NAME

argus.conf – **argus** resource file.

SYNOPSIS

argus.conf

COPYRIGHT

DESCRIPTION

This is the canonical argus configuration file. All options that argus supports can be turned on or modified using this configuration format. Argus will search for a system /etc/argus.conf file and will open it and use it to seed all configuration options.conf. Previous versions of Argus supported searching for argus.conf in $ARGUSPATH, $ARGUSHOME, $ARGUSHOME/lib, $HOME, and $HOME/lib, but this support is deprecated. All values in this file can be overriden by command line options, or other configuration files of this format when specified in using the -F option.

Argus will read any number of configuration files using the -F option, and command-line order is very important.

Variable Syntax

Variable assignments must be of the form:

VARIABLE=

with no white space between the VARIABLE and the '=' sign. Quotes are optional for string arguments, but if you want to embed comments, then quotes are required.

ARGUS_FLOW_TYPE / ARGUS_FLOW_KEY

The Argus can be configured to support a large number of flow types. The Argus can provide either type, i.e. uni-directional or bi-directional flow tracking and the flow can be further defined by specifying the key. The argus supports a set of well known key strategies, such as 'CLASSIC_5_TUPLE', 'LAYER_3_MATRIX', 'LAYER_2_MATRIX', formulate key strategies from a list of the specific objects that the Argus understands. See the man page for a complete description.

The default is the classic 5-tuple IP flow, CLASSIC_5_TUPLE.

There is no commandline equivalent.

ARGUS_FLOW_TYPE="Bidirectional"
ARGUS_FLOW_KEY="CLASSIC_5_TUPLE"

ARGUS_DAEMON

Argus is capable of running as a daemon, doing all the right things that daemons do. When this configuration is used for the system daemon process, say for /etc/argus.conf, this variable should be set to "yes".

In the examples seen in the ./support/Startup/argus scripts, this value is set to "yes", as the system startup strategy requires the program to daemonize themselves, returning a value to the system, hopefully quickly. Some systems, however, want to daemonize the tasks themselves, and those cases, the value must be set to "no".

which requires that this variable be set to "yes".

The default value is to not run as a daemon.

Commandline equivalent -d

ARGUS_DAEMON=no

ARGUS_MONITOR_ID

Argus Monitor Data is uniquely identifiable based on the source identifier that is included in each output record. This is to allow you to work with Argus Data from multiple monitors at the same time. The ID is 32 bits long, and argus suppors a number of formats as legitimate values. Argus support unsigned ints, IPv4 addresses and 4 bytes strings, as values.

The formats are discerned from the values provided. Double-quoted values are treated as strings, and are truncated to 4 characters. Non-quoted values are tested for whether they are hostnames, and if not, then they are tested wheter they are numbers.

The configuration allows for you to use host names, however, do have some understanding how ‘hostname‘ will be resolved by the nameserver before commiting to this strategy completely.

For convenience, argus supports the notion of "‘hostname‘" for assigning the probe’s id. This is to support management of large deployments, so you can have one argus.conf file that works for a lot of probes.

For security, argus does not rely on system programs, like hostname.1. It implements the logic of hostname itself, so don’t try to run arbitrary programs using this method, because it won’t work.

Commandline equivalent -e

ARGUS_MONITOR_ID=‘hostname‘ // IPv4 address returned
ARGUS_MONITOR_ID=10.2.45.3 // IPv4 address
ARGUS_MONITOR_ID=2435 // Number
ARGUS_MONITOR_ID="en0" // String

ARGUS_ACCESS_PORT

Argus monitors can provide a real-time remote access port for collecting Argus data. This is a TCP based port service and the default port number is tcp/561, the "experimental monitor" service. This feature is disabled by default, and can be forced off by setting it to zero (0).

When you do want to enable this service, 561 is a good choice, as all ra* clients are configured to try this port by default.

Commandline equivalent -P

ARGUS_ACCESS_PORT=561

ARGUS_BIND_IP

When remote access is enabled (see above), you can specify that Argus should bind only to a specific IP address. This is useful, for example, in restricting access to the local host, or binding to a private interface while capturing from another.

You can provide multiple addresses, separated by commas, or on multiple lines.

The default is to bind to any IP address.

Commandline equivalent -B

ARGUS_BIND_IP="::1,127.0.0.1"
ARGUS_BIND_IP="127.0.0.1"
ARGUS_BIND_IP="192.168.0.68"

ARGUS_INTERFACE

By default, Argus will open the first appropriate interface on a system that it encounters. For systems that have only one network interface, this is a reasonable thing to do. But, when there are more than one suitable interface, you should specify the interface(s) Argus should use either on the command line or in this file.

Argus can track packets from any or all interfaces, concurrently. The interfaces can be tracked as:

1. independant - this is where argus tracks flows from each interface independant from the packets seen on any other interface. This is useful for hosts/routers that have full-duplex interfaces, and you want to distinguish flows based on their interface. There is an option to specify a distinct srcid to each independant modeler.

2. duplex - where argus tracks packets from 2 interfaces as if they were two half duplex streams of the same link. Because there is a single modeler tracking the 2 interfaces, there is a single srcid that can be passed as an option.

3. bonded - where argus tracks packets from multiple interfaces as if they were from the same stream. Because there is a single modeler tracking the 2 interfaces, there is a single srcid that can be passed as an option.

Interfaces can be specified as groups using '[',']' notation, to build flexible definitions of packet sources. However, each interface should be referenced only once (this is due to performance and OS limitations, so if your OS has no problem with this, go ahead).

The lo (loopback) interface will be included only if it is specifically indicated in the option.

The syntax for specifying this either on the command line or in this file:

```
-i ind:all
-i dup:en0,en1/srcid
-i bond:en0,en1/srcid
-i dup:[bond:en0,en1],en2/srcid
-i en0/srcid -i en1/srcid  (equivalent '-i ind:en0/srcid,en1/srcid')
-i en0 en1     (equivalent '-i bond:en0,en1')
```

In all cases, if there is a "-e srcid" provided, this is used as the default. If a srcid is specified using this option, it overrides the default.

Srcid's are specified using the notion used for ARGUS_MONITOR_ID, as above.

Commandline equivalent -i

ARGUS_INTERFACE=any
ARGUS_INTERFACE=ind:all
ARGUS_INTERFACE=ind:en0/192.168.0.68,en2/192.168.2.1
ARGUS_INTERFACE=ind:en0/"en0",en2/19234
ARGUS_INTERFACE=en0

ARGUS_GO_PROMISCUOUS

By default, Argus will put its interface in promiscuous mode in order to monitor all the traffic that can be collected. This can put an undo load on systems.

If the intent is to monitor only the network activity of the specific system, say to measure the performance of an HTTP service or DNS service, you'll want to turn promiscuous mode off.

The default value goes into prmiscuous mode.

Commandline equivalent -p

ARGUS_GO_PROMISCUOUS=yes

ARGUS_CHROOT_DIR

Argus supports chroot(2) in order to control the file system that argus exists in and can access. Generally used when argus is running with privileges, this limits the negative impacts that argus could inflict on its host machine.

This option will cause the output file names to be relative to this directory, and so consider this when trying to find your output files.

Commandline equivalent -c dir

ARGUS_CHROOT_DIR=/chroot_dir

ARGUS_SETUSER_ID

Argus can be directed to change its user id using the setuid() system call. This is can used when argus is started as root, in order to access privileged resources, but then after the resources are opened, this directive will cause argus to change its user id value to a 'lesser' capable account. Recommended when argus is running as daemon.

Commandline equivalent -u user

ARGUS_SETUSER_ID=user

ARGUS_SETGROUP_ID

Argus can be directed to change its group id using the setgid() system call. This is can used when argus is started as root, in order to access privileged resources, but then after the resources are opened, this directive can be used to change argu's group id value to a 'lesser' capable account. Recommended when argus is running as daemon.

Commandline equivalent -g group

ARGUS_SETGROUP_ID=group

ARGUS_OUTPUT_FILE

Argus can write its output to one or a number of files, default limit is 5 concurrent files, each with their own independant filters.

The format is:
ARGUS_OUTPUT_FILE=/full/path/file/name
ARGUS_OUTPUT_FILE=/full/path/file/name "filter"

Most sites will have argus write to a file, for reliablity and performance. The example file name is used here as supporting programs, such as ./support/Archive/argusarchive are configured to use this file.

Commandline equivalent -w

ARGUS_OUTPUT_FILE=/var/log/argus/argus.out

ARGUS_OUTPUT_STREAM

Argus can write its output to one or a number of remote hosts. The default limit is 5 concurrent output streams, each with their own independant filters.

The format is:
ARGUS_OUTPUT_STREAM="URI [filter]"
ARGUS_OUTPUT_STREAN="argus-udp://host:port 'tcp and not udp'"

Most sites will have argus listen() for remote sites to request argus data, but for some sites and applications sending records without registration is desired. This option will cause argus to transmit records that match the optional filter, to the configured targets using UDP as the transport mechanism.

Commandline equivalent -w argus-udp://host:port

ARGUS_OUTPUT_STREAM=argus-udp://224.0.20.21:561

ARGUS_SET_PID

When Argus is configured to run as a daemon, with the -d option, Argus can store its pid in a file, to aid in managing the running daemon. However, creating a system pid file requires privileges that may not be appropriate for all cases.

When configured to generate a pid file, if Argus cannot create the pid file, it will fail to run. This variable, and the directory the pid is written to, is available to override the default, in case this gets in your way.

The default value is to generate a pid. The default path for the pid file, is '/var/run'.

No Commandline equivalent

ARGUS_SET_PID=yes
ARGUS_PID_PATH=/var/run

ARGUS_FLOW_STATUS_INTERVAL

Argus will periodically report on a flow's activity every ARGUS_FLOW_STATUS_INTERVAL seconds, as long as there is new activity on the flow. This is so that you can get a view into the activity of very long lived flows. The default is 60 seconds, but this number may be too low or too high depending on your uses.

The default value is 60 seconds, but argus does support a minimum value of 1. This is very useful for doing measurements in a controlled experimental environment where the number of flows is < 1000.

Commandline equivalent -S

ARGUS_FLOW_STATUS_INTERVAL=60

ARGUS_MAR_STATUS_INTERVAL

Argus will periodically report on a its own health, providing interface status, total packet and bytes counts, packet drop rates, and flow oriented statistics.

These records can be used as "keep alives" for periods when there is no network traffic to be monitored.

The default value is 300 seconds, but a value of 60 seconds is very common.

Commandline equivalent -M

ARGUS_MAR_STATUS_INTERVAL=300

ARGUS_DEBUG_LEVEL

If compiled to support this option, Argus is capable of generating a lot of debug information.

The default value is zero (0).

Commandline equivalent -D

ARGUS_DEBUG_LEVEL=0

ARGUS_GENERATE_PACKET_SIZE

Argus can be configured to generate packet size information on a per flow basis, which provides the max and min packet size seen . The default value is to not generate this data.

Commandline equivalent -Z

ARGUS_GENERATE_PACKET_SIZE=yes

ARGUS_GENERATE_JITTER_DATA

Argus can be configured to generate packet jitter information on a per flow basis. The default value is to not generate this data.

Commandline equivalent -J

ARGUS_GENERATE_JITTER_DATA=no

ARGUS_GENERATE_MAC_DATA

Argus can be configured to not provide MAC addresses in it audit data. This is available if MAC address tracking and audit is not a requirement.

The default value is to not generate this data.

Commandline equivalent -m

ARGUS_GENERATE_MAC_DATA=no

ARGUS_GENERATE_APPBYTE_METRIC

Argus can be configured to generate metrics that include the application byte counts as well as the packet count and byte counters.

Commandline equivalent -A

ARGUS_GENERATE_APPBYTE_METRIC=no

ARGUS_GENERATE_TCP_PERF_METRIC

Argus by default, generates extended metrics for TCP that include the connection setup time, window sizes, base sequence numbers, and retransmission counters. You can suppress this detailed information using this variable.

No commandline equivalent

ARGUS_GENERATE_TCP_PERF_METRIC=yes

ARGUS_GENERATE_BIDIRECTIONAL_TIMESTAMPS

Argus by default, generates a single pair of timestamps, for the first and last packet seen on a given flow, during the obseration period. For bi-directional flows, this results in loss of some information. By setting this variable to 'yes', argus will store start and ending timestamps for both directions of the flow.

No commandline equivalent

ARGUS_GENERATE_BIDIRECTIONAL_TIMESTAMPS=no

ARGUS_CAPTURE_DATA_LEN

Argus can be configured to capture a number of user data bytes from the packet stream.

The default value is to not generate this data.

Commandline equivalent -U

ARGUS_CAPTURE_DATA_LEN=0

ARGUS_FILTER_OPTIMIZER

Argus uses the packet filter capabilities of libpcap. If there is a need to not use the libpcap filter optimizer, you can turn it off here. The default is to leave it on.

Commandline equivalent -O

ARGUS_FILTER_OPTIMIZER=yes

ARGUS_FILTER

You can provide a filter expression here, if you like. It should be limited to 2K in length. The default is to not filter.

No Commandline equivalent

ARGUS_FILTER=""

ARGUS_PACKET_CAPTURE_FILE

Argus allows you to capture packets in tcpdump() format if the source of the packets is a tcpdump() formatted file or live packet source.

Specify the path to the packet capture file here.

ARGUS_PACKET_CAPTURE_FILE="/var/log/argus/packet.out"

ARGUS_SSF

Argus supports the use of SASL to provide strong authentication and confidentiality protection.

The policy that argus uses is controlled through the use of a minimum and maximum allowable protection strength, which is standard for SASL based appliations. Set these variable to control this policy. The default is no security policy.

ARGUS_MIN_SSF=0
ARGUS_MAX_SSF=0

ARGUS_ENV

Argus supports setting environment variables to enable functions required by the kernel or shared libraries. This feature is intended to support libraries such as the net pf_ring support for libpcap as supported by code at http://public.lanl.gov/cpw/

Setting environment variables in this way does not affect internal argus variable in any way. As a result, you can't set ARGUS_PATH using this feature.

Care should must be taken to assure that the value given the variable conform's to your systems putenv.3 system call. You can have as many of these directives as you like.

The example below is intended to set a libpcap ring buffer length to 300MB, if your system supports this feature.

ARGUS_ENV="PCAP_MEMORY=300000"

ARGUS_TUNNEL_DISCOVERY

Argus can be configured to discover tunneling protocols above the UDP transport header, specifically Teredo (IPv6 over UDP). The algorithm is simple and so, having this on by default may generate false tunnel matching.

The default is to not turn this feature on.

ARGUS_TUNNEL_DISCOVERY=no

ARGUS_EVENT_DATA

Argus supports the generation of host originated processes to gather additional data and statistics. These include periodic processes to poll for SNMP data, as an example, or to collect host statistics through reading procfs(). Or single run programs that run at a specified time.

These argus events, are generated from the complete list of ARGUS_EVENT_DATA directives that are specified here.

The syntax is:

```
Syntax is: "method:path|prog:interval[:postproc]"
   Where:  method = [ "file" | "prog" ]
       pathname | program = "%s"
       interval = %d[smhd] [ zero means run once ]
       postproc = [ "compress" | "compress2" ]
```

ARGUS_EVENT_DATA="file:/proc/vmstat:30s:compress"
ARGUS_EVENT_DATA="prog:/usr/local/bin/ralsof:30s:compress"

ARGUS_KEYSTROKE

This version of Argus supports keystroke detection and counting for TCP connections, with specific algorithmic support for SSH connections.

The ARGUS_KEYSTROKE variable turns the feature on. Values for this variable are:

ARGUS_KEYSTROKE="yes" - turn on TCP flow tracking
ARGUS_KEYSTROKE="tcp" - turn on TCP flow tracking
ARGUS_KEYSTROKE="ssh" - turn on SSH specific flow tracking
ARGUS_KEYSTROKE="no" [default]

The algorithm uses a number of variables, all of which can be modifed using the ARGUS_KEYSTROKE_CONF descriptor, which is a semicolon (';') separated set of variable assignments. Here is the list of supported variables:

DC_MIN - (int) Minimum client datagram payload size in bytes
DC_MAX - (int) Maximum client datagram payload size in bytes
GS_MAX - (int) Maximum server packet gap
DS_MIN - (int) Minimum server datagram payload size in bytes
DS_MAX - (int) Maximum server datagram payload size in bytes
IC_MIN - (int) Minimum client interpacket arrival time (microseconds)

LCS_MAX - (int) Maximum something - Not sure what this is
GPC_MAX - (int) Maximum client packet gap
ICR_MIN - (float) Minimum client/server interpacket arrival ratio
ICR_MAX - (float) Maximum client/server interpacket arrival ratio

All variables have default values, this variable is used to override those values. The syntax for the variable is:

ARGUS_KEYSTROKE_CONF="DC_MIN=20;DS_MIN=20"

ARGUS_KEYSTROKE="no"
ARGUS_KEYSTROKE_CONF=""

SEE ALSO

argus(8)