Figure 1:

LAB: RPKI

Part-1: Installing RPKI Validator

- `#` super user command.
- `$` normal user command.
- Username `apnic` and password `training`.

VM Details

```
[group01.apnictraining.net] [192.168.30.1]
[group02.apnictraining.net] [192.168.30.2]
......
[group10.apnictraining.net] [192.168.30.10]
[group11.apnictraining.net] [192.168.30.11]
......
[group20.apnictraining.net] [192.168.30.20]
[group21.apnictraining.net] [192.168.30.21]
......
[group30.apnictraining.net] [192.168.30.30]
```

Preinstalled packages

To save time, the following essential packages have been preinstalled on the containers:

- `curl`
- `wget`
- `GCC (GNU C toolchain)`
- `rsync`

Lab Setup

For this lab, we will use OctoRPKI from Cloudflare as the relying party or the RPKI validator.

1. Login to the server (SSH using the username and password given above), where **X** is your group number:

   ```
   ssh apnic@192.168.30.X
   ```

2. Update the repository:

   ```
   sudo apt update && sudo apt upgrade
   ```

3. Download the validator:

   ```
   wget https://github.com/cloudflare/cfrpki/releases/download/v1.1.4/octorpki_1.1.4_a
   dpkg -i octorpki_1.1.4_amd64.deb
   ```

4. Download the standard TALs. Note that by downloading ARIN's TAL, you agree to be bound by ARIN's Relying Party Agreement (RPA):

```
mkdir tals
cd tals
wget https://raw.githubusercontent.com/cloudflare/cfrpki/master/cmd/octorpki/tals/a
wget https://raw.githubusercontent.com/cloudflare/cfrpki/master/cmd/octorpki/tals/a
wget https://raw.githubusercontent.com/cloudflare/cfrpki/master/cmd/octorpki/tals/l
wget https://raw.githubusercontent.com/cloudflare/cfrpki/master/cmd/octorpki/tals/r
wget https://www.arin.net/resources/manage/rpki/arin-rfc7730.tal -O arin.tal
cd ..
```

5. Run the validator:

```
nohup octorpki -output.sign=false > out 2> err &
```

6. Use the following command to retrieve the validated ROA payloads (produces a list of ASNs and prefixes). If this command produces the string "File not ready yet", then the validator is still working through the initial synchronisation process, which generally takes a few minutes. By default, the server will resynchronise its state every 20 minutes.

```
curl localhost:8080/output.json
```

NOTE: Now the validator is ready to feed the validated cache to an rpki-rtr server, which in turn handles requests from BGP-speaking routers through the RTR (RPKI-to-Router) protocol.

Part-2: RTR session

Validator side

GoRTR is Cloudflare's rpki-rtr server component.

1. Download the rpki-rtr server:

```
wget https://github.com/cloudflare/gortr/releases/download/0.11.4/gortr_0.11.4_amd6
dpkg -i gortr_0.11.4_amd64.deb
```

2. Run the server, listening for rpki-rtr requests on port 8282, where X is your group number:

```
nohup gortr -bind=192.168.30.X:8282 -metrics.addr=:8081 -verify=false -cache=http:/
```

(Unchanged from this point.)

Router Side

Topology

The topology below has 8 routers (`R13, R14, ...R20`), each with a unique ASN (`AS135533 - AS135540`).

Figure 2:

Address plan & ROA table

images/addr_plan.png

Router	AS#	fa0/1 (to eBGP peers)	e1/1 (to Validator)	ROA/route
R13	135533	172.16.0.1/30	192.168.30.13/24	61.45.248.0/24
R14	135534	172.16.0.2/30	192.168.30.14/24	61.45.249.0/24
R15	135535	172.16.0.5/30	192.168.30.15/24	61.45.250.0/24
R16	135536	172.16.0.6/30	192.168.30.16/24	61.45.251.0/24
R17	135537	172.16.0.9/30	192.168.30.17/24	61.45.252.0/24
R18	135538	172.16.0.10/30	192.168.30.18/24	61.45.253.0/24
R19	135539	172.16.0.13/30	192.168.30.19/24	61.45.254.0/24
R20	135540	172.16.0.14/30	192.168.30.20/24	61.45.255.0/24

Lab Notes

- For this lab, the RPKI Validator (Routinator) has been installed and configured by the instructor as shown in the topology. The validator's IP address is `192.168.30.240`
- To simplify the configuration, the routers will establish eBGP session in pairs as shown below:

```
R13<-->R14
R15<-->R16
R17<-->R18
R19<-->R20
```

- Each router also has a connection to the RPKI validator to allow RTR (**rpki-to-router**) sessions.
- ROAs have already been created for each of the prefixes with corresponding origin AS numbers (`AS135533 - AS135540`) as shown in the table above.

Lab Exercise

1. Telnet to your group's router as shown below:

```
telnet 192.168.30.254 2013 [R13]
telnet 192.168.30.254 2014 [R14]
telnet 192.168.30.254 2015 [R15]
telnet 192.168.30.254 2016 [R16]
telnet 192.168.30.254 2017 [R17]
telnet 192.168.30.254 2018 [R18]
telnet 192.168.30.254 2019 [R19]
telnet 192.168.30.254 2020 [R20]
```

2. If you see the following message during router bootup, enter `no`:

```
Would you like to enter the initial configuration dialog? [yes/no]:
```

3. You also might see the following service configuration messages when the IOS boots:

```
%Error opening tftp://192.168.30.254/network-confg (Timed out)
%Error opening tftp://192.168.30.254/cisconet.cfg (Timed out)
%Error opening tftp://192.168.30.254/router-confg (Timed out)
%Error opening tftp://192.168.30.254/ciscortr.cfg (Timed out)
```

   - Please disable this inbuilt feature and save the config to prevent it from happening during the next boot up:

```
no service config
do wr
```

NOTE: If you need to reload your router, DO NOT issue the reload command (please ask the instructor)!

4. Configure the host name and the interface to the validator (example for R13 below). Refer the address plan table:

```
hostname R13
no logging console
!
interface ethernet1/1
 description link to RPKI-Validator
 ip address 192.168.30.13 255.255.255.0
 no shutdown
```

5. Verify connectivity between the router and the Validator

```
ping 192.168.30.240
```

6. Configure the interface connecting to your eBGP peer (example for **R13** below). Refer the address plan table:

```
interface fa0/1
 description link to R14
 ip address 172.16.0.1 255.255.255.252
 no shutdown
```

7. Verify connectivity to your eBGP peer (talk to your neighbor if there is no reachability). Example for **R19** to check its physical connection to **R20**:

```
ping 172.16.0.14
```

8. Configure eBGP with your neighbor (make sure its the correct neighbor). Example below for **R13**'s eBGP session with **R14**:

```
router bgp 135533
 neighbor 172.16.0.2 remote-as 135534
 !
 address-family ipv4 unicast
  neighbor 172.16.0.2 activate
```

9. Make sure the eBGP session is up with your neighbor

```
sh bgp ipv4 unicast summary
```

   - **Note:** You will not see any prefixes received from your neighbor yet.

10. Announce the correct prefix (based on the address plan table above) to your neighbor. Example below is for **R15**:

```
ip route 61.45.250.0 255.255.255.0 null 0
!
router bgp 135535
 address-family ipv4 unicast
 network 61.45.250.0 mask 255.255.255.0
```

11. Check/Verify routes learned from your neighbor. Example, for **R19** to verify received routes from its neighbor **R20**:

```
sh bgp ipv4 unicast neighbors 172.16.0.14 routes
```

12. Verify the BGP table:

```
sh bgp ipv4 unicast
```

13. Verify the routing table for BGP learned routes

```
sh ip route bgp
```

14. Setup RTR (rpki-to-router) session with the RPKI validator. Example for **R13**:

```
router bgp 135533
  bgp rpki server tcp 192.168.30.240 port 3323 refresh 900
```

NOTE: *Since the router will now pull the validated ROA cache using the RTR protocol from the Validator, it might take a while.*

15. Verify the RTR session with the Validator

```
sh ip bgp rpki servers
```

OR

```
sh bgp ipv4 unicast rpki servers
```

- The output should look like something below:

```
BGP SOVC neighbor is 192.168.30.240/3323 connected to port 3323
Flags 192, Refresh time is 900, Serial number is 0, Session ID is 15578
InQ has 0 messages, OutQ has 0 messages, formatted msg 1
Session IO flags 3, Session flags 4000
Neighbor Statistics:
    Prefixes 39736
    Connection attempts: 1
    Connection failures: 0
    Errors sent: 0
    Errors received: 0
Connection state is ESTAB, I/O status: 1, unread input bytes: 0
……
```

16. Look at all the valid ROAs learned from the Validator

```
sh bgp ipv4 unicast rpki table
```

- Should output a list of ROAs (origin-AS, max-length) like below:

```
65373 BGP sovc network entries using 5752824 bytes of memory
69579 BGP sovc record entries using 1391580 bytes of memory

Network             Maxlen  Origin-AS  Source  Neighbor
1.0.0.0/24          24      13335      0       192.168.30.240/3323
1.1.1.0/24          24      13335      0       192.168.30.240/3323
1.9.0.0/16          24      4788       0       192.168.30.240/3323
1.9.12.0/24         24      65037      0       192.168.30.240/3323
1.9.21.0/24         24      24514      0       192.168.30.240/3323
1.9.23.0/24         24      65120      0       192.168.30.240/3323
1.9.31.0/24         24      65077      0       192.168.30.240/3323
1.9.65.0/24         24      24514      0       192.168.30.240/3323
1.34.0.0/15         24      3462       0       192.168.30.240/3323
1.36.0.0/19         19      4760       0       192.168.30.240/3323
```

17. Now check the BGP table again to see how the routes learned from your neighbors are tagged with the RPKI validation states of **Valid**, **Invalid** or **Not Found**:

```
show bgp ipv4 unicast
```

- Since we have created the ROAs corresponding to the prefixes used in this lab, you should see all of them tagged as valid (**V**). Example below for **R14**:

Figure 3:

- Also verify the routing table (you should see the valid routes in the routing table)

```
sh ip route bgp
```

18. Let us now try to announce some **Invalid** routes (or hijack someone's routes).

- Go ahead and announce routes (refer the ip address plan) that belong to other groups. In the example below, **R13** in **AS135533** is announcing **R20**'s prefix (**AS135540**):

```
ip route 61.45.255.0 255.255.255.0 null 0
!
router bgp 135533
 address-family ipv4 unicast
  network 61.45.255.0 mask 255.255.255.0
```

- Verify the BGP table on **R14** (your eBGP neighbor).

```
sh bgp ipv4 unicast
```

- You will see that the route `61.45.255.0` learned from its neighbor **R13** has been tagged as **Invalid** (**I**). Discuss within your group why it is Invalid?

Figure 4:

- Now, look at the routing table:

```
sh ip route bgp
```

OR

```
sh ip route
```

  - You will notice that the Invalid route has **NOT** be been inserted in the routing table.

Figure 5:

  - **NOTE: The default Cisco IOS behaviour is to not include invalid routes for best path selection!**
  - If you dont want to drop invalids with Cisco IOS, you need to explicitly tell BGP to include invalids for best path selection (under respective AFs) as shown below for **R14**:

```
router bgp 135534
 address-family ipv4 unicast
  bgp bestpath prefix-validate allow-invalid
```

    * Verify the routing table to see how BGP behaves with the above command:

```
sh ip route bgp
```

* The Invalid route now appears in the routing table of **R14** as shown below:

Figure 6:

19. Let us have a look at **Not Found** routes - routes for which there are no correspnding ROAs (neither valid or invalid, perhaps not created yet). These make up more than 86% of the global routing table, which indicates many people haven't created ROAs for their prefixes!

    - Let us announce some special use prefixes (RFC5735), for which there should not be any existing ROAs. Example below for **R13** announcing documentation prefix **203.0.113.0/24**

      ```
      ip route 203.0.113.0 255.255.255.0 null 0
      !
      router bgp 135533
       address-family ipv4 unicast
        network 203.0.113.0 mask 255.255.255.0
      ```

    - A look at **R14**'s BGP table:

Figure 7:

    - And **R14**'s routing table shows the Not Found routes are included in the best path selection:

Figure 8:

20. If we do not want to drop Invalids, we can follow best practice recommendations in **RFC7115** to prefer Valids over Not Found or Invalids, and prefer Not Found over Invalid origins.

    - Define a routing policy that prefers **Valids > Not Founds > Invalids**

      ```
      route-map ROUTE-VALIDATION permit 10
       match rpki valid
       set local-preference 200
      !
      route-map ROUTE-VALIDATION permit 20
       match rpki not-found
       set local-preference 100
      !
      route-map ROUTE-VALIDATION permit 30
       match rpki invalid
       set local-preference 50
      ```

    - Apply the route-map to inbound updates from your neighbor. Example below for **R20**:

      ```
      router bgp 135540
       address-family ipv4 unicast
        neighbor 172.16.0.13 route-map ROUTE-VALIDATION in
      ```

    - Refresh the routes learned from your neighbor (telling them to resend their routes without tearing down the BGP session). Example below for **R14**:

      ```
      clear bgp ipv4 unicast 172.16.0.1 soft in
      ```

    - Now verify the BGP table (example **R14** below) to see the policy in action: `sh bgp ipv4 unicast`

Figure 9:

End of Lab