



**The Laboratory of Computer Networks Security
(UE19CS326)**

Documented by Anurag.R.Simha

SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	15/11/2021
Section	:	A
Week	:	7

The Table of Contents

The Setup	2
Overview	3
Task 1: VM Setup	3
Task 2: Setting Up the Firewall	6
Task 3: Bypassing the Firewall Using VPN	8
Task 4: Demonstration	17

The Setup

For the experimentation of various attacks, two virtual machines were employed.

1. The VPN Client machine (10.0.2.13)

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:64 errors:0 dropped:0 overruns:0 frame:0
          TX packets:76 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10162 (10.1 KB)  TX bytes:8567 (8.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:83 errors:0 dropped:0 overruns:0 frame:0
          TX packets:83 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:22308 (22.3 KB)  TX bytes:22308 (22.3 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$
```

2. The VPN Server machine (10.0.2.14)

```
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5469 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1658 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7769582 (7.7 MB)  TX bytes:132022 (132.0 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:166 errors:0 dropped:0 overruns:0 frame:0
          TX packets:166 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:31069 (31.0 KB)  TX bytes:31069 (31.0 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):~$
```

Overview

Organisations, Internet Service Providers (ISPs), and countries often block their internal users from accessing certain external sites. This is called *egress filtering*. For example, to prevent work-time distraction, many companies set up their egress firewalls to block social network sites, so their employee cannot access those sites from inside their network. The most commonly used technology to bypass egress firewalls is Virtual Private Network (VPN). The objective of this lab is to perceive the functioning of VPN and the manner VPN can aid in bypassing egress firewalls. An unembellished VPN is setup in this lab, and is employed to bypass firewalls.

Task 1: VM Setup

There are two virtual machines employed in this laboratory experiment, where, both are connected to the ‘NAT network’ adapter enabled. VM1 is inside the firewall and VM2 is outside the firewall. The objective is to help the machine inside the firewall to reach out to the external sites blocked by the firewall.

1. The network adapters are configured.

Server machine (10.0.2.14)

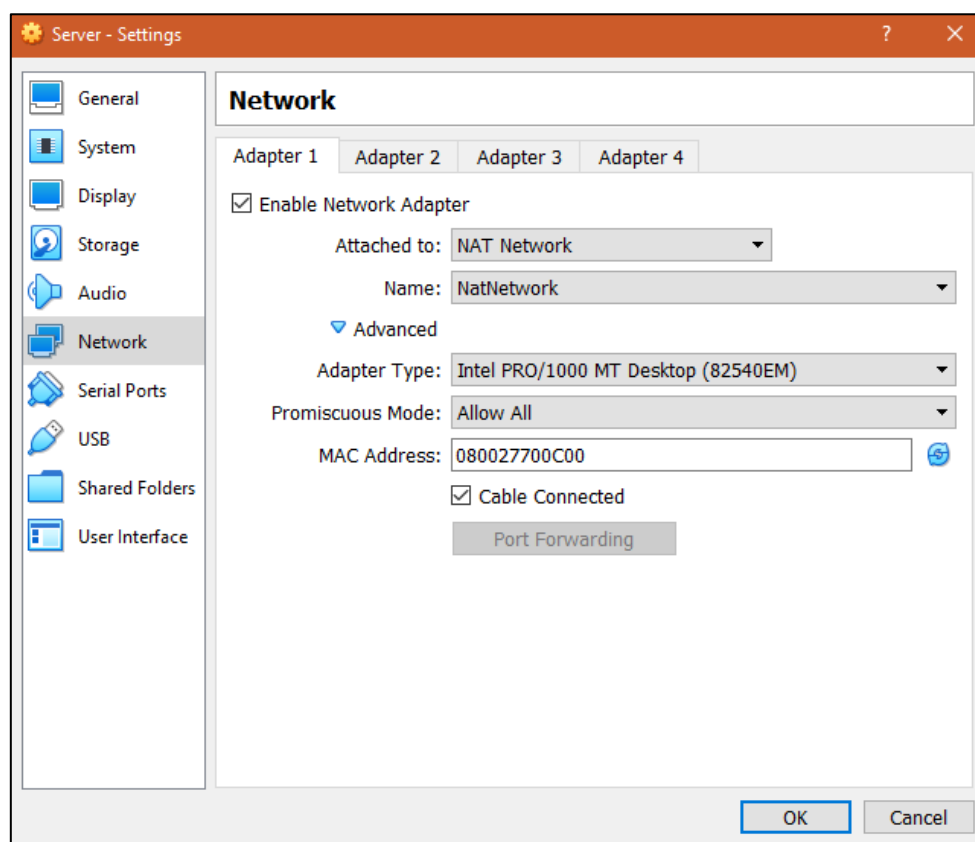


Fig. 1(a): The adapter on the server machine is connected to the NAT Network.

Client machine (10.0.2.13)

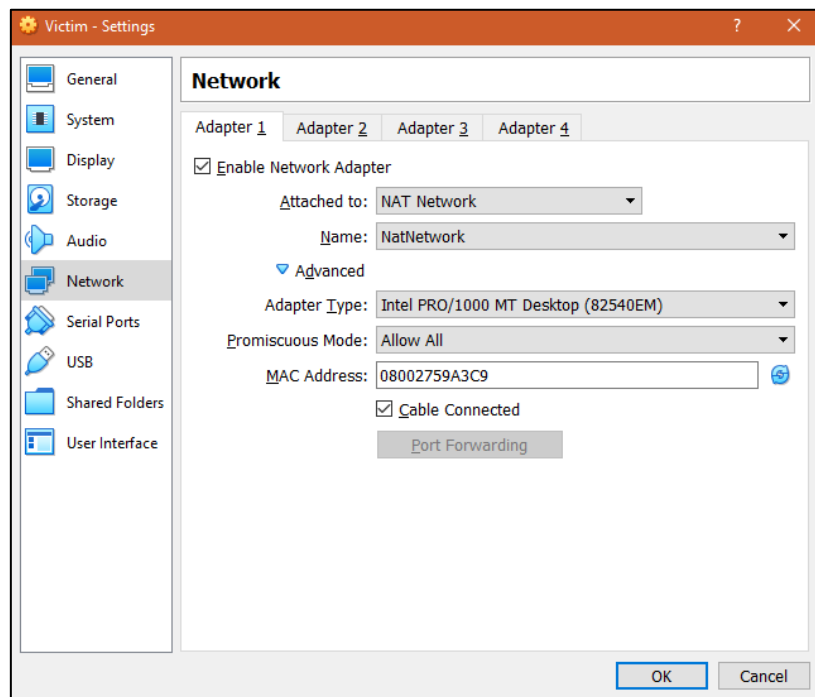


Fig. 1(b): The adapter on the client machine is connected to the NAT Network.

2. Configuring the connections (under the edit section) and examining them.

On the server machine:

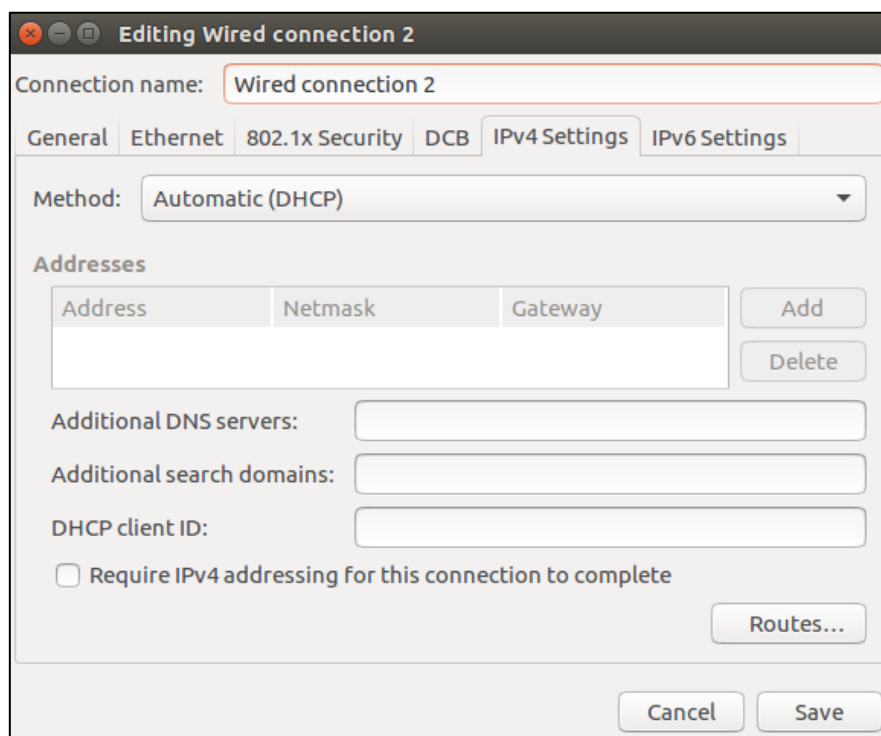


Fig. 1(c): The address and gateway are automatically configured to the NAT network (Wired Connection 2).

For inspection's sake, the command 'ifconfig' is used.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2) :~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
        inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:5469 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1658 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:7769582 (7.7 MB)  TX bytes:132022 (132.0 KB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:166 errors:0 dropped:0 overruns:0 frame:0
        TX packets:166 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:31069 (31.0 KB)  TX bytes:31069 (31.0 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2) :~$
```

Fig. 1(d): Testing the triumph of the connections.

Henceforth, the test yields a triumphant result in the connections.

On the client machine, the IP address is set by DHCP (Automatic):

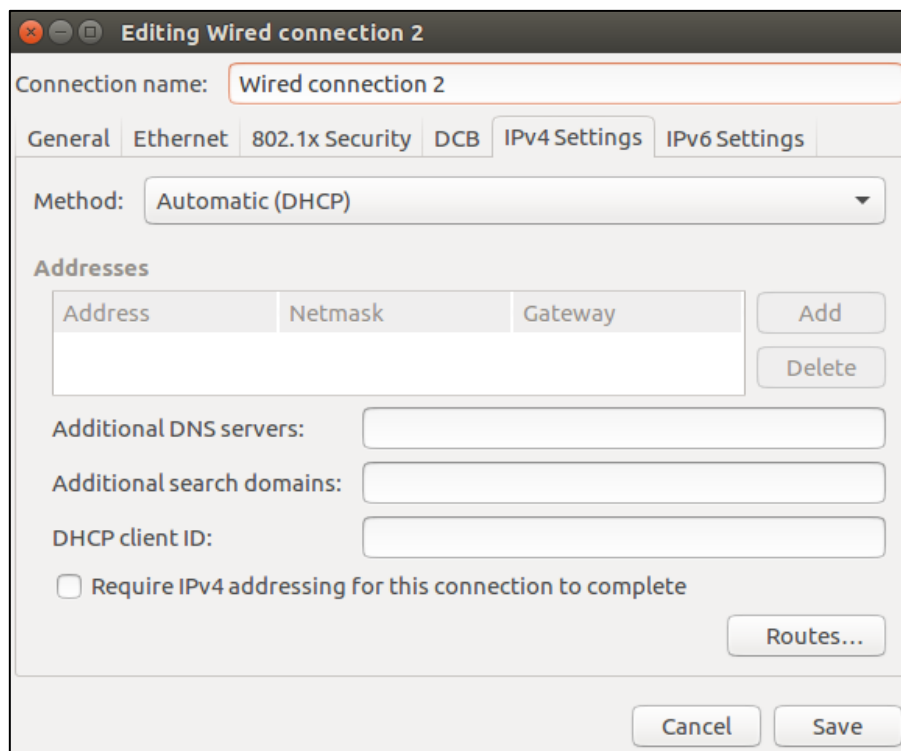


Fig. 1(e): The address and gateway are automatically configured by the DHCP (Wired Connection 2).

For inspection's sake, the command 'ifconfig' is used.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:178 errors:0 dropped:0 overruns:0 frame:0
          TX packets:131 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:34049 (34.0 KB)  TX bytes:14332 (14.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:113 errors:0 dropped:0 overruns:0 frame:0
          TX packets:113 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24439 (24.4 KB)  TX bytes:24439 (24.4 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Fig. 1(f): Testing the triumph of the connections.

Henceforth, the test yields a triumphant result in the connections.

Task 2: Setting Up the Firewall

In this task, a firewall is setup on VM1 to block the access of a target website. The following steps must be abided.

1. The website to block access for VM 1 is first selected. Here, the domain, www.example.net is the choice. For, it's IP address remains constant.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ ping www.example.net
PING www.example.net (93.184.216.34) 56(84) bytes of data.
64 bytes from 93.184.216.34: icmp_seq=1 ttl=48 time=216 ms
64 bytes from 93.184.216.34: icmp_seq=2 ttl=48 time=218 ms
64 bytes from 93.184.216.34: icmp_seq=3 ttl=48 time=218 ms
64 bytes from 93.184.216.34: icmp_seq=4 ttl=48 time=216 ms
64 bytes from 93.184.216.34: icmp_seq=5 ttl=48 time=216 ms
^C
--- www.example.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 216.443/217.362/218.368/0.874 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$
```

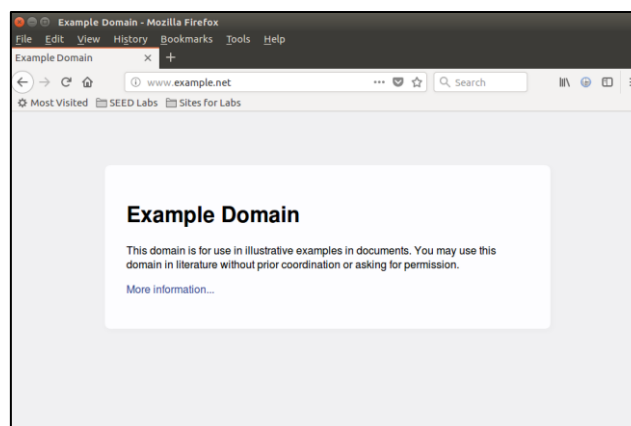


Fig. 2(a): A ping and access to google is successful.

Therefore, the IP address is 93.184.216.34.

2. The firewall is now activated.

The commands:

```
sudo ufw enable
```

```
sudo ufw deny out on enp0s3 to 93.184.216.0/24
```

```
sudo ufw status
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ sudo ufw enable
Firewall is active and enabled on system startup
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ sudo ufw deny out on enp0s3 to 93.184.216.0/24
Rule added
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ sudo ufw status
Status: active

To Action From
--
93.184.216.0/24 DENY OUT Anywhere on enp0s3

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$ ping www.example.net
PING www.example.net (93.184.216.34) 56(84) bytes of data:
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
^C
--- www.example.net ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4095ms

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):~$
```

Fig. 2(b): Activating the firewall and testing its triumph.

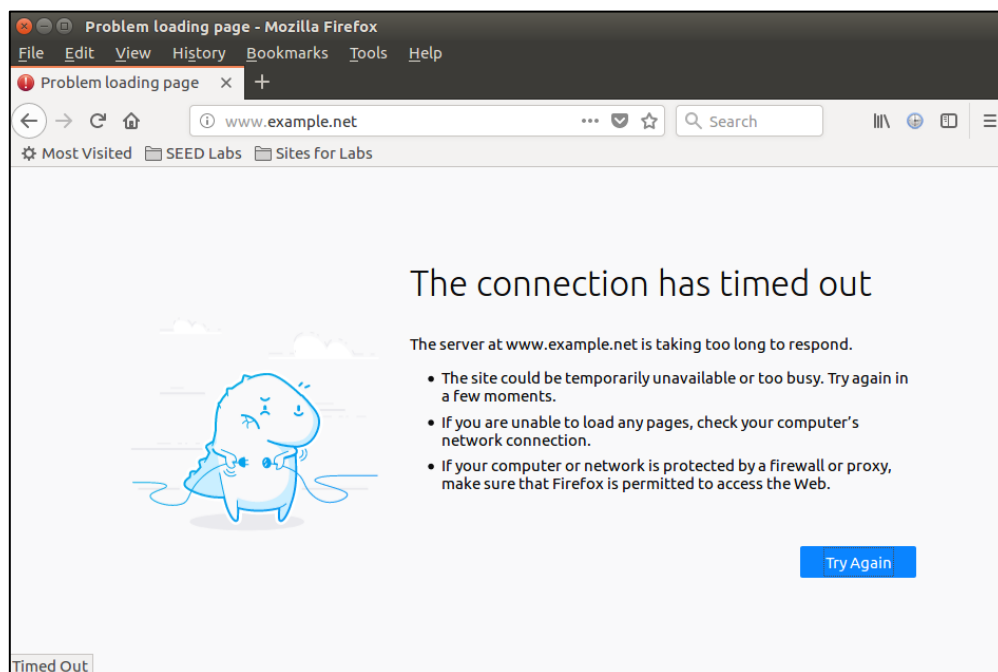


Fig. 2(c): The access remains restricted.

From figure 2(b), it's manifested that the access to google.com is unauthorised and remains insuperable (till the firewall stays activated).

There's zilch to manifest from a Wireshark packet capture.

Source	Destination	Protocol
10.0.2.13	10.0.2.14	DNS
10.0.2.14	10.0.2.13	DNS

Fig. 2(d): Capturing the packets on Wireshark when the access is unauthorised.

Task 3: Bypassing the Firewall Using VPN

Here, a VPN tunnel is established between VM1 (VPN Client VM) and VM2 (VPN Server VM). When a user on VM1 tries to access a blocked site, the traffic will not directly go through its network adapter, because it will be blocked. Instead, the packets to the blocked site from VM1 will be routed to the VPN tunnel and arrive at VM2. Once they arrive there, VM2 will route them to the final destination. When the reply packets come back, it will come back to VM2, which will then redirect the packets to the VPN tunnel, and eventually get the packet back to VM1. That is the manner by which VPN aids VM1 to bypass firewalls. Below is the pictorial representation of this description.

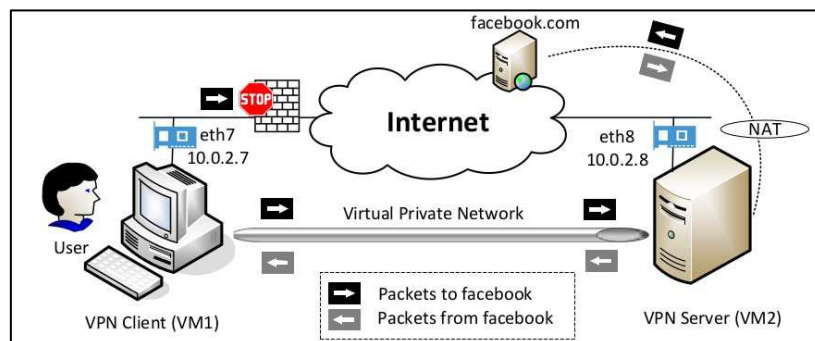


Fig. 3(a): Contacting the blocked website via VPN.

The VPN client and server programs connect to the hosting system via a TUN interface, through which they do two things:

1. Get the IP packets from the hosting system, aiding the packets that can be sent through the tunnel.
2. Get the IP packets from the tunnel, and then forward it to the hosting system, which will forward the packet to its final destination.

In the diagram below, the functioning of a virtual private network is pictorially explained. This diagram is the pedestal to triumph the further task(s) in this experiment.

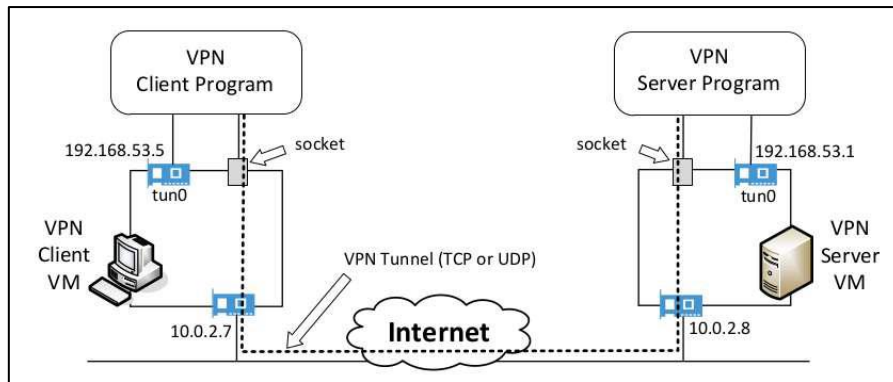


Fig. 3(b): Functioning of the VPN.

Below are the programs, vpnserver.c and vpnclient.c

Name: vpnserver.c

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <sys/ioctl.h>

#define PORT_NUMBER 55555
#define SERVER_IP "10.0.2.14"
#define BUFF_SIZE 2000

struct sockaddr_in peerAddr;

int createTunDevice() {
    int tunfd;
    struct ifreq ifr;
    memset(&ifr, 0, sizeof(ifr));

    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;

    tunfd = open("/dev/net/tun", O_RDWR);
    ioctl(tunfd, TUNSETIFF, &ifr);

    return tunfd;
}

int initUDPServer() {
    int sockfd;
    struct sockaddr_in server;
    char buff[100];
```

```

memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(PORT_NUMBER);

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bind(sockfd, (struct sockaddr*) &server, sizeof(server));

// Wait for the VPN client to "connect".
bzero(buff, 100);
int peerAddrLen = sizeof(struct sockaddr_in);
int len = recvfrom(sockfd, buff, 100, 0,
                  (struct sockaddr *) &peerAddr, &peerAddrLen);

printf("Connected with the client: %s\n", buff);
return sockfd;
}

void tunSelected(int tunfd, int sockfd){
    int len;
    char buff[BUFF_SIZE];

    printf("Got a packet from TUN\n");

    bzero(buff, BUFF_SIZE);
    len = read(tunfd, buff, BUFF_SIZE);
    sendto(sockfd, buff, len, 0, (struct sockaddr *) &peerAddr,
          sizeof(peerAddr));
}

void socketSelected (int tunfd, int sockfd){
    int len;
    char buff[BUFF_SIZE];

    printf("Got a packet from the tunnel\n");

    bzero(buff, BUFF_SIZE);
    len = recvfrom(sockfd, buff, BUFF_SIZE, 0, NULL, NULL);
    write(tunfd, buff, len);
}

int main (int argc, char * argv[]) {
    int tunfd, sockfd;

    tunfd = createTunDevice();
    sockfd = initUDPServer();

```

```

// Enter the main loop
while (1) {
    fd_set readFDSet;

    FD_ZERO(&readFDSet);
    FD_SET(sockfd, &readFDSet);
    FD_SET(tunfd, &readFDSet);
    select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

    if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd);
    if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd);
}
}

```

Name: vpnclient.c

```

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <sys/ioctl.h>

#define BUFF_SIZE 2000
#define PORT_NUMBER 55555
#define SERVER_IP "10.0.2.14"
struct sockaddr_in peerAddr;

int createTunDevice() {
    int tunfd;
    struct ifreq ifr;
    memset(&ifr, 0, sizeof(ifr));

    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;

    tunfd = open("/dev/net/tun", O_RDWR);
    ioctl(tunfd, TUNSETIFF, &ifr);

    return tunfd;
}

int connectToUDPServer(){
    int sockfd;
    char *hello="Hello";

    memset(&peerAddr, 0, sizeof(peerAddr));
    peerAddr.sin_family = AF_INET;

```

```

peerAddr.sin_port = htons(PORT_NUMBER);
peerAddr.sin_addr.s_addr = inet_addr(SERVER_IP);

sockfd = socket(AF_INET, SOCK_DGRAM, 0);

// Send a hello message to "connect" with the VPN server
sendto(sockfd, hello, strlen(hello), 0,
        (struct sockaddr *) &peerAddr, sizeof(peerAddr));

return sockfd;
}

void tunSelected(int tunfd, int sockfd){
    int len;
    char buff[BUFF_SIZE];

    printf("Got a packet from TUN\n");

    bzero(buff, BUFF_SIZE);
    len = read(tunfd, buff, BUFF_SIZE);
    sendto(sockfd, buff, len, 0, (struct sockaddr *) &peerAddr,
           sizeof(peerAddr));
}

void socketSelected (int tunfd, int sockfd){
    int len;
    char buff[BUFF_SIZE];

    printf("Got a packet from the tunnel\n");

    bzero(buff, BUFF_SIZE);
    len = recvfrom(sockfd, buff, BUFF_SIZE, 0, NULL, NULL);
    write(tunfd, buff, len);
}

int main (int argc, char * argv[]) {
    int tunfd, sockfd;

    tunfd = createTunDevice();
    sockfd = connectToUDPServer();

    // Enter the main loop
    while (1) {
        fd_set readFDSet;

        FD_ZERO(&readFDSet);
        FD_SET(sockfd, &readFDSet);

```

```

    FD_SET(tunfd, &readFDSet);
    select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

    if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd);
    if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd);
}
}

```

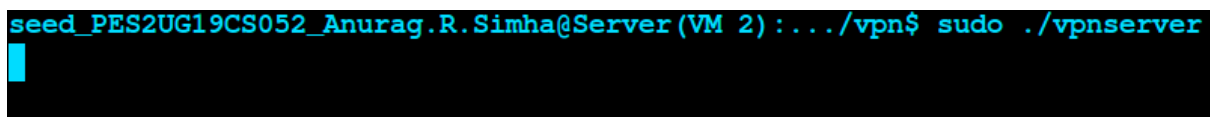
Step 1: Running the VPN server program (On the server machine/VM 2)

First, VPN server program (vpnsrv.c) is put into effect on the Server VM. After the program runs, a virtual TUN network interface will appear in the system (It's visible using the "ifconfig -a" command; the name of the interface is tun0). This new interface is not yet configured, so it's configured by providing an IP address. 192.168.53.1 is the IP address used for this interface.

The following commands are run:

```
sudo ./vpnsrv
```

This command instigates the program.



```

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2): .../vpn$ sudo ./vpnsrv

```

Fig. 3(c): The program has instigated.

On another terminal, the command below is put into force.

```
sudo ifconfig tun0 192.168.53.1/24 up
```

This command leads to the opening of the tunnel.

```
ifconfig
```

This aids in examining the activation of the tunnel.

On the outset of this program, to establish a mode of contact between the client machine and the server machine, a configuration of the tunnel interface is a vital desideratum. Therefore, with an IP address assigned to it, the tunnel is opened by the aid of the command that's mentioned above. From the screenshot that's underneath this description, it's facile to espy the opening of a tunnel. The rectangle with a border that's red in colour encompasses this observation.

```

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ sudo ifconfig tun0 192.168.53.1/24 up
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
            inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:5901 errors:0 dropped:0 overruns:0 frame:0
            TX packets:2175 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:7880605 (7.8 MB)  TX bytes:175813 (175.8 KB)

lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING  MTU:65536  Metric:1
            RX packets:261 errors:0 dropped:0 overruns:0 frame:0
            TX packets:261 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1
            RX bytes:37286 (37.2 KB)  TX bytes:37286 (37.2 KB)

tun0        Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
            inet addr:192.168.53.1  P-t-P:192.168.53.1  Mask:255.255.255.0
            inet6 addr: fe80::f8ab:3580:fdal:ca76/64 Scope:Link
            UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:500
            RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$

```

Fig. 3(d): The interface, tun0 is activated.

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

This command aids in forwarding the IP packets.

```

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$

```

Fig. 3(e): The IP forwarding action is enabled.

Step 1: Running the VPN client program (On the client machine/VM 1)

Here, the program, vpnclient.c is executed. To the tunnel interface, 192.168.53.5 is the IP address assigned.

Below are the commands list:

```
sudo ./vpnclient 10.0.2.14
```

This command line instruction commences the client program to connect with the server program thus achieving (fractionally) a VPN connection.

```

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$ sudo ./vpnclient 10.0.2.14
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN

```

```

seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2) :.../vpn$ sudo ./vpnsrv
Connected with the client: Hello
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel

```

Fig. 3(f): Activating the VPN client program, and receiving the response on the server machine.

It's observed that, before configuring the tunnel on the client machine, there's no packet received from TUN. But, on a triumphant configuration of the tunnel interface, there're packets received over both the machines. Thus, both sides communication is (fractionally) achieved.

Next, on another terminal, the tunnel interface is activated.

```
sudo ifconfig tun0 192.168.53.5/24 up
```

This command activates the tunnel.

```

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1) :.../vpn$ sudo ifconfig tun0 192.168.53.5/24 up
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1) :.../vpn$ ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
            inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::5f33:85f1:5546:41d0/64  Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:179 errors:0 dropped:0 overruns:0 frame:0
            TX packets:254 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:84575 (84.5 KB)  TX bytes:27348 (27.3 KB)

lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128  Scope:Host
            UP LOOPBACK RUNNING  MTU:65536  Metric:1
            RX packets:116 errors:0 dropped:0 overruns:0 frame:0
            TX packets:116 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1
            RX bytes:25103 (25.1 KB)  TX bytes:25103 (25.1 KB)

tun0        Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
            inet addr:192.168.53.5  P-t-P:192.168.53.5  Mask:255.255.255.0
            inet6 addr: fe80::cc28:b503:24c1:69a2/64  Scope:Link
            UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:500
            RX bytes:0 (0.0 B)  TX bytes:96 (96.0 B)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1) :.../vpn$

```

Fig. 3(g): The tunnel is open on VM 1 now.

Step 3: Setting Up Routing on the Client and Server VMs.

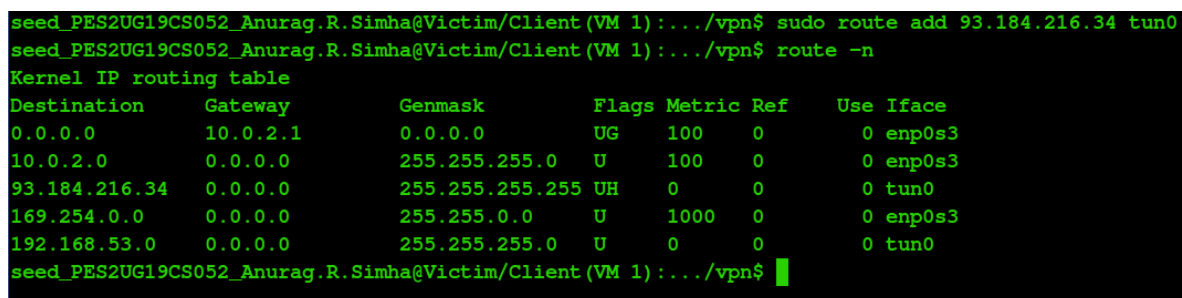
After the above two steps, the tunnel will be established. Before the tunnel can be used, the routing paths must be setup on both client and server machines to direct the intended traffic through the tunnel.

Q. Please analyse and provide screenshots of which command need to be implemented on the VPN client and the server.

Below are the commands:

(Only) On the client machine:

```
sudo route add 93.184.216.34 tun0
route -n
```



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$ sudo route add 93.184.216.34 tun0
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$ route -n
Kernel IP routing table
Destination    Gateway      Genmask      Flags Metric Ref    Use Iface
0.0.0.0        10.0.2.1     0.0.0.0      UG    100    0      0 enp0s3
10.0.2.0       0.0.0.0      255.255.255.0 U    100    0      0 enp0s3
93.184.216.34  0.0.0.0      255.255.255.255 UH    0      0      0 tun0
169.254.0.0    0.0.0.0      255.255.0.0  U    1000   0      0 enp0s3
192.168.53.0   0.0.0.0      255.255.255.0 U    0      0      0 tun0
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$
```

Fig. 3(h): Adding the entry to the routing table on VM 1/VPN Client.

There's unreservedly no sine qua non to configure anything over the server machine. This setting is apposite to triumph the task.

A question that arises to one's mind is the rationale behind this choice to the routing table. Rudimentary knowledge on routing conveys that on accessing anything over the internet, packets are bound to uphold the instructions under the routing table and follow in the footsteps directed by it. Henceforth, when the entry above is made, (on pinging) ICMP packets or (on browsing) DNS packets ought to travel by dint of the tunnel interface that's manifested as 'tun0'. With this rule in place, the routing table fuelled by a couple of rules under the iptables (in the next step), overshadow the instructions under the firewall table, thus leading the packets to reach the desired websites in a preferably facile manner.

Step 4: Setting Up NAT on Server VM:

When the final destination sends packets back to users, the packet will be sent to the VPN Server first. This is due to the reason that the server has access to the google server and returns it to the client. To reach the Internet, these packets will go through another NAT, which is provided by VirtualBox, but since the source IP is the Server VM, this second NAT will have no problem relaying

back the returned packets from the Internet to the Server VM. The following commands can enable the NAT on the Server VM:

```
sudo iptables -F
```

```
sudo iptables -t nat -F
```

Next, a rule is added on post routing position to the NAT Network adapter (enp0s3) connected to the VPN server.

```
sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ sudo iptables -F
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ sudo iptables -t nat -F
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3
seed_PES2UG19CS052_Anurag.R.Simha@Server (VM 2):.../vpn$
```

Fig. 3(i): Adding rules to iptables.

Task 4: Demonstration

Now, the website, www.example.net is attempted for a connection. Previously, in task 2, it was noticed that the access to this domain was unauthorised when the firewall remained on. In this task, it's obligatory to receive packets, and access the website despite the activation of a firewall.

The screenshots below demonstrate the triumph in connecting to the website, www.example.net.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$ ping www.example.net
PING www.example.net (93.184.216.34) 56(84) bytes of data:
64 bytes from 93.184.216.34: icmp_seq=1 ttl=47 time=218 ms
64 bytes from 93.184.216.34: icmp_seq=2 ttl=47 time=218 ms
64 bytes from 93.184.216.34: icmp_seq=3 ttl=47 time=219 ms
64 bytes from 93.184.216.34: icmp_seq=4 ttl=47 time=218 ms
64 bytes from 93.184.216.34: icmp_seq=5 ttl=47 time=217 ms
^C
--- www.example.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 217.338/218.531/219.066/0.694 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client (VM 1):.../vpn$
```

Fig. 4(a): A ping to the website yields triumphant results.

This occurs despite an activation of the firewall. The figure below is the manifestation.

- P.T.O -

In the figure 4(d), it's observed that a ping to the website, www.example.net is indeed triumphant. Here, the UDP packets (blue colour) are those that occur in the tunnel traffic. The packets in pink are the ICMP packet which is the manifestation of a triumphant ping action.

Source	Destination	Protocol	Length	Info
::1	::1	UDP	64	45564 → 48121 Len=0
10.0.2.13	10.0.2.14	DNS	77	Standard query 0x94ec A www.example.net
10.0.2.13	10.0.2.14	DNS	77	Standard query 0xd11d AAAA www.example.net
192.168.53.5	93.184.216.34	HTTP	482	GET / HTTP/1.1
10.0.2.13	10.0.2.14	UDP	510	48943 → 55555 Len=466
10.0.2.14	10.0.2.13	DNS	226	Standard query response 0x94ec A www.example.net A 93.184.216.34
10.0.2.14	10.0.2.13	DNS	238	Standard query response 0xd11d AAAA www.example.net AAAA 2606:2800:4000:0000:0000:0000:0000:0000
10.0.2.14	10.0.2.13	UDP	84	55555 → 48943 Len=40
93.184.216.34	192.168.53.5	TCP	56	80 → 43324 [ACK] Seq=1433290 Ack=3524476439 Win=31916 Len=0
10.0.2.14	10.0.2.13	UDP	383	55555 → 48943 Len=339
93.184.216.34	192.168.53.5	HTTP	355	HTTP/1.1 304 Not Modified
192.168.53.5	93.184.216.34	TCP	56	43324 → 80 [ACK] Seq=3524476439 Ack=1433589 Win=31088 Len=0
10.0.2.13	10.0.2.14	UDP	84	48943 → 55555 Len=40

Fig. 4(e): The Wireshark packet capture on accessing the website.

In the figure 4(e), it's observed that an access to the website, www.example.net is indeed triumphant. The DNS and UDP packets are sent and received adding to the clarity. There are HTTP packets received too, which augments the clarity of the action.
