NetWitness® Platform XDR

Version 12.3.0.0

NwConsole User Guide

NETWITNESS

Platform XDR

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

This product may include software developed by parties other than RSA. The text of the license agreements applicable to third-party software in this product may be viewed on the product documentation page on NetWitness Community. By using this product, a user of this product agrees to be fully bound by terms of the license agreements.

Note on Encryption Technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when using, importing or exporting this product.

Distribution

Use, copying, and distribution of any RSA Security LLC or its affiliates ("RSA") software described in this publication requires an applicable software license.

RSA believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." RSA MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Miscellaneous

This product, this software, the associated documentations as well as the contents are subject to NetWitness' standard Terms and Conditions in effect as of the issuance date of this documentation and which can be found at https://www.netwitness.com/standard-form-agreements/.

© 2023 RSA Security LLC or its affiliates. All Rights Reserved.

September, 2023

Contents

Access NwConsole and Help

NetWitness Console, also known as NwConsole, is a multi-platform terminal application that provides powerful tools and command line access to Core services, such as Decoder, Log Decoder, Concentrator, Broker, and Archiver. While most users complete their tasks and investigations through the NetWitness user interface, some advanced users, such as administrators and developers, require direct access to the services without going through the user interface. NwConsole enables you to enter commands from the command line or run multiple commands from a file.

This topic describes how to access NwConsole and view the internal help within NwConsole.

Extensive help information is available within the NetWitness console, also known as NwConsole. You can access this help from the NetWitness command line.

Prerequisites

All NetWitness appliances have the NwConsole application installed. You can also install it on Windows, Mac, and CentOS to connect and interact with a Core service.

NwConsole is available from the command line on a NetWitness appliance. If you are accessing a Core appliance remotely, you need to have the NetWitness Console application installed on a Windows, Mac, or CentOS machine. To obtain the NetWitness Console application installer, contact NetWitness Customer Care.

Access NwConsole

To run NwConsole from the command line on a NetWitness appliance or on a terminal emulator, at the <$> prompt, type `NwConsole` (Linux) or `nwconsole` (Windows). The actual command is `NwConsole`, but Windows is not case sensitive. NetWitness Console is displayed as shown in the following example.

```
Last login: Thu Sep 24 14:00:42 on console
usxx<username>m1:~ <username>$ NwConsole
RSA NetWitness Platform Console 11.2.0.0.6105
Copyright 2001-2020, RSA Security Inc. All Rights Reserved.

Type "help" for a list of commands or "man" for a list of manual
pages.
>
```

View Help

NwConsole provides help on individual commands as well as help on specific topics.

Caution: To get the latest information, view the command and help topics within NwConsole.

View a List of Commands

To view a list of available commands and their descriptions, at the (>) prompt, type `help`. The following example shows a list of available commands.

```
> help
Local commands:
    avro2nwd      - Convert AVRO files to NWD files
    avrodump      - Display schema and contents of AVRO file (for debugging)
    blockspeed    - Tests various write block sizes to determine best setting
    compileflex   - Compile all flex parsers in a directory
    createflex    - Create a flex parser that matches tokens read from a file
    dbcheck       - Perform a database integrity check over one or more
                    session, meta, packet, log or stat db files
    diskspeed     - Measures the speed of the disk(s) mounted at a specified
                    directory
    echo          - Echos the passed in text to the terminal
    encryptparser - Encrypt all parsers in a directory
    feed          - Create and work with feed files
    fmanip        - Manipulate a file with XOR and check for embedded PEs
    hash          - Creates or verifies hashes of database files
    help          - Provides help information for recognized console commands
    history       - Displays, erases or executes a command in the command
                    history
    httpAggStats  - Tests HTTP aggregation and reports statistics as it
                    continues
    log           - Perform operations on a log database
    logParse      - Parse line delimited logs on stdin and post results to
                    stdout
    logfake       - Create a fake log pcap file
    lua           - Execute a lua script
    makec3        - Generate C3 Test Data
    makepcap      - Convert packet database files to pcap or log files
    man           - Displays a list of topics or opens a specific manual page
                    on a topic
    metaspeed     - Tests read performance over an existing meta db
    netbytes      - Display statistics on network interface utilization
    nwdstrip      - Convert full NWD file into just session and meta file
    pause         - Wait for user input when running a script file
    reindex       - reindex a collection
```

```
    sdk             - Execute SDK commands based on the C SDK library, type "sdk
                      help" for more information
    sleep           - Sleeps for the specified milliseconds
    timeout         - Globally change the timeout for waiting for a response from
                      a service
    tlogin          - Open a trusted SSL connection to an existing service
    topQuery        - Returns the top N longest running queries from the audit
                      log (either a file or from the log API)
    vslice          - Validate index slices

Remote commands (executed on the connected service, see "login"):
    login           - Connect to a remote service.  Once connected, type help to
                      see commands available for remote execution.

For detailed help, type "help <command>"
>
```

View Detailed Help on a Command

To view detailed information about a command, type `help <command>`. The following example shows help for the `logParse` command after typing `help logParse`.

```
For detailed help, type "help <command>"
> help logParse
Usage: logParse {in=<pathname>} {indir=<pathname>} [out=<pathname>]
                [content=<c2|c3>] [device=<device,[device...]>]
                [path=<log-parsers-config-path>] [metaonly] [srcaddr=<src
                address>] [srcaddrfile=<filename,IP Address>]
Parse line delimited logs on stdin and post results to stdout

    in          - The input source file. "in=stdin" means interactive typing of
                  log.
    indir       - The input source files parent directory
    out         - The output file or output file parent directory if input is
                  set by indir. If not specified, use stdout as output.
    content     - Content version, either c2 or c3. Default is c2.
    device      - Comma delimited device list specifying devices that is
                  enabled. Default enable all devices.
    path        - The logparsers configuration path. Default will find
                  configuration file like logdecoder.
    metaonly    - The output will only contains parsed meta, otherwise will
```

```
                print log message after metas.
    srcaddr     - The source address of the all the logs
    srcaddrfile - The source address for logs in one input file, in the format
                filename,ipaddress
>
```

View a List of Help Topics

To view a list of help topics, type `man`. The following example shows a list of help topics.

```
> man
List of topics:

    Introduction
    Connecting to a Service
    Monitoring Stats
    Feeds
    Converting Packet DB Files to PCAP
    Packets
    Verifying Database Hashes
    SDK Content
    SDK Content Examples
    Troubleshooting

Type "man <topic>" for help on a specific topic, partial matches are acceptable
>
```

View a Specific Help Topic

To view help about a specific topic, type `man <topic>`. The following example shows the Packets help topic after typing `man Packets`.

```
Type "man <topic>" for help on a specific topic, partial matches are acceptable
> man Packets
                                    Packets
                                    =======

The *packets* command can be used to generate a pcap or log file based on a
list of Session IDs, a time period or a where clause.  The command is quite
flexible and can be used on any running service that has access to the raw
data from a downstream component.  Before running the command, you must first
```

```
*login* to a service and then change directory to the appropriate sdk node,
(e.g., "cd /sdk").  Unlike the *makepcap* command, which only works on the
local file system, this command is meant to be used on a remote service.

    login ...
    cd /sdk
    packets where="service=80 && time='2018-03-01 15:00:00'-'2018-03-01
    15:10:00'" pathname="/tmp/march-1.pcap"
Write 10 minutes of HTTP only packets from March 1st, to the file
/tmp/march-1.pcap.  All times are in UTC.

    packets time1="2018-04-01 12:30:00" time2="2018-04-01 12:35:00"
    pathname=/media/sdd1/packets.pcap.gz
Write all packets between the two times to a gzip compressed file at
/media/sdd1/packets.pcap.gz

    packets time1="2018-04-01 12:30:00" time2="2018-04-01 12:35:00"
    pathname=/media/sdd1/mylogs.log
Write all logs between the two times to a plaintext file at
/media/sdd1/mylogs.log.  Any pathname ending with .log indicates that the
format of the output file should be plaintext line-delimited logs.
>
```

Caution: To get the latest information, view the command and help topics within NwConsole.

Quit NwConsole

To exit the NwConsole application, type `quit` at the command line.

Basic Command Line Parameters and Editing

NwConsole is like a Swiss army knife; it contains many tools buried underneath its command line interface. NwConsole is multi-platform. Executables are available for CentOS (which ships on appliances), Windows, and Mac. NwConsole is included on all hosts.

Basic Command Line Parameters

Here are some basic command line parameters:

- `-f` To run a set of commands from a file, use the `-f` attribute as shown here:
  `NwConsole -f /tmp/<somefile.script>`
- `-e` You can use the `-e` attribute (which is similar to the -f attribute) to run a set of commands from a file and allow environment variable substitution within the file using `$ENV_VAR` syntax, as shown here:
  `NwConsole -e /tmp/<somefile.script>`
  Use `\$` to escape a dollar sign and `\\` to escape a backslash.
- `-c` To pass in a list of commands from the command line, use the `-c` attribute as shown here:

  `NwConsole -c <command1> -c <command2> -c <command3>`

  This is not recommended except for very simple scripts. The Bash interpreter can jumble quoted strings if you do not escape properly. If you have non-obvious errors passing through the command line, switch to reading from a file to see if that fixes the issues.
- `-i` Normally, the NwConsole exits after running commands passed by a file or command line. If you want to keep the interactive prompt open after the commands are executed, include `-i` in the command line.
- `-q` To suppress command information messages and only see command output, use `-q` for Quiet mode. This makes it easy to pipe command output (`stdout`) to other commands.

  You can also run NwConsole and type the commands in the console window.

When you use the `-c` option, you can use additional quotes and escape quotes to include embedded spaces for the `where` clause. For example:

```
where="\"tcp.dstport=443 && time='2020-09-07 17:25:00'-'2020-09-08
05:24:59'\""
```

The following example shows how to use the `-c` option for pcap extraction.

```
[root@nwadmin1 ~]# NwConsole -c login 127.0.0.1:56003:ssl <user> <passowrd> -
c cd sdk -c packets
where="\"tcp.dstport=443 && time='2020-09-07 17:25:00'-'2020-09-08
05:24:59'\""
pathname="/var/netwitness/ny_sdwan_tcp_443_20200930.pcap"
RSA NetWitness NextGen Console 11.3.1.0
```

```
Copyright 2001-2020, RSA Security Inc. All Rights Reserved.

>login ...
Successfully logged in to 127.0.0.1:56003 as session 168886
>cd sdk
[127.0.0.1:56003] /sdk
>packets where="tcp.dstport=443 && time='2020-09-07 17:25:00'-'2020-09-08
05:24:59'"
pathname=/var/netwitness/ny_sdwan_tcp_443_20200930.pcap
Writing packets to /var/netwitness/ny_sdwan_tcp_443_20200930.pcap (100%)
```

Line Editing

You can use the keys in the following table when editing a command.

Key	Descripton
Ctrl-U	Clears the current line
Ctrl-W	Deletes the word that the cursor is on
Ctrl-A	Moves the cursor to the beginning of the line
Ctrl-E	Moves the cursor to the end of the line
Ctrl-F	Moves the cursor forward to the next word
Ctrl-B	Moves the cursor backwards to the previous word
Up arrow	Displays the previously executed command
Down arrow	Displays the command executed after the current command (only valid if the up arrow has been pressed)
Left arrow	Moves the cursor to the previous character
Right arrow	Moves the cursor to the next character
Tab	Provides context sensitive completion of most commands and their parameters. The Tab key is very helpful for editing. For example, to view the *Connecting to a Service* help topic, at the command line, you can type `man con` and then press the `Tab` key. NwConsole completes the command for you: `man Connecting to a Service` Press **Enter** to run the command and view the topic.
`history`	Displays a numbered list of previous commands
`history execute=#`	Executes a previous command, which is also equivalent to typing `!#` For example, `!1` executes the previous command.
`history clear`	Clears all command history
`history erase=#`	Erases a specific command from the history buffer. History is automatically stored from one session to the next.

Connecting to a Service

To connect and interact with a NetWitness Core service (Decoder, Concentrator, Broker, Archiver, and so on), you must first issue the `login` command. You must have an account on that service. You can type `help login` at any time for more information. Here is the syntax of the `login` command:

`login <hostname>:<port>[:ssl] <username> [password]`

For example: `login 10.10.1.15:56005:ssl someuser`

If you do not include the password, the NwConsole prompts you.

If you have set up proper trust between NwConsole and the endpoint, you can use the `tlogin` command and avoid having to enter a password. Setting up trust is beyond the scope of this documentation, but it involves adding NwConsole's SSL cert to the endpoint with the `send /sys peerCert op=add --file-data=<pathname of cert>` command. You must first use a normal login with the proper permissions before you can add a peer cert for subsequent trusted logins.

Once connected, you can interact with the endpoint service through a virtual file system. Instead of files, what you are looking at are the nodes of that service. Some nodes are folders and have child nodes, forming a hierarchical structure. Each node serves a purpose and all of them support a subset of commands like `info` and `help`. The `help` message returns information about the commands that each node supports. When you first log on, you are on the root node, which is the path `/`, just like a Linux or Mac system. To see a list of nodes under `/`, type the `ls` command.

All services have nodes like `sys` and `logs`. To interact with the **/logs** API, you can first send the `help` command to the **/logs** node. To do this, you must use the `send` message, which has this syntax:

```
Usage: send {node pathname} {message name} [name=value [name=value]]
            [--file-data=<pathname>] [--string-data=<text>] [--binary-data=<text>]
            [--file-format={binary,params,params-list,string,params-binary}]
            [--output-pathname=<pathname>] [--output-append-pathname=<pathname>]
            [--output-format={text,json,xml,html}]
Sends a command to a remote pathname.  For remote help, use "send <pathname>help" for
details.
    pathname                - The node pathname to retrieve information on
    message                 - The command (message) to send
    parameters              - Zero or more name=value parameters for the command
    --file-data             - Loads data from a file and send as either a BINARY
                              message or as a PARAMS_BINARY message if other
                              parameters exist. Providing the --file-format parameter
will
                              override automatic detection.
   --file format            - Treat file input as the provided format
  --string-data            - Sends text as a STRING message type
   --binary-data            - Send text as either a BINARY message type or as a
                              PARAMS_BINARY message type if other parameters
                              exist
    --output-pathname       - Writes the response output to the given pathname,
```

```
                              overwriting any existing file
  --output-append-pathname - Writes the response output to the given pathname,
                              will append output to an existing file
  --output-format          - Writes the response in one of the given formats,
                              the default is text
```

Parameters are formatted in `<name>=<value>` pairs, with whitespace between each pair. If a `<value>` has whitespace, you should place each value in double quotes. If the value itself should have double quotes, you can escape them by preceding with a backslash. For example: `send /sdk query query="select * where time=\"12-31-2019 14:30:00\"-u" size=100`

You must also escape backslashes; a double backslash is treated as a single backslash. It is also possible to pass special characters by giving their hex value using `\x##`. For example, `\x0a` would pass a line feed in the parameter value.

To send a `help` message, you would send this:

```
send /logs help
```

And your response would look something like this:

```
description: A container node for other node types
  security.roles: everyone,logs.manage
  message.list: The list of supported messages for this node
  ls:  [depth:<uint32>] [options:<string>] [exclude:<string>]
  mon:  [depth:<uint32>] [options:<uint32>]
  pull:  [id1:<uint64>] [id2:<uint64>] [count:<uint32>] [timeFormat:<string>]
  info:
  help:  [msg:<string>] [op:<string>] [format:<string>]
  count:
  stopMon:
  download:  [id1:<uint64>] [id2:<uint64>] [time1:<date-time>] [time2:<date-time>]
    op:<string> [logTypes:<string>] [match:<string>] [regex:<string>]
    [timeFormat:<string>] [batchSize:<uint32>]
  timeRoll:  [timeCalc:<string>] [minutes:<uint32>] [hours:<uint32>] [days:<uint32>]
    [date:<string>]
```

To get more information about a specific message or command, you can specify the `msg=<message name>` on the help command as a parameter. For example, look at the `pull` message help:

```
send /logs help msg=pull
```

```
 pull: Downloads N log entries
  security.roles: logs.manage
  parameters:
  id1 - <uint64, optional> The first log id number to retrieve, this is
     mutually exclusive with id2
  id2 - <uint64, optional> The last log id number that will be sent, defaults to
```

```
       most recent log message when id1 or id2 is not sent
    count - <uint32, optional, {range:1 to 10000}> The number of logs to pull
    timeFormat - <string, optional, {enum-one:posix|simple}> The time format used in
each
       log message, default is posix time (seconds since 1970)
```

The built-in message help says that this command grabs the last N log entries if you omit ID1 and ID2. To look at the last 10 log entries:

```
send /logs pull count=10 timeFormat=simple
```

Almost all of the commands on the service follow this simple format. The only commands that do not are the ones that require more complicated handshaking, like importing a PCAP to a Decoder. To import a PCAP, use the NwConsole `import` command, which takes care of the complicated communication channel handshaking.

Some parameters are specific to NwConsole's `send` command and are not actually sent to the service. You can use these parameters to change the output format of the response, write the response to a file, or read a file from the local machine and send it to the service. The local parameters to NwConsole's `send` command all start with two dashes --.

- `--output-format` — This parameter changes the normal output of the command from plain text to one of these types: JSON, XML, or HTML. The format `values` is a text output that only writes values (query output, stat/config values, and so on) without any other decorations.
- `--output-pathname` — Instead of writing the output to the terminal, the output is written to the specified pathname (truncates any existing file).
- `--output-append-pathname` — This is the same as `--output-pathname` except that it appends the output to an existing file (or creates the file if it does not exist).
- `--file-data` — Reads in a file and uses it as the command payload. This is useful for commands like `/sys fileEdit`. The following example shows how you can send an updated **index-concentrator-custom.xml** file using NwConsole:

  ```
  send /sys fileEdit op=put filename=index-concentrator-
  custom.xml --file-data="/Users/user/Documents/index-
  concentrator-custom.xml"
  ```

- `--file-format` — When reading an input file with `--file-data`, this parameter forces NwConsole to interpret the file as a specific type of input. The allowed enumerations are: `binary`, `params`, `params-list`, `string` and `params-binary`. As an example, to send a file of application rules (`*.nwr`) to a Decoder, you can use this command:

  ```
  send /decoder/config/rules/application replace --file-
  data=/path/rules.nwr --file-format=params-list
  ```

- `--string-data` — Sends the command payload as a string instead of a list of parameters.
- `--binary-data` — Sends the command payload as binary instead of a list of parameters.

Example Streaming Query to JSON file (could be a large result set):

```
send /sdk query size=0 query="select * where service=80 && time='2018-
03-05 13:00:00'-'2018-03-05 13:59:59'" --output-format=json --output-
pathname=/tmp/query.json
```

One thing to note about the `send` command is the fact that, by default, there is a timeout of 30 seconds waiting for a response. Some commands (like the query above) may take longer to receive results. To avoid a premature client-side timeout, you can use the `timeout [secs]` command to increase the wait. For instance, `timeout 600` would wait 10 minutes for a response before timing out. Once enacted, it takes effect for all subsequent commands.

To navigate around the virtual node hierarchy of the service, you can use the `cd` command like you would on any command shell. This covers the basics of connecting and interacting with a service. Once you are connected, the `help` command lists all the commands that you can use to interact with the endpoint. These commands do not display when you are not connected to an endpoint.

Monitoring Stats

You can use NwConsole to watch statistics (stats) change on a service in real time. However, be warned that this can result in a LOT of output. If you are not careful and monitor too many nodes, the screen scrolls by too quickly to be useful.

As a simple example, if you log on to a Decoder, you can monitor the capture rate in real time. To do this, issue these commands after connecting to a Decoder:

```
cd /decoder/stats
```

```
mon capture.rate
```

That is all you need to do! Now, any time the capture rate changes, it outputs into the console window.

You can add another monitor:

```
mon capture.avg.size
```

Now it watches those two stats and outputs those values when they change. You may have noticed that as you tried to type the second command, the output from the original monitor was messing up your display. This is the problem with monitoring stats. It is not really meant for doing more than just watching the stats after the first command is entered.

However, you can stop the monitoring by typing `delmons` and pressing **Enter**. Just ignore the output while you type and it returns you to a proper command prompt. If you want to monitor many stats at once, you can give the path of the parent stat folder and it monitors all of the stats underneath it. For example, typing `mon /decoder/stats` or `mon .` (they are equivalent) monitors everything. Be prepared for a lot of output! Remember to enter `delmons` if it is scrolling too fast.

Useful Commands

The following NwConsole commands are useful when interacting with NetWitness Core services:

- **feed**: Enables you to create and work with feed files.
- **makepcap**: Converts Packet database (DB) files to PCAP.
- **packets**: Retrieves packets or logs from the logged in service.
- **hash**: Creates or verifies hashes of database files.

The following sections as well as the NwConsole help and topic information (man) pages, provide additional information.

Feeds

The `feed` command provides several utilities for creating and examining feed files. A feed file contains the definition and data of a single feed in a format that has been precompiled for efficient loading by a Decoder or Log Decoder. For a complete reference on feed definitions, see the "Feed Definitions File" topic in the *Decoder Configuration Guide*. Go to the NetWitness All Versions Documents page and find NetWitness Platform guides to troubleshoot issues.

create

```
feed create <definitionfile> [-x <password>]
```

The `feed create` command generates feed files for each feed defined in a feed definition file. A definition file is an XML document that contains one or more definitions. Each feed definition specifies a data file and the structure of that data file. The resulting feed files will be created in the same directory as the definition file with the same name as the data file, but with the extension changed to **.feed** (for example, **datafile.csv** results in **datafile.feed**). Any existing files with the target name will be overwritten without a prompt.

```
$ ls
 example-definition.xml    example-data.csv
 $ NwConsole
 RSA NetWitness Console 11.3.0.0.0
 Copyright 2001-2020, RSA Security Inc.  All Rights Reserved.

 Type "help" for a list of commands or "man" for a list of manual pages.
 > feed create example-definition.xml
 Creating feed Example Feed...
 done.  2 entries, 0 invalid records
 All feeds complete.
 > quit
 $ ls
 example-definition.xml    example-data.feed    example-data.csv
 $
```

Optionally, feed files can be obfuscated using the option `-x` followed by a password of at least 16 characters (no spaces). This will be applied to all feeds defined in the definition file. In addition to the feed file, a token file will be generated for each feed file. The token file must be deployed with the corresponding feed file.

```
feed create example-definition.xml -x 0123456789abcdef
```

stats

```
feed stats <feedfile>
```

The `feed stats` command provides summary information for an existing, un-obfuscated feed file. Specifying an obfuscated feed file will result in an error.

```
> feed stats example.feed
 Example Feed stats:
 version  : 0
 keys count  : 1
 values count: 2
 record count: 2
 meta key  : ip.src/ip.dst
 language keys:
    alert    Text
```

dump

```
feed dump <feedfile> <outfile>
```

The `feed dump` command generates a normalized, key-value pair listing of an un-obfuscated feed file. You can use the resulting file to validate a feed file or assist in determining which records were considered invalid when the feed was created. Specifying an obfuscated feed file will result in an error. If `outfile` exists, the command will abort without overwriting the existing file.

```
feed dump example.feed example-dump.txt
```

Converting Packet DB Files to PCAP

You can use the `makepcap` command to quickly convert any Packet DB file to a generic PCAP file, preserving the capture time order. This command offers many options (see `help makepcap`), but is easy to use. All it really needs is the Packet DB directory (with the `source=<pathname>` parameter) to get started.

Note: You must stop the Decoder or Archiver service before running this command. If you want to generate a PCAP while the service is running, see the `packets` command.

- `makepcap source=/var/lib/netwitness/decoder/packetdb`
  This command converts every Packet DB file into a corresponding PCAP file in the same directory. If the disk is almost full, see the next command.

- `makepcap source=/var/lib/netwitness/decoder/packetdb dest=/media/usb/sde1`
  This command writes all of the output PCAPs to the directory at **/media/usb/sde1**.

- `makepcap source=/var/lib/netwitness/decoder/packetdb dest=/media/usb/sde1 filenum=4-6`
  This command only converts the files numbered 4 through 6 and skips all other files. In other words, it converts the Packet DB files: **packet-000000004.nwpdb**, **packet-000000005.nwpdb**, and **packet-000000006.nwpdb**.

- `makepcap source=/var/lib/netwitness/decoder/packetdb time1="2020-03-01 14:00:00" time2="2020-03-02 07:30:00" fileType=pcapng`
  This command only extracts packets with a timestamp between March 1st, 2020 at 2 PM and March 2nd, 2020 before or on 7:30 AM. It writes the file as `pcapng` in the same directory as the source. All timestamps are UTC.

Packets

You can use the `packets` command to generate a PCAP or log file based on a list of Session IDs, a time period, or a `where` clause. This command is very flexible, and you can use it on any running service that has access to the raw data from a downstream component. Before running the command, you must first `login` to a service and then change directory to the appropriate SDK node (for example, `cd /sdk`). Unlike the `makepcap` command, which only works on the local file system, you use this command for a remote service.

```
login ...
cd /sdk
packets where="service=80 && time='2020-03-01 15:00:00'-'2020-03-01 15:10:00'"
pathname="/tmp/march-1.pcap"
```

This command writes 10 minutes of HTTP-only packets from March 1st to the file **/tmp/march-1.pcap**. All times are in UTC.

- `packets time1="2020-04-01 12:30:00" time2="2020-04-01 12:35:00" pathname=/media/sdd1/packets.pcap.gz`
  This command writes all packets between the two times to a GZIP compressed file at **/media/sdd1/packets.pcap.gz**.

- `packets time1="2020-04-01 12:30:00" time2="2020-04-01 12:35:00" pathname=/media/sdd1/mylogs.log`
  This command writes all logs between the two times to a plaintext file at **/media/sdd1/mylogs.log**. Any pathname ending with **.log** indicates that the format of the output file should be plaintext line-delimited logs.

Verifying Database Hashes

By default, Archiver writes an XML file for every DB file that is written. This XML file ends with the extension **.hash** and contains a hash of the file along with other pertinent information. You can use the `hash` command to verify that the DB file has not been tampered with by reading the hash stored in the XML file and then rehashing the DB file to verify that the hash is valid. The command does accept wildcards, so something like `/var/netwitness/archiver/database0/default/packetdb/*.hash` will verify all hashes in that directory. You can also use the `--output-pathname` and `--output-format` parameters to write out the verification in json, xml or text formats for scripting purposes. If the DB files are not found in the location that the XML file says it should be in, then the command will attempt to find the DB files in the current directory or the directory where the hash file is found.

```
hash op=verify
hashfile=/var/lib/netwitness/archiver/database0/alldata/packetdb/packe
t-000004880.nwpdb.hash
```

This command verifies that the Packet DB file **packet-000004880.nwpdb** still matches the hash in the XML file **packet-000004880.nwpdb.hash**. For proper security, the hash file should be stored somewhere else to prevent the XML file from being tampered with (such as write-once only media), but the hash command itself is not affected by where it is stored.

SDK Content Command

One of the powerful commands in NwConsole is `sdk content`. It contains numerous options to do just about anything, at least as far as extracting content from the NetWitness Core stack. You can use it to create PCAP files, log files, or extract files out of network sessions (for example, to grab all of the pictures from email sessions). It can append files, have a maximum size assigned before creating a new file, and automatically clean up files when the directory grows too large. It can run queries in the background to find new sessions. It breaks queries into manageable groups and performs those operations automatically. When the group is exhausted, it does a requery to get a new set of data for further operations. The list of options for the SDK content command is very extensive.

Because the command has so many options, this document provides examples of commands for different use cases.

Before you can run `sdk content`, there are a few commands (like logging into a service) that you need to run first. Here are some examples:

- First connect to a service:
  `sdk open nw://admin:netwitness@10.10.25.50:50005`
- If you need to connect over SSL, use the nws protocol:
  `sdk open nws://admin:netwitness@10.10.25.50:56005`
- Keep in mind that you are passing a URL and must URL encode it properly. If the password is `p@ssword`, the URL looks like this:
  `sdk open nw://admin:p%40ssword@10.10.25.50:50005`
  This also applies to username.
- Once you log in, you can set an output directory for the commands: `sdk output <some pathname>`
- For command line help, type: `sdk content`

Before you try the example commands, it is important to understand the `sessions` parameter. This parameter is very important and controls how much or how little data you want to grab (the `where` clause is also important). The sessions parameter is either a single session ID or a range of session IDs. All NetWitness Core services work with session IDs, which start at 1 and increase by 1 for every new session added to the service (network or log session). Session IDs are 64-bit integers, so they can get quite large.

For example, to keep it simple, assume we have a Log Decoder that has ingested and parsed 1000 logs. On the service, you now have 1000 sessions with session IDs from 1 to 1000 (session ID 0 is never valid). If you want to operate over all 1000 sessions, you pass `sessions=1-1000`. If you only want to operate over the last 100 sessions, you pass `sessions=901-1000`. Once the command finishes processing session 1000, it exits back to the console prompt.

Many times, however, we do not care about specific session ranges. We just want to run a query over all of them and process the sessions that match a query. Here are some shortcuts that simplify this:

- The letter `l` (lowercase L) means lower bound or the lowest session ID.
- The letter `u` means the highest session ID. In fact, it actually means the highest session ID for future sessions as well. In other words, if you pass `sessions=l-u`, this special range means operate over all the current sessions in the system, but also do not quit processing, and as new sessions enter the system, process those, too. The command pauses and waits for new sessions once it reaches the last session on the service. To summarize, the command never exits and goes into continuous processing mode. It runs for days, months, or years, unless it is killed.
- If you do not want the command to run forever, you can pass `now` for the upper limit. This determines the last session ID on the service at the time the command starts and processes all sessions until it reaches that session ID. Once it reaches that session ID, the command exits, regardless of how many sessions may have been added to the service since the command started. So, for the example Log Decoder, `sessions=200-now` starts processing at session 200 and goes all the way to session 1000 and quits. Even if another 1000 logs were added to the Log Decoder after the command started, it still exits after processing session 1000.
- The parameter `sessions=now-u` means start at the very last session and continue processing all new sessions that come in. It does not process any existing sessions (except the last one), only new sessions.
- If you want to run this continually and have it remember the last session it processed between invocations, use the `sessionPersist={pathname}` parameter. The pathname should be a valid filename where the command will write the last successful session id. Upon restart, it will read the last session id and pick up where it left off.

For example commands and what they do, type `man sdk content examples` or see SDK Content Command Examples.

SDK Content Command Examples

The first NwConsole SDK content command example below is simple and shows all of the commands that you need to enter. After that, the examples show only the `sdk content` commands. The first example creates a log file and grabs the first 1000 logs out of a Concentrator aggregating from a Log Decoder:

- ```
  sdk open nw://admin:netwitness@myconcentrator.local:50005
  sdk output /tmp
  sdk content sessions=1-1000 render=logs append=mylogs.log
  fileExt=.log
  ```
  This script outputs 1000 logs (assuming sessions 1 through 1000 exist on the service) to the file **/tmp/mylogs.log**. The logs are in a plain text format. The parameter `fileExt=.log` is necessary to indicate to the command that we want to output a log file.

- ```
  sdk content sessions=1-1000 render=logs append=mylogs.log
  fileExt=.log includeHeaders=true separator=","
  ```
  This command grabs the same 1000 logs as above, but it parses the log header and extracts the log timestamp, forwarder, and other information, and puts them in a CSV formatted file.
  Example CSV: `1422401778,10.250.142.64,10.25.50.66,hop04b-LC1,%MSISA-4:81.136.243.248...`
  The timestamp is in Epoch time. The `includeHeaders` and `separator` parameters can only be used on NetWitness installs 10.4.0.2 and later.

- ```
  sdk content sessions=1-now render=logs append=mylogs.log
  fileExt=.log includeHeaders=true separator=","
  where="risk.info='nw35120'"
  ```
  This command writes a log file across the current session range, but only logs that match `risk.info='nw35120'`. Keep in mind that when you add a `where` clause, it performs a query in the background to gather the session IDs for export. The query should be run on a service with the proper fields indexed (which is typically a Broker or Concentrator). In this case, since you are querying the field `risk.info`, double-check the service where you run the command to make sure it is indexed at the value level (`IndexValues`, see **index-concentrator.xml** for examples). By default, most Decoders only have time indexed. If you use any field but time in the `where` clause, you need to move the query from the Decoder to a Concentrator, Broker, or Archiver with the proper index levels for the query. You can find more information on indexing and writing queries in the *NetWitness Core Database Tuning Guide*.

- ```
  sdk content sessions=1-now render=logs append=mylogs.log
  fileExt=.log includeHeaders=true separator=","
  where="threat.category exists && time='2020-01-05 15:00:00'-'2020-01-05 16:00:00'"
  ```
  This command is the same as above, but it only searches for matching logs between 3 PM and 4 PM (UTC) on Jan 5, 2020 that have a meta key `threat.category`. Again, because this query has a field other than time in the `where` clause (`threat.category`), it should be run on a service with

`threat.category` indexed at least at the `IndexKeys` level (the operators `exists` and `!exists` only require an index at the key level, although values work fine, too).

- `sdk content sessions=1-now render=logs append=mylogs fileExt=.log where="event.source begins 'microsoft'" maxFileSize=1gb`
  This command creates multiple log files, each one no larger than 1 GB in size. It prepends the filenames with **mylogs** and appends them with the date-time of the first packet/log timestamp in the file. Some example filenames: **mylogs-1-2020-Jan-28T11_08_14.log**, **mylogs-2-2020-Jan-28T11_40_08.log** and **mylogs-3-2020-Jan-28T12_05_47.log**. On versions older than 10.5, the T separator between date and time is a space.

- `sdk content sessions=1-now render=pcap append=mypackets where="service=80,21 && time='2020-01-28 10:00:00'-'2020-01-28 15:00:00'" splitMinutes=5 fileExt=.pcap`
  This command grabs all the packets between the five-hour time period for service types 80 and 21 and writes a PCAP file. Every five minutes, it starts a new PCAP file.

- `sdk content time1="2020-01-28 14:00:00" time2="2020-01-28 14:15:00" render=pcap append=mydecoder fileExt=.pcap maxFileSize=512mb sessions=1-now`
  Pay attention to this command. Why? It works for both packets and logs and is *extremely fast*. The downside is that you get everything between the two time ranges and you cannot use a `where` clause. Again, it starts streaming everything back almost immediately and does not require a query to run first on the backend. Because everything is read using sequential I/O, it can completely saturate the network link between the server and client. It starts creating files prepended with **mydecoder** and splits to a new file once it reaches 512 MBs in size.

- `sdk tailLogs`
  or (the equivalent command):
  `sdk content render=pcap console=true sessions=now-u`
  This is a fun little command. It actually uses `sdk content` behind the scenes. The purpose of this command is to view all incoming logs on a Log Decoder. As logs come into the Log Decoder (you can also run it on a Broker or Concentrator), they are output on the console screen. It is a great way to see if the Log Decoder is capturing and what exactly is coming into the Log Decoder. This command runs in continuous mode. Do not use it if the Log Decoder is capturing at a high ingest rate (this command cannot keep up with it). However, it is helpful for verification or troubleshooting purposes.

- `sdk tailLogs where="device.id='ciscoasa'" pathname=/mydir/anotherdir/mylogs`
  This command is the same as above, except it only outputs logs that match the `where` clause and instead of outputting to the console, it writes them to a set of log files under `/mydir/anotherdir` that do not grow larger than 1 GB. Obviously, you can accomplish this with the `sdk content` command as well, but this command requires a little less typing if you like the default behavior.

- `sdk content sessions=now-u render=pcap where="service=80" append=web-traffic fileExt=.pcap maxFileSize=2gb maxDirSize=100gb`
  This command starts writing PCAPs of all web traffic from the most recent session and all new incoming sessions that match `service=80`. It writes out PCAPs no larger than 2 GBs and if all the PCAPs in the directory grow larger than 100 GBs, it deletes the oldest PCAPs until the directory is 10% smaller than the maximum size. Keep in mind that the directory size checking is not exact and it only checks every 15 minutes by default. You can adjust the number of minutes between checks by passing `cacheMinutes` as a parameter, but this only works with versions 10.5 and later.

- `sdk content sessions=79000-79999 render=nwd append=content-%1%.nwd metaFormatFilename=did`
  This is a basic backup command. It grabs 1000 sessions and outputs the full content (sessions, meta, packets, or logs) to the NWD (NetWitness Data Format) format. NWD is a special format that can be re-imported to a Packet or Log Decoder without reparsing. So essentially, the original parsed session imports without changes. The timestamp does not change as well, so if it was originally parsed six months ago, the timestamp upon import will be retained as six months ago.

  Note: Do not expect great performance with this command, especially with packets. Gathering the packets for a session involves a lot of random I/O and can drastically slow down the export. Logs do not suffer as much from this problem (only one log per session), but behind the scenes, this command uses the /sdk content API, which is not a performance-minded streaming API like /sdk packets.

  The `metaFormatFilename` parameter is very helpful in this command. If this command is run on a Concentrator with more than one service, the NWD filenames will be created with the `did` meta for each session (the `%1%` in the append parameter is substituted with the value of `did`). Each filename will indicate exactly which Decoder the data came from.

- `sdk content session=1-u where="service=80,139,25,110" render=files cacheMinutes=10 dir=/tmp/content-files maxDirSize=200mb`
  This is another fun little command. It works very similar to our old Visualize product if you pair the output directory with something like Windows Explorer in Icon mode. It extracts files from all web, email, and SMB traffic. This includes all kinds of files, such as images, zip files, videos, PDFs, office documents, text files, executables, and audio files. If it extracts malware, your virus scanner will flag it. Nothing will be executed by the command, so it does not infect the machine (unless you try to execute it yourself). However, it can be useful because if you do find malware, the filename indicates the session ID where it was extracted. You can then query that session ID and see which host the malware possibly infected and take action. You can filter what gets extracted with the parameters `includeFileTypes` or `excludeFileTypes` (see the command help). For example, adding `excludeFileTypes=".exe;.dmg;.msi"` prevents executables and installers from being extracted. This command just runs nonstop extracting files from all existing and any new sessions. After the directory gets littered with more than 200 MBs of files, it automatically starts cleaning up the files every 10 minutes.

  Note: This command only makes sense for packet sessions, not logs.

- `sdk content session=1-now render=files where="time='2015-01-27 12:00:00'-'2015-01-27 13:00:00' && (service=25,110,80)" subdirFileTypes="audio=.wav;.mp3;.aac; video=.wmv;.flv;.mp4;.mpg;.swf; documents=.doc;.xls;.pdf;.txt;.htm;.html images=.png;.gif;.jpg;.jpeg;.bmp;.tif;.tiff archive=.zip;.rar; other=*" renameFileTypes=".download|.octet-stream|.program|.exe;.jpeg|.jpg" maxDirSize=500mb`
  This command extracts files from HTTP and email sessions from a one-hour period and then groups the extracted files into directories specified by the `subdirFileTypes` parameter. For example, any extracted audio file with the extension `.wav`, `.mp3`, or `.aac` will be placed into the subdirectory `audio`, which will be created under the specified output directory. The same goes for all the other groups specified in that parameter. Some files will also be automatically renamed based on their file extension, which is handled by `renameFileTypes`. Any file with an extension `.download`, `.octet-stream` or `.program` will be renamed to `.exe`. Files with the extension `.jpeg` will be renamed `.jpg`. Once the top-level directory exceeds 500 MBs, the oldest files are cleaned. This command stops at the last session at the time the command started.

- `sdk content session=1-u render=111 where="filename exists" maxDirSize=20mb fileHash=md5,sha1,sha256 linkMeta=sessionid,time,did,service_uuid deviceType=filehash logDecoder="<hostname>:<port>[:ssl]" sessionPersist=/tmp/last-session.txt`
  If you want to extract file hashes from network protocols like HTTP, SMB, POP3 and SMTP, this is the command you want to use. Not only will it give you the file hash, but it will send an event to a Log Decoder with this information, which will be processed into a session for querying. For example, use `sdk open` to talk to a Broker or Concentrator that is aggregating from one or more Network Decoders. Set the `where` clause to find sessions of interest that have embedded files in the protocols we support for file extraction (use Live to install Malware Analysis content for example). Next, set up a Log Decoder to receive the file hash information. This information will go into the meta keys `checksum`, `checksum.algo`, `filename`, `filename.size` and `link`. The meta key link is used to indicate the session (with the file that has been hashed) on the device used by `sdk open`. When hashes of interest are discovered on the Log Decoder, the link meta can be used to find the original packet session that contains the file. The parameters `fileHash`, `linkMeta` and `deviceType` can be used to customize how the log is generated that is sent to the Log Decoder. The ports for sending Syslog to a Log Decoder are usually 514 or 6514 (for SSL). You may need to add a `table-map-custom.xml` file on the Log Decoder to properly save the meta keys sent in the file hash log. For the `linkMeta` parameter, `service_uuid` is a special meta key that inserts the UUID of the service the session ID belongs to. This can be used to link back to the original session from the Log Decoder. `sessionPersist` is used to write out our session IDs to pick up where we left off from the last run.

- `sdk search session=1-now where="service=80,25,110" search="keyword='party' sp ci"`
  This command searches all packets and logs (the `sp` parameter) for the keyword `party`. If party is

found anywhere in the packets or logs, it outputs the session ID along with the text it found and the surrounding text for context. The `where` clause indicates that it only searches web and email traffic. The `ci` parameter means that it is a case-insensitive search. You can substitute `regex` for `keyword` and it performs a regex search.

- `sdk search session=1-now search="keyword='checkpoint' sp ci" render=log append=checkpoint-logs.log fileExt=.log`
  This is an interesting command example. It searches all logs (or it could be packets) for the keyword `checkpoint` and if that keyword is seen, it extracts the log to a file **checkpoint-logs.log**. There are all kinds of possibilities with this command. Essentially, when a hit is detected, it hands off the session to the content call. So any parameters you pass to `sdk search` that it does not recognize are just passed along to the content call. This allows the full capabilities of the `sdk content` call, but they only work on those sessions with content search hits.

Commands Used for Troubleshooting

NwConsole provides the following commands that are helpful when troubleshooting NetWitness:

- **whatIsWrong**: Provides a snapshot of a service's configuration, stats, and failure and warning logs for a specified period of time in the past.
- **dbcheck**: Performs consistency checking of database files.
- **topQuery**: Helps pinpoint queries that are taking an excessively long time to run.
- **netbytes**: Troubleshoots the network connections on the current host.
- **netspeed**: Troubleshoots the connection between the host computer running NwConsole and the remote computer connected to it using the `login` command.

The following sections, as well as the NwConsole help and topic information (man) pages, provide additional information.

whatIsWrong

When a service is not working correctly, the reason is usually somewhere in the logs that the service has written. You can use the `whatIsWrong` console command to obtain a snapshot of a service's configuration, stats, and failure and warning logs (with surrounding context logs) for a specified past period of time, which defaults to the previous seven days. You can save the results of running `whatIsWrong` into a specified plain-text file. The output of this command can be a useful starting point to help determine what is currently wrong with a service.

To use the `whatIsWrong` console command, log on to the service to troubleshoot using the `login` command, and run the `whatIsWrong` command.

Note: Use `help whatIsWrong` to see all of the available parameters, including the number of days or hours to look back for events, the pathname to store results, whether or not to append or overwrite the results file, and the delimiter to use for log fields. You can also limit the number of most recent logs used to find context, and you can specify how many context logs per warning or failure log to retrieve.

Whenever you receive a request for logs for a Core service, you should run the `whatIsWrong` command first and use the results collected as a starting point.

dbcheck

The `dbcheck` command is used to perform consistency checking of database files (session, meta, packets, logs, stats, and so on). This might be necessary when a service cannot start because of errors in the consistency of the database files. Normally a service would automatically recover and correct any consistency issues on startup, but there are times when this does not occur. When a service starts (like Decoder), it typically does not read or open most database files in order to start quickly. It assumes most files are in a consistent state and only does a cursory check of the most recently written files. If there are problems, `dbcheck` can perform those consistency checks, but ONLY if the service is not running.

Caution: Do not attempt to run this command while a service is running.

For example, you can check a single file:

```
dbcheck /var/lib/netwitness/decoder/packetdb/packet-000000001.nwpdb
```

You can also use wildcards to check multiple files:

```
dbcheck /var/lib/netwitness/decoder/metadb/meta-00000002*.nwmdb
```

topQuery

The `topQuery` command can help pinpoint queries that are taking an excessively long time to run. This command parses the audit logs of a service and returns the top N longest running queries for the specified time period.

The easiest way to run it is to log on to the service (usually a Broker or Concentrator) and type `topQuery`. The default behavior is to return the top 100 longest running queries for the last seven days.

Type `help topQuery` for the list of parameters. Here are some additional examples with explanations:

- `topQuery hours=12 top=10`
  This command returns the top 10 queries for the last 12 hours.

- `topQuery time1="2020-03-01 00:00:00" time2="2020-03-14 00:00:00"`
  This command returns the top 100 queries between March 1, 2020 and March 14, 2020. Times are in UTC, not local.

- `topQuery input=/var/log/messages output=/tmp/top20.txt top=20 user=sauser1`
  Instead of connecting to a service, it parses the Syslog audit messages for the top 20 queries in the last seven days, but only for queries executed by user sauser1. It writes the top 20 queries to **/tmp/top20.txt** instead of the console screen. The parameter user is a regex, so you can specify multiple usernames by writing something like `user="(sauser1|sauser2)"`.

netbytes

The `netbytes` command is very useful for troubleshooting the network connections on the current host. It displays continuous send and receive statistics for all network interfaces. Once executed, you must press **Ctrl-C** to exit this command, which also exits NwConsole.

netspeed

The `netspeed` command is used to troubleshoot the connection between the host computer running NwConsole and the remote computer connected to it through the `login` command. You must supply the amount of bytes to transfer and it will time the speed of the connection. The netspeed command is very useful for troubleshooting aggregation performance issues that might be network related.

```
login somedecoder:50004 admin ...
netspeed transfer=4g
```

To troubleshoot the connection between a Concentrator and a Decoder, SSH into the Concentrator, run NwConsole, and then log on to the Decoder and run `netspeed`. The output from the command gives you an indication of the maximum network throughput. If it is much less than the standard 1 Gbps interface, it could indicate a network issue.