PSAwise2324Team10Aufgabe02

Network Set Up

Connecting VMs within the same subnet

List network interfaces with `ip a`
You can tell the network adapters apart by comparing their MAC addresses (open VM settings, go to Network and then Advanced to see the MAC address of a certain adapter).
Going forward, we configure the `enp0s8` interface for network communication.
We can temporarily assign new static IP addresses using the command
`sudo ip addr add 192.168.10.2/24 dev enp0s8`
Where `dev` indicates the device to which the new IP address is being added.
For persistent network configuration, there are multiple tools that can be used, such as systemd-networkd and NetworkManager. We chose to use systemd-networkd in combination with the pre-installed netplan tool as we need a lightweight, minimalistic option without a GUI for our Ubuntu server.

Connecting subnets via systemd-networkd and netplan

Navigate to `/etc/netplan`. If there is no configuration file, run `sudo netplan generate` to create a new config file. Open the config file with an editor such as vim and edit it to look like the following configuration snippets. In our case the config file is named `00-installer-config.yaml`

- Router VM config file:
  (192.168.255.254 is the central router of the PSA network, 192.168.10.1 is our subnet's router; 192.168.255.10 is an IP address that belongs to us, within the PSA network)

```
network:
  ethernets:
    enp0s3:
      dhcp4: yes
    enp0s8:
      dhcp4: no
      addresses: [192.168.10.1/24, 192.168.255.10/24]
      routes:
        - to: 192.168.0.0/16
          via: 192.168.255.254
version: 2
renderer: networkd
```

- Other VMs config file:

```
network:
  ethernets:
    enp0s3:
      dhcp4: yes
    enp0s8:
      dhcp4: no
      addresses: [192.168.10.<VM number>/24]
      routes:
        - to: 192.168.0.0/16
          via: 192.168.10.1
version: 2
renderer: networkd
```

We turned off DHCP on this device in order to add static IP addresses and added routes to allow the routers of other teams to be reached.

After saving and closing the file, run `sudo netplan apply` to apply the configuration.

Since the routes we configured here are technically suboptimal (they are required to be like this by the assignment), we also had to turn off ICMP redirects on our team router (VM1). To do this, simply add the following lines to /etc/sysctl.conf, then run `sudo sysctl -p` to apply them and restart the networking service with `sudo systemctl restart systemd-networkd`. Make sure there isn't another config in /etc/sysctl.d that might override these settings. To verify that sending redirects is turned off, you can run `sudo sysctl -a | grep send_redirects`.

```
net.ipv4.ip_forward = 1

# Do not accept ICMP redirects (prevent MITM attacks)
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0

# Do not send ICMP redirects
net.ipv4.conf.all.send_redirects = 0 # this wasn't enough for some reason, so
we also set everything to 0 individually
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.enp0s3.send_redirects = 0
net.ipv4.conf.enp0s8.send_redirects = 0
net.ipv4.conf.lo.send_redirects = 0
```

Proxy

As suggested in the assignment, we have also added the necessary configuration for the usage of a proxy server. Add the following lines to `/etc/environment` :

```
export http_proxy="http://proxy.in.tum.de:8080/"
export https_proxy="http://proxy.in.tum.de:8080/"
export HTTP_PROXY="http://proxy.in.tum.de:8080/"
export HTTPS_PROXY="http://proxy.in.tum.de:8080/"
export ftp_proxy="http://proxy.in.tum.de:8080/"
export FTP_PROXY="http://proxy.in.tum.de:8080/"
```

You can check if the environment variable has been properly added by executing `set | grep -i proxy`.

In order to use `apt` properly with the proxy, you should also create a file /etc/apt/apt.conf.d/95proxies containing this:

```
Acquire::http::Proxy "http://proxy.in.tum.de:8080/";
Acquire::https::Proxy "http://proxy.in.tum.de:8080/";
Acquire::ftp::Proxy "http://proxy.in.tum.de:8080/";
```

After you have finished editing the config files, reboot the VM to make sure the new changes have been applied.

Firewall

We decided to use nftables instead of iptables because it is a new, improved version of it, developed by the same organization.

First of all, install nftables

```
sudo apt update
sudo apt install nftables
```

Don't forget to enable and start the nftables service by running the following commands:

```
systemctl enable nftables
systemctl start nftables
```

To configure a new firewall, you need to create a new table and add each rule to the suited chain (nput, output, forward) using the following command structure:

```
nft (add | create) chain [<family>] <table> <name> [ { type <type> hook <hook> [device <device>] priority <priority> \; [policy <policy> \;] } ]
nft (delete | list | flush) chain [<family>] <table> <name>
```

You can also directly edit the configuration, which is stored in `/etc/nftables.conf`, using a text editor such as vim and directly copy-pasting the configuration below.

We configured the rules for incoming traffic by adding rules to the input chain. The forward chain only deals with forwarded traffic, which is only relevant for the router, so in most of our VMs, it is empty except for the "poliy drop;" statement. What you see below is the firewall for all the machines except the router. The output chain handles outgoing traffic.

All traffic outside our network that doesn't use the proxy, excepting the update servers of Ubuntu, is blocked due to the policy being set to drop. This makes the firewall block all traffic that isn't "whitelisted". To find the IP of the ubuntu update server, we looked up the IP address of the mirror listed in `/etc/apt/sources.list` using `nslookup <url>`.
Once the configuration has been saved, it has to be applied using `sudo systemctl restart nftables`

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
        chain input {
                type filter hook input priority filter; policy drop;

                # Allow ICMP
                ip6 nexthdr icmpv6  accept
                ip protocol icmp  accept

                # Allow DHCP client requests only from our subnet
                udp dport {67,68} ip saddr != 192.168.10.0/24 drop

                # Allow DNS server
                udp sport 53 accept
                # Allow traffic from loopback
                iifname lo accept

                # Allow SSH traffic
                tcp dport 22 accept

                # Allow HTTP (Port 80)
                tcp dport 80  accept

                # Allow HTTPS (Port 443)
                tcp dport 443  accept

                # Allow traffic from proxy server
                ip saddr 131.159.0.2 accept

                # Allow traffic from our subnet
                ip saddr 192.168.0.0/16 accept

                # Allow traffic from Ubuntu update server
                ip saddr 141.30.62.24 tcp dport 80  accept

        }
        chain forward { # for routed traffic
                type filter hook forward priority filter; policy accept;
        }
        chain output {
                type filter hook output priority filter; policy drop;
                # Allow traffic to loopback
                oifname lo accept

                # Allow ICMP
                ip6 nexthdr icmpv6  accept
                ip protocol icmp  accept

                # Allow DHCP server responses
                ip daddr 192.168.10.0/24 udp sport 67 accept

                # Allow DNS server
```

```
                udp dport 53 accept

                # Allow traffic to our subnet
                ip daddr 192.168.0.0/16 accept

                # Allow traffic to subnet of FMI building in Garching
                ip daddr 131.159.0.0/16 accept

                # Redirect traffic to the proxy server
                ip daddr 131.159.0.2 tcp dport 8080

                # Allow ssh response
                tcp sport 22 accept

                # Allow HTTP response
                tcp sport 80  accept

                # Allow HTTPS response
                tcp sport 443  accept

                # Allow traffic to Ubuntu update server
                ip daddr 141.30.62.24 tcp dport 80  accept

                # Drop other DHCP packets
                udp dport {67,68} drop
                udp sport {67, 68} drop

        }
}
```

The router firewall requires forward chain rules on top of the rules in the configuration file below. We decided it would be safest for all input and output rules to appear again in the forward chain. Therefore, the following part has to be added to the router firewall in place of the forward chain:

```
        chain forward { # for routed traffic
                type filter hook forward priority filter; policy accept;

                # Allow ICMP
                ip6 nexthdr icmpv6  accept
                ip protocol icmp  accept

                ## mirror of input rules ##

                # Allow DNS traffic
                udp sport 53 accept

                # Allow traffic from loopback
                iifname lo accept

                # Allow SSH traffic
                tcp dport 22 accept

                # Allow HTTP (Port 80)
                tcp dport 80  accept
```

```
            # Allow HTTPS (Port 443)
            tcp dport 443  accept

            # Allow traffic from proxy server
            ip saddr 131.159.0.2 accept

            # Allow traffic from our subnet
            ip saddr 192.168.0.0/16 accept

            # Allow traffic from Ubuntu update server
            ip saddr 141.30.62.24 tcp dport 80  accept

            ## mirror of output rules ###
            # Allow DNS traffic
            udp dport 53 accept

            # Allow traffic to our subnet
            ip daddr 192.168.0.0/16 accept

            # Allow traffic to subnet of FMI building in Garching
            ip daddr 131.159.0.0/16 accept

            # Redirect traffic to the proxy server
            ip daddr 131.159.0.2 tcp dport 8080

            # Allow ssh response
            tcp sport 22 accept

            # Allow HTTP response
            tcp sport 80  accept

            # Allow HTTPS response
            tcp sport 443  accept

            # Allow traffic to Ubuntu update server
            ip daddr 141.30.62.24 tcp dport 80  accept


    }
```

Testing

```
#!/bin/bash

failed_tests=0

fail() {
    ((failed_tests++))
    echo "FAIL $@"
}

ok() {
    echo "OK $@"
}


pingtest() {
  if ping -4 -c 2 $1 &> /dev/null; then
    ok $2
  else
    fail $3
  fi
}

# Get my own private IP address at interface enp0s8
ip_address=$(ifconfig enp0s8 | grep -oP 'inet \K\S+')
echo "My IP address is $ip_address!"

# Extract last octet of my IP address
last_octet=$(echo $ip_address | awk -F'.' '{print $4}')

# Extract first three octets of my IP address
first_three_octets=$(echo $ip_address | cut -d'.' -f1-3)

# Get my partner's IP address
if [ $last_octet -eq 1 ]; then
 partner_ip="$first_three_octets.2"
else
 partner_ip="$first_three_octets.1"
fi

echo "My partner's IP address is $partner_ip!"

# Test my partner's reachability
pingtest $partner_ip "Ping to partner successful" "Ping to partner failed"

# Test router connection
pingtest 192.168.255.254 "Connection to router is successful" "Connection to
router is not successful"

# Test other team connection
pingtest 192.168.1.1 "Can ping team 1" "Cannot ping team 1"
pingtest 192.168.5.1 "Can ping team 5" "Cannot ping team 5"

# Test firewall
# check if firewall enabled
```

```
if systemctl is-enabled nftables &> /dev/null
then
    ok "Firewall enabled"
else
    fail "Firewall disabled"
fi

# check if firewall is running
if systemctl is-active nftables &> /dev/null
then
    ok "Firewall running"
else
    fail "Firewall not running"
fi

# proxy active
if set | grep -i "proxy" &> /dev/null
then
    ok "Proxy active"
else
    fail "No Proxy"
fi


# web search possible
wget -U "$user_agent" -O - "https://www.google.com/search?q=$search_query"
&>/dev/null
if [ $? -eq 0 ];
then
    echo "OK Web search works"
else
    fail "Web search doesn't work"
fi

# DNS works
pingtest google.com "Can resolve domain names" "Domain name resolution failed"

# port 22 reachable
nc -zv -w 1 localhost 22 &> /dev/null 2>&1
if [ $? -eq 0 ];
then
    ok "Port 22 open"
else
    fail "Port 22 closed"
fi

# other (well known) network ports unreachable
ports_open=0
for port in {1..1024}; do
    if [ $port -ne 22 ];
    then
        nc -zv -w 1 localhost $port &> /dev/null 2>&1
        if [ $? -eq 0 ];
        then
            fail "A port other than 22 is reachable ($port)"
            ((ports_open++))
```

```
        fi
    fi
done
if [ $ports_open -eq 0 ];
then
    ok "No superflous ports reachable"
fi

echo "Tests failed: $failed_tests"
```