Enable Dynamic Routing

In the previous example, we configured our IPSec VPN between the Student Pod and the HeadEnd VPN server. At this stage we have a secure path between both networks, but we are not aware of any subnets on the other side of the VPN. We have two options to address this:

- Create Static routes for all the remote networks
- Configure a dynamic routing protocol such as OSPF

While creating static routes is usually an easier answer, we decided that OSPF would be a far more scalable answer to this problem. The advantage of using a dynamic routing protocol is that we can also learn about all the other student pods as they come on line.

Look at the code

In this example, we are going to use a simple Python script to connect to our lab device and commit a single configuration.

To keep things as simple as possible, we have embedded all of our Junos configuration directly into the script itself.

```
from jnpr.junos.utils.config import Config
from jnpr.junos import Device


dev = Device( user='netconf', host='172.16.0.1', password='test123' )
dev.open()

pod_number = "1"

set_cfg = """
set interfaces lo0.0 family inet address 10.255.255.{0}/32
set security zones security-zone trust interfaces lo0.0

set security zones security-zone vpn host-inbound-traffic system-services a
set security zones security-zone vpn host-inbound-traffic protocols ospf

set protocols ospf area 0 interface st0.{0}
set protocols ospf area 0 interface lo0.0 passive
""".format(pod_number)

'''
If you run the script with the wrong variables, the below delete commands s

delete interfaces lo0.0
delete security zones security-zone trust interfaces lo0.0
delete security zones security-zone vpn host-inbound-traffic system-service
delete security zones security-zone vpn host-inbound-traffic protocols ospf
delete protocols ospf
'''

dev.bind(cu=Config)
dev.cu
dev.cu.lock()

dev.cu.load(set_cfg, format='set')

commit_diff = dev.cu.diff()
print "Diff: %s" % (commit_diff)
dev.cu.commit()
dev.cu.unlock()

dev.close()
```

Lets step through the process:

- In the first two lines we tell Python to load both the Configuration elements of the PyEZ library as well as the standard device libraries
- Next we establish a NETCONF session to our lab SRX using fixed credentials
- To make this script a little more scalable we define a variable called "pod_number" and we assign it the value we received from the regisrtation process.
    - **Please modify this value in your editor to match your correct pod number.**
- We create a simple multiline string called "set_cfg" and insert normal set commands we would use on the Junos CLI
    - In this example you can see that we have replaced some of the set commands with "{0}" which is used by Python to indicate that we should insert some variables at this point in the string
    - The ".format(pod_number)" is used to indicate which variable should be inserted into the formatted string.
        - Note: Even though we reference the "{0}" varable twice in the string, we only need to reference the variable once in the format().
- Mistakes are bound to happen, so we have included the commands you may need to paste to the SRX to role back these changes if you made a mistake with your pod number
- The next three lines open a new candidate configuration on the SRX and ensure that we lock the configuration so that nobody else can make changes until we have completed our changes.
- To load the our formatted string into the SRX we call the "dev.cu.load(set_cfg, format='set')" command to send our string and tell Python which format we are using (In this case our string is using Junos set format)
- Following good configuration practices we perform a Junos "show | compare" using the "dev.cu.diff()" function and print that value to screen.
- The remaining lines commit the configuration, unlock the configuration and close the NETCONF session.

NOTE: Best Practice programming should include Exception Handling code to catch any errors, but in the interest of keeping this example simple we have not included those features here.

Running the script

To execute the script for this exercise, ensure that you are in the examples directory and call the script as shown below.

```
vagrant@NetDevOps-Student:~$ cd /vagrant/examples/
vagrant@NetDevOps-Student:/vagrant/examples$
vagrant@NetDevOps-Student:/vagrant/examples$ python ./conf-ospf.py
Diff:
[edit interfaces]
+   lo0 {
+       unit 0 {
+           family inet {
+               address 10.255.255.1/32;
+           }
+       }
+   }
[edit]
+  protocols {
+      ospf {
+          area 0.0.0.0 {
+              interface st0.1;
+              interface lo0.0 {
+                  passive;
+              }
+          }
+      }
+  }
[edit security zones security-zone trust interfaces]
      ge-0/0/1.0 { ... }
+     lo0.0;
[edit security zones security-zone vpn]
+     host-inbound-traffic {
+         system-services {
+             all;
+         }
+         protocols {
+             ospf;
+         }
+     }

vagrant@NetDevOps-Student:/vagrant/examples$
```

If everything works to plan, your output should show the output of the "show | compare" and then return you to the prompt.

Verify Our Changes

From our SRX session we can verify that our changes are applied correctly

```
root@NetDevOps-SRX01> show configuration interfaces lo0
unit 0 {
    family inet {
        address 10.255.255.1/32;
    }
}

root@NetDevOps-SRX01>
root@NetDevOps-SRX01> show configuration protocols
ospf {
    area 0.0.0.0 {
        interface st0.1;
        interface lo0.0 {
            passive;
        }
    }
}

root@NetDevOps-SRX01>
```