CSCI4430/ESTR4120 (Spring 2018)

Assignment 3: Implementing NAT using NFQUEUE

Due on Apr 19, 2018 (Thur), 23:59:59

Abstract

In this assignment, you will implement a NAT application using the software library NFQUEUE. The NAT application can forward TCP traffic.

1 Setup

1.1 Topology

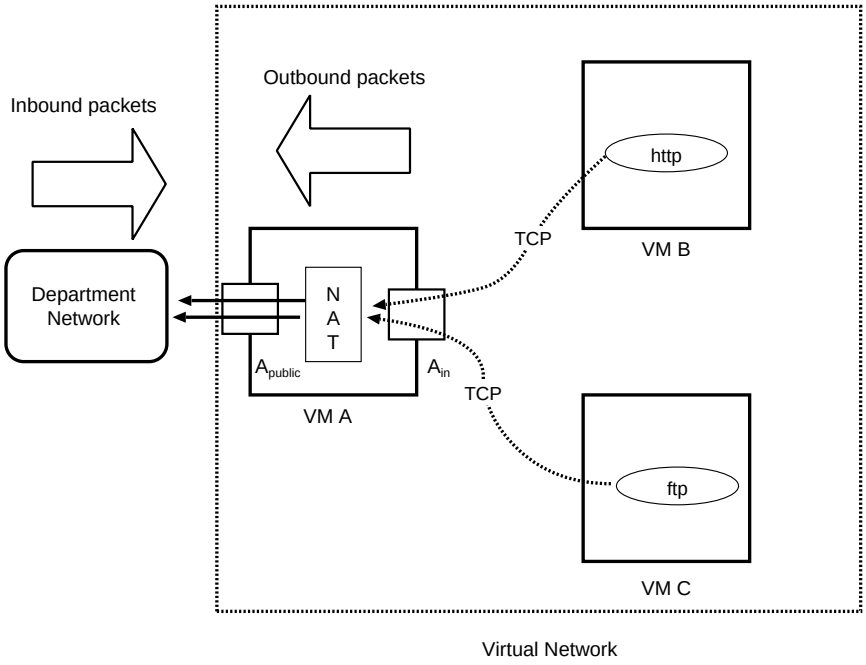

Figure 1: Setup of this assignment.

Figure 1 shows the setup of this assignment. VM A has a network card configured with a public IP address (i.e., A_{public}), and it will serve as the NAT gateway.

By properly configuring the route tables of VM B and VM C, VM A can relay any inbound/outbound traffic for VM B and VM C. Let A_{in} be the internal IP address of VM A. Then we execute the following command in both VM B and VM C to add the default gateway:

```
sudo route add default gw A_in
```

Note that you are NOT allowed to use any built-in NAT services, including the `nat` table in `iptables`. Otherwise, you will get zero marks.

```
IP="10.3.1.54"       # public interface
LAN="10.0.54.0"      # private LAN network address (without subnet mask)
MASK="24"            # subnet mask
echo "1" >  /proc/sys/net/ipv4/ip_forward
iptables -t nat -F
iptables -t filter -F
iptables -t mangle -F
iptables -t filter -A FORWARD -j NFQUEUE --queue-num 0 -p tcp -s ${LAN}/${MASK} \
     ! -d ${IP} --dport 10000:12000
iptables -t mangle -A PREROUTING -j NFQUEUE --queue-num 0 -p tcp -d ${IP} \
     --dport 10000:12000
```

Table 1: `iptables` configuration.

1.2 Assumptions and Restrictions

We make the following assumptions and restrictions.

- You only need to relay TCP traffic. Other protocols (e.g., UDP and ICMP) can be ignored. We won't generate any traffic aside TCP in our demo. To play safe, you can drop any unexpected packet.
- All new flows are initiated from the internal network.
- You only need to relay outbound traffic to one of the reachable workstations in the department. The reachable workstations include: linux1 - linux8. Since we don't translate UDP, we don't support DNS mappings. Thus, you should connect to them using their IP addresses directly.
- For each reachable workstation, you can only host a server on port 10000 - 12000 (inclusive) using TCP sockets.
- The translated source port at VM A must also be in the range 10000 - 12000 (inclusive). We assume that no other processes are using the port 10000 - 12000 while the NAT program is running.
- The NAT program maintains a translation table that keeps track of all NAT mappings of TCP. For each translation, the NAT program should always choose the **smallest, available port number** among all flows.
- We will provide you the implementation of computing the checksums (for details, please refer to tutorials).

1.3 `iptables`

In VM A, you need to configure `iptables` to use NFQUEUE. Table 1 shows the rules to be used.

The first three `iptables` commands clear all the existing rules. The 4th and 5th `iptables` commands redirect outbound and inbound TCP traffic, respectively.

The above script takes three inputs: (i) `$IP` is the IP address of public interface of VM A (i.e., A_{public} in Figure 1); (ii) `$LAN` is the network address of the private network (without the subnet mask); and (iii) `$MASK` is the subnet mask value. You must update the inputs for your own network.

2 TCP Translation

TCP translation starts with the outbound SYN packet and ends with the completion of the 4-way handshake or a RST packet.

1. For each outbound packet,

   (a) The NAT program searches if the **source IP-port pair** of the packet has already been stored in the translation table.

   (b) If not, then the NAT program creates a new entry in the translation table if and only if **the outbound packet is a SYN packet**. The entry should contain:

   - the source IP-port pair;
   - the newly assigned port number (between 10000 and 12000)

   (c) If the packet is not a SYN packet and the NAT program cannot find any matched entries in the translation table, the program should **drop** the packet.

   (d) If the packet is not a SYN packet but the program can find a matched entry, the program will use the previously assigned port number.

   (e) Finally, the NAT program translates the source IP address and the source port number of the packet, modifies the IP and TCP headers of the packet accordingly, and forwards it.

2. For each inbound packet,

   (a) The NAT program searches if the destination port of the inbound packet matches any one of the entries in the TCP translation table.

   (b) If yes, the NAT program translates its destination IP address and port number, modifies the IP and TCP headers of the packet accordingly, and sends it to the target VM.

   (c) If not, the NAT program should drop the packet.

3. **Connection monitoring requirement.** For each translation entry, an entry should stay valid until the 4-way handshake of the target connection has finished. In other words, the NAT program should monitor the progress of the 4-way handshake. Note that the 4-way handshake can be initiated by any one of the connection endpoints. Here we assume the 4-way handshake is always complete.

   Note that RST packets may arrive at your NAT program. Whenever a RST packet comes, the RST packet will be translated. The translation of that flow will stop since the flow should cease to exist after the delivery of the RST packet. To simplify your task, you do not need to validate the sequence number or the acknowledgment number of the RST packet.

   The entry in NAT can be removed if and only if the 4-way handshake is complete or RST packet is received.

To enable us to check the content of the NAT table, you should **display(print out) all NAT mappings** on the screen whenever there is an update in the NAT table (e.g., a new entry is added or an existing entry is deleted). Each displayed mapping should show the four fields: original source address, original source port, translated source address, and translated source port. You are free to define the display format. And remove other debugging messages from your submission for us to check the NAT table during the demo.

3 Traffic Shaping

We use token bucket to control the transmission rate.

1. Token bucket. The token bucket holds a fixed number of logical tokens (bucket size). Tokens are generated and placed into the token bucket at a constant rate (fill rate), whose unit is the number of tokens generated per second (n/s). When the token bucket becomes full, subsequently generated

tokens are discarded. So there are two parameters to configure a token bucket: **bucket size** and **fill rate**. The number of tokens in a bucket is initialized as the bucket size. When a packet arrives, transmit it if there is a token. Otherwise, wait until getting an available token. It consumes a token to transmit a packet.

2. Threads. The NAT program leverages multi-threading, which has at least two threads:

   (a) A thread for receiving packets. It receives packet from the queue we specified (i.e., queue 0 as shown in Table 1) and then handles it, which triggers the callback function. In the callback function, your program requires to check whether there is available buffer in user space. If yes, buffer the packet in user space; if not, drop the packet. Here we define the maximum number of packets in buffer as 10.

   (b) A thread for processing packets. It conducts TCP translation for each buffered packet and sets verdict accordingly. Every transmission requires to get an available token from the token bucket.

4 Deliverables

Here are some possible testcases:

- Connections using "`nc`";
- Connections using "`nc -p xxx`", meaning that the source port is set to "`xxx`";
- Parallel TCP connections;
- Connecting to a closed port using TCP;
- Closing TCP connections from the VM;
- Closing TCP connections from the remote host, etc;
- Correctly displaying NAT mappings.

5 Submission

You are required to submit a set of C source codes that can be compiled into one executable file. The program should be successfully compiled without any warning message. We will provide the `iptables` script shown in Table 1 during the demo. You *must* submit a Makefile to generate an executable file **nat** correctly. Your program should run as the following:

`sudo ./nat <IP> <LAN> <MASK> <bucket size> <fill rate>`.

Your program should pass each testcase within three minutes during the demo. Otherwise, there is no marks for that testcase.

Both the executable file and the script will be executed with the root privilege. Be sure that your program does not carry any "*dangerous*" command inside. Please refer to the submission guidelines on our course homepage.

The deadline is **April 19 (Thur), 23:59:59.**

Have fun. :)