Packet Sniffing and Spoofing

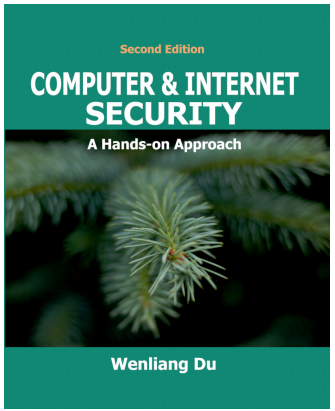
NT101 – NETWORK SECURITY

Giảng viên: Nghi Hoàng Khoa | khoanh@uit.edu.vn

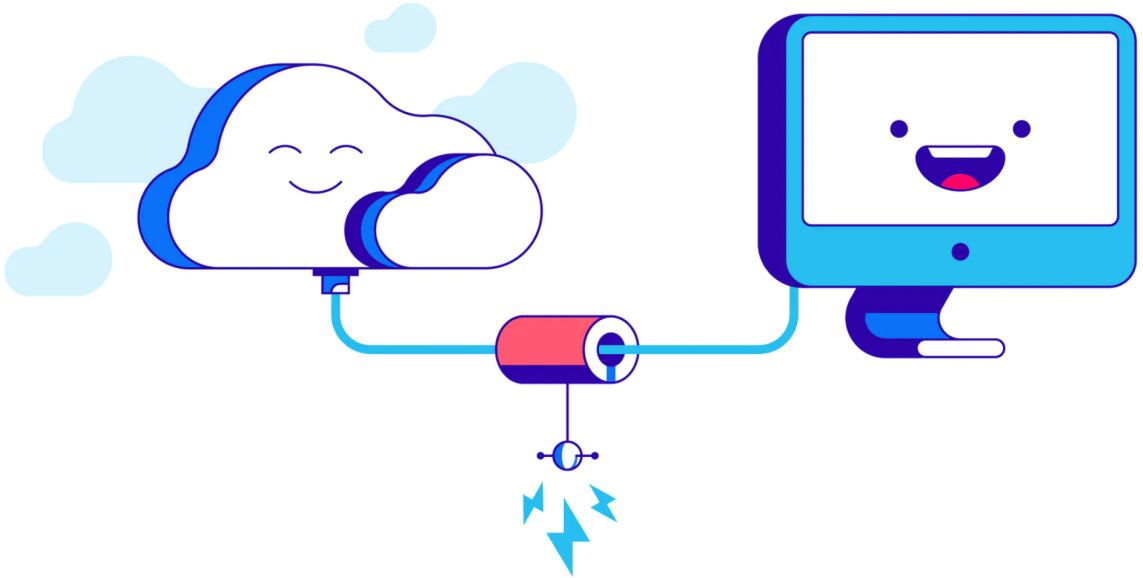
Where we are today…

- **Outline**
  - Sending/ Receiving packets
  - Sniffing packets
  - Spoofing packets
  - Scapy vs C
  - Byte order: Endianness
- **Reading:**
  - SEED book, Chapter 15

Acknowledgement:
Slides are adapted from Internet Security: A Hands-on approach (SEED book) 2nd Edition - 2019
Wenliang Du - Syracuse University

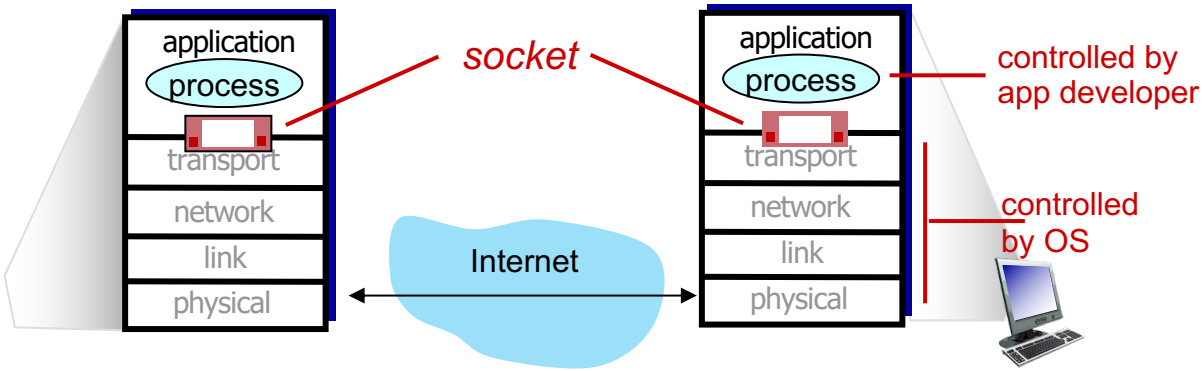


Gửi và nhận gói (packet)



Socket

- Quá trình gửi/nhận thông điệp đến/từ socket
- Socket tương tự như “door”
  - Quá trình “đẩy” thông điệp ra ngoài
  - Quá trình gửi dựa vào hạ tầng transport ở một phía khác của “door” để gửi thông điệp đến socket tại quá trình nhận
  - two sockets involved: one on each side



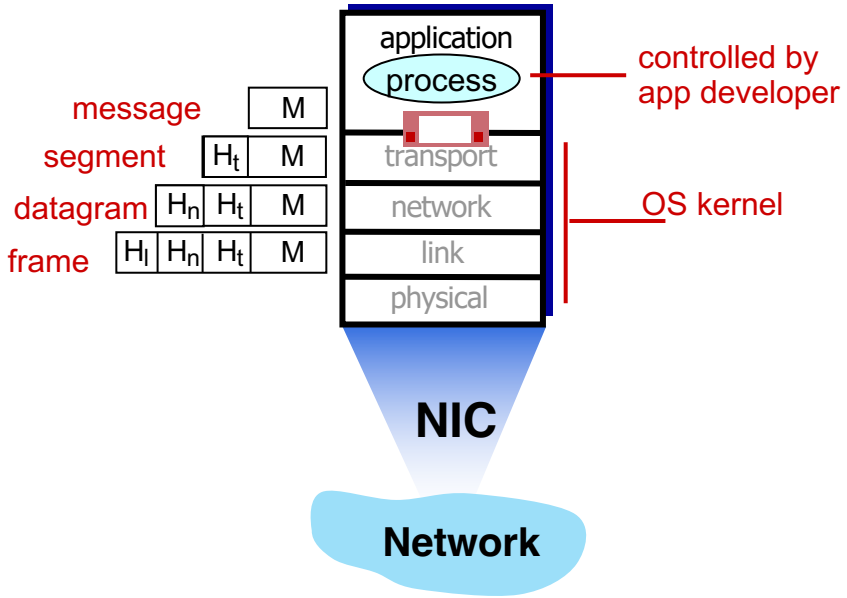
Socket APIs và làm thế nào để gửi packet?

- Ví dụ UDP Client

```python
#!/usr/bin/python3

import socket

IP = "127.0.0.1"
PORT = 8080
data =  b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

- Kết quả thực thi

```
$ nc -luv 8080
Listening on [0.0.0.0] (family 0, port 8080)
Hello World!
```



C++ version

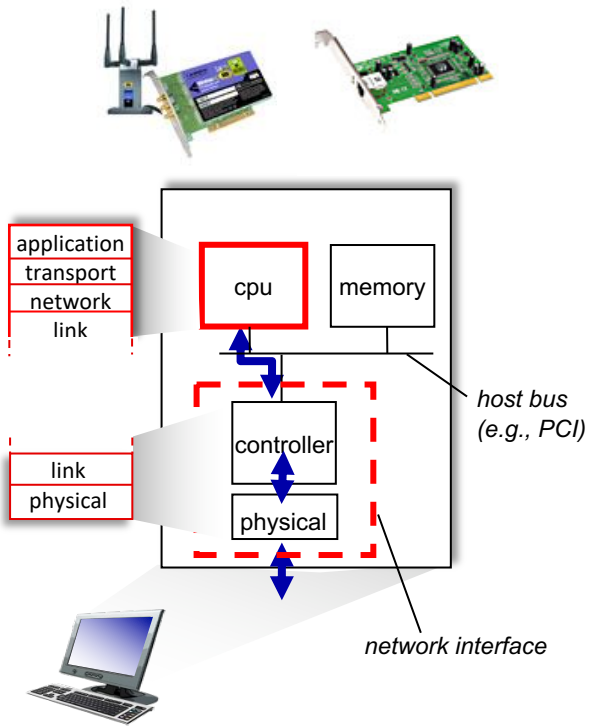
Socket APIs và làm thế nào để gửi packet?

```cpp
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>

int main ()
{
    struct sockaddr_in dest_info;
    const char* data = "Hello World! \n";

    // Create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Provide needed information about destination
    memset((char *) &dest_info, 0, sizeof(dest_info));
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = inet_addr("127.0.0.1");
    dest_info.sin_port = htons(8080);

    // Send the packet out
    sendto(sock, data, strlen(data), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));

    close(sock);
}
```

Python is built on top of C++

Làm thế nào để nhận được các gói?

- **NIC (Network Interface Card)** là thiết bị vật lý hoặc logical link giữa máy và mạng
- Mỗi NIC có một địa chỉ MAC
- Mọi NIC trên mạng sẽ “hear” tất cả các frame trên dây
- NIC kiểm ra địa chỉ đích cho mọi gói tin, nếu địa chỉ khớp với địa chỉ card MAC, nó sẽ tiếp tục được sao chép vào buffer của kernel.



Nhận Packet bằng Socket

```python
import socket

IP = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

while True:
    data, (ip, port) = sock.recvfrom(1024)
    print("Sender: {} and Port: {}".format(ip, port))
    print("Received message: {}", format(data))
```

```
$ python udp-server.py
Sender: 192.168.1.48 and Port: 56633
Received message: {} b'Hello UIT!\n'
Sender: 192.168.1.48 and Port: 56633
Received message: {} b'Welcome to NT101 course\n'
```

```
$ nc -u 192.168.1.48 9090
Hello UIT!
Welcome to NT101 course
```

Có cần thiết phải bind một số port cụ thể trong:

- Chương trình **Client**?
- Chương trình **Server**?

Nhận Packet bằng Socket

Create the socket

Provide information about server

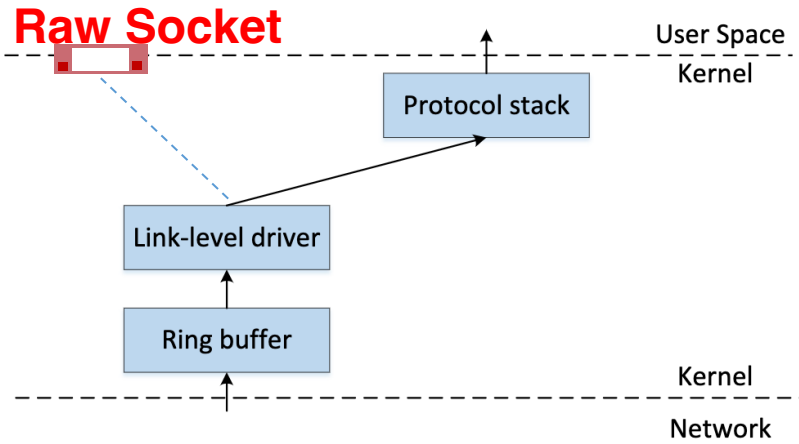
Receive packets

```
// Step ①
int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Step ②
memset((char *) &server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(9090);

if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
    error("ERROR on binding");

// Step ③
while (1) {
    bzero(buf, 1500);
    recvfrom(sock, buf, 1500-1, 0,
             (struct sockaddr *) &client, &clientlen);
    printf("%s\n", buf);
}
```

Packet Sniffing

Làm thế nào để lấy một bản sao của gói tin?

- Packet sniffing mô tả quá trình thu thập dữ liệu trực tiếp khi chúng truyền qua mạng



Nhận các gói tin bằng cách sử dụng Raw Socket

Creating a raw socket

Capture all types of packets

```
// Create the raw socket
int sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));   ①

// Turn on the promiscuous mode.
mr.mr_type = PACKET_MR_PROMISC;                              ②
setsockopt(sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr,     ③
                 sizeof(mr));

// Getting captured packets
while (1) {
    int data_size=recvfrom(sock, buffer, PACKET_LEN, 0,      ④
                &saddr, (socklen_t*)sizeof(saddr));
    if(data_size) printf("Got one packet\n");
}
```

Enable the promiscuous mode

Wait for packets

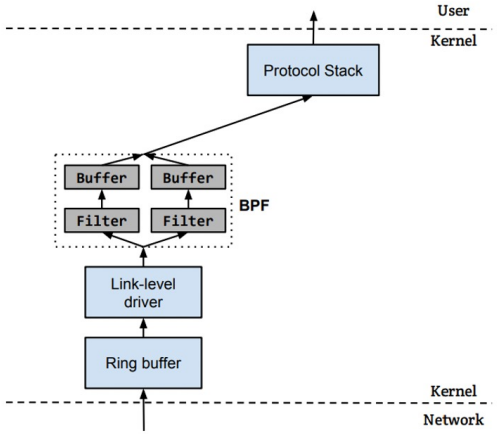
Promiscuous Mode

- Các frame không dành cho NIC đã biết sẽ bị loại bỏ
- Khi hệ thống trong chế độ promiscuous, NIC sẽ chuyển mọi frame nhận được từ mạng đến kernel.
- Nếu một chương trình được đăng ký sniffer với kernel, nó sẽ có thể thấy tất cả các gói tin
- Trong Wi-Fi, nó được gọi là Monitor Mode

BSD Packet Filter (BPF)

- BPF cho phép user-program đính kèm bột filter vào socket, filter này ra lệnh cho kernel loại bỏ các gói không mong muốn.
- Ví dụ về mã BPF đã biên dịch:

```
struct sock_filter code[] = {
  { 0x28,  0,  0, 0x0000000c }, { 0x15,  0,  8, 0x000086dd },
  { 0x30,  0,  0, 0x00000014 }, { 0x15,  2,  0, 0x00000084 },
  { 0x15,  1,  0, 0x00000006 }, { 0x15,  0, 17, 0x00000011 },
  { 0x28,  0,  0, 0x00000036 }, { 0x15, 14,  0, 0x00000016 },
  { 0x28,  0,  0, 0x00000038 }, { 0x15, 12, 13, 0x00000016 },
  { 0x15,  0, 12, 0x00000800 }, { 0x30,  0,  0, 0x00000017 },
  { 0x15,  2,  0, 0x00000084 }, { 0x15,  1,  0, 0x00000006 },
  { 0x15,  0,  8, 0x00000011 }, { 0x28,  0,  0, 0x00000014 },
  { 0x45,  6,  0, 0x00001fff }, { 0xb1,  0,  0, 0x0000000e },
  { 0x48,  0,  0, 0x0000000e }, { 0x15,  2,  0, 0x00000016 },
  { 0x48,  0,  0, 0x00000010 }, { 0x15,  0,  1, 0x00000016 },
  { 0x06,  0,  0, 0x0000ffff }, { 0x06,  0,  0, 0x00000000 },
};

struct sock_fprog bpf = {
   .len = ARRAY_SIZE(code),
   .filter = code,
};
```



(b) Packet flow with filter

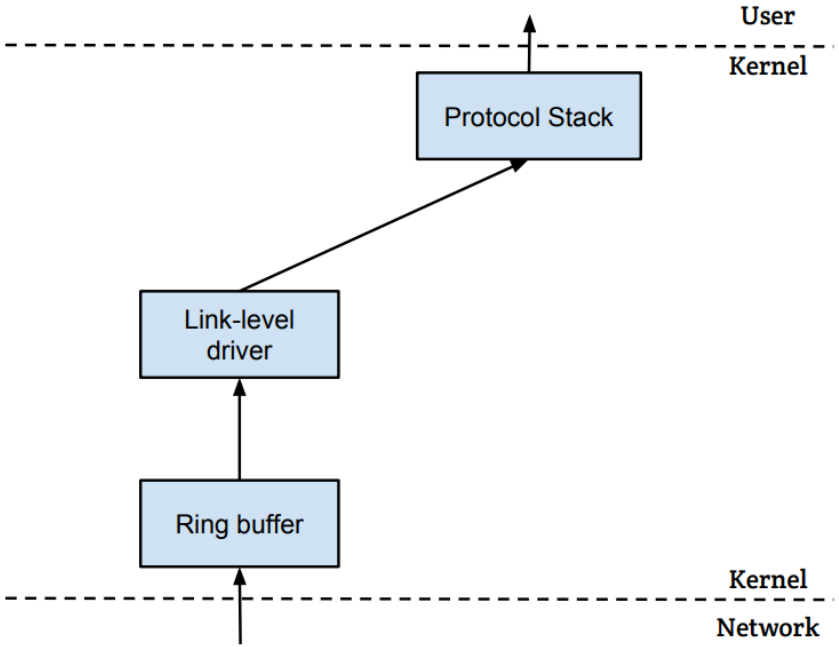
BSD Packet Filter (BPF)

- BPF pseudo-code đã biên dịch có thể được đính kèm vào một socket thông qua `setsockopt()`

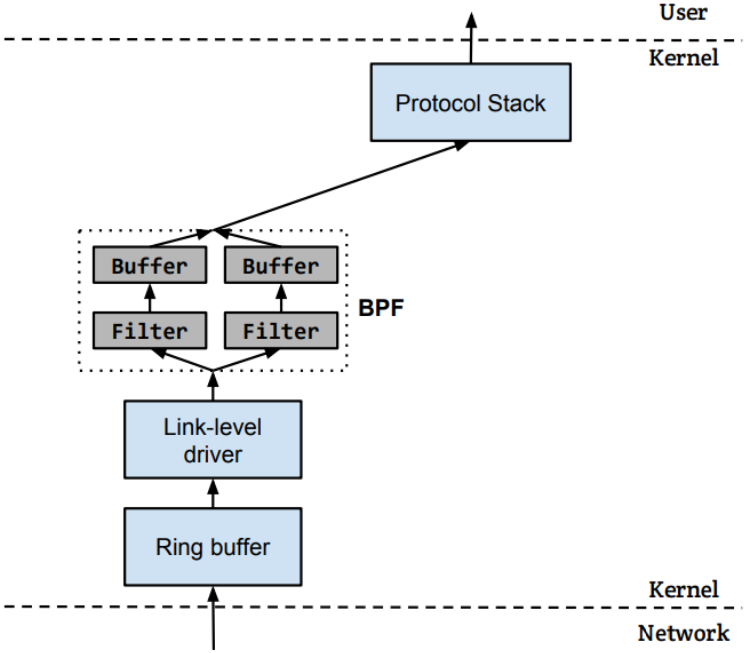
```
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf))
```

- Khi một gói tin được nhận bởi kernel, BPF sẽ được gọi
- Gói được chấp nhận sẽ được đẩy lên ngăn xếp giao thức (protocol stack)

Packet Flow With/Without Filters



(a) Packet flow



(b) Packet flow with filter

Giới hạn của phương pháp tiếp cận

- Chương trình này không “portable” trên các hệ điều hành khác nhau
- Thiết lập bộ lọc không dễ dàng
- Do đó, thư viện PCAP đã được tạo ra
  - Vẫn sử dụng các raw socket bên trong, nhưng API là tiêu chuẩn cho tất cả nền tảng. Thông tin về OS bị ẩn bởi triển khai API.
  - Cho phép lập trình viên chỉ định các rule filter bằng các lập ra các biểu thức Boolean có thể đọc.

PCAP: Packet Capture API

- Ban đầu xuất phát từ tcpdump
- Được hỗ trợ bởi nhiều nền tảng:
  - Linux: **libpcap**
  - Windows: **Winpcap** and **Npcap**
- Được viết bằng C. Trình wrappers triển khai trên ngôn ngữ khác
- Nền tảng cho nhiều công cụ:
  - Wireshark, tcpdump, Scapy, Nmap, Snort

Packet Sniffing Using the PCAP API

```
char filter_exp[] = "ip proto icmp";
```

Khởi tạo raw socket, thiết lập mạng ở promiscuous mode.

Filter

```
// Step 1: Open live pcap session on NIC with name eth3
handle = pcap_open_live("eth3", BUFSIZ, 1, 1000, errbuf);  ①

// Step 2: Compile filter_exp into BPF psuedo-code
pcap_compile(handle, &fp, filter_exp, 0, net);             ②
pcap_setfilter(handle, &fp);                               ③

// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);                   ④
```

Gọi chức năng này cho mọi gói tin được capture

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");
}
```

https://github.com/kevin-w-du/BookCode/tree/master/Sniffing_Spoofing/C_sniff

Packet Sniffing Using the PCAP API

```c
#include <pcap.h>
#include <stdio.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");

}

int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "ip proto icmp";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("eth0", BUFSIZ, 1, 1000, errbuf);

  // Step 2: Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);
  pcap_setfilter(handle, &fp);

  // Step 3: Capture packets
  pcap_loop(handle, -1, got_packet, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}
```

```
$ apt install libpcap-dev
$ gcc 05-sniff-pcap.cpp -o 05-pcap-sniff -lpcap
$ sudo ./05-pcap-sniff
Got a packet
Got a packet
Got a packet
Got a packet
```

```
# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.115 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.107 ms
```

Processing Captured Packet

```
/* Ethernet header */
struct ethheader {
  u_char  ether_dhost[ETHER_ADDR_LEN]; /* destination host address */
  u_char  ether_shost[ETHER_ADDR_LEN]; /* source host address */
  u_short ether_type;                  /* IP? ARP? RARP? etc */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;
  if (ntohs(eth->ether_type) == 0x0800) { ... } // IP packet
  ...
}
```

The **packet** argument contains a copy of the packet, including the Ethernet header. We typecast it to the Ethernet header structure.

Now we can access the field of the structure

Processing Captured Packet

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;

  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *)
                           (packet + sizeof(struct ethheader)); ①

    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));   ②
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));     ③

    /* determine protocol */
    switch(ip->iph_protocol) {                                  ④
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");
            return;
        case IPPROTO_UDP:
            printf("   Protocol: UDP\n");
            return;
```

Find where the IP header starts and typecast it to the IP Header structure.

Now we can easily access the fields in the IP header.

Processing Captured Packet

```
$ sudo ./06-sniff-improve
       From: 172.17.0.2
         To: 172.17.0.3
   Protocol: ICMP
       From: 172.17.0.3
         To: 172.17.0.2
   Protocol: ICMP
       From: 172.17.0.2
         To: 172.17.0.3
   Protocol: ICMP
       From: 172.17.0.3
         To: 172.17.0.2
   Protocol: ICMP
       From: 172.17.0.2
         To: 172.17.0.3
   Protocol: ICMP
       From: 172.17.0.3
         To: 172.17.0.2
   Protocol: ICMP
       From: 172.17.0.2
         To: 172.17.0.3
   Protocol: ICMP
       From: 172.17.0.3
         To: 172.17.0.2
   Protocol: ICMP
```

Host A: 172.17.0.3

```
# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.099 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.115 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.107 ms
```

Host B: 172.17.0.2

Task: Tự thực hiện lại các ví dụ này

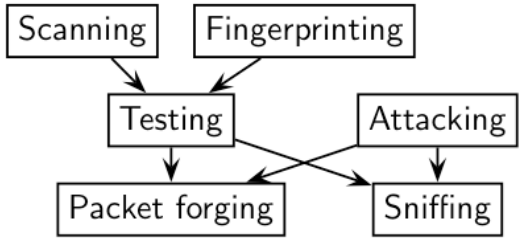
Further Processing Captured Packet

- Nếu muốn xử lý thêm gói tin, chẳng hạn như in ra header của TCP, UDP và ICMP, chúng ta có thể sử dụng kỹ thuật tương tự.
  - Di chuyển con trỏ đến đầu header tiếp theo và type-cast
  - Cần sử dụng trường độ dài header trong IP header để tính toán kích thước thực của IP header
- Trong ví dụ sau, nếu biết header tiếp theo là ICMP, có thể lấy một con trỏ đến phần ICMP bằng cách thực hiện như sau:

```
int ip_header_len = ip->iph_ihl * 4;
u_char *icmp = (struct icmpheader *)
               (packet + sizeof(struct ethheader) + ip_header_len);
```

Sniffing using Scapy

- Scapy là gì?
  - Mô-đun (hoặc chương trình) mạnh mẽ để thao tác packet
    - Packet parsing
    - Packet sending and receiving
    - Packet sniffing and spoofing
    - Adding new protocols
    - Many more applications
- Cài đặt:
  - **`sudo pip3 install scapy`**
  - Import Scapy module in Python program
  - **`from scapy.all import *`**



Packing Sniffing Using Scapy (cont.)

```python
#!/usr/bin/python3
from scapy.all import *

print("SNIFFING PACKETS.........")

def print_pkt(pkt):
    print("Source IP:", pkt[IP].src)
    print("Destination IP:", pkt[IP].dst)
    print("Protocol:", pkt[IP].proto)
    print("\n")

pkt = sniff(filter='icmp',prn=print_pkt)
```

```
$ sudo python 07-scapy-sniff.py
SNIFFING PACKETS.........
Source IP: 192.168.1.48
Destination IP: 8.8.8.8
Protocol: 1


Source IP: 8.8.8.8
Destination IP: 192.168.1.48
Protocol: 1


Source IP: 192.168.1.48
Destination IP: 8.8.8.8
Protocol: 1


Source IP: 8.8.8.8
Destination IP: 192.168.1.48
Protocol: 1


Source IP: 192.168.1.48
Destination IP: 8.8.8.8
Protocol: 1


Source IP: 8.8.8.8
Destination IP: 192.168.1.48
Protocol: 1
```

```
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=115 time=30.297 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=30.030 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=46.680 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=29.712 ms
```

Example: Packet summary

Packing Sniffing Using Scapy (cont.)

```
#!/usr/bin/python3

from scapy.all import *

pkt = sniff(iface='en0', filter='icmp or udp',count=10)
pkt.summary()
```

```
$ sudo python 08-scapy-sniff.py
Ether / IP / ICMP 192.168.1.48 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.1.48 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.1.48 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.1.48 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.1.48 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.1.48 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.1.48 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.1.48 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.1.48 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.1.48 echo-reply 0 / Raw
```

```
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=115 time=30.297 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=30.030 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=46.680 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=29.712 ms
```

Packing Sniffing Using Scapy (cont.)

```python
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    hexdump(pkt)
    pkt.show()
    print("---------------------------")

f = "udp and dst portrange 50-55 or icmp"
sniff(iface = 'en0', filter = f, prn=process_packet)
```

```
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=115 time=30.297 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=30.030 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=46.680 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=29.712 ms
```

```
$ sudo python 09-scapy-sniff-callback.py
0000  8C 85 90 7A 6A E0 70 D9 31 E7 EE A8 08 00 45 88  ...zj.p.1.....E.
0010  00 54 00 00 00 00 73 01 75 39 08 08 08 08 C0 A8  .T....s.u9......
0020  01 30 00 00 DB 84 2B CE 00 03 5F 8C 5F 05 00 05  .0....+..._._...
0030  4F 10 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15  O...............
0040  16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25  .......... !"#$%
0050  26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37                                           67
###[ Ethernet ]###
  dst       = 8c:85:90:7a:6a:e0
  src       = 70:d9:31:e7:ee:a8
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x88
     len       = 84
     id        = 0
     flags     =
     frag      = 0
     ttl       = 115
     proto     = icmp
     chksum    = 0x7539
     src       = 8.8.8.8
     dst       = 192.168.1.48
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xdb84
        id        = 0x2bce
        seq       = 0x3
###[ Raw ]###
           load      =....
```

Get Information of Protocol classes

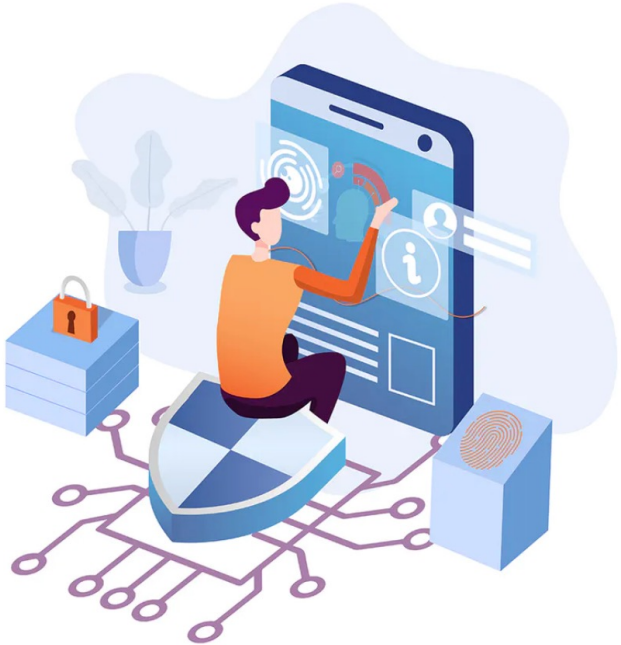
Get Attribute names

```
>>> ls(IP)
version    : BitField  (4 bits)           = (4)
ihl        : BitField  (4 bits)           = (None)
tos        : XByteField                   = (0)
len        : ShortField                   = (None)
id         : ShortField                   = (1)
flags      : FlagsField  (3 bits)         = (<Flag 0 ()>)
frag       : BitField  (13 bits)          = (0)
ttl        : ByteField                    = (64)
proto      : ByteEnumField                = (0)
chksum     : XShortField                  = (None)
src        : SourceIPField                = (None)
dst        : DestIPField                  = (None)
options    : PacketListField              = ([])
```

Get Method names

```
>>> help(IP)
Help on class IP in module scapy.layers.inet:


class IP(scapy.packet.Packet, IPTools)
 |  IP(*args, **kargs)
 |
 |  Add more powers to a class with an "src" attribute.
 |
 |  Method resolution order:
 |      IP
 |      scapy.packet.Packet
 |      scapy.base_classes.BasePacket
 |      scapy.base_classes.Gen
 |      scapy.base_classes._CanvasDumpExtended
 |      IPTools
 |      builtins.object
```

Packet Spoofing



Packet Spoofing

- Khi một số thông tin quan trọng trong gói tin bị giả mạo, ta coi đó là hành vi giả mạo
- Nhiều cuộc tấn công mạng dựa trên việc giả mạo gói tin gói tin.
- **Hãy xem lại cách gửi gói tin mà không bị giả mạo**

Sending Packets Without Spoofing

Python version

```python
#!/usr/bin/python3

import socket

IP = "172.17.0.2"
PORT = 9090
data =  b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

C++ version

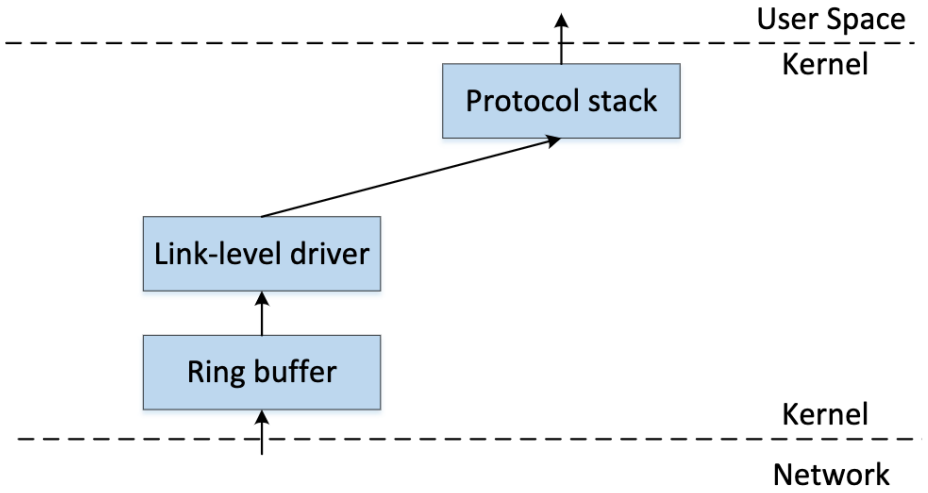
```cpp
int main ()
{
    struct sockaddr_in dest_info;
    const char* data = "Hello World! \n";

    // Step 1: Create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Step 2: Provide needed information about destination
    memset((char *) &dest_info, 0, sizeof(dest_info));
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = inet_addr("172.17.0.2");
    dest_info.sin_port = htons(9090);

    // Step 3: Send the packet out
    sendto(sock, data, strlen(data), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));

    close(sock);
}
```

Testing: Sử dụng lệnh netcat (nc) để chạy máy chủ UDP trên 172.17.0.2. Sau đó, chạy các chương trình trên từ một máy khác. Có thể thấy rằng thông điệp đã được gửi đến máy chủ:

```
$ nc -luv 9090
Hello, World!
```

Spoofing Packets Using Raw Sockets

- Có hai bước chính trong quá trình giả mạo gói tin:
  - Xây dựng gói tin
  - Gửi gói tin ra ngoài



Spoofing Packets Using Raw Sockets

```
/*************************************************************
  Given an IP packet, send it out using a raw socket.
*************************************************************/
void send_raw_ip_packet(struct ipheader* ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
                     &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

We use *setsockopt()* to enable *IP_HDRINCL* on the socket. (include IP header)

For raw socket programming, since the destination information is already included in the provided IP header, we do not need to fill all the fields

Since the socket type is raw socket, the system will send out the IP packet as is.

Spoofing Packets: Constructing the Packet

```
char buffer[1500];

memset(buffer, 0, 1500);

/*****************************************************
   Step 1: Fill in the ICMP header.
 *****************************************************/
struct icmpheader *icmp = (struct icmpheader *)
                          (buffer + sizeof(struct ipheader));
icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.

// Calculate the checksum for integrity
icmp->icmp_chksum = 0;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp,
                             sizeof(struct icmpheader));
```

Find the starting point of the ICMP header, and typecast it to the ICMP structure

Fill in the ICMP header fields

Spoofing Packets: Constructing the Packet

```
/*****************************************************
   Step 2: Fill in the IP header.
 *****************************************************/
struct ipheader *ip = (struct ipheader *) buffer;
ip->iph_ver = 4;
ip->iph_ihl = 5;
ip->iph_ttl = 20;
ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
ip->iph_destip.s_addr = inet_addr("10.0.2.5");
ip->iph_protocol = IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct icmpheader));
```

Typecast the buffer to the IP structure

Fill in the IP header fields

Finally, send out the packet

```
send_raw_ip_packet (ip);
```

Spoofing UDP Packets (cont.)

- Việc xây dựng các gói UDP cũng tương tự, ngoại trừ việc chúng ta cần include dữ liệu payload.

```
memset(buffer, 0, 1500);
struct ipheader *ip = (struct ipheader *) buffer;
struct udpheader *udp = (struct udpheader *) (buffer +
                                        sizeof(struct ipheader));

/*********************************************************
   Step 1: Fill in the UDP data field.
 *********************************************************/
char *data = buffer + sizeof(struct ipheader) +
                      sizeof(struct udpheader);
const char *msg = "Hello Server!\n";
int data_len = strlen(msg);
strncpy (data, msg, data_len);

/*********************************************************
   Step 2: Fill in the UDP header.
 *********************************************************/
udp->udp_sport = htons(12345);
udp->udp_dport = htons(9090);
udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
udp->udp_sum =  0; /* Many OSes ignore this field, so we do not
                      calculate it. */
```

Spoofing UDP Packets (cont.)

```
/*************************************************************
   Step 3: Fill in the IP header.
 *************************************************************/

...... /* Code omitted here; same as that in Listing 12.6 */
ip->iph_protocol = IPPROTO_UDP; // The value is 17.
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct udpheader) + data_len);
```

Testing: Sử dụng lệnh nc để chạy máy chủ UDP trên 10.0.2.5. Sau đó, ta giả mạo một gói UDP từ một máy khác. Ta có thể thấy rằng gói UDP giả mạo đã được nhận bởi máy chủ.

```
seed@Server(10.0.2.5):$ nc -luv 9090
Connection from 1.2.3.4 port 9090 [udp/*] accepted
Hello Server!
```

Spoofing ICMP & UDP Using Scapy

Constructing Packets

```
>>> a = IP(src='1.2.3.4', dst='10.20.30.40')
>>> b = UDP(sport = 1234, dport = 1020)
>>> c = 'Hello World'
>>> pkt = a/b/c

>>> pkt.show()
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.20.30.40
  \options   \
###[ UDP ]###
     sport     = 1234
     dport     = 1020
     len       = None
     chksum    = None
###[ Raw ]###
        load      = 'Hello World'
```

Layer stacking

```
>>> a = IP()/UDP()/Raw("hello")
>>> a
<IP  frag=0 proto=udp |<UDP  |<Raw  load='hello' |>>>
```

Operator Overloading

```
def __div__(self, other):
    if isinstance(other, Packet):
        cloneA = self.copy()
        cloneB = other.copy()
        cloneA.add_payload(cloneB)
        return cloneA
    elif isinstance(other, (bytes, str)):
        return self / conf.raw_layer(load=other)
    else:
        return other.__rdiv__(self)
__truediv__ = __div__
```

Spoofing ICMP & UDP Using Scapy

```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src="1.2.3.4", dst="93.184.216.34")   ①
icmp = ICMP()                                 ②
pkt = ip/icmp                                 ③
pkt.show()
send(pkt,verbose=0)                           ④
```

ICMP Spoofing

```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED UDP PACKET.........")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)       # UDP Layer
data = "Hello UDP!\n"                   # Payload
pkt = ip/udp/data      # Construct the complete packet
pkt.show()
send(pkt,verbose=0)
```

UDP Spoofing

Sniffing and Then Spoofing

- Trong nhiều tình huống, trước tiên ta cần capture các gói, sau đó giả mạo phản hồi dựa trên các gói đã capture được
- Procedure (using UDP as example):
  - Sử dụng PCAP API để capture các gói quan tâm
  - Tạo một bản sao từ gói đã capture
  - Thay thế trường data UDP bằng một thông điệp mới và hoán đổi trường nguồn và trường đích
  - Gửi phản hồi giả mạo

Sniffing and spoofing UDP Packet (cont.)

```
void spoof_reply(struct ipheader* ip)
{
    const char buffer[1500];
    int ip_header_len = ip->iph_ihl * 4;
    struct udpheader* udp = (struct udpheader *) ((u_char *)ip +
                                                  ip_header_len);
   if (ntohs(udp->udp_dport) != 9999) {
       // Only spoof UDP packet with destination port 9999
       return;
   }

   // Step 1: Make a copy from the original packet
   memset((char*)buffer, 0, 1500);
   memcpy((char*)buffer, ip, ntohs(ip->iph_len));
   struct ipheader  * newip  = (struct ipheader *) buffer;
   struct udpheader * newudp = (struct udpheader *) (buffer +
   ip_header_len);
   char *data = (char *)newudp + sizeof(struct udpheader);
```

Sniffing and spoofing UDP Packet (cont.)

```
    // Step 2: Construct the UDP payload, keep track of payload size
    const char *msg = "This is a spoofed reply!\n";
    int data_len = strlen(msg);
    strncpy (data, msg, data_len);

    // Step 3: Construct the UDP Header
    newudp->udp_sport = udp->udp_dport;
    newudp->udp_dport = udp->udp_sport;
    newudp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    newudp->udp_sum =  0;

    // Step 4: Construct the IP header (no change for other fields)
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 50; // Rest the TTL field
    newip->iph_len = htons(sizeof(struct ipheader) +
                           sizeof(struct udpheader) + data_len);

    // Step 5: Send out the spoofed IP packet
    send_raw_ip_packet(newip);
}
```

Sniffing and Then Spoofing Using Scapy (cont.)

```
#!/usr/bin/python3
from scapy.all import *

def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data

     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)

     send(newpkt,verbose=0)

pkt = sniff(filter='icmp and src host 192.168.1.48',prn=spoof_pkt)
```

Sniffing and Then Spoofing Using Scapy

```
$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4): 56 data bytes
64 bytes from 1.2.3.4: icmp_seq=0 ttl=64 time=3.627 ms
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=4.144 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=4.080 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=4.511 ms
```

```
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=64 time=5.522 ms
64 bytes from 8.8.8.8: icmp_seq=0 ttl=115 time=29.351 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=3.867 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=30.108 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=3.545 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=102.155 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=4.234 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=29.132 ms (DUP!)
```

```
$ sudo python 10-scapy-sniff-spoof.py
Original Packet.........
Source IP :  192.168.1.48
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 192.168.1.48
Original Packet.........
Source IP :  192.168.1.48
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 192.168.1.48
```

```
$ sudo python 10-scapy-sniff-spoof.py
Original Packet.........
Source IP :  192.168.1.48
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 192.168.1.48
Original Packet.........
Source IP :  192.168.1.48
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 192.168.1.48
```

Packet Spoofing: Scapy v.s C

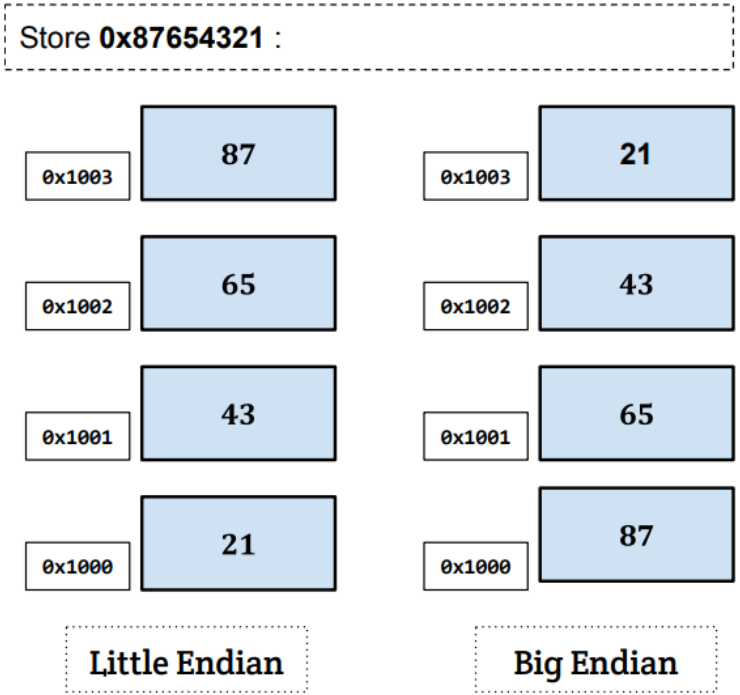
- **Python + Scapy**
  - Pros: xây dựng các gói ton rất đơn giản
  - Cons: chậm hơn nhiều so với mã C
- **C Program (using raw socket)**
  - Pros: nhanh hơn nhiều
  - Cons: xây dựng các gói tin rất phức tạp
- **Hybrid Approach**
  - Sử dụng Scapy để xây dựng gói tin
  - Sử dụng C để sửa đổi một chút các gói tin và sau đó gửi các gói tin

Example: Kaminsky DNS attack *(will be discussed)*

Endianness

Endianness

- Endianness: một thuật ngữ đề cập đến thứ tự mà một mục dữ liệu multi-byte nhất định được lưu trữ trong bộ nhớ.
  - Little Endian: lưu trữ byte dữ liệu quan trọng nhất ở địa chỉ cao nhất
  - Big Endian: lưu trữ byte dữ liệu quan trọng nhất ở địa chỉ thấp nhất



Endianness trong truyền thông mạng

- Các máy tính có byte order khác nhau sẽ “misunderstand” lẫn nhau.
  - **Solution:** đồng ý về một order chung để giao tiếp
  - Nó được gọi “network order”, giống như big endian order
- Tất cả các máy tính cần chuyển đổi dữ liệu giữa “host order” và “network order”

Macro	Description
`htons()`	Convert unsigned short integer from host order to network order.
`htonl()`	Convert unsigned integer from host order to network order.
`ntohs()`	Convert unsigned short integer from network order to host order.
`ntohl()`	Convert unsigned integer from network order to host order.

Summary

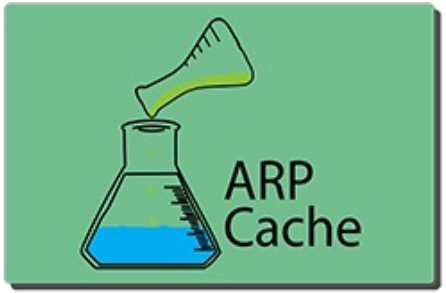
- Socket programming
- Packet sniffing
  - Using raw socket
  - Using PCAP APIs
- Packet spoofing
  - Using C++
  - Using Scapy
- Sniffing and the spoofing
- Byte order: Endianness

Bài tập về nhà

- **Packet Sniffing and Spoofing Lab:** (https://seedsecuritylabs.org/Labs_20.04/Networking/Sniffing_Spoofing/)
  - Bài tập này có 2 mục tiêu:
    - học cách sử dụng các công cụ và hiểu các công nghệ bên dưới các công cụ này.
    - sinh viên sẽ viết các chương trình sniffer và spoofing đơn giản, đồng thời hiểu sâu về các khía cạnh kỹ thuật của các chương trình này
- **Làm việc theo nhóm (nhóm đồ án) hoặc cá nhân.**

Buổi kế tiếp

- Chuẩn bị
  - Chủ đề dự kiến: **Link (MAC) Layer and atttacks**
  - Tài liệu:
    - **SEED book, Chapter 16**
      - Refs: https://www.handsonsecurity.net/resources.html
    - **SEED Lab: ARP Cache Poisoning Attack Lab**
      - Refs: **https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/**



Hôm nay, **kết thúc!**

- Nghi Hoàng Khoa
- khoanh@uit.edu.vn
- www.inseclab.uit.edu.vn

- NT101 – An toàn Mạng máy tính