Configuring NAT via NETCONF

!!!While this example works it is not recommended to run manually as it can cause insanity!!!

This is an example of how to configure NAT via the NETCONF interface directly. It CAN be run to configure NAT and it will work correctly. However doing so is not for the feint of heart. Be careful if you are copy and pasting XML as there can be issues in incorrectly inserting characters into the NETCONF commands.

Network Address Translation or NAT is a key feature on the SRX. This technology is used to swap IP addresses out in packets to hide internal IP addressing or to use a single IP address that can hide thousands behind it.

In this section of the lab we will be using the manual NETCONF protocol to configure NAT. The goal here is to let you learn how to use NETCONF manually to configure NAT for your lab.

Tools

- NetDevOps VM
- ssh command line
- XML

Network Interface SNAT

To access the rest of the lab you must first configure source NAT or SNAT on your vSRX instance. This will allow the NetDevOps VM to access hosts outside of your own laptop. Each student has the same IP address block between their NetDevOps appliance and the vSRX instance. This allows for a consistent experience for each student so each of the exercises can be completed with the same documentation. The downside of this topology is that your NetDevOps VM can not access the rest of the lab. However we have a vSRX sitting directly in front of the NetDevOps VM. We will use this with SNAT on our external interface in the lab to hide our internal IP address.

Opening a NETCONF session

First lets open up a NETCONF session from the command line of the NetDevOps VM

```
vagrant@NetDevOps-Student:~$ ssh root@172.16.0.1 -s netconf
```

Once connected you will get the typical hello response

Response

```
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<hello>
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</ca
    <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:
    <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</cap
    <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>1985</session-id>
</hello>
]]>]]>
```

To be a good citizen you should also send a hello back to the server. **This step is not required, but it is good to be polite.**

Request

```
<hello>
    <capabilities>
        <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
        <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
        <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-com
        <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0<
        <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?proto
        <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
        <capability>http://xml.juniper.net/dmi/system/1.0</capability>
    </capabilities>
</hello>
]]>]]>
```

Response

```
Junos will not send a response. It already said hello to you.
```

Open configuration for changes

We must now open the configuration so we can load in the NAT configuration. This is the same as if we typed "configure" from the CLI. This gives us a candidate configuration to work with.

Request

```
<rpc message-id="1">
  <open-configuration>
  </open-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt

</rpc-reply>
]]>]]>
```

Configuring NAT

Now that we have a candidate configuration we can load in XML which allows for NAT to occur on the untrust interface. We are specifying the source address to be that of the NetDevOps VM. For simplicity we specify the entire subnet even though there is only a single host.

Request

```
<rpc message-id="2">
    <load-configuration action="merge">
    <configuration>
            <security>
                <nat>
                    <source>
                        <rule-set>
                            <name>LabNAT</name>
                            <from>
                                <zone>trust</zone>
                            </from>
                            <to>
                                <zone>untrust</zone>
                            </to>
                            <rule>
                                <name>1</name>
                                <src-nat-rule-match>
                                <source-address>172.16.0.0/24</source-addre
                                </src-nat-rule-match>
                                <then>
                                    <source-nat>
                                        <interface>
                                        </interface>
                                    </source-nat>
                                </then>
                            </rule>
                        </rule-set>
                    </source>
                </nat>
            </security>
    </configuration>
</load-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <load-configuration-results>
        <ok/>
    </load-configuration-results>
</rpc-reply>
]]>]]>
```

Commit NAT configuration

Now it is time to enact the configuration. This is done with a simple commit. If you get disconnected from your netconf connection your candidate configuration is still open and you can still commit it. This is a nice feature of the Junos configuration model. It leaves the state of the configuration on the device and you optionally do not have to worry about that in your application.

Request

```
<rpc message-id="3">
    <commit-configuration>
        <log>Add SNAT configuration</log>
    </commit-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <ok/>
</rpc-reply>
]]>]]>
```

Verifying NAT

Now the NAT configuration should be correctly applied to your vSRX instance.

Validate connectivity

To validate the connectivity you can issue a ping command from your NetDevOps VM. If everything is working correctly you should see a response from the host.

```
vagrant@NetDevOps-Student:~$ ping 10.10.0.5
PING 10.10.0.10 (10.10.0.5) 56(84) bytes of data.
64 bytes from 10.10.0.5: icmp_seq=1 ttl=64 time=0.482 ms
64 bytes from 10.10.0.5: icmp_seq=2 ttl=64 time=0.480 ms
64 bytes from 10.10.0.5: icmp_seq=3 ttl=64 time=0.520 ms
64 bytes from 10.10.0.5: icmp_seq=4 ttl=64 time=0.548 ms

--- 10.10.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.480/0.507/0.548/0.036 ms
vagrant@NetDevOps-Student:~$
```

If the ping request is not working? Then why?