CS60: Setting up your virtual machine environment

Sergey Bratus, Spring 2017

April 20, 2017

Why? The purpose of this setup is to give you a Linux virtual machine where you (a) have root and can use IP raw sockets and run Wireshark, (b) can configure networking from the ground up, (c) can use this machine as if it were on a LAN with your main computer (instead of running in a VM on it). That way you can ping it, connect to it, and send it packets, to which it would reply however you program it to.

This setup is only slightly more complex than a typical VM setup.

What? We'll be using VirtualBox (`http://virtualbox.org/`) and Ubuntu Linux. If you strongly prefer another Linux distribution, talk to me; I may be able to accommodate your preference after some testing.

Differences from standard Ubuntu configuration. Our VM comes with NetworkManager disabled. NetworkManager is a very intrusive program that takes over network interfaces; doing non-trivial networking with it running is hardly possible.

Also, your VM's interface will be called *eth0*, as it used to in standard Linux. *Systemd* authors recently pushed through a scheme that renames network interfaces to names such as *enp1s3*, which means "the Ethernet interface connected to the PCI bus number 1, slot 3". Whatever the reasons for this renaming scheme,[1] it is silly for VMs where the PCI bus is purely imaginary (i.e., emulated, like everything else). So I reverted the VM back to the standard Linux naming.

How to configure VirtualBox networking. There are two ways here, one fast, another more instructive.

Bridged mode: The easiest way to configure networking in VirtualBox is to set it to "Bridged Adapter". You can do this while your VM is running; click on the little two-computer icon on the bottom bar of VirtualBox or from the menu bar follow *Machine* > *Settings* > *Network*. Choose *Bridged Adapter* and select a physical network interface of your computer to bridge with.

The VM will try to share this interface with your computer; most of the time this works—except for "Dartmouth Public", where it doesn't (argh!). Your laptop and you VM will have two different IP addresses on you LAN, and you will be able to ping one from the other.

To get full connectivity in bridged mode you just need to run a DHCP client and request an IP and other parts of a network configuration such as the default gateway and the DNS servers.

Run `dhclient -v eth0` and watch the show. You should see a DHCPDISCOVER and/or a DHCPREQUEST going out, a DHCPOFFER in response, and a DHCPACK with your new shiny IP address on the same LAN as your host computer. If you only see requests but no responses, you may be on Dartmouth Public, or similar wireless network where such bridging doesn't work; then consider either using wired Ethernet and bridging with its interface, or switch to Plan B, Host-only Adapter.

If, however, you get an IP via DHCP, you should be able to ping that IP from the host's terminal, and see pings going back and forth; you should also see them by running `tcpdump -i eth0 -n` inside your VM.

Host-only Adapter: This mode is a bit more complex. You will need to create a new virtual interface by going through *VirtualBox VM* > *Preferences* > *Network* > *Host-only Network* tab. As you add an interface, it will be called something like *vboxnet0*. Once you create it, you can edit its properties (by clicking on the "screwdriver" icon in the tab). This lets you set the network address of that interface on your own (host)

[1] See discussion here: `https://github.com/systemd/systemd/issues/3715`.

computer, and also sets the LAN. Usually, you'd see something like **192.168.56.1** and **255.255.255.0** set there.

This means your computer will have an additional network interface with the address of 192.168.56.1 and an emulated Ethernet LAN connected to it. Your VM will look to your computer like it is connected to that LAN. Do not set up the DHCP server in the neighboring tab; we will configure all interfaces manually.

Set up *eth0* in the VM: `ifconfig eth0 192.168.56.100 up` (or use some other address on this LAN). After this, you should be able to ping your VM from your host computer with `ping 192.168.56.100`. You should see the pings and responses coming in on the VM if you run `tcpdump -i eth0 -n` at the VM's root shell.

So your VM has connectivity to the host, but not to the outside. While this should be enough for Lab 3, you can do more and configure your host to be the gateway/default router and NAT for the VM, so that it can connect to the Internet.

BTW, if instead of Host-only Adapter you configure a *NAT Network*, you will get much easier connectivity (NAT for your VM will be set up automatically, without having to set it manually below. However, your host computer will not have an interface on that network; only other VMs (if any) will appear on that LAN. This will make testing connectivity to your VM awkward, as you would need to start another VM and join the same NAT network, rather than simply working from the hosts terminal.

Configuring full connectivity with Host-mode Adapter on MacOS Inside the VM, you need to set up the virtual interface *vboxnet0* as the default gateway. `route add default gw 192.168.56.1` should do the job. After that, `route -n` should show two routes,

```
0.0.0.0         192.168.56.1    0.0.0.0         UG    0   0   0  eth0
192.168.56.0    0.0.0.0         255.255.255.0   U     0   0   0  eth0
```

(which means that destination IPs in 192.168.56.0/24 are for direct delivery, their MACs found via ARP, and everything else goes to 192.168.56.1, with its MAC in the Ethernet header, no matter what the destination IP.)

At this point, if you `ping 8.8.8.8` from the VM and run `tcpdump -i vboxnet0 -n` on the host, you should see the pings but, of course, not responses. For that, you need to configure your Mac to act as a forwarder/router *and* a NAT box. This is like "Internet sharing", but MacOS GUIs won't let you configure sharing for the likes of *vboxnet0*. You need to do so with the BSD command line tools that Darwin/MacOS inherited from BSD.

Here is how I did it. First, forwarding:

```
# sudo sysctl -w net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1

# sudo sysctl -w net.inet.ip.fw.enable=1
net.inet.ip.fw.enable: 1 -> 1
```

Then NAT. NAT is provided by the famous FreeBSD utility PF (PacketFilter).[2] On MacOS, PF is the basis of the MacOS firewall. The following commands first write the NAT rule into a file, then flush any existing NAT rules, then apply the new one.

```
// To see if any NAT rules are active:

# pfctl -s nat
No ALTQ support in kernel
ALTQ related functions disabled
nat on en4 inet from 192.168.56.0/24 to any -> (en4) round-robin

// This says translate every packet that would leave on en4 and
```

[2]See `https://pleiades.ucsc.edu/hyades/PF_on_Mac_OS_X` for details, `http://krypted.com/mac-security/a-cheat-sheet-for-using-pf-in-os-x-lion-and-up/` for some quick rules, `https://murusfirewall.com/Documentation/OS+X+PF+Manual.pdf` for more.

```
//   has a source IP matching  192.168.56.0/24 to the IP address
//   currently on en4. But I need en0, my wireless interface, not en4,
//   which was my Thunderbold Ethernet adapter.

// To flush these rules:

# pfctl -F nat
No ALTQ support in kernel
ALTQ related functions disabled
nat cleared

// Make new rule:

# echo "nat on en0 from 192.168.56.1/24 to any -> (en0)" > nat.rules

# cat nat.rules
nat on en0 from 192.168.56.1/24 to any -> (en0)

// Install the new rule into PF:

# pfctl -f nat.rules -e
pfctl: Use of -f option, could result in flushing of rules
present in the main ruleset added by the system at startup.
See /etc/pf.conf for further details.

No ALTQ support in kernel
ALTQ related functions disabled
pfctl: pf already enabled

// And check that they took:

# pfctl -s nat
No ALTQ support in kernel
ALTQ related functions disabled
nat on en0 inet from 192.168.56.0/24 to any -> (en0) round-robin
```

And now you should be able to ping 8.8.8.8 from your VM and should see ICMP packets going back and forth through *vboxnet0*.

Now you only need to configure a DNS server in the VM. Put `nameserver 8.8.8.8` into `/etc/resolv.conf`. Note: this file gets clobbered by `dhclient`, so check its contents if your DNS name lookup doesn't seem to work.

I often check if my `dhclient` is running, with `ps ax | grep dh`, and kill it with `killall dhclient` if I don't need it, i.e., if my configuration is currently static (but was dynamic on some prior network).