Understanding Qubes networking and firewall

Understanding firewalling in Qubes

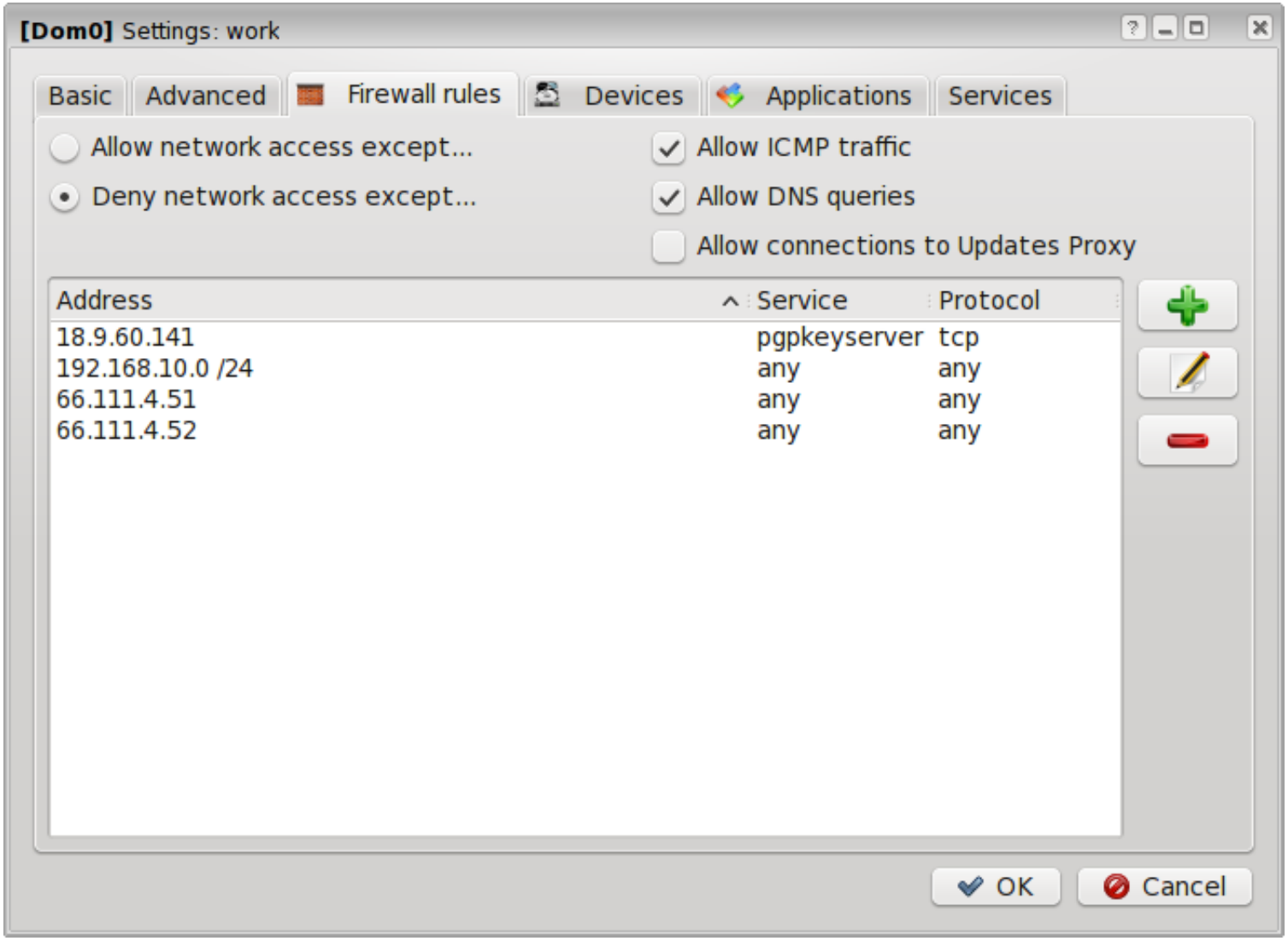
Every qube in Qubes is connected to the network via a FirewallVM, which is used to enforce network-level policies. By default there is one default FirewallVM, but the user is free to create more, if needed.

For more information, see the following:

- **https://groups.google.com/group/qubes-devel/browse_thread/thread/9e231b0e14bf9d62**
- **http://theinvisiblethings.blogspot.com/2011/09/playing-with-qubes-networking-for-fun.html**

How to edit rules

In order to edit rules for a given qube, select it in the Qubes Manager and press the “firewall” button:



Note that if you specify a rule by DNS name it will be resolved to IP(s) *at the moment of applying the rules*, and not on the fly for each new connection. This means it will not work for servers using load balancing. More on this in the message quoted below.

Alternatively, one can use the `qvm-firewall` command from Dom0 to edit the firewall rules by hand. The firewall rules for each VM are saved in an XML file in that VM's directory in dom0:

```
/var/lib/qubes/appvms/<vm-name>/firewall.xml
```

Please note that there is a 3 kB limit to the size of the `iptables` script. This equates to somewhere between 35 and 39 rules. If this limit is exceeded, the qube will not start.

It is possible to work around this limit by enforcing the rules on the qube itself by putting appropriate rules in `/rw/config`. See **below**. In complex cases, it might be appropriate to load a ruleset using `iptables-restore` called from `/rw/config/rc.local`.

Reconnecting VMs after a NetVM reboot

Normally Qubes doesn't let the user stop a NetVM if there are other qubes running which use it as their own NetVM. But in case the NetVM stops for whatever reason (e.g. it crashes, or the user forces its shutdown via qvm-kill via terminal in Dom0), then there is an easy way to restore the connection to the NetVM by issuing:

```
qvm-prefs <vm> -s netvm <netvm>
```

Normally qubes do not connect directly to the actual NetVM which has networking devices, but rather to the default sys-firewall first, and in most cases it would be the NetVM that will crash, e.g. in response to S3 sleep/restore or other issues with WiFi drivers. In that case it is only necessary to issue the above command once, for the sys-firewall (this assumes default VM-naming used by the default Qubes installation):

```
qvm-prefs sys-firewall -s netvm netvm
```

Enabling networking between two qubes

Normally any networking traffic between qubes is prohibited for security reasons. However, in special situations, one might want to selectively allow specific qubes to establish networking connectivity between each other. For example, this might be useful in some development work, when one wants to test networking code, or to allow file exchange between HVM domains (which do not have Qubes tools installed) via SMB/scp/NFS protocols.

In order to allow networking between qubes A and B follow these steps:

- Make sure both A and B are connected to the same firewall vm (by default all VMs use the same firewall VM).
- Note the Qubes IP addresses assigned to both qubes. This can be done using the `qvm-ls -n` command, or via the Qubes Manager preferences pane for each qube.
- Start both qubes, and also open a terminal in the firewall VM
- In the firewall VM's terminal enter the following iptables rule:

```
sudo iptables -I FORWARD 2 -s <IP address of A> -d <IP address of B> -j ACCEPT
```

- In qube B's terminal enter the following iptables rule:

```
sudo iptables -I INPUT -s <IP address of A> -j ACCEPT
```

- Now you should be able to reach B from A – test it using e.g. ping issued from A. Note however, that this doesn't allow you to reach A from B – for this you would need two more rules, with A and B swapped.
- If everything works as expected, then the above iptables rules should be written into firewallVM's `qubes-firewall-user-script` script which is run on every firewall update, and A and B's `rc.local` script which is run when the qube is launched. The `qubes-firewall-user-script` is necessary because Qubes orders every firewallVM to update all the rules whenever a new connected qube is started. If we didn't enter our rules into this "hook" script, then shortly our custom rules would disappear and inter-VM networking would stop working. Here's an example how to update the script (note that, by default, there is no script file present, so we will probably be creating it, unless we had some other custom rules defined earlier in this firewallVM):

```
[user@sys-firewall ~]$ sudo bash
[root@sys-firewall user]# echo "iptables -I FORWARD 2 -s 10.137.2.25 -d 10.137.2.6 -j ACCEPT" >> /rw/config/qubes-firew
[root@sys-firewall user]# chmod +x /rw/config/qubes-firewall-user-script
```

- Here is an example how to update `rc.local`:

```
[user@B ~]$ sudo bash
[root@B user]# echo "iptables -I INPUT -s 10.137.2.25 -j ACCEPT" >> /rw/config/rc.local
[root@B user]# chmod +x /rw/config/rc.local
```

Port forwarding to a qube from the outside world

In order to allow a service present in a qube to be exposed to the outside world in the default setup (where the qube has sys-firewall as network VM, which in turn has sys-net as network VM) the following needs to be done:

- In the sys-net VM:
  - Route packets from the outside world to the sys-firewall VM
  - Allow packets through the sys-net VM firewall
- In the sys-firewall VM:
  - Route packets from the sys-net VM to the VM
  - Allow packets through the sys-firewall VM firewall
- In the qube:
  - Allow packets in the qube firewall to reach the service

As an example we can take the use case of a web server listening on port 443 that we want to expose on our physical interface eth0, but only to our local network 192.168.x.0/24.

1. Route packets from the outside world to the FirewallVM

From a Terminal window in sys-net VM, take note of the 'Interface name' and 'IP address' on which you want to expose your service (i.e. eth0, 192.168.x.x)

```
ifconfig | grep -i cast
```

> Note: The vifx.0 interface is the one connected to your sys-firewall VM so it is *not* an outside world interface…

From a Terminal window in sys-firewall VM, take note of the 'IP address' for interface Eth0

```
ifconfig | grep -i cast
```

Back into the sys-net VM's Terminal, code a natting firewall rule to route traffic on the outside interface for the service to the sys-firewall VM

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -d 192.168.x.x -j DNAT --to-destination 10.137.1.x
```

Code the appropriate new filtering firewall rule to allow new connections for the service

```
iptables -I FORWARD 2 -i eth0 -d 10.137.1.x -p tcp --dport 443 -m conntrack --ctstate NEW -j ACCEPT
```

> Note: If you want to expose the service on multiple interfaces, repeat the steps described in part 1 for each interface

Verify you are cutting through the sys-net VM firewall by looking at its counters (column 2)

```
iptables -t nat -L -v -n
```

```
iptables -L -v -n
```

Try to connect to the service from an external device

```
telnet 192.168.x.x 443
```

Once you have confirmed that the counters increase, store these command in '/rw/config/rc.local'

```
sudo nano /rw/config/rc.local
```

```
#!/bin/sh


####################
# My service routing
```

```
# Create a new firewall natting chain for my service
if iptables -t nat -N MY-HTTPS; then

# Add a natting rule if it did not exit (to avoid cluter if script executed multiple times)
  iptables -t nat -A MY-HTTPS -j DNAT --to-destination 10.137.1.x

fi


# If no prerouting rule exist for my service
if ! iptables -t nat -n -L PREROUTING | grep --quiet MY-HTTPS; then

# add a natting rule for the traffic (same reason)
  iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -d 192.168.0.x -j MY-HTTPS
fi


######################
# My service filtering

# Create a new firewall filtering chain for my service
if iptables -N MY-HTTPS; then

# Add a filtering rule if it did not exit (to avoid cluter if script executed multiple times)
  iptables -A MY-HTTPS -s 192.168.x.0/24 -j ACCEPT

fi

# If no prerouting rule exist for my service
if ! iptables -n -L FORWARD | grep --quiet MY-HTTPS; then

# add a natting rule for the traffic (same reason)
  iptables -I FORWARD 2 -d 10.137.1.x -p tcp --dport 443 -m conntrack --ctstate NEW -j MY-HTTPS
```

```
fi
```

Finally make this file executable, so it runs at each boot

```
sudo chmod +x /rw/config/rc.local
```

2. Route packets from the FirewallVM to the VM

From a Terminal window in the VM, take note of the 'IP address' for interface Eth0 (i.e. 10.137.2.x)

```
ifconfig | grep -i cast
```

Back into the sys-firewall VM's Terminal, code a natting firewall rule to route traffic on its outside interface for the service to the qube

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -d 10.137.1.x -j DNAT --to-destination 10.137.2.y
```

Code the appropriate new filtering firewall rule to allow new connections for the service

```
iptables -I FORWARD 2 -i eth0 -s 192.168.0.0/24 -d 10.137.2.y -p tcp --dport 443 -m conntrack --ctstate NEW -j ACCEPT
```

> Note: If you do not wish to limit the IP addresses connecting to the service, remove the ` -s 192.168.0.1/24 `

Once you have confirmed that the counters increase, store these command in '/rw/config/qubes-firewall-user-script'

```
#!/bin/sh


####################
# My service routing
```

```
# Create a new firewall natting chain for my service
if iptables -t nat -N MY-HTTPS; then

# Add a natting rule if it did not exit (to avoid cluter if script executed multiple times)
  iptables -t nat -A MY-HTTPS -j DNAT --to-destination 10.137.2.x

fi


# If no prerouting rule exist for my service
if ! iptables -t nat -n -L PREROUTING | grep --quiet MY-HTTPS; then

# add a natting rule for the traffic (same reason)
  iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -d 10.137.1.x -j MY-HTTPS
fi


######################
# My service filtering

# Create a new firewall filtering chain for my service
if iptables -N MY-HTTPS; then

# Add a filtering rule if it did not exit (to avoid cluter if script executed multiple times)
  iptables -A MY-HTTPS -s 192.168.x.0/24 -j ACCEPT

fi

# If no prerouting rule exist for my service
if ! iptables -n -L FORWARD | grep --quiet MY-HTTPS; then

# add a natting rule for the traffic (same reason)
  iptables -I FORWARD 4 -d 10.137.2.x -p tcp --dport 443 -m conntrack --ctstate NEW -j MY-HTTPS
```

```
fi
```

Finally make this file executable (so it runs at every Firewall VM update)

```
sudo chmod +x /rw/config/qubes-firewall-user-script
```

3. Allow packets into the qube to reach the service

Here no routing is required, only filtering. Proceed in the same way as above but store the filtering rule in the '/rw/config/rc.local' script.

```
######################
# My service filtering

# Create a new firewall filtering chain for my service
if iptables -N MY-HTTPS; then

# Add a filtering rule if it did not exit (to avoid cluter if script executed multiple times)
  iptables -A MY-HTTPS -s 192.168.x.0/24 -j ACCEPT

fi

# If no prerouting rule exist for my service
if ! iptables -n -L FORWARD | grep --quiet MY-HTTPS; then

# add a natting rule for the traffic (same reason)
  iptables -I INPUT 5 -d 10.137.2.x -p tcp --dport 443 -m conntrack --ctstate NEW -j MY-HTTPS

fi
```

This time testing should allow connectivity to the service as long as the service is up :-)

Where to put firewall rules

Implicit in the above example **scripts**, but worth calling attention to: for all qubes *except* ProxyVMs, iptables commands should be added to the `/rw/config/rc.local` script. For ProxyVMs (`sys-firewall` inclusive), iptables commands should be added to `/rw/config/qubes-firewall-user-script`. This is because a ProxyVM is constantly adjusting its firewall, and therefore initial settings from `rc.local` do not persist.