Firewall Evasion Lab: Bypassing Firewalls using VPN

The SEED project was funded by multiple grants from the US National Science Foundation.

1 Overview

Organizations, 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. For political reasons, many countries set up egress filtering at their ISPs to block their people from accessing selected foreign websites. Unfortunately, these firewalls can be easily bypassed, and services/products that help users bypass firewalls are widely available on the Internet. The most commonly used technology to bypass egress firewalls is Virtual Private Network (VPN). In particular, this technology is widely used by smartphone users that are affected by egress filtering; there are many VPN apps (for Android, iOS, and other platforms) that can help users bypass egress firewalls.

The learning objective of this lab is for students to see how VPN works in action and how VPN can help bypass egress firewalls. We will implement a very simple VPN in this lab, and use it to bypass firewalls. A typical VPN depends on two pieces of technologies: IP tunneling and encryption. The tunneling technology is the most essential one to help bypass firewalls; the encryption technology is for protecting the content of the traffic that goes through the VPN tunnel. For the sake of simplicity, we will only focus on the tunneling part, so the traffic inside our tunnel is not encrypted. We have a separate VPN lab, which covers both tunneling and encryption. If readers are interested, they can work on our VPN lab to learn how to build a complete VPN. In this lab, we only focus on how to use VPN tunnel to bypass firewalls. This lab covers the following topics:

- Firewall
- VPN

Readings and videos. Detailed coverage of firewalls, firewall evasion techniques, and VPN can be found in the following

- Chapters 17 and 19 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.
- Sections 8 and 9 of the SEED Lecture, *Internet Security: A Hands-on Approach*, by Wenliang Du. See details at `https://www.handsonsecurity.net/video.html`.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Lab Tasks

2.1 Task 1: VM Setup

We need two machines, one inside the firewall, and the other outside the firewall. The objective is to help the machine inside the firewall to reach out to the external sites blocked by the firewall. We use two virtual machines, VM1 and VM2, for these two machines. VM1 and VM2 are supposed to be two machines connected via the Internet through routers. This setup may require more than two VMs. For the sake of simplicity, we use a LAN to emulate the Internet connection. Basically, we simply connect VM1 and VM2 to a LAN using the `NAT Network` adapter. Figure 1 depicts the lab setup.

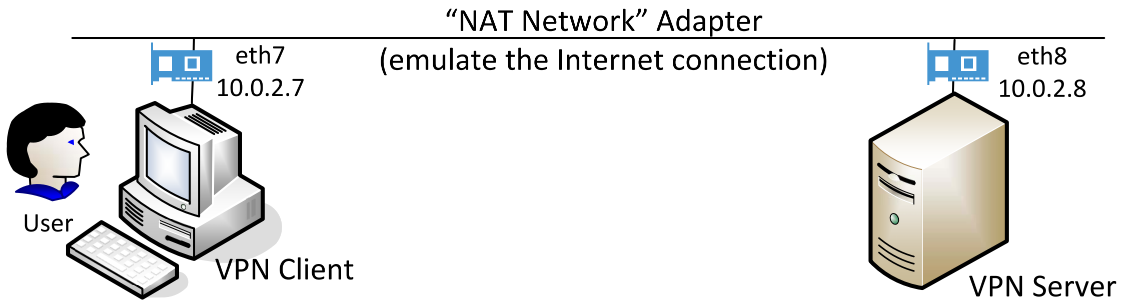

Figure 1: Lab Environment Setup

2.2 Task 2: Set up Firewall

In this task, you will set up a firewall on VM1 to block the access of a target website. You need to make sure that the IP address of the target web site is either fixed or in a fixed range; otherwise, you may have trouble completely blocking this website. Please refer to the Firewall Lab for details about how to blocking websites.

In the real world, the firewall should run on a separate machine, not on VM1. To minimize the number of VMs used in this lab, we put the firewall on VM1. Setting up the firewall on VM1 requires the superuser privilege, and so does the setup of the VPN tunnel. One may immediately say that if we already have the superuser privilege, why cannot we just simply disable the firewall on VM1. This is a good argument, but keep in mind, we put the firewall on VM1 simply because we do not want to create another VM in the lab environment. Therefore, although you have the superuser privilege on VM1, you are not allowed to use the privilege to reconfigure the firewall. You have to use VPN to bypass it.

Compared to putting the firewall on an external machine, putting the firewall on VM1 does have a small issue that we need to deal with. When we set up the firewall to block packets, we need to make sure not to block the packets from getting to the virtual interface used by the VPN, or even our VPN will not be able to get the packets. Therefore, we cannot set the firewall rule before the routing, nor can we set the firewall rule on the virtual interface. We just need to set the rule on VM1's real network interface, so it will not affect the packets that go to the virtual interface.

We can use the `ufw` program to set up firewall rules. This program is a front-end of the `iptables` program. One can directly use `iptables`, but for this experiment, it is more convenient to use `ufw`.

```
$ sudo ufw enable
$ sudo ufw deny out on eth12 to 128.230.210.0/24
$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
128.230.210.0/24          DENY OUT    Anywhere on eth12
```

Please identify a website that you would like to block, set up the firewall, and then demonstrate that your firewall is working and the target IP address is no longer reachable. Provide screenshots in your lab report.

2.3 Task 3: Bypassing Firewall using VPN

The idea of using VPN to bypass firewall is depicted in Figure 2. We establish a VPN tunnel 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 how the VPN helps VM1 to bypass firewalls.

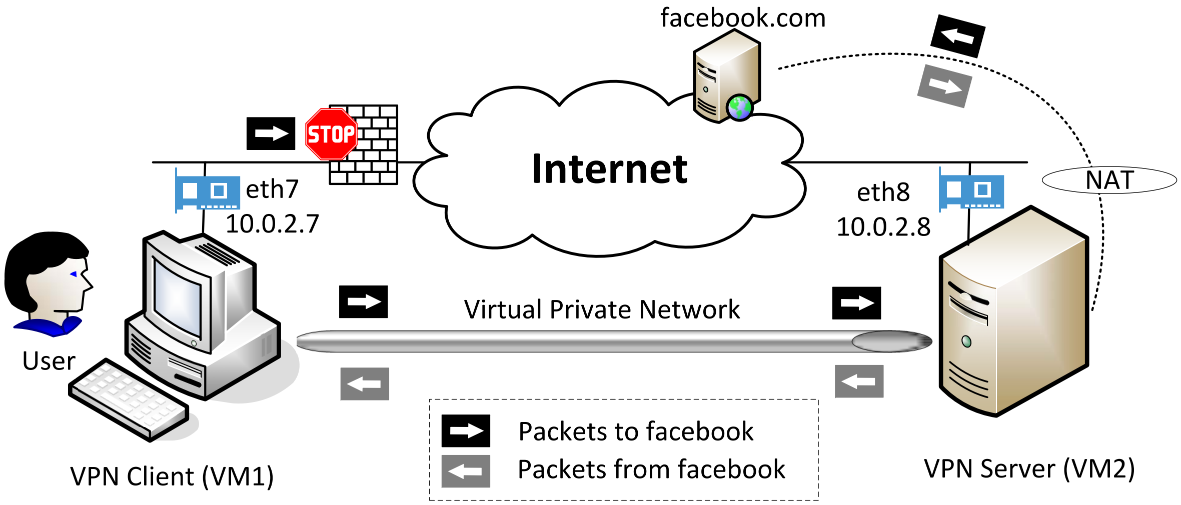

Figure 2: Bypassing firewall using VPN

We have created a sample VPN program, including a client program (`vpnclient`) and a server program (`vpnserver`), both of which can be downloaded from this lab's web site. This simple VPN program only establishes a VPN tunnel between the client and server; it does not encrypt the tunnel traffic. The program is explained in detail in Chapter 16 of the SEED book titled *Computer Security: A Hands-on Approach*; the chapter also explains how TUN/TAP works and how to use it to create VPN.

The `vpnclient` and `vpnserver` programs are the two ends of a VPN tunnel. They communicate with each other using either TCP or UDP via the sockets depicted in Figure 3. In our sample code, we choose to use UDP for the sake of simplicity. The dotted line between the client and server depicts the path for the VPN tunnel. The VPN client and server programs connect to the hosting system via a TUN interface, through which they do two things: (1) get IP packets from the hosting system, so the packets can be sent through the tunnel, (2) get IP packets from the tunnel, and then forward it to the hosting system, which will forward the packet to its final destination. The following procedure describes how to create a VPN tunnel using the `vpnclient` and `vpnserver` programs.

Step 1: Run VPN Server. We first run the VPN server program `vpnserver` on the Server VM. After the program runs, a virtual TUN network interface will appear in the system (we can see it using the

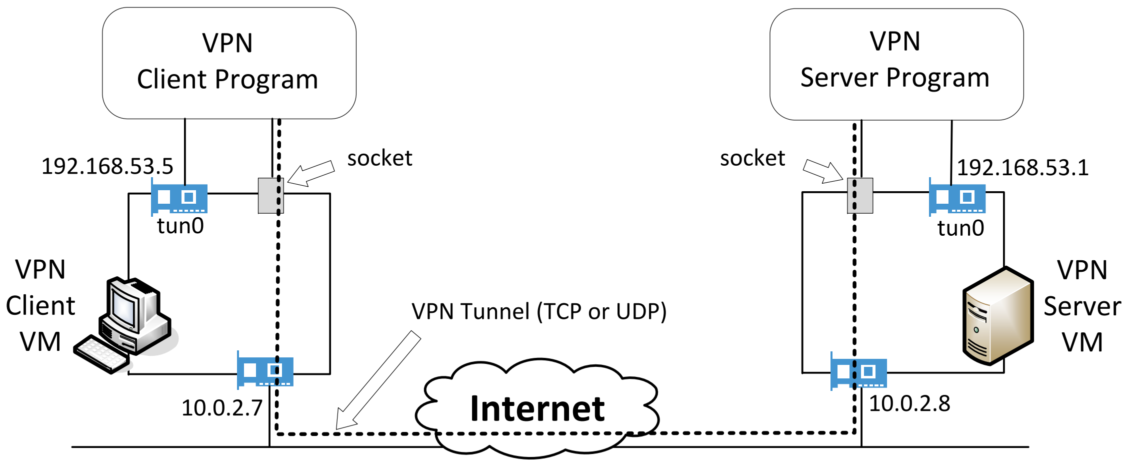

Figure 3: VPN client and server

"`ifconfig -a`" command; the name of the interface will be `tun0` in most cases, but they can be `tunX`, where `X` is a number). This new interface is not yet configured, so we need to configure it by giving it an IP address. We use `192.168.53.1` for this interface, but you can use other IP addresses.

Run the following commands. The first command will start the server program, and the second command assigns an IP address to the `tun0` interface and then activates it. It should be noted that the first command will block and wait for connections, so we need to find another window run the second command.

```
$ sudo ./vpnserver

Run the following command in another window:
$ sudo ifconfig tun0 192.168.53.1/24 up
```

Unless specifically configured, a computer will only act as a host, not as a gateway. The VPN Server needs to forward packets to other destinations, so it needs to function as a gateway. We need to enable the IP forwarding for a computer to behave like a gateway. IP forwarding can be enabled using the following command:

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

Step 2: Run VPN Client. We now run the VPN client program on the Client VM. We run the following command on this machine (the first command will connect to the VPN server program running on `10.0.2.8`. This command will block as well, so we need to find another window to configure the `tun0` interface created by the VPN client program. We assign IP address `192.168.53.5` to the `tun0` interface (you can choose other IP addresses).

```
On VPN Client VM:
$ sudo ./vpnclient 10.0.2.8

Run the following command in a different window
$ sudo ifconfig tun0 192.168.53.5/24 up
```

Step 3: Set Up Routing on Client and Server VMs. After the above two steps, the tunnel will be established. Before we can use the tunnel, we need to set up routing paths on both client and server machines to direct the intended traffic through the tunnel. We can use the `route` command to add an routing entry. The following example shows how to route the `10.20.30.0/24`-bound packets to the interface `eth0`.

```
$ sudo route add -net 10.20.30.0/24 eth0
```

To bypass firewalls on the Client VM, you need to set up routing entries accordingly, so the traffics to the blocked site will be routed towards the VPN. You need to think about what routing entries to add in order to bypass the firewall.

Step 4: Set Up NAT on Server VM. When the final destination sends packets back to users, the packet will be sent to the VPN Server first (think about why and write down your answer in the report). The return packet will reach the VPN Server's NAT adapter first (because the source IPs of all the outgoing packets from the Server VM are changed to the NAT's external IP address (which is basically the host computer's IP address in our setup). Usually, the NAT will replace the destination IP address with the IP address of the original packet (i.e. `192.168.53.5` in our case), and give it back to whoever owns the IP address. Unfortunately, we have a problem here.

Before the NAT sends out the packet, it needs to know the MAC address of the machine who owns `192.168.53.5`, so it sends an ARP request. Our private network is virtual, and this IP address belongs to the `tun0` interface on the VPN Client. therefore, `192.168.53.5` will not receive the ARP request (even if it does, it has no use). The NAT will then drop the packet, because the recipient does not exist.

The actual recipient should be the VPN Server VM, even though it does not own the IP address `192.168.53.5`. If we can configure the NAT as a gateway, we can ask the NAT to route the packets for `192.168.53.5` to the VPN Server, which will eventually deliver the packets through the tunnel to the VPN Client. However, we have not figured out how to configure the NAT as a gateway in VirtualBox, we did come up two work-around solutions. One idea is to "fool" the NAT to believe that the MAC address of `192.168.53.5` is the VPN Server VM's MAC address, so the packet will be delivered to the VPN Server by the NAT. We can achieve this using an ARP cache poisoning on the NAT, basically telling the NAT before hand about the MAC address of `192.168.53.5`.

A better solution to get round the limitation of the NAT is to create another NAT right on the Server VM, so all packets coming out of the Server VM will have this VM's IP address as their source IP. 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. Using this solution, we do not need to use ARP cache poisoning to "fool" the NAT any more. The following commands can enable the NAT on the Server VM (in your case, the name of the `NAT Network` adapter may not be called `eth8`; you just need to find its real name on your VM):

```
Clean all iptables rules:
$ sudo iptables -F
$ sudo iptables -t nat -F

Add a rule on postrouting position to the NatNetwork adapter (eth8)
   connected to VPN server.
$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o eth8
```

Demonstration. If you have done the steps above correctly, you should be able to bypass the firewall. You should show that you can reach the blocked web site from Client VM via the VPN. Your solution should not only work for web traffic, but also for all other traffic. For example, if the blocked machine runs a `telnet` server, you should be able to `telnet` to this blocked server from Client VM.

In your lab report, you should provide the evidence to show that your traffic did go through the VPN tunnel, not through some "side doors". The best way to show that is to capture the network traffic using Wireshark, and describe the path of your packets using the captured traffic. Without such an evidence, we have no idea whether your success is due to a mis-configured firewall (i.e. the targeted web site is not blocked at all) or due to your VPN.

3 Submission Requirement

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.