InvalidRoutesReporter Documentation

Release latest

Pier Carlo Chiodi

Jun 30, 2017

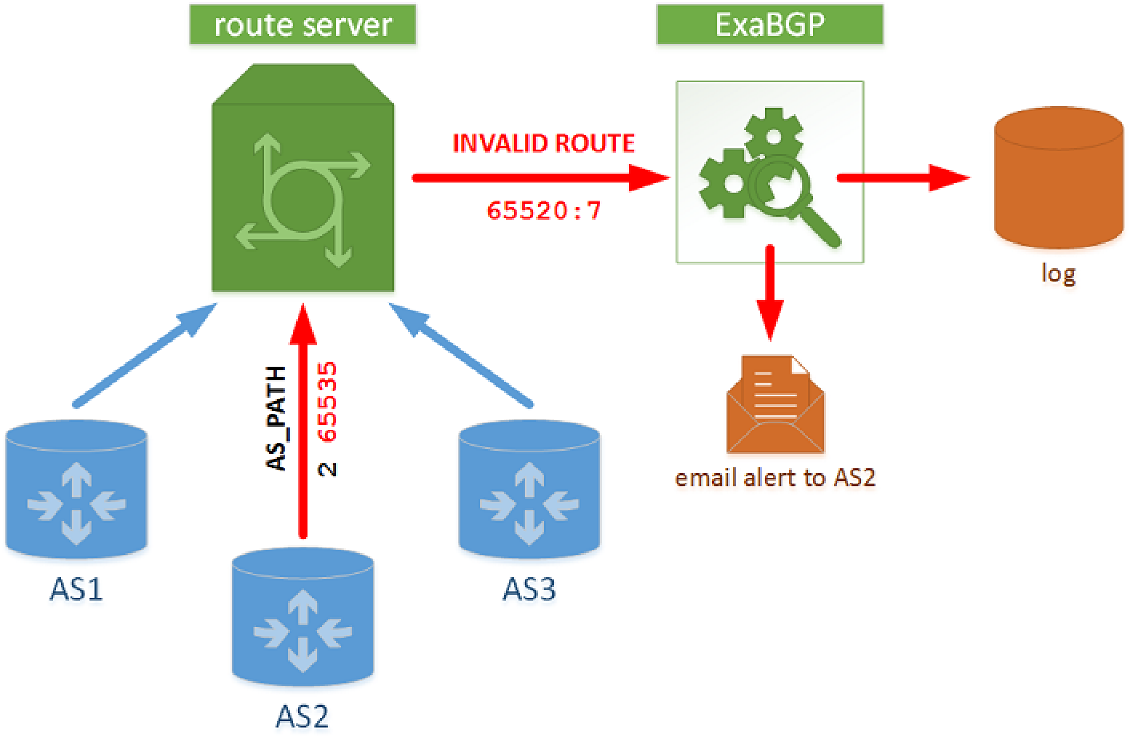
Contents

This script is intended to be used as an ExaBGP process to elaborate and report/log invalid routes that have been tagged with meaningful dedicated BGP communities by route servers.

Invalid routes are those routes that, for some reason, didn't pass the route server's validation process (invalid/private ASNs in the AS_PATH, bogon prefixes, invalid NEXT_HOP, IRRDBs data mismatch, ...). Route servers, instead of discarding them, can keep these routes and tag them with a BGP community that describes the reason for which they have been considered as invalid.

A session with an ExaBGP-based route collector can be used to announce these invalid routes to this script, that finally processes them, extracts the reject reason and uses this information to log a record or to send an email alert to the involved networks.

If deployed in conjunction with ARouteServer, the "tag" reject policy option can be used to easily setup the route server to work together with this script.



CHAPTER 1

Installation

This script has no external dependencies, so it can be fetched from GitHub or installed using pip:

```
$ # download the script from GitHub, then run it...
$ curl -O -L https://raw.githubusercontent.com/pierky/invalidroutesreporter/master/
↪scripts/invalidroutesreporter.py
$ ./invalidroutesreporter.py --help
$
$ # ... or install it using pip
$ pip install invalidroutesreporter
$ invalidroutesreporter.py --help
```

CHAPTER 2

How it works

The script is supposed to be executed by ExaBGP in order to receive and process routes that route servers announce to it.

Three BGP communities are used to determine:

1. if the routes it receives are considered as invalid (*reject-community*),
2. optionally, the reason that led these routes to be considered so (*reject-reason-community*),
3. optionally, the ASN of the peer that actually announced the routes (*rejected-route-announced-by-community*).

In particular, the *rejected-route-announced-by-community* is supposed to be added by the route server on the basis of the BGP session it has toward every client and to be set with the ASN of the peer.

All these BGP communities can be configured by the user: the first one must be given in a straight form (example: `65520:0`) while the other ones need to be set using a regular expression pattern that matches, respectively, the reason code (example: `^65520:(\d+)$`) and the ASN of the network that announced the invalid routes (example: `^rt:65520:(\d+)$`). Since the matching is done on a textual representation basis, standard, extended and large BGP communities can be used indiscriminately.

Example:

- reject-community: `65520:0`
- reject-reason-community: `^65520:(\d+)$`
- rejected-route-announced-by-community: `^rt:65520:(\d+)$`
- route tagged with: `65520:0, 65520:7, rt:65520:666`
- resulting reason code: `7` (that, in ARouteServer, it means "Invalid ASN in AS_PATH")
- resulting peer ASN: `666`

So...

1. The script looks for the *reject-community* to determine if the route it received from the route servers is considered as invalid.

2. If the *reject-community* is found and a *reject-reason-community* has been configured it extracts the reject reason code from the route.

   If used in conjunction with ARouteServer, the list of codes and their meaning is reported within the Reject policy and invalid routes tracking section of ARouteServer's docs.

3. Unless the `--peer-asn-only` command line option has been given, for those routes that are tagged with a *reject-community* the script performs a lookup on a list of *networks* in search of 1) the left-most ASN in the AS_PATH attribute and 2) the IP address used in the NEXT_HOP attribute. The result of this lookup identifies the *recipients* of the alert that will be generated.

   Note: Please note that this lookup's result does not always determine the real network that announced the invalid route: for example, in case of a network that announces a route with an unauthorized left-most ASN or NEXT_HOP address the result of the lookup will be the networks of which the ASN and the peer's address have been used. Here the *rejected-route-announced-by-community* can help to determine the actual network that announced the route (see the next bullet).

   Example of the networks list:

   ```
   {
     "AS1": {
       "neighbors": ["192.0.2.11", "2001:db8:1:1::11", "192.0.2.12",
   ↪"2001:db8:1:1::12"]
     },
     "AS2": {
       "neighbors": ["192.0.2.21", "2001:db8:1:1::21"]
     },
     "AS3": {
       "neighbors": ["192.0.2.31", "2001:db8:1:1::31"]
     },
     "AS23" : {
       "neighbors": ["192.0.2.23"]
     }
   }
   ```

4. If the *rejected-route-announced-by-community* is configured, the script also tries to extract the peer ASN from the communities attached to the route. If an ASN is found and it is also present in the *networks* list it is added to the *recipients* of the alert.

5. For each involved network (*recipients*), the invalid routes collected by the script are added to a buffer; finally, the script triggers some actions on the basis of the *alerters* that have been configured. It can send an email or simply log the route on a file.

In order to convert numeric reject reason codes into a textual description the `--reject-reasons-file` command line argument must be used to provide a JSON file containing the conversion matrix:

```
{
  "1": "Invalid AS_PATH length",
  "2": "Prefix is bogon",
  "3": "Prefix is in global blacklist",
...
```

The reason code/description matrix used by ARouteServer is provided within the `example/arouteserver_reject_reasons.json` file.

CHAPTER 3

Usage

The script must be executed as an ExaBGP process configured to receive JSON parsed update messages. The first mandatory argument is the path to the *networks* configuration file; then, one or more *alerter* configuration files must be supplied. The default BGP communities values are `65520:0` for the *reject-community* and `^65520:(\d+)$` for the *reject reason community*: to set them, the `--reject-community` and the `--reject-reason-pattern` command line arguments can be used. The `--rejected-route-announced-by-pattern` argument can be used to configure the *rejected-route-announced-by-community* pattern. For more options run `./invalidroutesreporter.py --help`.

An ExaBGP configuration example follows:

```
neighbor 192.0.2.2 {
        [...]

        process invalid_routes_reporter {
                run /etc/exabgp/invalidroutesreporter.py /etc/exabgp/networks.json /
↪etc/exabgp/log.alerter.json /etc/exabgp/email.alerter.json;
                encoder json;
                receive {
                        update;
                        parsed;
                }
        }
}
```

CHAPTER 4

Configuration

Note: The configuration files used by this script can be automatically generated starting from the `clients.yml` file used by ARouteServer to build configuration for the route server. This topic is further expanded in the rest of the document.

Details about the configuration of this script follow. An example can be found within the example directory on GitHub.

Networks configuration file

This is a JSON file containing the list of ASNs and their peers IP addresses.

```
{
  "AS1": {
    "neighbors": ["192.0.2.11", "2001:db8:1:1::11", "192.0.2.12", "2001:db8:1:1::12"]
  },
  ...
}
```

Alerter configuration file

Alerters are also configured using JSON files. There are more kinds of *alerters* that implement different actions, but they share a common set of configuration options:

```
{
  "type": "ALERTER_TYPE",

  // The following settings are ment to be interpreted on a
  // recipient-by-recipient basis. Global values can be configured
  // here and they are then inherited by recipients, but each
  // recipient can also be configured with its own values.

  // Optional.
  // Do not perform any action if the last one has been performed
  // less than "min_wait" seconds ago.
  "min_wait": 300,

  // Optional.
  // It is the max number of routes that will be buffered for each
  // recipient.
  // When this buffer is full, perform the action implemented by
  // the alerter, unless the "min_wait" timer has not expired.
  // If the "min_wait" timer is not expired and the buffer is full,
  // any further route will not be buffered and will be lost.
  "max_routes": 30,

  // Optional.
  // Even if the number of buffered routes is less than "max_routes",
  // perform the action if the last one has been performed more than
  // "max_wait" seconds ago.
  "max_wait": 900,

  "recipients": {

    // Recipient ID must be given in the format "AS<n>" and must
    // match the ASN reported in the networks configuration file.
    "recipient ID": {

      // Optional.
      // Recipients inherit settings from the general configuration
      // above, or they may have their own settings configured here.
      //"max_routes": 30,
      //"max_wait": 900,
      //"min_wait": 300,

      // Optional.
      // The "info" section is used by specific alerters; its
      // content depends on the alerters needs.
      "info": {
      }
    },

    // Optional.
    // The wildcard recipient ID "*" matches for any route that is
    // processed by the script.
    "*": {
    }
  }
}
```

Depending on the value of `ALERTER_TYPE`, the following *alerters* can be configured:

- `log`: invalid routes are logged into a file.

  The following configuration options are used for this kind of *alerter*.

  ```
  {
    "type": "log",
  ```

```
  "path": "path to the log file",

  // Optional.
  // If True, routes will be appended to the log file, otherwise
  // the log file will be overwritten every time the process is
  // started.
  "append": False,

  // Optional.
  // This is the template used to make the lines that will be
  // logged.
  // Available macros:
  // - id: the recipient ID
  // - ts: the timestamp the route is received by the script,
  //       seconds since epoch
  // - ts_iso8601: as above, in ISO 8601 format
  // - prefix, next_hop, as_path: route's attributes
  // - reject_reason_code: the code of the reason for which the
  //   route has been considered as invalid by the route server
  // - reject_reason: the description of the reject_reason_code
  // - announced_by: the ASN of the peer that announced the route
  "template": "{id},{ts},{prefix},{as_path},{next_hop},{reject_reason_code},
↪{reject_reason},{announced_by}"
}
```

- `email`: an email is sent to the recipients previously identified during the lookup.

  The following configuration options are used for this kind of *alerter*.

```
{
  "type": "email",

  "host": "smtp server address",

  // Optional.
  "port": 25,

  "from_addr": "noc@acme-ix.net",

  // Optional.
  "subject": "Bad routes received!",

  // Path to a file that contains the template that will be used
  // to build the body of the email message.
  // Available macros:
  // - id: the recipient ID
  // - from_addr: the "from_addr" option
  // - subject: the "subject" option
  // - routes_list: the list of routes that have been buffered,
  //   in the following format:
  //   prefix:      xxx
  //    - AS_PATH:  xxx
  //    - NEXT_HOP: xxx
  //    - reject reason: xxx
  //    - announced by: xxx
  "template_file": "path to the template file used for the body",

  // Optional.
  "username": "SMTP username",
  "password": "SMTP password",

  "recipients": {
    "recipient ID": {
      "info": {
        // Email addresses used to send messages to this
        // recipient.
        "email": ["email1", "email2"]
      }
    }
  }
}
```

Automatically generating configs from ARouteServer

The `build_networks_config_from_arouteserver.py` script can be used to automatically build *networks* and *alerters* configuration files starting from the `clients.yml` file used by ARouteServer. To run it, the PyYAML package must be installed on the system: `pip install PyYAML`.

Email addresses needed by the `email` *alerter* can be also automatically gathered from PeeringDB.

The usage of this script is pretty straightforward: please use the `--help` for details about the arguments it needs.

Examples:

```
$ ./build_networks_config_from_arouteserver.py examples/rich/clients.yml
{
  "AS10745": {
    "neighbors": [
      "192.0.2.22",
      "2001:db:1:1::22"
    ]
  },
  "AS3333": {
    "neighbors": [
      "192.0.2.11"
    ]
  }
}
```

```
$ ./build_networks_config_from_arouteserver.py examples/rich/clients.yml \
>     --networks ~/invalid_routes_collector/networks.json \
>     --email ~/invalid_routes_collector/email.alerter.json \
>     --fetch-email-from-peeringdb
Fetching contacts from PeeringDB: AS10745...
Fetching contacts from PeeringDB: AS3333...
$ cat ~/invalid_routes_collector/email.alerter.json
{
  "host": "smtp_server_address",
  "type": "email",
  "recipients": {
    "AS10745": {
      "info": {
        "email": [
          "ganderson@arin.net"
        ]
      }
    },
    "AS3333": {
      "info": {
        "email": []
      }
    }
  },
  "template_file": "/etc/exabgp/template",
  "from_addr": "noc@acme-ix.net"
}
```

CHAPTER 5

Integration with ARouteServer

- In order to have the route server announcing invalid routes to `invalidroutesreporter`, ARouteServer is set to use the "tag" reject policy option and a .local site-specific file where the session to ExaBGP is configured.
  - ARouteServer `general.yml` configuration file:

```
cfg:
  [...]
  filtering:
    reject_policy:
      policy: tag
  [...]
  communities:
    reject_cause:
      std: 65520:dyn_val
    rejected_route_announced_by:
      ext: rt:65520:dyn_val
```

  - Content of `/etc/bird/footer4.local` on the route server:

```
filter invalid_routes_only {
   if ((65520, 0) ~ bgp_community) then
      accept;
   reject;
}
protocol bgp InvalidRoutesCollector {
   local as 999;
   neighbor 192.0.2.99 as 65534;
   rs client;
   add paths tx;
   secondary;

   import none;
   export filter invalid_routes_only;
}
```

  - Command to build the route server's configuration:

    ```
    $ arouteserver bird --ip-ver 4 --use-local-files footer4
    ```

- The ExaBGP side is configured with a session to the route server and a process that executes the `invalidroutesreporter` script:

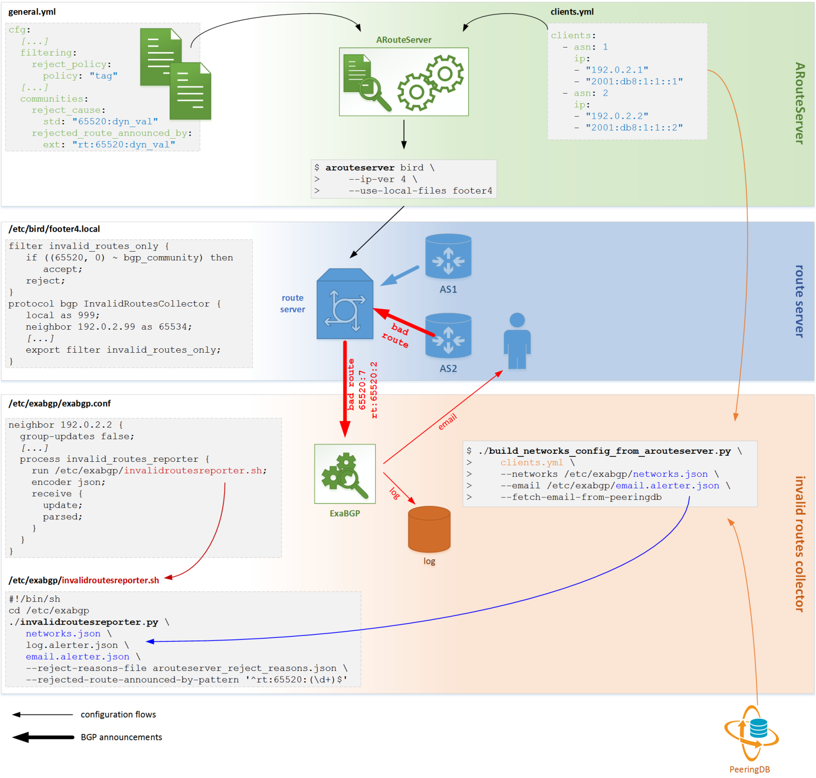
  - content of `exabgp.conf`:

    ```
    neighbor 192.0.2.2 {
            router-id 192.0.2.99;
            local-address 192.0.2.99;
            local-as 65534;
            peer-as 999;
            group-updates false;
            add-path receive;

            family {
                    ipv4 unicast;
                    ipv6 unicast;
            }

            process invalid_routes_reporter {
                    run /etc/exabgp/invalidroutesreporter.sh;
                    encoder json;
                    receive {
                            update;
                            parsed;
                    }
            }
    }
    ```

  - content of `invalidroutesreporter.sh`:

    ```
    #!/bin/sh
    cd /etc/exabgp
    ./invalidroutesreporter.py \
            networks.json \
            log.alerter.json \
            email.alerter.json \
            --reject-reasons-file arouteserver_reject_reasons.json \
            --rejected-route-announced-by-pattern '^rt:65520:(\d+)$'
    ```

An integration diagram follows (click to enlarge).



general.yml

```
cfg:
  [...]
  filtering:
    reject_policy:
      policy: "tag"
  [...]
  communities:
    reject_cause:
      std: "65520:dyn_val"
    rejected_route_announced_by:
      ext: "rt:65520:dyn_val"
```

clients.yml

```
clients:
  - asn: 1
    ip:
    - "192.0.2.1"
    - "2001:db8:1:1::1"
  - asn: 2
    ip:
    - "192.0.2.2"
    - "2001:db8:1:1::2"
```

```
$ arouteserver bird \
>     --ip-ver 4 \
>     --use-local-files footer4
```

/etc/bird/footer4.local

```
filter invalid_routes_only {
   if ((65520, 0) ~ bgp_community) then
      accept;
   reject;
}
protocol bgp InvalidRoutesCollector {
   local as 999;
   neighbor 192.0.2.99 as 65534;
   [...]
   export filter invalid_routes_only;
}
```

/etc/exabgp/exabgp.conf

```
neighbor 192.0.2.2 {
  group-updates false;
  [...]
  process invalid_routes_reporter {
    run /etc/exabgp/invalidroutesreporter.sh;
    encoder json;
    receive {
      update;
      parsed;
    }
  }
}
```

```
$ ./build_networks_config_from_arouteserver.py \
>     clients.yml \
>     --networks /etc/exabgp/networks.json \
>     --email /etc/exabgp/email.alerter.json \
>     --fetch-email-from-peeringdb
```

/etc/exabgp/invalidroutesreporter.sh

```
#!/bin/sh
cd /etc/exabgp
./invalidroutesreporter.py \
   networks.json \
   log.alerter.json \
   email.alerter.json \
   --reject-reasons-file arouteserver_reject_reasons.json \
   --rejected-route-announced-by-pattern '^rt:65520:(\d+)$'
```

CHAPTER 6

Status

Currently this tool is in a beta status and needs testing. Please consider this before using it in production!

CHAPTER 7

Author

Pier Carlo Chiodi - https://pierky.com

Blog: https://blog.pierky.com Twitter: @pierky