Notes from Exploring Packets

Instructions

Make a copy of this document, rename it to “exploring-packets-notes” and move it to your CSE 523 Google Docs collection. If at any point in this exercise you feel stuck, raise your hand and get some guidance. When you reach each GATE below, switch over to the Tracking Progress document and update your position. Try to be efficient with your time.

Overview

Today we will explore how to log, inspect, and manipulate network packets generated by both the loopback and data interfaces in your VM. Keep detailed notes below (place your comments in between the provided horizontal lines); you will be referring to these in the future to do your work.

Part 1: Logging and examining packets

For this activity, you will be working with 2 Ubuntu VMs with X forwarding. Unfortunately, we need to do some configuration before our VMs are in working order.

Download this file cse523.onl into your class VM, and move it into the ~/.onldir directory. This directory should have been created from Homework 2, but if for some reason you do not have the directory run RLI.jar (which will automatically create it) and then close it immediately.

The file cse523.onl is the experiment file you will be using for this in-class assignment. Open a fresh RLI.jar, set up the tunnel, and open this file in the RLI.

You should see a topology with two virtual machines, connected by a shared link. Go through the normal reservation and commit process that you learned in Homework 2. Wait for the commit to complete.

Note that I have configured the password for the virtual machines to be ‘password’ without the quotes; you will need that soon. You can check this via the Topology menu. In that menu there is a “Show VM Password” option.

In a terminal window that is SSHed into the base ONL directory, paste and run the following command:

```
wget --no-check-certificate 'https://docs.google.com/uc?
export=download&id=0B7PIdb_FGgbsWlNKVURoMjBFWDg' -O onl_scripts.tar.gz
```

This will download configuration scripts to your base level directory in ONL.

Unpack the tarball with the command

```
tar zxvf onl_scripts.tar.gz
```

You should see “config.sh” and “vm_setup.sh”. Once your VMs are up and running, run the command

```
./vm_setup.sh password
```

to

1) setup passwordless SSH login to your VMs and
2) to place the necessary configure script (config.sh) in the base directory of each VM instance.

Now you want to SSH into each of your two VMs. If you click on the center of the VMs in the RLI topology, you can see the “VM host” name of each virtual machine, e.g. “vm12c01v01”. You can use this name directly, i.e. without the dollar sign, when SSHing. When you first SSH into each VM, you will need to run

```
sudo ./config.sh
```

and enter the password when prompted. You will see a funny response to your sudo invocation:

```
sudo: unable to resolve host vm12c01v01
```

This is what the script is trying to fix. After the script has run, type “sudo ls” and ensure that you receive proper “ls” output and not the weird message again.

Once you have completed the configuration steps, enter the current date and time below.

To start, we are going to use a single VM. Select one of the two Ubuntu VM’s, and note the hostname, MAC address, and IP address in the space below. Do this for the data interface only.

GATE 1

1.1 tcpdump and wireshark basics

To start capturing packets, we will need to use the program tcpdump in a separate terminal window. Open a new SSH connection to the ONL base directory with X-forwarding enabled. Note: this is done by supplying the -X flag to your normal SSH command. Now SSH into the **same VM** as the previous window, again with X-forwarding enabled. The first command again verifies that you can use the sudo command correctly.

```
sudo ls
sudo tcpdump -i control -w packets.pcap
```

As tcpdump is running, it creates a record of all packets being sent on the control interface, including the traffic generated by your ssh session. In the other SSH window to the same VM, run the command

```
pgrep tcpdump
```

and make note of the process ID (pid) returned by the command. Toggle back to the ssh connection running tcpdump and hit "^C", i.e. CTRL+C, to stop the packet capture. Record below the tcpdump pid and how many packets were captured by the tcpdump session.

At this point, the file "packets.pcap" should contain information about the packets sent while tcpdump was active. We can look at these packets using the program wireshark, which can be started with:

```
wireshark packets.pcap
```

Wireshark displays three panes of information (top, middle, bottom). Take a look at these, and describe below what kind of information is contained in each of the three panes.

What packet protocols do you see represented in the top pane?

In the top pane, you should see multiple SSH packets. Click one. Now, look at the second pane, (you may need to do some window pane resizing) and explain in the space below the protocol layers that you see being used in this packet.

Close wireshark.

GATE 2

1.2 Generate and examine traffic

For this part of the exercise, we will observe traffic generated by visiting a website. Your Ubuntu VM is isolated from the regular Internet, so we will use a page generated by your VM. Start a new tcpdump capture with the following command:

```
sudo tcpdump -i lo -w packets.pcap
```

Notice that this time we are using the loopback (`lo`) interface because we are dealing with traffic

that never leaves the VM. Now, in the other SSH window, we'll use the command-line tool wget to download the webpage hosted on the VM:

```
wget localhost
```

Stop the tcpdump capture. Restart wireshark and open packets.pcap. Record below the number of packets that were recorded, and list the protocol types you see.

Among the TCP packets, you should see one or two with protocol HTTP. Right-click the first HTTP packet in the top pane, and choose "Follow TCP Stream." Explain what you see below; also copy-paste the text in red following your explanation below.

Close the TCP Stream window. You should see a sequence of TCP packets around your HTTP packets. Look at the first three TCP packets, and note their ports and directions. Explain below what these three packets are doing.

Enter the string "http.response" in the filter text field (erasing what used to be there), and apply. Click the first packet. In the second pane, notice the last line in the pane: "Line-based text data: text/html." Right-click that line, choose Copy -> Bytes (Printable Text Only). Copy what you see below, then explain what the text is.

Exit wireshark.

GATE 3

Part 2: Crafting and sending packets

2.1 Scapy basics

Scapy is a Python-based tool for creating, sending and receiving packets. Start it in a console window as follows.

```
sudo scapy
```

Scapy is based on Python, so you can use Python syntax on the scapy command line. Use the command ls() to list supported network protocols. Similarly, lsc() lists available commands.

List the supported commands (ie, the lsc() output) below.

Scapy has a quirky syntax that takes some getting used to. You can create an IP packet and display its contents as follows.

```
p=IP()
p.show()
```

List the output below.

As you can see, scapy uses default values for fields. You can also set them with dot-notation.

```
p.src="128.252.19.221"
```


You can build packets up by layer by using the divide operator, /, as follows.

```
e=Ether()
p=IP()
t=TCP()
pkt=e/p/t
pkt.show()
```


Copy the pkt.show() output below. (Also note that you don't have to maintain variables for each layer; you could also use pkt=Ether()/IP()/TCP().)

Scapy also makes it easy to create sets of packets. For example, enter the following on the scapy command line.

```
pl=IP(dst="128.252.19.0/30")
for p in pl:
    print p.dst

```

Include and explain the output below. If you have trouble using tabs in scapy, any number of spaces will work. They logically deduce which scope block a line is in. (There is an extra blank line after the 'print p.dst' command)

You can find more information about scapy and its usage here.

GATE 4

2.2 Sending and receiving packets setup

Scapy includes a number of commands to send and receive packets; you likely spotted several of them in the lsc() output you captured above.

To send and receive one or more IP packets, use the sr() command. For Ethernet frames, use srp().

Unlike the previous sections of this exercise, we will now send packets over our VM network using the data interfaces. Record the hostname, MAC address, and IP address of the other VM's data interface in the space below.

For the rest of the exercise, this document will refer to the VM you've been using for Gates 1-4 as <VM1>, and the second VM as <VM2>. When you see either <VM1> or <VM2> in a command, replace it with the relevant IP address.

To test that everything is working correctly, log back into <VM1> and give the following command:


```
ping -c 5 <VM2>
```

If you have 0% packet loss, then everything is working correctly. If you lose any packets, alert the instructor or TA.

GATE 5

2.3 Scanning

Remain logged into <VM1> and start scapy with "sudo scapy". Scan port 80 at <VM2> with the following command. Note, sr1() sends a packet, but returns just the first response.

```
sr1(IP(dst="<VM2's IP address>")/TCP(dport=80,flags="S"))
```

Explain what you see in the response.

You can use the following command to scan a local IP network to check for machines listening

on port 80; the command sends a single TCP SYN packet. You have to replace the IP.dst network to match the one you are on. For the commands below, wait a little after issuing the first command. Between the first and second command, the <Ctrl-c> indicates that you should press both Ctrl and c on the keyboard, and not type out "<Ctrl-c> in scapy.

```
ans,unans=sr(IP(dst="192.168.1.0/30")/TCP(dport=80,flags="S") )
<Ctrl-c>
ans.summary(lambda(s,r): r.sprintf("%IP.src% %TCP.sport% is alive") )
```

Note the Ctrl-C in there. Run the command, and copy your output below.

Unfortunately, due to a current bug in scapy, the scan will not pick up the HTTP server running on localhost despite it being part of the local network.

You can also scan for a range of ports. The following command targets a single machine, but scans ports 3790 through 3794.

```
ans,unans=sr(IP(dst="<VM2>")/TCP(dport=(3790,3794),flags="S") )
ans.summary(lambda(s,r): r.sprintf("%IP.src% %TCP.sport% %TCP.flags%
(RA:closed, SA:open)") )
```


Run the command, and copy your output below.

To find all hosts on an Ethernet, use the following. (Note again that you may need to modify the destination network to match the one your VM is on.)

```
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/
ARP(pdst="192.168.1.0/24"),timeout=2)
ans.summary(lambda (s,r): r.sprintf("%Ether.src% %ARP.psrc%") )
#same as
#arping("192.168.1.0/24")
```

Run the command, and copy your output below.

Gate 6

2.4 ARP spoofing

It is a curious fact that ARP is a stateless protocol, so you can send ARP responses to

machines that never sent requests! So, we can use scapy to send a forged ARP response to a machine to make it think that your MAC is the gateway IP's MAC. Thus, all traffic from the target machine destined for the Internet will come to you.

This will not work in the particular ONL experiment we designed because it does not have a default gateway / router. The class size is too big to have that many individual routers for an in-class assignment. However, with some simplifying assumptions, we can still see how ARP can be spoofed.

Let's assume that there *is* a gateway router in our ONL network, and its IP address is 192.168.1.3. We can force VM2 to think that VM1 is the default gateway, so it will send all of its Internet traffic to our machine. In a real setting, at this point there are a couple of options. We can store a copy of those packets for later analysis, and send the original packets onward to the true default gateway. Alternatively, we could perform a denial of service (DoS) attack by dropping / ignoring all of the packets sent to us. However, in ONL it will be sufficient to convince VM2 that we are the default gateway.

Though you've likely gotten this information in above sections, we will aggregate it down here. Fill out the configuration lines below:

```
VICTIM IP:  <VM2's IP>
VICTIM MAC: <VM2's MAC>
GATEWAY IP: 192.168.1.3
MY MAC:     <VM1's MAC>
```


Scapy will actually provide MY MAC by default, but it is nice to see all of the information necessary for the ARP response. Given the assumptions above, the following scapy code will create an ARP packet that when sent will force VICTIM to send its Internet traffic to my machine. **Do not run it just yet though.**

```
pp=Ether(dst="<VM2's MAC>")/ARP(op="who-has", psrc="192.168.1.3",
pdst="<VM2's IP>")
pp.show()
sendp(pp)
```

Make sure you understand what this code is accomplishing. If not, ask the instructor or TA for help. Before we actually run this code, we want to observe the change in state on the victim machine. On VM2, run the command `arp -a` to see the contents of its ARP cache, and paste the results below.

Before:

Now, switch over to VM1 and run the scapy code we outlined above (after filling in the

necessary information).

Switch back to VM2, run the command `arp -a` again, and paste the results below. Make the newly added entry bold.

After:

When you are all done, please close your ONL experiment and cancel your reservation to free the resources for other people to use.

COMPLETE