NetWitness® Platform XDR

Version 12.0

NetWitness Shell User Guide

Contact Information

NetWitness Community at https://community.netwitness.com contains a knowledge base that answers common questions and provides solutions to known problems, product documentation, community discussions, and case management.

Trademarks

RSA and other trademarks are trademarks of RSA Security LLC or its affiliates ("RSA"). For a list of RSA trademarks, go to https://www.rsa.com/en-us/company/rsa-trademarks. Other trademarks are trademarks of their respective owners.

License Agreement

This software and the associated documentation are proprietary and confidential to RSA Security LLC or its affiliates are furnished under license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the copyright notice below. This software and the documentation, and any copies thereof, may not be provided or otherwise made available to any other person.

No title to or ownership of the software or documentation or any intellectual property rights thereto is hereby transferred. Any unauthorized use or reproduction of this software and the documentation may be subject to civil and/or criminal liability.

This software is subject to change without notice and should not be construed as a commitment by RSA.

Third-Party Licenses

NETWITNESS
Platform XDR

NetWitness® Platform XDR

Version 12.0

NetWitness Shell User Guide

July, 2022

NETWITNESS
Platform XDR

NetWitness® Platform XDR

Version 12.0

NetWitness Shell User Guide

Contents

NETWITNESS
Platform XDR

NetWitness® Platform XDR

Version 12.0

NetWitness Shell User Guide

NETWITNESS
Platform XDR

NetWitness® Platform XDR

Version 12.0

NetWitness Shell User Guide

NETWITNESS
Platform XDR

Shell

This guide describes the shell utility `nw-shell` that can be used to troubleshoot the operations of NetWitness management services like `security-server`, `investigate-server`, and `correlation-server`. It is the equivalent of the `NwConsole` utility used to interact with NetWitness capture services like Decoders and Concentrators. The shell utility is independent of the business logic of the service, and works the same with most NetWitness services.

Features

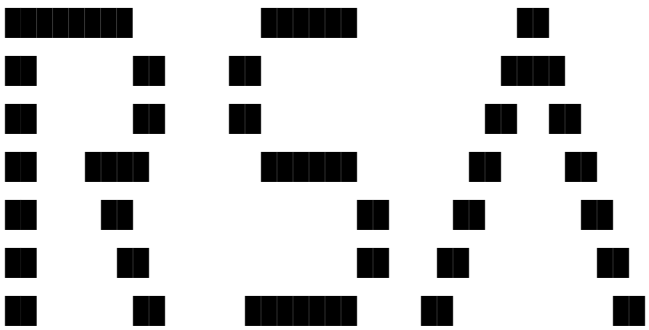
The `nw-shell` utility implements the following features:

- Supports secure connections to the local NetWitness service instances.
- Supports navigation of the service `tree` to explore its operational state (for more information, see Tree View).
- Provides an intuitive display of configuration, metrics, and health-check information to help with troubleshooting.
- Supports scripting to automate simple administration tasks in field deployments.
- Supports Linux, OSX and Windows terminals.

Installation

The `nw-shell` utility can be installed with the `rsa-nw-shell` RPM as shown here:

```
$ sudo yum install rsa-nw-shell
$ /usr/local/bin/nw-shell
```



```
RSA NetWitness Shell. Version: 4.0.0
See  "help" to list available commands, "help connect" to get started.
offline »
```

Usage

The primary goal of `nw-shell` is to help a human operator explore the runtime state of a NetWitness service. It is, essentially, an interactive program that invokes APIs on running NetWitness services. Each NetWitness service includes a system API that exposes its runtime state as a *logical* tree. The shell leverages the Tree API structure to present a hierarchical view of a service that is similar to a file system view. Users can navigate the tree by using the `cd` command to access directories, and can view or modify the corresponding configuration at the location, or invoke API methods and view the current state of components in the node.

The set of commands available to users at a given time depend on the current shell *context*, for example, their placement inside the *logical* tree. Certain commands, however, are always available, and we begin with a description of those commands.

System Commands

Shell *system* commands operate on the shell itself, instead of the nodes of a connected service. These commands are always available.

Command	Function	Example
`help [<command>]`	Provides help on available commands or a particular command.	`help` or `help connect`
`clear`	Clears the screen (shortcut `Ctrl-l`).	`clear`
`exit` or `quit`	Exits the shell.	`quit`
`history`	Displays the history of previously-run commands.	`history`

- Use the `help <command>` to explore the available commands. It is always available.
- `nw-shell` supports tab completion of a command, and the applicable parameter names wherever possible. For example, pressing the **Tab** key after typing **`e`** completes the command to `exit`.
- `nw-shell` also supports a non-interactive mode where it executes scripts from a provided file, used by specifying the absolute filepath prefixed by the `@` argument. See Shell for details.

Available Commands

```
Built-In Commands
        clear: Clear the shell screen.
        exit, quit: Exit the shell.
        help: Display help about available commands.
        history: Display or save the history of previously run commands
```

```
        script: Read and execute commands from a file.
        stacktrace: Display the full stacktrace of the last error.

Context Commands
      * cd: Change the current node. Usage: cd <path>
        connect: Connect to a service. One of --service or --port must be
specified.Usage: connect [--service <service>[.<id>]] [--broker
amqp://localhost/rsa/system] [--host localhost] [--port] [--insecure false]
      * where: Which service shell is connected to?

Token Commands
        login: Authenticate to a service. Usage: login [connect-parameters]
        login-insecure: Authenticate to a service providing user and password
on the command prompt. The password is recorded in the shell history so this
command must be used with care.Usage: login-insecure --user <user> --password
<password> [connect-parameters]
      * logout: Clear the authentication context: logout
      * whoami: Who am I?

Tree Node Commands
      * json: Print the current node as a JSON string
      * show: Pretty print the current node

Tree Node List Commands
      * config: Summarize configuration of the current subtree
      * health: Summarize health of the current subtree
      * ls: List the children of the current node. Usage: ls [<filter>] [--
values] [--types]
      * lsv: Shorthand for ls --values. Usage: lsv [<filter>] [--types]
      * method: Summarize methods of the current subtree
      * metrics: Summarize metrics of the current subtree
      * snapshot: Snapshot the current subtree

Tree Node Method Commands
      * invoke: Invokes the method that exists on the current method type
```

```
node. Usage: invoke [argument] [--file jsonfile]

Tree Node Value Commands
      * get: Get the value of the current node
      * set: Set the value of the current node. Usage: set <new-value>

Commands marked with (*) are currently unavailable.
Type `help <command>` to learn more.
```

Help, History Command Usage

```
offline » help connect
    connect - Connect to a service. One of --service or --port must be
specified. Usage: connect [--service <service>[.<id>]] [--broker
amqp://localhost/rsa/system] [--host localhost] [--port] [--insecure false]
    ...
    ...
offline » history
help
help connect
history
offline »
```

You can navigate previously-typed commands in `nw-shell` by using the Up and Down arrow keys, which can help minimize typing by recalling previously-executed commands.

Authentication Commands

As an administration and monitoring tool, it is important that `nw-shell` authenticates users before handing them control over a running service. The following commands manage the shell authentication context:

Command	Function
`login`	Authenticates and retains tokens for future interactions.
`logout`	Clears the security context and token.
`whoami`	Prints a summary of the current identity.

The `login` command can be used to authenticate the user and establish an identity. Service operators can log in once (against the NetWitness Security Server) and then use the token to connect to multiple services and perform administration based on the roles assigned. This single-sign-on workflow is simplified by separating the `login` and `connect` commands to allow the operator to authenticate once (using `login`) and then switch services seamlessly using `connect`.

In general, the `login` command takes the same parameters as `connect` (described below) to specify the service that performs the credential validation. For example:

```
offline » login
user: admin
password: ********
admin@offline » connect --service respond-server
INFO: Connected to respond-server (23e1dab7-0658-41a5-bb1e-d716a37d5ea5)
admin@respond-server:Folder:/rsa »
admin@respond-server:Folder:/rsa » connect --service investigate-server
INFO: Connected to investigate-server 2f21db20-4b50-48cf-8f7f-c0be0d1d1d12)
admin@investigate-server:Folder:/rsa »
```

The current logged-in identity can be confirmed at any point using the `whoami` command. For example:

```
admin@offline » whoami
```

Subject	admin
Issued By	security-server-ebeb90da-761b-4f14-b9ec-30410499d2b8
Issued At	2018-10-04T17:22:52.041Z
Expires After	2018-10-05T05:22:52.041Z
Roles	[ebeb90da-761b-4f14-b9ec-30410499d2b8]

The shell security context can be cleared by using the `logout` command.

```
security-server:Folder:/rsa » logout
security-server:Folder:/rsa » whoami
You are not logged in.
```

Some operations, such as setting certain configuration properties or invoking a method, require a certain RBAC. To perform such privileged operations, you need to authenticate with an identity that has the necessary permissions. For example:

```
security-server:Configuration » /rsa/logging/operations/max-file-count » get
10
security-server:Configuration:/rsa/logging/operations/max-file-count » set 15
ERROR: Failed to set the node value: Access is denied
security-server:Configuration:/rsa/logging/operations/max-file-count » login
user: admin
```

```
password: **********
admin@security-server:Configuration:/rsa/logging/operations/max-file-count »
set 15
security-server:Configuration:/rsa/logging/operations/max-file-count » get
15
```

Context-Changing Commands

The following commands change the current shell context.

Command	Function	Options	Examples
`connect`	Connect to a service	`servicehost`, `port`, `http`, `broker`	`connect --service security-server.d6a55b48...`
`cd`	Change context to a node		`cd /rsa/security`, `cd ../sys`, `cd ..`, `cd`

Connecting to a Service

The shell can connect to services over AMQP or HTTP(S), however, AMQP is the preferred and default option.

To connect to a specific instance of a service, `name` and `serviceId` needs to be supplied in the format:
`connect --service {service-name}.{serivceId}`
For example, `foo-server.d6a55b48-6103-46bd-9ead-3b4d589b302b`.

If the service identifier is skipped, it is assumed to be `any`, for example, `connect --service foo-server` will connect with any service named `foo-server` that is connected to the AMQP broker.

Change Node

The `cd` command can be used to change the current node. Just like `cd` on a file system shell, it takes relative or absolute paths as input and changes the current context to the node at that path.

```
offline » cd
Command 'cd' was found but is not currently available because you are not
connected to any service.


offline » connect
```

```
security-server:Folder:/rsa » cd log
security-server:Folder:/rsa/logging » cd ../security
security-server:Component:/rsa/security » cd /rsa/security/fips-mode
security-server:Configuration:/rsa/security/fips-mode » cd
security-server:Folder:/rsa »
```

The shell prompt summarizes and presents the user's current context. It starts off with `offline`, and once the shell is connected to a service, it displays the service name it is connected to, and the type and the path of the current node. Once the user is authenticated, the userId is included in the prompt.

Invoking `cd` before connecting to a service reminds the user that some commands work only in certain contexts.

Note: Some commands are available only in certain contexts. For example, `cd` works only when the shell is *online*. The Help command lists all the commands, but commands marked with (*) are unavailable in the given context.

Certain node-specific commands are enabled only when the current node is of a certain type. For example, method nodes support a command `invoke`, which is not enabled for any other node type.

Node Display Commands

The following commands are available with all node types. They do not take any arguments, and display the node details for the user to review.

Command	Function
`show`	Pretty-print the current node.
`json`	Print the current node as a JSON string.

For example:

```
security-server:Configuration:/rsa/security/authentication/token-
lifetime » show
```

Configuration	/rsa/security/authentication/token-lifetime
value	10 hours
valueType	com.rsa.asoc.launch.api.helpers.Seconds
defaultValue	10
description	The time-to-live on a token.

```
security-server:Configuration » /rsa/security/authentication/token-lifetime »
json
{
  "path"  :  "/rsa/security/authentication/token-lifetime",
  "type"  :  "Configuration",
```

```
  "value"  :  "10 HOURS",
  "parent"  : {
    "path"  :  "/rsa/security/authentication",
    "type"  :  "Component"
  },
  "attributes"  : {
    "defaultValue"  : 10,
    "valueType"  :  "com.rsa.asoc.launch.api.helpers.Seconds",
    "description"  :  "The time-to-live on a token."
  }
}
```

The output of `json` dumps the API payload and may contain more details than those shown by `show`.

Node Value Commands

`Configuration`, `Metric` and `Health` nodes have values. Their current values can be obtained using the `get` command.

Command	Function	Examples
`get`	Get the current value of the node.	
`set`	Update the value of the node.	`set false`, `set '{"enabled": false}'`

For example:

```
security-server:Gauge:/rsa/process/hostname » get
"hostxyz.corp.emc.com"
security-server:Gauge:/rsa/process/hostname » cd /rsa/transport/http/secure
security-server:Configuration:/rsa/transport/http/secure » get
true
```

The value of a `Configuration` node can be changed by invoking the `set` command. For example:

```
admin@security-server:Configuration:/rsa/security/pki/tls-protocols »
show
```

Configuration	/rsa/security/pki/tls-protocols
value valueType description	[TLSv1.2] java.lang.String[] This property controls the TLS protocol versions supported by the applications.

```
admin@security-server:Configuration:/rsa/security/pki/tls-protocols » set '
["SSLv3"]'
admin@security-server:Configuration:/rsa/security/pki/tls-protocols » value
[
  "SSLv3"
]
```

Node List Commands

`Folder`, `Component` and `Method` nodes can contain other nodes as their children. You can use the `ls` command to list the children nodes, their types, and where available, their current values.

Command	Function	Options	Examples
`ls <filter>`	List the child nodes of the current node.	`types`, `values`	`ls`, `ls cfg`, `ls --values`
`lsv <filter>`	Shorthand for `ls <filter> --values`	`types`	`lsv metrics ls name:fips`

For example:

```
security-server:Component:/rsa/process » ls
jvm                       Component
modules                   Component
current-time-utc          Gauge
current-time-utc-pretty   Gauge
fips140-mode              Gauge
hostname                  Gauge
mode                      Gauge
service-id                Gauge
service-name              Gauge
status                    Gauge
uptime                    Gauge
uptime-pretty             Gauge
version                   Gauge
version-full              Gauge
version-raw               Gauge
```

```
ready                    Method
shutdown                 Method

security-server:Component:/rsa/process » ls comp
jvm      Component
modules  Component

security-server:Component:/rsa/process » ls name:uptime
uptime         Gauge
uptime-pretty  Gauge

security-server:Component:/rsa/process » ls name:uptime --values
uptime         Gauge  3713854
uptime-pretty  Gauge  1 hour 1 minute 53 seconds

security-server:Component:/rsa/process » lsv
jvm
modules
current-time-utc         1483978142782
current-time-utc-pretty  2017-01-09T16:09:02.782Z
fips140-mode             true
hostname                 hostxyz.corp.emc.com
mode                     Normal
service-id               1fb7572a-4d87-497e-a4da-802819c10a72
service-name             no-op-server
status                   Running
uptime                   512064
uptime-pretty            8 minutes 32 seconds
version                  0.0
version-full             0.0.0.0
version-raw              0.0.0.0
ready
shutdown
```

Listing commands are not available on nodes that do not have children (for example, configuration or metric nodes):

```
security-server:Component:/rsa/process » cd hostname
security-server:Gauge:/rsa/process/hostname » ls
Command 'ls' exists but is not currently available because you are not logged
in.
```

Note: Values displayed in `ls` and `lsv` may be truncated for presentation purposes. Use `get` on the value node to get the complete (unaltered) value of a node.

Method Node Commands

The `method` command shows all the available methods and all the sub-folders under the current node.

```
security-server:Component:/rsa/process » method
/rsa/process/ready
/rsa/process/shutdown
```

On a node that is of `Method` type , the `invoke` command will invoke the method. For example:

```
security-server:Method:/rsa/process/ready » invoke
{
  "ready" : true,
  "serviceId" : "84f36740-5ae7-409f-b14f-b17e98703983",
  "marketingVersion" : "0.0"
}
```

If the `method` signature takes `input`, then `invoke <input>` will invoke the `method` with given parameters.

```
security-server:Method:/rsa/health/get » show
```

Method	/rsa/health/get
output	java.util.List<com.rsa.asoc.launch.api.health.api.HealthStatus>
input	java.lang.String
description	Get the details of a single composite health check @param id The health check id @return The composite health check

Metric	Value
failed	0
invoked	0
timer	0.0

```
Method:/rsa/health/get » invoke 'rsa.health.checks.security-pki'
[
  {
    "name" : "rsa.security.pki.pki-health",
    "status" : "Unhealthy",
    "details" : {
      "Reason" : "Using a self-signed certificate"    }
  }
]
```

For inline `method` execution with input payload, in cases where the input contains any special characters such as `\, \n, \r, \t , ",` the input needs to be properly escaped with a backslash (`\`).

In cases where `method` takes a complex payload, payload can be supplied with a file reference. In this case, payload does not need any special escaping.

```
Method:/rsa/configuration/collections/register » invoke --file
/tmp/jsonfile.txt
```

Run State of a Service

Once `nw-shell` is connected to a service, commands like `health`, `metrics`, `config` or `snapshot` can be used to retrieve the current state of the service at the current node, and all the sub-folders under the current node.

Health

The `health` command lists the health of the components, and the all the sub-folders, under the current node.

```
admin@security-server:Component:/rsa/process » health
/rsa/process/jvm/memory-health      Healthy
/rsa/process/modules/module-health  Healthy
```

Respond Server APIs

You can use `nw-shell` to retrieve and modify the incident details. In order for this to work, make sure that the relevant APIs are exposed. You can perform following actions using these APIs.

Retrieve Incidents

Retrieve Incident Details

Retrieve Incidents

Add Alerts to an Incident
Delete Alerts from an Incident

Retrieve Incidents

You can retrieve incidents using a single criteria. The criteria can include name, priority, risk score, status, alert count, etc. The `invoke` command returns all the incidents with the matching criteria.

Note: You must export the output to a file to avoid rollover in the shell output. You can query using single meta filters only, for example, filter by name or priority. Depending on the load, a large number of results might cause the request time-out.

Sample Query

```
admin@respond-server:Folder:/rsa » cd respond/incident
admin@respond-server:Component:/rsa/respond/incident » cd find-incident-by-criteria
admin@respond-server:Method:/rsa/respond/incident/find-incident-by-criteria »
invoke '["<meta name>", "<meta value>", "<number of records>"]'
```

Examples

The following example command reads the query from an input file and exports the results to an output file.

```
invoke --file /root/incident_req.txt outputFile /root/output_incidents.json
```

In the above example, the `incident_req.txt` file contains the following information.

```
[root@SA ~]# cat incident_req.txt
["status", "ASSIGNED", "500"]
```

The output is exported to the `output_incidents.json` file that contains 500 incidents with ASSIGNED status.

The following are some more examples:

- Retrieve incidents with `testinc for rule` as name and a maximum of 10 incidents.
  `["name", "testinc for rule", "10"]`
- Retrieve incidents with HIGH priority and a maximum of 50 incidents.
  `["priority", "HIGH", "50"]"]`
- Retrieve incidents with ASSIGNED status and a maximum of 500 incidents.
  `["status", "ASSIGNED", "500"]`

Retrieve Incident Details

The `invoke` command is used to list the details of specific incidents.

Sample Query

```
admin@respond-server:Folder:/rsa » cd respond/incident
admin@respond-server:Component:/rsa/respond/incident » cd find-by-ids
```

```
admin@respond-server:Method:/rsa/respond/incident/find-by-ids » invoke '["INC-1", "INC-2"]'
```

The output is incident details in JSON format displayed in `nw-shell`.

Examples

You can use an input file containing the request to list the incident details.

```
admin@respond-server:Method:/rsa/respond/incident/find-by-ids » invoke --file /tmp/findIncidents.txt
```

The output is incident details in JSON format displayed in `nw-shell`.

In the above example, the file `/tmp/findIncidents.txt` file contains the following information.

```
[root@NWAPPLIANCE27144 ~]# cat /tmp/findIncidents.txt

["INC-10", "INC-12"]
```

You can store the incident details to an output file.

```
admin@respond-server:Folder:/rsa » cd respond/incident
admin@respond-server:Component:/rsa/respond/incident » cd /rsa/respond/alert/add-alerts-to-incident
admin@respond-server:Method:/rsa/respond/incident/find-by-ids » invoke '["INC-1", "INC-2"]' outputFile /root/result_output.json
```

The output is incident details in JSON format stored in the `result_output.json` file.

Retrieve Alerts

You can retrieve alerts using a single criteria. The criteria can include any field that is part of the alert record (alert name, alert type, alert source, etc). The `invoke` command returns the alert with the matching criteria.

Note: You must export the output to a file to avoid rollover in the shell output. You can query using single meta filters only, for example, filter by name or type. Depending on the load, a large number of results might cause the request time-out.

Sample Query

```
admin@respond-server:Folder:/rsa » cd respond/alert
admin@respond-server:Component:/rsa/respond/alert » cd find-alerts-by-criteria
admin@respond-server:Method:/rsa/respond/alert/find-alerts-by-criteria »
invoke '["<meta name>", "<meta value>", "<number of alerts to retrieve>",
"<list of alert fields (include fields)>"]'
```

Note: If you provide `<null>` as the `<list of alert fields>` value, the output returns all the alert fields.

Examples

The following example command reads the query from an input file and exports the results to an output file.

```
invoke --file /root/alerts_req.txt outputFile /root/output_alerts.json
```

In the above example, the `alerts_req.txt` file contains the following information.

```
[root@SA ~]# cat alerts_req.txt
["alert.source","Event Stream Analysis", "20", "alert.severity"]
```

The output is exported to the `output_alerts.json` file that contains 20 alerts from Event Stream Analysis and the alert severity field included.

The following are some more examples:

- Retrieve 20 alerts from Event Stream Analysis with the alert severity field included.
  `["alert.source","Event Stream Analysis", "20", "alert.severity"]`
- Retrieve 20 alerts from Event Stream Analysis with all the alert fields included.
  `["alert.source","Event Stream Analysis", "20", "null"]`

Add Alerts to an Incident

The `invoke` command is used to add alerts to an existing incident. You will require alert IDs and Incident ID.

Sample Query:

```
admin@respond-server:Folder:/rsa » cd respond/alert

admin@respond-server:Component:/rsa/respond/incident » cd
/rsa/respond/alert/add-alerts-to-incident
admin@respond-server:Method:/rsa/respond/alert/add-alerts-to-incident » invoke
'["INC-1","603dcc5091e818391ee57e1d","603dcc5091e818391ee57e1c",
"603dcc5091e818391ee57e1e"]'
```

Alternatively, you can specify the add alert query in an input file.

```
invoke --file /tmp/add_alerts.txt
```

In the above example, the `/tmp/add_alerts.txt` file contains the following information.

```
[root@NWAPPLIANCE27144 ~]# cat /tmp/add_alerts.txt

["INC-1","603dcc5091e818391ee57e1d","603dcc5091e818391ee57e1c",
"603dcc5091e818391ee57e1e"]
```

The first parameter is the incident ID (INC-1). The rest of the parameters are alert IDs (for example, 603dcc5091e818391ee57e1e) to be added to the incident mentioned in the first parameter.

The output is a list of alert IDs added to the incident.

Delete Alerts from an Incident

The `invoke` command is used to remove alerts from an existing incident. You will require alert IDs and Incident ID.

Sample Query:

```
admin@respond-server:Folder:/rsa » cd respond/alert

admin@respond-server:Component:/rsa/respond/incident » cd
/rsa/respond/alert/delete-alerts-by-alert-ids
admin@respond-server:Method:/rsa/respond/alert/delete-alerts-by-alert-ids »
invoke '["INC-1","603dcc5091e818391ee57e1d","603dcc5091e818391ee57e1c",
"603dcc5091e818391ee57e1e"]'
```

Alternatively, you can specify the delete alert query in an input file.

```
invoke --file /tmp/delete_alerts.txt
```

In the above example, the file `/tmp/delete_alerts.txt` file contains the following information.

```
[root@NWAPPLIANCE27144 ~]# cat /tmp/delete_alerts.txt

["INC-1","603dcc5091e818391ee57e1d","603dcc5091e818391ee57e1c",
"603dcc5091e818391ee57e1e"]
```

The first parameter is the incident ID (INC-1). The rest of the parameters are alerts IDs (for example, 603dcc5091e818391ee57e1e) to be removed from the incident mentioned in the first parameter.

The output is a list of alert IDs removed from the incident.

Metrics

The `metrics` command lists all of the available metrics, and all the sub-folders, under the current node.

```
admin@respond-server:Component:/rsa/tree/node » metrics
/rsa/tree/node/get/invoked   14
/rsa/tree/node/get/timer     5666541.077918259
/rsa/tree/node/list/invoked  12
/rsa/tree/node/list/timer    6652400.379371729
```

In the above example, `/rsa/tree/node/get/invoked 14` shows that the method `get` is invoked `14` times, and `timer` shows that it took on average `5666541.077918259` nano seconds to process the `get` request.

Using `cd` into the `timer` node shows more details, such as rate of requests, for the current node for per-minute and five minutes. For example:

```
admin@respond-server:Timer:/rsa/tree/node/get/timer » show
```

Timer	/rsa/tree/node/get/timer
value	3267513.7521543643
valueType	java.lang.Double
stdDeviation	4381863.808014679
fiveMinuteRate	0.018725581010540414
max	181326026
count	27
sampleSize	27
oneMinuteRate	0.038516144812437314
min	810240
fifteenMinuteRate	0.009425563630894371
median	2146170.0
meanRate	5.189274295248098E-4
mean	3267513.7521543643

Config

The `config` command shows all the configuration properties in the current node and all the sub-folders under the current node.

```
admin@respond-server:Component:/rsa/logging » config
/rsa/logging/audit/max-file-count       10
/rsa/logging/audit/max-file-size        10 MB
/rsa/logging/forward/categories         [Audit]
/rsa/logging/forward/destination        SYSLOG_UDP
/rsa/logging/forward/enabled            true
/rsa/logging/forward/host               localhost
/rsa/logging/forward/port               50514
/rsa/logging/forward/secure             false
/rsa/logging/levels
/rsa/logging/operations/max-file-count  10
/rsa/logging/operations/max-file-size   10 MB
```

Snapshot

The `snapshot` command combines `metrics`, `configuration`, and `health` of the current state of the service node and all the sub-folders under the current node. This command is very useful for troubleshooting purposes. When opening a support case, take a `snapshot` dump at the root node `\rsa` of a service that might be having issues, and attach it to the case.

```
admin@security-server:Component:/rsa/security » snapshot
/rsa/security/account/external/get-all/invoked      16
/rsa/security/account/external/get-all/timer        NaN
/rsa/security/account/external/get/failed           6
/rsa/security/account/external/get/invoked          6
/rsa/security/account/external/get/timer            NaN
/rsa/security/account/force-password-change/invoke  1
/rsa/security/account/force-password-change/timer   NaN
/rsa/security/account/get/invoked                   123
/rsa/security/account/get/timer                     902163.0
/rsa/security/account/password-policy/cannot-inclu  false
/rsa/security/account/password-policy/min-chars     8
/rsa/security/account/password-policy/min-lower-ch  0
/rsa/security/account/password-policy/min-non-lati  0
/rsa/security/account/password-policy/min-numeric-  0
/rsa/security/account/password-policy/min-special-  0


----
```

Scripting

`nw-shell` supports non-interactive executions that can take commands from files. For example:

```
> cat /tmp/foo.script
connect --service security-server
cd /rsa/security/pki/ciphers
get
> time nw-shell @/tmp/foo.script
INFO: Connected to security-server (52c7b92c-23d5-4b3e-9973-31d8b1b27ec4)
[
```

```
  "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256",
  "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
  "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
  "TLS_DHE_RSA_WITH_AES_128_CBC_SHA"
]
nw-shell    5.89s user 0.33s system 305% cpu 2.032 total
```

`nw-shell` also has a built-in `script` which can be used to execute scripts in the shell once the shell is launched.

```
 offline » script --file /tmp/foo.script
 INFO: Connected to security-server (52c7b92c-23d5-4b3e-9973-31d8b1b27ec4)
 [
   "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
   "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256",
   "TLS_DHE_RSA_WITH_AES_128_GCM_SHA256",
   "TLS_RSA_WITH_AES_128_CBC_SHA256"
 ]
```

Note: `nw-shell` is primarily meant to help a human to explore the runtime state of a NetWitness Platform service. For most other purposes, it is usually best to use the service published APIs.

Troubleshooting Commands

The `nw-shell` utility provides troubleshooting commands to help recover from misconfigurations that may cause the service to work improperly.

fix-keystore

When a service's keystore is corrupt or its SSL trust is broken with the rest of NetWitness, you can use the `fix-keystore` command to fix this issue. This command reads the keystore file of the services running on the host and attempts to reestablish trust with the NetWitness Platform CA. For example, the following is the command to repair the respond-server 's keystore:

```
>> fix-keystore --service respond-server
```

This command repairs the keystore `keystore.p12` that is located at `/etc/netwitness/respond-server`. It does not modify the service's original keystore. The repaired keystore can be found with the suffix

`.good`, for example, `keystore.p12.good`. If no fixes can be made to the keystore, this command does not write the new `keystore.p12.good` file.

Once the keystore repair is successful, the system admin can restore the service's communication by renaming `keystore.p12.good` to `keystore.p12` and restart the service.

print-keystore

The `print-keystore` command prints the requested NetWitness Platform Service's keystore certificates to the given output file in JSON format. Unless verbose is enabled, only a few key attributes of the certificates are printed to the file. For example, the following command prints the respond-server's keystore certificates.

```
>> print-keystore respond-server outputFile /tmp/crts
```

With verbose enabled:

```
>> print-keystore respond-server outputFile /tmp/crts verbose
```

reconstruct-keystore

The `reconstruct-keystore` command can be used when a service's keystore needs to be restored to its original clean state.

The following command reconstructs the respond-server's keystore to the new file `/etc/netwitness/respond- server/keystore.p12.new` (it does not modify the original keystore).

```
>> reconstruct-keystore --service respond-server
```

To restore the service back to its healthy state, restore `keystore.p12.new` to `keystore.p12`, and restart the service.

Advanced Customization

The following JVM system properties can be used to customize certain aspects of the `nw-shell` presentation:

System Property	Controls	Default Value
`console.width`	The width of text.	100
`console.colors`	The use of colors.	true
`console.prompt`	The prompt.	`%s%s%s »`
`timeout`	The API timeout.	30s

These can be specified using the `JAVA_OPTS` environment variable. For example:

```
> export JAVA_OPTS="-Dtimeout=100s -Dconsole.width=40 -
Dconsole.prompt=%s%s%s>"
> export JAVA_OPTS="-Dconsole.colors=false"
> nw-shell
```

The API timeout is a time unit which can use typical time unit values such as 30 MINUTES, 30 mins, 30 m, 5 seconds, 5 sec, 5s, 5, 8000 milliseconds, or 8000 ms (ms = milliseconds).

The value of `console.prompt` must contain placeholders (for example, `%s`) for three strings which are replaced, in order, by:

1. The node type
2. A separator (`:`)
3. The node path

Setting `console.colors` to `false` will not turn off *all* use of colors, it only disables colors used by `nw-shell` in its output. The underlying libraries that the program uses may still output some text in color.

Tree View

A NetWitness service, such as `security-server` or `investigate-server`, is a collection of components working together to implement the goals of the service. This topic describes the support for exposing these components and their associated elements (for example, configuration parameters, metrics, API methods, health checks, and so on) in a `tree-view`. The `tree-view` is a hierarchically-nested arrangement of `tree-nodes`, each of which corresponds to a distinct functional area of the service. The arrangement enables the end-user to navigate the nodes at run-time and to explore the current state of the service. See Shell for an interactive tool that uses the tree-view.

Features

The `tree-view` implements the following features:

- Supports encapsulation of functionally related elements under the same `tree-node`.
- Supports hierarchical linkage of `tree-node` s to enable navigation from one node to another.

Implementation

The current implementation of the `tree-view` is described in the following sections.

Node Types

The `tree-view` is comprised of `tree-node` instances taken from the following types.

Type	Represents
`Component`	A service component with associated configurations, metrics, health, APIs, etc.
`Configuration`	A configuration parameter that controls some aspect of the service behavior (for example, `rsa.filesystem.prefix`).
`Metric`	A *quantitative* measure of some aspect of the component operations (for example, Requests Processed).
`Health`	A *qualitative* assessment of some aspect of the component (for example, "Using too much memory").
`Method`	An API that can be invoked remotely to interact with the component (for example, `/rsa/security/pki/setCertificate`)
`Parameter`	A payload parameter that is an input or an output from an API.
`Folder`	A logical grouping of nodes that are related to one another but are not a component (for example, the root `/rsa`).

Node Structure

All `tree-nodes` have the same structure. In particular, they include the following elements:

Item	Information
`path`	The node path is a backslash (/) separated path, for example, `/rsa/security/pki/api-secure`.
`type`	The type of the node, taken from the previous table.
`attributes`	A map of string keys to attribute values. The available keys depend on the type of the node.
`value`	The value of this node. This can be null if the node has no associated value (or if it has a null value).
`children`	A list of child nodes, if the node has any.
`parent`	The parent node. This is only null for the root node.