College of Computer and Information Science
Northeastern University
CS4700 & CS5700: Fundamentals of Computer Networks

Fall 2011

Problem Set 6 (due November 28, 2010).
[200 points]

Important Notes:

1. Late submissions will result in a 10% penalty per day (e.g., 2.5 days late result in 25% penalty).
2. You can use the Internet to get some help, but you should use your own words, examples, and code when answering the questions.
3. All students should do PART I. Undergraduate students have the option to do either PART II or PART III (teams who do PART III *instead* of PART II will get 20 extra credit points). Graduate students are required to do PART III (teams who do PART II *in addition* to PART III will get 20 extra credit points plus an additional 20 points if they also do the bonus assignment).

PART I (all students): Using the TUN driver and RAW sockets

Step 1 (20 points): Environment Setup
Setup a minimal linux virtual machine using VirtualBox (https://www.virtualbox.org/). A minimal debian installation is less than 1GB in size (e.g., www.debian.org using the "Smaller CDs" network installation http://www.debian.org/distrib/netinst).

Notes:

- You don't need to install the gui but need to an **`ssh`** server running on the VM.
- Make sure that you have the gcc compiler installed and an editor (e.g., **`vi`**). You will also need tools such **`ifconfig`**, and **`tcpdump`**. All can be installed from the comman line using for example the **`apt-get`** command.

Step 2 (40 points): Using the TUN device/driver
First read some general information about the TUN device/driver on the Wikipedia entry:
http://en.wikipedia.org/wiki/TUN/TAP

You can find more specific information about the TUN device/driver in Linux by reading the following document, which comes with Linux kernel:
http://lxr.linux.no/linux+v2.6.33/Documentation/networking/tuntap.txt

At this point, you should have some knowledge about what the TUN interface is. Next, download the template code for this problem from the following address:
http://www.ccs.neu.edu/home/noubir/Courses/CS4700-5700/F11/problems/PS6/tun.c

Note: you can use **`wget`** from inside the VM to download it to VM directly, or you can download it to your current directory and then copy it to VM using **`sftp`**).

Carefully, go through the tun.c code. This program allocates the next available TUN interface, reads every packet that comes in and prints its size. In you current terminal (let's call it terminal 1), compile the code as follows:

```
gcc -Wall tun.c -o tun
```

Open another SSH connection to the VM (call it terminal 2) and run the program with root privileges e.g.,

```
sudo ./tun
```

This will allocate the first tunneling interface, called **tun0**. In terminal 1, run the following command:

```
sudo ifconfig tun0 55.55.55.55 netmask 0xffffff00 up
```

This will assign the IP address 55.55.55.55 to the tun0 interface and will also add a route to the routing table, to route all the traffic destined for the 55.55.55.0/24 network through the tun0 interface. Now you can send traffic to any host in the 55.55.55.0/24 network and the tun program will intercept them and print the packet sizes on the screen, e.g. if you run the following command in terminal 2:

```
ping 55.55.55.155
```

You can observe the following output in terminal 1:

```
read 84 bytes from tunnel interface tun0.

read 84 bytes from tunnel interface tun0.
...
```

Assignment: As you can see, the tun program prints just the size of the packet. You should modify the code in tun.c to print out detailed information about the packet, including protocol, packet length, source and destination ports in case it's a TCP/UDP packet. You will need to submit your modified tun.c program and a report showing a typical execution of the program (this part should be submitted under folder **TUN**).

Step 3 (40 points): Using Raw Sockets

First, get some basic understanding of raw sockets work by reading the following tutorial:

http://www.enderunix.org/docs/en/rawipspoof/

Note that there are many raw sockets tutorials on the Internet, you can read anyone that you find more understandable to you.

Next, download the sample code for this problem from the following address:
http://www.ccs.neu.edu/home/noubir/Courses/CS4700-5700/F11/problems/PS6/icmp.c

This program creates an ICMP packet with a hard coded destination, sends it and prints a message if it receives a response to the packet it sent. It is a simple example demonstrating the use of raw sockets.

Assignment: write two different programs that will simulate a UDP client and server. You will write the UDP client using raw sockets, specifically, it will create and send a UDP packet to port 3333 of the IP on which your server will be running. And you will also write a UDP server, which will use the regular sockets API. It will listen on port 3333 and print information about the received packet, specifically its source IP and source port. Submit both your code, and a report showing a typical execution of your program (this part should be submitted under folder **RAW-SOCKETS**).

As a fun exercise, in your client, you can set the source IP to be something other than that of your client – you are using raw sockets, you have control over that – and observe the server to print as a source IP something different from the actual client IP.

Note that in this exercise you can run the server part of your project in lab machines, or you can run both the client and the server on the VM, but you cannot run the client on a lab machine since creating a raw socket requires administrative privileges.

Hints for debugging: you can use **`tcpdump`** (similar to wireshark without the gui) to see the packets you generated. Be careful about the order of bytes (little/big endian) when specifying the ports information.

PART II. (100 points) Port Forwarding (only for undergraduate students).

In this assignment you will implement a TCP tunnel in Java. Imagine you maintain your own server at home which also acts as a mail server. You are running a POP3 server on it and use your favorite POP3 client to access your email when you are away. The details of POP3 protocol are irrelevant, you can think of POP3 server as a TCP server running on port 110 and it gives access to your mailbox if you connect to it with a POP3 client. The problem with POP3 is that it is a plain-text protocol, i.e. if someone was looking at the wire, they could see your password, on the other hand you do not want to abandon your favorite mail client. The application that you develop in this project will help you to solve the problem.

1) Run your tunneling application at your home machine using the following command:

```
home:~$ java Tunnel --listen 12345 --redir 127.0.0.1:110
```

This will start your tunnel application which will listen on port 12345 at your home machine; it will redirect everything it receives to port 110 of localhost where your POP3 server is listening.

2) Run your tunneling application at your laptop using the following command:

```
laptop:~$ java Tunnel --listen 110 --redir home_server_IP:12345
```

This will start your tunnel application which will listen on port 110 at your laptop; it will redirect everything it receives to port 12345 of your home server machine.

3) Configure your POP3 client to use localhost instead of your home server, as you can guess port can remain the same, i.e. 110. Note that your traffic forwarding being unencrypted it can be intercepted by third parties.

Assignment: Submit both your code and a report showing a typical execution of your program (this part should be submitted under folder **PORT-FORWARDING**). If you don't have access to a POP3 mail server, you can demonstrate that your program is functioning correctly by forwarding other applications traffic or forwarding between different machines (e.g., your laptop to login.ccs.neu.edu / mail.ccs.neu.edu).

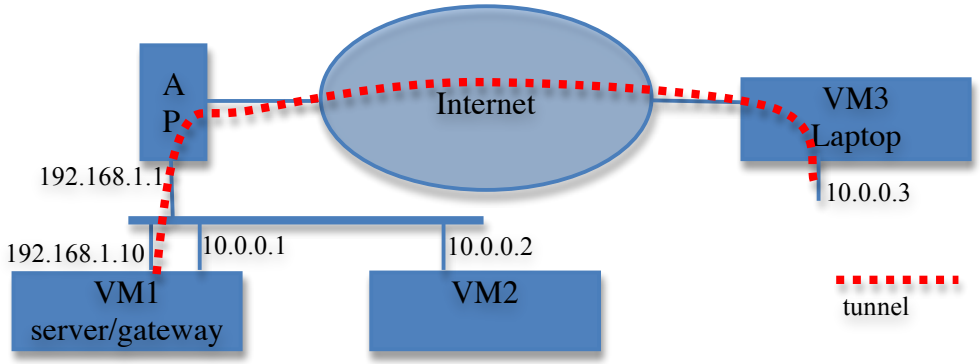
Bonus assignment (20 points):
The basic assignment is not a proper tunnel since it does not use any form of encapsulation or encryption, in the literature it's referred to as port forwarding as well. Encrypt all the forwarded traffic using AES encryption with a fixed shared secret. Use Java JCE library. This part should be submitted under folder **PORT-FORWARDING-SECURE**.

PART III (100 points): VPN (required for graduate students)

This part builds on the knowledge you gained in Part I. You have to write a VPN program that creates a tun interface and a UDP socket and has two modes of operation: (a) server mode (b) client mode. When run as a server it will read data from a UDP socket and write it to the raw socket interface (and vice versa), when run as a client, it will read data from the tun interface and write it to the UDP socket (and vice versa). Since you will need to deal with two file descriptors, you may find the select() system call helpful.

To demonstrate your system, you need three virtual machines (VM1, VM2, VM3).

1. VM1 and VM2 can run in your home network. The tunnel creates a virtual private network for subnet 10.0.0/24. VM1 should have two network interfaces: (1) one that allows a connection to the Internet e.g., 192.168.1.10 (this IP address is obtained from the DHCP server running on the AP), (2) a second network interface with manually set IP address 10.0.0.1 (can easily be added on VirtualBox -> Settings -> Network menu).
2. VM2 should have at least a network interface with manually set IP address 10.0.0.2 (can easily be added on VirtualBox -> Settings -> Network menu).
3. You program running on VM3 will create a virtual network interface (tun) with manually set IP address 10.0.0.3.



Assignment:
Next, you should run your program in server mode on VM1 and in client mode on host VM3. This will create tun0 interface in VM3. You need to do several things to get the whole system to work, for example, you need to use the **`route`** command on VM2 to specify that traffic to 10.0.0.3 should be routed through 10.0.0.1. Since, the tunnel will using UDP communication between VM1 and VM3, you should make sure that VM1 can receive packets from the external internet (e.g., using port forwarding on the AP).

Hints:

- You might need to enable IP forwarding (`echo 1 > /proc/sys/net/ipv4/ip_forward`)

- You might need to use **`iptables`** to queue (in iptables QUEUE) the packets forwarded from VM2 to VM3 throught VM1. You can use the following sample program to access the queued packets: http://www.ccs.neu.edu/home/noubir/Courses/CS4700-5700/F11/problems/PS6/iptable_queue.tar

Submit both your code, and a report showing a typical execution of your program (e.g., ping before and after running the vpn client/server).