    "enabled": true
}'
```

System Data

/api/v1/system

- `GET system/product-info` – *get product info*
- `GET system/update-check` – *check for updates*
- `GET system/installed-integrations` – *view installed integrations*
- `GET system/whois` – *get system WHOIS data*
- `GET system/reverse-dns` – *get system reverse DNS data*

/api/v1/system/status

- `GET /cpu` – *get CPU usage*
- `GET /memory` – *get memory usage*
- `GET /root-drive` – *get root drive space*

Examples

Check for Updates

```
curl -X GET "http://{NNA_Server}/api/v1/system/update-check" \
  -H "Authorization: Bearer {your_API_key}"  \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Check for Installed Integrations

```
curl -X GET "http://{NNA_Server}/api/v1/system/installed-integrations" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Check Memory Usage

```
curl -X GET "http://{NNA_Server}/api/v1/system/status/memory" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" | jq
```

Users & Roles Endpoints

/api/v1/users

- `GET users/{id?}` – *get user data*
- `POST users/` - *create a user*
- `PUT users/{id}` – *modify a user*
- `DELETE users/{id}` – *delete a user*
- `GET users/{id}/language` – *view user language*

/api/v1/roles

- `GET roles/{type?}` – *get role data*
- `POST roles/` - *create a role*
- `PUT roles/{id}` – *modify a role*
- `DELETE roles/{id}` – *delete a role*

Example

Creating a New User

This example creates a locally authenticated Admin role user with API access.

```
curl -X POST "http://{NNA_Server}/api/v1/users" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
    "username": "{username}",
    "email": "{email_address}",
    "password": "SecurePassword123!",
    "confirm_password": "SecurePassword123!",
    "active": true,
    "first_name": "Some",
    "last_name": "Person",
    "company": "Awesome Tech",
    "phone": "555-867-5309",
    "apiaccess": true,
    "language": "en_US",
    "type": "local",
    "theme": "dark",
    "role_id": 1,
    "auth_server_id": null,
    "auth_server_data": null
  }'
```

A note about User PUT calls:

Currently the User endpoint is one of the few that will need all required fields you would use in a POST to be passed in PUT calls.

Checks

/api/v1/checks endpoints:

- `GET checks/{id?}` – *get check data*
- `POST checks/` - *create a check*
- `PUT checks/{id}` – *modify a check*
- `DELETE checks/{id}` – *delete a check*
- `POST checks/{id}/force` – *force a check to run*
- `PATCH checks/{id}/toggle` – *toggle check active/inactive*

Examples

Fetching the Status of a Check

The return will be a Nagios exit code, so `0` = OK, `1` = Warning, and `2` = Critical.

```
curl -X GET "http://{NNA_Server}/api/v1/checks/1" \
-H "Authorization: Bearer {your_API_key}" \
-H "Content-Type: application/json" \
-H "Accept: application/json" | jq '.last_code'
```

Forcing a Check to Run

```
curl -X POST "http://{NNA_Server}/api/v1/checks/{id}/force" \
-H "Authorization: Bearer {your_API_key}" \
-H "Content-Type: application/json" \
-H "Accept: application/json"
```

Creating Checks

Checks are unique in that they require steps to be defined in calls. Suricata checks require 2 steps, which other checks require 3.

Creating a Suricata Check

```
curl -X POST "http://{NNA_Server}/api/v1/checks" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
    "check_type": "suricata",
    "step1": {
      "name": "Suricata-Nmap-API-Made",
      "association": {
        "type": "signature_id",
        "id": 9999999
      },
      "metric": "alert_count",
      "warning_threshold": 0,
      "critical_threshold": 1,
      "check_frequency": "Hourly",
      "lookback_period": "1d"
    },
    "step2": {
      "user": [1]
    }
  }'
```

Creating a Flow Check

```
curl -X POST "http://{NNA_Server}/api/v1/checks" \
-H "Authorization: Bearer {your_API_key}" \
-H "Content-Type: application/json" \
-H "Accept: application/json" \
-d '{
   "check_type": "flow_source",
   "step1":{
   "name": "Flow Check Example",
   "association": {"
      type":"source",
      "id":"1"}
  },
"step2":{
   "metric": "bytes",
   "forcecheck": false,
   "warning_threshold": "10",
   "critical_threshold": "500",
   "queries":[
     {"location": "destination",
      "location_type": "port",
      "location_bool": "is",
      "location_value": "80"}]
  },
"step3":{
   "user": [1]},
   "nagios":[],
   "snmp_receiver":[],
   "command":[]
}'
```

Backups Endpoints

/api/v1/backups

- `GET backups/` - *get backups data*
- `POST backups/` - *create a backup*
- `DELETE backups/{filename}` – *delete a backup*
- `GET backups/{filename}/download` – *download a backup file*

Example

Create a System Backup

```
curl -X POST "http://{NNA_Server}/api/v1/backups" \
  -H "Authorization: Bearer {your_API_key}" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json"
```

Using Wireshark as a Call Composition Aid

Some calls are quite complex (such as the /checks POST call detailed above), and determining the fields that must be passed in your call can be challenging.

This is an excellent use-case for Wireshark packet capture and analysis, which is conveniently available in Network Analyzer. Since the Network Analyzer executes functions via the API, all of the data you seek can be found in the packets created when tasks are executed in the UI. Note that this method will not work if you've configured TLS, as the API call packets will then be encrypted.

Full details on the Wireshark integration can be found here:

[Installing and Using Wireshark with Network Analyzer]

Before proceeding with the following approach, ensure that Wireshark is installed.

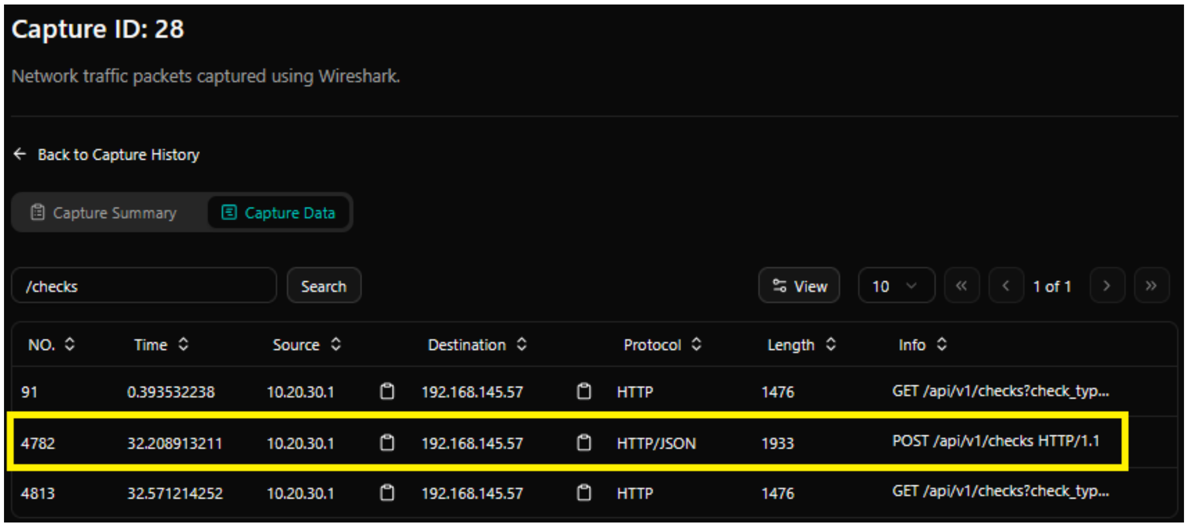
1. To begin, start a Wireshark capture of your primary Network Analyzer server interface, of a long enough duration to give you time to carry out the task you want to craft a call for.

2. Next, conduct the task in the user interface, which will create a packet containing the details you're seeking (eg create a certain type of check, modify a user, etc…).

3. Once you've taken the target action, you can stop the Wireshark scan if it hasn't already reached the end of the set duration and stopped automatically.

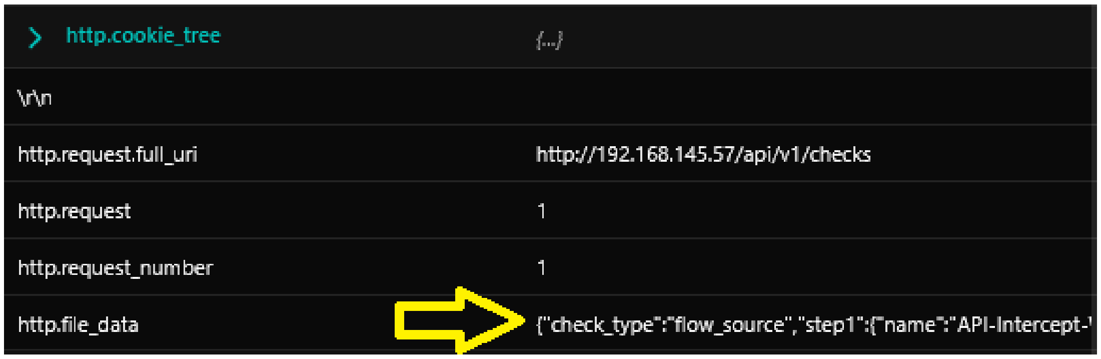
4. Go to **Wireshark > Capture History**, and click the eye icon next to the new entry.

5. Go to the **Capture Data** tab, and do a search for the call related to the action you took (eg search for something like `/checks` or `POST` if you created a check).



NO.	Time	Source	Destination	Protocol	Length	Info
91	0.393532238	10.20.30.1	192.168.145.57	HTTP	1476	GET /api/v1/checks?check_typ...
4782	32.208913211	10.20.30.1	192.168.145.57	HTTP/JSON	1933	POST /api/v1/checks HTTP/1.1
4813	32.571214252	10.20.30.1	192.168.145.57	HTTP	1476	GET /api/v1/checks?check_typ...

6. Click through once you've located the packet you're after, then go to the **http > http.cookie_tree > http.file_data** section of the **Advanced Details** tab.



> http.cookie_tree	{...}
\r\n	
http.request.full_uri	http://192.168.145.57/api/v1/checks
http.request	1
http.request_number	1
http.file_data	{"check_type":"flow_source","step1":{"name":"API-Intercept-'

7. Copy this data and paste it into a text editor to view the details and compose your call:

```
{"check_type":"flow_source","step1":{"name":"API-Wireshark-
Intercept","association":{"type":"source","id":"1"}},"step2":{"metric":"byte
s","forcecheck":false,"warning_threshold":"10","critical_threshold":"500","q
ueries":[{"location":"destination","location_type":"port","location_bool":"i
s","location_value":"80"}]},"step3":{"user":[4],"nagios":[],"snmp_receiver":
[],"command":[]}}
```

Finishing Up

This completes the documentation o Using the REST API in Nagios Network Analyzer 2026. If you have additional questions or other support-related questions, please visit us at our Nagios Support Forum, Nagios Knowledge Base, or Nagios Library:

Visit Nagios Support Forum Visit Nagios Knowledge Base Visit Nagios Library