Virtual Private Network (VPN) Lab

Final Submission Due: April 1

1 Overview

A Virtual Private Network (VPN) is used for creating a private scope of computer communications or providing a secure extension of a private network into an insecure network such as the Internet. VPN is a widely used security technology and can be built upon IPSec or Secure Socket Layer (SSL). These are two fundamentally different approaches for building VPNs. This lab focuses on the SSL-based VPNs, often referred to as SSL VPNs.

The learning objective of this lab is for students to master the network and security technologies underlying SSL VPNs. The design and implementation of SSL VPNs exemplify a number of security principles and technologies, including crypto, integrity, authentication, key management, key exchange, and Public-Key Infrastructure (PKI). To achieve this goal, students will implement a simple SSL VPN for `Lubuntu`.

2 Before You Begin

2.1 Remove Old Virtual Machines

Make sure you have a clean VirtualBox configuration before you begin this lab. To do that follow the following steps:

1. Start by making sure you have no running virtual machines on your mcXX machine. To do that run the following command:

   ```
   VBoxManage list runningvms
   ```

   If this command shows any running virtual machines stop them with the following command:

   ```
   VBoxManage controlvm VMNAME poweroff
   ```

   Replace VMNAME with the name of the virtual machine to stop.

2. Now make sure that there are no VirtualBox processes running:

   ```
   ps aux | grep -v grep | grep VBox | grep $USER
   ```

   If this command output any running VBox processes make sure to kill them.

3. Now that you do not have any running virtual machines, to ensure you have a clean configuration run the following command and make it doesn't output anything:

   ```
   VBoxManage list vms
   ```

   If the command outputs the existence of any virtual machines make sure you unregister and delete them with the following command:

```
VBoxManage unregistervm VMNAME --delete
```

Replace `VMNAME` with the name of the virtual machine to delete.

NOTE: this command will remove all contents of the virtual machine. Make sure to backup anything you want to save from the virtual machine before unregistering it.

2.2 Remove Old NAT Networks

Make sure you have removed all existing NAT networks before you begin the lab. To ensure you have a clean network configuration run the following command and make sure it doesn't output anything:

```
VBoxManage list natnets
```

If the command outputs the existence of any NAT network make sure you remove it with the following commands:

```
VBoxManage natnetwork stop --netname NETNAME
VBoxManage natnetwork remove --netname NETNAME
VBoxManage dhcpserver remove --netname NETNAME
```

Replace `NETNAME` with the name of the network you want to remove.

2.3 Stop VirtualBox Processes

At this point there shouldn't be any VirtualBox processes running, but to make sure you have a clean configuration run the following command to check for VirtualBox processes:

```
ps aux | grep $USER | grep VBox | grep -v grep
```

If any VirtualBox processes are listed kill them with the following command:

```
kill PID
```

Replace `PID` with the process identifier output from the `ps` command.

3 Lab Tasks

In this lab, you will implement a simple VPN for `Linux` called `MiniVPN`.

3.1 Task 1: Create a Host-to-Host Tunnel using TUN/TAP

The enabling technology for the TLS/SSL VPNs is TUN/TAP, which is now widely implemented in modern operating systems. TUN and TAP are virtual network kernel drivers; they implement network devices that are supported entirely in software. TAP (as in network tap) simulates an Ethernet device and it operates with layer-2 packets such as Ethernet frames; TUN (as in network TUNnel) simulates a network layer device and it operates with layer-3 packets such as IP packets. With TUN/TAP, you can create virtual network interfaces.

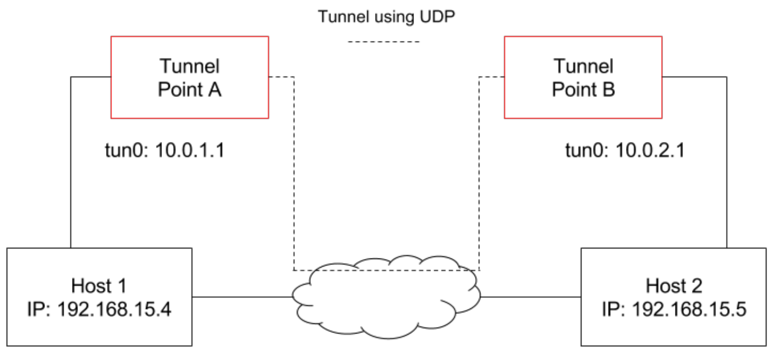

Figure 1: Host-to-Host Tunnel

A user-space program is usually attached to the TUN/TAP virtual network interface. Packets sent by an operating system via a TUN/TAP network interface are delivered to the user-space program. On the other hand, packets sent by the program via a TUN/TAP network interface are injected into the operating system network stack; to the operating system, it appears that the packets came from an external source through the virtual network interface.

When a program is attached to a TUN/TAP interface, the IP packets that the computer sends to this interface will be piped into the program; on the other hand, the IP packets that the program sends to the interface will be piped into the computer, as if they came from the outside through this virtual network interface. The program can use the standard `read()` and `write()` system calls to receive packets from or send packets to the virtual interface.

Davide Brini has written a tutorial article on how to use TUN/TAP to create a tunnel between two machines. The URL of the tutorial is:

`http://backreference.org/2010/03/26/tuntap-interface-tutorial`

The tutorial provides a program called `simpletun`, which connects two computers using the TUN tunneling technique. Read this tutorial and familiarize yourself with the setup instructions.

A version of `simpletun` that has been modified for this lab has been provided on the course web page(Piazza/resources). You can download the file into your virtual machine.

Once downloaded into your virtual machine, compile it using the following command:

```
$ gcc -o simpletun simpletun.c
```

Creating Host-to-Host Tunnel. The following procedure shows how to create a host-to-host tunnel using the `simpletun` program. The `simpletun` program can run as both a client and a server. When it is running with the `-s` flag, it acts as a server; when it is running with the `-c` flag, it acts as a client.

1. **Launch two virtual machines.** For this task, you will launch two virtual machines on the same host machine. The following assumes the IP addresses for the two machines are `192.168.15.4`, and

`192.168.15.5`, respectively. See the configuration in Figure 1. Make sure you have your virtual machines set up correctly on the network and have the port forwarding set.

2. **Tunnel Point A:** use Tunnel Point A as the server side of the tunnel. Point A is on machine `192.168.15.4` (see Figure 1). It should be noted that the client/server concept is only meaningful when establishing the connection between the two ends. Once the tunnel is established, there is no difference between client and server; they are simply two ends of a tunnel. Run the following command (the `-d` flag asks the program to print out the debugging information):

   ```
   On Machine 192.168.15.4:
   # sudo ./simpletun -i tun0 -s -d
   ```

   After the above step, your virtual machine will now have multiple network interfaces, one is its own Ethernet card interface, and the other is the virtual network interface called `tun0`. This new interface is not yet configured, so you need to configure it by assigning an IP address.

   It should be noted that the above command will block and wait for connections, so, we need to find another window to configure the `tun0` interface. To do this open up another SSH session to the virtual machine. Run the following commands (the first command will assign an IP address to the interface `"tun0"`, and the second command will bring up the interface):

   ```
   On Machine 192.168.15.4:
   # sudo ip addr add 10.0.1.1/24 dev tun0
   # sudo ifconfig tun0 up
   ```

3. **Tunnel Point B:** you use Tunnel Point B as the client side of the tunnel. Point B is on machine `192.168.15.5` (see Figure 1). You run the following command on this machine (The first command will connect to the server program running on `192.168.15.4`, which is the machine that runs the Tunnel Point A. This command will block as well, so you need to find another window for the second and the third commands):

   ```
   On Machine 192.168.15.5:
   # sudo ./simpletun -i tun0 -c 192.168.15.4 -d
   # sudo ip addr add 10.0.2.1/24 dev tun0
   # sudo ifconfig tun0 up
   ```

4. **Routing Path:** After the above two steps, the tunnel will be established. Before you can use the tunnel, you need to set up the routing path on both machines to direct the intended outgoing traffic through the tunnel. The following routing table entry directs all the packets to the `10.0.2.0/24` network (`10.0.1.0/24` network for the second command) through the interface `tun0`, from where the packet will be hauled through the tunnel.

   ```
   On Machine 192.168.15.4:
   # sudo route add -net 10.0.2.0 netmask 255.255.255.0 dev tun0

   On Machine 192.168.15.5:
   # sudo route add -net 10.0.1.0 netmask 255.255.255.0 dev tun0
   ```

5. **Using the Tunnel:** Now you can access `10.0.2.1` from `192.168.15.4` (and similarly access `10.0.1.1` from `192.168.15.5`). You can test the tunnel using `ping` and `ssh`:

   ```
   On Machine 192.168.15.4:
   ```

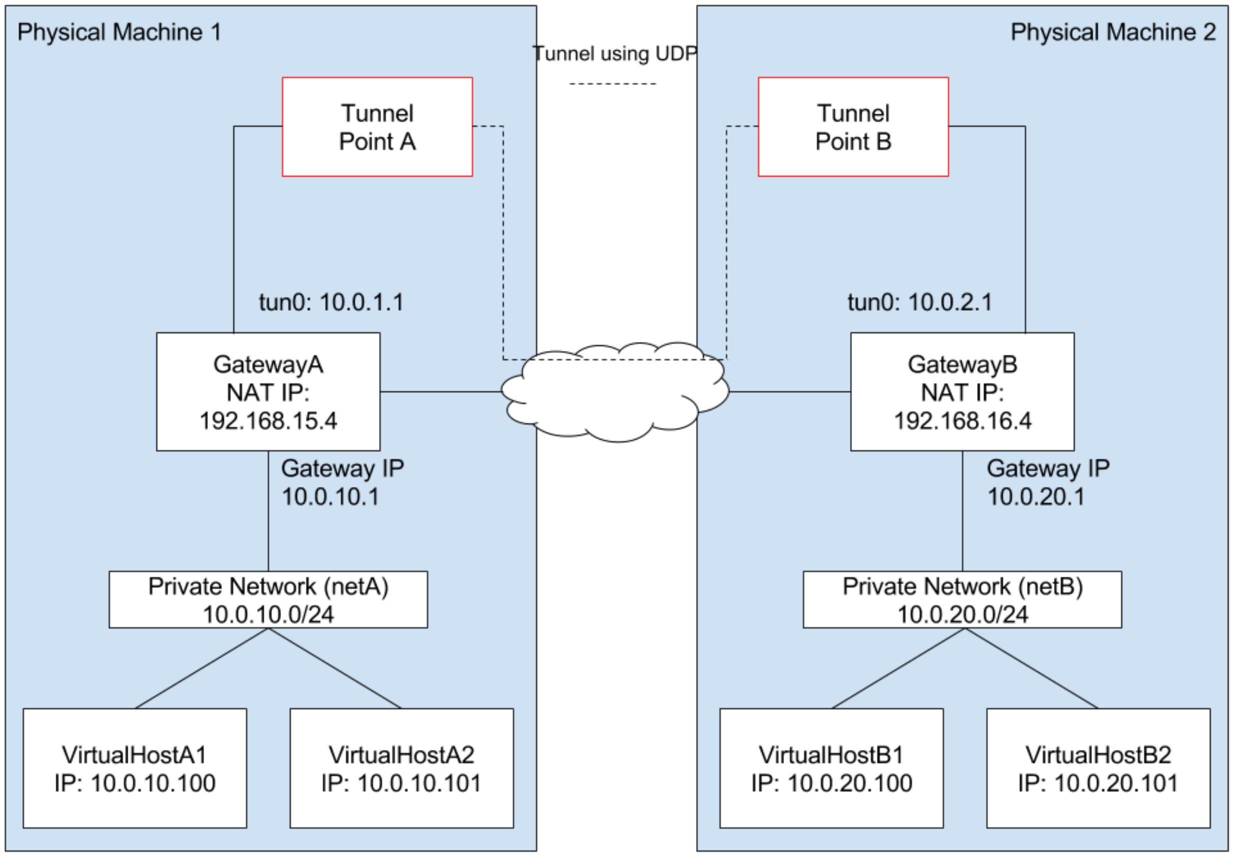
Physical Machine 1
Tunnel using UDP
Physical Machine 2
Tunnel Point A
Tunnel Point B
tun0: 10.0.1.1
tun0: 10.0.2.1
GatewayA NAT IP: 192.168.15.4
GatewayB NAT IP: 192.168.16.4
Gateway IP 10.0.10.1
Gateway IP 10.0.20.1
Private Network (netA) 10.0.10.0/24
Private Network (netB) 10.0.20.0/24
VirtualHostA1 IP: 10.0.10.100
VirtualHostA2 IP: 10.0.10.101
VirtualHostB1 IP: 10.0.20.100
VirtualHostB2 IP: 10.0.20.101

Figure 2: Gateway to Gateway Tunnel

```
$ ping 10.0.2.1
$ ssh 10.0.2.1

On Machine 192.168.15.5:
$ ping 10.0.1.1
$ ssh 10.0.1.1
```

UDP Tunnel: The connection used in the `simpletun` program is a TCP connection, but our VPN tunnel needs to use UDP. Therefore you need to modify `simpletun` and turn the TCP tunnel into a UDP tunnel. Think about why it is better to use UDP in the tunnel, instead of TCP write your answer in your lab report.

3.2 Task 2: Create a Private Network using a Gateway

In this task, you need to create a private network between a computer and a gateway. To demonstrate this, you need two physical computers. On one computer, you run a virtual machine within to set up the gateway and the private network. You then use a virtual machine in the other computer to communicate to the host(s) on the private network. The physical computers should be two from the mcXX.cs.purdue.edu pool. See the `Lab_Setup.pdf` file for additional information on the "mc" machines.

Two new virtual machine image (OVA) files have been provided on the course web page for you to use as the base for your gateways (GatewayA and GatewayB). They have been preconfigured with all the necessary network interfaces, `iptables` rules, and DHCP configurations needed for the gateway to properly route packets for your client machines given the IP configuration used in Figure 2. Follow the following steps to import and set up your gateways and client machines.

1. **Create GatewayA and the Client Virtual Machine:** To setup the virtual machine for GatewayA, use the following steps:

   (a) **Choose an MC machine:** Using the information in `Lab_Setup.pdf` choose an mcXX.cs.purdue.edu machine to use for the 'A' side of your configuration.

   (b) **Import the Gateway Virtual Machine:**

   Import the virtual machine using the import command on `VBoxManage`. NOTE: Because GatewayA has 2 network interface cards (NICs) so you must specify the disk location using `--unit 7` rather than `--unit 6`. Make sure you follow the parameters for the command as they appear below:

   ```
   VBoxManage import \
    /homes/cs528/vms/cs528_vm_gwa.ova \
    --vsys 0 --vmname GatewayA --unit 7 \
    --disk /scratch/<USERID>/lab3/GatewayA/GatewayA.vmdk
   ```

   NOTE: Replace `<USERID>` with your user name. Make sure you use `cs528_vm_gwa.ova` as the image file.

   (c) **Setup the NAT network and DHCP server:** To set up the NAT network for Gateway A run:

   ```
   VBoxManage natnetwork add --netname <USERID>netA \
     --network "192.168.15.0/24" --enable --dhcp on
   ```

   ```
   VBoxManage natnetwork start --netname <USERID>netA
   ```

   NOTE: Replace `<USERID>` with your user name in order to make the network unique. For example if your user id is `jdoe` your network name would be `jdoenetA`.

   (d) **Modify the NIC configuration for NAT and internal network:** Modify the GatewayA virtual machine to use the network names specific to your configuration.

   ```
   VBoxManage modifyvm GatewayA --nat-network1 <USERID>netA
   ```

   Then follow the steps in lab setup to set port forwarding rules for gateway so that you can ssh into it.

   ```
   VBoxManage modifyvm GatewayA --nic2 intnet --intnet2 <USERID>inetA
   ```

   NOTE: As with NAT networks, internal network names must also be unique. Make sure you use the correct internal network name by replacing `<USERID>` with your user name. In this case use your USERID followed by inetA (for internal network A)

(e) **Finalize the Gateway Configuration:** Start your GatewayA virtual machine. All other settings for IP addresses and forwarding rules have been set for GatewayA in the provided virtual machine image. No more configuration should be needed for GatewayA.

(f) **Create a Client on the Private Network:** Import a virtual machine to use as the client on the private network using the base virtual machine image for the course (not the gateway image). This virtual machine will need to be modified to be on the internal network.

Import the virtual machines using the import command on VBoxManage. Refer to `Lab_Setup.pdf` for more information.

```
VBoxManage import \
 CS528_2018.ova \
 --vsys 0 --vmname VirtualHostA1 --unit 6 \
 --disk /scratch/<USERID>/lab3/VirtualHostA1/VirtualHostA1.vmdk
```

NOTE: Replace `<USERID>` with your user name. Also remember to use vm image provided on Piazza here.

To set the network to the internal network use the following command:

```
VBoxManage modifyvm VirtualHostA1 --nic1 intnet --intnet1 <USERID>inetA
```

NOTE: Make sure you use the correct internal network name.

(g) **Connect to the client virtual machine:** The last thing to do is verify that you can connect to the client virtual machine (VirtualHostA1). Don't forget to start the client virtual machines before trying to connect. Since this machine does not have a connection to the NAT network, in order to connect to it you must first connect to the gateway:

```
ssh cs528user@mcXX.cs.purdue.edu -p <PORT>
```

Replace `<PORT>` with the port number you assigned for the port forwarding rules.

From the gateway attempt to ssh to the client virtual machine. VirtualHostA1 should have IP address 10.0.10.100.

To connect to VirtualHostA1 from GatewayA use the command:

```
ssh cs528user@10.0.10.100
```

2. **Exploring the Network Configuration on the Gateway:** Now that you have successfully imported GatewayA, VirtualHostA1, and configured the network names, the following explains the details of how the gateway is configured. Make sure you understand how the `iptables` and routing information works since you will need to expand on this when you create your VPN program.

(a) **The Gateway Virtual Machine:** The basis for the gateway is the virtual machine image provided for previous labs.

(b) **Using `VirtualBox` Internal Network:** By definition a gateway connects one or more networks together. For the configuration in Figure 2 the gateway connects (bridges) a private internal network with the NAT network to allow clients to access the NAT network while maintaining private IP addresses. For the private network, a `VirtualBox` internal network is used. The gateway virtual machine already had a virtual network interface card (NIC) of type NAT network that you used to ssh into the virtual machine (as well as connect to the outside world). An additional NIC was added to the gateway to receive network traffic from the private network.

(c) **Setting a Static IP for the Gateway:** The next step in setting up the gateway was to make sure that it has a static IP address on the private network since clients will need to know which IP address to send their packets. As such, it is important that the gateway's IP never changes.

To set a static IP address the following changes where made to `/etc/network/interfaces` on the gateway virtual machine. From the command line GatewayA display the contents of the file using `cat`:

```
cat /etc/network/interfaces
```

You should see something that looks like the following:

```
cs528user@cs528vm:/etc/network$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto enp0s3
iface enp0s3 inet dhcp

auto enp0s8
iface enp0s8 inet static
address 10.0.10.1
netmask 255.255.255.0
gateway 192.168.15.1
```

Here `enp0s3` is used to connect to the external (NAT) network as in the previous lab so it is set to use DHCP. The network interface `enp0s8` is connected to the private network and is given the static IP 10.0.10.1. Notice that the gateway for this interface is the same gateway as the NAT network. This will ensure that all traffic received on the private (internal) network is sent to the NAT network allowing client virtual machines on the private network to reach the "outside world".

(d) **Setting IP Forwarding on the Gateway:** Unless specifically configured, a computer will only act as a host, not as a gateway. In `Linux`, IP forwarding needs to be explicitly enabled. IP forwarding was enabled using the following command:

```
sudo sysctl net.ipv4.ip_forward=1
```

(e) **Setting the IP Tables Rules for Forwarding:** By default the gateway does not know which interfaces to forward incoming packets on. To tell the gateway how to forward packets, the `iptables` interface is used. To view the saved `iptables` configuration use the following command:

```
sudo iptables-save
```

You should see an output that looks similar to the following:

```
# Generated by iptables-save v1.4.21 on Tue May 10 11:25:02 2016
*filter
```

```
:INPUT ACCEPT [210:27264]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [239:57155]
-A FORWARD -i enp0s8 -j ACCEPT
COMMIT
# Completed on Tue May 10 11:25:02 2016
# Generated by iptables-save v1.4.21 on Tue May 10 11:25:02 2016
*nat
:PREROUTING ACCEPT [1:576]
:INPUT ACCEPT [1:576]
:OUTPUT ACCEPT [4:587]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o enp0s3 -j MASQUERADE
COMMIT
# Completed on Tue May 10 11:25:02 2016
```

The set of rules after `*filter` tell the `iptables` interface how to filter incoming data. Here it is set to accept all packets routed through interface `enp0s8` (the interface to the private network).

The set of rules after `*nat` tell the `iptables` interface how to handle the packets for a new connection. Here the rule states to masquerade data sent out of `enp0s3`. IP masquerading is one way in which `Linux` performs network address translation. Setting this rule essentially turns GatewayA into a NAT router.

(f) **Creating a DHCP Server for the Internal Network:** The gateway also acts as the DHCP server for the clients. On the Linux virtual machine image provided this is done using the udhcpd program. Setting up the udhcpd program involves editing the `/etc/udhcpd.conf` file. Display the contents of the file using `head`, only the first 20 lines of the file are relevant to this lab:

```
head -n 20 /etc/udhcpd.conf
```

You should see something that looks like the following:

```
# Sample udhcpd configuration file (/etc/udhcpd.conf)

# The start and end of the IP lease block

start           10.0.10.100     #default: 192.168.0.20
end             10.0.10.200     #default: 192.168.0.254


# The interface that udhcpd will use

interface       eth15           #default: eth0


# The gateway IP that will be given to clients
```

```
opt     router  10.0.10.1
```

Here the values are fairly self explanatory. The `start` and `end` values tell the DHCP server the start and end range of IP addresses to give out to clients. The `interface` value is the network interface to use for receiveing and responding to DHCP requests. Finally, the `opt router` value sets what gateway to tell clients to use.

Now that you understand from a high level how the gateway was created and configured, you will need to import and configure GatewayB its the client machine.

3. **Create GatewayB and the Client Virtual Machine:** To setup the virtual machine for GatewayB, use the following steps:

   (a) **Choose an MC machine:** Using the information in `Lab_Setup.pdf` choose an mcXX.cs.purdue.edu machine to use for the 'B' side of your configuration.

   (b) **Import the Gateway Virtual Machine:**

   Import the virtual machine using the import command on `VBoxManage`. NOTE: Because GatewayB has 2 network interface cards (NICs) so you must specify the disk location using `--unit 7` rather than `--unit 6`. Make sure you follow the parameters for the command as they appear below:

   ```
   VBoxManage import \
    /homes/cs528/vms/cs528_vm_gwb.ova \
    --vsys 0 --vmname GatewayB --unit 7 \
    --disk /scratch/<USERID>/lab3/GatewayB/GatewayB.vmdk
   ```

   NOTE: Replace `<USERID>` with your user name. Make sure you use `cs528_vm_gwb.ova` as the image file.

   (c) **Setup the NAT network and DHCP server:** To set up the NAT network for Gateway A run:

   ```
   VBoxManage natnetwork add --netname <USERID>netB \
     --network "192.168.16.0/24" --enable --dhcp on
   ```

   ```
   VBoxManage natnetwork start --netname <USERID>netB
   ```

   NOTE: Replace `<USERID>` with your user name in order to make the network unique. For example if your user id is `jdoe` your network name would be `jdoenetB`.

   (d) **Modify the NIC configuration for NAT and internal network:** Modify the GatewayB virtual machine to use the network names specific to your configuration.

   ```
   VBoxManage modifyvm GatewayB --nat-network1 <USERID>netB
   ```

   Then follow the steps in lab setup to set port forwarding rules for gateway so that you can ssh into it.

   ```
   VBoxManage modifyvm GatewayB --nic2 intnet --intnet2 <USERID>inetB
   ```

   NOTE: As with NAT networks, internal network names must also be unique. Make sure you use the correct internal network name by replacing `<USERID>` with your user name. In this case use your USERID followed by inetB (for internal network B)

(e) **Finalize the Gateway Configuration:** Start your GatewayB virtual machine. All other settings for IP addresses and forwarding rules have been set for GatewayB in the provided virtual machine image. No more configuration should be needed for GatewayB.

(f) **Create a Client on the Private Network:** Import a virtual machine to use as the client on the private network using the base virtual machine image for the course (not the gateway image). This virtual machine will need to be modified to be on the internal network.

Import the virtual machines using the import command on VBoxManage. Refer to `Lab_Setup.pdf` for more information.

```
VBoxManage import \
 CS528_2018.ova \
 --vsys 0 --vmname VirtualHostB1 --unit 6 \
 --disk /scratch/<USERID>/lab3/VirtualHostB1/VirtualHostB1.vmdk
```

NOTE: Replace `<USERID>` with your user name. Also remember to use vm image provided on Piazza here.

To set the network to the internal network use the following command:

```
VBoxManage modifyvm VirtualHostB1 --nic1 intnet --intnet1 <USERID>inetB
```

NOTE: Make sure you use the correct internal network name.

(g) **Connect to the client virtual machine:** The last thing to do is verify that you can connect to the client virtual machine (VirtualHostB1). Don't forget to start the client virtual machines before trying to connect. Since this machine does not have a connection to the NAT network, in order to connect to it you must first connect to the gateway:

```
ssh cs528user@mcXX.cs.purdue.edu -p <PORT>
```

Replace `<PORT>` with the port number you assigned for the port forwarding rules.

From the gateway attempt to ssh to the client virtual machine. VirtualHostB1 should have IP address 10.0.20.100.

To connect to VirtualHostB1 from GatewayB use the command:

```
ssh cs528user@10.0.20.100
```

Once your client virtual machines are setup, running, and you can ssh to them. You can test to see if you set up your gateway correctly by trying to ping an external IP address from the client virtual machine. For example, from VirtualHostB1 (10.0.20.100) try to ping www.google.com. You should receive a response. However, you should not be able to ping VirtualHostA1 (10.0.10.100) from VirtualHostB1 (10.0.20.100). That part comes next.

3.3 Task 3 Create a Gateway-to-Gateway Tunnel

In this task, you need to go a step further to establish a tunnel between two gateways of different private networks. With this tunnel, any host from one private network can communicate with the hosts on the other private network using the tunnel. The setup for such a gateway-to-gateway tunnel is depicted in Figure 2. The instructions here describe how to set up a tunnel using the `simpletun` program from the previous task which creates a TCP tunnel. NOTE: that for your VPN program you will need to use UDP as the transport layer protocol.

1. **Create Port Forwarding Rules for the Tunnel:** Unlike the previous task, the tunnel here is between two physical machines and as such needs to bridge the external network. Since your virtual machines are using NAT networking, the network configuration needs to know how to send incoming tunnel packets to the gateway virtual machine. To do that you'll need to set up port forwarding rules. By default the `simpletun` program uses port 55555 for traffic, however, you will need to change this to use one of the ports from the range previously assigned to you.

   Set up the port forwarding rule for the 'A' side of your configuration with the following:

   ```
   VBoxManage natnetwork modify --netname <USERID>netA \
       --port-forward-4 "vpn:tcp:[]:<PORT>:[192.168.15.4]:<PORT>"
   ```

   NOTE: Make sure you are running on the physical machine that runs the 'A' side of your network. Replace `<USERID>` and `<PORT>` with your user name and port number respectively. Also make sure you use the correct network name. In this case it is the NAT network, not the internal network.

   Set up the port forwarding rule for the 'B' side of your configuration with the following:

   ```
   VBoxManage natnetwork modify --netname <USERID>netB \
       --port-forward-4 "vpn:tcp:[]:<PORT>:[192.168.16.4]:<PORT>"
   ```

   NOTE: Again, make sure you are running the command on the physical machine that runs the 'B' side of your network. Replace `<USERID>` and `<PORT>` with your user name and port number respectively. Also make sure you use the correct network name. In this case it is the NAT network, not the internal network.

2. **Starting the Tunnel Point A:** As with the previous task use the `simpletun` program to start a tunnel on the 'A' size of your network:

   ```
   On GatewayA (192.168.15.4):
   # sudo ./simpletun -i tun0 -p <PORT> -s -d
   ```

   NOTE: Replace `<PORT>` with the port number you chose when you created your forwarding rule.

   As before this command will block so you will need to open a new session to GatewayA and run the following to set the routing rule.

   ```
   On GatewayA (192.168.15.4):
   # sudo ip addr add 10.0.1.1/24 dev tun0
   # sudo ifconfig tun0 up
   # sudo route add -net 10.0.20.0 netmask 255.255.255.0 dev tun0
   ```

   As before the first command sets the IP address for GatewayA on the tunnel, the second command starts the tunnel interface (tun0), and the third command adds a routing table entry to route all traffic destined for the 10.0.20.0 (GatewayB's private network) down the tunnel.

3. **Starting the Tunnel Point B:** As with the previous task use the `simpletun` program to start a tunnel on the 'B' size of your network:

   ```
   On GatewayB (192.168.16.4):
   # sudo ./simpletun -i tun0 -p <PORT> -c <GATEWAYAIP> -d
   ```

NOTE: Replace `<PORT>` with the port number you chose when you created your forwarding rule. Also replace `<GATEWAYAIP>` with the external IP address of the mcXX machine that is running GatewayA. Since GatewayA is on a NAT network, GatewayB cannot connect to it directly using its IP address internal to the NAT network since it can't "see" that IP on it's network. This is why you had to set the port forwarding rule in the previous step. Instead use the external IP address of the mcXX machine that is running GatewayA and the port forwarding rule will handle the rest.

As before this command will block so you will need to open a new session to GatewayB and run the following to set the routing rule.

```
On GatewayB (192.168.16.4):
# sudo ip addr add 10.0.2.1/24 dev tun0
# sudo ifconfig tun0 up
# sudo route add -net 10.0.10.0 netmask 255.255.255.0 dev tun0
```

As before the first command sets the IP address for GatewayB on the tunnel, the second command starts the tunnel interface (tun0), and the third command adds a routing table entry to route all traffic destined for the 10.0.10.0 (GatewayA's private network) down the tunnel.

At this point you should now have a tunnel configured as pictured in Figure 2 with one exception. The `simpletun` program creates a TCP tunnel and in Figure 2 it shows a UDP tunnel.

You should now be able to connect from one host on the private network to the other. Try that with `ssh`.

1. Connect to VirtualHostA1 - Remember you must first connect to GatewayA and then ssh to VirtualHostA1
2. Open an ssh connection to VirtualHostB1 from VirtualHostA1:

   ```
   ssh cs528user@10.0.20.100
   ```

If the connection works, then you know you've configured your tunnel correctly. The tunnel is not secure since `simpletun` sends network traffic down the tunnel without encryption or authentication. The next task adds security to the tunnel.

3.4 Task 4: Create a Virtual Private Network (VPN)

At this point, you have learned how to create a network tunnel. Now, if you can secure this tunnel, you will essentially get a VPN. This is what you are going to achieve in this task. To secure this tunnel, you need to achieve two goals, confidentiality and integrity. The confidentiality is achieved using encryption, i.e., the contents that go through the tunnel is encrypted. Modern VPN software usually supports a number of different encryption algorithms. For the `MiniVPN` in this lab, you only need to support a single encryption algorithm.

The integrity goal ensures that nobody can tamper with the traffic in the tunnel or launch a replay attack. Integrity can be achieved using various methods. In this lab, you only need to support the Message Authentication Code (MAC) method. The AES encryption algorithm and the HMAC-SHA256 algorithm are both implemented in the `OpenSSL` library. There are plenty of online documents explaining how to use the `OpenSSL`'s crypto libraries.

Both encryption and MAC need a secret key. Although the keys can be different for encryption and MAC, for the sake of simplicity, assume that the same key is used. This key has to be agreed upon by both sides of the VPN. For this task, assume that the key is already provided. Agreeing upon the key will be implemented in the next task.

For encryption, the client and the server also need to agree upon an Initial Vector (IV). For security purpose, you should not hard-code the IV in your code. The IV should be randomly generated for each VPN tunnel. Agreeing upon the IV will also be implemented in the next task.

NOTE: Your `MiniVPN` must use a UDP connection for data transmission. A TCP connection can be used for starting the VPN connections and key exchange.

NOTE: The `MiniVPN` program will be running on your gateway virtual machines (GatewayA and GatewayB). To ensure that data packets (sent via UDP) are correctly sent to your gateway virtual machines, you will need to add another port forwarding rule for UDP packets.

3.5 Task 5: Authentication and Key Exchange

Before a VPN is established, the VPN client must authenticate the VPN server, making sure that the server is not a fraudulent one. On the other hand, the VPN server must authenticate the client (i.e. user), making sure that the user has the permission to create such a VPN tunnel. After the authentication is done, the client and the server will agree upon a session key for the VPN tunnel. This session key is only known to the client and the server. The process of deriving this session key is called key exchange.

Step 1: Authenticating VPN Server A typical way to authenticate the server is to use public-key certificates. The VPN server needs to first get a public-key certificate from a Certificate Authority (CA), such as Verisign. When the client makes the connection to the VPN server, the server will use the certificate to prove it is the intended server. The `HTTPS` protocol in the Web uses a similar way to authenticate web servers, ensuring that you are talking to an intended web server, not a fake one.

In this lab, `MiniVPN` should use such a method to authenticate the VPN server. You can implement an authentication protocol (such as SSL) from scratch, using the crypto libraries in `OpenSSL` to verify certificates or you can use the `OpenSSL`'s SSL functions to directly make an SSL connection between the client and the server. When using `OpenSSL`'s functions the verification of certificates will be automatically carried out by the SSL functions. Guidelines on making such a connection can be found in the guidelines section.

Before the authentication, both client and server need to set up their public/private keys and certificates properly, so they can verify each other's public-key certificate.

Step 2: Authenticating VPN Client (i.e. User) There are two common ways to authenticate the user. One is using the public-key certificates, namely, users need to get their own public-key certificates. When they try to create a VPN with the server, they need to send their certificates to the server, which will verify whether they have permissions for such a VPN. `OpenSSL`'s SSL functions also support this option if you specify that the client authentication is required.

Since users usually do not have their public-key certificates, a more common way to authenticate users is to use the traditional user name and password approach. Namely, after the client and the server have established a secure TCP connection between themselves, the server can ask the client to type the user name and the password, and the server then decides whether to allow the user to proceed depending on whether the user name and password matches with the information in the server's user database.

In this lab, you can pick either of them to implement.

Step 3: Key Exchange If you use `OpenSSL`'s SSL functions, after the authentication, a secure channel will be automatically established (by the `OpenSSL` functions). However, we are not going to use this TCP connection for our tunnel, because our VPN tunnel uses UDP. Therefore, we will treat this TCP connection as the control channel between the client and the server. Over this control channel, the client and the server will agree upon a session key for the data channel (i.e. the VPN tunnel). They can also use the control channel for other functionalities, such as updating the session key, exchanging the Initial Vector (IV), terminating the VPN tunnel, etc.

At the end of this step, you should be able to use the session key to secure the tunnel. In other words, you should be able to test Task 4 and Task 5 together.

Step 4: Securing the Tunnel Using the session key, the client and the server can secure the VPN tunnel between them. Two objectives need to be accomplished. First, the data going through the tunnel must be encrypted, so no eavesdropper can learn the data in the tunnel. Second, the integrity of the data in the tunnel must be preserved. If anybody has tampered with the data, the receiver can detect that, and can thus discard the data. These two objectives can be achieved using the symmetric-key encryption and one-way hash algorithms. An actual VPN product is able to support a variety of these algorithms. In this lab, you only need to support one encryption algorithm (with one specific encryption mode) and one one-way hash algorithm. The choice of which algorithm to use is up to you. Please make sure you document which algorithm you use and discuss why you chose to use it (giving advantages and potential disadvantages) in your analysis report.

NOTE: As part of securing the channel, not only will you need to handle confidentiality (via encryption) and integrity (via HMAC), you will need to be able to protect against man-in-the-middle attacks. Consider a malicious user in the middle between each end of your VPN. What security flaws might they be able to exploit and how do you prevent against them, for example listening to key exchange and authentication, performing replay attacks, etc. Your VPN must be able to handle these types of attacks. Make sure you discuss, in your analysis report, what attacks your VPN prevents and what you did to prevent them.

4 Guidelines

4.1 Create certificates

In order to use `OpenSSL` to create certificates, you have to have a configuration file. The configuration file usually has an extension `.cnf`. It is used by three `OpenSSL` commands: `ca`, `req` and `x509`. The manual page of it can be found using a Google Search. You can also get a copy of the configuration file from `/usr/lib/ssl/openssl.cnf`. After copying this file into your current directly, you need to create several sub-directories as specified in the configuration file (look at the `[CA_default]` section):

```
dir             = ./demoCA          # Where everything is kept
```

```
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
new_certs_dir   = $dir/newcerts     # default place for new certs.

database        = $dir/index.txt    # database index file.
serial          = $dir/serial       # The current serial number
```

For the `index.txt` file, simply create an empty file. For the `serial` file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file `openssl.cnf`, you can create certificates for the three parties involved, the Certificate Authority (CA), the server, and the client.

Certificate Authority (CA). For this lab, you are allowed to create your own CA, and then you can use this CA to issue certificates for servers and users. You will create a self-signed certificate for the CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign another certificate. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two files: `ca.key` and `ca.crt`. The file `ca.key` contains the CA's private key, while `ca.crt` contains the public-key certificate.

Server. Now that you have your own trusted CA, you can now ask the CA to issue a public-key certificate for the server. First, you need to create a public/private key pair for the server. The server should run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file `server.key`:

```
$ openssl genrsa -des3 -out server.key 1024
```

Once you have the key file, you can generates a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity).

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

Client. The client can follow the same steps to generate an RSA key pair and a certificate signing request:

```
$ openssl genrsa -des3 -out client.key 1024
$ openssl req -new -key client.key -out client.csr -config openssl.cnf
```

Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, you will use your own trusted CA to generate certificates:

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
             -config openssl.cnf
$ openssl ca -in client.csr -out client.crt -cert ca.crt -keyfile ca.key \
             -config openssl.cnf
```

If OpenSSL refuses to generate certificates, it is very likely that the names in your requests do not match with those of the CA. The matching rules are specified in the configuration file (look at the [policy_match] section). You can change the names of your requests to comply with the policy, or you can change the policy. The configuration file also includes another policy (called policy_anything), which is less restrictive. You can choose that policy by changing the following line:

```
"policy = policy_match"
```

To the line:

```
"policy = policy_anything"
```

4.2 Create a secure TCP connection using OpenSSL

In this lab, students need to know how to use OpenSSL APIs to establish a secure TCP connection. The following tutorial might be useful to you for this lab.

- http://www.ibm.com/developerworks/linux/library/l-openssl.html

Also, in the home directory on the gateway image is a file demo_openssl.tar.gz which contains a sample client and server application using OpenSSL. It includes how to get the peer's certificate, how to verify the certificate, how to check the private key for a certificate, etc.

You should be able to run the samples using the following:

- Untar the package

  ```
  tar xzf demo_openssl.tar.gz
  ```

- Run "make", and then you should be able to get the programs compiled.

4.3 An example: using telnet in our VPN

The following steps and figures might also be helpful to fully understand how packets from an application flow to its destination through the MiniVPN when users run telnet 10.0.20.100 from a host machine, which is VirtualHostA of a host-to-gateway VPN. The other end of the VPN is on a gateway, which is connected to the 10.0.20.0/24 network, where the telnet server 10.0.20.100 resides.

Figure 3 shows how a packet flow from the telnet client to the server.

1. The data of the packet starts from the telnet program.
2. The kernel will construct an IP packet, with the destination IP address being 10.0.20.100.
3. The kernel then sends the packet to the gateway using the default route.
4. The gateway receives the packet and the kernel there needs to decide which network interface the packet should be routed through: enp0s3 or tun0. Recall that enp0s3 connects the clients to the

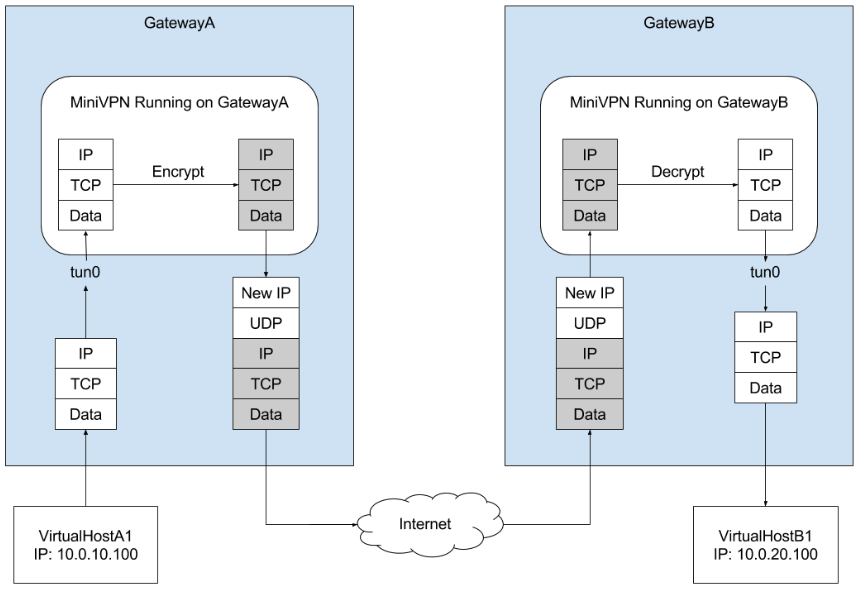

Figure 3: Telnet packet traveling from client to server through VPN tunnel

"outside world" using NAT and `tun0` is the tunnel interface for the VPN. You need to set up your routing table correctly for the kernel to pick `tun0`. Once the decision is made, the kernel will set the source IP address of the packet using the IP address of the network interface, which is `10.0.2.1`.

5. The packet will reach your VPN program (Point A) through the virtual interface `tun0`, then it will be encrypted, and then be sent back to the kernel through a UDP port (not through the `tun0` interface). This is because our VPN program use the UDP as our tunnel
6. The kernel will treat the encrypted IP packet as UDP data, construct a new IP packet, and put the entire encrypted IP packet as its UDP payload. The new IP's destination address will be the other end of the tunnel. This will ultimately be the external IP address of the target physical machine.
7. The packet will now flow through the Internet, with the original `telnet` packet being entirely encrypted, and carried in the payload of the packet. This is why it is called a *tunnel*.
8. The packet will reach GatewayB's physical machine through its outside interface and using a port forwarding rule be sent to GatewayB.
9. The kernel will give the UDP payload (i.e. the encrypted IP packet) to the VPN program running on GatewayB, which is waiting for UDP data. This is through the UDP port.
10. The VPN program will decrypt the payload, and then feed the decrypted payload, which is the original `telnet` packet, back to the kernel through the virtual network interface `tun0`.
11. Since it comes through a network interface, the kernel will treat it as an IP packet (it is indeed an IP packet), look at its destination IP address, and decide where to route it. Remember, the destination IP address of this packet is `10.0.20.100`. If your routing table is set up correctly, the packet should be routed through `enp0s8`, because this is the interface that connects to the `10.0.20.0/24` network.
12. The `telnet` packet will now be delivered to its final destination `10.0.20.100`.

4.4 Miscellaneous notes

Our client (or server) program is going to listen to both TCP and UDP ports, these two activities may block each other. It is better if you can `fork()` two processes, one dealing with the TCP connection, and the other dealing with UDP. These processes need to be able to communicate with each other. You can use the Inter-process call (IPC) mechanisms for the communication. The simplest IPC mechanism is unnamed pipe, which should be sufficient for us. You can learn IPC from online documents.

5 Submission

5.1 What to Submit

Your submission directory should contain:

- All source files written by you to perform the lab tasks.
- A README text file describing how to compile and execute your source code.
- A detailed report containing an explanation of the observations. Name this report *Analysis-lab3.pdf*. Be sure to put your name in your report.

5.2 How to submit

Submit every time you have a stable version of any part of the functionality. Make sure to submit early to make sure you do not miss the deadline due to any last minute congestion.

1. Login or ssh to an mcXX machine, e.g., mc01.cs.purdue.edu
2. In the parent directory of your submission directory, type the command:

   turnin -c cs528 -p lab3 <submission-dir-name>

   where <submission-dir-name> is the name of the directory containing your files to be submitted. For example, if your program is in a directory `/homes/abc/assignment/src`, make sure you cd to the directory `/homes/abc/assignment` and type:

   turnin -c cs528 -p lab3 src

3. If you wish to, you can verify your submission by typing the following command:

   turnin -v -c cs528 -p lab3

   Do not forget the **-v** above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission).

   Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

   We will check that the submission time-stamp is **before the due date**; we will not accept your submission if its time-stamp is after the due date, even by 3 minutes. Do NOT submit after 11:59 PM Eastern Time.

NOTE: Do not submit your virtual machine disk image. Copy the files for submission off of your virtual machine and into a directory on the mcXX machine.

5.3 Demonstration

You will also need to demonstrate your `MiniVPN` software to the teaching assistant. Please sign up for a demonstration time slot with the teaching assistant. Please take the following into consideration when you prepare for your demonstration:

- The total time of the demo will be 15 minutes, no additional time will be given, so practice your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. The teaching assistant will NOT even touch the keyboard during the demonstration; so you should not depend on the teaching assistant to test your system. If you fail to demo some important features of your system, it will be assumed that your system does not have those features.
- You need to practice before you come to the demonstration. Debugging will not be allowed during the demonstration.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to the teaching assistant. You are given 15 minutes to show how good your system is. If you have implemented a great system, but fail to show how good it is, it will be reflected in your final grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in the demonstration.