eom Documentation

Release 1.5

Parsons Corp.

Jun 10, 2016

Contents

Contents:

CHAPTER 1

EOM - An Emulation and Operations Monitoring Tool

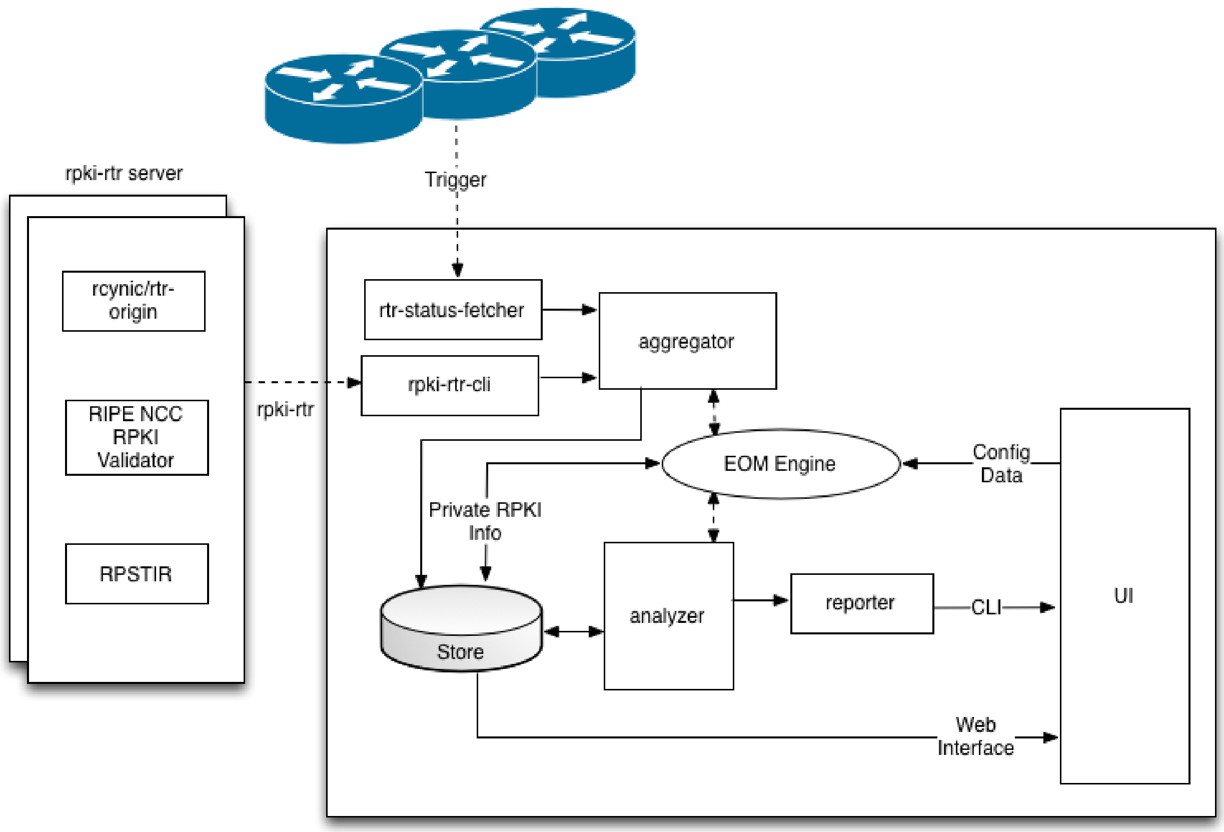
The purpose of this tool is to provide an emulation environment in which an ISP (or other BGP AS) can analyze the impact of planned deployment of RPKI validation in local conditions, without impacting their routing operations.

In the current version the tool simply enables the operator to assess if best route selection changes when validation is enabled. In the future the EOM Tool could assist the operator in scenarios such as the following (note that these features are implemented yet).

- Assessing the manner in which the best path changes when different localpref/weight settings are used.
- Combining data from multiple routers in different ASNs to check for any routing-level inconsistencies.
- Assessing potential problems associated with validating prefixes in a multihoming environment.
- Assessing whether an ISP's reliance on multiple rpki-rtr manager instances could result in routing-level inconsistencies.
- Testing custom prefix assignment and certification conditions for simulation of failures and other special scenarios - e.g. resource transfers.

1.1 High-level overview

The block diagram above highlights the different sub-components within the EOM tool. At a high level the EOM tool fetches RPKI information from one or more RPKI caches, polls RIB related information from routers, analyzes the different pieces of data retrived from these data sources against various parameters, and display its results through some front-end.



The following sub-sections describe the functionality associated with each EOM tool sub-component.

1.1.1 rpki-rtr-cli

This component provides the client interface to the rpki-rtr protocol. It builds upon the existing rpki-rtr implementation from rpki.net but instead of communicating with a router this module interfaces with an aggregator module that maintains a local store of validated RPKI information.

The rpki-rtr-cli module can communicate with multiple rpki-rtr manager instances in order to be able to mimic cases where an ISP relies on multiple rpki-rtr manager instances for serving validated RPKI information to its various ASBR routers.

1.1.2 rtr-status-fetcher

This module is responsible for fetching various pieces of status information from a router, including its routing table and next hop neighbors. The data is fetched using the Trigger module which queries each configured router for its 'sh ip bgp' output.

Note that the event loop structure in Trigger is incompatible with the rpki-rtr module. This issue may have to be revisited in later releases of the EOM software in order to make the data fetch operations from the two modules more compatible.

The rtr-status-fetcher module saves all fetched data into a local store through the aggregator module.

1.1.3 aggregator

This module is responsible for storing and retrieving validated rpki route objects associated with various rpki-rtr manager instances and router status information for the different routers queried.

1.1.4 EOM-engine

The EOM-Engine module provides the central functionality for enabling a network operator to configure and develop various types of operational scenarios and router policies to study the impact of RPKI validation on their network operations.

This module is still under development, but is expected to interface with the UI module to gather user input on the types of analyses that are to be performed. It will access previously saved router and RPKI cache data through the aggregator module and prepare the data for subsequent analysis by other modules.

In order to build scenarios that include misconfigured or particular forms of RPKI validation data, the EOM engine will also interface with a private RPKI store that will allow the user to build test ROAs on the fly to simulate various prefix assignment and certification conditions.

1.1.5 analyzer

This module performs the analysis over the different streams of data that are made available to it. The primary function of this module is to detect and flag differences in the routing state. It stores the results of its processing in persistent storage through the aggregator module.

1.1.6 reporter

The reporter takes the raw results generated by the analyzer module and transforms that data to a form that is useful for user consumption. The output format is currently simple text, but it could be changed to other more useful formats as needed.

1.1.7 UI

The User Interface module provides the engine for displaying the various pieces of EOM data. In the future, this module will also provide the configuration interface for the users, to enable them to specify the location and parameters associated with the different routers and RPKI stores, and to enable them to define the parameters associated with their different scenarios of interest.

CHAPTER 2

EOM Installation

2.1 Dependencies

The following packages must be installed prior to installation of the eom package.

- sqlite3
- os
- trigger
- asyncore
- argparse
- sys
- time
- collections
- pprint
- json
- rpki.rtr
- subprocess
- re
- netaddr
- pipes
- pyparsing
- logging

2.2 Trigger setup

EOM assumes that the operator already has a pre-configured Trigger setup for router management. Information on configuring Trigger can be found on https://trigger.readthedocs.org/en/latest

2.3 Building rpki-rtr

The EOM tool relies on a (modified) rpki-rtr module in order to pull data from a rpki-rtr server. The code that provides this functionality must be built separately. In order to do so, use the following sequence of steps:

```
$ autoconf
$ ./configure
$ make
```

The specific changes made to the stock rpki-rtr distribution from the RPKI.net software can be viewed in the patch file stored in the 'patches' directory.

2.4 Installation of the EOM package (with CLI tools)

The package can be installed by running the setup.py script as follows

```
$ python setup.py install
```

2.5 Installation of EOM UI

The EOM tool comes with a Django-based GUI. Currently this interface is mainly for viewing EOM results and cannot be used to configure the EOM tool.

In order to run the Django app in production it must be installed in conjunction with an existing webserver instantiation. The specific instructions for setting this up is out of scope for this document, but may be found online. For example, see https://docs.djangoproject.com/en/1.9/howto/deployment/wsgi/modwsgi/

In order to run the Django app in test mode the following command may be used:

```
$ cd EOM-dist/eom_ui
$ python manage.py runserver
...
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Note that the eom module needs to be configured first. See the configuration section for instructions on configuring the EOM tool.

2.6 License

- The EOM is distributed with a BSD-like license. See the LICENSE file for more information.
- The license information for the Trigger module is at https://trigger.readthedocs.org/en/latest/license.html
- The pyparsing module is distributed with a MIT license. See http://pyparsing.wikispaces.com/share/view/5698048

CHAPTER 3

EOM Configuration

There are two steps required as part of the EOM tool configuration. The first is the database initialization step, while the second is defining a configuration file for routine EOM operation. This section describes both of these steps.

3.1 Database initialization

First initialize an empty database as follows

```
$ cd EOM-dis/eom
$ python eom --sql-database /path/to/db/eom_db.sqlite --init-db
```

Next, ensure that the eom_ui settings file references the correct database file.

In EOM-dist/eom_ui/eom_ui/settings.py ensure that the DATABASES section refers to the correct file. For example

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join('/path/to/db', 'eom_db.sqlite'),
    }
}
```

Finally, migrate the local database customizations to the above defined DB file.

```
$ cd EOM-dist/eom_ui
$ python manage.py migrate
```

3.2 Defining the EOM configuration file

A sample EOM configuration file is shown below

```
rtr_rib:
    device: quagga.vm
    poll_interval: 86400
    reset: N
    realm: local

sql_database: /path/to/db/eom_db.sqlite
```

```
rpki_serv:
    proto: tcp
    host: rpki-validator.realmv6.org
    port: 8282
    force: 0
    reset: N
```

The rtr_rib section specifies the device(s) that must be polled for RIB information.

Here:

- device refers to the device identifier for our router in the Trigger configuration.
- poll_interval specifies the duration that the EOM tool must wait between repeated queries to the router for RIB information.
- reset specifies whether the rtr_cache table is to be flushed prior to the initial poll operation.
- realm refers to the Trigger module realm that is associated with the credentials for this router device.

The rpki_serv section specifies the rpki-rtr server(s) that must be polled for RPKI information.

- proto specifies the protocol to use for querying the rpki-rtr server instance.
- the host/port fields sepcify the location of the running rpki-rtr server instance.
- force specifies the version to use. Set this value to -1 if no particular version is to be forced.
- reset specifies whether the rtr_cache table is to be flushed prior to the initial poll operation.

Note that when executed from the command line, the rpki_serv and rtr_rib options are specified through a ':' separated list of fields. That is, the above configuration could be specified to the eom tool as follows:

```
$python eom --sql-database /path/to/db/eom_db.sqlite \
           --rtr-rib quagga.vm:86400:N:local \
           --rpki-serv tcp:rpki-validator.realmv6.org:8282:0:N
```

The above example shows a single rtr_rib and rpki_serv instance. If multiple devices need to be specified the it could be done so as follows:

```
rtr_rib:
    - device: quagga.vm
      poll_interval: 86400
      reset: N
      realm: local
    - device: quagga2.vm
      poll_interval: 86400
      reset: N
      realm: local

sql_database: /path/to/db/eom_db.sqlite

rpki_serv:
    - proto: tcp
      host: rpki-validator.realmv6.org
      port: 8282
      force: 0
      reset: N
    - proto: tcp
      host: ca0.rpki.net
      port: 43779
      force: -1
      reset: N
```

CHAPTER 4

EOM CLI Examples

4.1 Command Line Usage

A typical execution of EOM would resemble the following:

```
$ python eom --sql-database eom_db.sqlite \
    --rtr-rib router1:86400:N:local \
    --rpki-serv tcp:localhost:8282:0:N
```

See the configuration section for additional details on what these options mean.

A configuration file may also be passed to the EOM CLI in lieu of passing the three options listed above. An example invocation of eom with a configuration file is:

```
$ python eom --config_file eom_config.yml
```

See the configuration section for addition details on the yml configuration file.

4.2 Command Line Output

A sample output might look like the following:

```
- : *>i    10.0.0.0/8    192.168.1.1    0    0    0    1 2 422    i
I : *>i    10.0.0.0/22    192.168.1.2    0    0    0    1 2 42    i
    (localhost:1234) 6:10.0.0.0/[16-24]
    (localhost:1234) 42:10.0.0.0/[16-20]
V : *>i    10.0.0.0/24    192.168.1.2    0    0    0    1 2 6    i
    (localhost:1234) 6:10.0.0.0/[16-24]
I : *>i    10.0.0.0/26    192.168.1.3    0    0    0    1 2 6    i
    (localhost:1234) 6:10.0.0.0/[16-24]
    (localhost:1234) 42:10.0.0.0/[16-20]
```

In this output the leading prefix may be one of '-' for unknown, 'V' for valid and 'I' for invalid routes. The output above indicates that the routes for 10.0.0.0/22 and 10.0.0.0/26 are invalid because 1) the ROA for 10.0.0.0/[16-20] does not cover the prefix length and 2) the ROA 10.0.0.0/[16-24] while covering the prefix length has a different origination ASN than the one in the route. 'localhost:1234' refers to the RPKI Router server that was used as the source for the ROAs.

The output indicates that a router that performs RPKI validation may prefer the route for /24 even though a route for the /26 prefix exists. In this case since the paths are the same it makes no difference. However in the case of the /22 route, the route that may be preferred may be the one for the /8 aggregate, which originates from a different ASN (422).

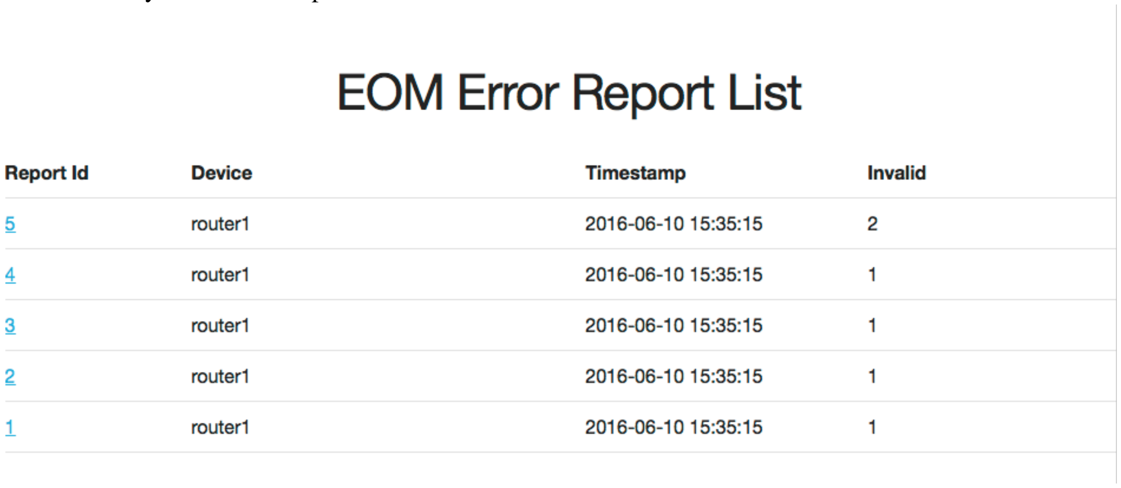
CHAPTER 5

EOM GUI Examples

The following examples assume that the Django app is running on 127.0.0.1:8000

5.1 Status Summary: http:/127.0.0.1:8000/status

This page gives a summary of the error reports availble.



EOM Error Report List

Report Id	Device	Timestamp	Invalid
5	router1	2016-06-10 15:35:15	2
4	router1	2016-06-10 15:35:15	1
3	router1	2016-06-10 15:35:15	1
2	router1	2016-06-10 15:35:15	1
1	router1	2016-06-10 15:35:15	1

The report Id links to the detailed report corresponding to the given entry.

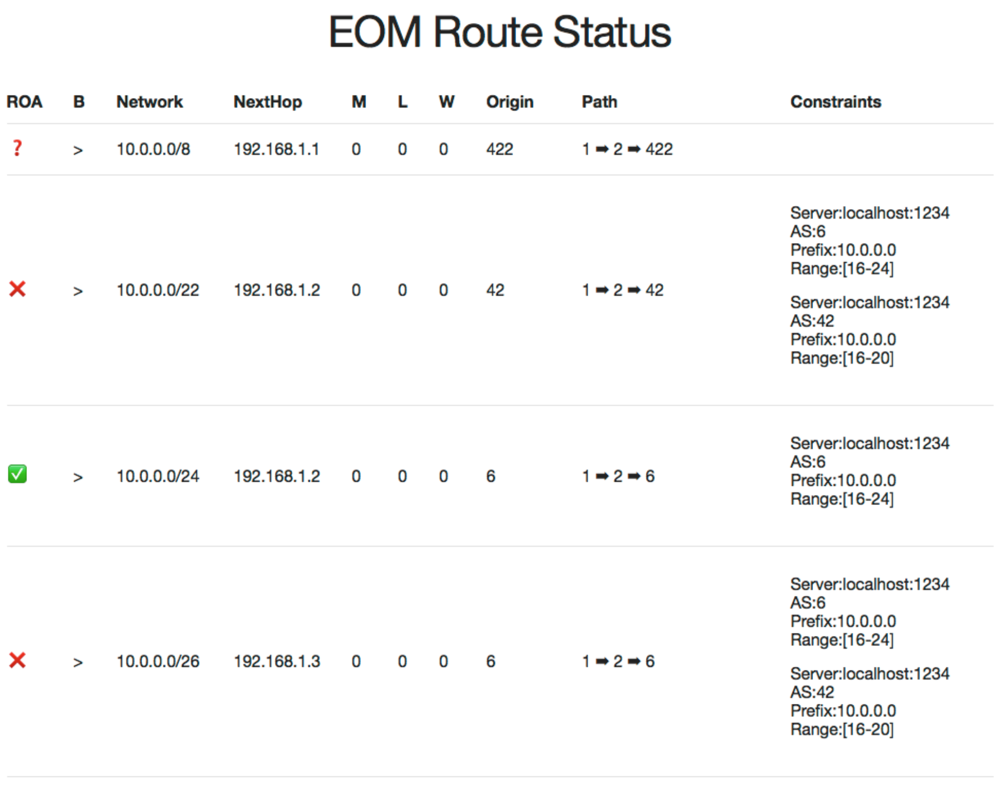
The Device name corresponds to the device identifier that was specified in the Trigger configuration

The timestamp specifies the time of creation of the report.

The Invalid field specifies that number of routes that were found to be invalid in the given report. The actual invalid routes can be viewed by looking at the detailed report.

5.2 Detailed Report: http:/127.0.0.1:8000/status/<id>

This page provides details on the routes that were found to be invalid for the given device, any covering routes that were found to be valid or unspecified, and the constraints that were either violated or satisfied for the given prefix and ASN combination.



EOM Route Status

ROA	B	Network	NextHop	M	L	W	Origin	Path	Constraints
?	>	10.0.0.0/8	192.168.1.1	0	0	0	422	1 ➡ 2 ➡ 422	
✗	>	10.0.0.0/22	192.168.1.2	0	0	0	42	1 ➡ 2 ➡ 42	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24] Server:localhost:1234 AS:42 Prefix:10.0.0.0 Range:[16-20]
✓	>	10.0.0.0/24	192.168.1.2	0	0	0	6	1 ➡ 2 ➡ 6	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24]
✗	>	10.0.0.0/26	192.168.1.3	0	0	0	6	1 ➡ 2 ➡ 6	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24] Server:localhost:1234 AS:42 Prefix:10.0.0.0 Range:[16-20]

The ROA status can be one of the following:

:The ROA constraints matched

:The ROA constraints did not match

:No ROA matching the give prefix was found

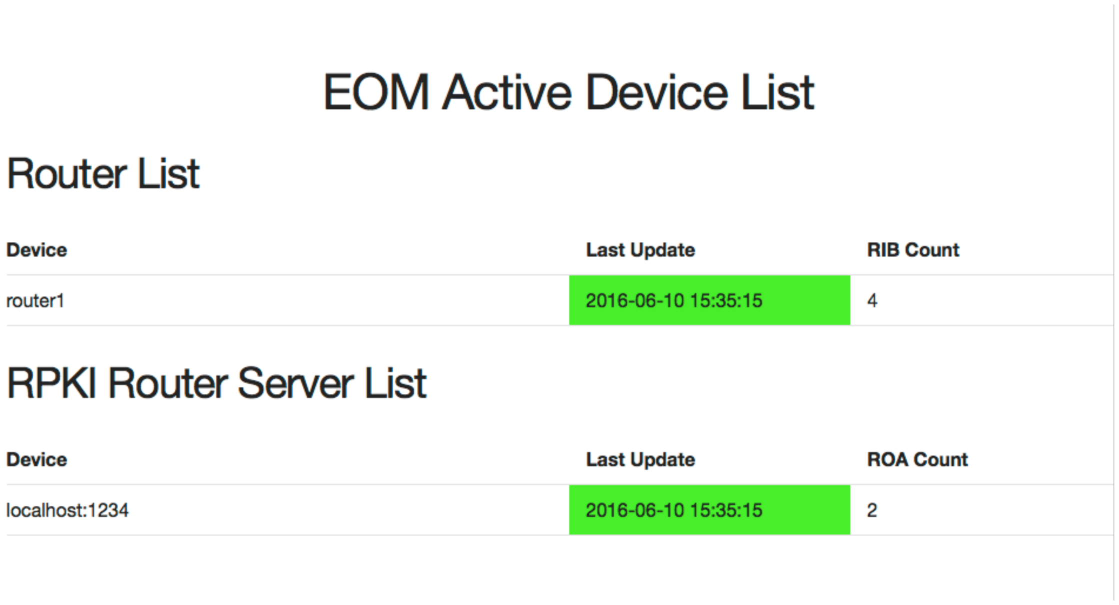
The Network, NextHop, Origin, and Path fields correspond to the equivalent fields in the RIB entry. The field labeled 'B' indicates if the path refers to a best path in the routing table.

The M, L and W fields map to the metric, local-pref and Weight fields of the RIB entry respectively.

Finally, the constraints field gives the RPKI constraints that were either met or violated for the given RIB entry. It lists a sequence of ASN number, Prefix, and the range for each ROA that matches the announced prefix in the RIB entry.

5.3 Devices Report: http:/127.0.0.1:8000/status/devices

This page lists the active rpki-router and Trigger devices that are configured to be polled for relevant data.



EOM Active Device List

Router List

Device	Last Update	RIB Count
router1	2016-06-10 15:35:15	4

RPKI Router Server List

Device	Last Update	ROA Count
localhost:1234	2016-06-10 15:35:15	2

The Device field lists the Device that is being polled.

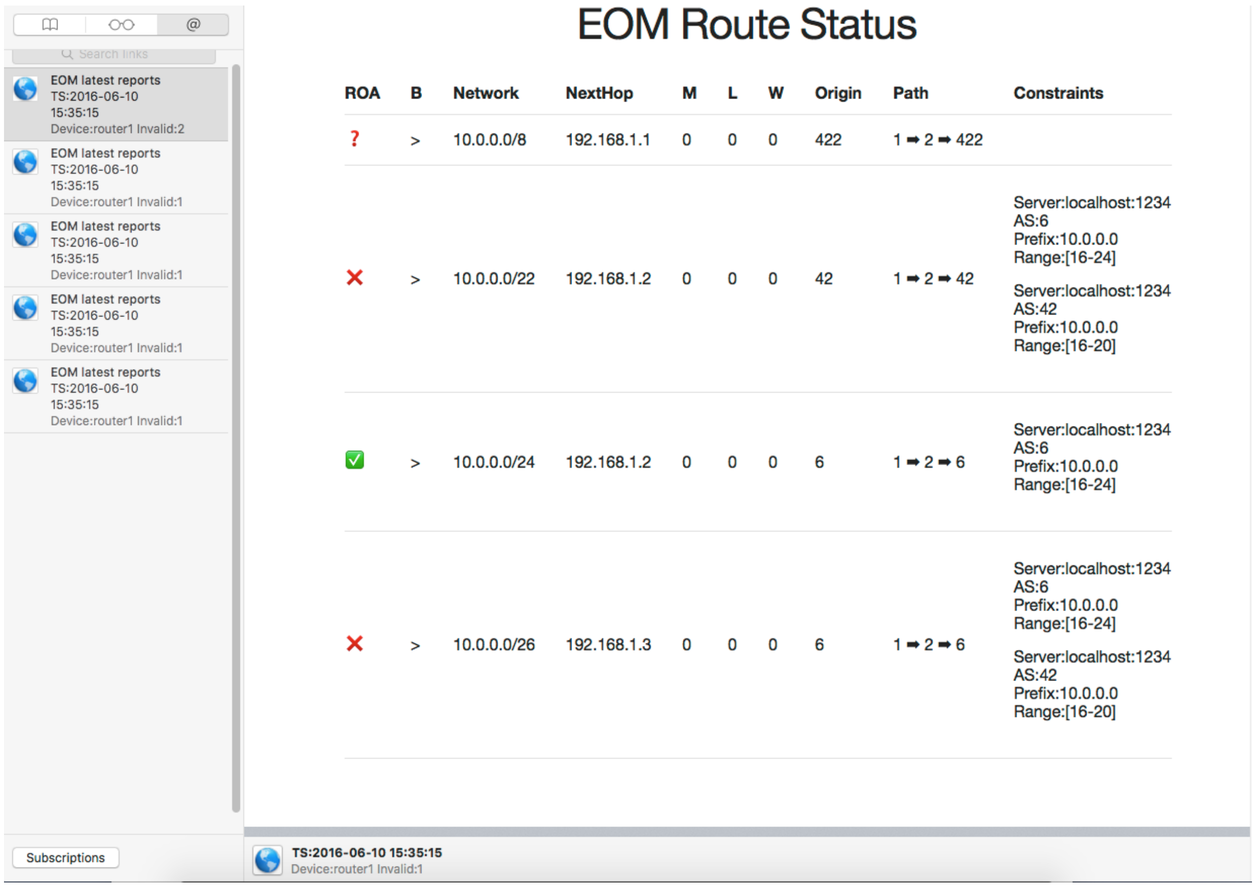
The Last Update field specifies the last time that this particular device was polled.

The RIB count field specifies the number of entries in the RIB that was pulled during the previous poll cycle.

The ROA count field specifies the number of prefixes that were identified in the last sync with the rpki-rtri server instance.

5.4 RSS Feed: http:/127.0.0.1:8000/status/feed

This page provides an RSS feed of the latest reports generated by the EOM monitor.



Search links

EOM latest reports TS:2016-06-10 15:35:15 Device:router1 Invalid:2

EOM latest reports TS:2016-06-10 15:35:15 Device:router1 Invalid:1

EOM latest reports TS:2016-06-10 15:35:15 Device:router1 Invalid:1

EOM latest reports TS:2016-06-10 15:35:15 Device:router1 Invalid:1

EOM latest reports TS:2016-06-10 15:35:15 Device:router1 Invalid:1

Subscriptions

EOM Route Status

ROA	B	Network	NextHop	M	L	W	Origin	Path	Constraints
?	>	10.0.0.0/8	192.168.1.1	0	0	0	422	1 ➡ 2 ➡ 422	
✗	>	10.0.0.0/22	192.168.1.2	0	0	0	42	1 ➡ 2 ➡ 42	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24] Server:localhost:1234 AS:42 Prefix:10.0.0.0 Range:[16-20]
✓	>	10.0.0.0/24	192.168.1.2	0	0	0	6	1 ➡ 2 ➡ 6	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24]
✗	>	10.0.0.0/26	192.168.1.3	0	0	0	6	1 ➡ 2 ➡ 6	Server:localhost:1234 AS:6 Prefix:10.0.0.0 Range:[16-24] Server:localhost:1234 AS:42 Prefix:10.0.0.0 Range:[16-20]

TS:2016-06-10 15:35:15
Device:router1 Invalid:1

The summary information for the feed contains the time stamp associated with the report generation, the device name and the number of invalid entries detected. Clicking on the RSS entry leads the user to the detail report page associated with that particular report as shown in the figure.

CHAPTER 6

eom package

6.1 Submodules

6.2 eom.aggregator module

Implementation of the data aggregation module for EOM.

This module interfaces with various fetcher modules and stores the retrieved data into a persistent data store (database).

class `eom.aggregator.`**`EOMAggregator`**(*dbfile='eom_default_db.sqlite'*, *init_db=False*)
A RPKI-Rtr and RIB data aggregator.

`get_covering`(*device*, *pfxstr_min*, *pfxstr_max*)
Fetch route entries that cover a given prefix range.

Select those routes whose advertised address range covers the given prefix range.

Arguments: device(string): the device name pfxstr_min(string): the lower value in the prefix range pfxstr_max(string): the upper value in the prefix range

`get_report_hash`(*consolidated*)
Get a hash value that corresponds to the consolidated routes

The consolidated routes are structured as a dict. The keys in the dict are the index values of the routes. The values against each key is a tuple with three values

Arguments: val1: valid invalid unknown status ('V'/'I'/'-') val2: RIB tuple with the following fields (status, pfx, pfxlen, pfxstr_min, pfxstr_max, nexthop, metric, locpref, weight, pathbutone, orig_asn, route_orig) val3: A list of tuples that reflect ROA constraints [(asn, prefix, prefixlen, max_prefixlen)...]

`get_rpki_rib`()
Fetch rib information in conjunction with rpki information.

Construct a database 'join' of the contents of currently available rpki-rtr information with currently available RIB information. The join is constructed over matching prefix ranges; that is, for cases where the rpki rtr ROA covers the route prefix in the RIB.

Arguments: None

`get_sql_connection`()
Get an instance to the sql connection object.

Args: None

init_analysis_tables()
Initialize RIB database tables.

Two tables are created. One stores the report ID associated with a given run, while the second stores the contents of the report indexed by the report id.

Args: None

init_rib_tables()
Initialize RIB database tables.

Two tables are created. One stores the rtr ID associated with a given device that is to be queried, while the second stores different route attributes gleaned from 'sh ip bgp' command.

Args: None

init_rpki_rtr_tables()
Initialize rpki-rtr database tables.

Three tables are created - one that keeps track of the rpki-rtr session, one that keeps track of the prefixes associated with each active rpki-rtr session, and finally one for storing router key information.

Args: None

reset_rpki_rtr_session(*host*, *port*)
Reset an existing rpki-rtr session.

Reset any existing rpki-rtr session for the given host and port.

Arguments: host (string): the rpki-rtr server host port (string): the rpki-rtr server port

reset_rtr_rib_session(*device*)
Reset a RIB query session.

Delete any state corresponding to a RIB query that was issued for a particular router device.

Arguments: device(string): The device identifier.

store_analysis_results(*data*, *ts*)
Store the result into the database

6.3 eom.analyzer module

6.4 eom.do_poll module

Fetch RIB information from routers.

This scripts queries the given device for its RIB information using the 'sh ip bgp' command. The script assumes that one of either Trigger or RANCID has been configured in order to enable the router to authenticate the issuance of such commands from the querying machine.

NOTE: That RANCID support appears to be not fully supported in Trigger, so this does not seem to work properly at the moment.

The output is returned as a JSON representation of a dict structure, indexed on the device name and the command run.

class eom.do_poll.**CommandWrapper**(*devicenames*)
Generic Command Wrapper Class.

Allows us to poll the attached net devices for BGP information. Note that this class must not be instantiated directly.

poll()

Poll device list for BGP RIB information.

Poll all the NetDevices that are associated with this object and query them for BGP RIB information using 'sh ip bgp'. The results are returned as a JSON string.

Args: None.

Returns: str: A JSON string representation of the results

class eom.do_poll.**RancidCommandWrapper**(*devicenames*)

Bases: *eom.do_poll.CommandWrapper*

Subclass for issuing router commands using RANCID

class eom.do_poll.**TriggerCommandWrapper**(*devicenames*, *realm*)

Bases: *eom.do_poll.CommandWrapper*

Subclass for issuing router commands using Trigger

6.5 eom.generic_poller module

Implementation of a generic poller.

This module implements an abstract class that serves as the base class for all poller modules. This includes an event queue that keeps track of events that are to be processed during the current time instance and the maintenance of the event log associated with all poll operations.

class eom.generic_poller.**EOMGenericPoller**(*conf*, *aggregator*)

A generic poller

cleanup()

Clean up at exit time.

This method must be overridden by sub-classes.

static get_ip_str(*ipint*)

Convert an IP int to an ordinal IP string.

Convert the IP int first to its dotted or string form. Then expand each component so that determining ranges between two of these strings becomes possible.

Args: ipint: int value of an IP address

Returns: Expanded string representation.

poll()

poll the associated device

This method must be overridden by sub-classes.

6.6 eom.reporter module

Implementation of generic reporter module for EOM.

This module implements a generic reporter module that interfaces with the User Interface (currently stdout).

class `eom.reporter.`**`EOMReporter`**

A generic reporter module

`get_rib_display_str`(*rib_tup*)

Get a string representation for the RIB tuple

Args: rib_tuple: A tuple containing RIB information

Returns: A string representation for the tuple.

`show`(*data*, *ts*)

Display data onto the User Interface

Args:

data(dict): An object containing particulars of the data to be displayed.

Returns: status(boolean): True if success; False if failure.

6.7 eom.rpki_rtr_cli module

6.8 eom.rtr_poller module

6.9 Module contents

This package contains scripts that enables an operator to assess the impact of RPKI validation on routing.

This package exports the following classes:

EOMAggregator: A RPKI-Rtr and RIB data aggregator. EOMAnalyzer: The analyzer module for EOM EOMGenericPoller: A generic poller EOMRPKIRtrCli: Poller for fetching RPKI prefix information from an RPKI-Rtr Manager instance EOMRtrRIBFetcher: Poller for fetching RIB information from a router instance. EOMReporter: A generic reporter module

CommandWrapper: Generic Command Wrapper Class RancidCommandWrapper: Subclass for issuing router commands using RANCID TriggerCommandWrapper: Subclass for issuing router commands using Trigger

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index