NAME

nafalize − NetSA Aggregated Flow aggregator and normalizer

SYNOPSIS

```
nafalize    [--in INPUT_SPECIFIER] [--out OUTPUT_SPECIFIER]
            [--nextdir PROCESSED_INPUT_DIRECTORY]
            [--faildir FAILED_INPUT_DIRECTORY]
            [--poll POLLING_DELAY] [--lock]
            [--log LOG_SPECIFIER] [--loglevel LOG_LEVEL]
            [--verbose] [--version] [--daemon] [--foreground]
            [--intype INPUT_TYPE] [--horizon FLOW_HORIZON]
            [--live] [--bpf BPF_EXPRESSION]
            [--rotate-delay ROTATE_DELAY] [--sid SOURCE_ID]
            AGGREGATION_EXPRESSION
```

DESCRIPTION

nafalize is the NAF (NetSA Aggregated Flow) tool suite flow aggregator and normalizer. It provides a flexible aggregation facility for producing time-series aggregates of network flow and packet data in a variety of formats. It produces output in the NAF aggregated flow format, which is based on IPFIX and is processed by the other NAF tools.

The aggregation operation performed by nafalize is defined by an *aggregation expression*, which specifies the time bin size to use for splitting the flow data into time series, the fields to be present in the aggregate flow key, the counts to maintain per aggregate flow key, optional filters for restricting the input flows, and an optional sort order for the output. See **AGGREGATION EXPRESSION SYNTAX** below for details on the aggregation expression.

nafalize takes its input from files containing flow or packet data, from packets captured from a network interface, or from IPFIX exporting processes via TCP or UDP (SCTP support is planned for a future release). The **−−in**, **−−intype**, and **−−live** options determine how nafalize will get its input.

nafalize presently writes its output to files or to standard output; the output file or directory is specified using the **−−out** option. nafalize's output is in the NAF aggregated flow format, which is an IPFIX message stream containing aggregated flow data. To convert the NAF format to whitespace-delimited text for processing by other tools or for human consumption, use the *nafscii*(1) tool included with the NAF distribution.

nafalize, like all NAF tools, operates by default in **once** mode, though it can also be run as a **daemon**. In daemon mode, nafalize will wait for new input to match its input specifier, and move processed input to the **−−nextdir** directory. This can be used to build "chains" of daemons for automated batch processing of flow data.

OPTIONS

Input Options

The input specifier and input type determine where nafalize will read its input from. If reading from a file (if the input type is not **ipfix-tcp** or **ipfix-udp**, and **−−live** is not present) and the input specifier is not given, nafalize defaults to reading from standard input.

−−intype *INPUT_TYPE*

INPUT_TYPE is an input type, defaulting to *pcap*. The following input types are supported:

pcap

Read packets from libpcap (i.e., *tcpdump*(1)) binary dumpfiles or from a pcap interface. See **PCAP Notes** below for special information concerning packet input.

ipfix

Read flows from IPFIX-formatted files, such as those produced by NetSA's **YAF** flow sensor. An IPFIX-formatted file is simply a serialized IPFIX message stream, so you can also read IPFIX flow data using a general utility such as *nc*(1) and process it with nafalize. All the IPFIX input types support IPFIX bidirectional flow collection using a provisional implementation of the mechanism described in IETF draft−trammell−ipfix−biflow−01.

ipfix-udp
Operate as an IPFIX Collecting Process, reading flows from a passive UDP socket recieved from an IPFIX Exporting Process.

ipfix-tcp
Operate as an IPFIX Collecting Process, reading flows from a passive TCP socket recieved from an IPFIX Exporting Process. nafalize presently only supports reading from a single Exporting Process over TCP at a time.

silk
Read flows from a CERT/NetSA SiLK tools binary flow file. The most common usage pattern for this input type is to pipe *rwfilter* (1) output directly into nafalize.

argus
Read flows from a QoSient Argus 2.0.6 binary flow file, as produced by argus, ra, or ragator.

naf Read pseudoflows from a NAF file, for reaggregating NAF data. Reaggregation is by nature a limited function. Unique counts are lost across reaggregation operations, and if flow data is reaggregated with bin sizes that are not integer multiples of each other, time accuracy may be lost.

−−in *INPUT_SPECIFIER*
INPUT_SPECIFIER is an input specifier. If reading from a file, this is a filename, a directory name, a file glob pattern (in which case it should be escaped or quoted to prevent the shell from expanding the glob pattern), or the string **−** to read from standard input.

If *INPUT_TYPE* is **ipfix-tcp** or **ipfix-udp**, *INPUT_SPECIFIER* is a passive socket specifier of the format [*address*,]*port*, where *address* can be * to bind to all active interfaces; if not given, this defaults to port 4739 on all interfaces. *address* may be an IPv4 address, an IPv6 address, or a hostname that will resolve to an IPv4 or IPv6 address. Note that IPv6 is only available on machines with dual-stack support.

If *INPUT_TYPE* is **pcap** and **−−live** is given, *INPUT_SPECIFIER* is a pcap interface name, and is required.

−−live
If present, capture packets from an interface using libpcap. Implies **−−intype pcap**. Requires sufficient privileges for raw packet capture; at this time, nafalize is not designed to run setuid or use privilege separation. If present, **−−in** is required, and *INPUT_SPECIFIER* must be a pcap interface name.

−−bpf *BPF_EXPRESSION*
If *INPUT_TYPE* is **pcap**, apply the filter in *BPF_EXPRESSION* to packets as they are read. *BPF_EXPRESSION* must be quoted so the shell will pass it to nafalize as a single argument. See the *tcpdump* (1) manpage for information on BPF expressions.

Output Options

The output specifier determines where NAF will write its output. The output specifier is optional. If reading standard input, output defaults to standard output. If reading from files on disk, output defaults to one file per input file, named as the input file in the same directory as the input file with a **.naf** extension. If reading from IPFIX TCP or UDP sockets, output defaults to files named with the port number and a timestamp in the current directory; likewise, if reading from a live pcap interface, output defaults to files named with the interface name and a timestamp in the current directory.

−−out *OUTPUT_SPECIFIER*
OUTPUT_SPECIFIER is an output specifier. If present, this should be a filename or a directory name, or the string **−** to write to standard output.

−−rotate−delay *ROTATE_DELAY*
When *INPUT_TYPE* is **ipfix-udp**, **ipfix-tcp**, or when **−−live** is present, *ROTATE_DELAY* is the rotation delay in seconds between the creation of each subsequent output file. The default delay is 300 seconds

(5 minutes).

Daemon Options

These options are used to run nafalize in daemon mode for batch processing of packet and flow files.

−−daemon
Run nafalize in daemon mode. Instead of processing its input then exiting, nafalize will continually look for new input matching its input specifier. This will cause nafalize to fork into the background and exit.

−−foreground
Instead of forking in **−−daemon** mode, stay in the foreground. Useful for debugging.

−−lock
Use lockfiles for concurrent file access protection. Highly recommended in **−−daemon** mode, especially if two NAF daemons are interacting through a given directory.

−−poll *POLLING_DELAY*
POLLING_DELAY is the polling delay in seconds; how long nafalize will wait for new input when none is available. The default is 60 seconds.

−−nextdir *PROCESSED_INPUT_DIRECTORY*
When reading from files, if this option is present, input files will be moved to *PROCESSED_INPUT_DIRECTORY* after they are successfully processed. The special string **delete** will cause successfully processed input to be removed instead. This option is required in daemon mode.

−−faildir *FAILED_INPUT_DIRECTORY*
When reading from files, if this option is present, input files will be moved to *FAILED_INPUT_DIRECTORY* if processing failed. The special string **delete** will cause failed input to be removed instead. This option is required in daemon mode.

Logging Options

These options are used to specify how log messages are routed. nafalize can log to standard error, regular files, or the UNIX syslog facility.

−−log *LOG_SPECIFIER*
Specifies destination for log messages. *LOG_SPECIFIER* can be a *syslog*(3) facility name, the special value **stderr** for standard error, or the *absolute* path to a file for file logging. Standard error logging is only available in **−−daemon** mode if **−−foreground** is present. The default log specifier is **stderr** if available, **user** otherwise.

−−loglevel *LOG_LEVEL*
Specify minimum level for logged messages. In increasing levels of verbosity, the supported log levels are **quiet**, **error**, **critical**, **warning**, **message**, **info**, and **debug**. The default logging level is **warning**.

−−verbose
Equivalent to **−−loglevel debug**.

−−version
If present, print version and copyright information to standard error and exit.

Aggregation Options

−−horizon *FLOW_HORIZON*
The *FLOW_HORIZON* is the window of time, in seconds, for which nafalize will keep flow data during aggregation to buffer out-of-order flows in the input. This should set as least as large as the active flow timeout of the flow sensor from which the flow records were generated. It defaults to 3600 seconds (1 hour) and is automatically adjusted to be an integer multiple of the bin size in the *AGGREGATION_EXPRESSION*.

−−sid *SOURCE_ID*

Force the source ID field on input records to the given 32−bit integer.

AGGREGATION EXPRESSION SYNTAX

The aggregation expression controls the aggregation operations nafalize will perform, and generally follows the order of nafalize's processing stages, as below:

bin [**uniform** | **start** | **end**] *bin-length* (**sec** | **min** | **hr**)

[**uniflow**] [**perimeter** *address-rangelist*]

[**filter** *filter-expression*]

aggregate *aggregation-phrase* [**aggregate** *aggregation-phrase*]...

The **bin** keyword is required, and defines the size of the time bins each flow will be placed into. Either **uniform**, **begin**, or **end** may be used to determine how flows spanning bin boundaries will be split or placed into bins: **uniform** splits the flows uniformly into each bin covered by the flow, **begin** places the flow completely into the bin containing its start time, and **end** places the flow completely into the bin containing its end time. Bin times may be specified in seconds, minutes, or hours.

If present, the **uniflow** keyword suppresses matching of unidirectional flows into bidirectional flows.

If present, the **perimeter** keyword activates perimeter mode. In perimeter mode, all flows that do not cross the perimeter specified by the *address-rangelist* are dropped, and bidirectional flows are selectively reversed such that the source address is outside the perimeter and the destination address is inside. See **Rangelist Syntax** for details on the rangelist syntax used for defining perimeters.

If present before any aggregation phrase, the **filter** keyword specifies a prefilter to be applied to all flows as they are read. See **Filter Expression Syntax** for details on filter expressions.

See **Aggregation Phrase Syntax** below for details on the aggregation phrases. The presence of multiple *aggregation-phrases* in the aggregation expression will cause nafalize to fan out to multiple files, each with its own aggregation; each of these phrases should have a **label** to differentiate output files.

Aggregation Phrase Syntax

Each aggregation phrase specifies the flow key fields to aggregate on, and the counts to maintain for each aggregated flow. Each aggregation can optionally be separately filtered and sorted, as well. Its format is shown below:

aggregate [**sip**[/*mask*]] [**dip**[/*mask*]] [**sp**] [**dp**] [**proto**] [**srcid**]

count [**hosts**] [**ports**] [**flows**] [**packets**] [**octets**]

[**filter** *filter-expression*] [**sort** *sort-expression*]

[**label** *output-label*]

If present, the words in the aggregation phrase have the following effects:

sip/*mask*

Include source IP address in the output flow key, masked to the given prefix length.

sip Include source IP address in the output flow key; equivalent to **sip/32**.

dip/*mask*

Include destination IP address in the output flow key, masked to the given prefix length.

dip Include destination IP address in the output flow key; equivalent to **dip/32**.

sp Include source transport port in the output flow key for TCP and UDP flows. The source transport port is 0 for other flows.

dp Include destination transport port in the output flow key for TCP and UDP flows, and ICMP type/code data in the destination port field for ICMP flows. The destination transport port is 0 for other flows.

proto

Include IP protocol identifier in the output flow key.

srcid

Include source identifier in the output flow key. The source identifier comes from the IPFIX observationDomainId information element for IPFIX input, or can be set using the **−−sid** option.

hosts

Count unique source and destination addresses appearing in each aggregated flow. Source host count is suppressed if the source IP address mask is 32, and destination host count is suppressed if the destination IP address mask is 32.

ports

Count unique source and destination transport ports appearing in each aggregated flow. Source port count is suppressed if the source port is included in the flow key, and destination port count is suppressed if the destination port is included in the flow key.

flows

Count forward and reverse uniflows for each aggregated flow record. A ‘‘forward’’ uniflow is a uniflow whose source and destination match the aggregated flow’s source and destination; a ‘‘reverse’’ uniflow is one whose source and destination are reversed from the aggregated flow’s source and destination. nafalize’s biflow matching algorithm uses the first uniflow seen in the input as the ‘‘forward’’ direction for the aggregated flow.

In general, the number of unanswered connection attempts can be approximated by subtracting the reverse flow count from the forward flow count.

In **uniflow** mode, only forward flows are counted, because nafalize does no flow matching. When reading packet data, only TCP flows are counted, and TCP flow counts are approximated by counting packets with the SYN flag set.

packets

Count forward and reverse packets for each aggregated flow record. Forward packets are packets in forward uniflows, and reverse packets are packets in reverse uniflows, as defined above.

In **uniflow** mode, only forward packets are counted, because nafalize does no flow matching.

octets

Count forward and reverse octets for each aggregated flow record. Forward octets are octets in forward uniflows, and reverse octets are octets in reverse uniflows, as defined above. Octet counts from nafalize are IP−only; they do not include Layer 2 header lengths.

In **uniflow** mode, only forward octets are counted, because nafalize does no flow matching.

filter *filter-expression*

Filter all flows before aggregation using the given filter expression. See **Filter Expression Syntax** for details on filter expressions. By default, no filtering is performed.

sort *sort-expression*

Sort aggregate flows using the given sort expression. See **Sort Expression Syntax** for details on sort expressions. By default, output is sorted by time bin only.

label *output-label*

Insert the *output-label* to the output filename for this aggregation before the extension. This is recommended when using multiple aggregation expressions for fanout; if not present in the fanout case, fanned out files will be labelled numerically in the order they appear in the aggregation expression.

Filter Expression Syntax

Each filter expression defines permitted values for each field in the flow; only if every field in a flow matches the filter does the flow pass the filter. Flows that do not pass the filter are simply dropped. Filter expressions are applied to flows during biflow matching (in the case of prefilters) and before aggregation.

Each filter expression takes the following form:

filter [**bin** *time-rangelist*] [**sip** [**not**] *address-rangelist*] [**dip** [**not**] *address-rangelist*] [**sp** [**not**] *numeric-rangelist*] [**dp** [**not**] *numeric-rangelist*] [**flows** *numeric-rangelist*] [**rev flows** *numeric-rangelist*] [**packets** *numeric-rangelist*] [**rev packets** *numeric-rangelist*] [**octets** *numeric-rangelist*] [**rev octets** *numeric-rangelist*]

The **bin** rangelist filters flows by bin start time; the bin start time must appear within the rangelist, inclusive, to pass the filter. The **sip**, **dip**, **sp**, **dp**, and **proto** rangelists filter flows by key fields; the **not** keyword on each of these rangelists inverts the rangelist. The **flows**, **packets**, and **octets** and their **rev** (reverse) counterparts filter flows by value fields. Since filters are applied before aggregation, these value field filters apply to individual flows, not to aggregate flows. See **Rangelist Syntax** below for details on rangelists.

Rangelist Syntax

A rangelist is simply a comma-separated list of ranges. Three types of ranges are supported: time, address, and numeric. The supported time ranges (used for **bin** filtering) are as follows:

YYYY−MM−DD
Matches a single day (UTC).

*YYYY−MM−DD hh***:***mm***:***ss*
Matches from the given time to the end of the same day (23:59:59 UTC).

YYYY−MM−DD − *YYYY−MM−DD*
Matches from the beginning of the first day through the end of the second (UTC).

*YYYY−MM−DD hh***:***mm***:***ss* − *YYYY−MM−DD hh***:***mm***:***ss*
Matches any time between the first and last times, inclusive.

The supported address ranges (used in **perimeter** mode and for **sip** and **dip** filtering) are as follows:

*a***.***b***.***c***.***d*
Matches a single IPv4 address.

*a***.***b***.***c***.***d*/*m*
Matches any IPv4 address in the specified CIDR block.

*a***.***b***.***c***.***d*−*e***.***f***.***g***.***h*
Matches any IPv4 address between the first and last addresses, inclusive.

The supported numeric ranges are as follows:

n Matches a single integer.

m−*n*
Matches integers between the first and last, inclusive.

<*m* Matches integers less than the given integer.

>*m* Matches integers greater than the given integer.

Sort Expression Syntax

Each sort expression defines the sort order of the aggregated flow output. Sort expressions take the following form:

sort (**srcid** | **sip** | **dip** | **proto** | **sp** | **dp** | **octets** | **rev octets** | **packets** | **rev packets** | **flows** | **rev flows** | **src hosts** | **dest hosts** | **src ports** | **dest ports**) [**asc** | **desc**] ... [**limit** *limit-count*]

Multiple fields may appear in the sort expression; subsequent fields are only compared when all previous fields are equal. If **limit** is present, only the first *limit-count* records will be output per time bin. This can be used to build top-N lists.

EXAMPLES

The following command-line will read SiLK data from standard input and count flows and packets per hour per source class-C network:

```
nafalize --intype silk
         bin 1 hr aggregate sip/24 count flows packets
```

The following command-line will read IPFIX data from UDP on the loopback interface and count octets per thirty minutes per destination address and port, for HTTP and HTTPS traffic only on well-known ports:

```
nafalize --intype ipfix-udp --in 127.0.0.1,4739
         bin 30 min aggregate dip dp count octets
         filter proto 6 dp 80, 433
```

The following command-line will read packet data from a dump file and count distinct hosts per five minute bin, sorted by source then destination host count in descending order

```
nafalize --intype pcap
         --in dumpfile.pcap --out hostcount.naf
         bin 5 min aggregate count hosts
         sort src host desc dest host desc
```

The following command-line will read packet data from an interface and return the top ten destination ports per minute, by octets received by the client. It will write a new NAF output file every 30 minutes (1800 sec).

```
nafalize --intype pcap --in en1
         --live --rotate-delay 1800
         bin 1 min aggregate dp count octets
         sort rev octets desc limit 10
```

SIGNALS

nafalize responds to **SIGINT** or **SIGTERM** by terminating input processing, aggregating and flushing any pending flows to the current output, and exiting.

CAVEATS

The following caveats apply to nafalize's operation.

ICMP Type and Code

The NAF aggregated flow format stores ICMP Type and Code information in the destinationTransportPort IPFIX information element (**dp** in aggregation, filter, and sort expressions). This is a non-standard usage of this IE. The type appears in the high-order byte of the destination port, and the code in the low-order byte. Source and destination ports are *not* reversed during bidirectional flow matching for ICMP flows in order to keep ICMP type and code information from also appearing in the source port; this is not as much of an issue as it may first appear as ICMP flows are inherently unidirectional anyway.

The following aspects of this behavior may be considered bugs: first, since the type is high−order, filtering for all codes of a specific ICMP type requires a complex rangelist. Second, biflow matching knows nothing about specific ICMP types and codes, so, for example, Echo Request/Echo Response flows will not be properly matched. No fix is presently planned for these issues.

PCAP Note

When aggregating packet data from libpcap packet capture files, or from live capture, NAF aggregates each single packet as its own ‘‘psuedoflow’’; it does no real flow assembly of its own. Flow counters for packet capture are 0 for non-TCP flows, and count packets with the SYN flag set for TCP flows. Use a flow sensor such as *yaf*(1) or *argus*(5) to preprocess the packet data if you need accurate flow counts.

BUGS

Known issues are listed in the **README** file in the NAF tools source distribution. Note that NAF should be considered alpha-quality software; not every concievable input and aggregation is exhaustively tested at each release, and specific features may be completely untested. Please be mindful of this before deploying NAF in production environments. Bug reports and feature requests may be sent directly to the author, Brian Trammell, via email at <bht@cert.org>.

AUTHORS

Brian Trammell <bht@cert.org>, for the CERT Network Situational Awareness Group, http://www.cert.org/netsa.

SEE ALSO

nafscii (1), *tcpdump* (1), *yaf* (1)