CSCI 4430 Tutorial
Project 2 (Part II)

Mi Zhang

mzhang@cse.cuhk.edu.hk

(Acknowledgement: Helen Chan)

Outline

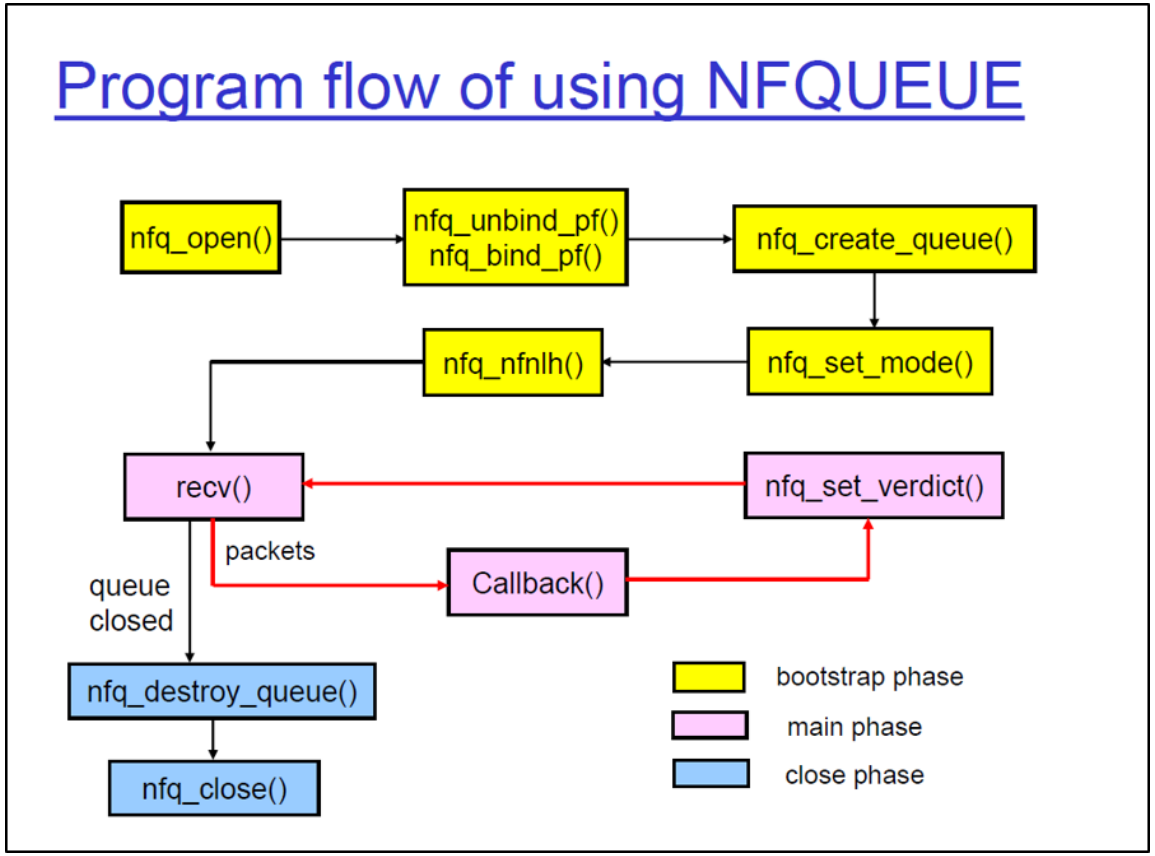
- Programming with NFQUEUE
  - Program flow with NFQUEUE
  - NFQUEUE Packet
- Header Structures
- Tips
  - NAT Table
  - Monitoring TCP Termination
  - Inbound vs. Outbound Traffic

Outline

- Programming with NFQUEUE
  - Program flow with NFQUEUE
  - NFQUEUE Packet
- Header Structures
- Tips
  - NAT Table
  - Monitoring TCP Termination
  - Inbound vs. Outbound Traffic

Program Flow with NFQUEUE

- Basic program flow



Program Flow with NFQUEUE

- Inside the `Callback()`,
  - Identify TCP packets
    - Make a decision (refer to spec. for detailed requirements):
      - Accept with translation
        - Recalculate checksums!
      - Accept without translation
      - Drop
  - Set the action: *`nfq_set_verdict()`*
    - e.g. `NF_ACCEPT`, `NF_DROP`

NFQUEUE Packet

- Fields of interest
  - NFQUEUE Packet Header

```
nfqnl_msg_packet_hdr *header;
header = nfq_get_msg_packet_hdr(pkt);
```

  - Packet ID
    - Necessary when calling `nfq_set_verdict()`

```
if (header != NULL) {
    id = ntohl(header->packet_id);
}
```

Outline

- Programming with NFQUEUE
  - Program flow with NFQUEUE
  - NFQUEUE Packet
- Header Structures
- Tips
  - NAT Table
  - Monitoring TCP Termination
  - Inbound vs. Outbound Traffic

Header Structures

- Definition: under "*/usr/include/netinet/*"
  - `ip.h`
  - `tcp.h`
- Inside the **payload** of an NFQUEUE packet

```
char* payload;
int data_len = nfq_get_payload(pkt, &payload);
```

  - `data_len` is useful for `nfq_set_verdict()` if you want to put new payload

Header Structures

- IP Header
  - Include header file:

```
#include <netinet/ip.h>
```

  - Access fields of interest:

```
struct iphdr *iph = (struct iphdr*) payload;

// source IP
iph->saddr;
// destination IP
iph->daddr;
// protocol
iph->protocol;
// checksum
iph->check;
```

Header Structures

- IP Header
  - Determine if the packet uses TCP:

```
#include <netinet/in.h>

if (iph->protocol == IPPROTO_TCP) {
    // TCP packets
} else {
    // Others, can be ignored
}
```

Header Structures

- TCP Header
  - Include header file

```
#include <netinet/tcp.h>
```

  - Access fields of interest:

```
struct tcphdr *tcph = (struct tcphdr *)
          (((char*) iph) + iph->ihl << 2);

// source port
tcph->source;
// destination port
tcph->dest;
// flags
tcph->syn; tcph->ack; tcph->fin; tcph->rst;
// checksum
tcph->check;
```

Header Structures

- Network Byte Ordering
  - Network-to-host: `ntohl()`, `ntohs()`
    - Read from header fields
  - Host-to-network: `htonl()`, `htons()`
    - Write to header fields

Outline

- Programming with NFQUEUE
  - Program flow with NFQUEUE
  - NFQUEUE Packet
- Header Structures
- Tips
  - NAT Table
  - Monitoring TCP Termination
  - Inbound vs. Outbound Traffic

NAT Table

- You can design and implement your own NAT (entries)
- Some suggested fields for the entry:
  - Translated port
  - Internal address pair, i.e. IP and port
  - TCP state

NAT Table

- Some desirable functions
  - Map an internal address pair {IP:port} to a translated port
  - Map a translated port to an internal address pair
  - Create a new translation entry, with an appropriate translated port assigned (smallest port number available)
  - Monitor the termination of TCP flow (by FIN or RST)
  - Expire TCP entries

NAT Table

- Printing the NAT table
  - **Four essential fields** for each mapping:
    - **Original source IP** address
    - Original source **port**
    - **Translated source IP** address
    - Translated source **port**
  - Print **ALL** mappings of the NAT table whenever the table is updated
    - No need to print the mappings if there is no update
    - We will only validate the printed table after a packet is processed

Monitoring TCP Termination

- Both the source and destination can initiate a 4-way handshake
- Two normal cases of termination:

- What about “connection reset”?
  - Deliver the RST packet and stop the translation, regardless of sequence number or acknowledgement number

Inbound vs. Outbound Traffic

- Always use the input parameters to obtain local network
  - `$ ./nat <public ip> <internal ip> <subnet mask>`
    - e.g., `$ ./nat 10.3.1.50 10.0.50.1 24`
- `inet_aton()`
  - Convert Internet host address from numbers-and-dots notation in IPv4 to binary from in network byte order
  - You may use `inet_pton()` as well

Inbound vs. Outbound Traffic

- Generate the subnet mask

```
int mask_int = atoi(subnet_mask);
unsigned int local_mask = 0xffffffff << (32 – mask_int)
```

- Determine whether source IP is in the internal network

```
if (ntohl(iph->saddr) & local_mask) == local_network) {
    // outbound traffic
} else {
    // inbound traffic
}
```

- How to generate *local_network* ?
  - Internal IP of a machine and the subnet mask

Today's Tutorial

- Programming with NFQUEUE
  - Program flow with NFQUEUE
  - NFQUEUE Packet
- Header Structures
- Tips
  - NAT Table
  - Monitoring TCP Termination
  - Inbound vs. Outbound Traffic

Important !!

- Project submission
  - **Deadline: Apr 22, 23:59:59**
  - **Always submit to the submission system**
    - **No submissions to tutors' emails will be accepted**
  - Project policy: refer to course homepage
- Remember to check your project carefully against the specification

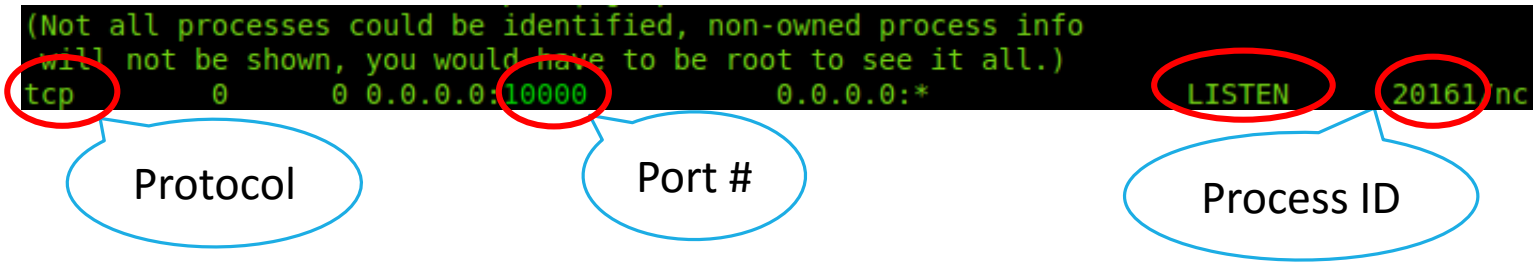
Extra

Testing your NAT program..?

- How to test whether your program works..?
  - Example client-server programs
  - Write up your own test programs
  - Use some Linux tools ☺
- Remember to setup the environment
  - Set **gateway** on VM B and VM C
  - Run the script for **iptables configuration** on VM A

Generate TCP Traffic

- On a department **Linux** workstation,
  - Listen to connections on a port:
    - `$ nc –v -l -p 10000`
    - You may further check if the port binding is there
      - `$ netstat -pnl | grep [port]`
      - e.g. `$ netstat -pnl | grep 10000`



Generate TCP Traffic

- On a department **Linux** workstation,
  - Find the workstation's IP,
    - `$ nslookup $HOST`
    - e.g. on linux7

```
linux7.cse.cuhk.edu.hk:/uac/gds/mzhang> nslookup linux7
Server:         137.189.91.188
Address:        137.189.91.188#53

Name:   linux7.cse.cuhk.edu.hk
Address: 137.189.88.150
```

- On VM A,
  - Start your NAT program, e.g.
    - `$ sudo ./nat 10.3.1 .[group_id]. 10.0.[group_id].1 24`

Generate TCP Traffic

- On VM B / VM C,
  - Connect to department machine,
    - `$ nc <department workstation IP> <port>`
- Send text from department workstation, or VM B / VM C
  - Type some words, press enter to send
  - See if the words show up on the other end

Others

- Check port assignment
  - Print the port assignment in your NAT program, or
  - Run Wireshark on **eth1 of VM A**, or
  - Run `tcpdump` in terminal on **eth1 of VM A**
    - Online Manual: http://linux.die.net/man/8/tcpdump
- Try multiple connections
  - Remember there are **more than one VM** available!
    - VM B, VM C
- Try possible cases in spec.