
snarf Documentation

Release 0.2.4

Carnegie Mellon University

January 06, 2015

CONTENTS

1	Overview	1
1.1	Requirements	2
1.2	Installation	2
1.3	License	3
1.4	Support	3
2	libsnarf	5
2.1	Alert API	5
2.2	Source API	10
2.3	Sink API	11
3	snarf-python	13
3.1	snarf — SNARF Python Interface	13
4	snarfd	17
5	GNU GENERAL PUBLIC LICENSE	23
	Index	31

OVERVIEW

`snarf` is a distributed alert reporting system. Applications can use `snarf`'s libraries to send network alert messages, which can then be routed to multiple destinations in a configurable manner. `snarf` is designed to allow application and script developers to emit network alert messages without being concerned with the details of how the messages will be formatted downstream, or what destinations they will be routed to.

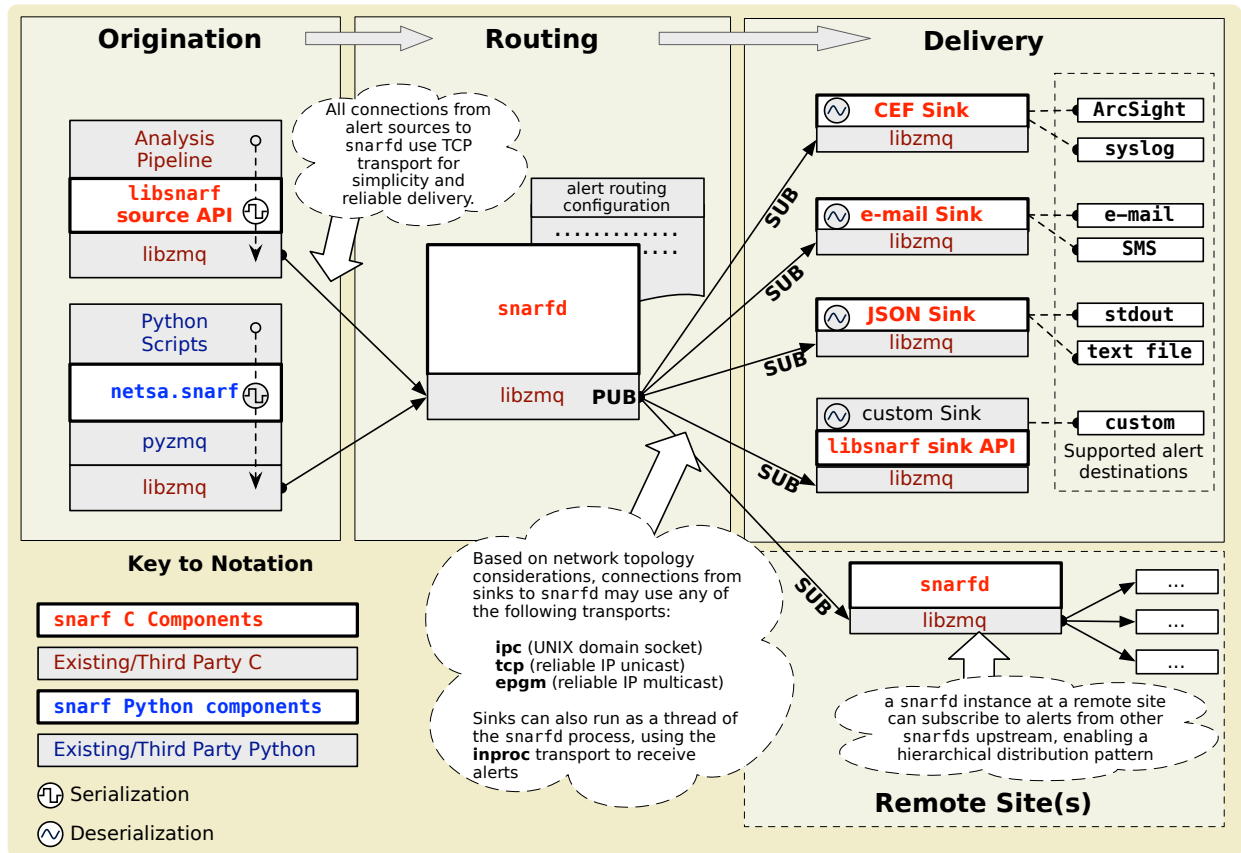
Alert processing happens in three steps:

- **origination** - An alert begins with an application call to either the alert and source APIs of *libsnarf* or *snarf-python*. Alert sources use these APIs to build alerts and send them to *snarfd*, an alert routing daemon.
- **routing** - *snarfd* matches fields in the alert's envelope against a set of configured routing rules, and routes the alert message to the appropriate destination(s) via a publish-subscribe mechanism. To support a distributed, hierarchical alerting architecture, *snarfd* processes can subscribe to alert channels on another *snarfd* upstream to distribute alerts from remote sites.
- **delivery** - Alert sinks subscribe to alert channels published by *snarfd*, delivering the alerts to the appropriate destination.

The `snarf` suite consists of the following software components:

- *libsnarf*, a C library for constructing, sending, and receiving alerts.
- *snarf-python*, a Python module for `snarf` alerting.
- *snarfd*, an alert routing daemon

The following system diagram details the flow of alert messages through these software components, and how these components interact with external systems.



1.1 Requirements

libsnarf requires the following third-party software:

- glib version 2.22 or later
- zeromq version 2.2.x
- protobuf-c version 0.15 or later

snarfd has the same requirements as libsnarf, along with:

- libyaml version 0.1.4 or later

snarf-python requires the following Python modules and third-party packages:

- pyzmq version 2.1.9 or later
- netsa-python version 1.4 or later
- protobuf version 2.3.0 or later

1.2 Installation

If you're running RedHat Enterprise Linux 7 or CentOS 7, we recommend using the [cert-forensics-repository](#) and installing using yum.

After configuring this repository, install snarf with, e.g.:

```
$ yum install snarf snarf-devel snarf-python
```

Installation of `libsнарf` and `snarfd` is accomplished through a standard `./configure && make && make install` process. If a supported Python installation is detected during installation, `snarf-python` will also be installed.

1.3 License

- GNU General Public License (GPL) Rights pursuant to Version 2, June 1991
- Government Purpose License Rights (GPLR) pursuant to DFARS 252.227-7013

1.4 Support

For help with snarf installation/usage, or to submit a bug report, send email to netsa-help@cert.org

LIBSNARF

`libsnarf` is a library that enables application developers to:

- build and parse snarf alert messages from C applications
- send alerts to a `snarfd` for routing to alert destinations
- subscribe to and receive alerts

These three capabilities correspond respectively to the `snarf` *Alert API*, *Source API*, and *Sink API*.

2.1 Alert API

A `snarf` alert is structured as follows:

Envelope	
generator generator_version timestamp severity analysis_tags	
Body	
field1	[value1, ...]
field2	[value1, ...]
...	

The envelope contains all mandatory fields that must be present in every alert. These fields are used to route the alert to the appropriate destination based on routing logic defined in the `snarfd` configuration file.

The fields and values present in the alert body are application-specific, but it is recommended that the set of fields and values are consistent among alerts tagged with a particular analysis tag. The receiver of the alert can then use these tags to determine which fields to expect in the alert body.

2.1.1 Constructing Alerts

To create a `snarf` alert object, use the `snarf_alert_new` function:

```
snarf_alert_t * snarf_alert_new (snarf_alert_severity_t severity, uint64_t timestamp)
```

Create a new alert.

The new alert will have the ‘severity’ and ‘timestamp’ fields set based on those arguments.

Parameters

- `severity` - the severity of the alert
- `timestamp` - 64-bit (epoch microsecond) timestamp of the alert

The severity field should be one of:

snarf_alert_severity_t enum

Values:

- `ALERT_VERYLOW` = 1 -
- `ALERT_LOW` = 2 -
- `ALERT_MEDIUM` = 3 -
- `ALERT_HIGH` = 4 -
- `ALERT_VERYHIGH` = 5 -

2.1.2 Adding Fields

Fields can be added to the alert with the following functions:

```
void snarf_alert_add_flow_v4 (snarf_alert_t * alert, uint64_t stime, uint32_t elapsed,  
                             uint32_t sip, uint32_t dip, uint16_t sport, uint16_t dport,  
                             uint8_t proto, uint32_t packets, uint32_t bytes, uint8_t  
                             flags, uint8_t flags_initial, uint16_t application_id, char *  
                             sensor_name, char * flow_class, char * flow_type)
```

Add an IPv4 flow to an alert.

The `sip` and `dip` arguments should be in the host machine's byte order, as in struct `sockaddr_in.sin_addr.s_addr`

```
void snarf_alert_add_flow_v6 (snarf_alert_t * alert, uint64_t stime, uint32_t elapsed,  
                             uint8_t sip[16], uint8_t dip[16], uint16_t sport, uint16_t  
                             dport, uint8_t proto, uint32_t packets, uint32_t bytes,  
                             uint8_t flags, uint8_t flags_initial, uint16_t application_id,  
                             char * sensor_name, char * flow_class, char * flow_type)
```

Add an IPv6 flow to an alert.

The `sip` and `dip` arguments should be in network byte order, as in `sockaddr_in6.sin6_addr`

```
void snarf_alert_add_text_field (snarf_alert_t * alert, const char * name, const char *  
                                value)
```

Add a text field to an alert.

Parameters

- `alert` - a snarf alert structure
- `name` - the name of the field to add
- `value` - the text to add

```
void snarf_alert_add_int_field (snarf_alert_t * alert, const char * name, int64_t value)
```

Add an integer field to an alert.

Parameters

- `alert` - a snarf alert structure
- `name` - the name of the field to add

- `value` - the integer to add

void **snarf_alert_add_double_field**(snarf_alert_t * *alert*, const char * *name*, double *value*)

Add a double field to an alert.

Parameters

- `alert` - a snarf alert structure
- `name` - the name of the field to add
- `value` - the double value to add

void **snarf_alert_add_ipset_field**(snarf_alert_t * *alert*, const char * *name*, uint8_t * *data*, size_t *len*)

Add an IPSet field to an alert.

Parameters

- `alert` - a snarf alert structure
- `name` - the name of the field to add
- `data` - the IPset's binary data
- `len` - the length of the IPset's binary data

void **snarf_alert_free**(snarf_alert_t * *alert*)

Free an alert's memory.

Parameters

- `alert` - Pointer to the alert structure to be freed

2.1.3 Accessing Alert Data

Alert sink code should use the following functions to work with alert objects once they are received:

int **snarf_alert_severity**(snarf_alert_t * *alert*)

Get an alert's severity as an integer.

Return

the alert's severity

Parameters

- `alert` - a snarf alert structure

snarf_field_t * **snarf_alert_get_field**(snarf_alert_t * *alert*, const char * *key*)

Retrieve a field from an alert.

Return

a snarf field structure

Parameters

- `alert` - a snarf alert structure
- `key` - the name of the field to retrieve

int **snarf_alert_field_value_count** (snarf_field_t * *field*)

Get the number of values in the given field.

snarf_value_t * **snarf_alert_field_value** (snarf_alert_t * *alert*, char * *fieldname*, int *index*)

Get a value stored in an alert field.

Return

a snarf value structure

Parameters

- *alert* - a snarf alert structure
- *fieldname* - the name of the field to retrieve
- *index* - an index into the list of values

2.1.4 Printing / Formatting Alert Messages

snarf Output Buffers

snarf output buffers handle some aspects of field delimiting and formatting for you.

To print alerts, first create an output buffer to hold the printed data:

snarf_output_buffer_t * **snarf_output_buffer_new** (size_t *len*)

To customize formatting in an alert buffer, use the following functions:

void **snarf_output_buffer_set_format** (snarf_output_buffer_t * *outbuf*, snarf_output_format_t *format*)

The built-in output formats are:

snarf_output_format_t enum

Values:

- SNARF_OUTPUT_BUFFER_RAW = 0 -
- SNARF_OUTPUT_BUFFER_DELIMITED -
- SNARF_OUTPUT_BUFFER_JSON -
- SNARF_OUTPUT_BUFFER_XML -

For the delimited output format, you can set the delimiter with:

void **snarf_output_buffer_set_delimiter** (snarf_output_buffer_t * *outbuf*, char *delim*)

Alert severity can be printed as either a symbolic name e.g. LOW, or an integer:

void **snarf_output_buffer_set_severity_format** (snarf_output_buffer_t * *outbuf*, snarf_output_severity_format_t *format*)

snarf_output_severity_format_t enum

Values:

- SNARF_OUTPUT_SEVERITY_FORMAT_INT = 0 -
- SNARF_OUTPUT_SEVERITY_FORMAT_NAME -

The format of the alert timestamp can be changed with:

```
void snarf_output_buffer_set_timestamp_format (snarf_output_buffer_t * outbuf,
                                              snarf_output_timestamp_format_t
                                              format)
```

The available timestamp formats are:

snarf_output_severity_format_t enum

Values:

- SNARF_OUTPUT_SEVERITY_FORMAT_INT = 0 -
- SNARF_OUTPUT_SEVERITY_FORMAT_NAME -

The format of any TCP flags printed in the alert's flow fields can be set with:

```
void snarf_output_buffer_set_tcp_flags_format (snarf_output_buffer_t * outbuf,
                                              snarf_output_tcp_flags_format_t
                                              format)
```

The available TCP flag formats are:

snarf_output_tcp_flags_format_t enum

Values:

- SNARF_OUTPUT_TCP_FLAGS_FORMAT_COMPACT = 0 -
- SNARF_OUTPUT_TCP_FLAGS_FORMAT_VERBOSE -

After you're done with the output buffer, free it with:

```
void snarf_output_buffer_free (snarf_output_buffer_t * outbuf)
```

Field Printing Helpers

Several helper functions are available for printing alert data:

```
void snarf_alert_print_envelope_field (snarf_output_buffer_t * outbuf, snarf_alert_t
                                         * alert, const char * fieldname)
```

Print a field from an alert's envelope.

Parameters

- *outbuf* - a snarf output buffer
- *alert* - a snarf alert
- *fieldname* - the name of the envelope field to print

```
void snarf_alert_print_value (snarf_output_buffer_t * outbuf, snarf_value_t * value)
```

Print a field value from an alert.

Parameters

- *outbuf* - a snarf output buffer
- *value* - the value to print

```
void snarf_alert_print_string (snarf_output_buffer_t * outbuf, char * str)
```

Print a string to an output buffer.

Parameters

- *outbuf* - a snarf output buffer
- *str* - the string to append

void **snarf_alert_print_string_raw** (snarf_output_buffer_t * *outbuf*, char * *str*)
Print a string to an output buffer with no field delimiting or special formatting.

Parameters

- *outbuf* - a snarf output buffer
- *str* - the string to append

void **snarf_alert_print_flow_field** (snarf_output_buffer_t * *outbuf*, snarf_value_t * *value*, const char * *fieldname*)
Print a flow field value to an output buffer.

Parameters

- *outbuf* - a snarf output buffer
- *value* - a flow value
- *fieldname* - the name of the flow field to print

void **snarf_alert_write_ipset** (const char * *filename*, snarf_value_t * *value*)
Write an IPset value to a file.

Parameters

- *filename* - the name of the file to write to
- *value* - an IP set value

2.2 Source API

To begin send `snarf` alerts, create a new alert source with:

snarf_source_t * **snarf_source_init** (char * *source_name*, char * *source_version*, char * *destination*)

Initialize an alert source.

Should be called once on program startup.

Return

a snarf alert source

Parameters

- *source_name* - name of the alerting software
- *source_version* - version of the alerting program
- *destination* - socket specifier for the remote socket to send alerts to, which may be NULL to print alerts locally

By convention, source names should be in the form of the reverse fully-qualified domain name of the software publisher, followed by the name of the software – e.g., if an alert detection program “foobar” is published by the “hackers” division of Acme Corporation at <http://www.hackers.acme.com/> the generator string should be “com.acme.hackers.foobar”.

The *dest* argument is a *socket specifier*.

Once the alert has been constructed using the alert API, it can be sent with:

```
int snarf_source_send_alert (snarf_source_t * source, char * tags, snarf_alert_t * alert)
```

Send an alert.

Sends the alert to the destination given when the alert context was created. The alert is freed once it has been sent.

Parameters

- *source* - a snarf alert source
- *tags* - a comma-separated list of tags that inform upstream components of what type of analysis was used to generate the alert.
- *alert* - a snarf alert

When your application is done sending alerts, you should free the source with:

```
void snarf_source_destroy (snarf_source_t * source)
```

Shutdown the alert source and free its resources.

Parameters

- *source* - a snarf alert source

2.3 Sink API

The `snarf` sink API facilitates the development of programs that wish to subscribe to alerts.

To begin receiving alerts, create a new alert sink with:

```
snarf_sink_t * snarf_sink_init (char * origin)
```

Initialize an alert sink.

The sink will receive alerts on socket 'origin'

Parameters

- *origin* - a socket specifier for the remote socket where alerts are published.

If you simply want to output alerts using one of the built-in output formats, you can do so by configuring the sink with the following function:

```
int snarf_sink_configure (snarf_sink_t * sink, const char * sink_id)
```

Configure an alert sink using one of the built-in alert output types.

Return

zero on success, nonzero on error

Parameters

- *sink* - a snarf sink
- *sink_id* - the identifier for the sink in the sink configuration
- *config* - a configuration object passed into the sink callback functions

For more custom alert processing, configure the sink with this function:

```
int snarf_sink_configure_full (snarf_sink_t      * sink,      snarf_sink_init_fn_t
                               init_fn,      snarf_sink_alert_fn_t  process_fn,
                               snarf_sink_destroy_fn_t  destroy_fn,  snarf_config_t
                               * config)
```

Configure an alert sink for custom processing using callback functions.

Return

zero on success, nonzero on error

Parameters

- *sink* - a snarf sink
- *init_fn* - the custom sink's initialization function
- *process_fn* - the custom sink's processing (alert dispatch) function
- *destroy_fn* - the custom sink's termination function
- *config* - a configuration object passed into the sink callback functions

To subscribe to specific alert channels, call:

```
int snarf_sink_subscribe (snarf_sink_t * sink, const char * channel)
```

Subscribe to the given alert channel.

Return

zero on success, nonzero on error

Parameters

- *sink* - a snarf sink
- *channel* - the name of the channel to subscribe to

Keep in mind that if you do not subscribe to a specific channel, you will receive all alerts broadcast on the sink's destination socket.

To begin processing alerts:

```
int snarf_sink_process (snarf_sink_t * sink)
```

Begin processing alerts using the sink's processing callback function.

The callback function will be executed in a background thread.

Return

zero on success, nonzero on error

Parameters

- *sink* - a snarf alert sink

To stop processing of alerts:

```
void snarf_sink_destroy (snarf_sink_t * sink)
```

Shutdown an alert sink, unsubscribing from the alert channel.

This will terminate the sink's background processing thread and free up any resources used.

Parameters

- *sink* - a snarf alert sink

Much like the `libsnarf` APIs, the structure of `snarf`'s application interfaces is divided into alert, source, and sink components.

3.1 `snarf` — SNARF Python Interface

3.1.1 `snarf.alert` — SNARF Alert Class

`snarf.alert`

The `snarf.alert` module provides classes to represent network alert messages and various components of such messages, such as network flows.

class `snarf.alert.Alert` (*fields*, *generator=None*, *generator_version=None*, *severity=1*, *timestamp=None*, *analysis_tags=None*)

A class representing network alerts. Each alert has a fixed set of required “envelope” fields and set of variable “alert body” fields.

classmethod `from_message` (*envelopemsg*, *bodymsg*)

Build an `Alert` object from a protobuf message.

to_message ()

Serialize an `Alert` object to a protobuf message.

get_generator ()

Returns an identifier representing the entity that generated the alert (e.g., “org.cert.netsa.pipeline”).

get_generator_version ()

Returns a string representation of the version of the generator.

get_timestamp ()

Returns a `datetime.datetime` object representing the time at which the alert was generated.

get_tags ()

Returns a list of strings corresponding to tags associated with an alert. Different generators may impose additional semantics on tags; it is up to the user to apply those semantics appropriately.

get_fields ()

Returns a dict of the alert fields.

add_tags (*tags*)

Tag an alert with one or more tags.

class `snarf.alert.AlertSeverity` (*severity*)

An object representing the perceived importance of an alert.

```
class snarf.alert.AlertFlow(stime, elapsed, sip, dip, sport, dport, proto, packets, bytes, flags,
                             flags_initial, sensor_name, flow_class, flow_type, application_id)
    Representation of a network flow.
```

3.1.2 snarf.source — SNARF Alert Source Class

snarf.source

The `snarf.source` module provides an interface for applications to send alerts to snarf alert destinations.

In the following example, an alert object is created for each line of input coming from the SiLK `rwcut(1)` tool:

```
from snarf import *
import fileinput

# create the source
source = Source("org.cert.netsa.test", "0.0.1",
               destination="tcp://localhost:5555",
               default_tags=["type=Evaluation"])

# generate input with:
# rwcut --no-titles --no-columns \
#       --fields=sTime,duration,sIP,dIP,sPort, \
#       dPort, protocol,packets,bytes,flags,initialFlags,sensor, \
#       class,type,application
for line in fileinput.input():

    (stime, elapsed, sip, dip, sport, dport, proto,
     packets, bytes, flags, flags_initial,
     sensor_name, flow_class, flow_type,
     application_id, dummy) = line.split('|')
    elapsed = int(float(elapsed) * 1000)
    sport = int(sport)
    dport = int(dport)
    proto = int(proto)
    packets = int(packets)
    bytes = int(bytes)
    application_id = int(application_id)
    f = AlertFlow(stime, elapsed, sip, dip, sport, dport, proto,
                  packets, bytes, flags, flags_initial,
                  sensor_name, flow_class, flow_type, application_id)
    a = Alert({'flow': f})
    source.send(a, tags=["foo=bar", "baz"])
```

```
class snarf.source.Source(name: str, version: str, endpoint:str, default_tags: list)
```

A class representing a producer of network alerts.

“name” and “version” should be the name and version of the program sending the alerts. “destination” is a socket specifier of the remote socket to send alerts to. If desired, a list of tags can be added to all alerts sent by this source by supplying the `default_tags` argument.

3.1.3 snarf.sink — SNARF Alert Sink Class

snarf.sink

The `snarf.sink` module provides an interface for applications to receive network alerts.

Using a snarf sink is simply a matter of defining a callback function that takes a `snarf.alert.Alert` object as its only argument, and processes that alert as needed. Then, create a `Sink` object, and pass the callback into the `Sink`'s constructor:

```
from snarf import *

def process_alert(alert):
    print "[%s %s] %d %s" % (alert.get_generator(),
                           alert.get_generator_version(),
                           alert.get_severity(),
                           alert.get_timestamp())

sink = Sink(process_alert,
            origin="tcp://localhost:5556",
            channel=channel)
# process_alert now being called for each alert in the background

# use the .stop() method when you're done
sink.stop()
```

class `snarf.sink.Sink` (*callback, endpoint: str, channel: str*)

A class representing a consumer of network alerts. The given callback function will be called for each alert received on the given channel.

stop()

Terminate a `Sink`'s background processing.

SNARFD

snarfd is an alert routing daemon. Configuring a snarfd is accomplished by defining:

- a set of *sockets* on which alerts are received and sent
- a list of *routes* that map alert attributes to *alert channels*

Routes are organized into *route groups* that all operate on the same set of sockets. Simpler installations with just a single receiving and sending socket will only need a single route group, but more complex architectures may create multiple route groups, e.g. a group for local alerts, groups for alerts received from remote sites, etc.

Each route in a route group is defined by a set of input attributes, which map to an output alert channel.

It may be easier to explain these concepts by looking at an example snarfd configuration file:

```
# SNARF configuration file
%YAML 1.1
---
# This configuration file allows you to customize how snarfd routes alerts.

# Begin snarfd configuration
snarfd:

    # if this is true, snarfd will reload the conf file when it's modified
    reload: true

    sockets:
        # Use the "sockets" section to define a list of sockets snarfd listens
        # on. The simplest mode of operation would be two sockets, one for
        # inbound messages and one for outbound messages. The format of each
        # entry in this list is:
        #
        # <name>:
        #     endpoint: <endpoint>
        #     type: [pub|sub|pull]
        #     identity: <identity>
        #     hwm: <hwm>
        #     channel: <channel>
        #
        # where:
        #
        #     <name> is a meaningful symbolic name for the socket, such as "in".
        #     No spaces are allowed. This symbolic name is used to refer to the
        #     socket in the "routes" section below.
        #
        #     <endpoint> is a socket specifier in protocol://address:port
        #     format, with "protocol" being one of:
```

```
#
#      * tcp: TCP sockets (reliable unicast)
#      * epgm: EPGM sockets (reliable multicast)
#      * ipc: UNIX domain sockets (for processes on the same
#      * host)
#
# <type> indicates what the socket does. Valid types are:
#
#      * pull - a socket for receiving alerts from alert sources
#      * pub - a socket for publishing alerts for alert subscribers
#      * sub - a socket for subscribing to published alerts
#
# <hwm> is an optional "high water mark" setting. This is an
# integer specifying how many messages to keep in memory while
# waiting for the receiver to get them. The default is 1024.
#
# <channel>, which is only valid for "sub" socket types, is the name
# of an alert channel to subscribe to on the remote snarfd. For
# more info on alert channels, see the "routes" section.

inbound:
    # A socket named "inbound" that receives alerts on TCP port 5555
    endpoint: tcp://127.0.0.1:5555
    type: pull

outbound:
    # a socket named "outbound" that publishes alerts on TCP port 5556
    endpoint: tcp://127.0.0.1:5556
    type: pub
    hwm: 256

remote:
    # a socket named "remote" that subscribes to another snarfd's
    # "events" channel
    endpoint: tcp://10.10.10.10:5557
    type: sub
    channel: events

stats:
    # snarfd can output alerts containing statistics on the number of alerts
    # processed. Change "enabled" to true and stats will be sent to the
    # <socket> channel every <interval> seconds.
    enabled: false
    socket: inbound
    interval: 60

routes:
    # The "routes" section contains directives which define how alerts are
    # routed to subscribers via the sockets defined above. For simplicity,
    # routes are organized into "route groups" that operate on the same
    # incoming and outgoing sockets.
    #
    # The structure of each route group is:
    #
    # - from: <in-socket>
    # - to: <out-socket>
    # - rules:
    #     -
```

```

#         - in:
#           - <attribute>:
#             - <value>
#             - <value>
#           - <attribute>:
#             - <value>
#             - <value>
#         - out:
#           - <channel>
#
# where:
#
# <in-socket> is the symbolic name of a socket defined in the
# "sockets" section above that receives alerts from alert sources.
# Only sockets of types "pull" and "sub" are valid as "in" sockets
# in route group definitions.
#
# <out-socket> is the symbolic name of a socket defined in the
# "sockets" section above that publishes alerts to alert sinks.
# Only sockets of type "pub" are valid as "out" sockets in route
# group definitions.
#
# Each item in the rules: section of a route group specifies routing
# rules that apply to messages received on the "in:" socket. The
# routing is based on a set of attributes and a list of values for
# each of those attributes. At this time, the recognized attributes
# are:
#
# * generator - identifies the software that detected and
#               generated the alert. By convention, generator strings
#               should be in the form of the reverse fully-qualified domain
#               name of the software publisher, followed by the name of the
#               software -- e.g., if an alert detection program "foobar" is
#               published by the "hackers" division of Acme Corporation at
#               <http://www.hackers.acme.com/>, the generator string should
#               be "com.acme.hackers.foobar"
#
# * tag - identifies one or more textual tags in the alert that
#         describe the alert contents. These are nothing more than
#         free-form textual strings, with no spaces allowed. Alert
#         sources can also define a key=value structure within these
#         tags, e.g. "color=blue", but this is optional.
#
# For each attribute in a routing rule, you may specify one or more
# values to match that attribute with. The syntax of these <value>
# definitions allows the wildcard characters ? and * to match any
# single character or any number of characters, respectively.
#
# <channel> should be an alphanumeric name for a published alert
# channel. Channels allow alert subscribers to indicate the type of
# alert messages they would like to receive without needing to know
# any details about the attributes of the alerts themselves. In
# this manner, snarfd can be used not only to route alerts, but to
# organize them into meaningful collections.
#
- # route group "inbound" to "outbound"
  - from: inbound
  - to: outbound

```

```
- rules:
  - # route all alerts generated by the "foobar" software from
    # Acme Corp. to the "foo" channel
    - in:
      - generator:
        - com.acme.hackers.foobar
    - out:
      - foo
  - # route all alerts with the "evilhosts" tag to the "evil"
    # channel
    - in:
      - tag:
        - evilhosts
    - out:
      - evil
  - # route all alerts generated by Acme Corp. with the "urgent"
    # or "critical" tags, as well as any tag beginning with "bad",
    # to the "zomg" channel
    - in:
      - generator:
        - com.acme.*
      - tag:
        - urgent
        - critical
        - bad*
    - out:
      - zomg

- # route group "remote" to "outbound"
  - from: remote
  - to: outbound
  - rules:
    - # route alerts from an upstream snarfd to the "feeds.events"
      # channel
      - in:
        - generator:
          - org.example.data
        - tag:
          - color=blue
      - out:
        - feeds.events

# The "sink" section contains configuration for various alert destinations.
# Each sub-section within the "sink" section denotes a single sink
# configuration. At a minimum, each "sink" sub-section must consist of an
# identifier for the sink and a type: attribute that specifies the type of the
# sink. Currently-supported sink types are "json", "email", and "cef".
sink:

  # A JSON (JavaScript Object Notation) sink. JSON allows any snarf alert
  # data to be expressed in textual form using the same field structure that
  # snarf uses internally.
  json:
    type: json
    # JSON sinks support an "output_file" option for specifying the location
    # of the output. If not specified, it defaults to standard output.
    # outut_file: foo.txt
```



```
# An email sink, which sends alert data to a specified email address.
email:
  type: email
  from: root          # the email address to send alerts from
  to: root            # the email address to send alerts from
  sms_format: false   # if true, alert format is shortened for SMS

# A CEF (Common Event Format) sink.  Used for sending alerts to ArcSight.
cef:
  type: cef

  # Configure syslog output for CEF alerts.
  syslog:
    enabled: true      # if true, send alerts to syslog (stdout otherwise)
    facility: local4    # syslog facility to use local1 .. local7 or user

  # Each entry in the "fields" sub-section maps a CEF output field to
  # a field in the snarf alert.  The format of each field mapping is:
  # - output_field:
  #   - field_type: input_field
  # where field_type is one of:
  #   flow (a flow field in the alert),
  #   field (a non-flow field in the alert)
  #   string (raw text to be included verbatim in the alert output)
  fields:
    - src:
      - flow: sip
    - dst:
      - flow: dip
    - spt:
      - flow: sport
    - dpt:
      - flow: dport
    - proto:
      - flow: proto
    - start:
      - flow: stime
    - end:
      - flow: etime
    - cn1:
      - flow: packets
    - cn1Label:
      - string: Packets count
    - in:
      - flow: bytes
    - cn2:
      - flow: elapsed
    - cn2Label:
      - string: duration of flow in msec
    - cs1:
      - flow: sensor_name
    - cs1Label:
      - string: sensor name
    - cs2:
      - flow: flow_class
    - cs2Label:
      - string: class of sensor
    - cs3:
```

```
- flow: flow_type
- cs3Label:
  - string: type of sensor
- cs4:
  - flow: icmp_type_code
- cs4Label:
  - string: ICMP type,code
- cs5:
  - flow: flags
- cs5Label:
  - string: TCP flags bitwise OR
- app:
  - flow: application_id
- sourceGeoCountryCode:
  - field: sip.cc
- destinationGeoCountryCode:
  - field: dip.cc

# End snarfd configuration
```

GNU GENERAL PUBLIC LICENSE

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free

software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include

anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is

implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER

PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```


This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

A

`add_tags()` (snarf.alert.Alert method), 13
`Alert` (class in snarf.alert), 13
`AlertFlow` (class in snarf.alert), 13
`AlertSeverity` (class in snarf.alert), 13

F

`from_message()` (snarf.alert.Alert class method), 13

G

`get_fields()` (snarf.alert.Alert method), 13
`get_generator()` (snarf.alert.Alert method), 13
`get_generator_version()` (snarf.alert.Alert method), 13
`get_tags()` (snarf.alert.Alert method), 13
`get_timestamp()` (snarf.alert.Alert method), 13

S

`Sink` (class in snarf.sink), 15
`snarf.alert` (module), 13
`snarf.sink` (module), 14
`snarf.source` (module), 14
`Source` (class in snarf.source), 14
`stop()` (snarf.sink.Sink method), 15

T

`to_message()` (snarf.alert.Alert method), 13