RIB Compliance using MDT

NANOG80 HACKATHON

18TH OCTOBER 2020

LAWRENCE BIRD, PETER MOOREY, YORDAN SUTANTO, VLADIMIR YAKOVLEV

HTTPS://GITHUB.COM/PETERMOOREY/NANOG-80-HACKATHON

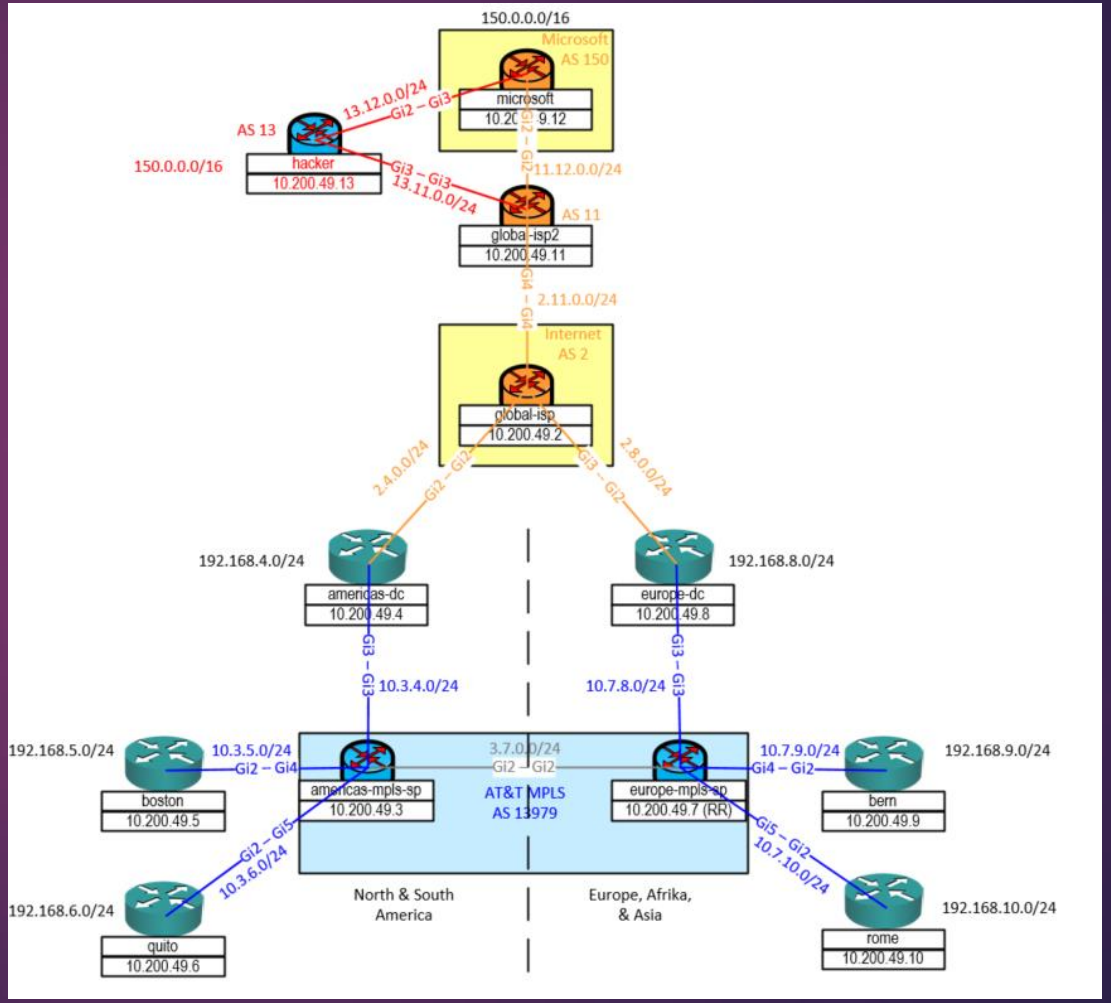
Goal

- Evaluate various technology stacks
- Evaluate network telemetry capabilities
- Develop a network capable of showcasing various routing scenarios
- Provide real-time evaluation of BGP routes to detect:
  - Poorly configured route-maps
  - Incorrect provider policies
  - Route hijacking

Technology Stack

Lab Setup

- Challenges:
  - Are the regional route preferences being honored for the default route?
  - Is traffic to/from our providers being hijacked?



Data Collection

templates > mdt_xml.jinja2

```
<config>
 <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
{% for id, sub in data["subscriptions"].items() %}
  <mdt-subscription>
   <subscription-id>{{ id }}</subscription-id>
   <base>
    <stream>yang-push</stream>
    <source-vrf>{{ sub["sourcevrf"] }}</source-vrf>
    <encoding>encode-kvgpb</encoding>
{% if sub["period"] == "on_change" %}
    <no-synch-on-start>false</no-synch-on-start>
{% else %}
    <period>{{ sub["period"] }}</period>
{% endif %}
    <xpath>{{ sub["xpath"] }}</xpath>
   </base>
   <mdt-receivers>
    <address>{{ sub["rx"]["ip"] }}</address>
    <port>{{ sub["rx"]["tcp_port"] }}</port>
    <protocol>grpc-tcp</protocol>
   </mdt-receivers>
  </mdt-subscription>
{% endfor %}
 </mdt-config-data>
</config>
```

```python
def main(grpc_host):

    hosts = []

    with open("mdt_subscriptions.yml", "r") as f:
        mdt_sub = safe_load(f)

    with open(INVENTORY_FILE, "r") as f:
        inventory = safe_load(f)

    for router in inventory['routers']:
        if inventory['routers'][router]['mdt_bgp']:
            hosts.append(inventory['routers'][router]['ip'])

    for subscription in mdt_sub['subscriptions']:
        mdt_sub['subscriptions'][subscription]['rx']['ip'] = grpc_host
    j2_env = Environment(loader=FileSystemLoader("."), trim_blocks=True, autoescape=True)
    template = j2_env.get_template("templates/mdt_xml.jinja2")
    new_config = template.render(data=mdt_sub)

    for host in hosts:

        connect_params = {
            "host": host,
            "username": USER,
            "password": PWD,
            "hostkey_verify": False,
            "allow_agent": False,
            "look_for_keys": False,
            "device_params": {"name": "csr"},
        }

        with manager.connect(**connect_params) as conn:
            print(f"NETCONF session connected: {host}")

            # Perform the update, and if success, print a message
            config_resp = conn.edit_config(target="running", config=new_config)
            if config_resp.ok:
                print(f"Added ({len(mdt_sub['subscriptions'])}) subscriptions")

        print(f"NETCONF session disconnected: {host}")
```

Policy Evaluation

```yaml
# BGP RIB Compliance Policy file for demo lab

root_source: Cisco-IOS-XE-bgp-oper:bgp-state-data/bgp-route-vrfs/bgp-route-vrf
root_path: bgp_route_af/bgp_route_filter/bgp_route_entry/bgp_path_entry/
regions:
    emea:
        - 10.200.49.9       # Bern
        - 10.200.49.10      # Rome
    americas:
        - 10.200.49.5       # Boston
        - 10.200.49.6       # Quito
    external:
        - 10.200.49.4       # Americas DC
        - 10.200.49.8       # Europe DC
policy:
    0.0.0.0/0:                          # Match default route
        match: explicit                 # Match type is explicit. RIB prefix must match 0.0.0.0/0 exactly
        region: americas                # Region to monitor for RIB updates on
        attributes:                     # List of attributes to evaluate and expected values
            community: 100:1            # Community value expected in RIB update
    0.0.0.0/0:                          # Match default route
        match: explicit                 # Match type is explicit. RIB prefix must match 0.0.0.0/0 exactly
        region: emea                    # Region to monitor for RIB updates on
        attributes:                     # List of attributes to evaluate and expected values
            community: 100:2            # Community value expected in RIB update
    192.168.0.0/16:
        match: any                      # any match will include any network prefix that is within the subnet, or the subnet itself
        region: external
        attributes:
            origin: 100
            # community: 100
            as_path: 13979 13979
    150.0.0.0/16:
        match: any
        region: external
        attributes:
            origin: 150

```

```python
def is_in_policy(self, prefix: str) -> str:
    try:
        target_network = IPNetwork(prefix)

        for prefix, policy in self.policies.items():
            parent_network = IPNetwork(prefix)

            if policy['match'] == 'any':
                if target_network in parent_network:
                    print(f"{target_network} in {parent_network}")
                    return str(parent_network)
            elif policy['match'] == 'explicit':
                    if target_network == parent_network:
                        print(f"{target_network} is {parent_network}")
                        return prefix
            else:
                raise NotImplementedError
    except KeyError:
        print("Malformed Policy")

    return None

def evaluate(self) -> bool:
    query = f'SELECT * FROM "{self.source}" WHERE time > now() - 15s'
    print(query)
    results = self.db.query(query)
    for result in results.get_points(self.source):
        prefix_path = 'bgp_route_af/bgp_route_filter/bgp_route_entry/prefix'
        prefix = result[prefix_path]
        print(f"Prefix is {prefix} on source {result['source']}")

        if prefix:
            compliance = {}
            policy_prefix = self.is_in_policy(prefix)
            print(f"Policy prefix: {policy_prefix}")
            print(f"evaluating prefix {prefix} in policy")
            if policy_prefix:
                policy = self.policies[policy_prefix]

                for key, value in policy['attributes'].items():
                    for attr in result:
                        if 'bgp_neighbor' not in attr:
                            if re.search(f"/{key}$", attr):
                                if result[attr] == value:
                                    compliance[f"{key}_compliance"] = 0
```

Visualization