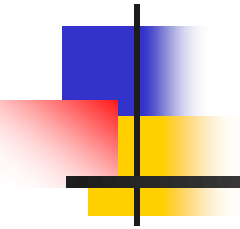
Packet Sniffing and Spoofing



Lecture 03

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

Processing Captured Packet

- *got_packet*(…) is invoked once a packet is captured
  - in *demonstration*: *print out a simple msg*
  - in *real world*: ***process*** packet, or even ***react to*** packet
    - e.g., capturing a *DNS request packet* and sending out a ***spoofed reply*** based on the content of packet
- *got_packet*(…)

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
```

  - *args:* a user-defined argument passed to the callback
    - NULL or point to some user data
  - *header*: a pointer to a *pcap_pkthdr* structure, which contains metadata about the captured packet
    - e.g., timestamp and length

Processing Captured Packet

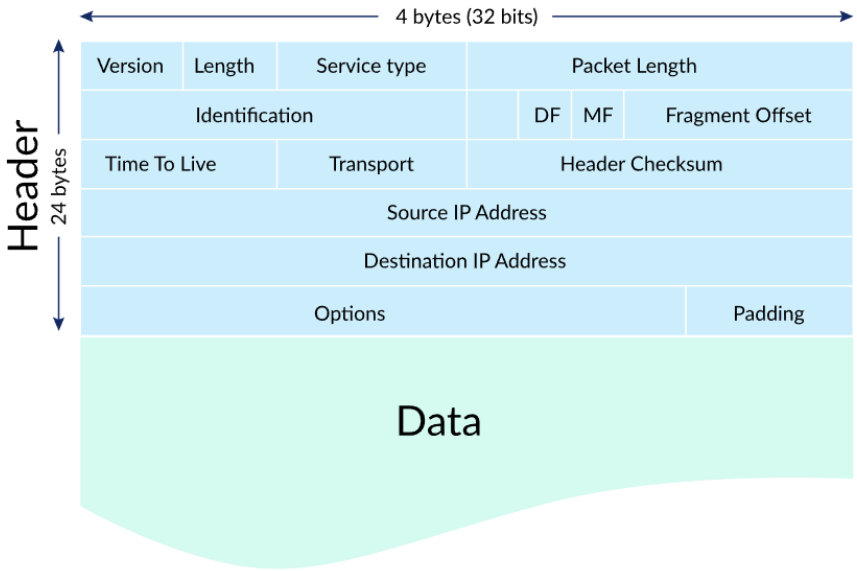
- *got_packet*(…) is invoked once a packet is captured
  - in *demonstration*: *print out a simple msg*
  - in *real world*: ***process*** packet, or even ***react to*** packet
    - e.g., capturing a *DNS request packet* and sending out a ***spoofed reply*** based on the content of packet
- *got_packet*(…)

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
```

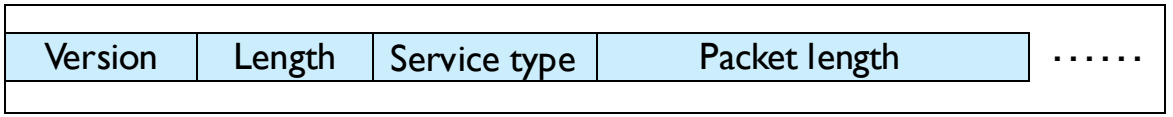
  - *u_char *packet*: a pointer points to the ***buffer*** that holds the ***packet***
    - *u_char* (*unsigned char*) indicates the contents of the buffer are a *sequence of characters* (have structures internally)
      - including an *ethernet frame* ethernet header placed at the beginning

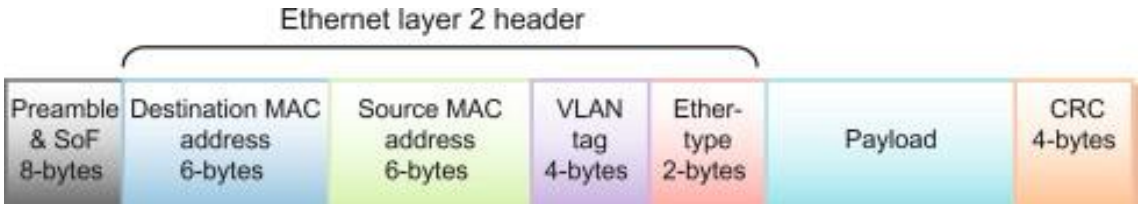
Processing Captured Packet

- How we visualize IP packet



- How computer stores packet





Processing Captured Packet (cont.)

inconvenient & not scalable

- Checking whether the type field (Ethertype) of ethernet header is IP or something: find the *offset* of type field and read its value
- Efficient idea: *struct:* a group of variables stored in contiguous memory
  1. case a buffer to a *struct* to treat the buffer as a *structure*;
  2. access its data using the *structure*'s field names

```
/* 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
  ...
}
```

IPv4

packet argument contains a copy of the packet, including the Ethernet header

- typecast it to the Ethernet header structure

access the field of the structure

Processing Captured Packet (cont.)

- More to do…: print out some info. from the IP header

```
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;
```

typecast to IP header structure

IP header structure offset

access IP header structure

- src. IP addr.
- des. IP addr.

find *where* the *IP header* starts and typecast it to the IP header structure

* *distance*: the size of ethernet header

access the fields in the IP header

Processing Captured Packet (cont.)

- Compile the program

  $ gcc –o sniff_improved sniff_improved.c –lpcap

- Run the program

  $ sudo ./sniff_improved

Packet Spoofing

- Typical socket programming:
  - we have controls over a few selected fields in the header
    - des. IP addr. (not src. IP addr.)
      - when the packet is sent out, the OS will put corresponding IP addr. in the src. IP field
- In network attacks, packets are constructed with *bogus*, *unrealistic*, and *targeted info.* in the headers.
  - TCP SYN flooding attack: src. IP addr. is randomly generated
  - TCP session hijacking attack:
    - use other people's IP addr
    - set correct sequence and port #s
  - sending packets like those is called ***packet spoofing***
    - critical info. in the packet is ***forged***

Packet Spoofing (cont.)

- Writing your own packet spoofing tool in C language can give you an idea how these tools are built

- Tools for spoofing packets: Netwox Scapy
- Scapy
  - packet manipulation tool
  - forge or decode packets of a wide number of protocols
  - send packets on the wire, capture them, match requests and replies

Sending Normal Packet Using Socket

- Normally sending out packets requires three steps for UDP client program:
  1. create a socket
  2. provide des. info., e.g., des. IP addr. and des. UDP port #
  3. call *sendto()* to send out a UDP packet with payload
     - The OS will construct the actual UDP packet based on the info provided

Sending Normal Packet Using Socket (cont.)

- Normally sending out packets requires three steps:

```
void main()
{
    struct sockaddr_in dest_info;
    char *data = "UDP message\n";

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

    // Step 2: Provide 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("10.0.2.5");
    dest_info.sin_port = htons(9090);

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

Testing:

- Use the netcat (nc) command to run a UDP server on 10.0.2.5.
- Run the program on another machine.

Output:

- The message has been delivered to the server machine

(port #)

```
seed@Server(10.0.2.5):$ nc -luv 9090
Connection from 10.0.2.6 port 9090 [udp/*] accepted
UDP message
```