NetWitness Log Decoder

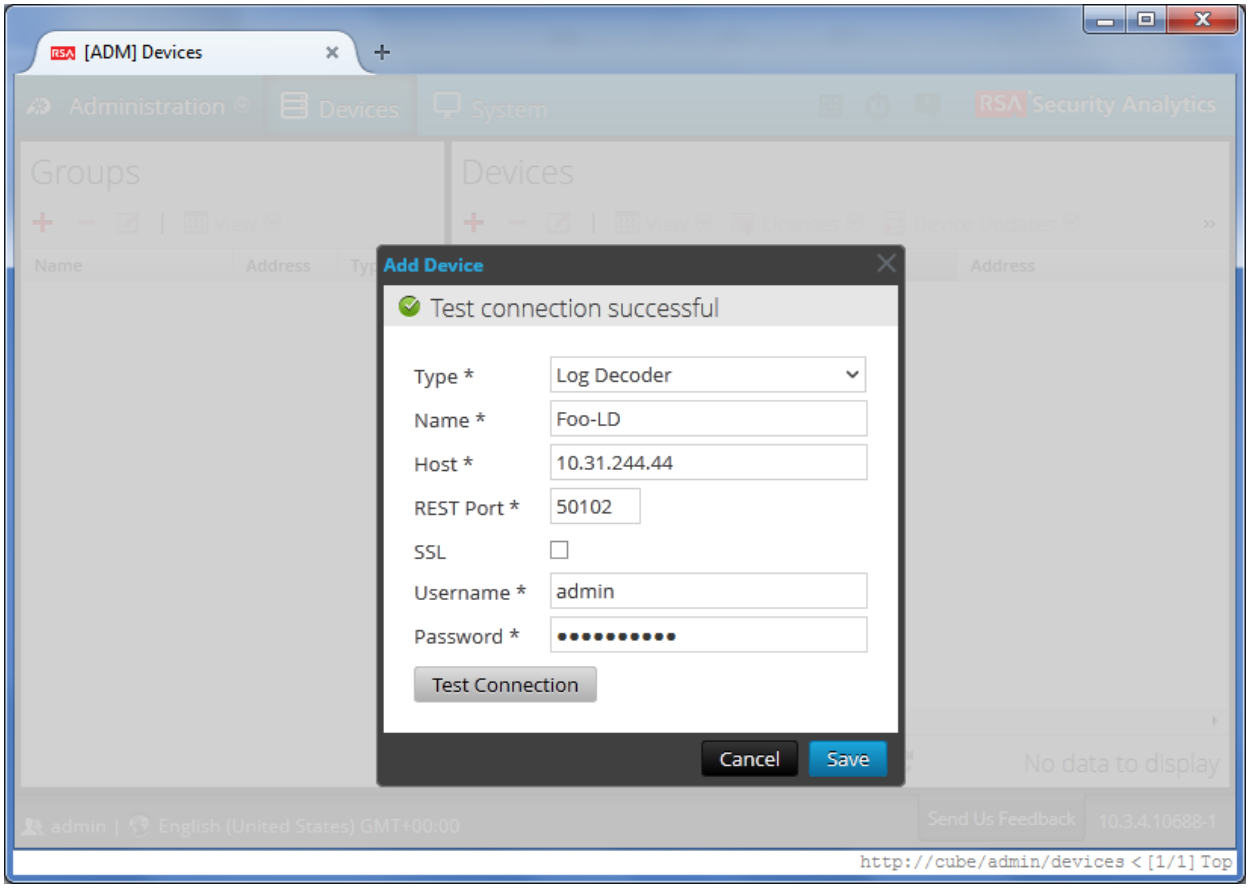
Susam Pal

RSA NetWitness

Basic Configuration

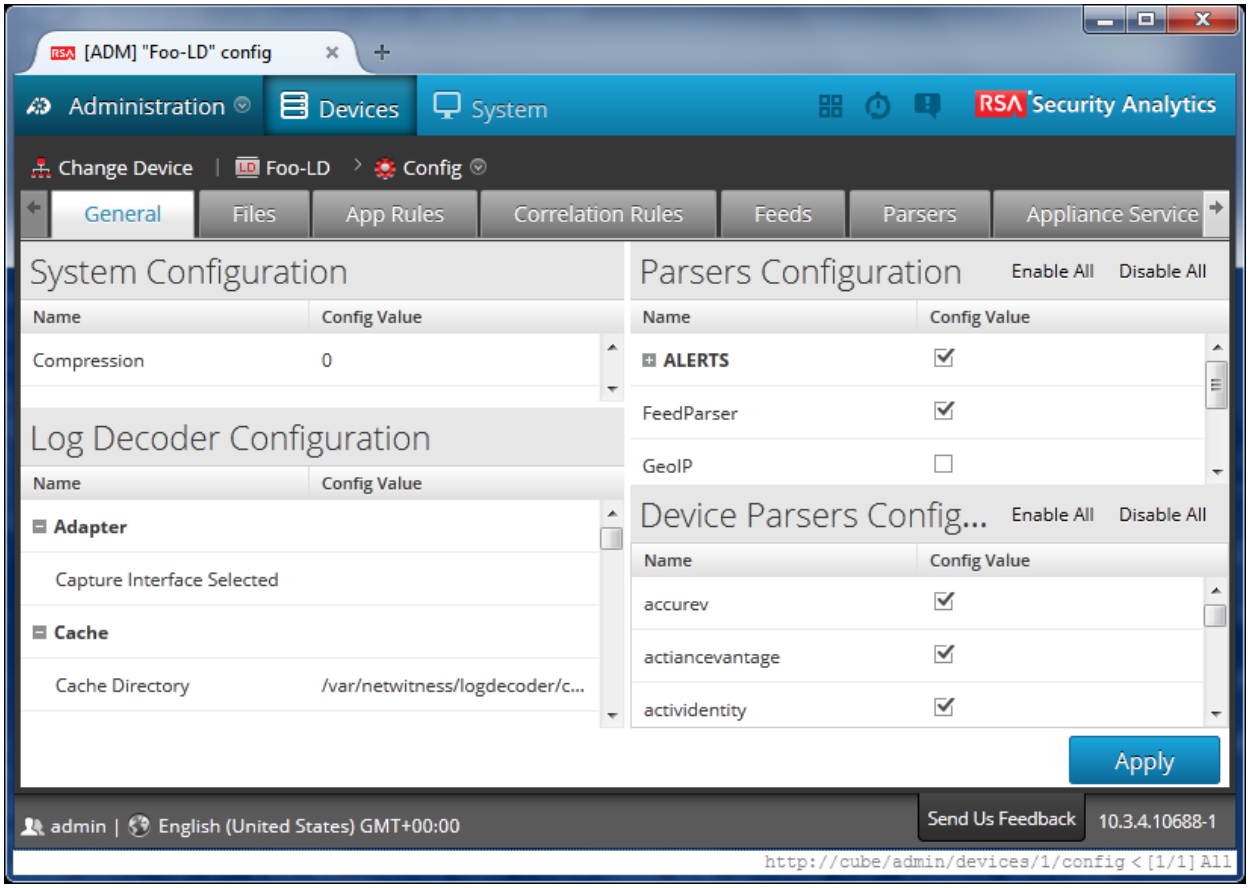
NetWitness Log Decoder

Basic Configuration



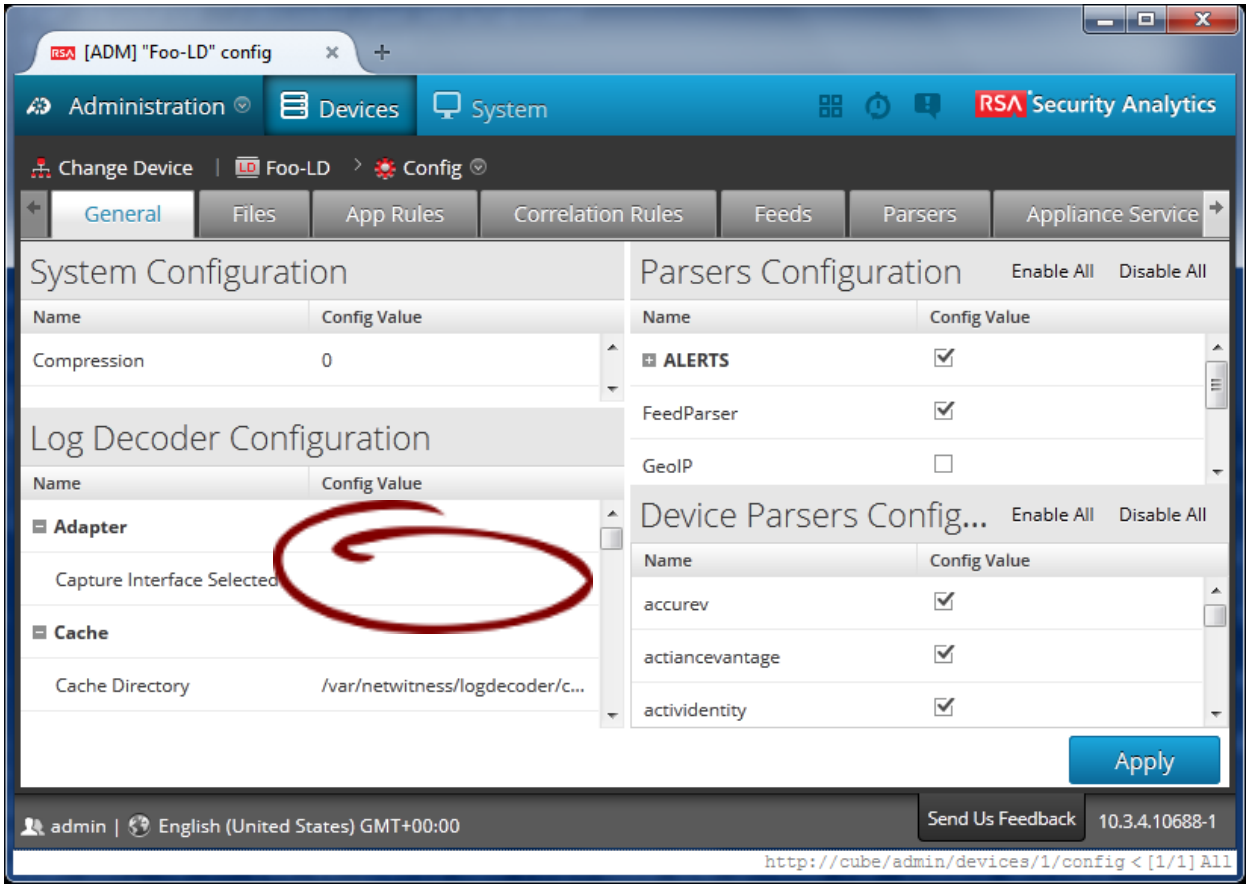
Add Log Decoder (Administration → Devices)

Basic Configuration



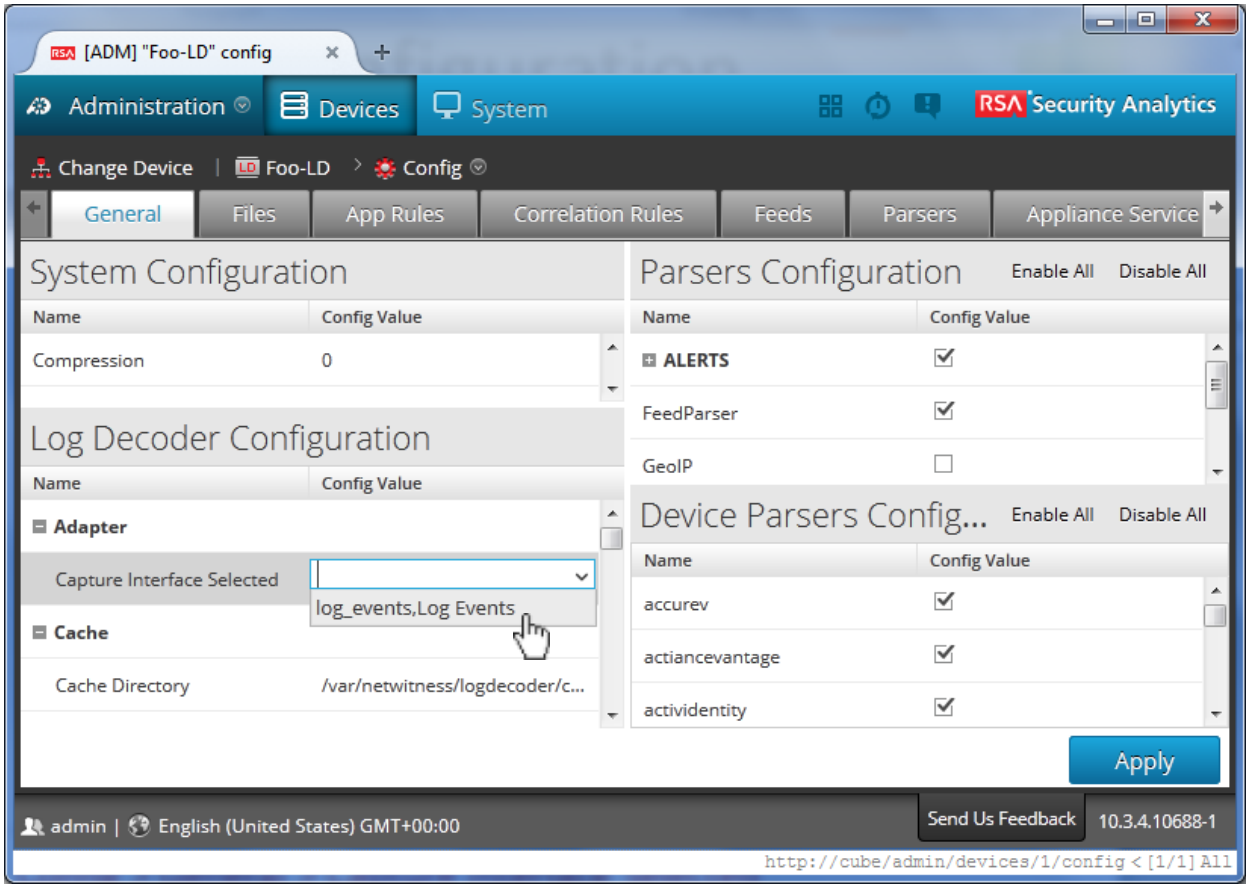
Default Configuration (LD → Config → General)

Basic Configuration



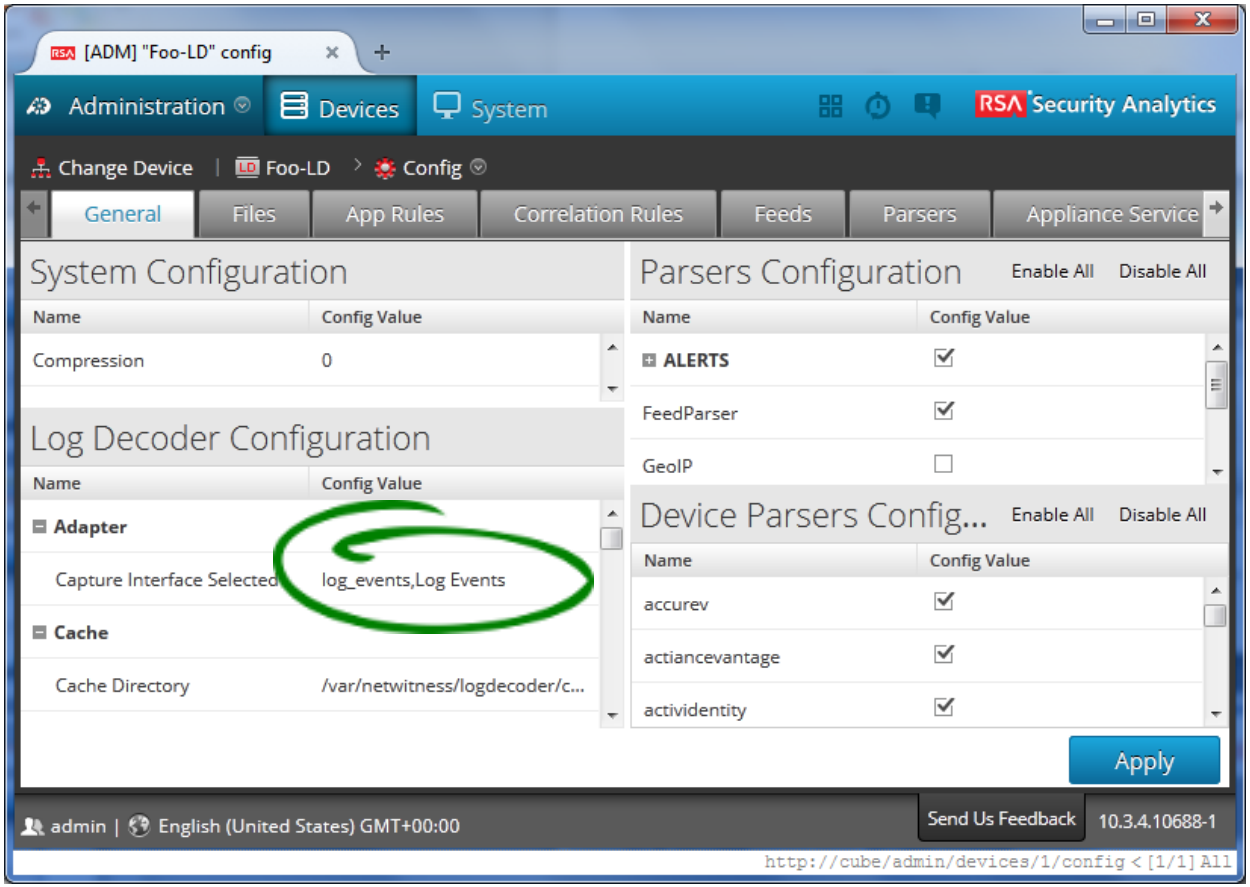
Capture Interface must be set

Basic Configuration



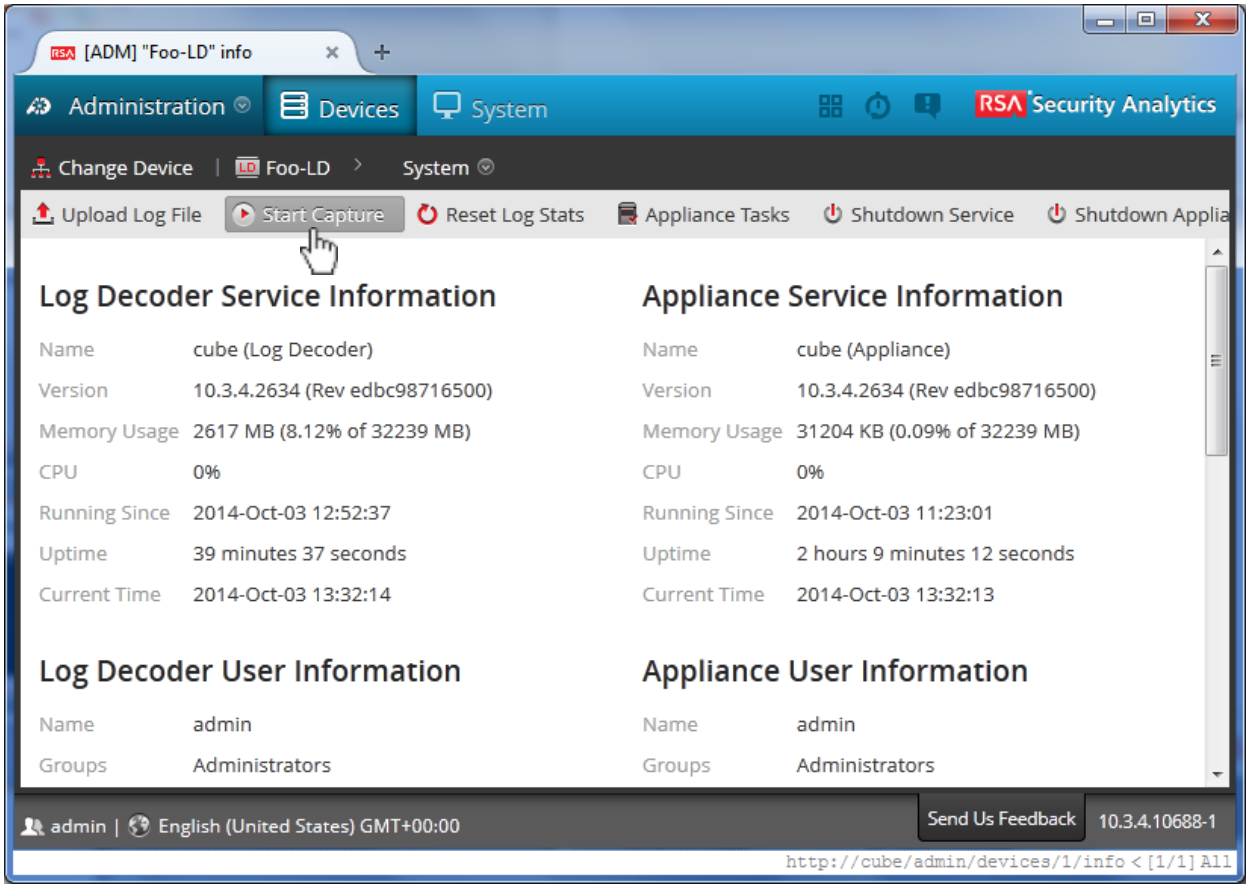
Set Capture Interface

Basic Configuration



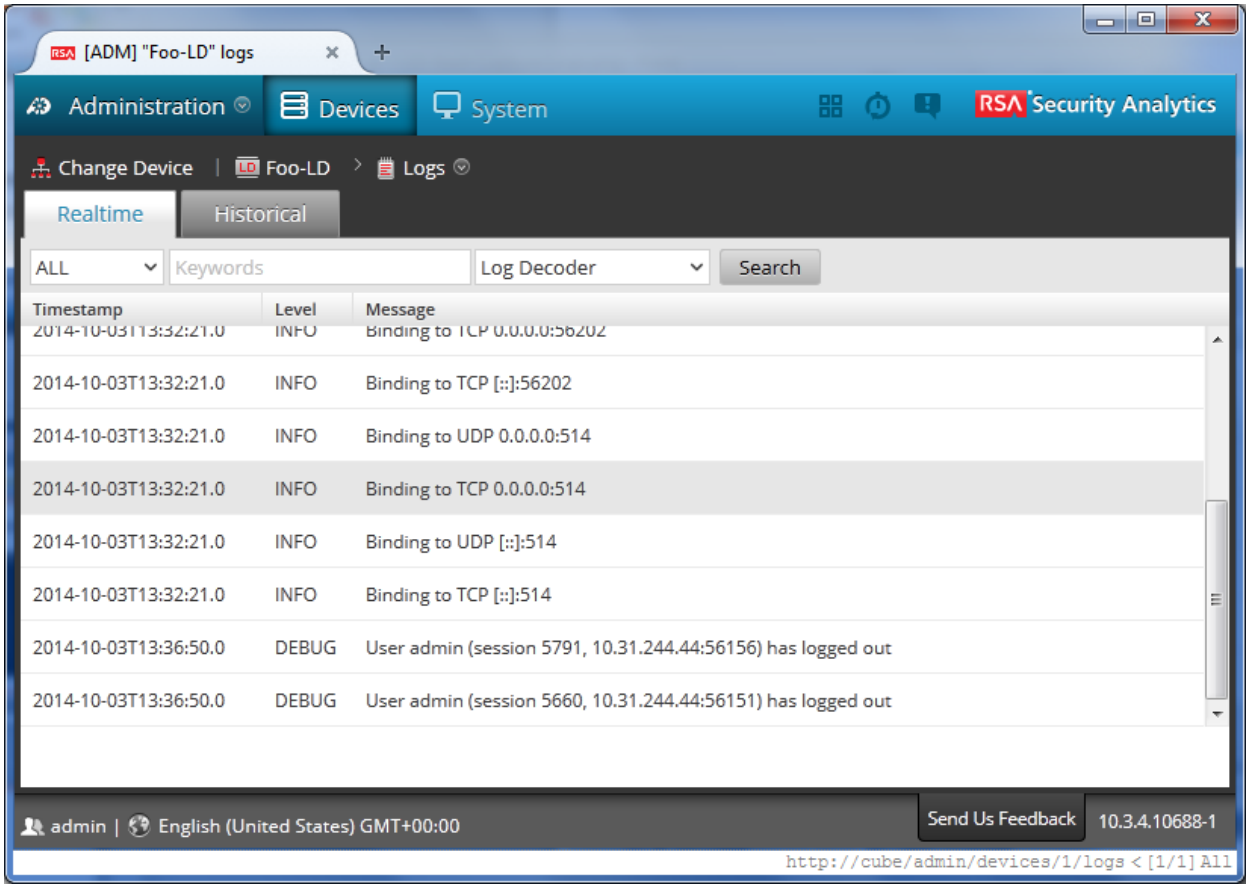
Set Capture Interface

Basic Configuration



Start Capture (LD → System)

Basic Configuration

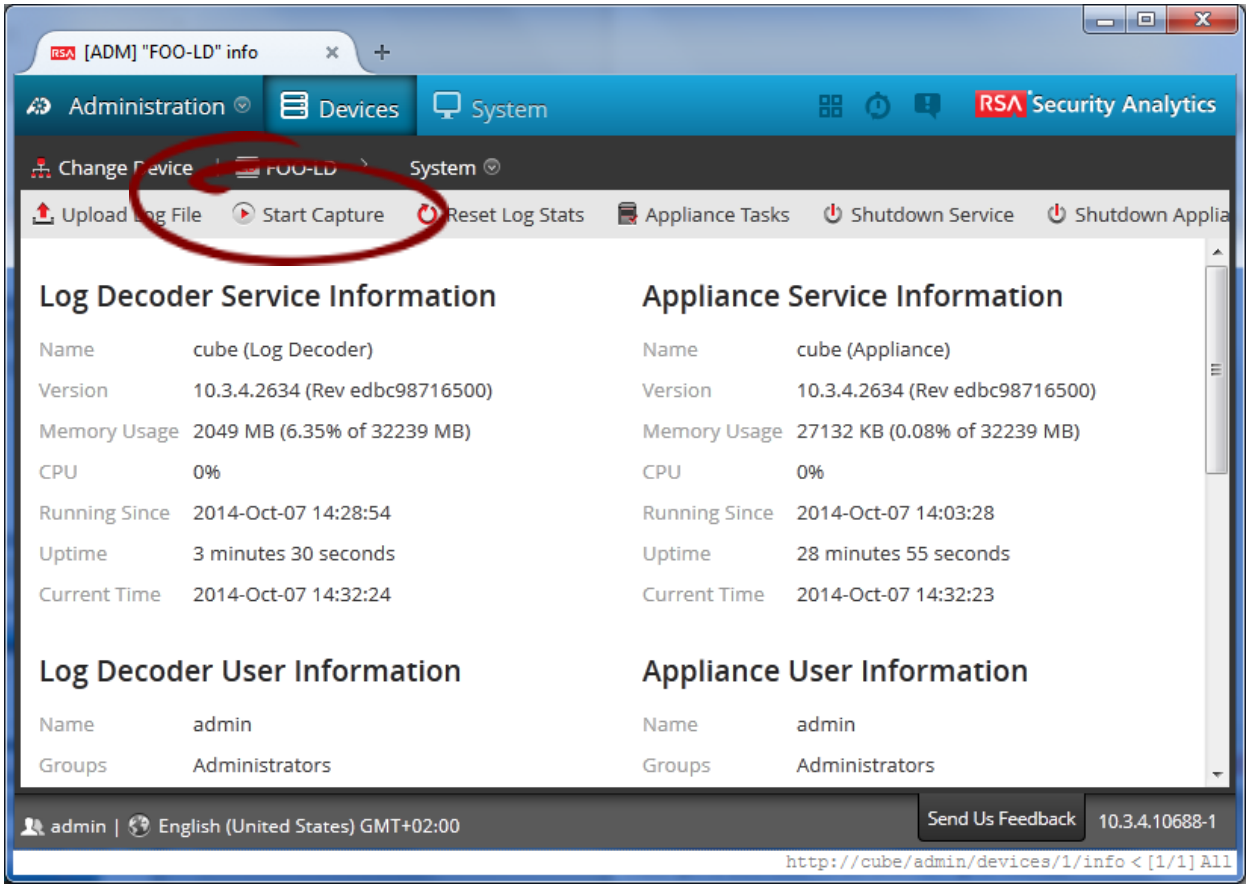


Logs you see on successful capture start (LD → Logs → Realtime/Historical)

Capture Failure

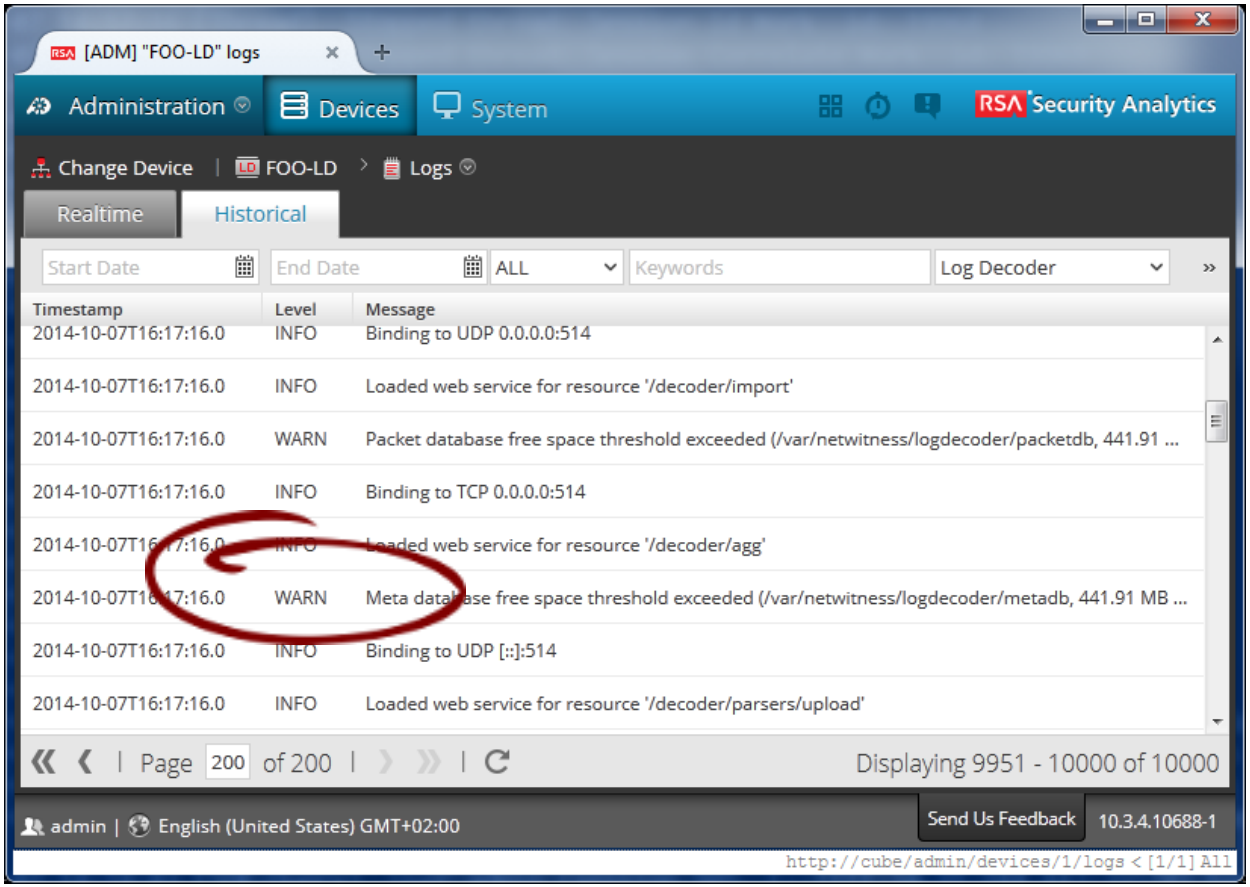
NetWitness Log Decoder

Capture Failure: Disk Space Issue



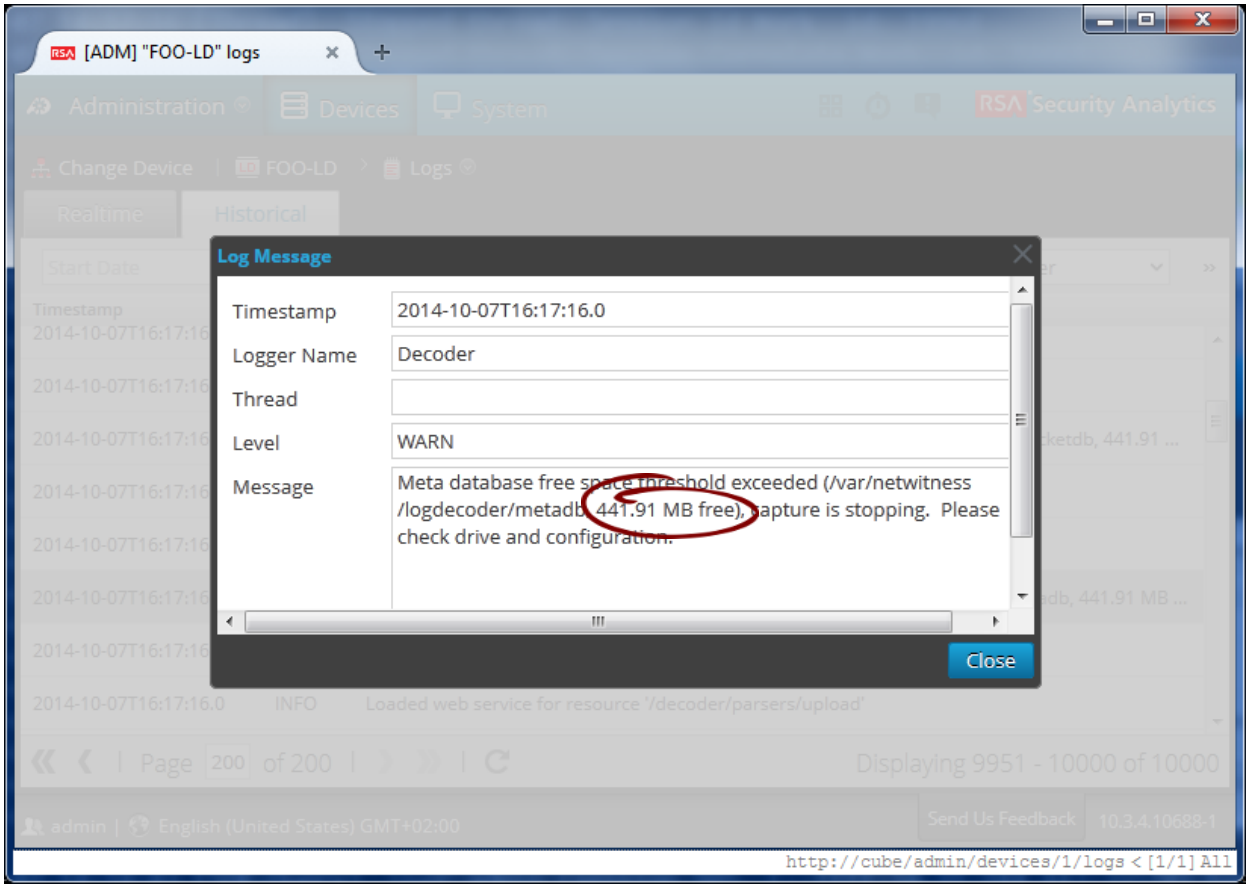
Symptom: 'Start Capture' button never turns into 'Stop Capture' button

Capture Failure: Disk Space Issue



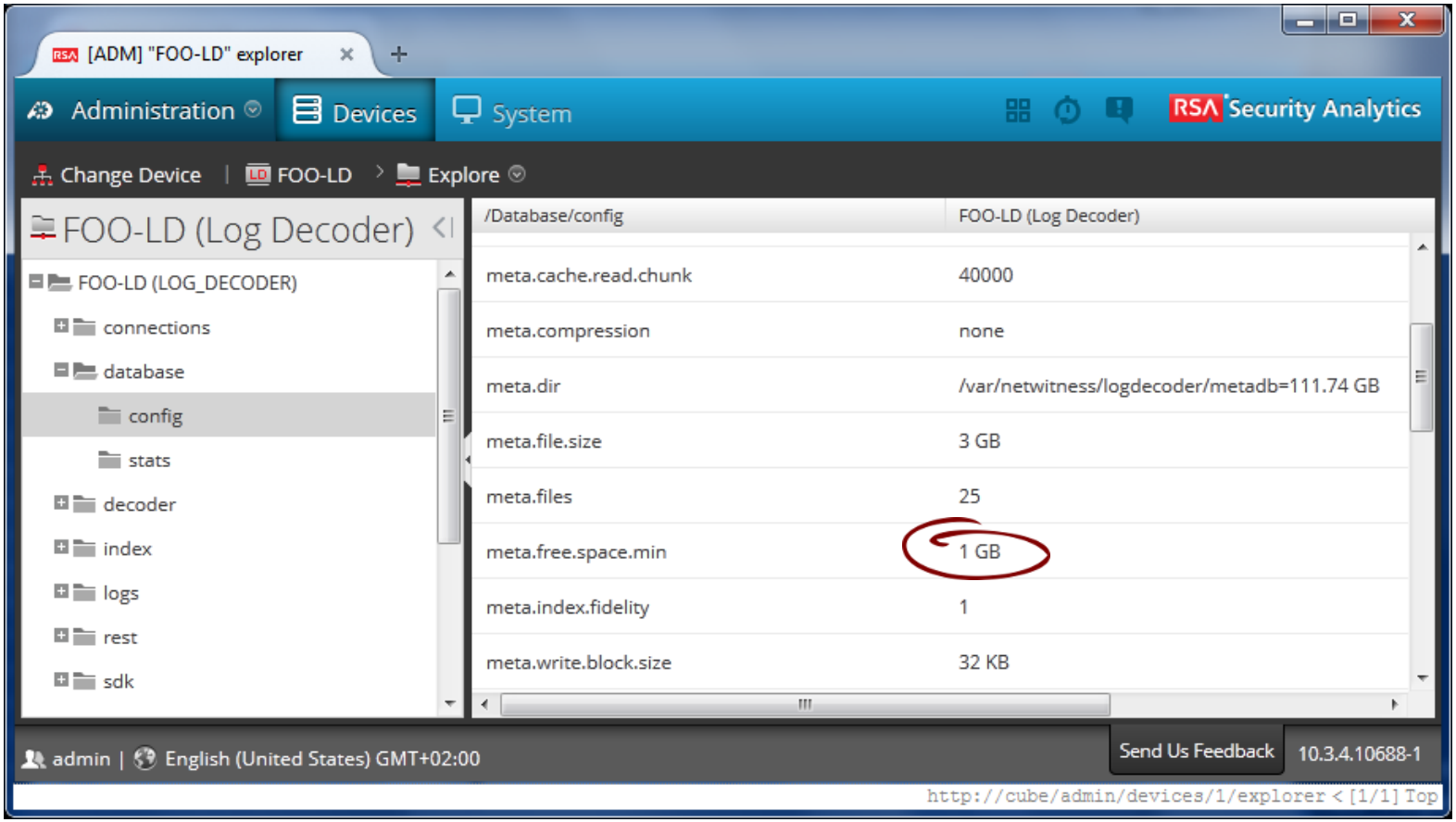
Clues in the logs (LD → Logs → Historical)

Capture Failure: Disk Space Issue



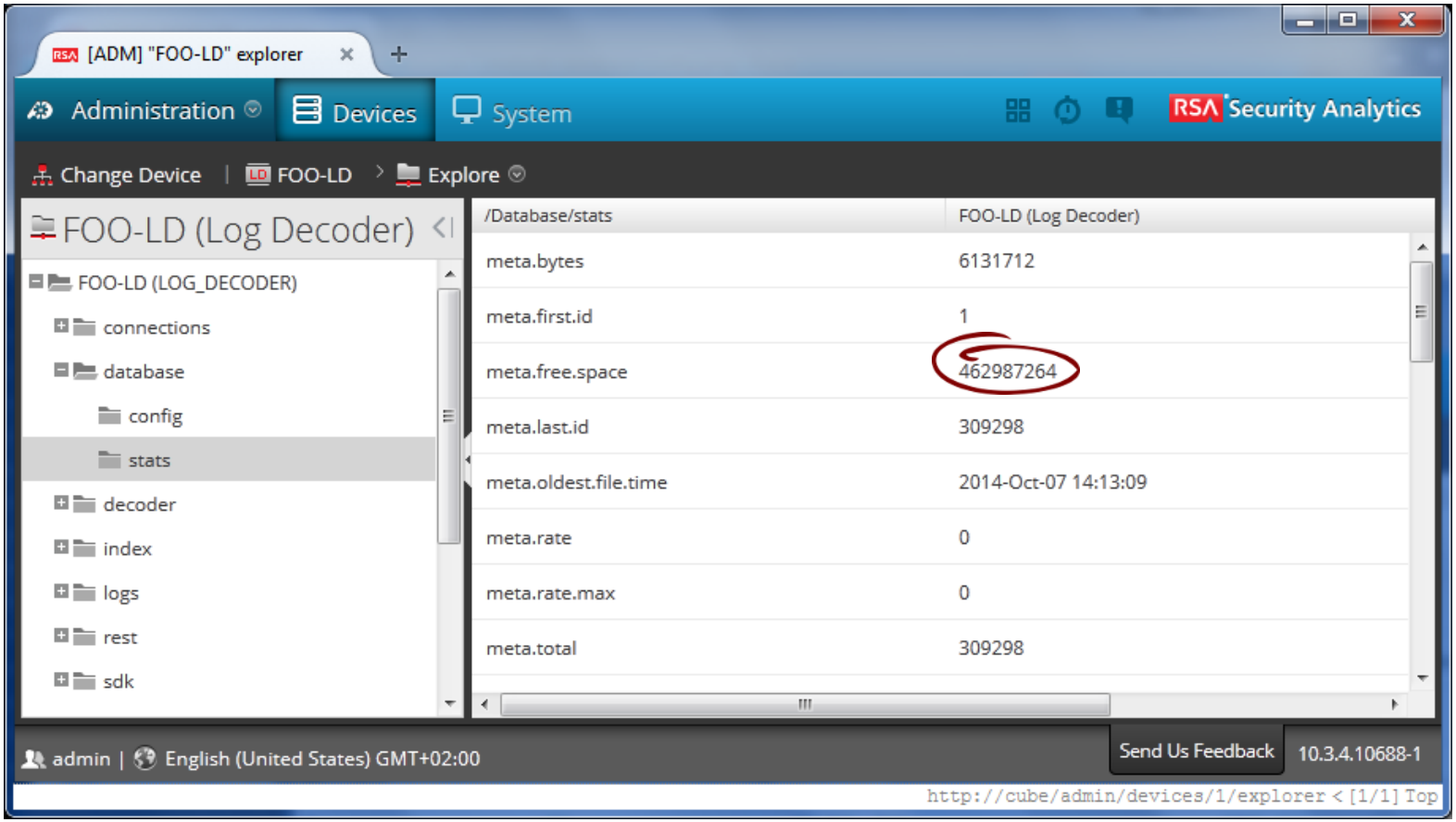
Clues in the logs: metadb is almost full

Capture Failure: Disk Space Issue



Database configuration (LD → Explore → /decoder/database/config)

Capture Failure: Disk Space Issue



Database statistics (LD → Explore → /decoder/database/stats)

Capture Failure: Disk Space Issue

These are the parameters in Log Decoder → Explore → /decoder/database/config that specify the DB locations and allocated maximum size.

meta.dir

packet.dir

session.dir

By default, they point to the following locations, respectively.

/var/netwitness/logdecoder/metadb

/var/netwitness/logdecoder/packetdb

/var/netwitness/logdecoder/sessiondb

By default, approximately 95% of the total disk space is allocated for each configuration. For example, if the metadb is on a partition of size 118 GB, then by default,

meta.dir = /var/netwitness/logdecoder/metadb=111.74 GB

Capture Failure: Disk Space Issue

These are the parameters in Log Decoder → Explore → /decoder/database/config that specify the minimum free space required for capture to start.

meta.free.space.min
packet.free.space.min
session.free.space.min

By default, the value of each is set to
(total disk space / 115) or 2 TB, whichever is less.

Example:
If metadb is on a partition of size 118 GB, then by default,
meta.free.space.min = 1 GB

Capture Failure: Disk Space Issue

The most common reason for metadb becoming full is creation of core dump files.

By default, Log Decoder's current working directory is /var/netwitness/logdecoder/metadb. As a result, any Log Decoder core dumps are written to this location.

A single core dump may be as large as 30 GB in size. A couple of core dumps may fill up metadb, if metadb's partition is small.

Log Decoder's current working directory is defined with the chdir command in /etc/init/nwlogdecoder.conf.

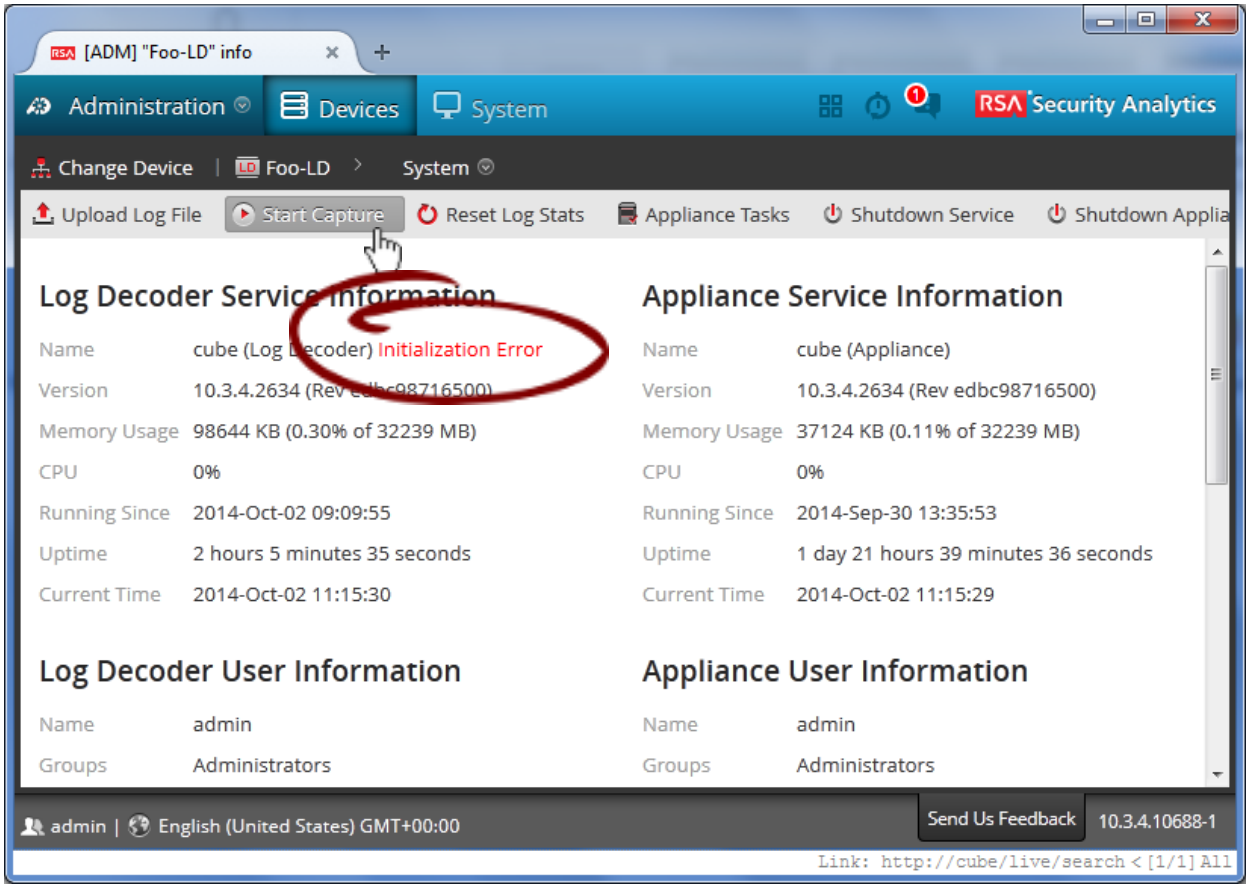
```
chdir /var/netwitness/logdecoder/metadb
```

If metadb is becoming full due to core dump files, consider changing this command to point to another location, say /var/netwitness/logdecoder/packetdb and restart nwlogdecoder. Use the command df -h to see disk space usage.

Capture Failure (continued …)

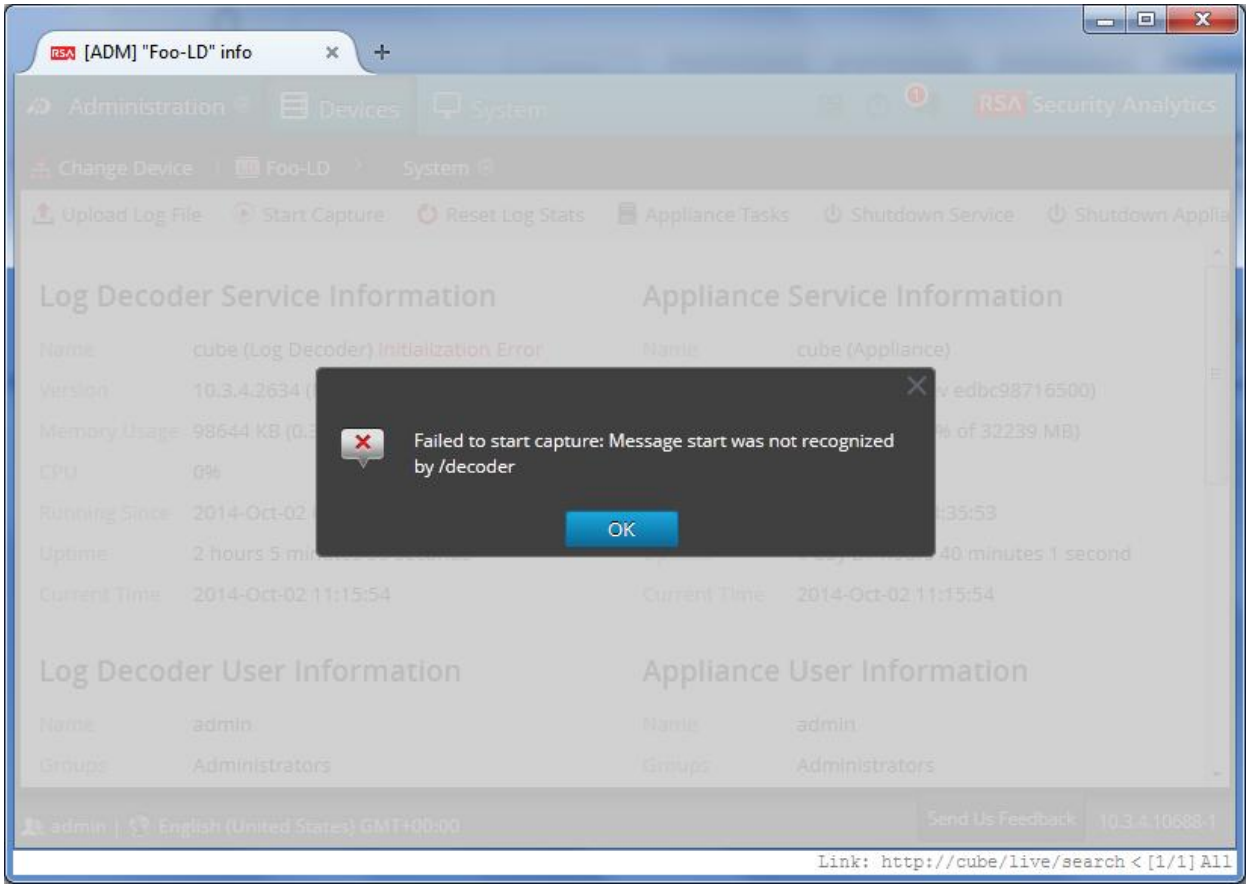
NetWitness Log Decoder

Capture Failure: Parser Error



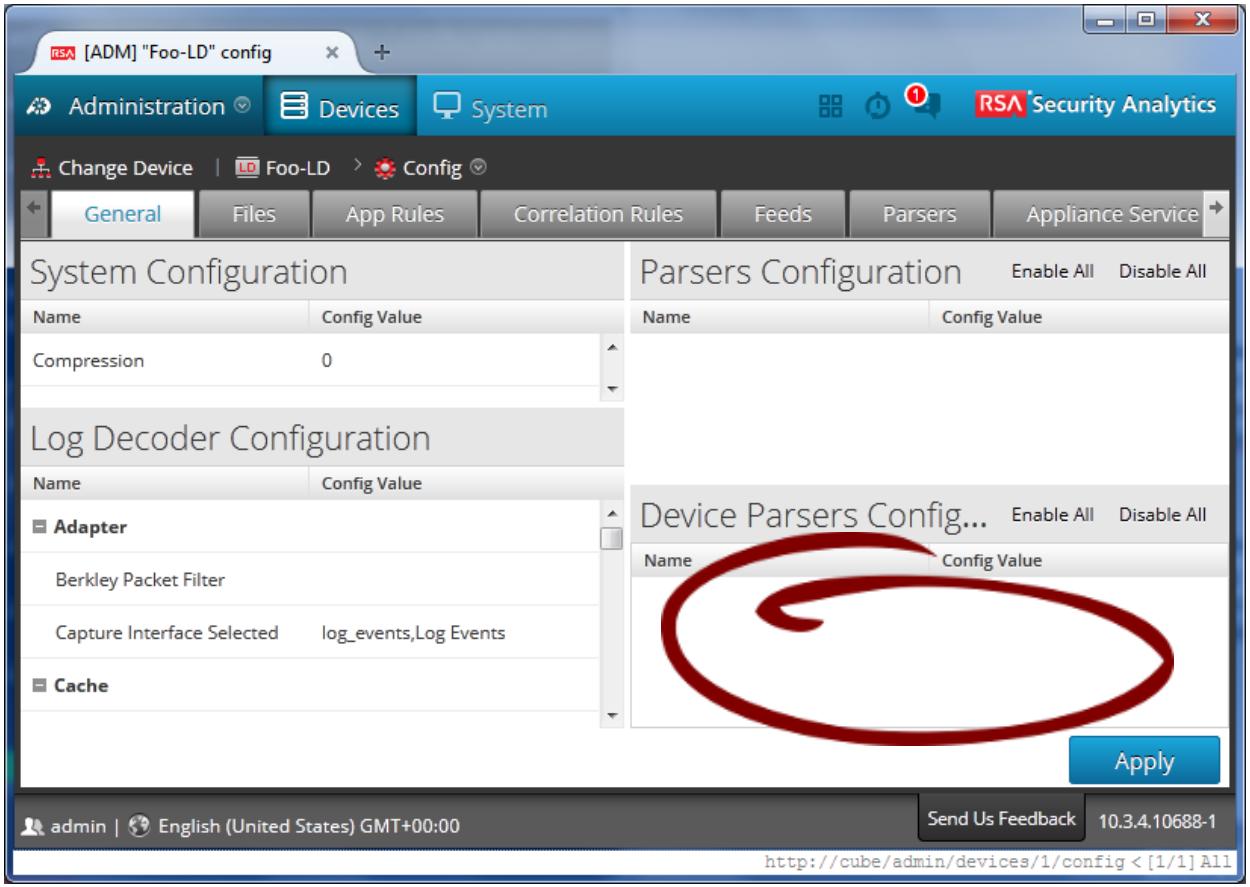
Symptom: 'Initialization Error' (LD → System)

Capture Failure: Parser Error



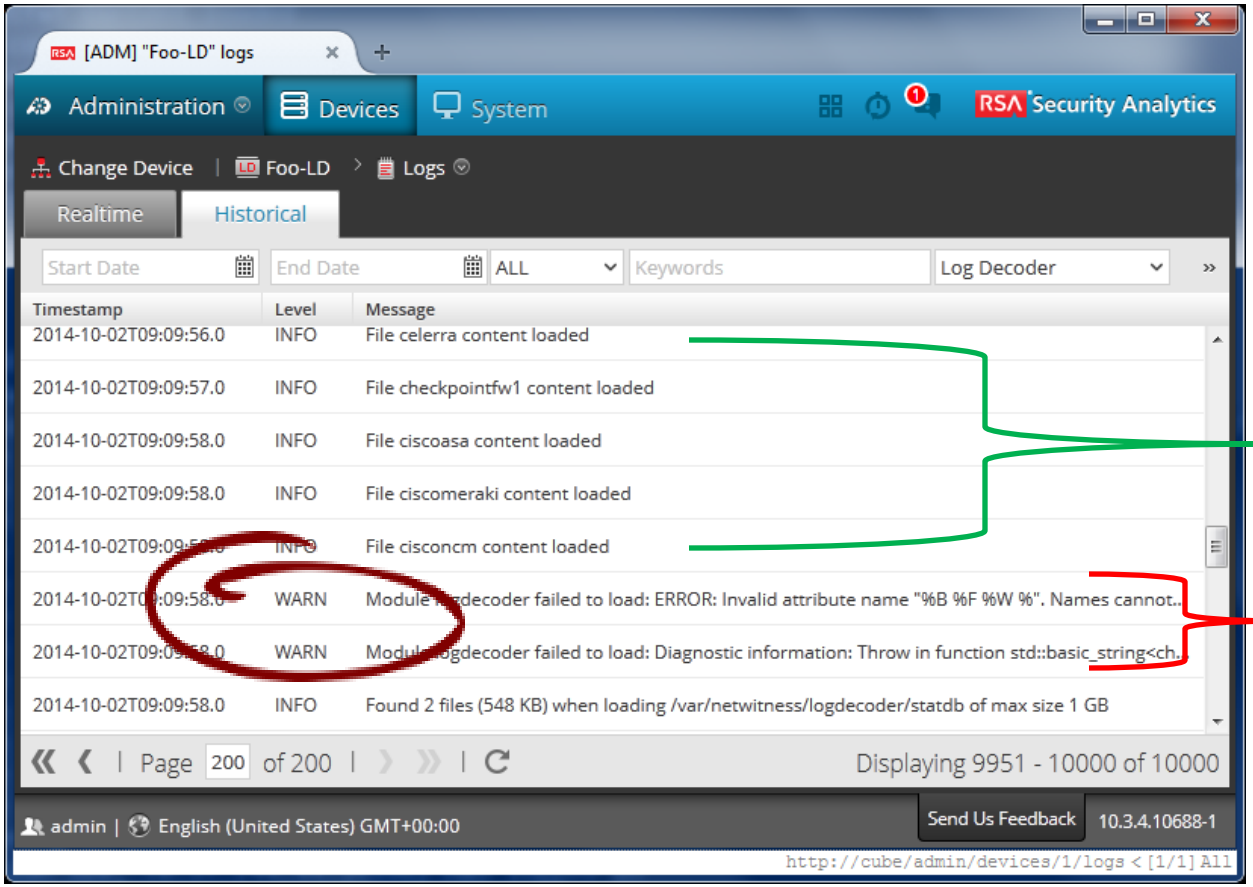
Symptom: ‘Failed to start capture’ message

Capture Failure: Parser Error



Symptom: No parsers listed

Capture Failure: Parser Error



What failed?
The parser that was about to be loaded after 'cisconcm'.

Why did it fail?
Something went wrong while reading "%B %F %W …".

Clues in the logs (LD → Logs → Historical)

Capture Failure: Parser Error

The /etc/netwitness/ng/etc/devices directory contains all the device parsers.

grep's -A *N* option allows us to display *N* number of lines after each match.

```
# ls /etc/netwitness/ng/envision/etc/devices | grep -A5 cisconcm
cisconcm    <-- We know this one could be loaded
cisconxos   <-- Did this one fail?
ciscopix
ciscorouter
ciscosecagent
ciscosecureacs
```

(The above command is run on Log Decoder.)

We know that the failure occurred after loading 'cisconcm'.

Did 'cisconxos' fail?

Capture Failure: Parser Error

```
# ls /etc/netwitness/ng/envision/etc/devices | grep -A5 cisconcm
cisconcm
cisconxos
ciscopix
ciscorouter
ciscosecagent
ciscosecureacs
```

(The above command is run on Log Decoder.)

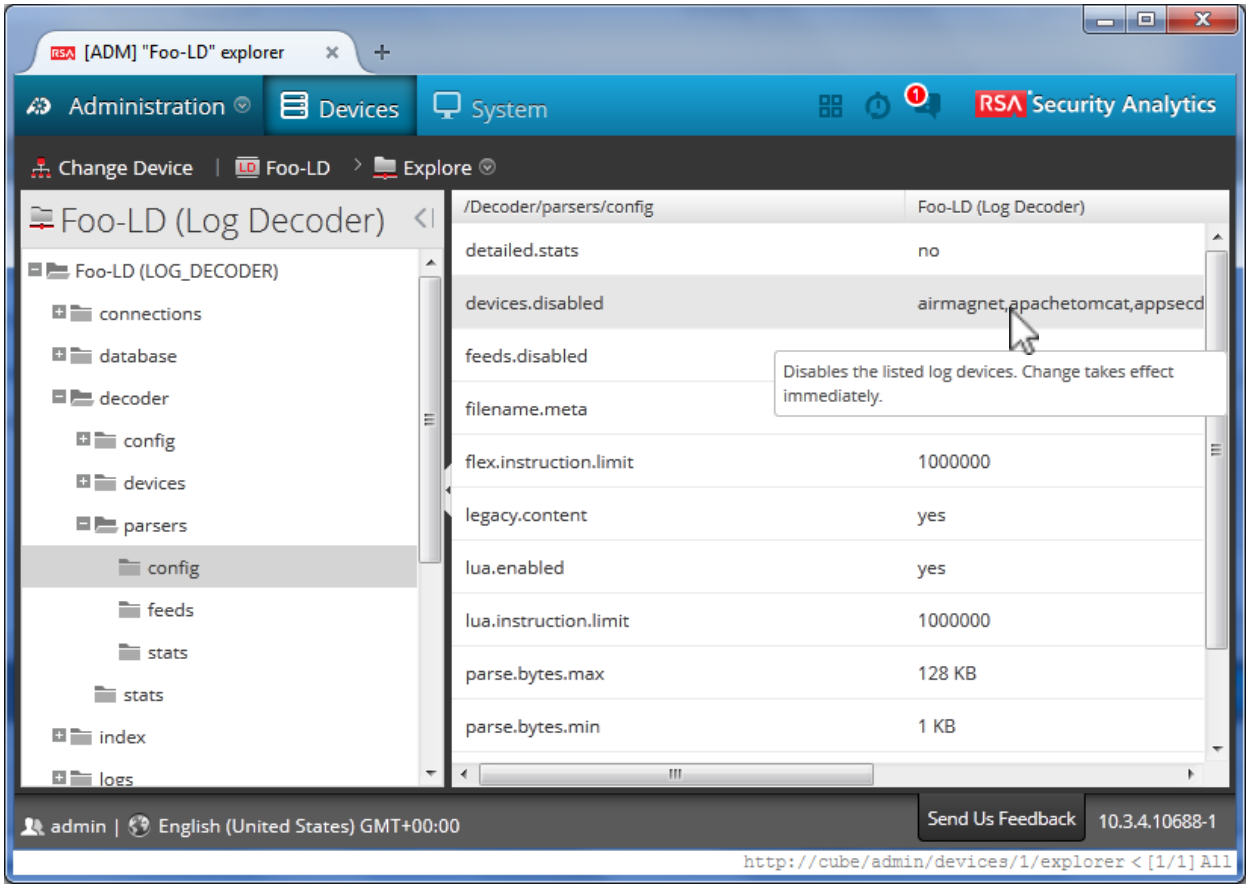
Did 'cisconxos' fail?

We need to consider that disabled parsers are not loaded, so disabled parsers cannot fail.

Is 'cisconxos' a disabled parser?

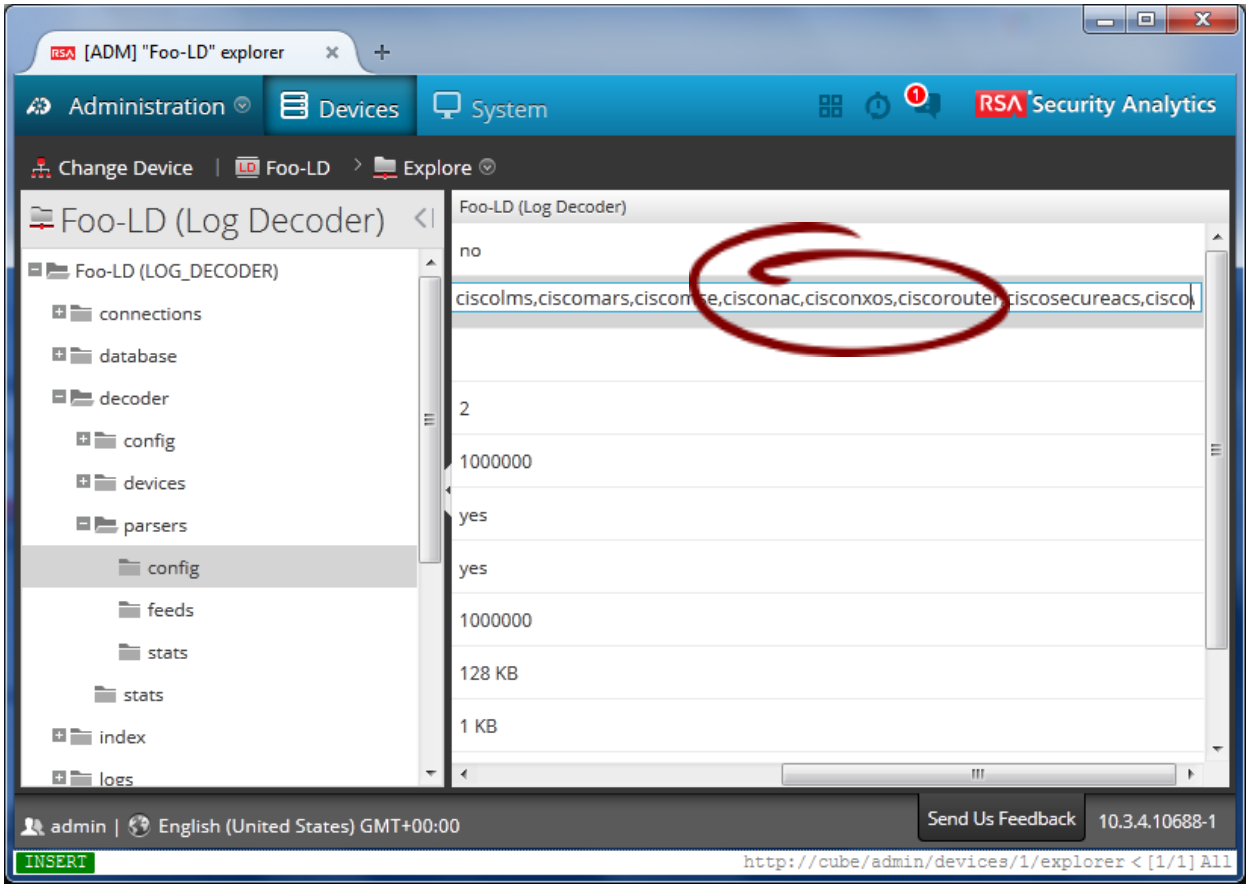
We need to check the REST resource /decoder/parsers/config/devices.disabled to find out.

Capture Failure: Parser Error



Is ‘cisconxos' a disabled parser?

Capture Failure: Parser Error



'cisconxos' is disabled but 'ciscopix' isn't

Capture Failure: Parser Error

A nifty shell command to check if a device parser is disabled.

These commands are run on Log Decoder.

```
# curl -su admin:netwitness http://localhost:50102/decoder/parsers/config/devices.disabled
| tr , "\n" | grep cisconxos
cisconxos
#
```

(The above command is run on Log Decoder.)

'cisconxos' exists in /decoder/parsers/config/devices.disabled, so it is disabled.

```
# curl -su admin:netwitness http://localhost:50102/decoder/parsers/config/devices.disabled
| tr , "\n" | grep ciscopix
#
```

(The above command is run on Log Decoder.)

'ciscopix' does not exist in /decoder/parsers/config/devices.disabled, so it is enabled.

Capture Failure: Parser Error

```
# ls /etc/netwitness/ng/envision/etc/devices | grep -A5 cisconcm
cisconcm
cisconxos
ciscopix
ciscorouter
ciscosecagent
ciscosecureacs
```

This one could be loaded

This one is disabled, so this cannot fail

This one is enabled, so this must have failed

Did 'cisconxos' fail?

No, because 'cisconxos' is disabled.

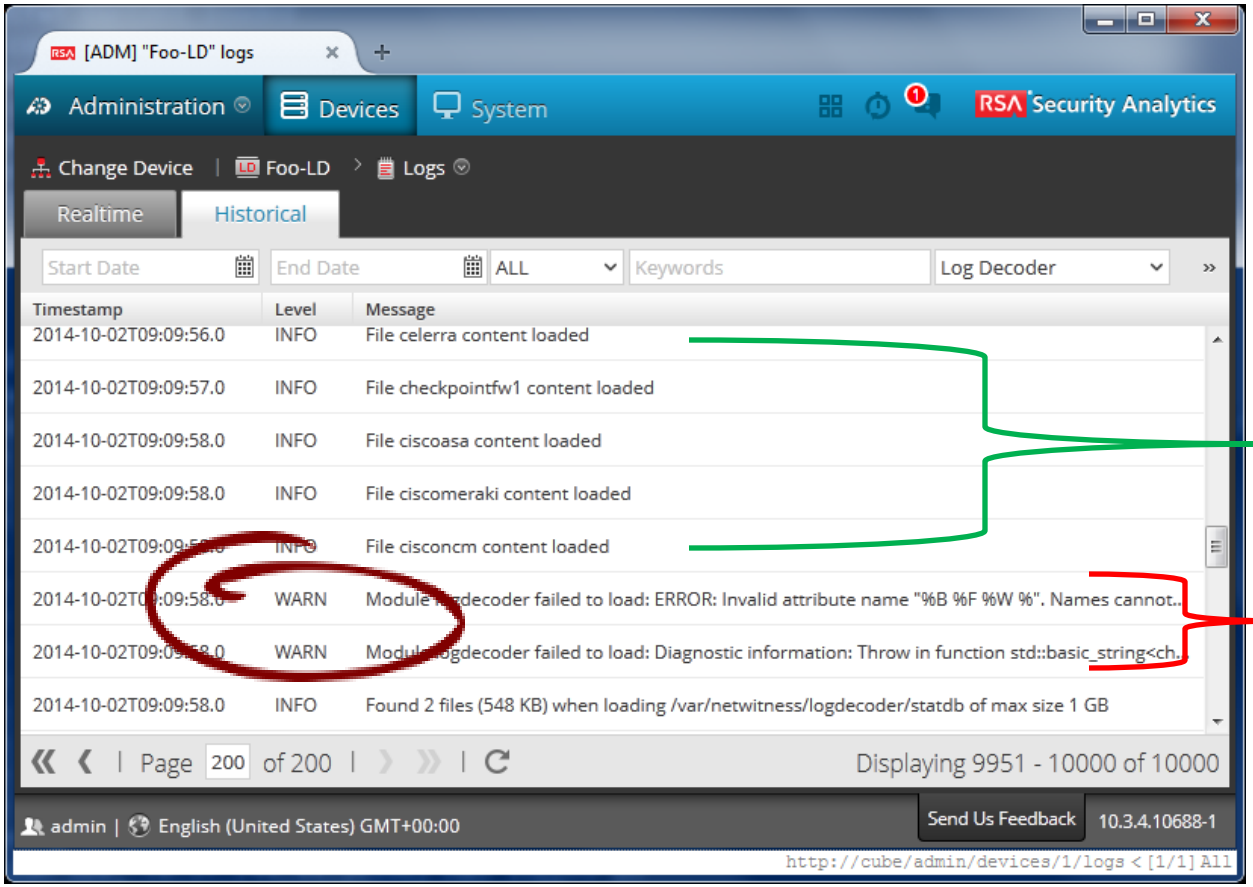
Did 'ciscopix' fail?

Yes, because 'ciscopix' is enabled, and it is the first enabled parser that comes after 'cisconcm', in alphabetical order.

What went wrong in the 'ciscopix' parser?

We need to check the logs and the parser to find out.

Capture Failure: Parser Error

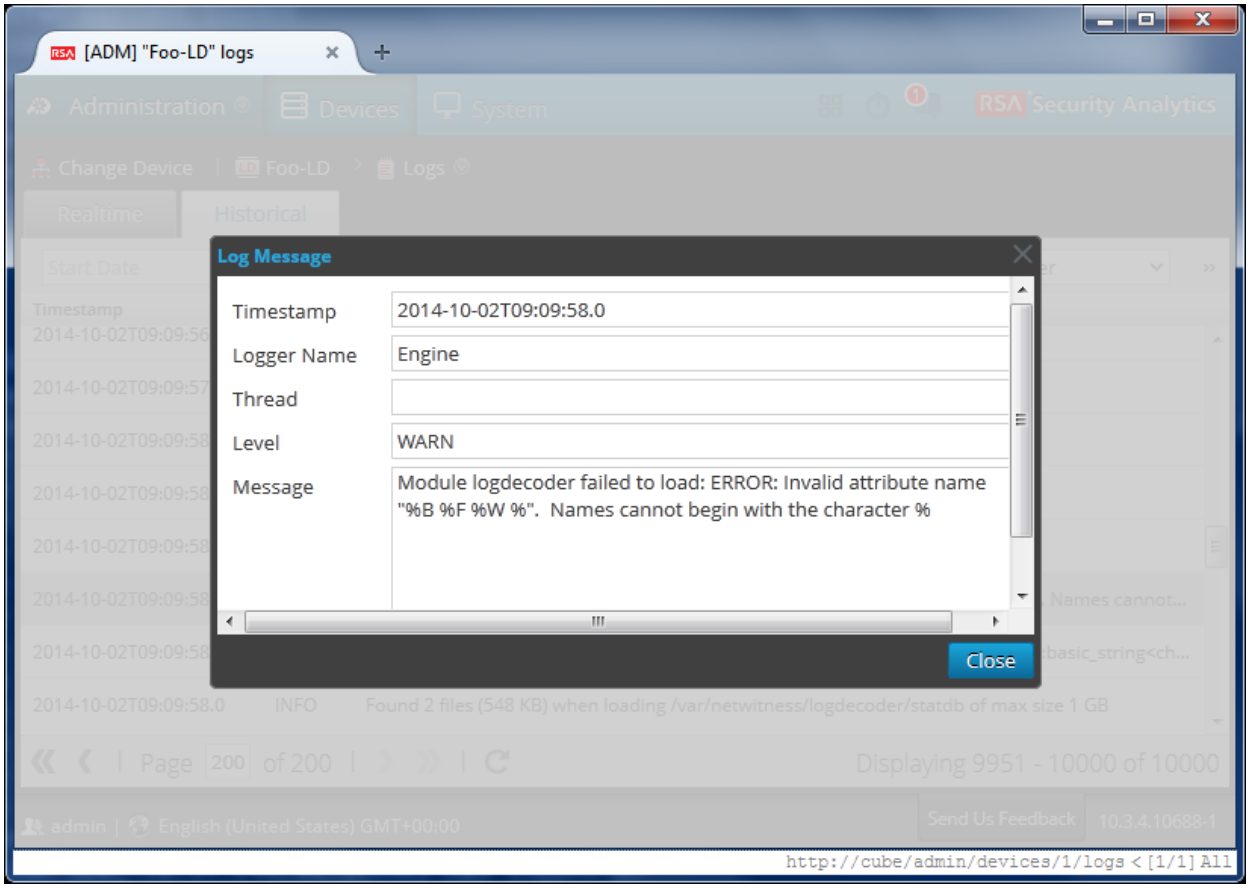


What failed?
The parser that was about to be loaded after 'cisconcm'.

Why did it fail?
Something went wrong while reading "%B %F %W …".

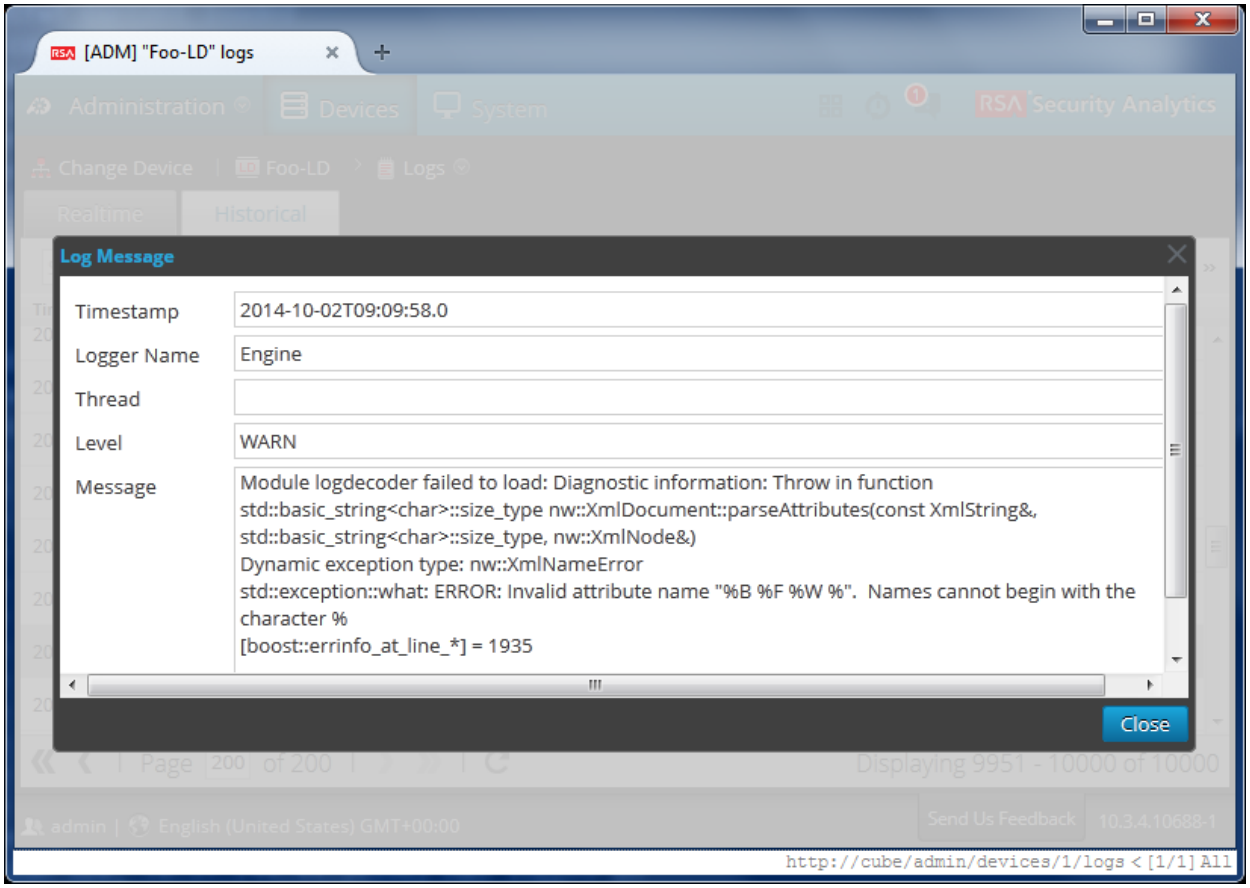
Clues in the logs (LD → Logs → Historical)

Capture Failure: Parser Error



Clues in the logs: '%B %F %W' are time format specifiers used in the device parsers

Capture Failure: Parser Error



Clues in the logs: XML related error. Device parsers are written in XML.

Capture Failure: Parser Error

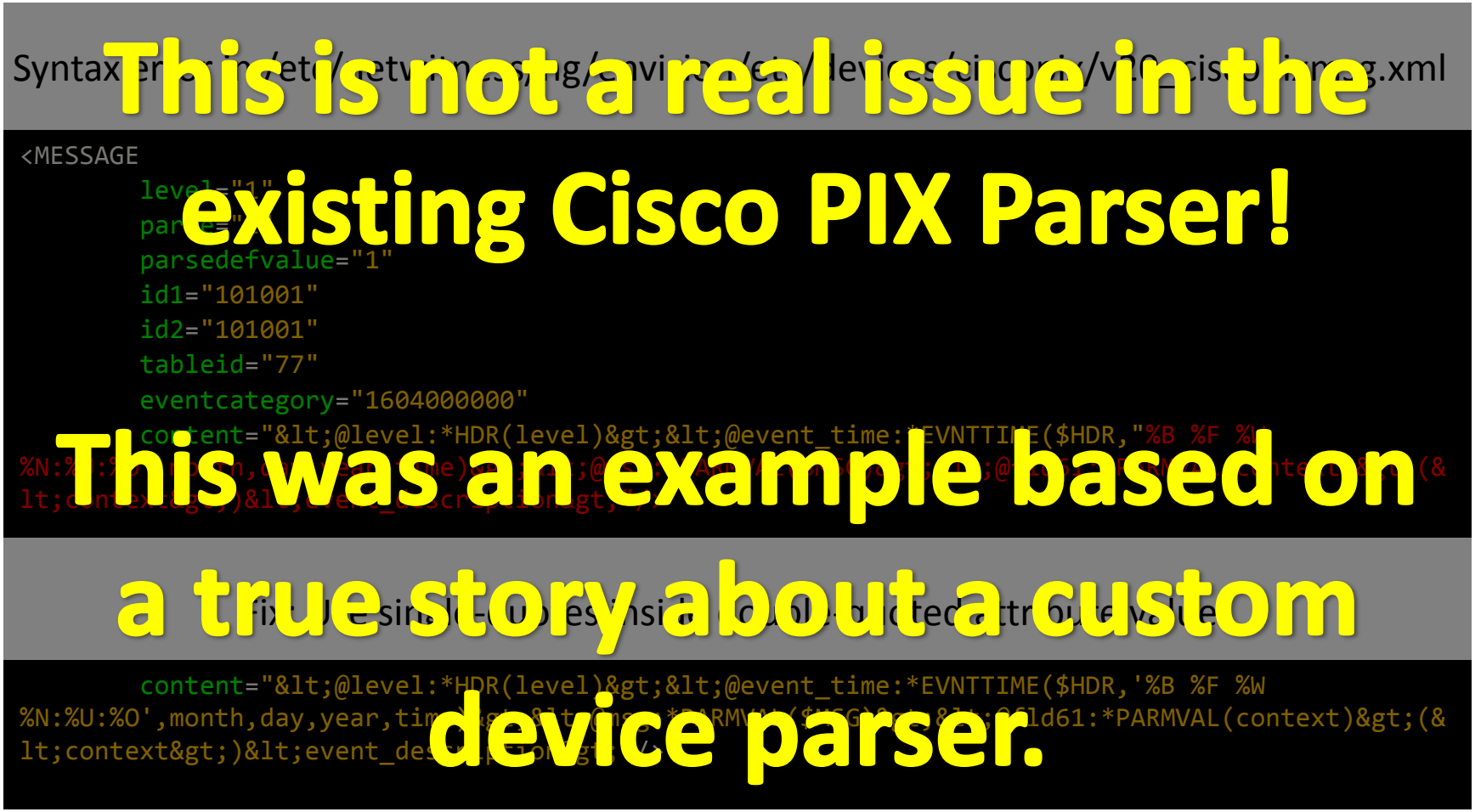
Syntax error in /etc/netwitness/ng/envision/etc/devices/ciscopix/v20_ciscopixmsg.xml

```
<MESSAGE
        level="1"
        parse="1"
        parsedefvalue="1"
        id1="101001"
        id2="101001"
        tableid="77"
        eventcategory="1604000000"
        content="&lt;@level:*HDR(level)&gt;&lt;@event_time:*EVNTTIME($HDR,"%B %F %W
%N:%U:%O",month,day,year,time)&gt;&lt;@msg:*PARMVAL($MSG)&gt;&lt;@fld61:*PARMVAL(context)&gt;(&
lt;context&gt;)&lt;event_description&gt;"/>
```

Fix: Use single-quotes inside double-quoted attribute values

```
        content="&lt;@level:*HDR(level)&gt;&lt;@event_time:*EVNTTIME($HDR,'%B %F %W
%N:%U:%O',month,day,year,time)&gt;&lt;@msg:*PARMVAL($MSG)&gt;&lt;@fld61:*PARMVAL(context)&gt;(&
lt;context&gt;)&lt;event_description&gt;"/>
```

Capture Failure: Parser Error



Log Ingestion

NetWitness Log Decoder

Log Ingestion

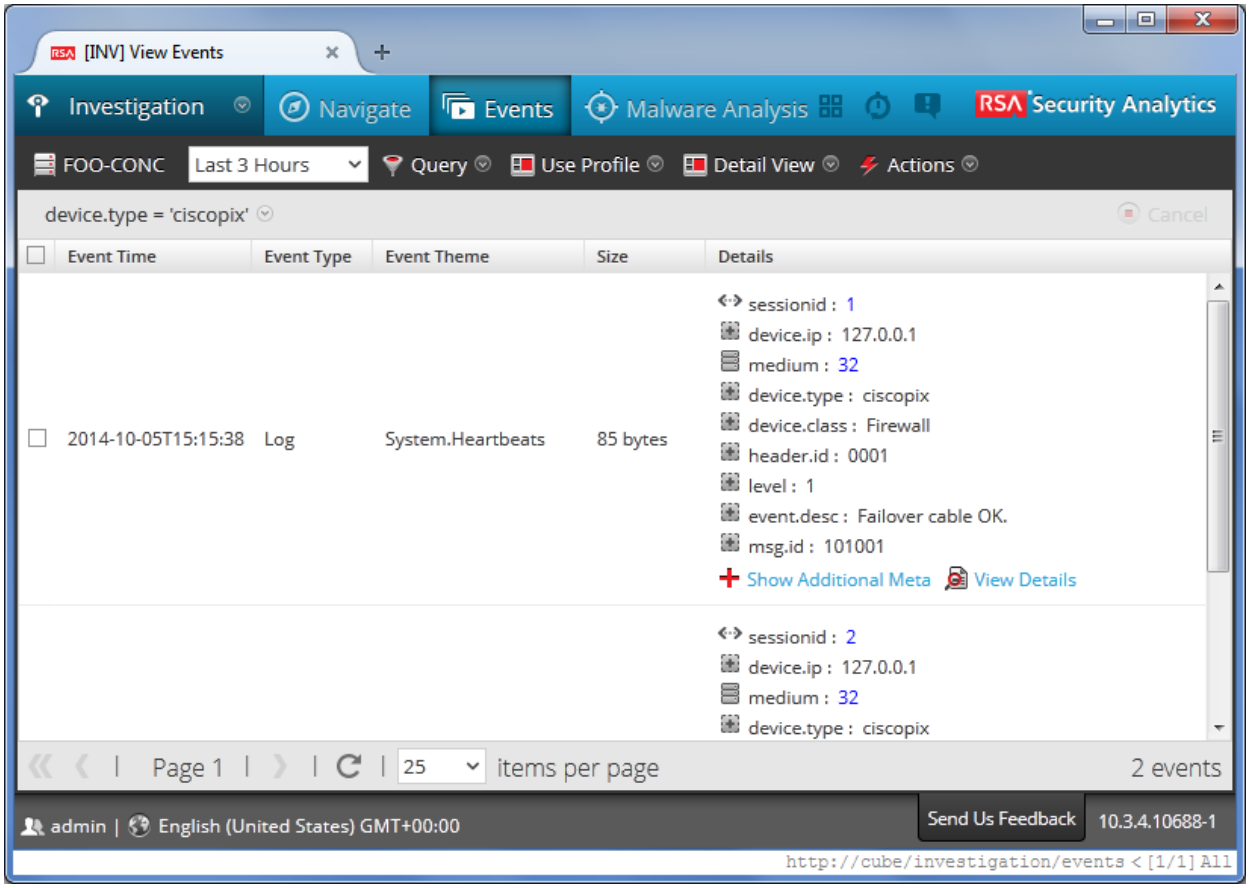
Feed some logs to Log Decoder.

In Bash, `/dev/<protocol>/<host>/<port>` is a special magic device that represents a local or remote host and port. Redirecting any output to it leads to establishing a connection with the `<host>:<port>` and sending the output as payload to it.

We'll use this feature to feed some logs to Log Decoder and demonstrate log ingestion concepts. We are not feeding logs from Log Collector to keep the demonstration simple and to help us type the logs by hand.

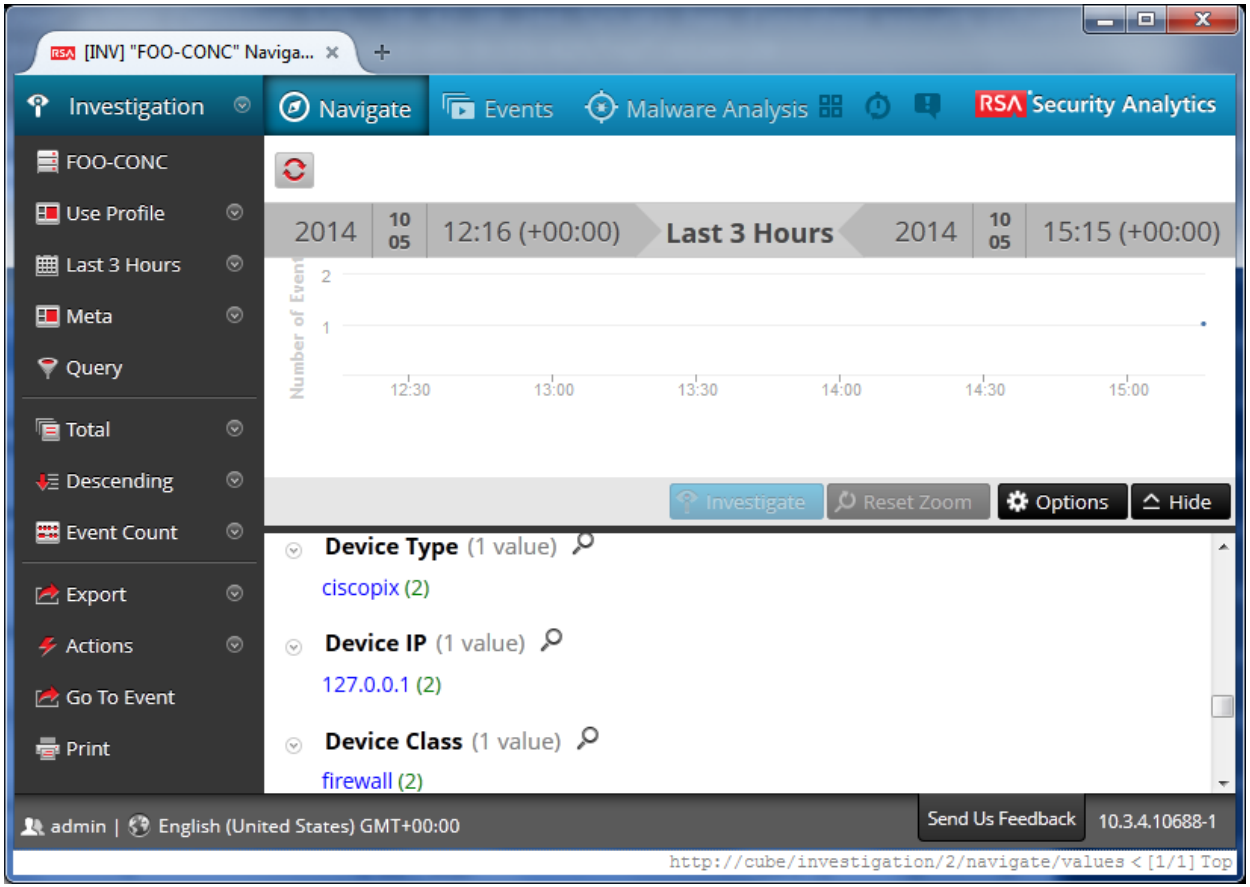
```
# echo "<1> %PIX-1-101001: (PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "<1> %PIX-1-101001: (PRIORITY) Bad failover cable." > /dev/tcp/127.0.0.1/514
```

Log Ingestion



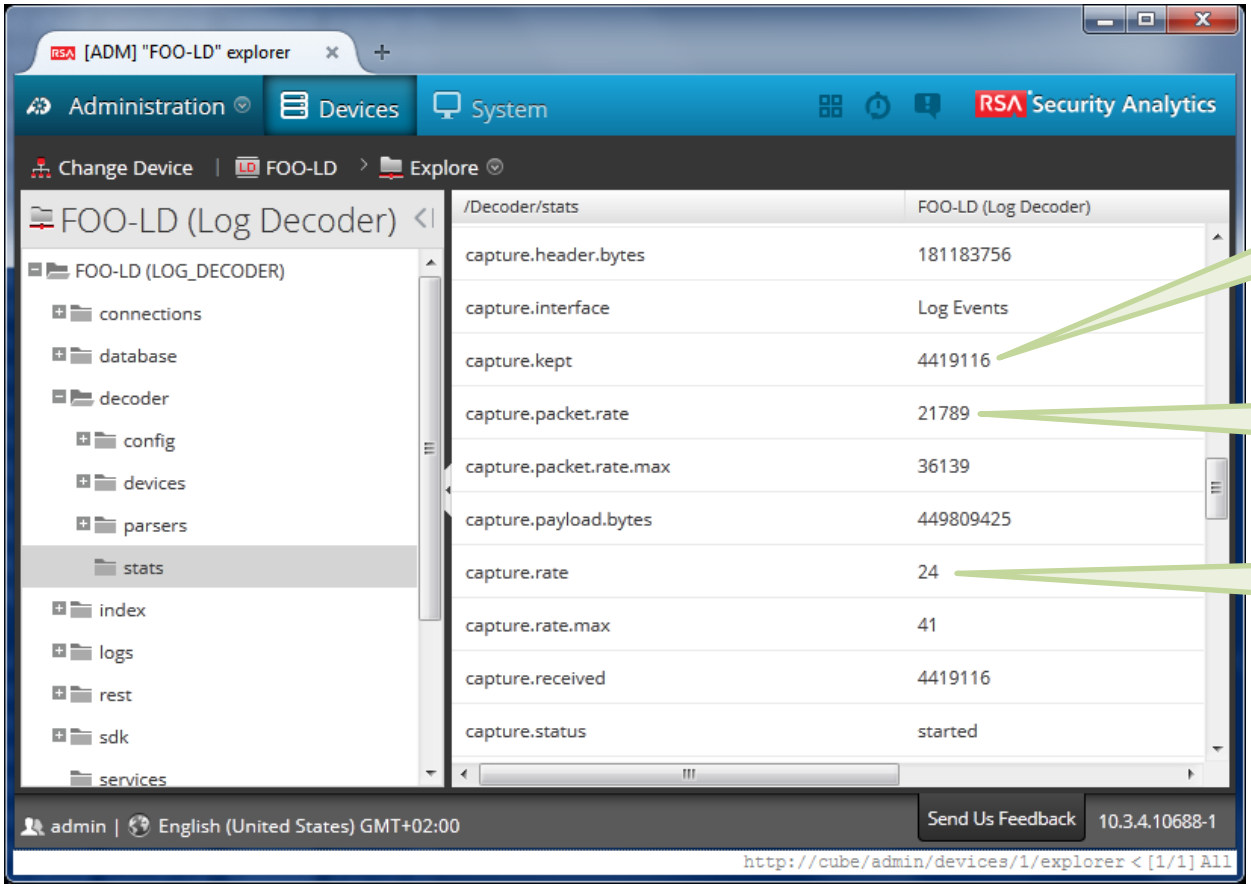
We can see the logs (Concentrator → Investigation → Events)

Log Ingestion



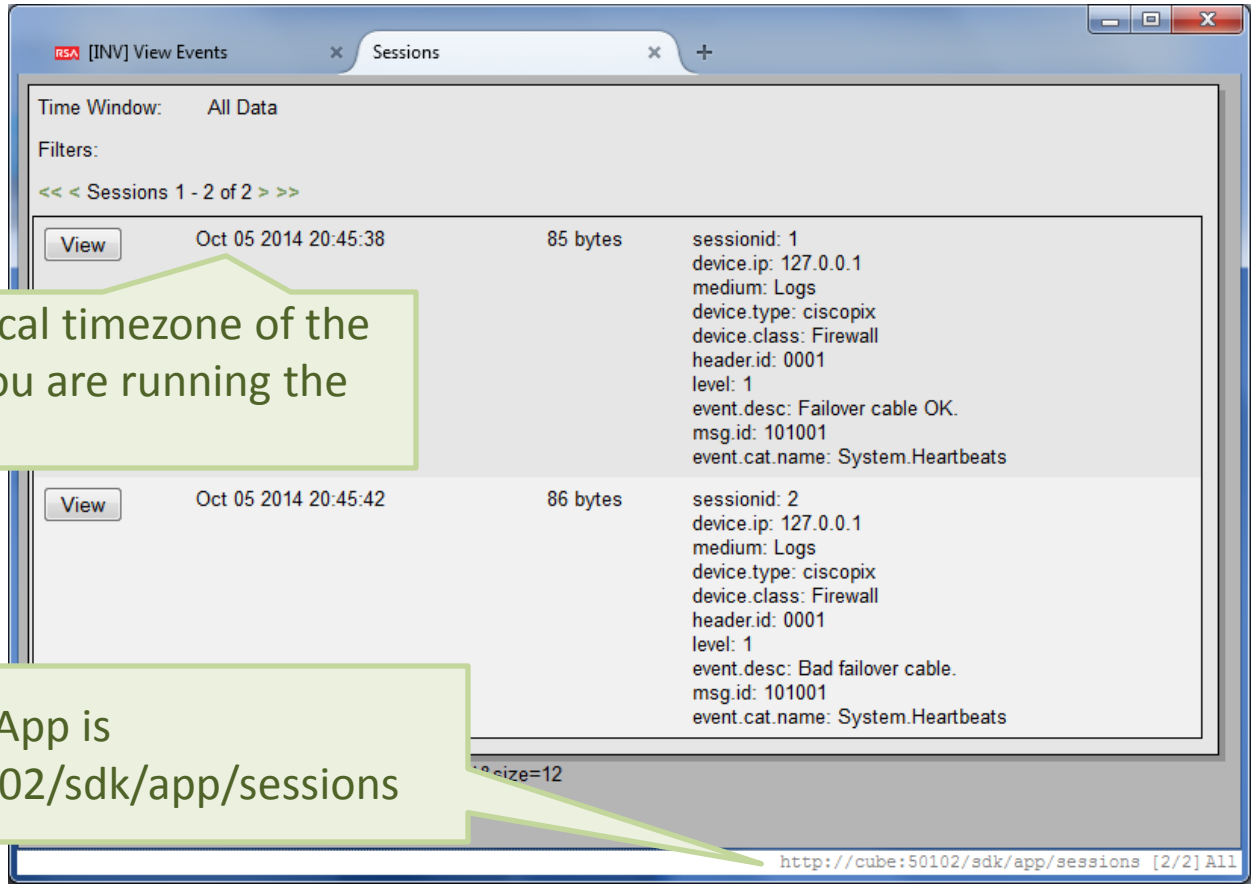
Meta values and their counts (Concentrator → Investigation → Navigate)

Log Ingestion



LD → Explore → /decoder/stats

Log Ingestion

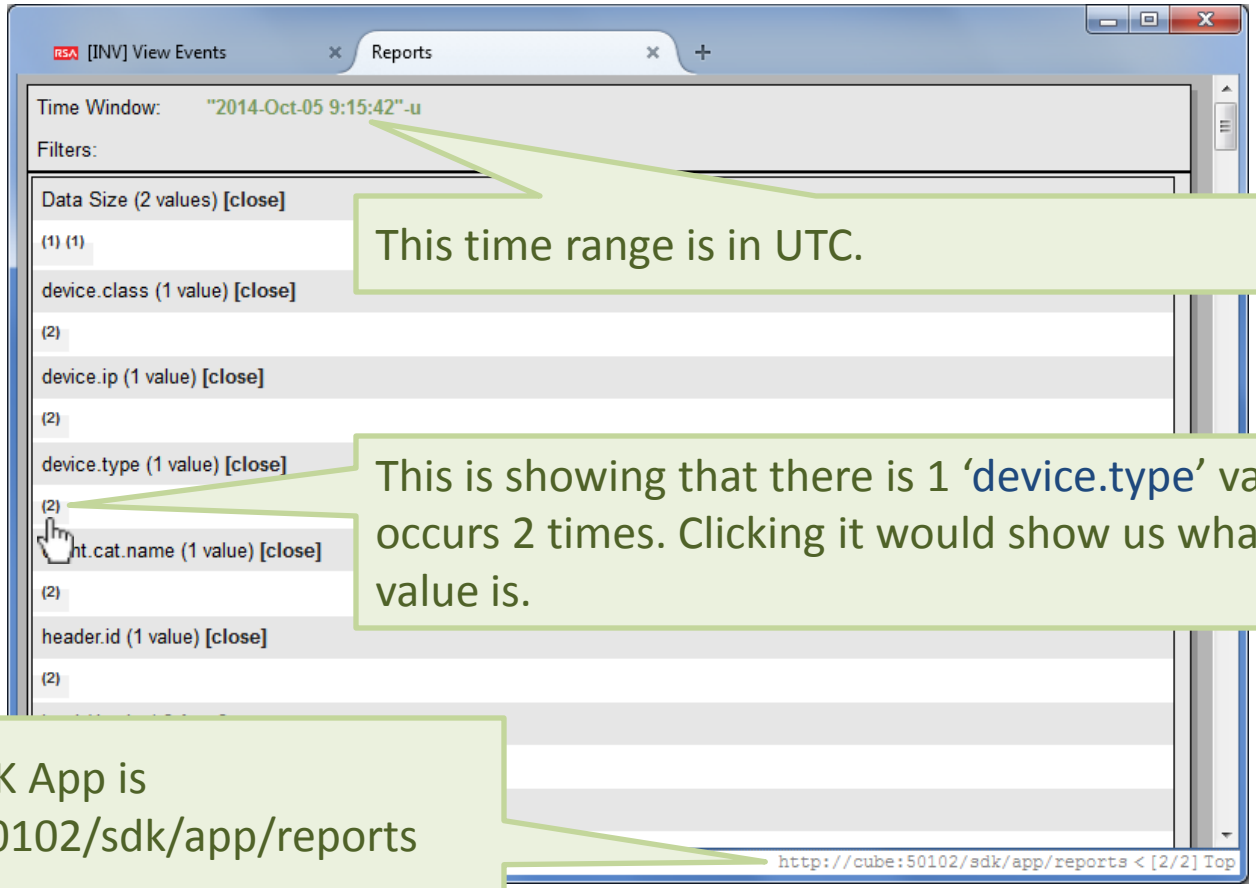


This time is in local timezone of the system where you are running the browser.

URL to this SDK App is http://<LD>:50102/sdk/app/sessions

The logs can also be seen at http://<LD>:50102/sdk/app/sessions

Log Ingestion



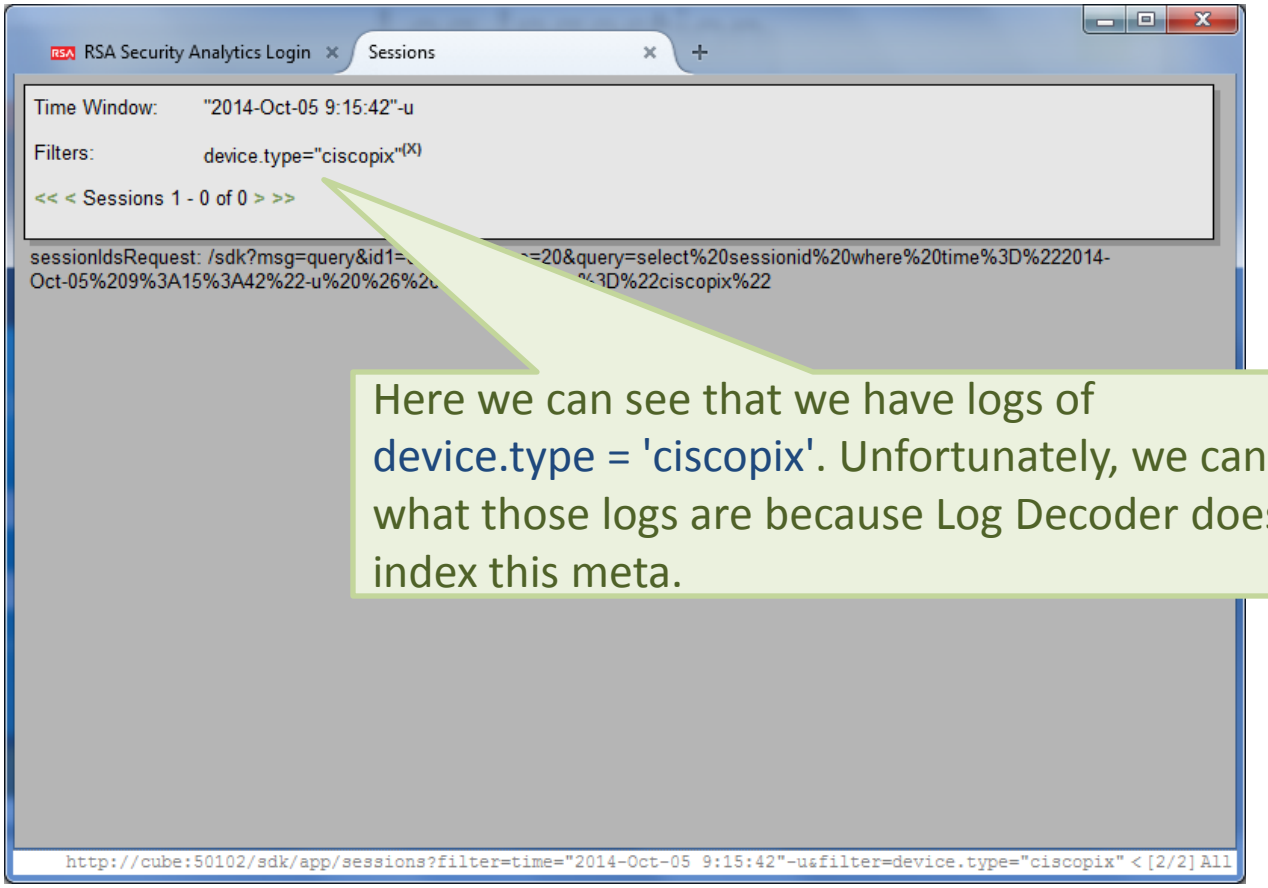
This time range is in UTC.

This is showing that there is 1 'device.type' value that occurs 2 times. Clicking it would show us what that value is.

URL to this SDK App is http://<LD>:50102/sdk/app/reports

Meta value counts can also be seen at http://<LD>:50102/sdk/app/reports

Log Ingestion



No results because Log Decoder does not index this meta

Log Ingestion

If we see /etc/netwitness/ng/index-logdecoder.xml, we'll find that only 'time' is indexed, i.e. it is defined with level="IndexValues". No other metas are indexed, i.e. they are defined with level="IndexNone".

This can be overridden in /etc/netwitness/ng/index-logdecoder-custom.xml but it is not recommended because Log Decoder is not meant to perform indexing on other metas. Concentrator is meant to do that.

Log Ingestion Issues

NetWitness Log Decoder

Log Ingestion Issues

While performing investigation on Concentrator, if desired logs are not available, first rule out parsing failure.

Log Decoder may have failed to parse the log, so the 'device.type' of the logs may be 'unknown'. Query for device.type = 'unknown' and see if the logs can be found.

If the desired logs are found with device.type = 'unknown' meta then it is not a log ingestion issue because the log is available. Instead, it is a log parsing issue.

Log Ingestion Issues

If it is confirmed that the logs cannot be found in Concentrator at all (even with device.type = 'unknown' filter), then check the following.

1. Can the logs be found in Log Decoder's SDK sessions app (http://<LD>:50102/sdk/app/sessions). Note: <LD> must be replaced with the IP address of the Log Decoder.
2. If there are too many logs in Log Decoder's SDK sessions app, they can be filtered by time range. The time range must be specified in UTC. An example: http://<LD>:50102/sdk/app/sessions?filter=time="2014-Oct-05 9:15:45"-"2014-Oct-05 17:15:46"
3. If the logs are available in Log Decoder's SDK sessions app, troubleshoot Concentrator. Is Concentrator aggregating?
4. If the logs are not available in Log Decoder's SDK sessions app, check if Log Decoder is capturing. Is TCP port 514 open? Is UDP port 514 open?
5. If Log Decoder is capturing, then check if any logs are arriving on port 514?

Log Ingestion Issues

Test connectivity to Log Decoder port 514.

In Bash, and any POSIX shell, `:` is a null command. It does nothing. Therefore, we can use the `:` command instead of the echo command if we just want to test connectivity to TCP port 514 of Log Decoder.

```
# : > /dev/tcp/127.0.0.1/514
-bash: connect: Connection refused
-bash: /dev/tcp/127.0.0.1/514: Connection refused
```

(The above command is run on Log Decoder.)

If connection could not be established, then error messages are displayed.

```
# : > /dev/tcp/127.0.0.1/514
#
```

(The above command is run on Log Decoder.)

If connection could be established, then there is no output.

Log Ingestion Issues

If Log Decoder is capturing, and yet the desired logs do not appear in Log Decoder's SDK sessions app (http://<LD>:50102/sdk/app/sessions) or in Concentrator → Investigation → Events page, then we need to find out if any logs are coming to Log Decoder.

The easiest way is to run tcpdump to capture all traffic on port 514.

```
# tcpdump -i any -w 514.pcap "port 514"
```

(The above command is run on Log Decoder.)

Run the above command on Log Decoder. Press 'Ctrl + C' when you think the Log Decoder should have received the desired logs.

All traffic to port 514 (TCP or UDP) would now be captured in the file 514.pcap.

This file may be viewed on the terminal (on Log Decoder itself) with the tcpdump command, or it may be transferred to a system with GUI Desktop and viewed with Wireshark. Also, please attach the PCAP file while opening a new JIRA ticket.

Log Ingestion Issues

Command to view a PCAP file on the terminal itself.

```
# tcpdump -nnXr 514.pcap | less
```

(The above command is run wherever the PCAP file is.)

The above command with the `-X` option displays the content of each packet as a hexadecimal and ASCII dump. On scrolling down, one can see the various packets received on port 514. Here is one for example.

```
15:15:38.875840 IP 127.0.0.1.36441 > 127.0.0.1.514: Flags [P.], seq 1:50,
ack 1, win 257, options [nop,nop,TS val 1470803681 ecr 1470803681], length
49
        0x0000:  4500 0065 932e 4000 4006 a962 7f00 0001  E..e..@.@..b....
        0x0010:  7f00 0001 8e59 0202 b015 3dce c3ad b9e7  .....Y....=.....
        0x0020:  8018 0101 fe59 0000 0101 080a 57aa aee1  .....Y......W...
        0x0030:  57aa aee1 3c31 3e20 2550 4958 2d31 2d31  W...<1>.%PIX-1-1
        0x0040:  3031 3030 313a 2028 5052 494f 5249 5459  01001:.(PRIORITY
        0x0050:  2920 4661 696c 6f76 6572 2063 6162 6c65  ).Failover.cable
        0x0060:  204f 4b2e 0a                             .OK..
```

Log Ingestion Issues

Another command to view a PCAP file on the terminal.

```
# tcpdump -nnAr 514.pcap | less
```

(The above command is run wherever the PCAP file is.)

The above command with the -A option displays the content of each packet without the link layer headers as an ASCII dump. The TCP layer header appears as junk. This output is convenient to search for patterns with the `/<pattern>` command in `less`.

Example: One may search for the string “Failover cable OK” with the `/Failover cable OK` command and jump to a packet like the following.

```
15:15:38.875840 IP 127.0.0.1.36441 > 127.0.0.1.514: Flags [P.], seq
1:50, ack 1, win 257, options [nop,nop,TS val 1470803681 ecr
1470803681], length 49
E..e..@.@..b.........Y....=.........Y.....
W...W...<1> %PIX-1-101001: (PRIORITY) Failover cable OK.
```

Unidentified Content Issues

NetWitness Log Decoder

Unidentified Content Issues

Valid log – A valid syslog payload

```
<1> %PIX-1-101001: (PRIORITY) Failover cable OK.
```

Typical log – Typically sent by Log Collector to Log Decoder

```
[][FOO-VLC1][10.0.0.1][1412278173101][] %PIX-1-101001: (PRIORITY) Failover cable OK.
```

Unknown log – A valid syslog payload but not supported by our device parsers

```
<1> hello world
```

Invalid log

```
(PRIORITY) Failover cable OK.
```

Another invalid log

```
foo bar
```

Unidentified Content Issues

Let us feed all five logs we saw in the previous slide to Log Decoder and see what happens.

```
# echo "<1> %PIX-1-101001: (PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "[][FOO-VLC1][10.0.0.1][1412278173101][] %PIX-1-101001: (PRIORITY) Failover
cable OK." > /dev/tcp/127.0.0.1/514
# echo "<1> hello world" > /dev/tcp/127.0.0.1/514
# echo "(PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "foo bar" > /dev/tcp/127.0.0.1/514
```

(The above commands are run on Log Decoder.)

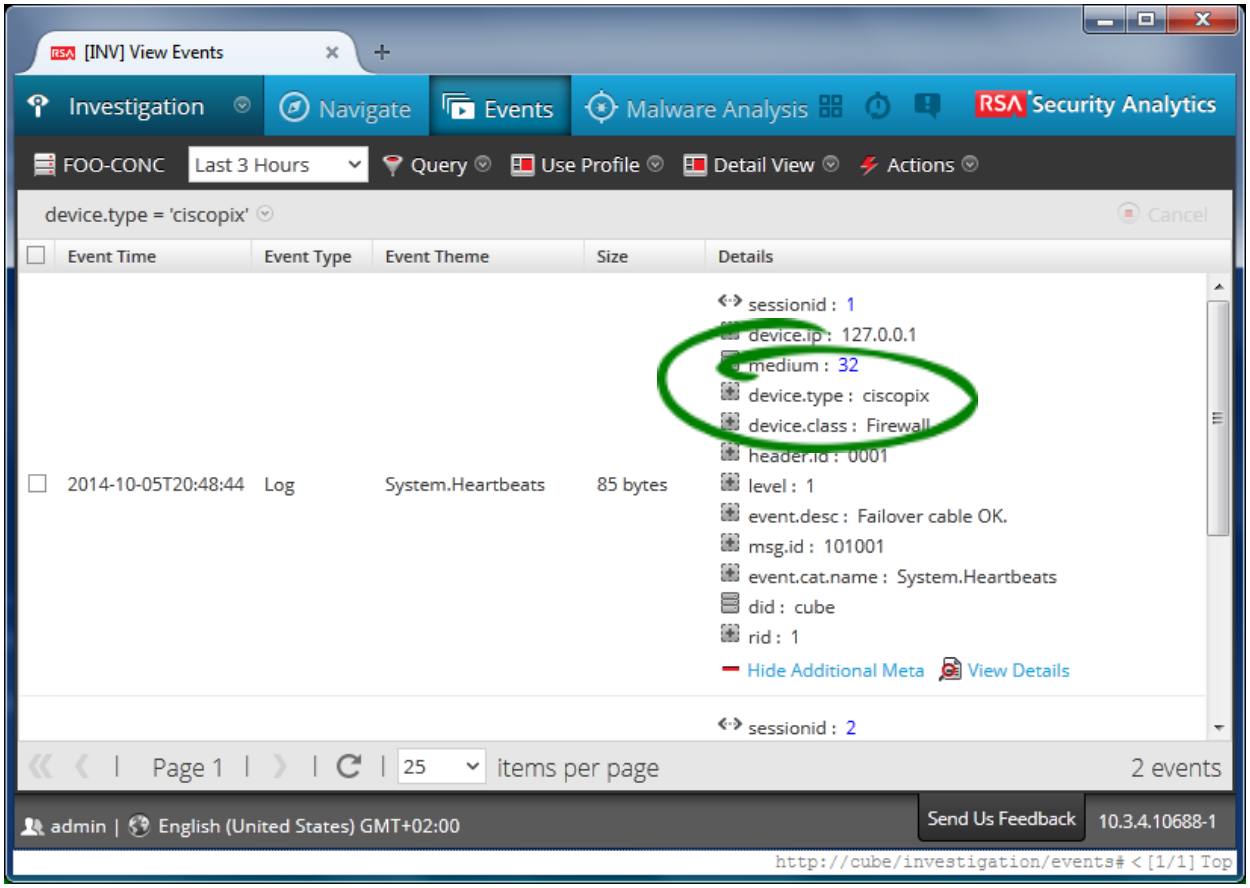
Valid log – Log Decoder accepts these logs. They have valid device.type meta.

Typical Log – Same as above

Unknown log – Log Decoder accepts these logs. But they have device.type = 'unknown'.

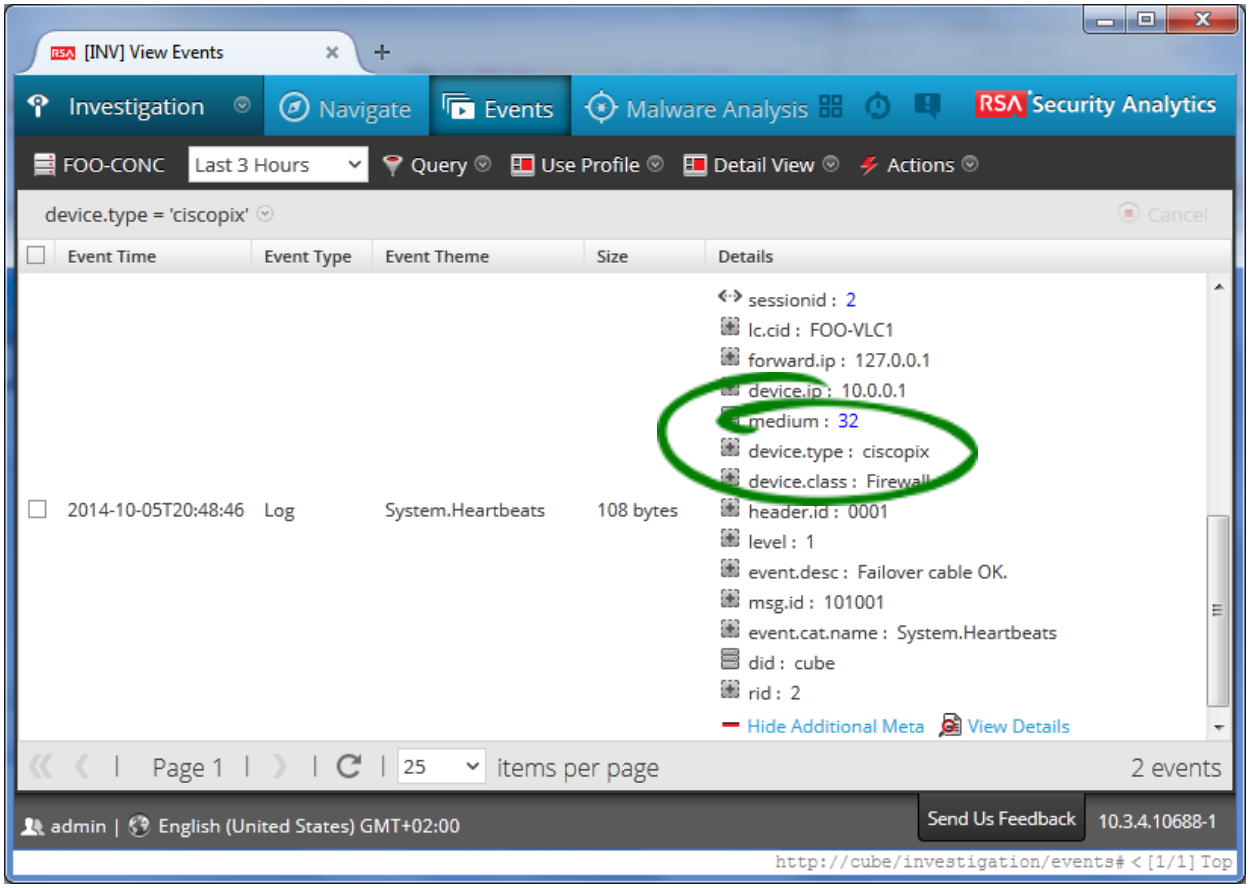
Invalid log – Log Decoder rejects these logs with ‘Unidentified content’ warnings.

Unidentified Content Issues



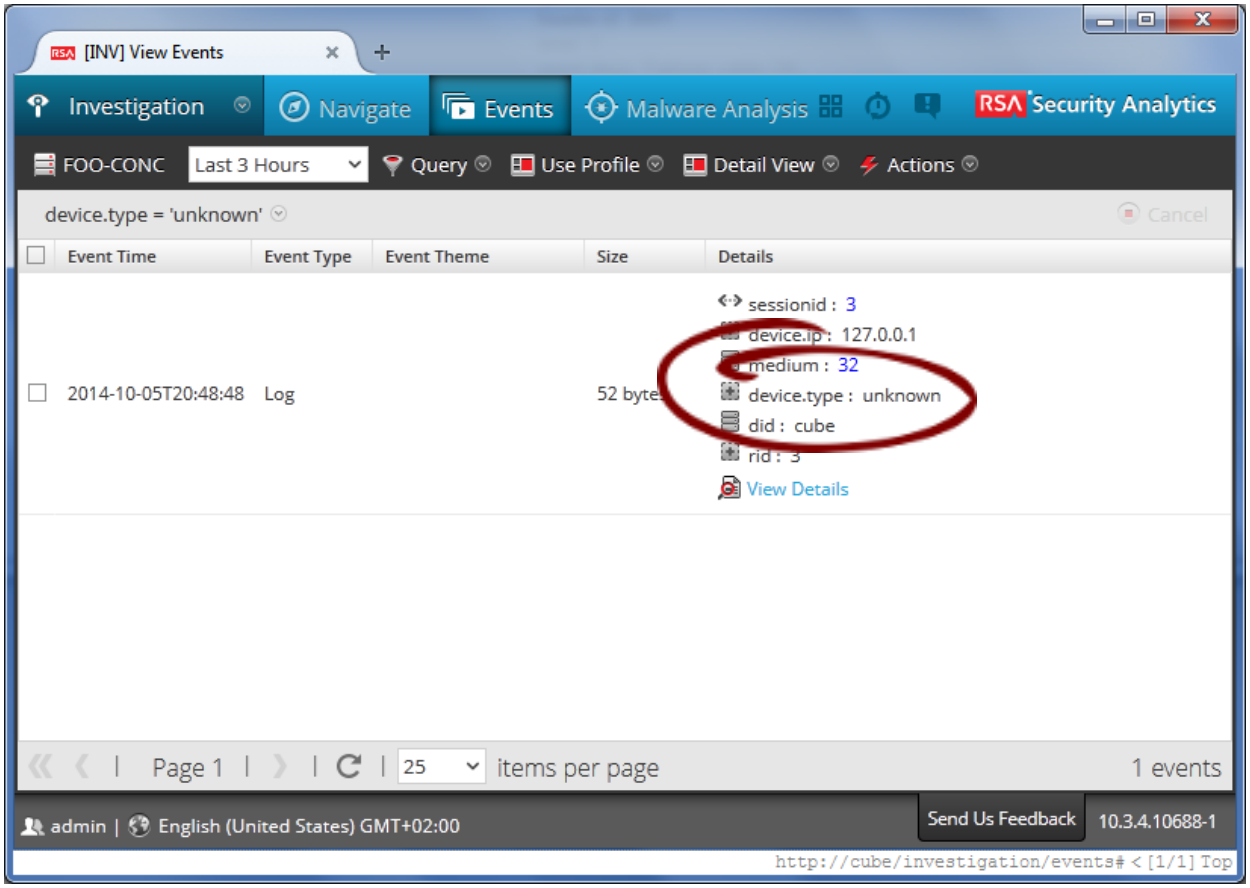
Valid log – Log Decoder accepts these logs. They have valid ‘device.type’ meta.

Unidentified Content Issues



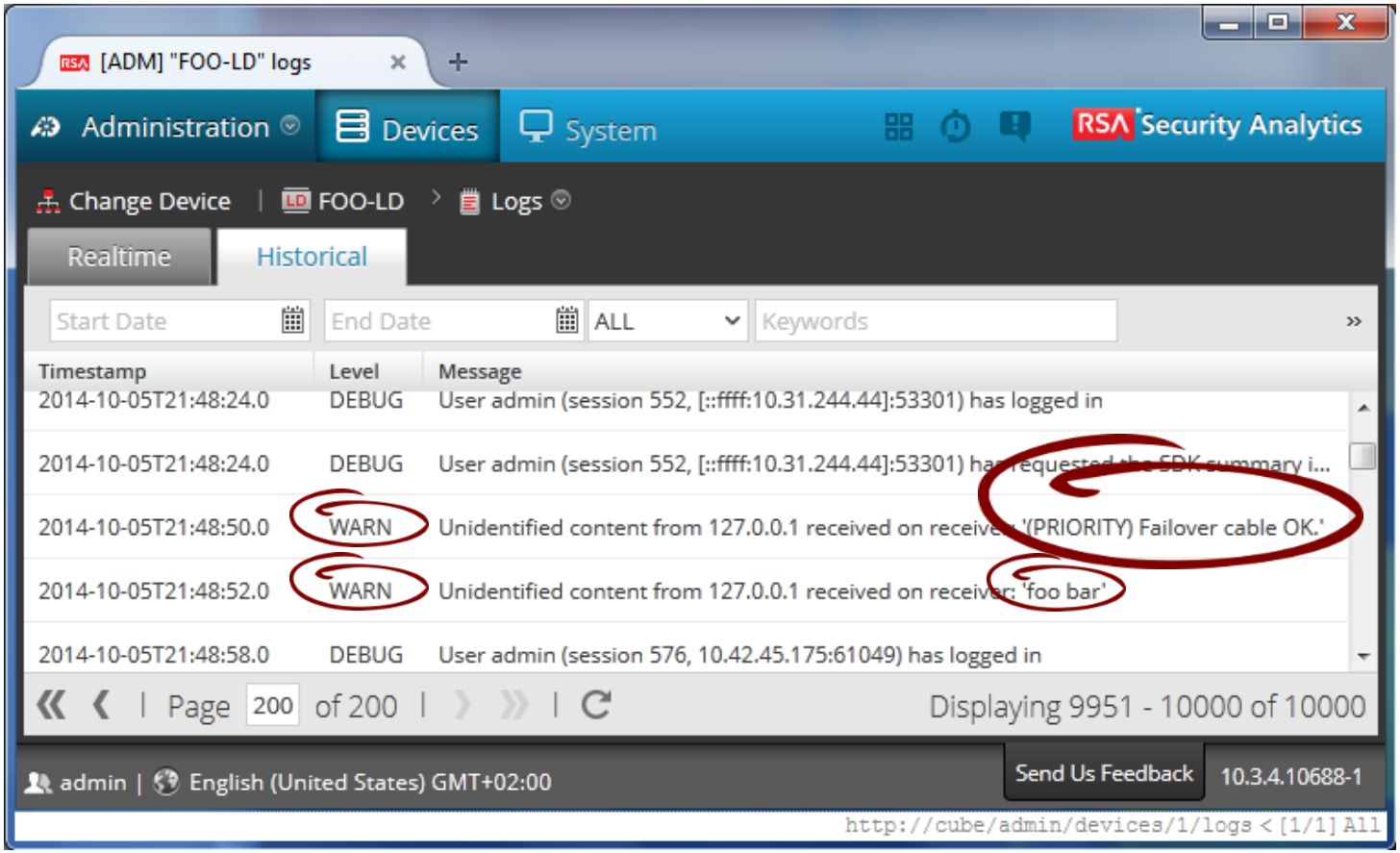
Typical log – Log Decoder accepts these logs. They have valid ‘device.type’ meta.

Unidentified Content Issues



Unknown log – Log Decoder accepts these logs. They have device.type = 'unknown'.

Unidentified Content Issues



Invalid log – Log Decoder rejects these logs with ‘Unidentified content’ warnings.

Unidentified Content Issues

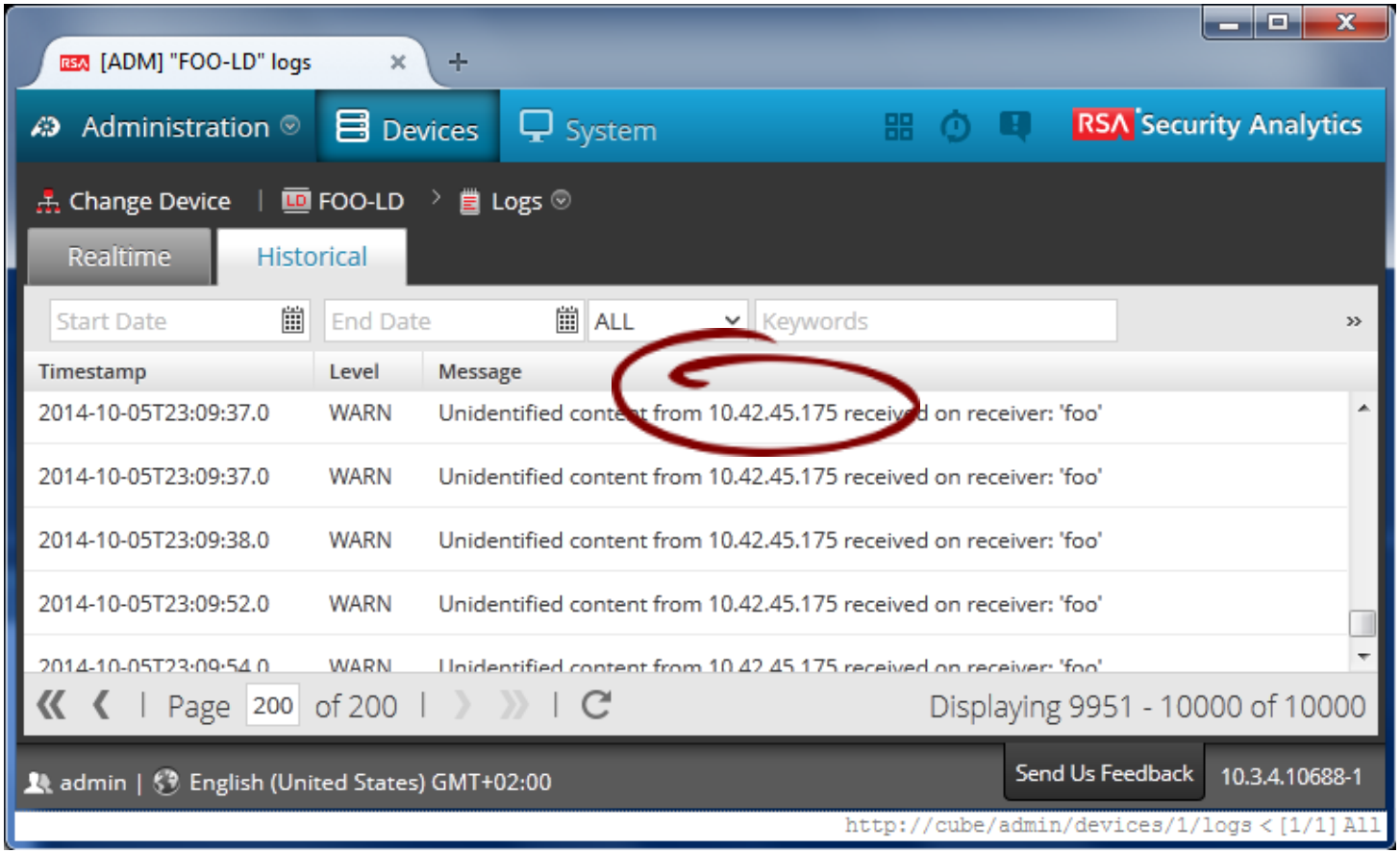
Ideally, invalid logs that cause ‘Unidentified content’ warnings should not reach Log Decoder.

But, in practice, sometimes invalid logs do reach Log Decoder.

Where are the invalid logs coming from?

We have to check the logs to figure out.

Unidentified Content Issues



Check the IP address in the 'Unidentified content' warnings

Unidentified Content Issues

If the invalid logs are coming from another system, we need to figure out why that system is sending invalid logs to our Log Decoder.

If the invalid logs are coming from ‘127.0.0.1’ then we need to figure which process is sending the invalid logs to Log Decoder.

Unidentified Content Issues

Command to figure the list of all processes connecting to Log Decoder.

```
# netstat -nopa | grep ":514\>"
```

(The above command is run on Log Decoder. The "\>" in the regular expression ensures that it matches ":514" but not ":5140" etc.)

One may see some other process in the output.

```
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.82:50706     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.100.200.23:47962     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.84:14529     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.85:38824     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:14529     10.101.202.84:514       ESTABLISHED 2743/rsyslogd       off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.86:52873     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:514           127.0.0.1:55773         ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:55773         127.0.0.1:514           ESTABLISHED 2578/NwLogCollector off (0.00/0/0)
tcp        0      0 :::514                  :::*                    LISTEN      7362/NwLogDecoder   off (0.00/0/0)
udp        0      0 0.0.0.0:514             0.0.0.0:*                           7362/NwLogDecoder   off (0.00/0/0)
udp        0      0 :::514                  :::*                                7362/NwLogDecoder   off (0.00/0/0)
```

Since in this output 'rsyslogd' seems to be connecting to Log Decoder, one needs to investigate if 'rsyslogd' is the culprit.

Unidentified Content Issues

Command to figure the list of all processes connecting to Log Decoder.

```
# netstat -nopa | grep ":514\>"
```

(The above command is run on Log Decoder. The "\>" in the regular expression ensures that it matches ":514" but not ":5140" etc.)

One may see some other process in the output.

Based on a true story.

```
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.82:50706     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.100.200.23:47962     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.84:14529     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.85:38824     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 10.101.202.84:14529     10.101.202.84:514       ESTABLISHED 2743/rsyslogd       off (0.00/0/0)
tcp        0      0 10.101.202.84:514       10.101.202.86:52873     ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:514           127.0.0.1:55773         ESTABLISHED 7362/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:55773         127.0.0.1:514           ESTABLISHED 2578/NwLogCollector off (0.00/0/0)
tcp        0      0 :::514                  :::*                    LISTEN      7362/NwLogDecoder   off (0.00/0/0)
udp        0      0 0.0.0.0:514             0.0.0.0:*                           7362/NwLogDecoder   off (0.00/0/0)
udp        0      0 :::514                  :::*                                7362/NwLogDecoder   off (0.00/0/0)
```

Since in this output 'rsyslogd' seems to be connecting to Log Decoder, one needs to investigate if 'rsyslogd' is the culprit.

Unidentified Content Issues

Command to figure the list of all processes connecting to Log Decoder.

```
# netstat -nopa | grep ":514\>"
```

(The above command is run on Log Decoder. The "\>" in the regular expression ensures that it matches ":514" but not ":5140" etc.)

Sometimes there may not be any other process to blame. So Log Collector becomes suspect.

```
tcp        0      0 0.0.0.0:6514            0.0.0.0:*               LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:32809         127.0.0.1:514           ESTABLISHED 2109/NwLogCollector off (0.00/0/0)
tcp        0      0 127.0.0.1:514           127.0.0.1:32809         ESTABLISHED 3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 :::6514                 :::*                    LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 :::514                  :::*                    LISTEN      3345/NwLogDecoder   off (0.00/0/0)
udp        0      0 0.0.0.0:514             0.0.0.0:*                           3345/NwLogDecoder   off (0.00/0/0)
udp        0      0 :::514                  :::*                                3345/NwLogDecoder   off (0.00/0/0)
```

In this case, we need to capture traffic arriving on port 514 with 'tcpdump' to understand what the invalid logs are and try to deduce where Log Collector is getting such logs from.

Unidentified Content Issues

Command to figure the list of all processes connecting to Log Decoder

```
# netstat -nopa | grep ":514\>"
```

(The above command is run on Log Decoder. The "\>" in the regular expression ensures that it matches ":514" but not ":5140" etc.)

Sometimes there may not be any other process to blame. So Log Collector becomes suspect.

```
tcp        0      0 0.0.0.0:6514            0.0.0.0:*               LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 127.0.0.1:33809         127.0.0.1:514           ESTABLISHED 3409/NwLogCollector off (0.00/0/0)
tcp        0      0 127.0.0.1:514           127.0.0.1:33809         ESTABLISHED 3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 :::6514                 :::*                    LISTEN      3345/NwLogDecoder   off (0.00/0/0)
tcp        0      0 :::514                  :::*                    LISTEN      3345/NwLogDecoder   off (0.00/0/0)
udp        0      0 0.0.0.0:514             0.0.0.0:*                           3345/NwLogDecoder   off (0.00/0/0)
udp        0      0 :::514                  :::*                                3345/NwLogDecoder   off (0.00/0/0)
```

In this case, we need to capture traffic arriving on port 514 with 'tcpdump' to understand what the invalid logs are and try to deduce where Log Collector is getting these logs from.

Based on a true story.

Log Collector was found to be sending logs with newlines. A fix was developed for Log Collector.

Unidentified Content Issues

We need to capture packets while the 'Unidentified content' warnings are being logged by Log Decoder, not after they are logged, so that we capture the packets that caused the issue.

This can be tricky if the 'Unidentified content' warnings comes in short bursts after long intervals. In such a case, we don't know when to run the tcpdump command and how long to wait.

So we wrote a tool called 'autocap' to automatically monitor /var/log/messages and start 'tcpdump' and capture the output of 'netstat' automatically when 'Unidentified content' warnings are found, and stop the capture when sufficient number of such warnings have been found.

Unidentified Content Issues

- Download ‘autocap’ from RSA and copy it to Log Decoder.
- In the shell, change your current directory to the directory where the script has been copied.
- Then execute the following command.

```
# nohup sh autocap -s "Unidentified content" > autocap.txt &
```

- The script would now log its progress to ‘autocap.txt’. You may use the following command to monitor its progress.

```
# tail -f autocap.txt
```

- In the end, the script will tell the user to send a tarball to RSA. Here is an example.

```
Send _autocap_2014-10-06_03-39-33.tar.gz to RSA.
```

- Pick up the .tar.gz file the script mentioned in its last message and use it for analysis. It’ll contain a PCAP with traffic for port 514 and second-by-second output from netstat for the duration when the issue was occurring.
- Attach this .tar.gz file to any JIRA ticket you create for ‘Unidentified content’ issue.

Log Parsing

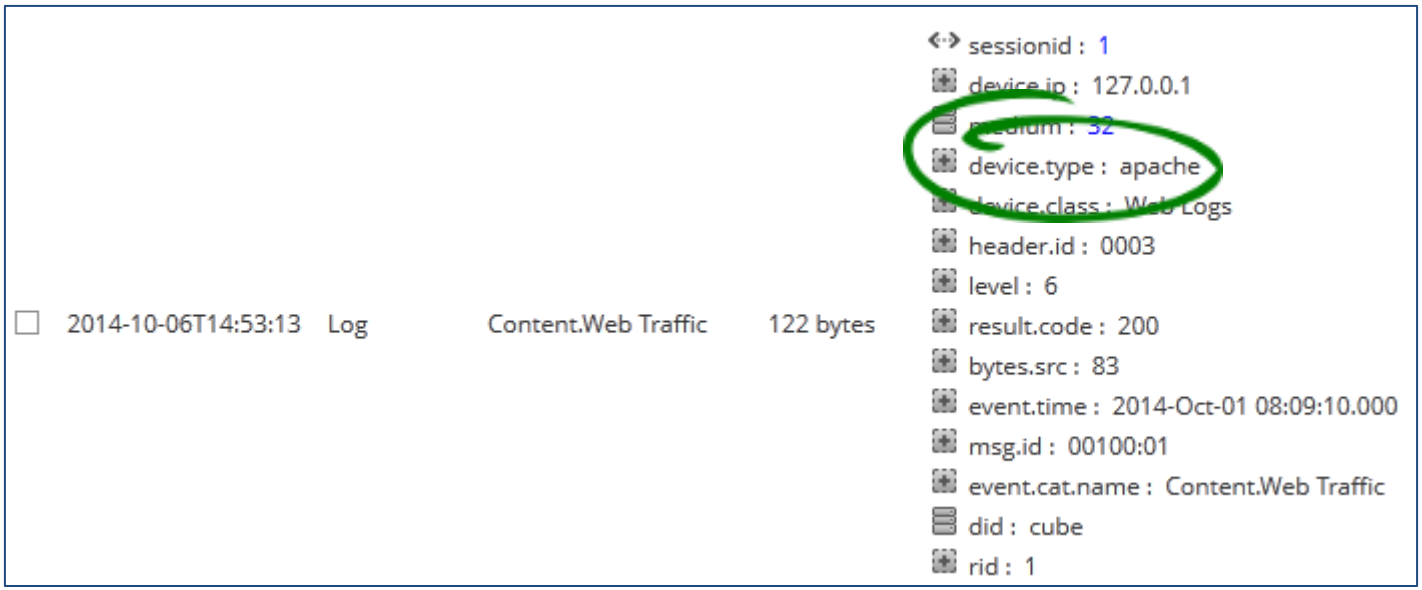
NetWitness Log Decoder

Log Parsing

This is an example of a valid and supported log.

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

When we feed this to Log Decoder, it parses fine and generates metas including device.type = 'apache'.



Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

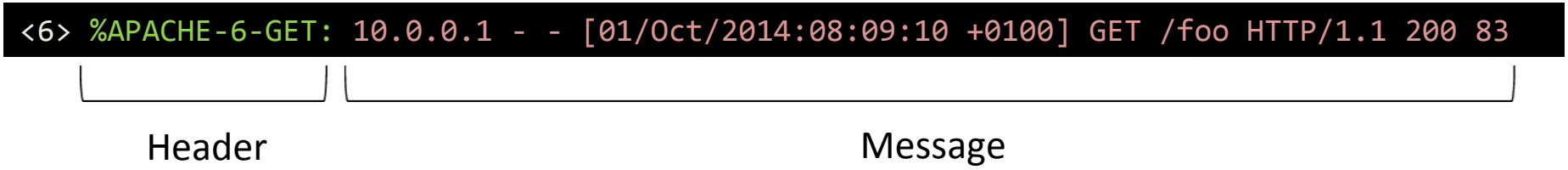
The device log parsers are written in XML.

Log Decoder reads these XMLs and creates parsers to parse logs as defined in the XML.

There is a parser directory corresponding to each supported device in /etc/netwitness/ng/envision/etc/devices.

For example, for Apache logs, the parser directory is /etc/netwitness/ng/envision/etc/devices/apache. This directory contains a file called v20_apachemsg.xml that defines the supported Apache logs.

Log Parsing



The header in the log should match a header definition in the device parser XML.

The message in the log should match a message definition in the device parser XML.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

This log matches the following header definition in /etc/netwitness/ng/envision/etc/devices/apache/v20_apachemsg.xml.

```
<HEADER
    id1="0003"
    id2="0003"
    content="%APACHE-&lt;level&gt;-&lt;messageid&gt;: &lt;!payload&gt;" />
```

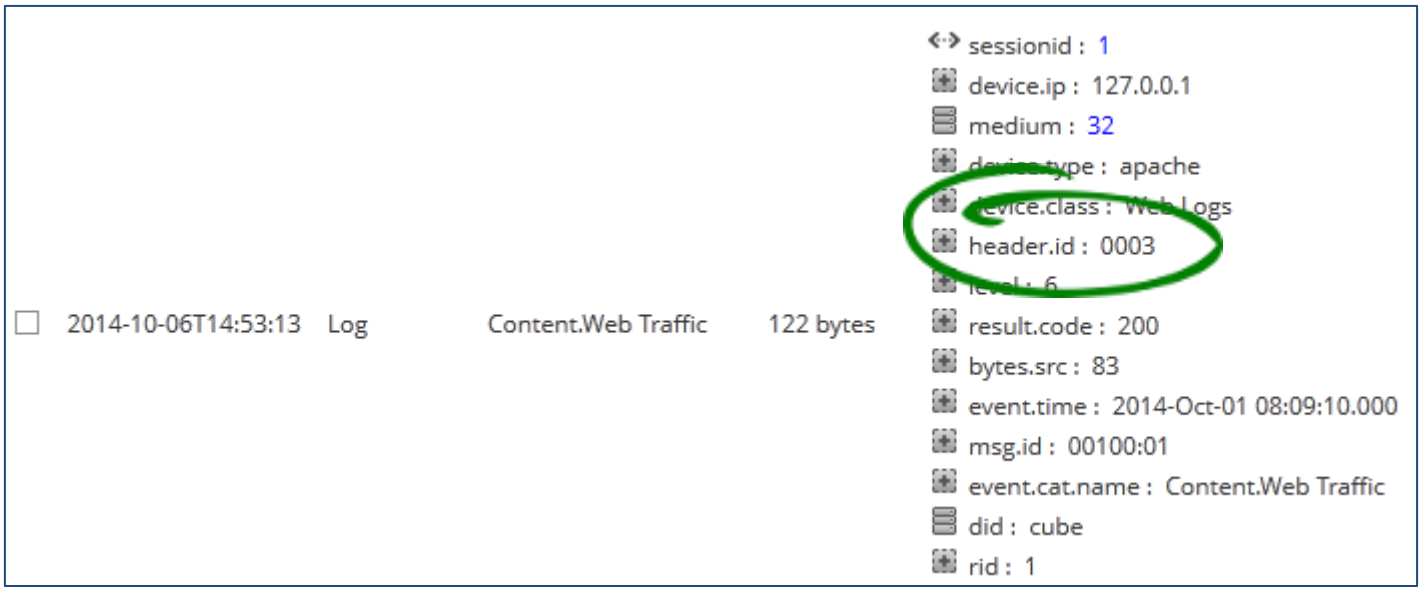
After a matching message definition is found for the message, a meta called 'header.id' is created with the id1 of the matching header definition as its value. For example, for this log, header.id = '0003' is created.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message



Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

This log matches this header definition in /etc/netwitness/ng/envision/etc/devices/apache/v20_apachemsg.xml.

```
<HEADER
    id1="0003"
    id2="0003"
    content="%APACHE-&lt;level&gt;-&lt;messageid&gt;: &lt;!payload&gt;" />
```

During the header match, the 'messageid' field in the header definition matches 'GET' in the log, so now a message definition with id2 = "GET" is looked up.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

Message definition with id2="GET" is found.

```
<MESSAGE
    level="6"
    tableid="82"
    parse="1"
    parsedefvalue="1"
    eventcategory="1204000000"
    sitetrack="1"
    id1="00100:01"
    id2="GET"
    summary="NIC_B_ADDRESS_ACCOUNTING;"
    content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
<MESSAGE
    level="6"
    tableid="82"
    parse="1"
    parsedefvalue="1"
    eventcategory="1204000000"
    sitetrack="1"
    id1="00100:01"
    id2="GET" ...
```

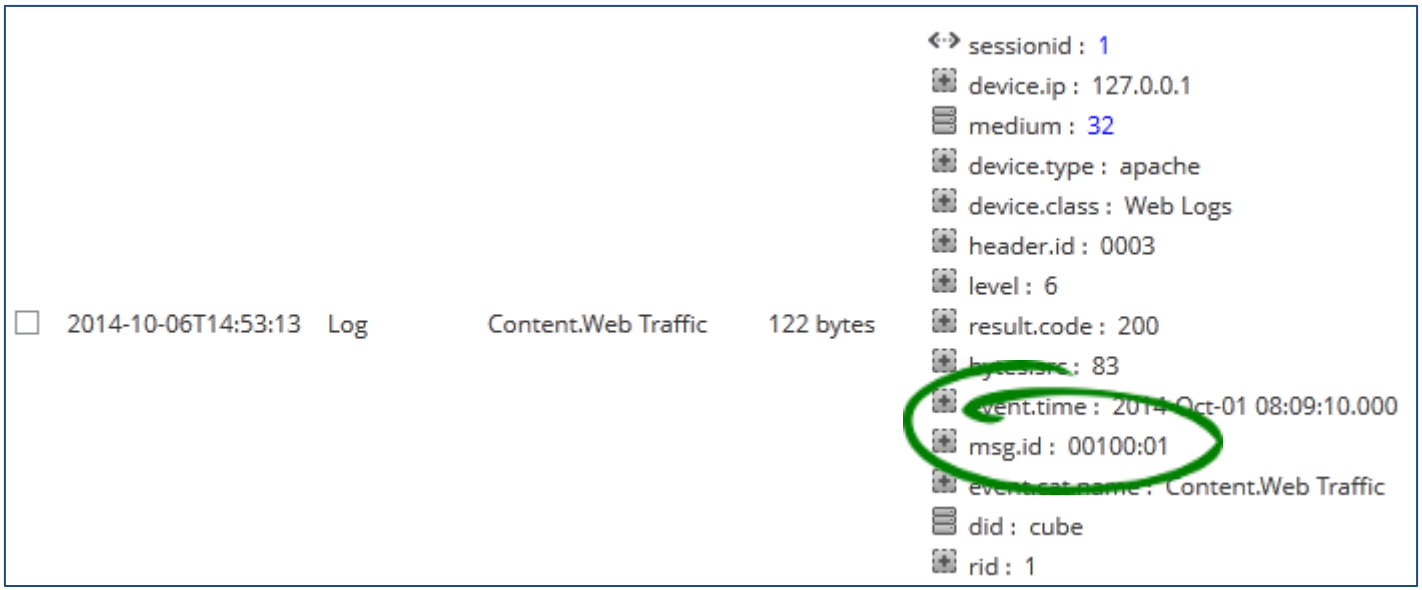
After a matching message definition is found for the message, a meta called ‘msg.id’ is created with the ‘id1’ attribute’s value. For example, for this log, msg.id = '00100:01' is created.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message



Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
<MESSAGE
    level="6"
    tableid="82"
    parse="1"
    parsedefvalue="1"
    eventcategory="1204000000"
    sitetrack="1"
    id1="00100:01" ...
```

The ‘eventcategory’ number is looked up in /etc/netwitness/ng/envision/etc/ecat.ini. It contains the following entry. The string is used as the value for ‘event.cat.name’ meta.

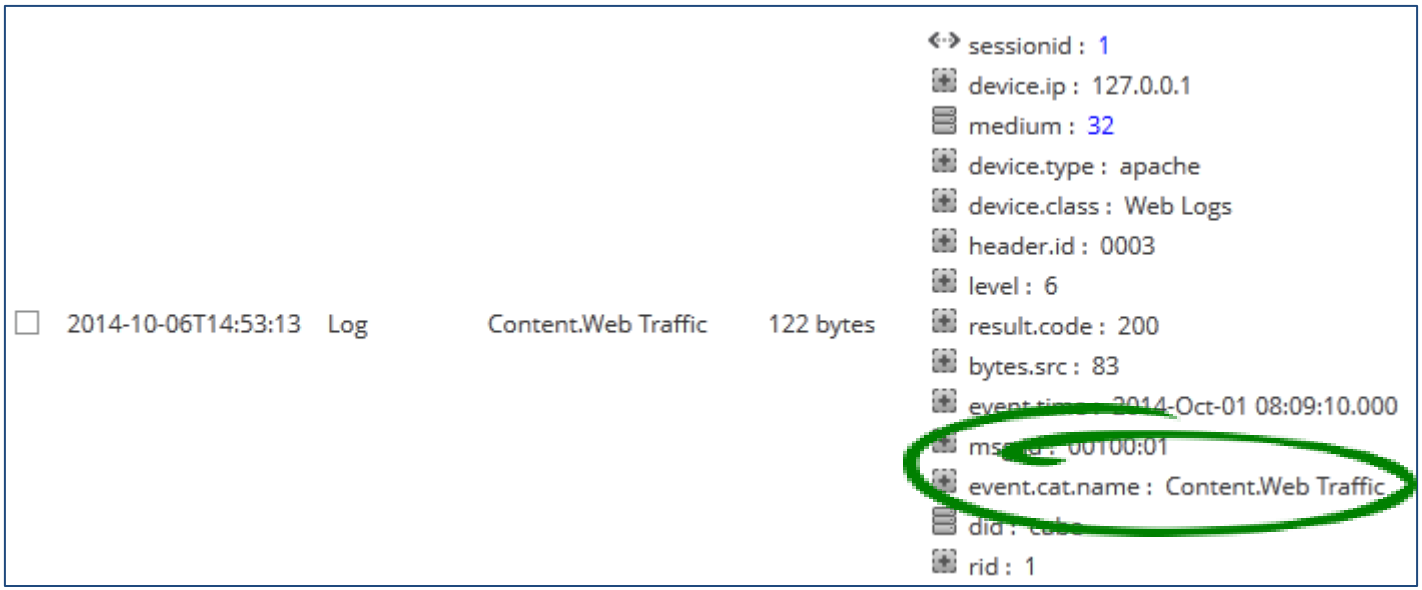
```
1204000000,Content.Web Traffic
```

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message



Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

Apart from the fields extracted directly from the log, an additional ‘event_time’ field is created by parsing the time extracted in the ‘fld7’ field.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

The parser fields are mapped to meta keys in /etc/netwitness/ng/envision/etc/table-map.xml.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

An example mapping in /etc/netwitness/ng/envision/etc/table-map.xml.

```
<mapping envisionName="event_time" nwName="event.time" flags="None" format="TimeT"
envisionDisplayName="EventDate/Time|EventTime|LastScanned|Created"/>
```

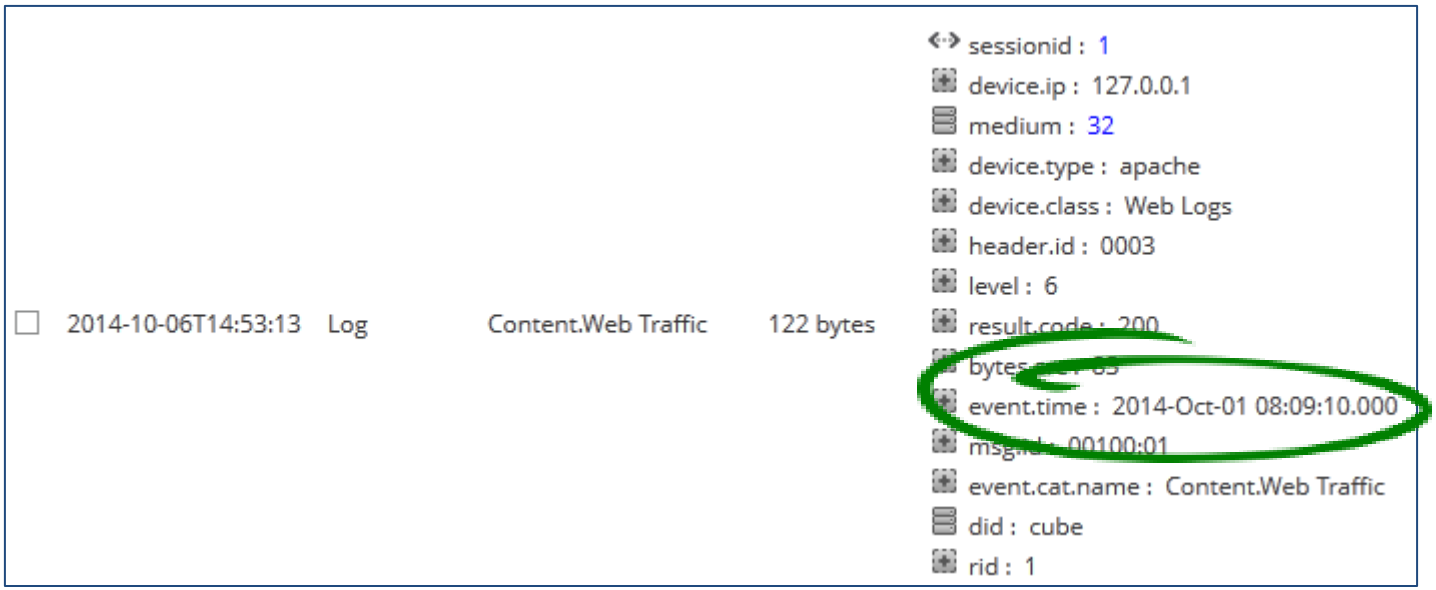
This maps the 'event_time' field in the device parser to 'event.time' meta. Since this meta has flags="None", this meta is written to the disk.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message



Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

An example mapping in /etc/netwitness/ng/envision/etc/table-map.xml.

```
<mapping envisionName="username" nwName="user.dst" flags="None"
envisionDisplayName="UserName|UserID|User|UserName|Username" nullTokens="none|-"/>
```

The 'username' field matches '-' (hyphen) in the log which is defined as a null token in this mapping, so this is not written to the disk.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

An example mapping in /etc/netwitness/ng/envision/etc/table-map.xml.

```
<mapping envisionName="webpage" nwName="web.page" flags="Transient"
envisionDisplayName="WebPage"/>
```

This maps the 'webpage' field in the device parser to 'web.page' meta. Since this meta has flags="Transient", this meta is not written to the disk.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

Message

```
...
content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

An example mapping in /etc/netwitness/ng/envision/etc/table-map.xml.

```
<mapping envisionName="sbytes" nwName="bytes.src" flags="None" format="UInt64"
nullTokens="(null)|-"/>
```

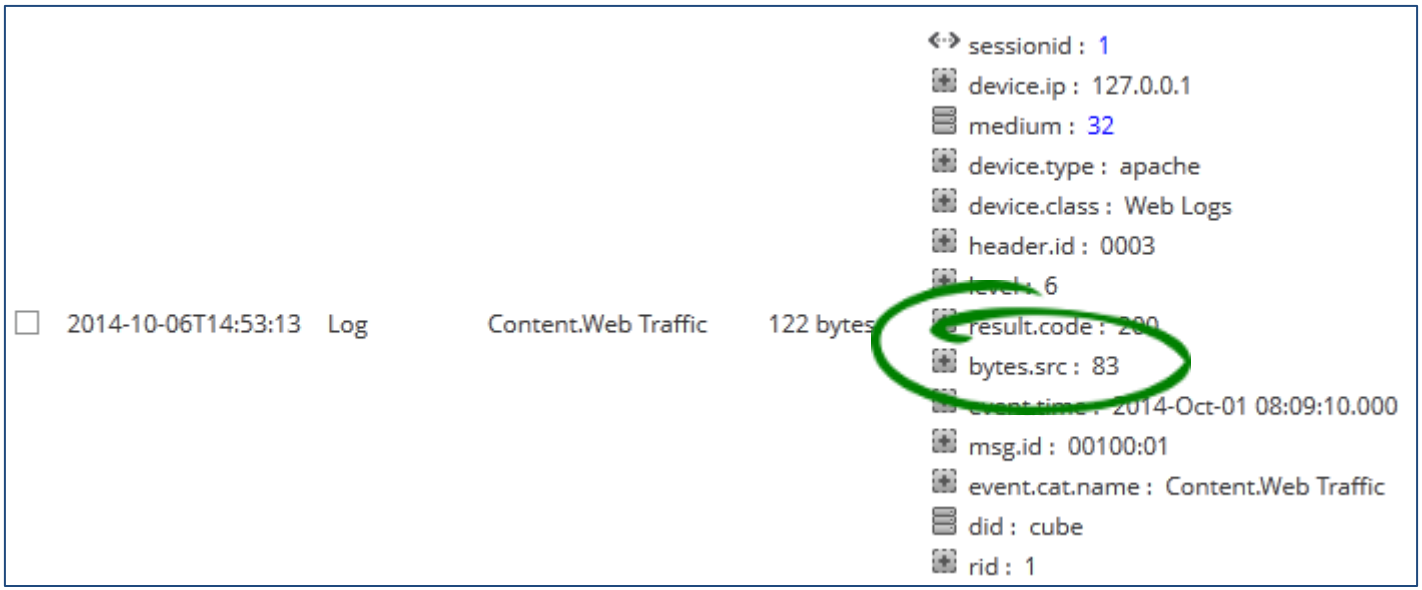
This maps the 'sbytes' field in the device parser to 'bytes.src' meta. Since this meta has flags="None", this meta is written to the disk.

Log Parsing

```
<6> %APACHE-6-GET: 10.0.0.1 - - [01/Oct/2014:08:09:10 +0100] GET /foo HTTP/1.1 200 83
```

Header

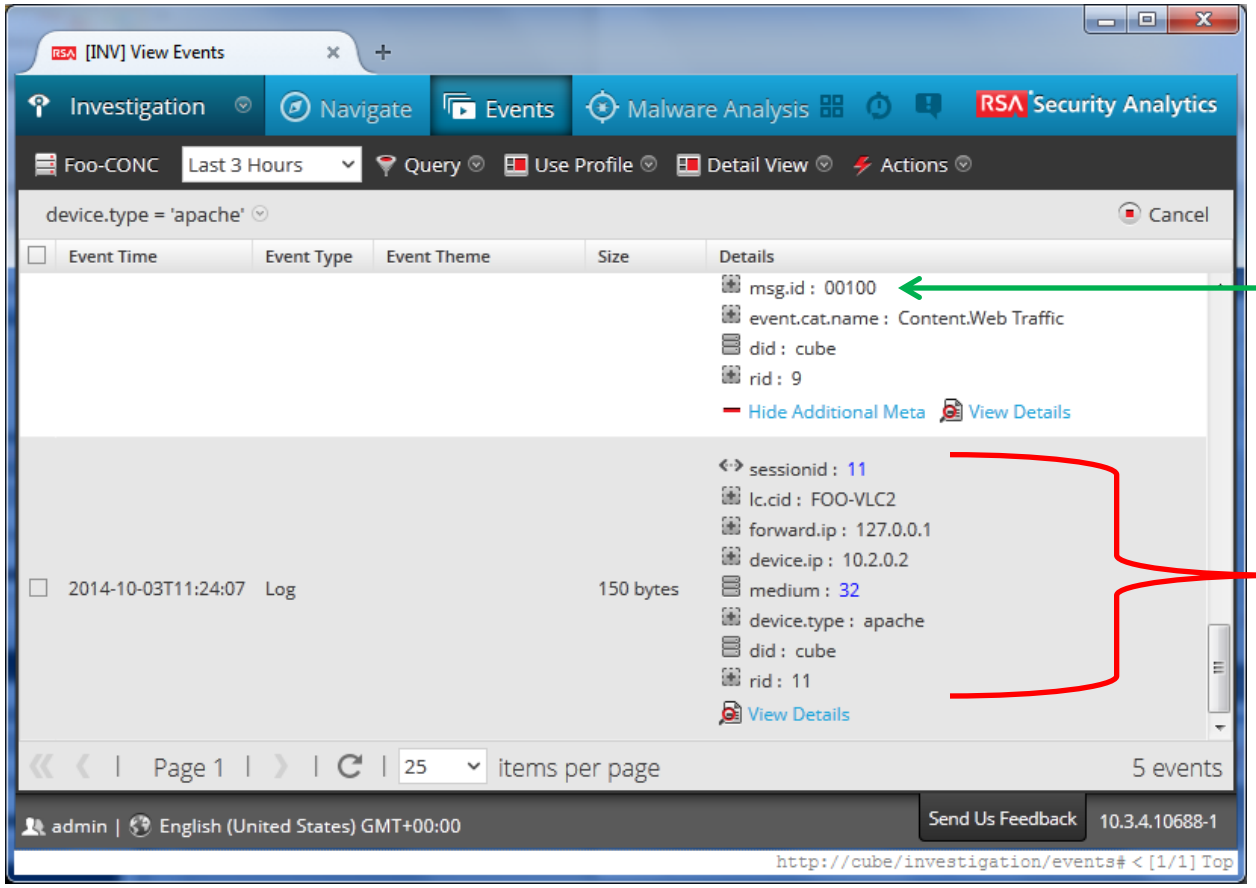
Message



Missing Metas

NetWitness Log Decoder

Missing Metas

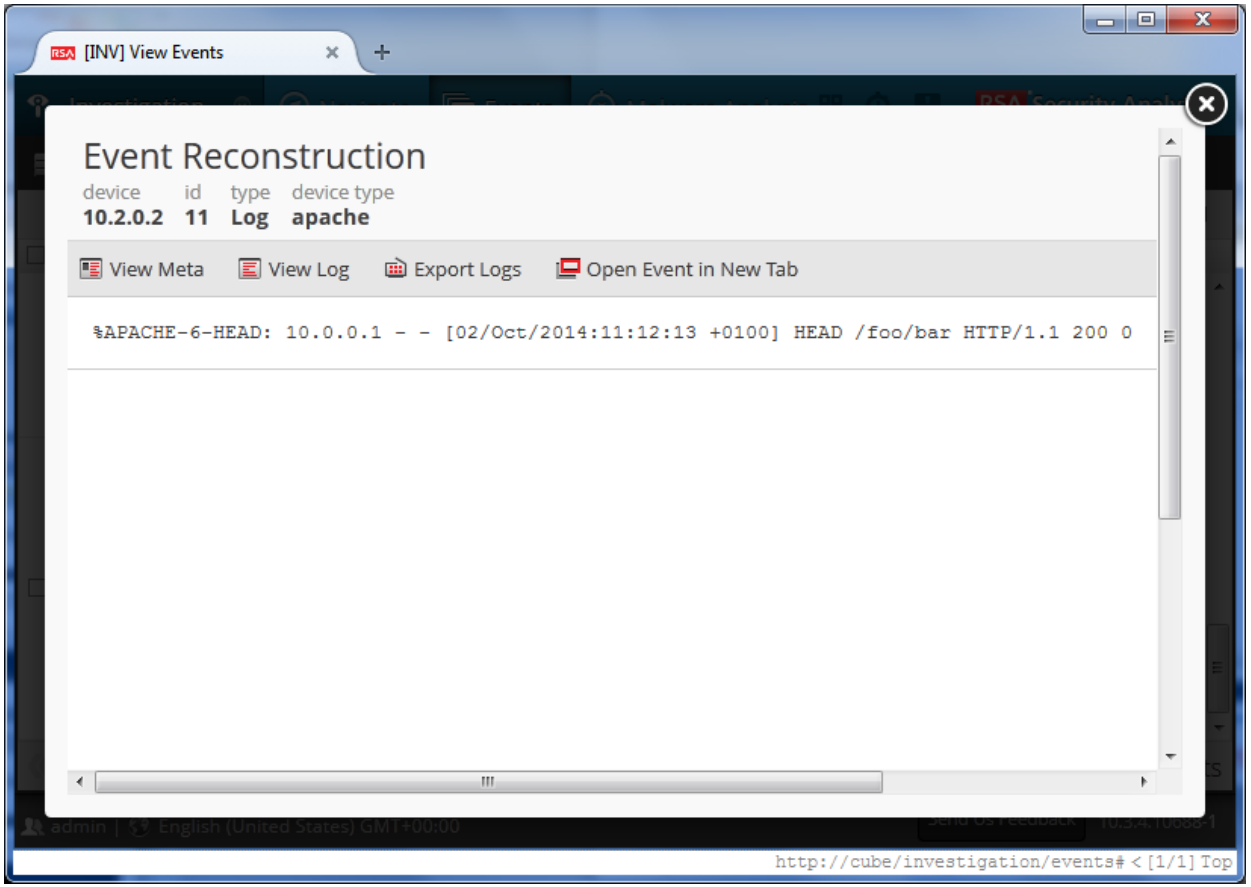


All logs should have msg.id meta

There is no msg.id meta for this log. This is an issue!

'device.type' is present but 'msg.id' is missing

Missing Metas



Is this a supported log?

Missing Metas

Log

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /foo/bar HTTP/1.1 200 0
```

When we see ‘device.type’ meta for a log, but we don’t see any ‘msg.id’ meta for it, it implies that the log matched a header definition of a device parser, but it did not match a message definition.

In fact, we can find a matching header definition in /etc/netwitness/ng/envision/etc/devices/apache/v20_apachemsg.xml

```
<HEADER
        id1="0003"
        id2="0003"
        content="%APACHE-&lt;level&gt;-&lt;messageid&gt;: &lt;!payload&gt;" />
```

Missing Metas

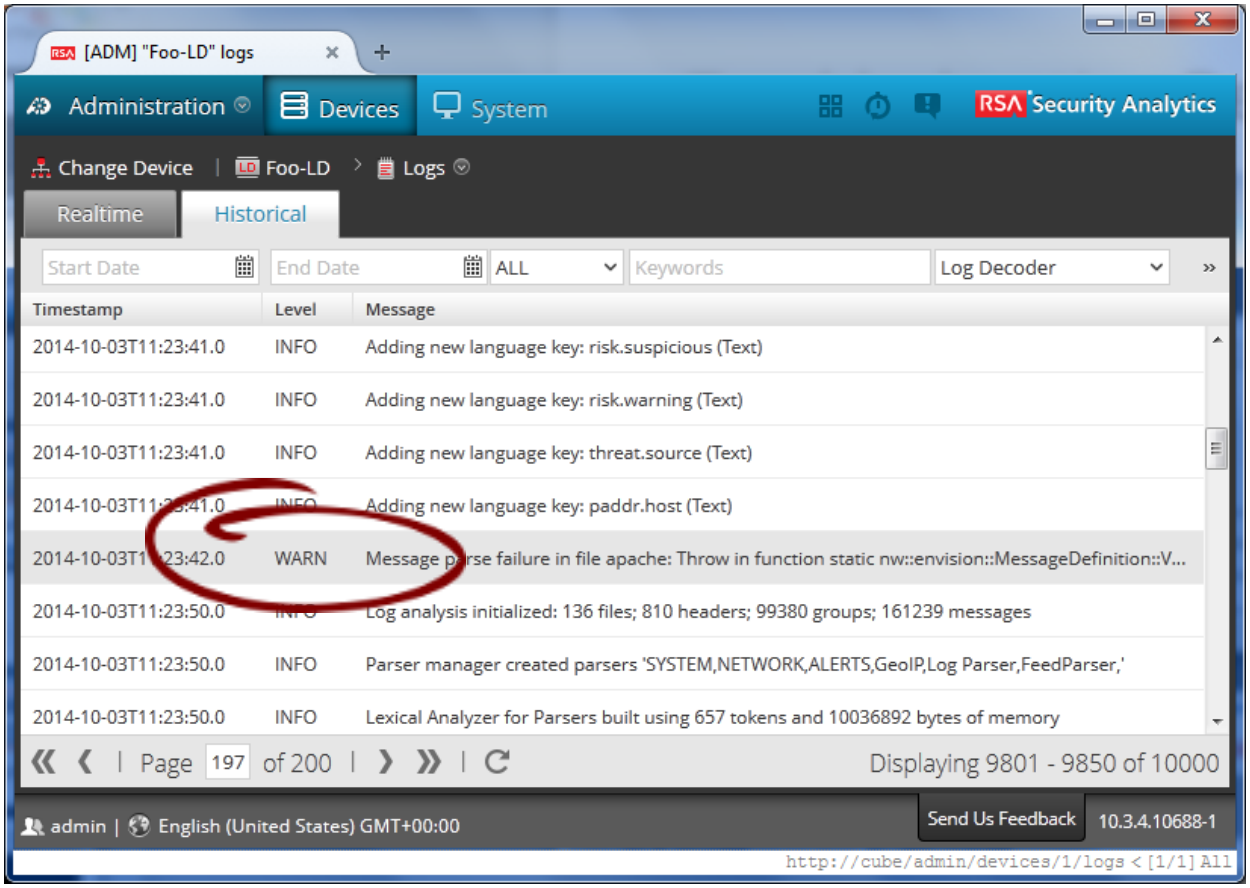
Log

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /foo/bar HTTP/1.1 200 0
```

There is a matching Message ID (id2="HEAD") as well in /etc/netwitness/ng/envision/etc/devices/apache/v20_apachemsg.xml

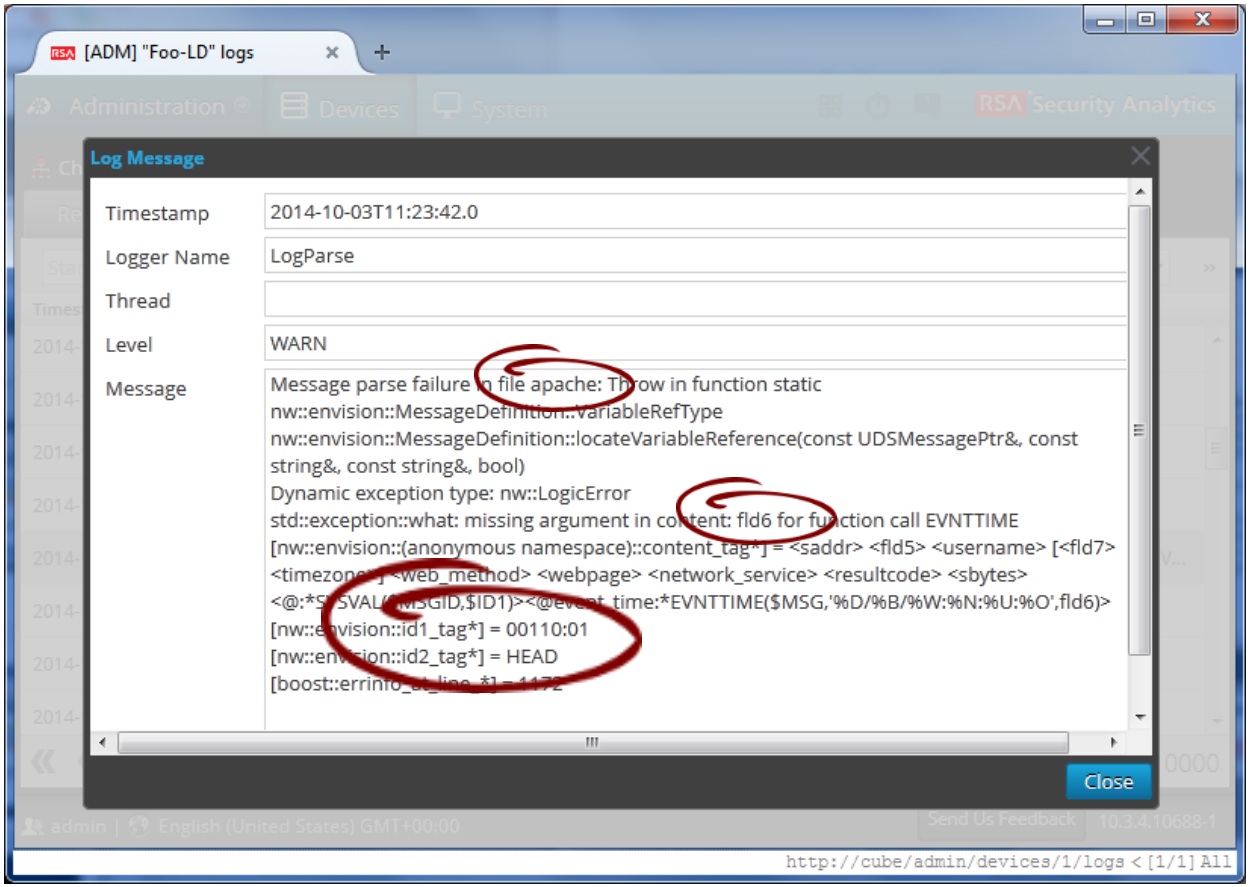
```
<MESSAGE
        level="6"
        tableid="82"
        parse="1"
        parsedefvalue="1"
        eventcategory="1204000000"
        sitetrack="1"
        id1="00110:01"
        id2="HEAD"
        summary="NIC_B_ADDRESS_ACCOUNTING;"
        content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld6)&gt; " />
```

Missing Metas



‘Message parse failure’ warning indicates a semantic error in device parser

Missing Metas



Error in ‘apache’ parser in the message tag with id1="00110:01" and id2="HEAD"

Missing Metas

Log

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /foo/bar HTTP/1.1 200 0
```

EVNTTIME($MSG, …) function must use variables defined in this message definition.
fld6 is used in EVNTTIME($MSG, …) but it is not defined anywhere here.

```
<MESSAGE
        level="6"
        tableid="82"
        parse="1"
        parsedefvalue="1"
        eventcategory="1204000000"
        sitetrack="1"
        id1="00110:01"
        id2="HEAD"
        summary="NIC_B_ADDRESS_ACCOUNTING;"
        content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld6)&gt; " />
```

Missing Metas

Log

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /foo/bar HTTP/1.1 200 0
```

fld7 in the definition below matches the event time in the log, so fld7 is the right argument to use with the EVNTTIME($MSG, …) function.

```
<MESSAGE
        level="6"
        tableid="82"
        parse="1"
        parsedefvalue="1"
        eventcategory="1204000000"
        sitetrack="1"
        id1="00110:01"
        id2="HEAD"
        summary="NIC_B_ADDRESS_ACCOUNTING;"
        content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

Missing Metas

Log

This is not a real issue in the existing Apache Parser!

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /baz/qux HTTP/1.1 200 0
```

EVNTTIME($MSG, ...) function must use variables defined in this message definition.
fld7 is used in EVNTTIME($MSG, ...) but it is not defined anywhere here.

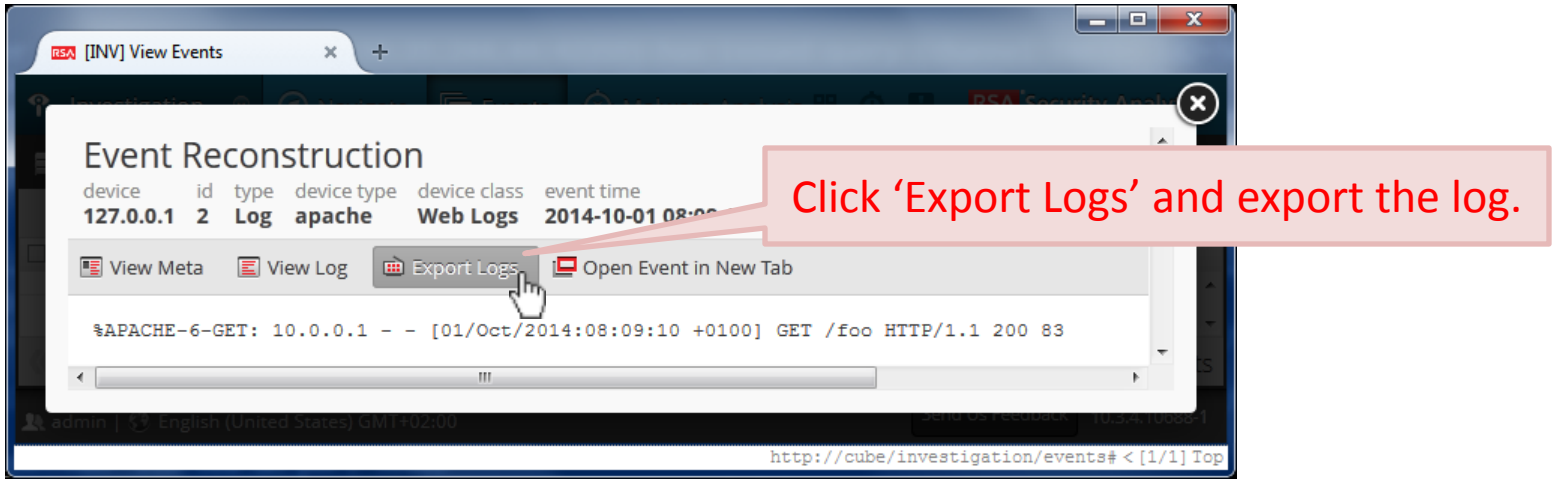
This was an example based on a true story about a custom device parser.

```
<MESSAGE
      level="6"
      tableid="82"
      parse="1"
      parsedefvalue="1"
      eventcategory="1204000000"
      sitetrack="1"
      id1="HEAD:1"
      id2="HEAD"
      summary="NIC_B_ADDRESS_ACCOUNTING;"
      content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt; &lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

Missing Metas

Data to provide while creating JIRA ticket when metas are missing:

1. nwtech dump
2. tar -cvzf envision.tar.gz /etc/netwitness/ng/envision
3. The log (Go to Concentrator's Investigate → Events, filter and find the log, click 'View Details' and then click 'Export Logs' to export it. The exported log can be downloaded by going to Profile → Jobs.



Missing Metas (continued…)

NetWitness Log Decoder

Missing Metas

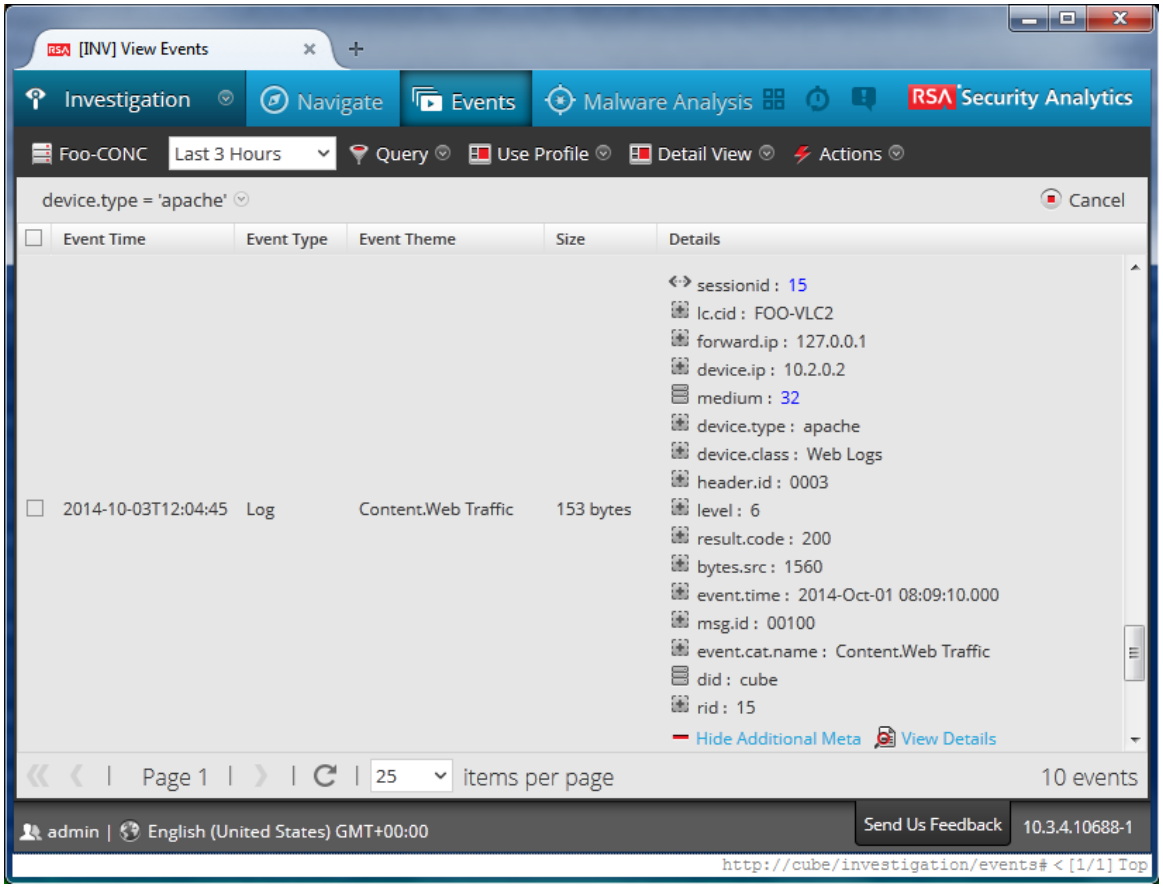
Log

```
%APACHE-6-HEAD: 10.0.0.1 - - [02/Oct/2014:11:12:13 +0100] HEAD /foo/bar HTTP/1.1 200 0
```

Message definition in the Apache parser

```
<MESSAGE
        level="6"
        tableid="82"
        parse="1"
        parsedefvalue="1"
        eventcategory="1204000000"
        sitetrack="1"
        id1="00110:01"
        id2="HEAD"
        summary="NIC_B_ADDRESS_ACCOUNTING;"
        content="&lt;saddr&gt; &lt;fld5&gt; &lt;username&gt; [&lt;fld7&gt; &lt;timezone&gt;]
&lt;web_method&gt; &lt;webpage&gt; &lt;network_service&gt; &lt;resultcode&gt; &lt;sbytes&gt;
&lt;@:*SYSVAL($MSGID,$ID1)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)&gt; " />
```

Missing Metas



The web page '/foo/bar' is missing from the list of metas

Missing Metas

The web page in the Apache log is parsed as ‘webpage’ field in the ‘apache’ device parser. This is an enVision field name.

enVision field names are mapped to Netwitness meta names in /etc/netwitness/ng/envision/etc/table-map.xml.

Custom mappings may be defined in /etc/netwitness/ng/envision/etc/table-map-custom.xml.

If there is no mapping for an enVision field, then the value for that field is not written to the disk.

If there is a mapping for an enVision field, but it is flagged as ‘Transient’, then the value for that field is not written to the disk.

Is there a mapping for enVision field in /etc/netwitness/ng/envision/etc/table-map.xml or /etc/netwitness/ng/envision/etc/table-map-custom.xml?

Missing Metas

Is there a mapping for enVision field in /etc/netwitness/ng/envision/etc/table-map.xml?

```
# grep 'envisionName="webpage"' /etc/netwitness/ng/envision/etc/table-map.xml
        <mapping envisionName="webpage" nwName="web.page" flags="Transient"
envisionDisplayName="WebPage"/>
```

It is defined, but it is flagged as 'Transient'!

Is there a mapping for enVision field in /etc/netwitness/ng/envision/etc/table-map-custom.xml?

```
# grep 'envisionName="webpage"' /etc/netwitness/ng/envision/etc/table-map-
custom.xml
grep: /etc/netwitness/ng/envision/etc/table-map-custom.xml: No such file or
directoryenvisionDisplayName="WebPage"/>
```

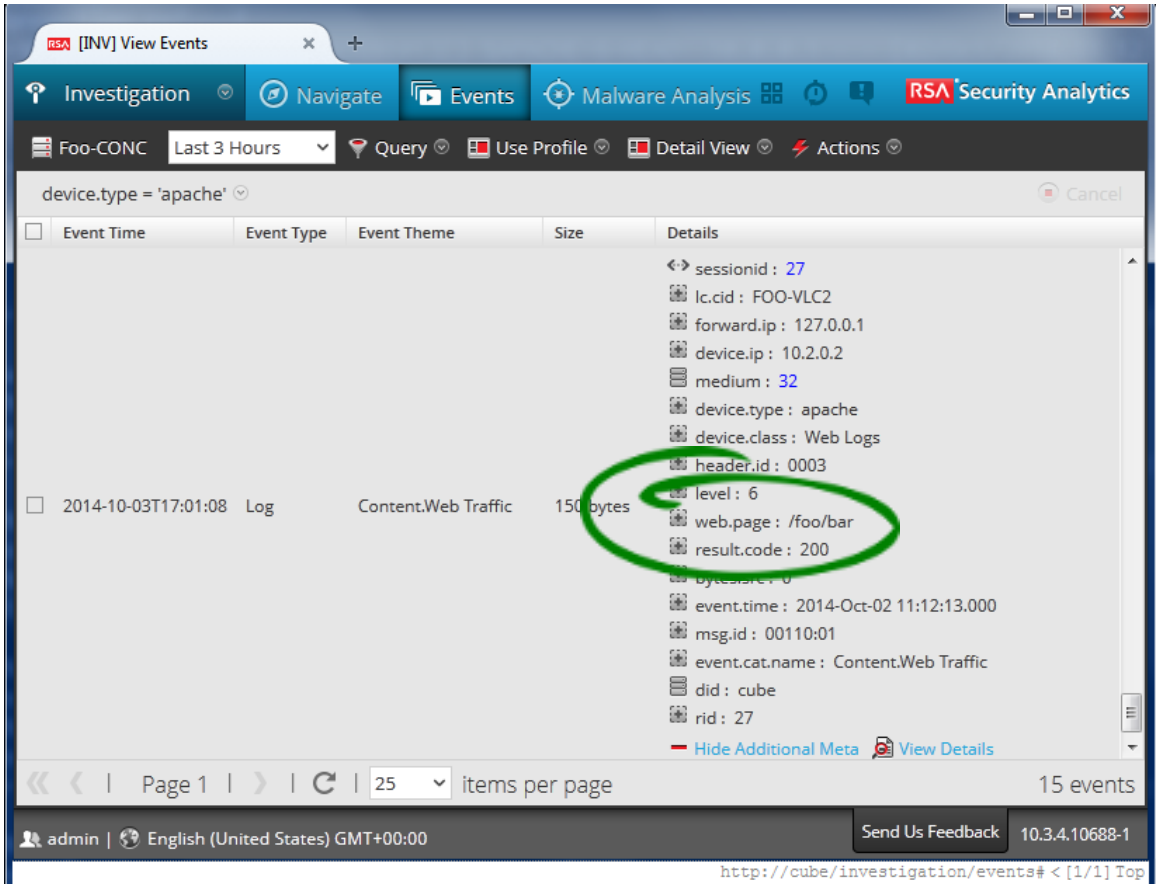
Missing Metas

If mapping for an enVision field is missing from /etc/netwitness/ng/envision/etc/table-map.xml, or if it is defined with `flags="Transient"`, the values of such a field is not written to the disk by Log Decoder, and thus not available in the query results.

To make it available, override it by defining it with `flags="None"` in /etc/netwitness/ng/envision/etc/table-map-custom.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<mappings>
    <mapping envisionName="webpage" nwName="web.page" flags="None"
             envisionDisplayName="WebPage"/>
</mappings>
```

Missing Metas

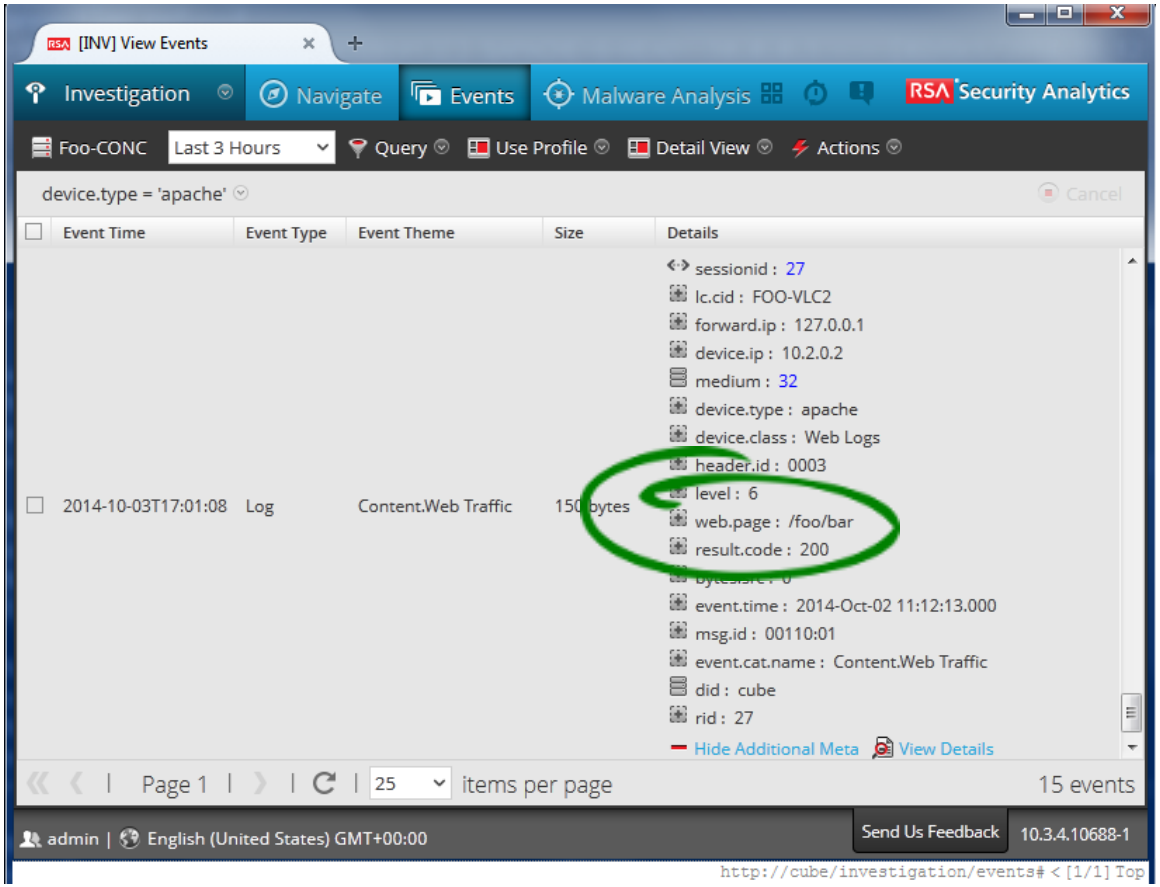


web.page meta is now available

Meta Query Issues

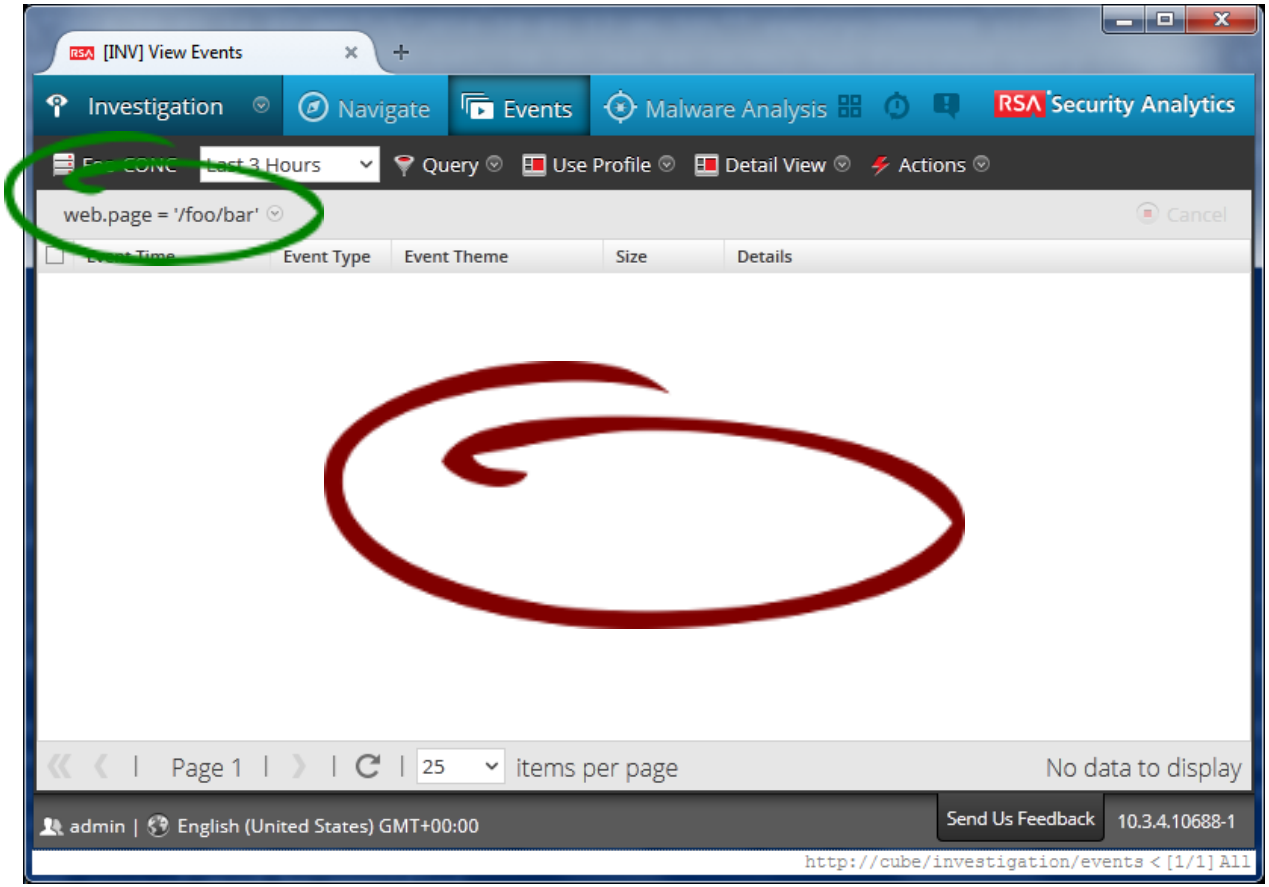
NetWitness Log Decoder

Meta Query Issues



web.page meta is available…

Meta Query Issues



… but we cannot use it in a query

Meta Query Issues

If a meta is available (i.e. we can see the meta value in Concentrator's Investigation → Events page) but we cannot use it in a query to filter results in Concentrator, then it is not a Log Decoder issue.

Log Decoder parsed the message and provided the meta. Log Decoder's job is done. This is a Concentrator indexing issue that needs to be fixed on Concentrator.

Meta Query Issues

The metas to be indexed are defined in /etc/netwitness/ng/index-concentrator.xml.

Custom indexing may be defined in /etc/netwitness/ng/index-concentrator-custom.xml.

If there is no indexing defined for a meta key, or
If the index level for a meta key is defined as 'IndexNone', then the meta is NOT indexed.

If the index level for a meta key is defined as 'IndexKeys' or 'IndexValues', then the meta is indexed.

Is index level defined as 'IndexKeys' or 'IndexValues' for the meta in /etc/netwitness/ng/index-concentrator.xml or /etc/netwitness/ng/index-concentrator-custom.xml?

Meta Query Issues

Is indexing enabled for the meta in /etc/netwitness/ng/index-concentrator.xml?

```
# grep 'name="web.page"' /etc/netwitness/ng/index-concentrator.xml
#
```

Is indexing enabled for the meta in /etc/netwitness/ng/index-concentrator-custom.xml?

```
# grep 'name="web.page"' /etc/netwitness/ng/index-concentrator-
custom.xml
grep: /etc/netwitness/ng/index-concentrator-custom.xml: No such file or
directory
```

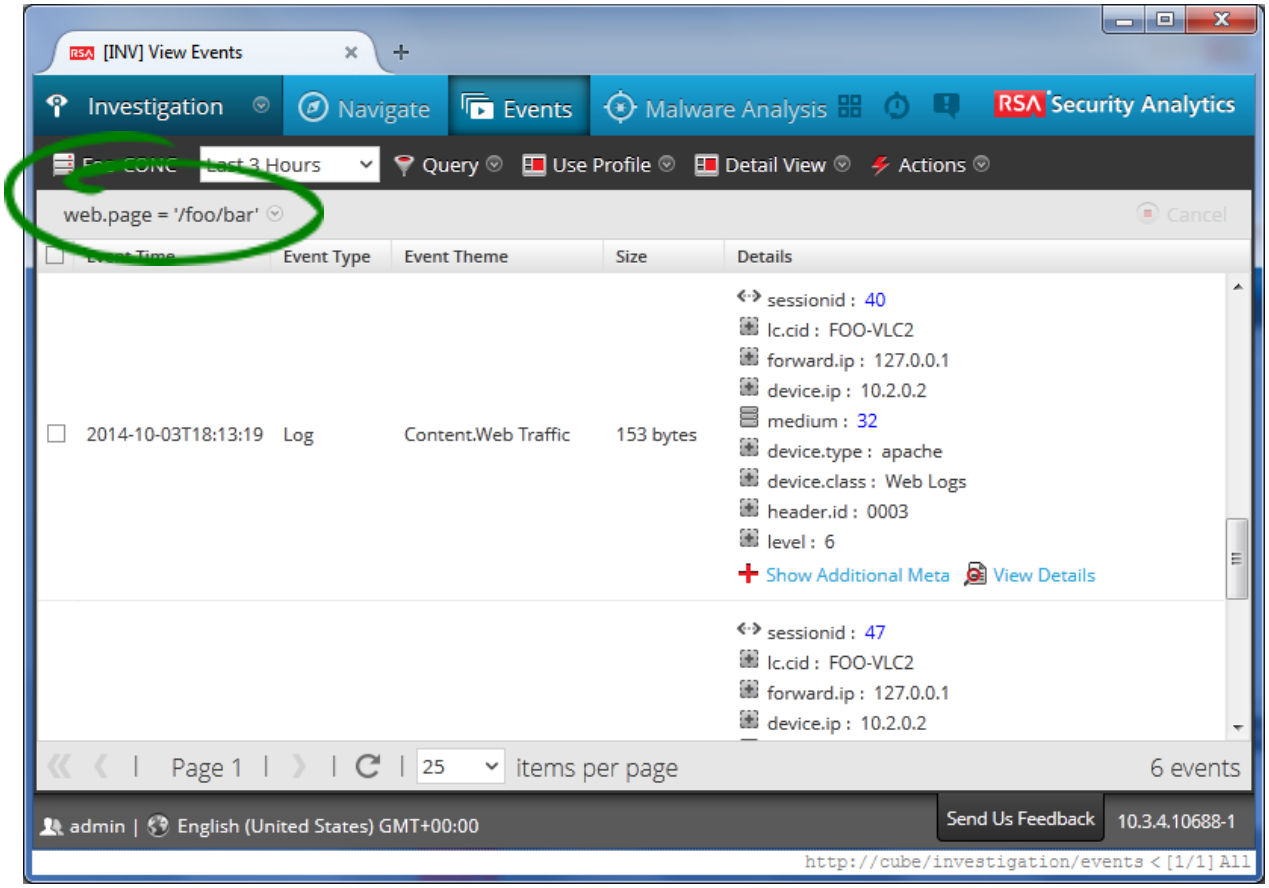
No, indexing is not enabled for the meta.

Meta Query Issues

We can enable indexing for a meta by adding a definition for it in etc/netwitness/ng/index-concentrator-custom.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<language level="IndexNone" defaultAction="Auto">
    <key description="Web Page" level="IndexValues" name="web.page"
         format="Text" valueMax="10000" />
</language>
```

Meta Query Issues



Querying web.page meta works after defining its index level

Log Forwarding

NetWitness Log Decoder

Log Forwarding

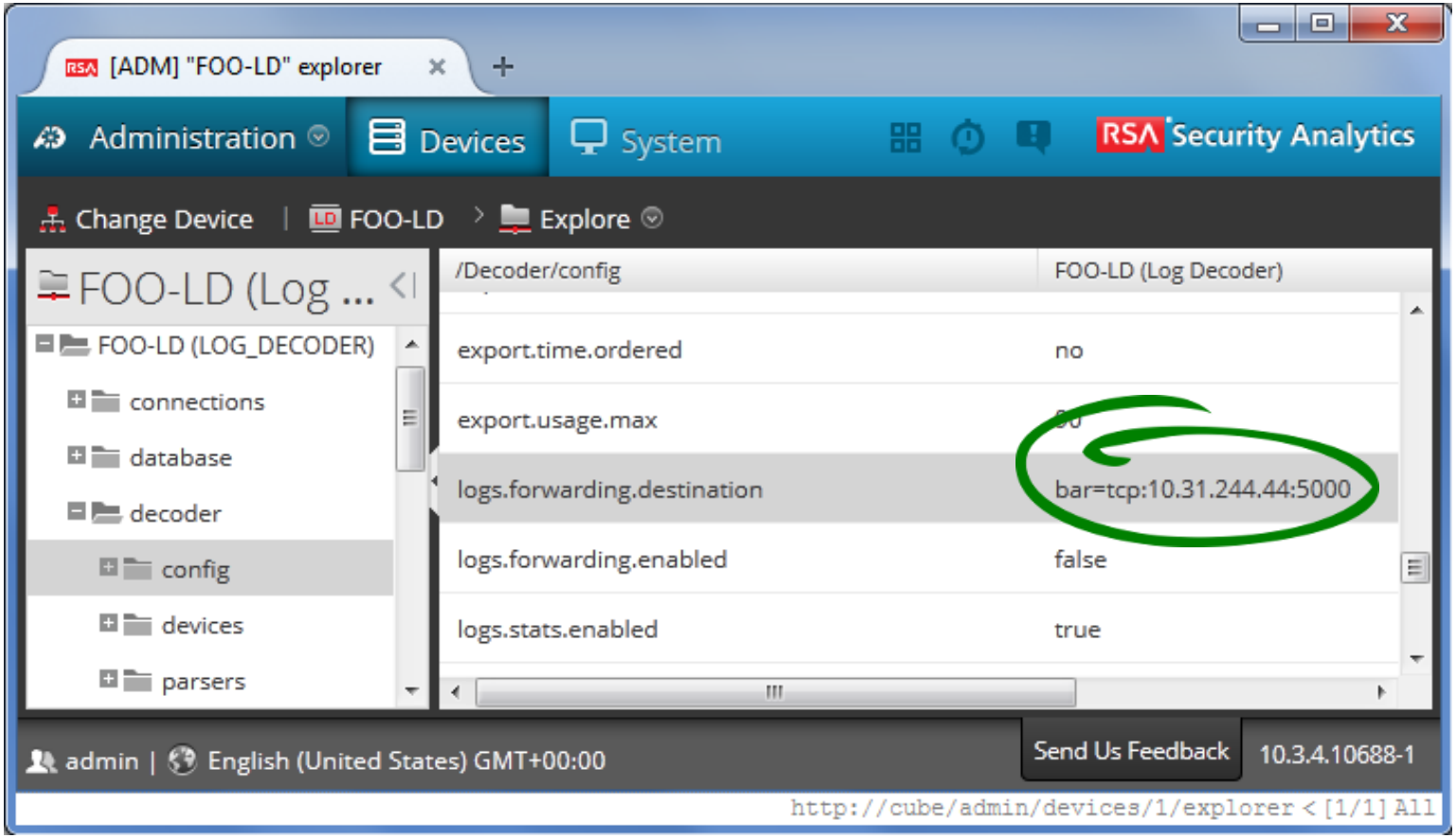
Since 10.3.2, Log Decoder can forward logs that it ingests to another destination.

Log Forwarding is done after the logs are parsed and before they are written to the disk.

Three steps must be performed to enable log forwarding:

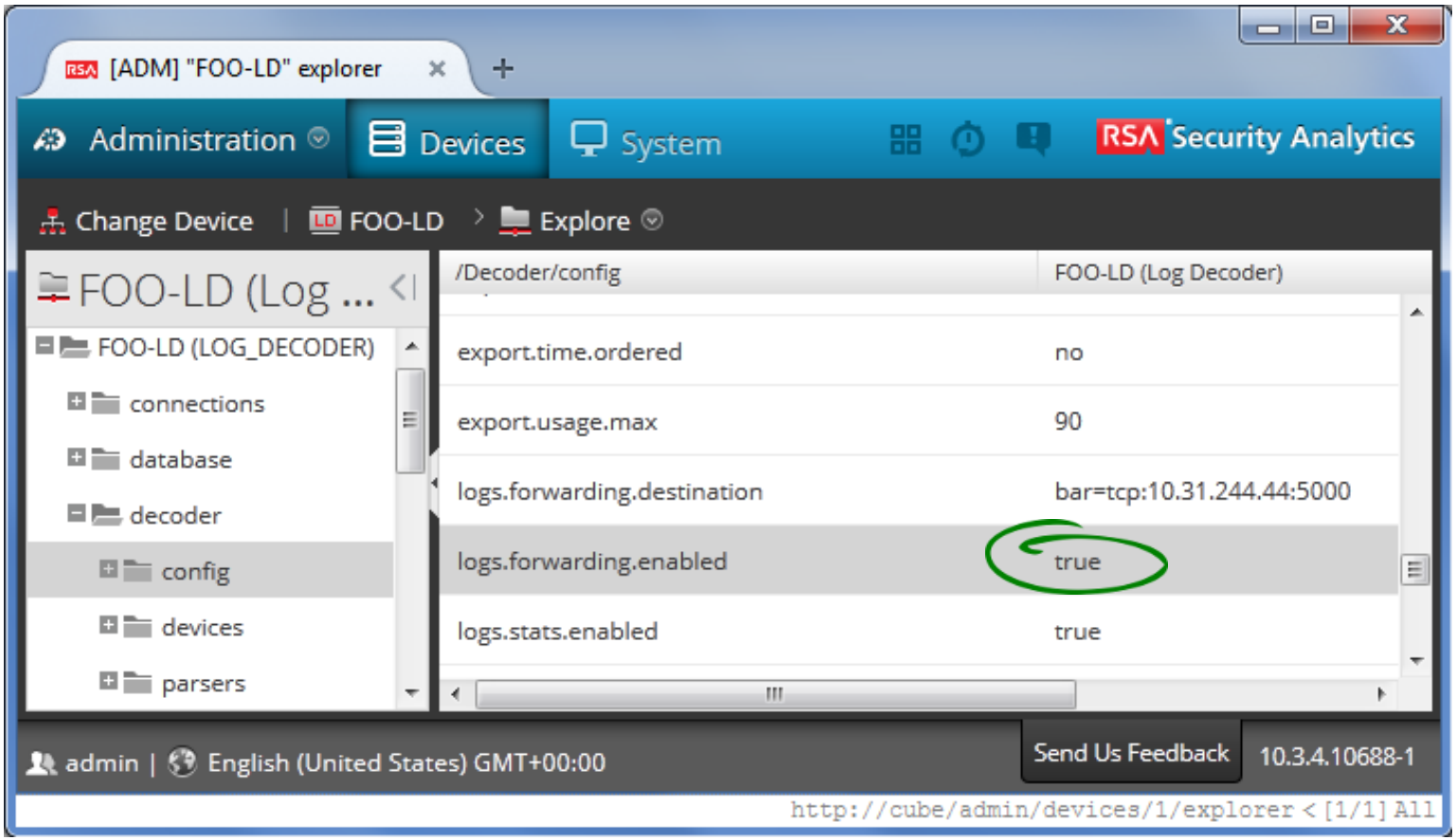
1. Define a destination
2. Enable syslog forwarding
3. Define an application rule

Log Forwarding



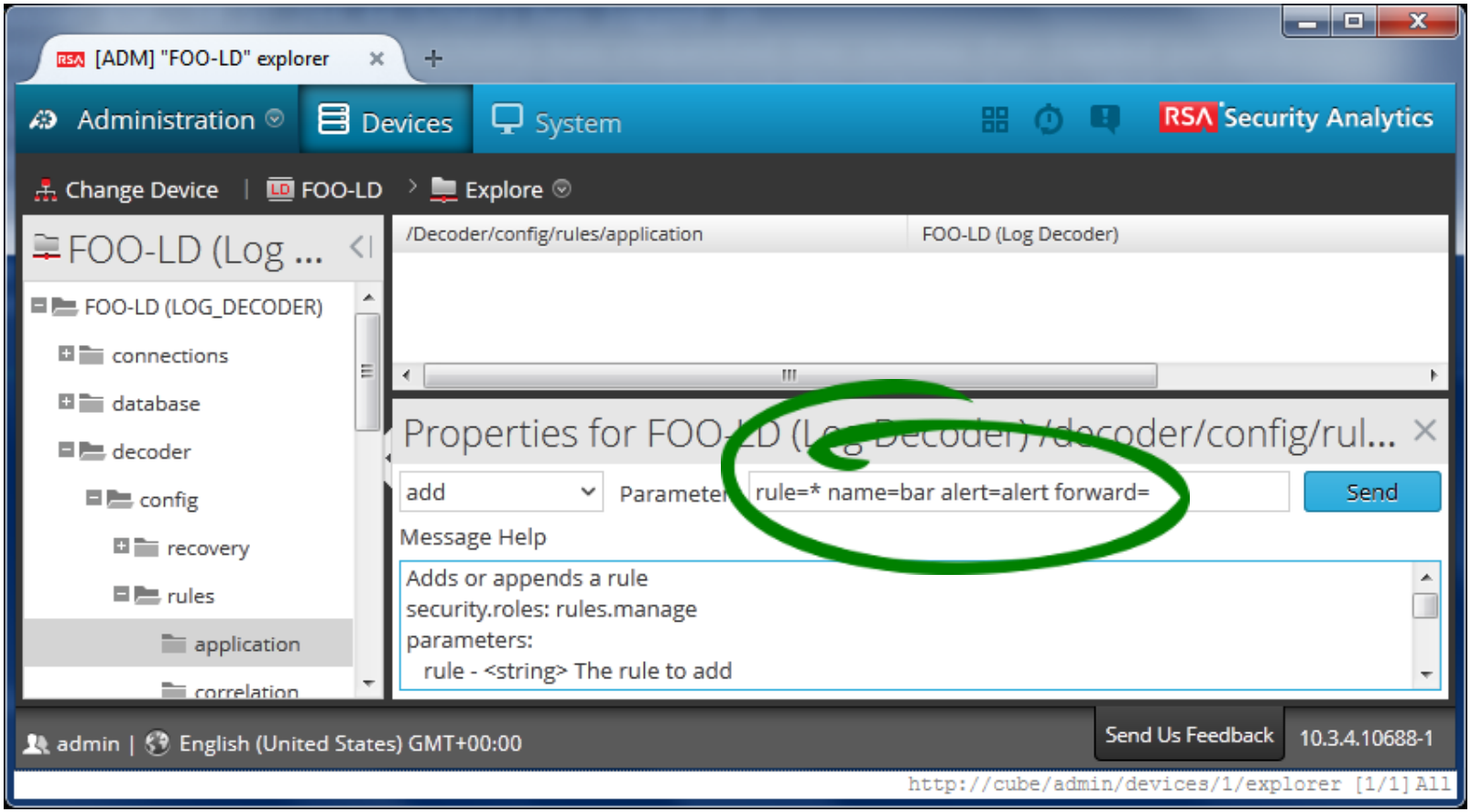
Step 1: Define a destination

Log Forwarding



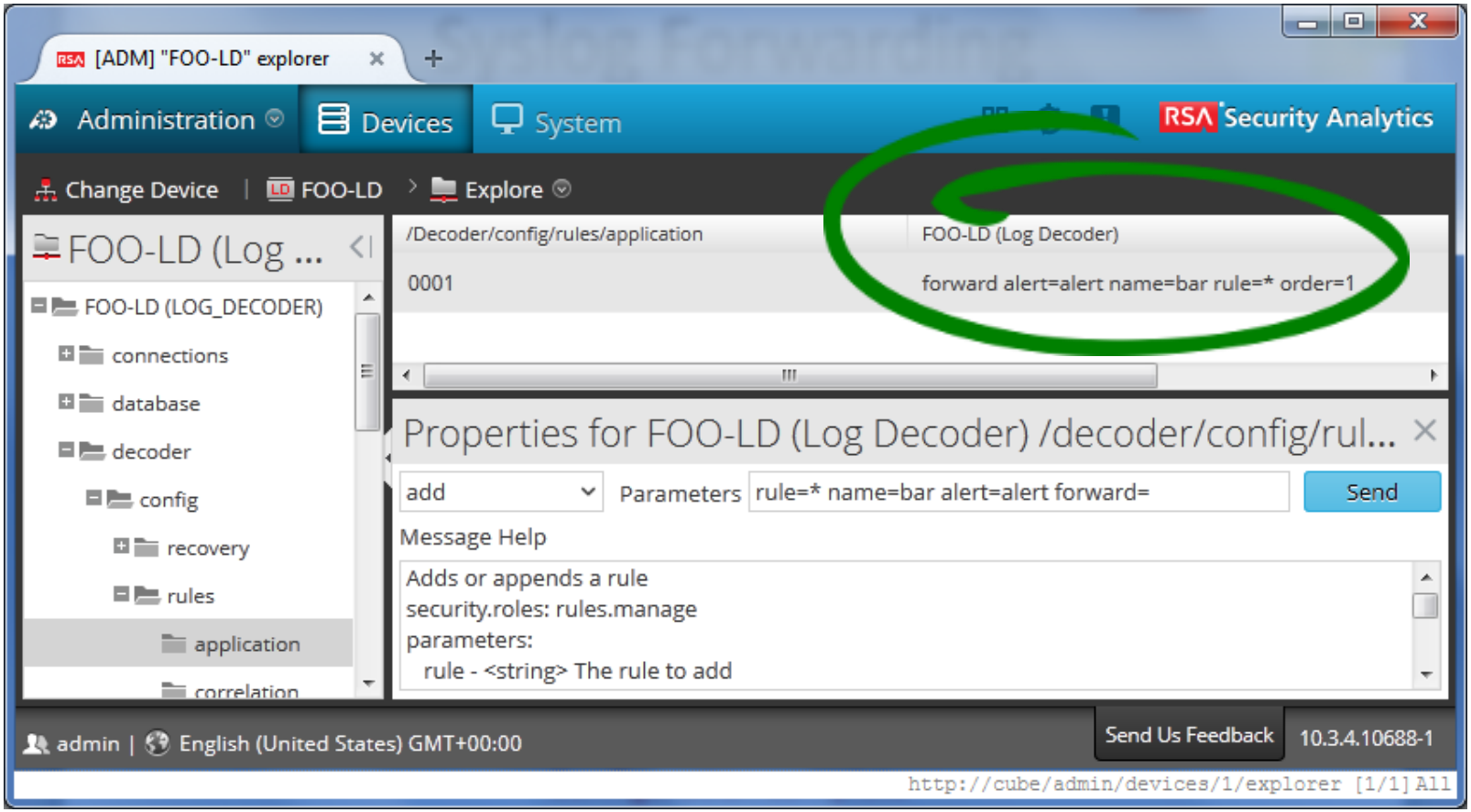
Step 2: Enable syslog forwarding

Log Forwarding



Step 3: Add an application rule

Log Forwarding



Configuration complete

Log Forwarding

We feed two Cisco Pix logs to Log Decoder.

```
# echo "<1> %PIX-1-101001: (PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "<1> %PIX-1-101001: (PRIORITY) Bad failover cable." > /dev/tcp/127.0.0.1/514
```

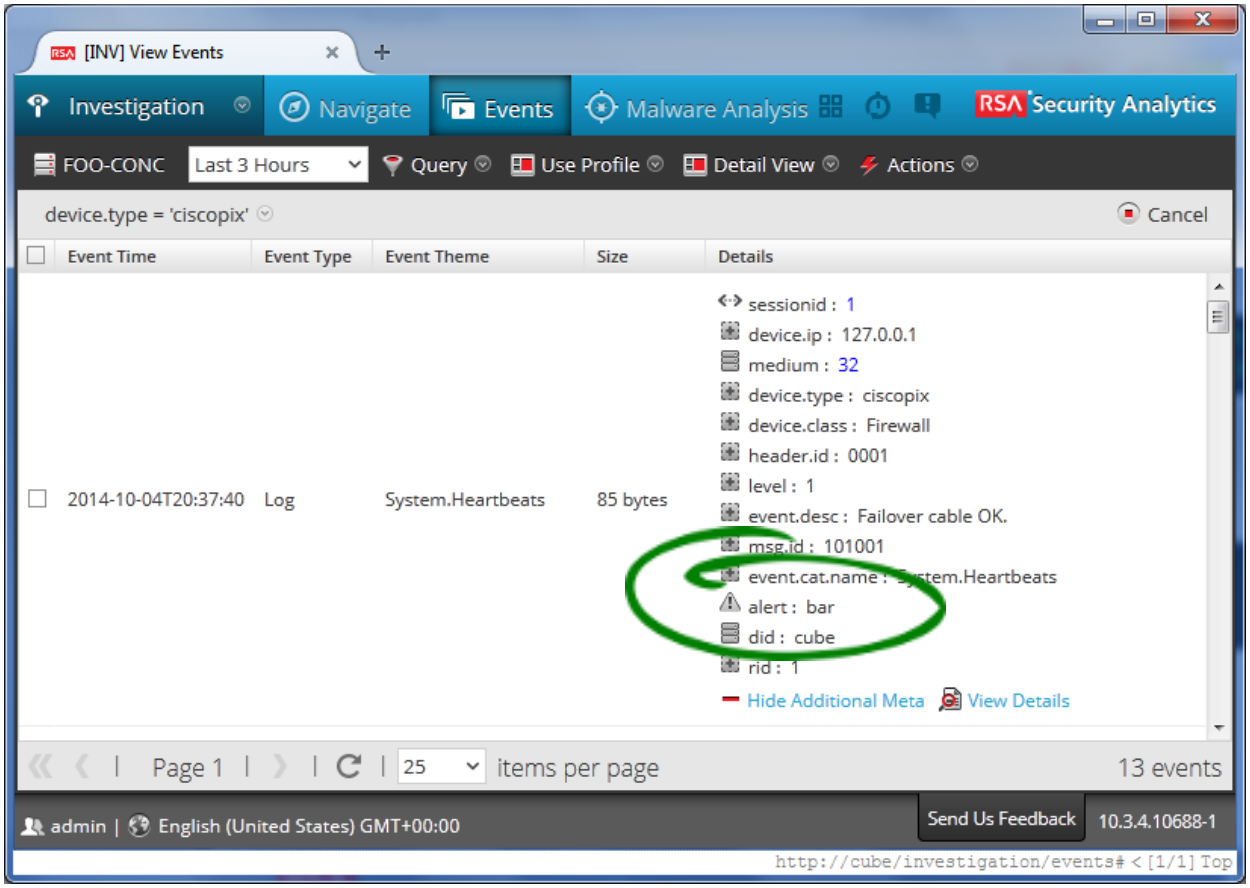
(The above command is run on Log Decoder.)

We see both logs forwarded to the destination

```
# nc -kl 5000
47 <1>%PIX-1-101001: (PRIORITY) Failover cable OK.
48 <1>%PIX-1-101001: (PRIORITY) Bad failover cable.
```

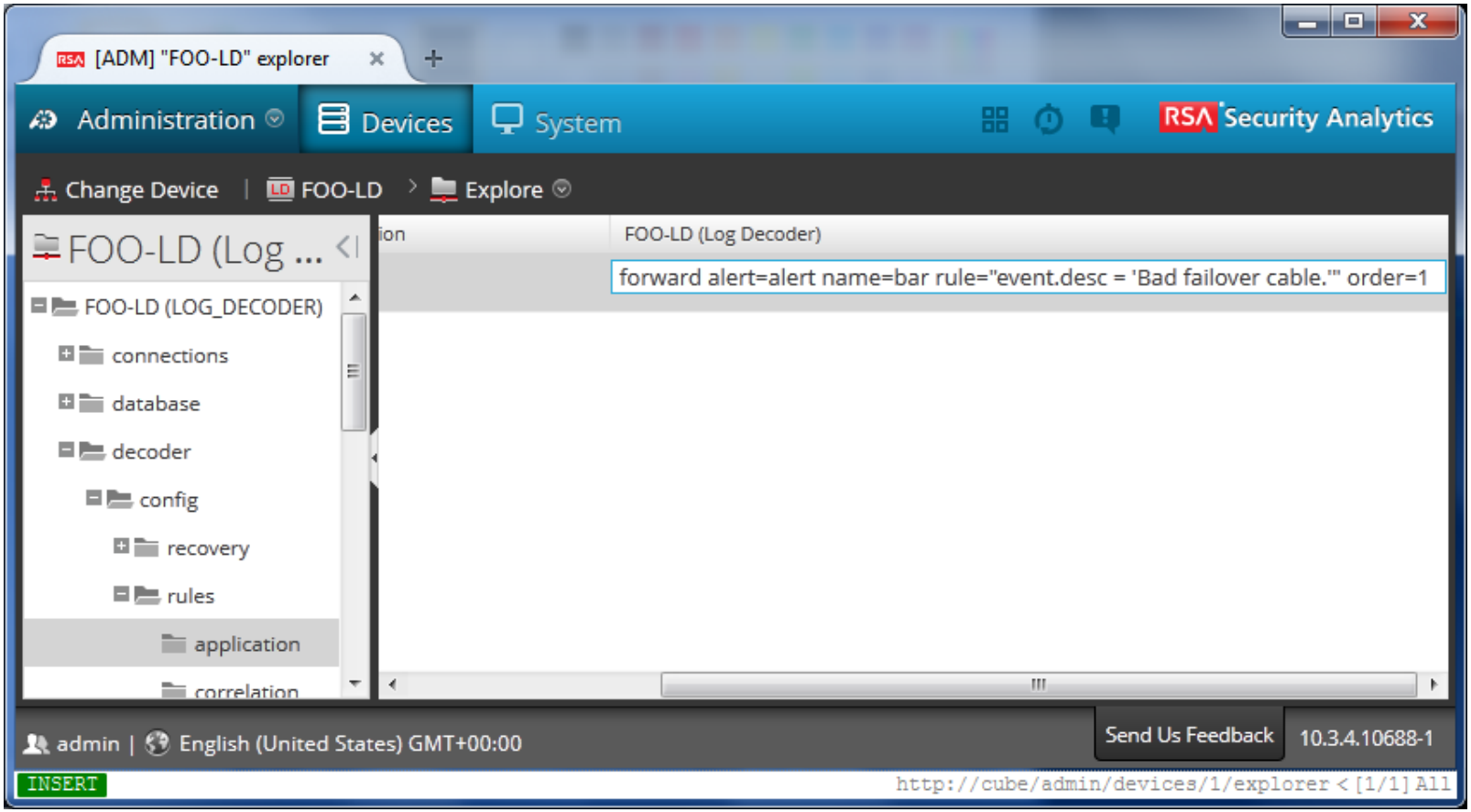
(The above netcat command is run on the destination. It listens on TCP port 5000 and prints all payloads it receives on this port.)

Log Forwarding



Log forwarding has led to the creation of a new meta called ‘alert’

Log Forwarding



Forward logs with event.desc = ‘Bad failover cable.’

Log Forwarding

We feed two Cisco Pix logs to Log Decoder.

```
# echo "<1> %PIX-1-101001: (PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "<1> %PIX-1-101001: (PRIORITY) Bad failover cable." > /dev/tcp/127.0.0.1/514
```

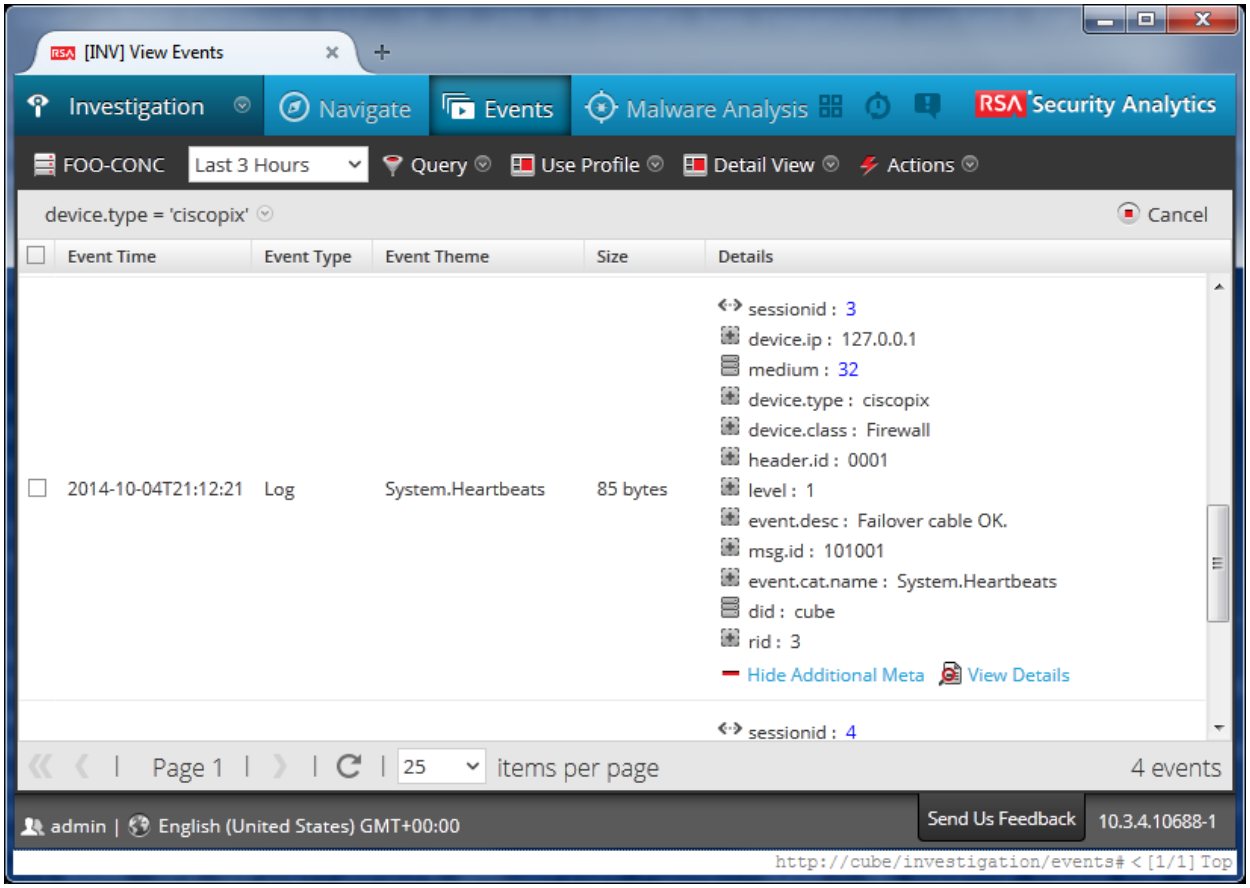
(The above command is run on Log Decoder.)

We see only one log forwarded to the destination, the one that matches the rule:
`event.desc = 'Bad failover cable.'`

```
# nc -kl 5000
48 <1>%PIX-1-101001: (PRIORITY) Bad failover cable.
```

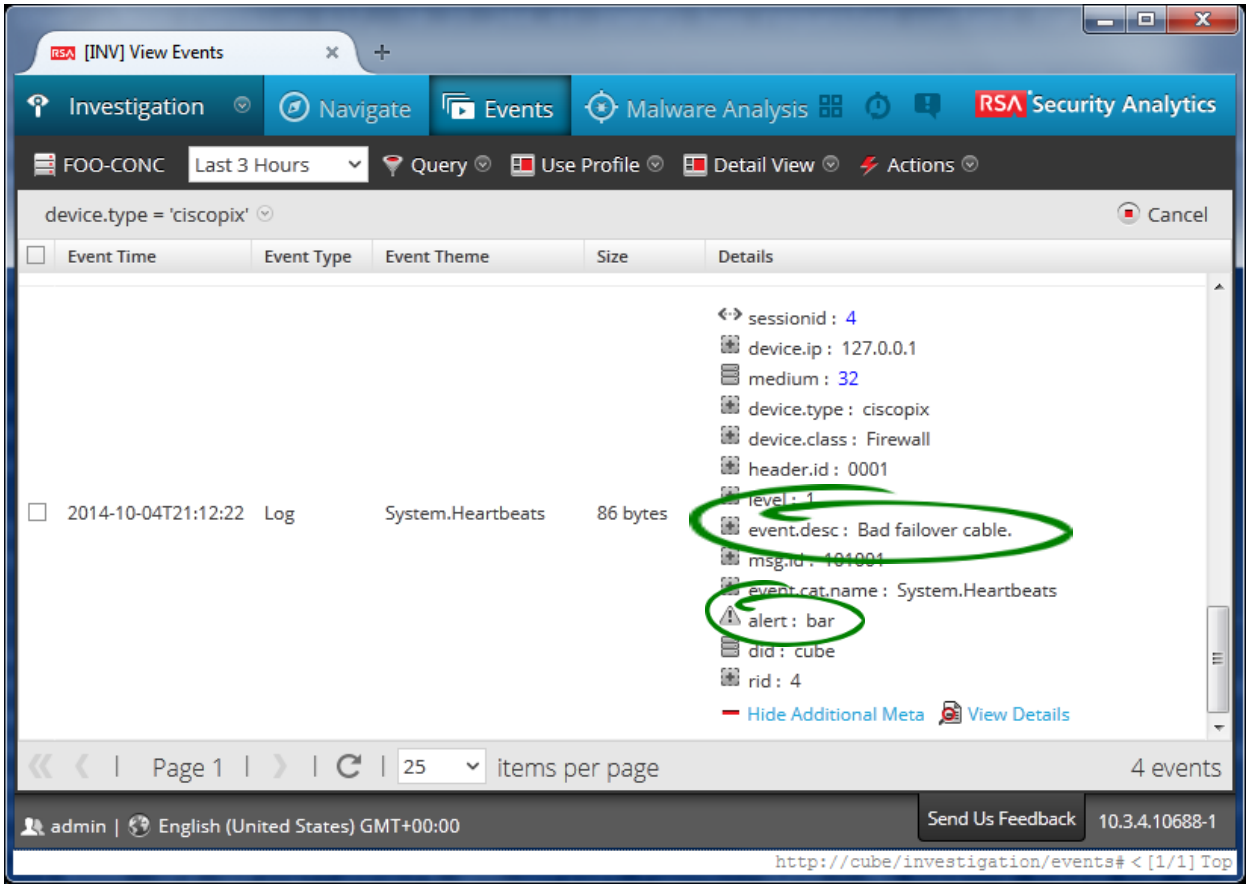
(The above netcat command is run on the destination. It listens on TCP port 5000 and prints all payloads it receives on this port.)

Log Forwarding



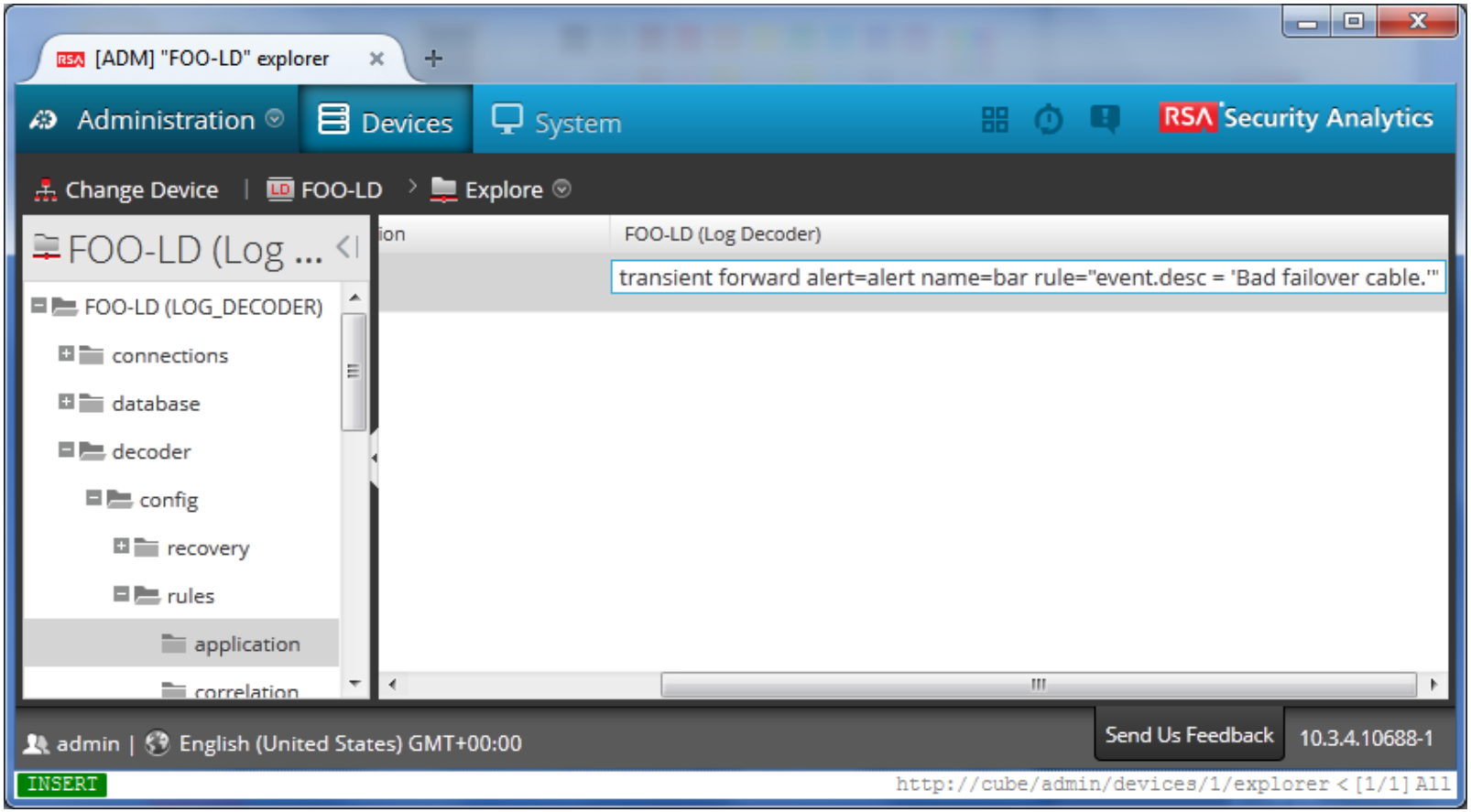
Now there is no ‘alert’ meta for the log that did not match the rule

Log Forwarding



Now there is ‘alert’ meta only for the log that matched the rule

Log Forwarding



Use ‘transient’ flag, so that the meta is not written to the disk

Log Forwarding

We feed two Cisco Pix logs to Log Decoder.

```
# echo "<1> %PIX-1-101001: (PRIORITY) Failover cable OK." > /dev/tcp/127.0.0.1/514
# echo "<1> %PIX-1-101001: (PRIORITY) Bad failover cable." > /dev/tcp/127.0.0.1/514
```

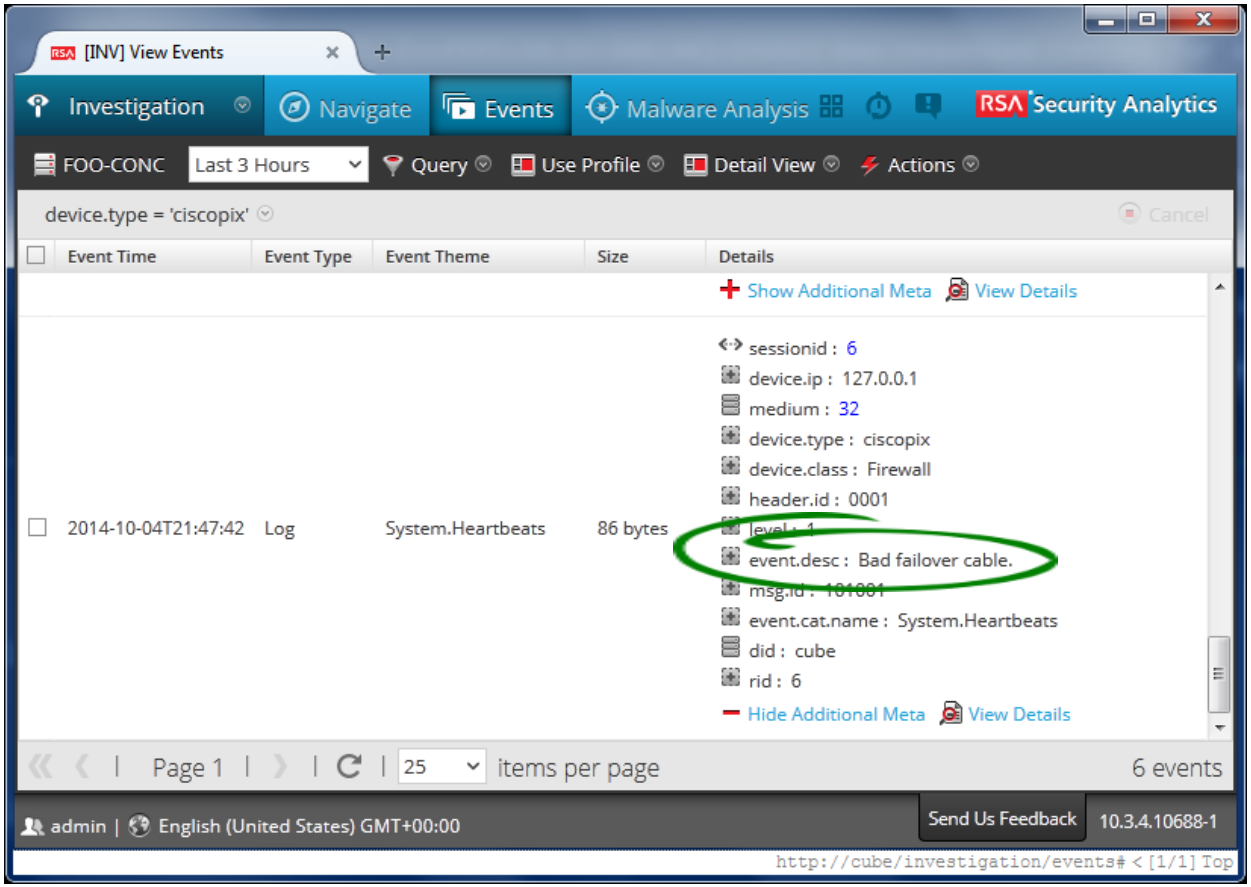
(The above command is run on Log Decoder.)

Just like before, we see only one log forwarded to the destination, the one that matches the rule: `event.desc = 'Bad failover cable.'`

```
# nc -kl 5000
48 <1>%PIX-1-101001: (PRIORITY) Bad failover cable.
```

(The above netcat command is run on the destination. It listens on TCP port 5000 and prints all payloads it receives on this port.)

Log Forwarding



No ‘alert’ meta for the log that matched the rule because it had ‘transient’ flag set

Log Forwarding Issues

NetWitness Log Decoder

Log Forwarding Issues

Check if the alert meta is being generated for the logs that we expect to be forwarded.

Note: If there is 'transient' flag in the application rule, then the alert meta won't be written to the disk, and thus cannot be seen in the Investigation module in the UI.

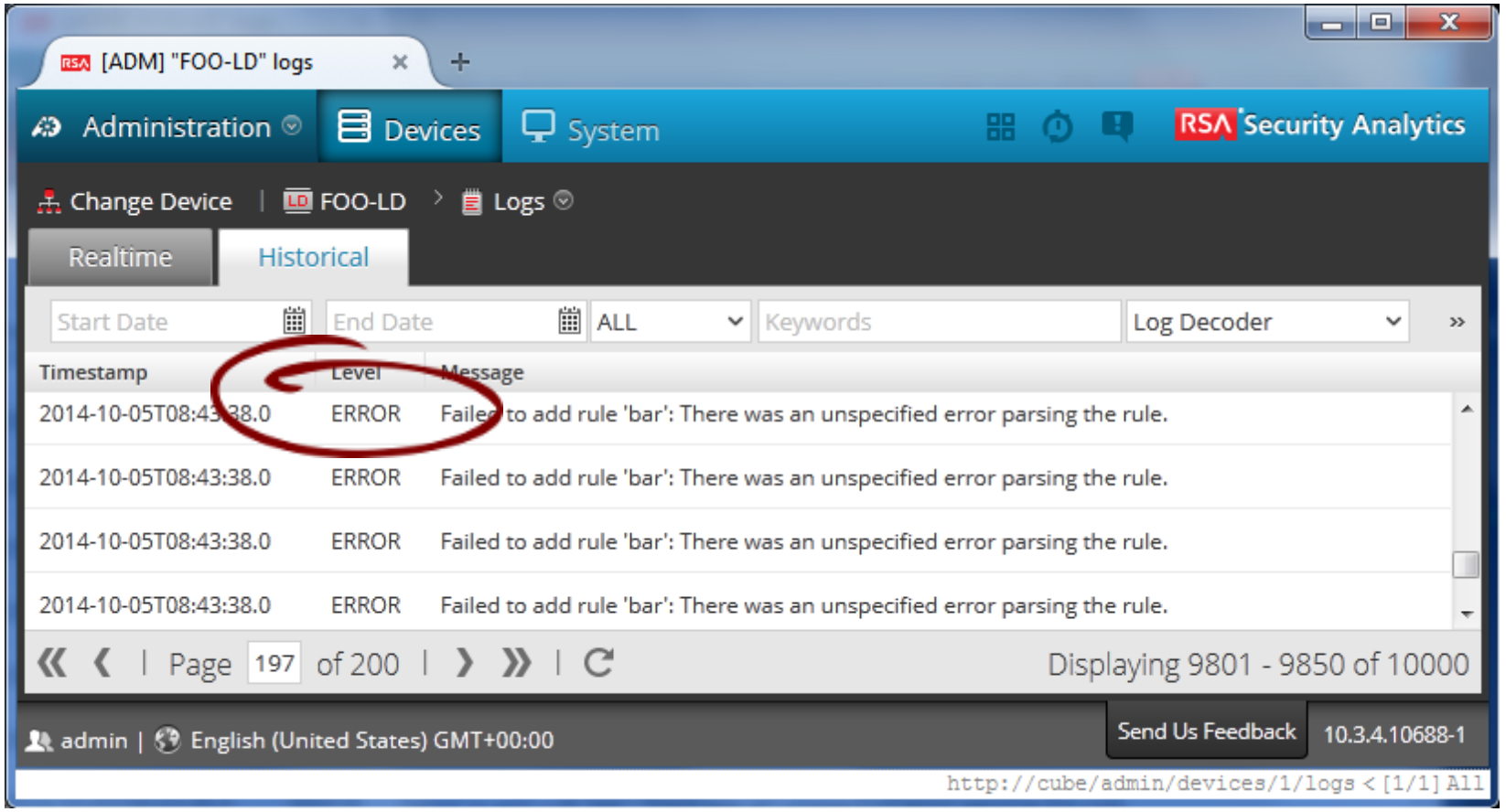
Therefore, to ease troubleshooting, remove 'transient' flag (if it is present) and check if the alert meta is being generated for the logs that we expect to be forwarded.

Log Forwarding Issues

If the ‘alert’ meta is not being generated for the logs that we expect to be forwarded, then there is an issue with the rule.

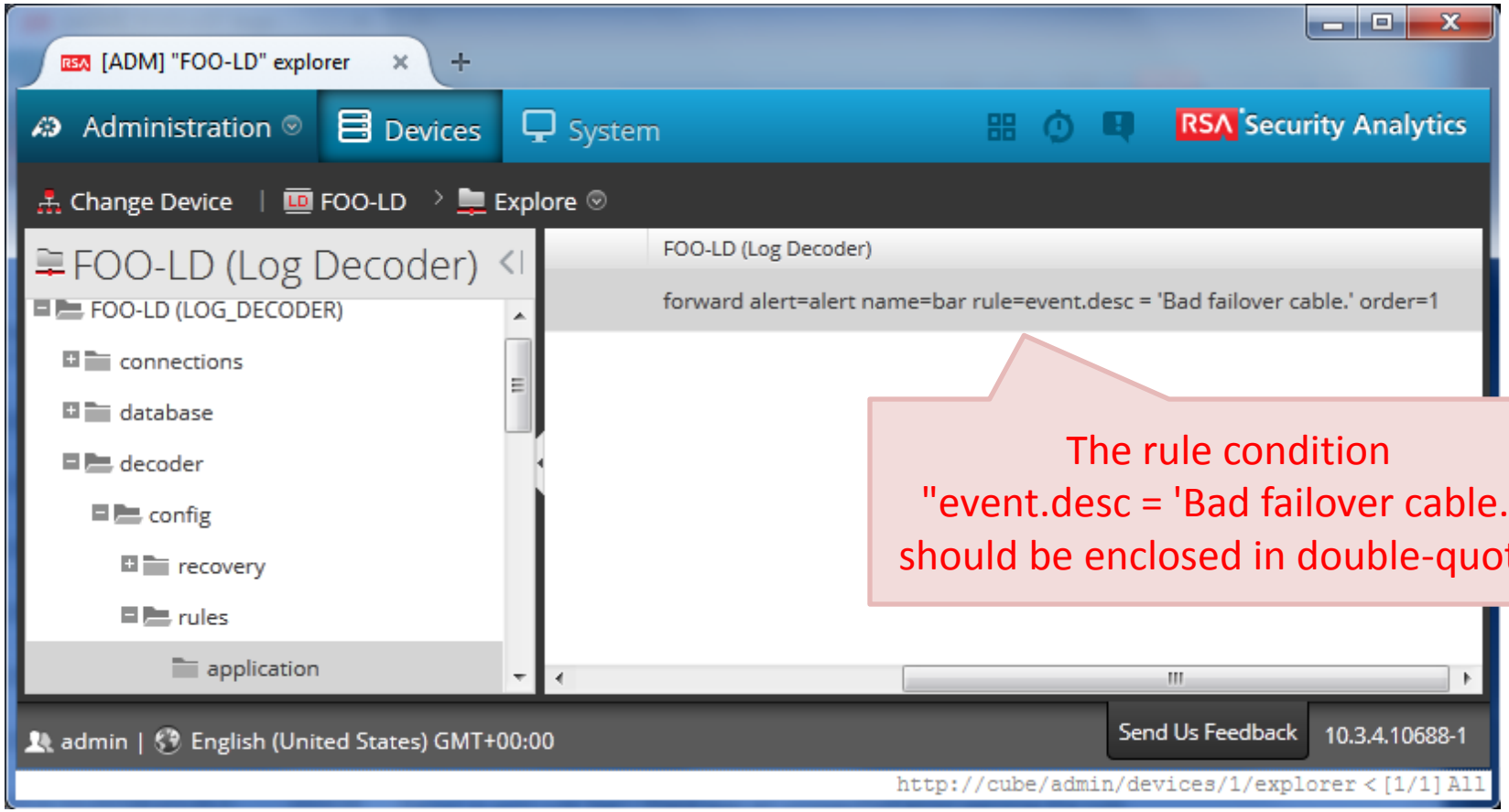
1. Check the logs.
2. Check the rule.

Log Forwarding Issues



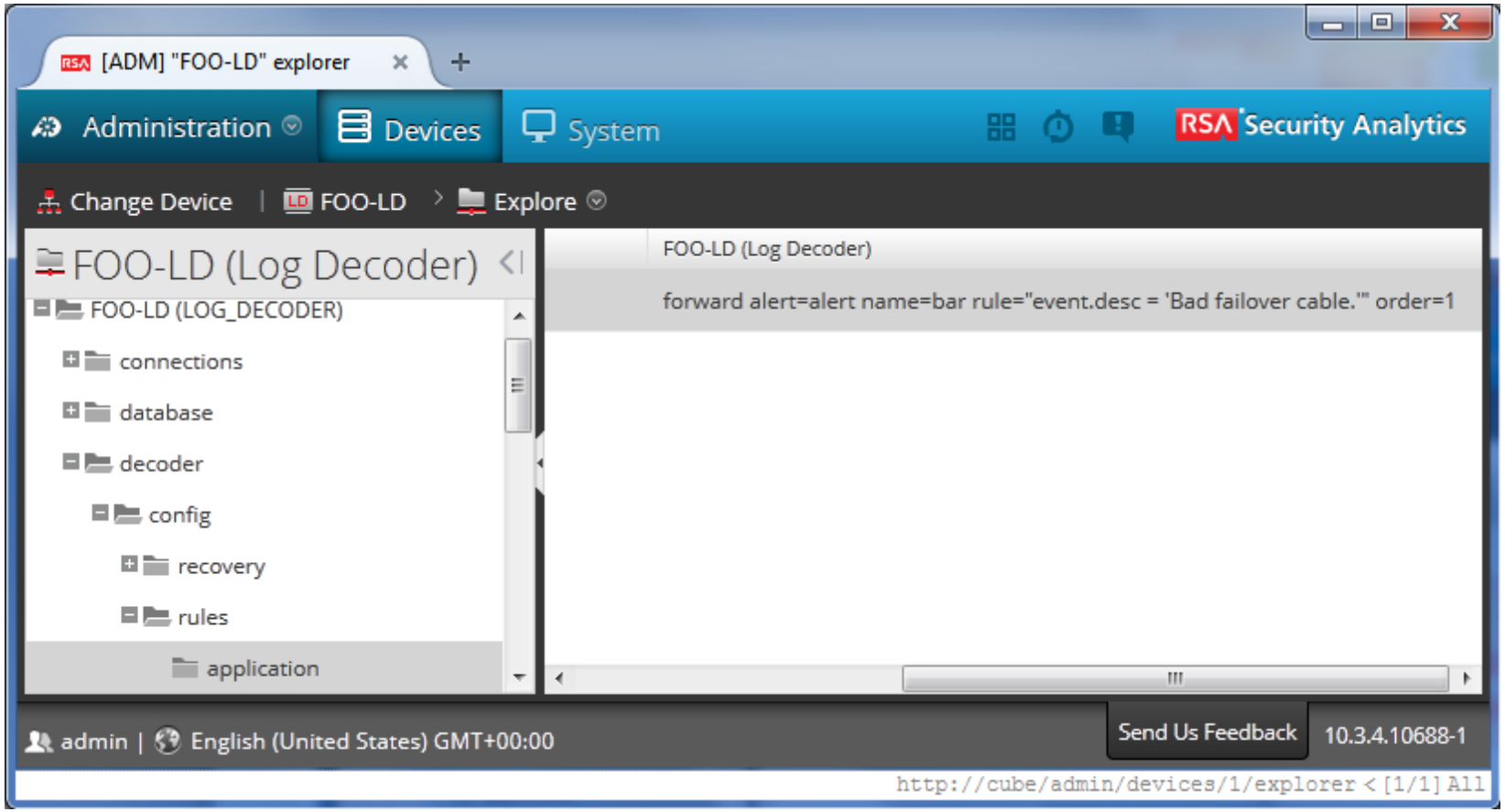
Check the logs

Log Forwarding Issues



Check the rule

Log Forwarding Issues



Fix the rule

Log Forwarding Issues

One of the most common errors encountered in the field: The 'alert' flag is not set.

Both 'forward' and 'alert' flags must be set.

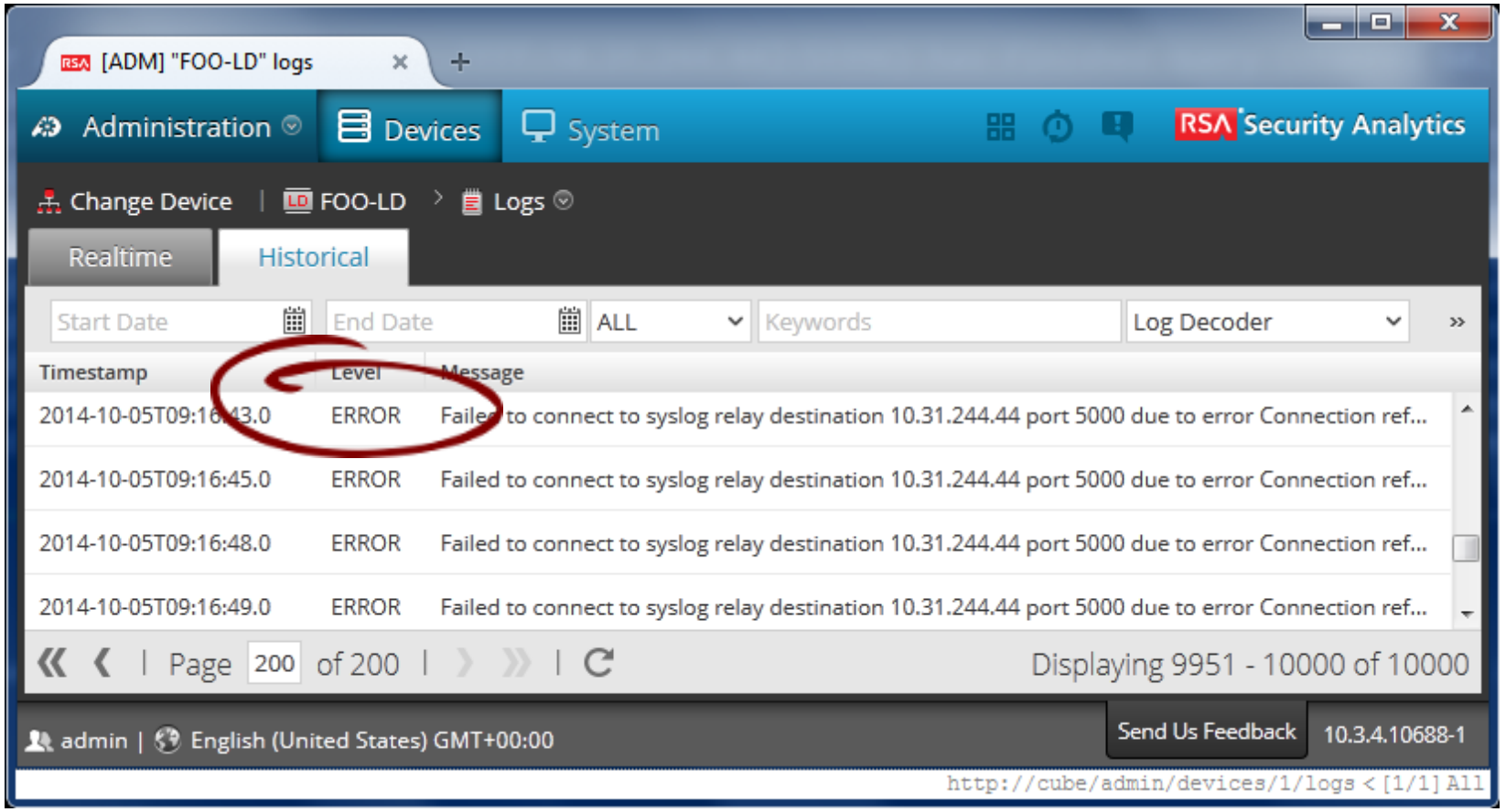
1. The 'alert' flag is necessary to generate the 'alert' meta that is required for forwarding rules.
2. The 'forward' flag is necessary to set the 'forward' flag on any log that matches the rule condition.

Log Forwarding Issues

If the rule is good, both ‘forward’ and ‘alert’ flags are set, and the ‘alert’ meta is being generated for the logs that should be forwarded, then check connectivity to the log forwarding destination.

1. Check the logs
2. Test connectivity to the destination

Log Forwarding Issues



Check the logs

Log Forwarding Issues

Test connectivity to the log forwarding destination.

In Bash, and any POSIX shell, `:` is a null command. It does nothing. Therefore, we can use the `:` command instead of the echo command if we just want to test connectivity to the log forwarding destination without sending any payload to it.

```
# : > /dev/tcp/10.31.244.44/5000
#
```

(The above command is run on Log Decoder to test connectivity to the log forwarding destination.)

If the connection could be established, then there is no output.

```
# : > /dev/tcp/10.31.244.44/5000
-bash: connect: Connection refused
-bash: /dev/tcp/10.31.244.44/5000: Connection refused
```

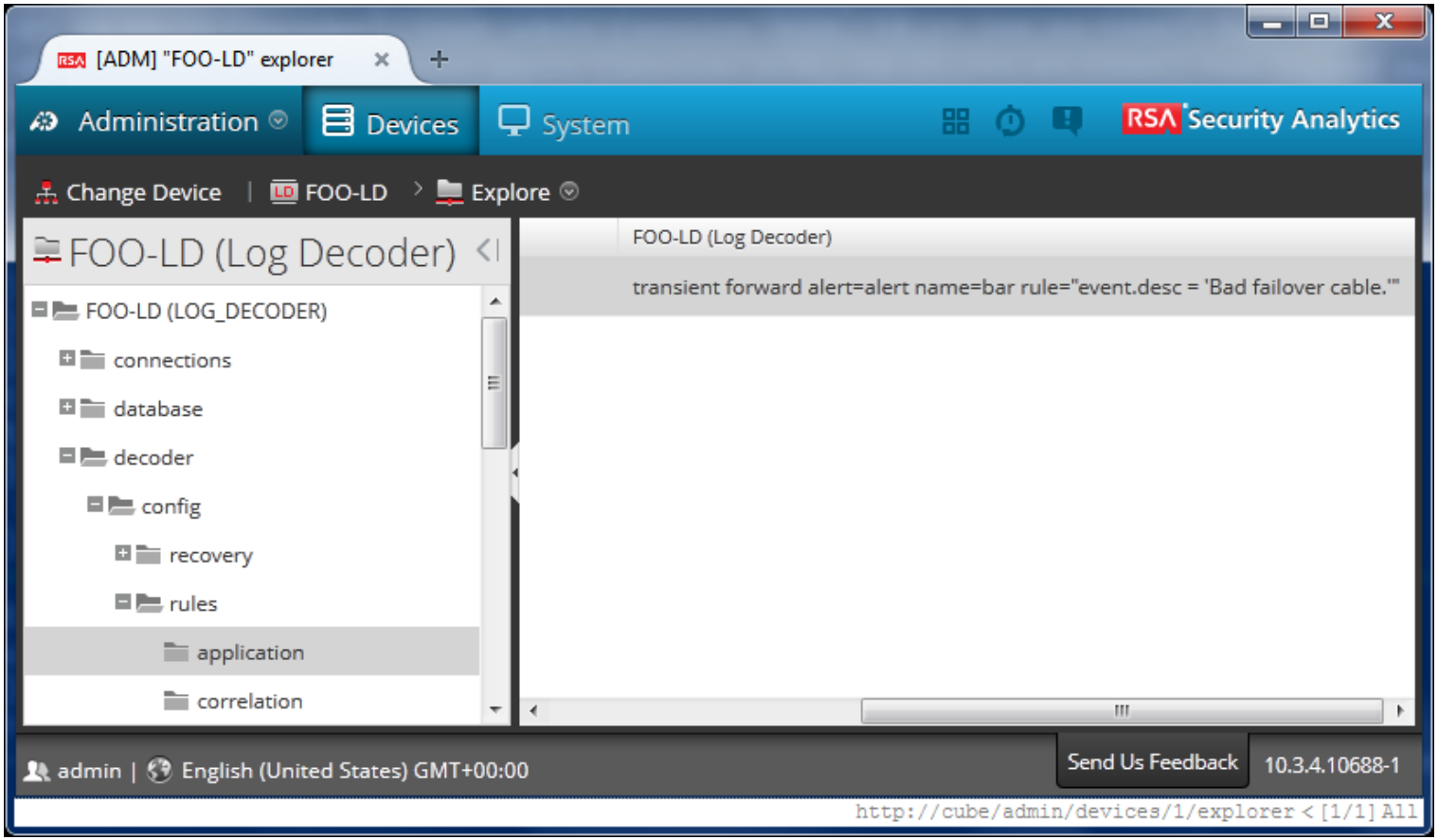
(The above command is run on Log Decoder to test connectivity to the log forwarding destination.)

If the connection could not be established, then error messages are displayed.

Log Forwarding Flag Issues

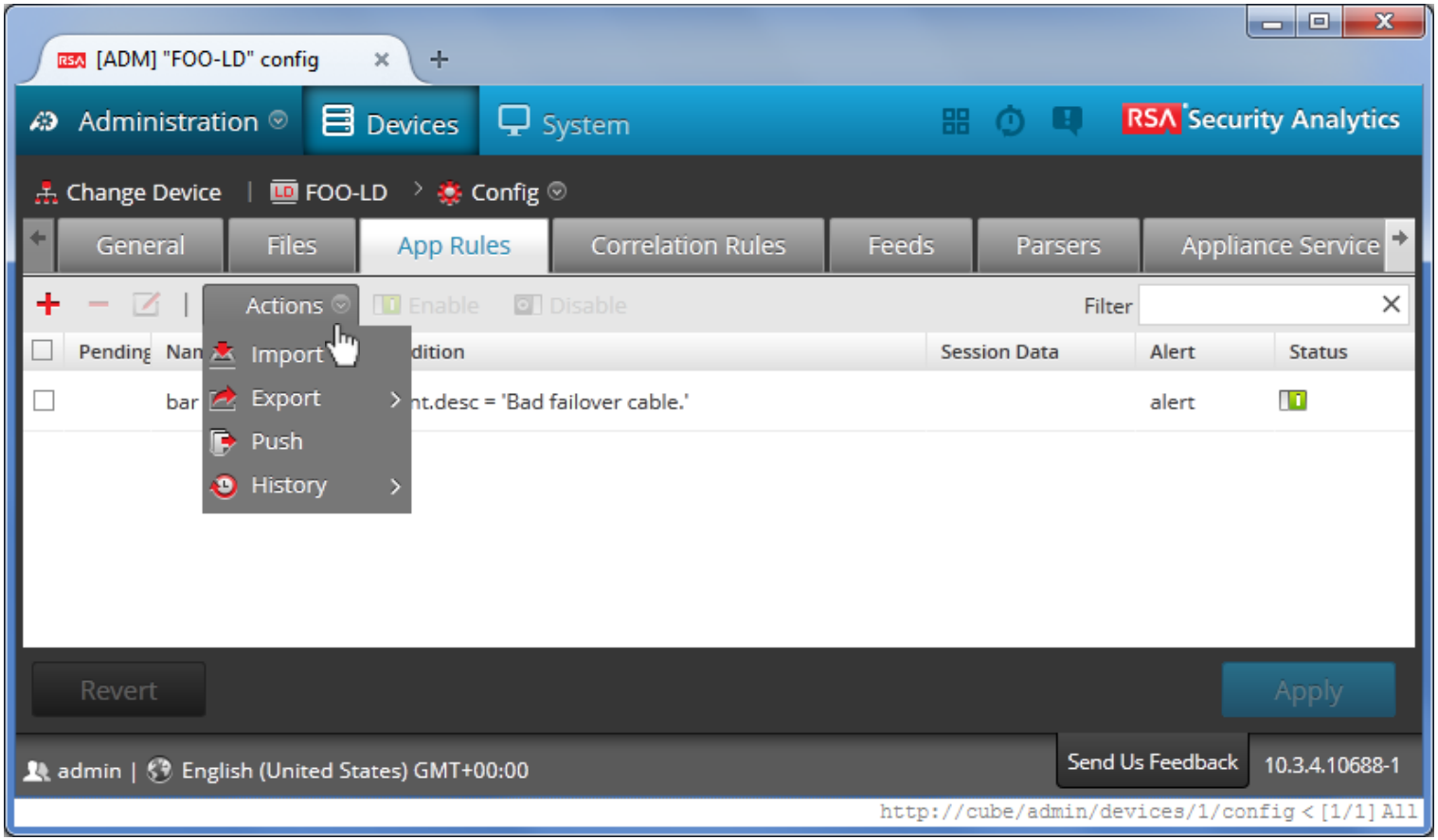
NetWitness Log Decoder

Log Forwarding Flag Issues



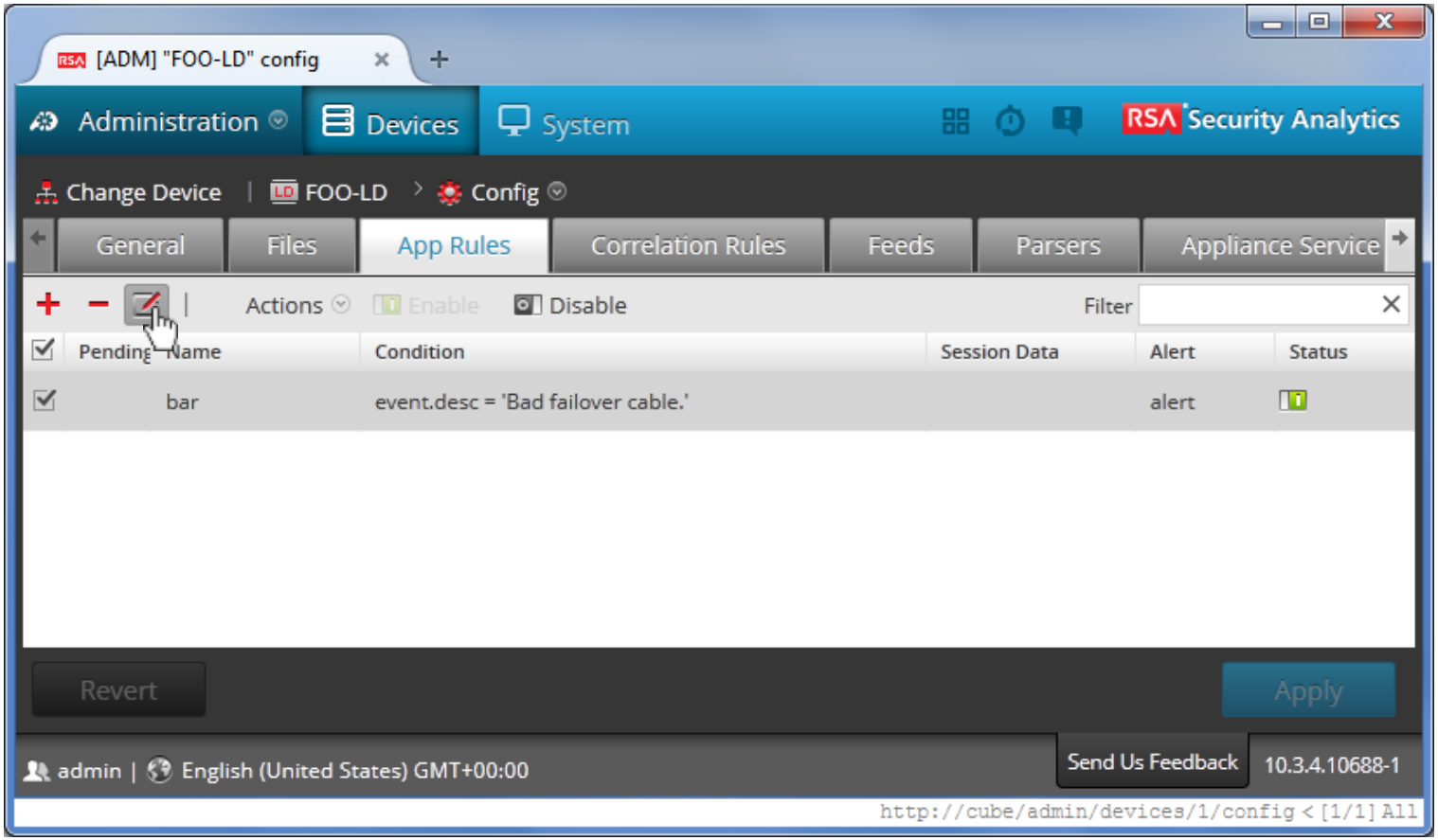
We have a properly configured log forwarding rule here …

Log Forwarding Flag Issues



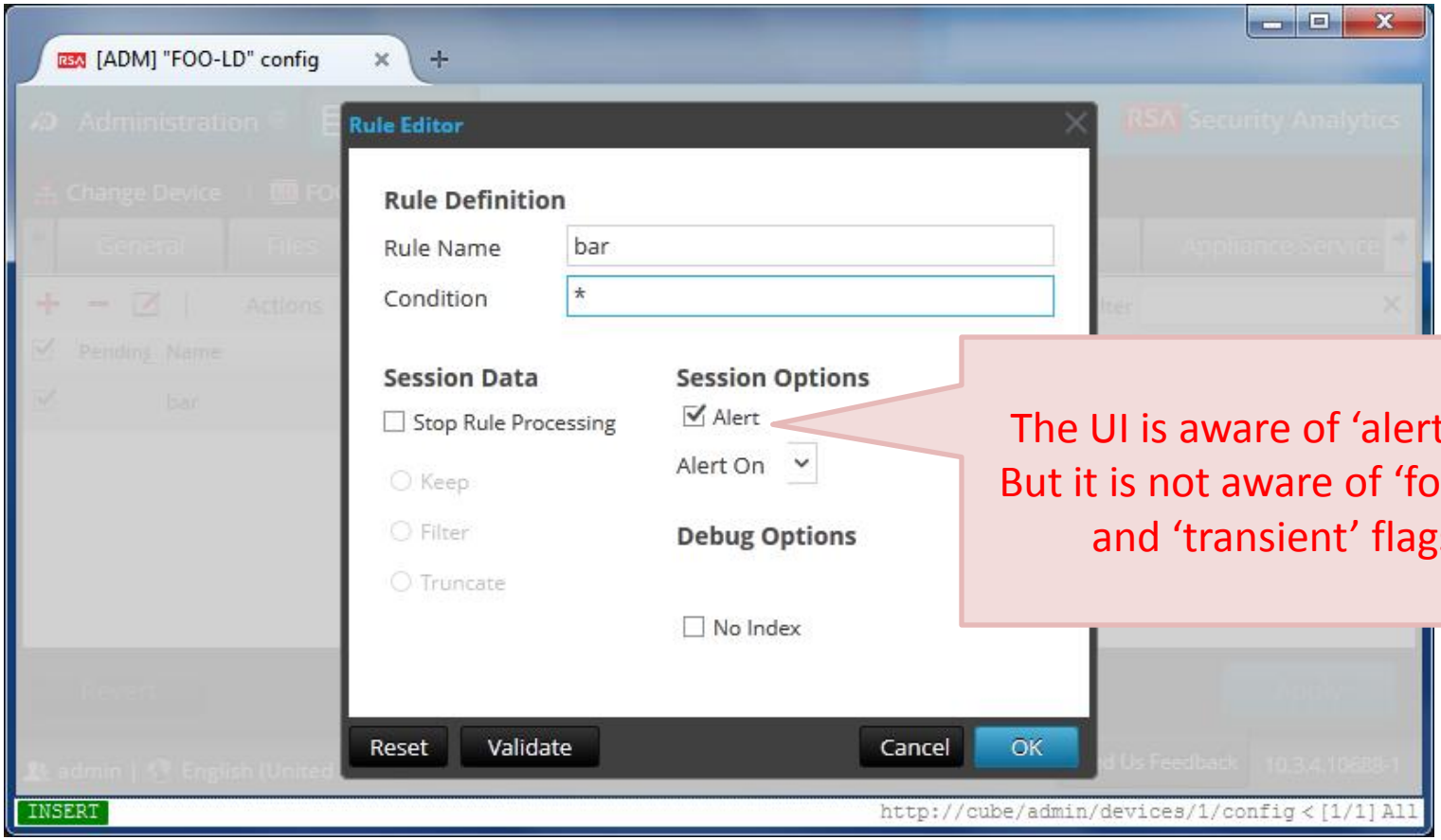
… but, in 10.3.4, if we perform any action, the ‘forward’ and ‘transient’ flags are lost

Log Forwarding Flag Issues



In 10.3.4, editing the rule also causes the ‘forward’ and ‘transient’ flags to be lost

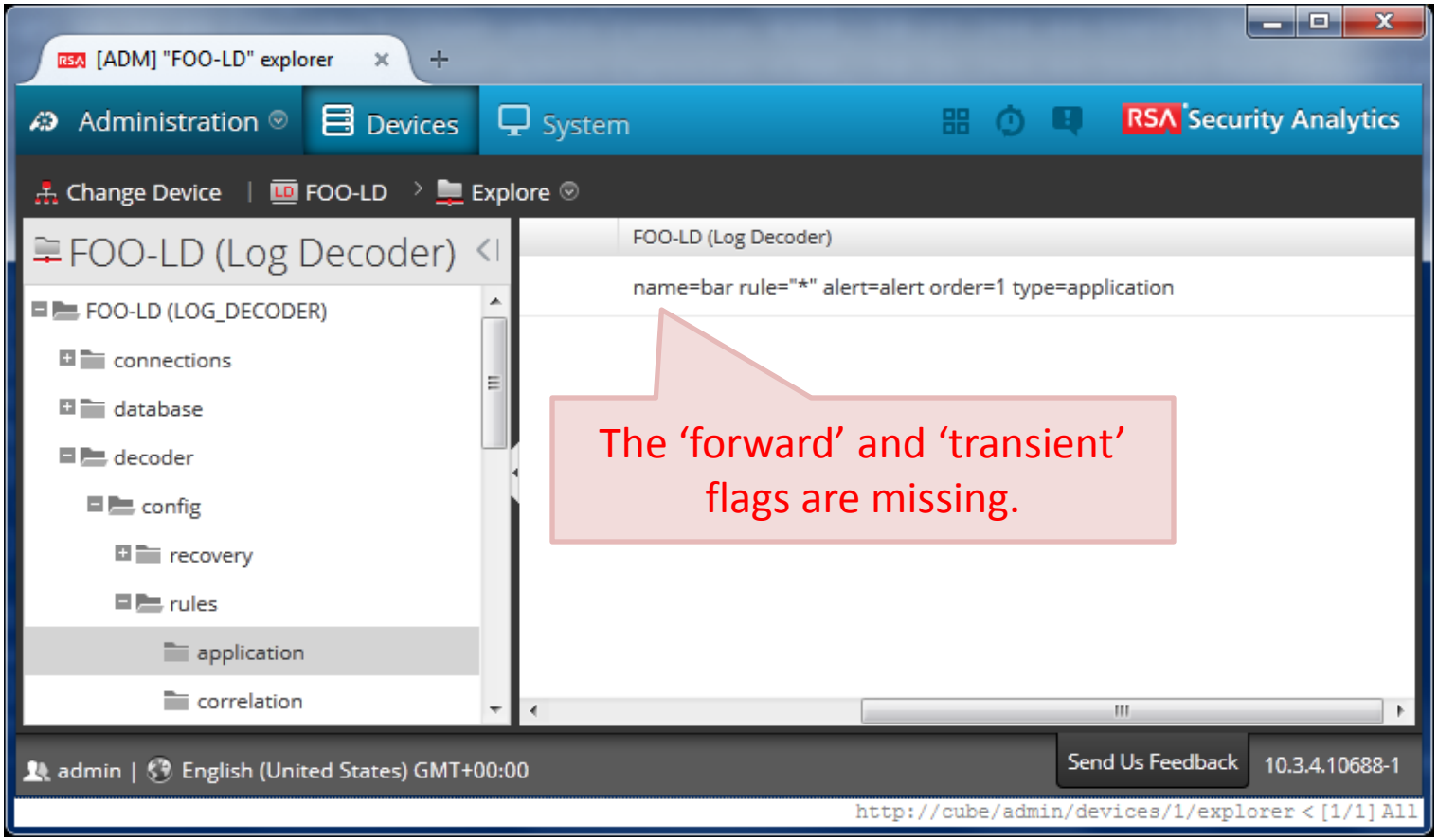
Log Forwarding Flag Issues



The UI is aware of 'alert' flag. But it is not aware of 'forward' and 'transient' flags.

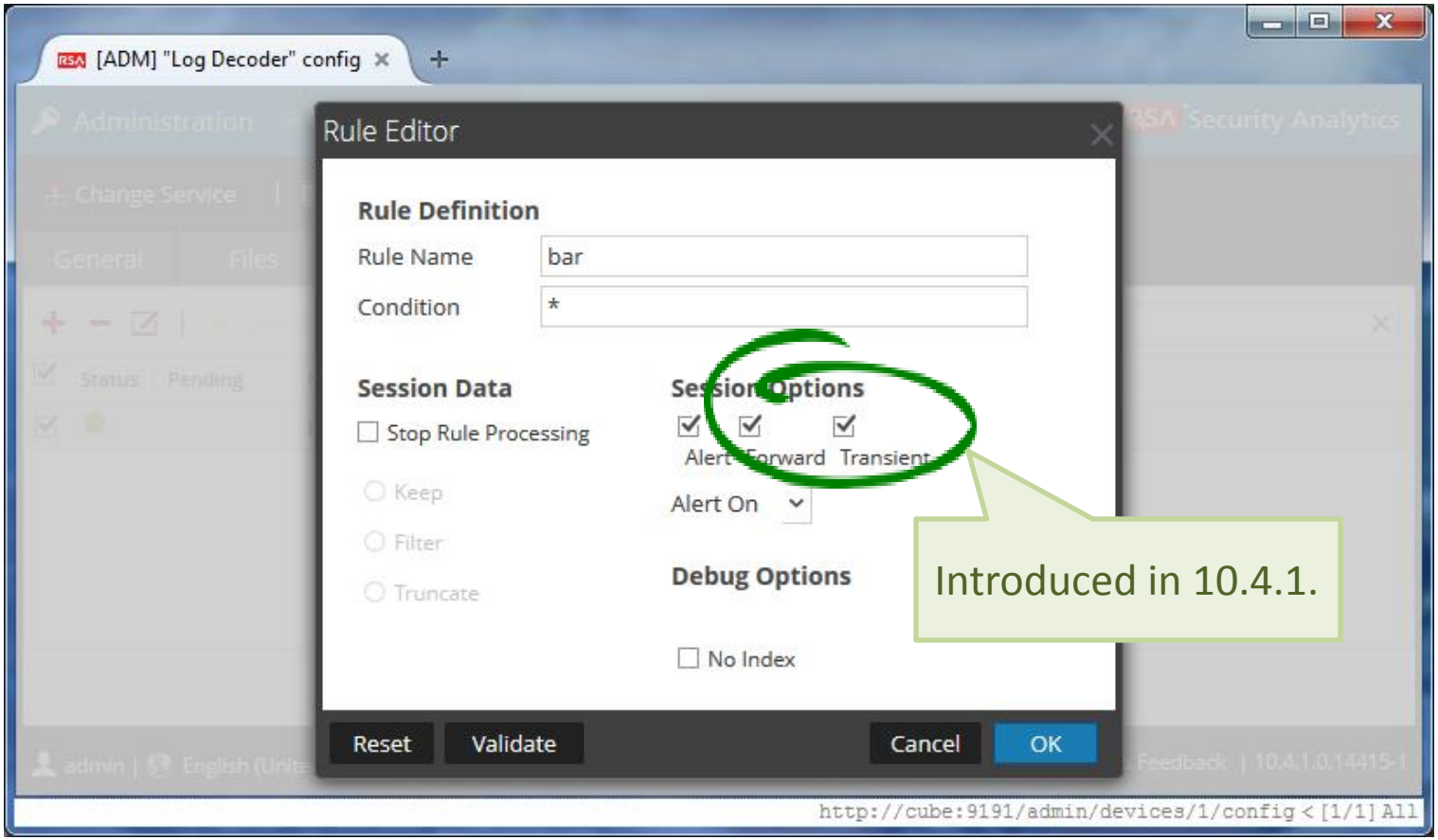
In 10.3.4, the UI is not aware of the 'forward' and 'transient' flags

Log Forwarding Flag Issues



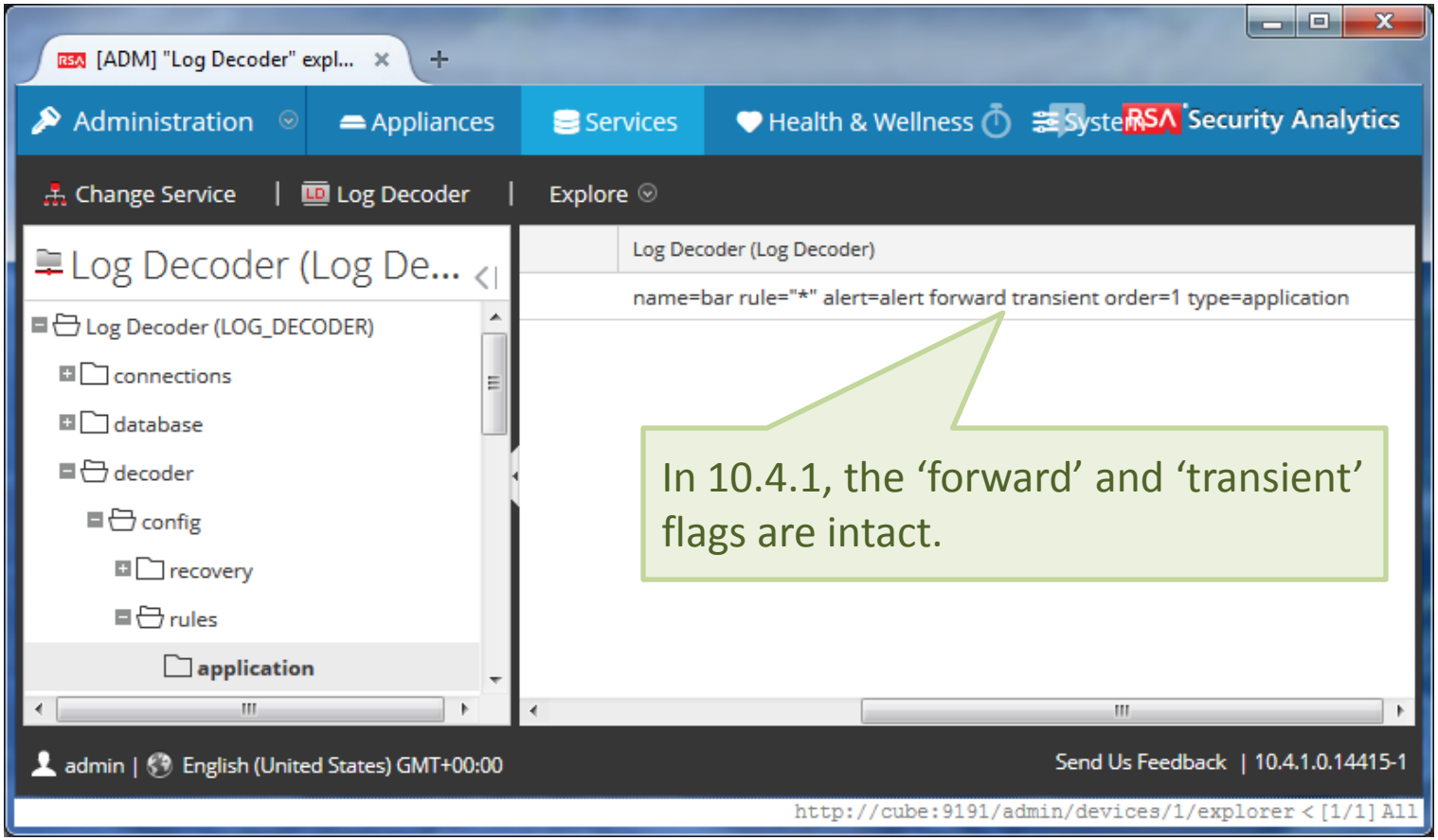
In 10.3.4, after editing a rule in UI, the 'forward' and 'transient' flags are missing

Log Forwarding Flag Issues



Since 10.4.1, the UI recognizes the ‘forward’ and ‘transient’ flags

Log Forwarding Flag Issues



In 10.4.1, after editing a rule, the 'forward' and 'transient' flags are intact

Log Forwarding Flag Issues

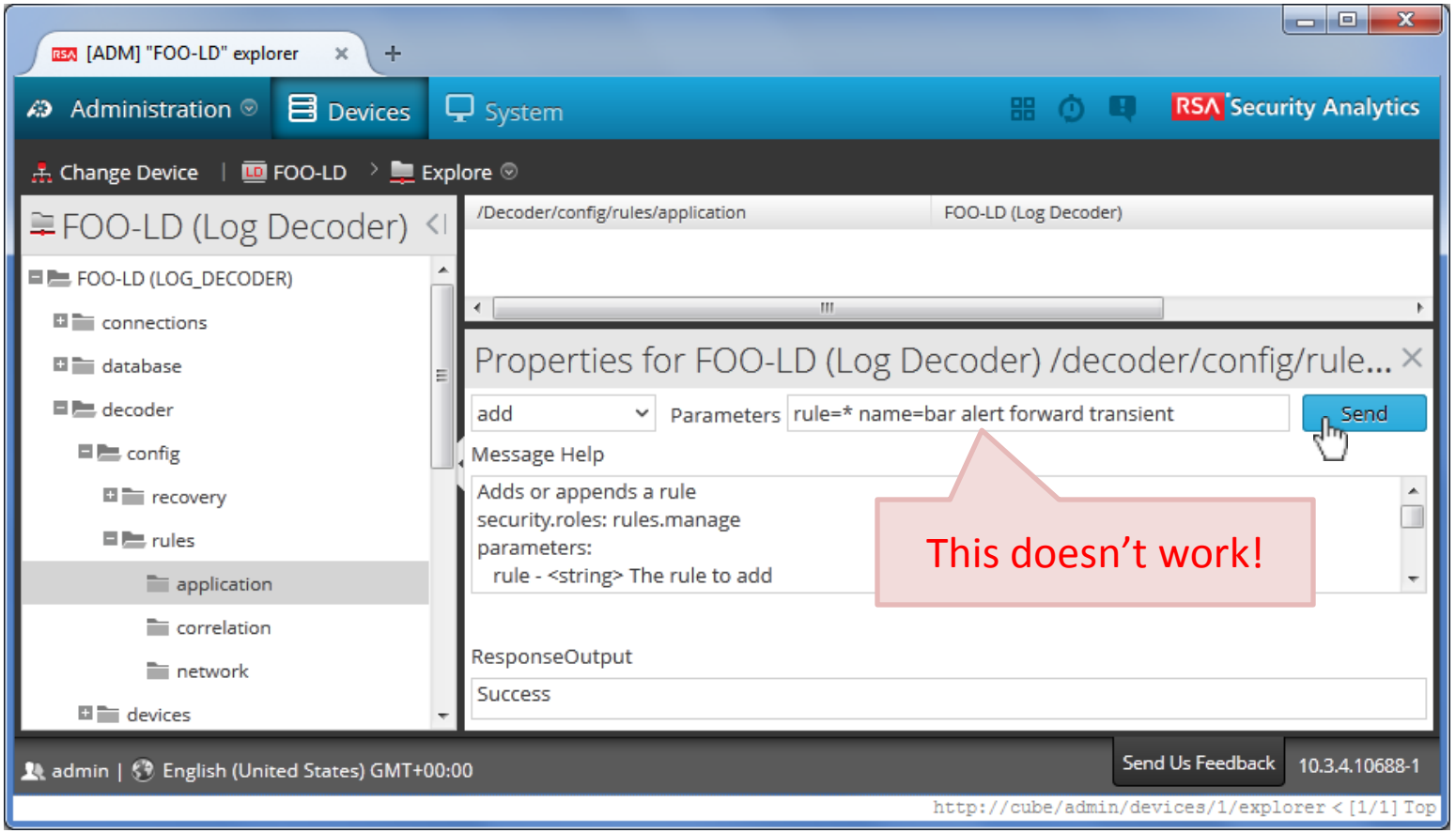
In 10.3.4 and prior versions, 10.4.0 and 10.4.0.1, whenever any action is performed on a log forwarding rule in UI → LD → Config → App Rules, the rule loses the 'forward' flag and the 'transient' flag, if present.

While losing the 'transient' flag does not affect log forwarding, losing the 'forward' flag causes log forwarding to break.

Since 10.4.1, there is a fix to make the UI recognize the 'forward' and 'transient' flags and retain it.

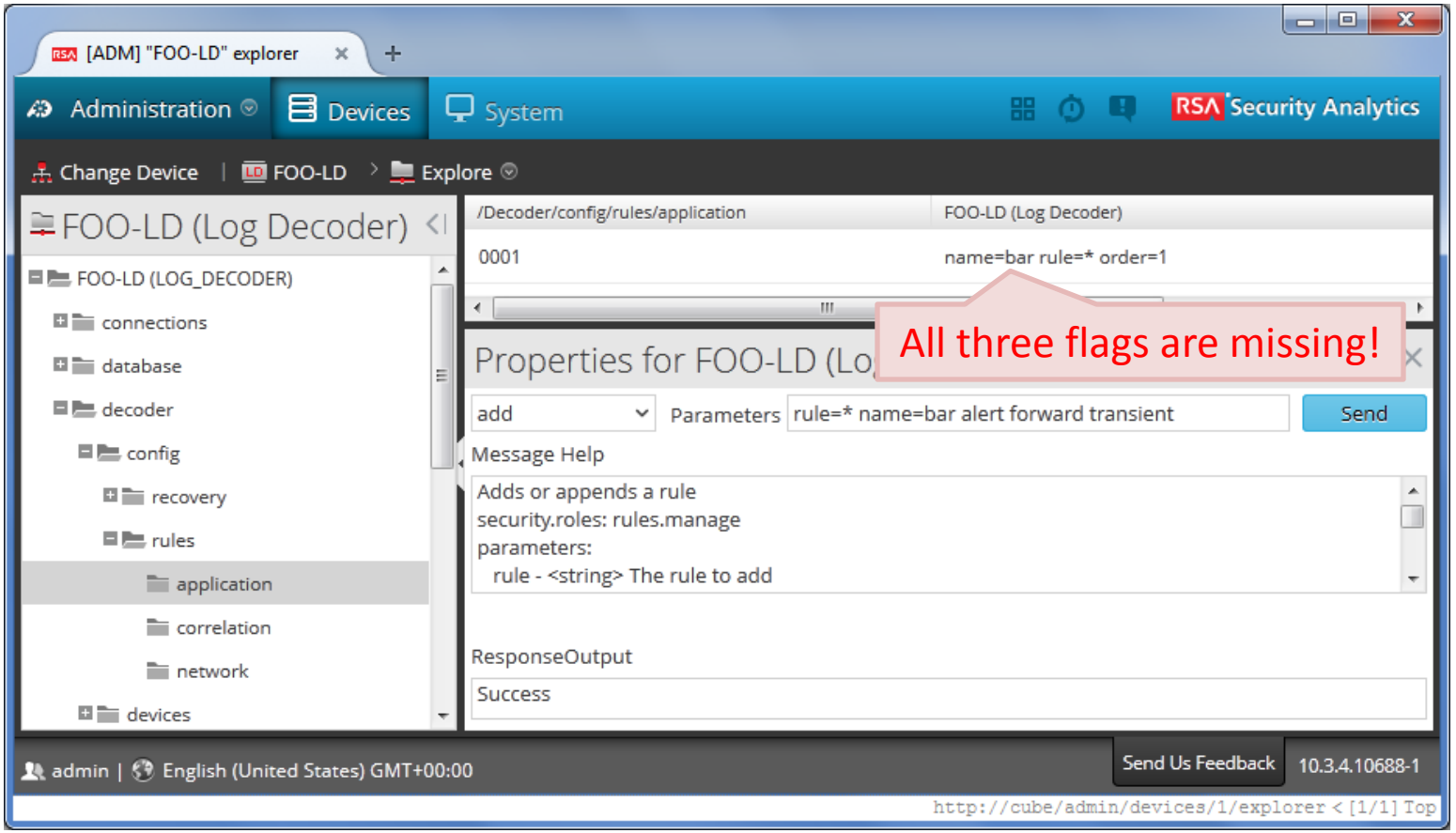
For earlier versions, stick to LD → Explore → /decoder/config/rules for defining and editing log forwarding rules.

Log Forwarding Flag Issues



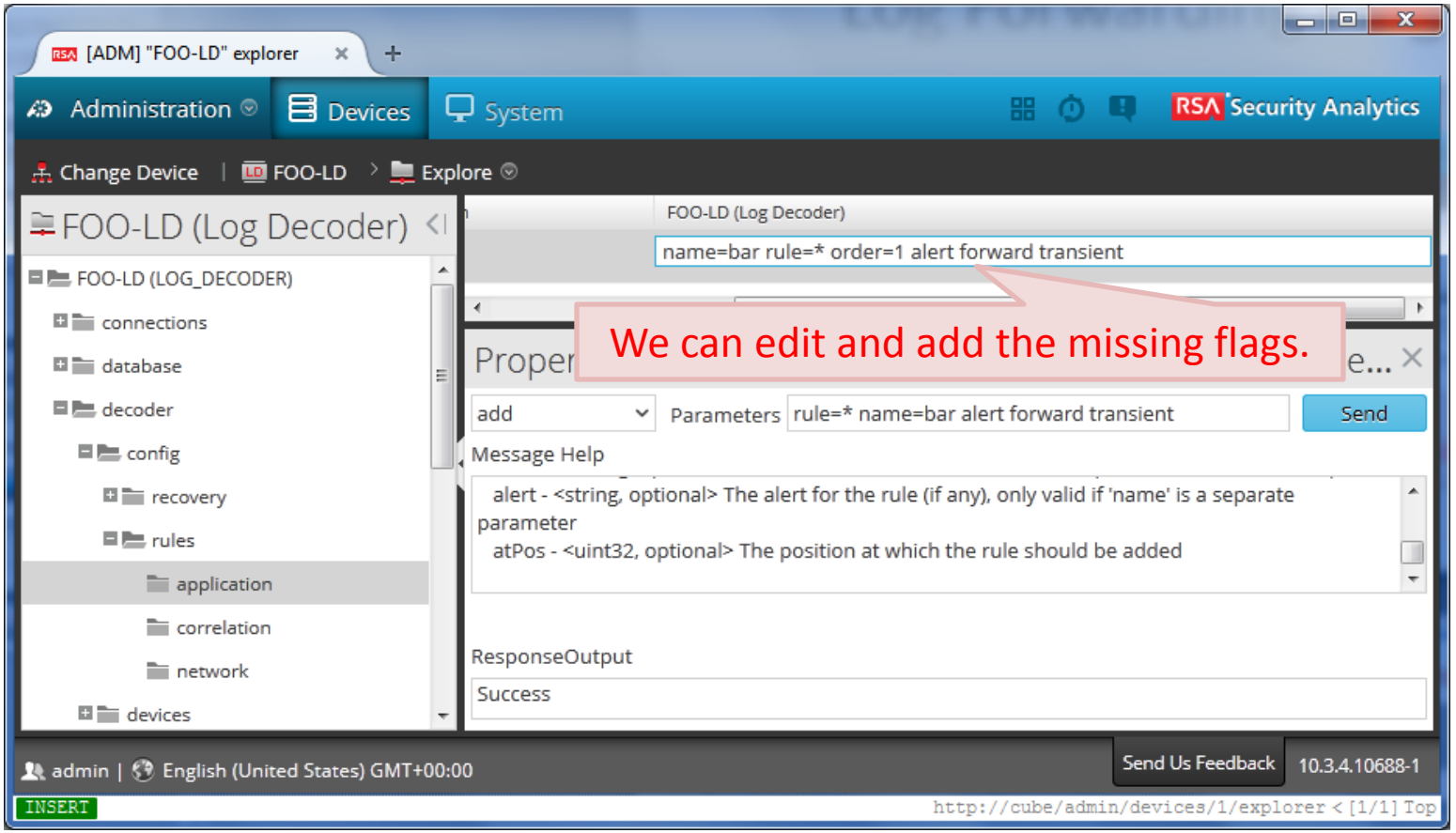
We want to add a rule with 'alert', 'forward' and 'transient' flags via 'Explore' view

Log Forwarding Flag Issues



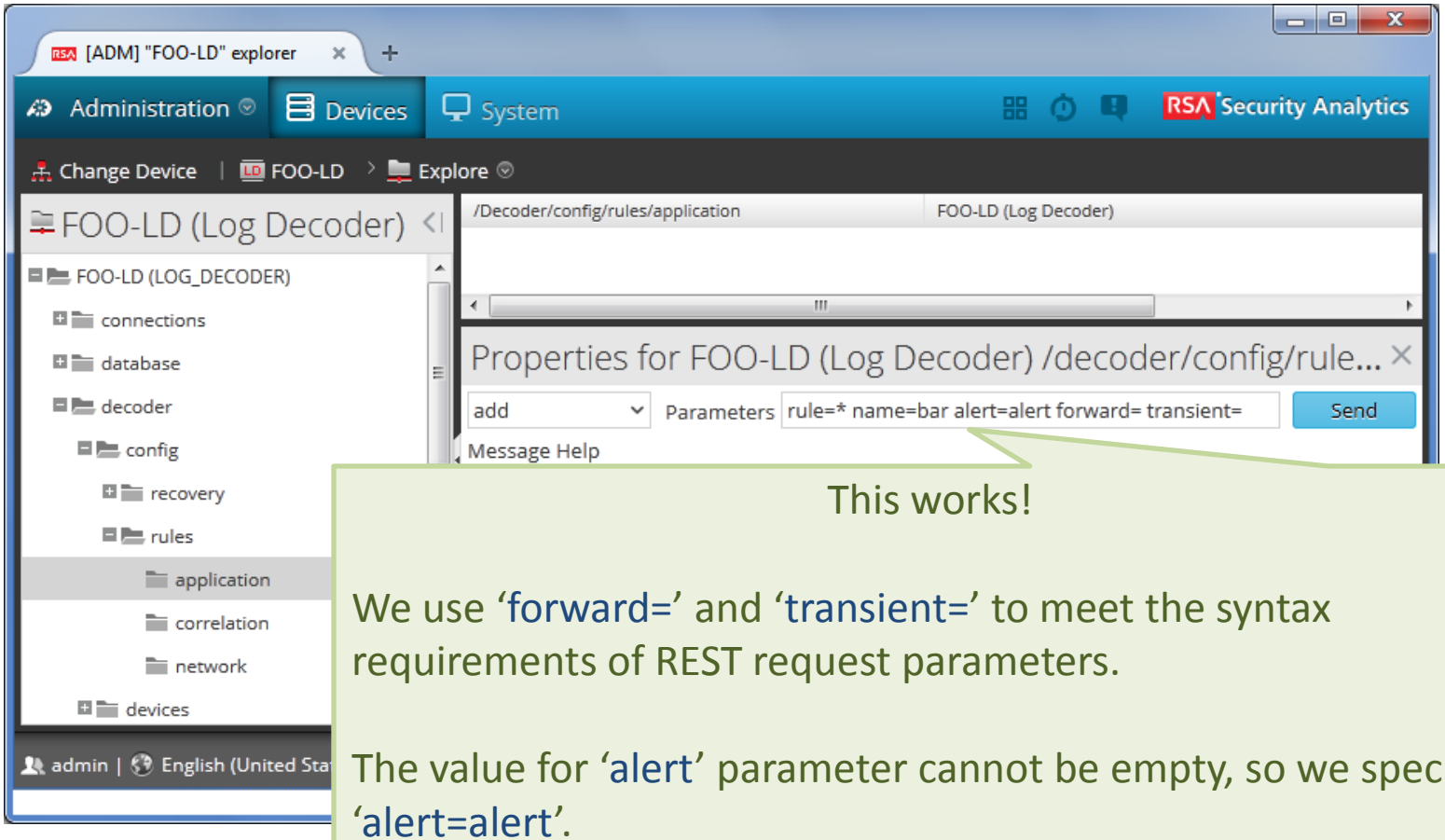
We lost the flags!

Log Forwarding Flag Issues



We can always edit the rule after editing, but there is another way …

Log Forwarding Flag Issues

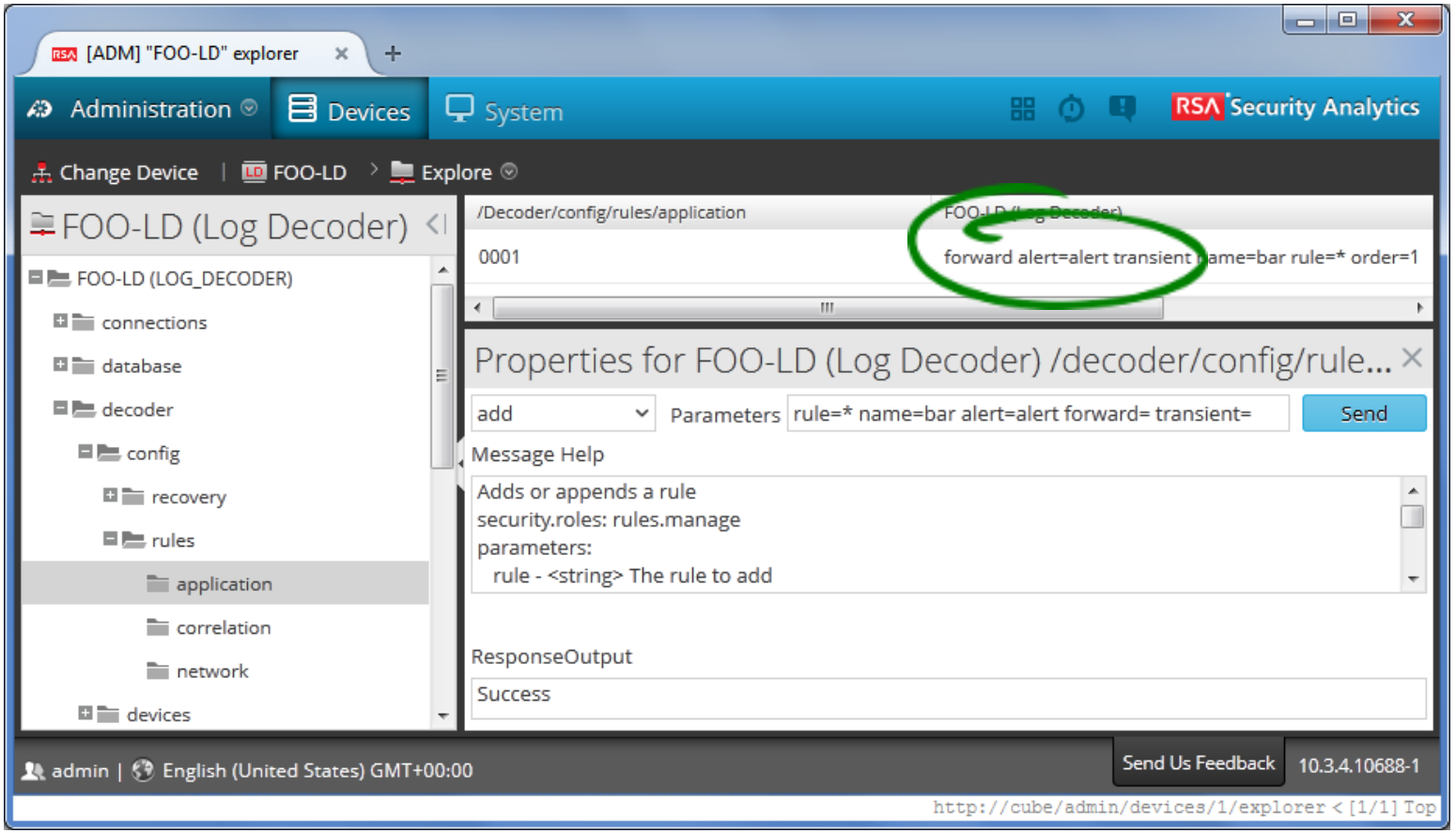


This works!

We use ‘forward=’ and ‘transient=’ to meet the syntax requirements of REST request parameters.

The value for ‘alert’ parameter cannot be empty, so we specify ‘alert=alert’.

Log Forwarding Flag Issues



All flags added

Thank You!