Digital Forensics
Network Forensics
Lecture 1
Network Sniffing

AKBAR S. NAMIN

SPRING 2018

Outline

- What is Network Sniffing ?
- Raw Socket Sniffer
- libpcap sniffer
- Decoding the Layers

Switched vs. Un-switched Network

Un-switched Network

Ethernet packets **pass through every device on the network**, expecting each system device to only look at the packets that are addressed to it.

Switched Network

Packets are **only sent to the port they are destined for**, according to their destination MAC address.

Promiscuous mode

Promiscuous mode

A special mode of Ethernet hardware in which network device **look at all packets**, regardless of the destination address.

- It is fairly trivial to set a device to the Promiscuous mode and this can be used for network sniffing.

Enable the Promiscuous Mode

- Packet-capturing programs, such as tcpdump, drop the device they are listening to into Promiscuous mode by default.
- Promiscuous mode can be set using *ifconfig* command.

```
# Set interface eth0 to promiscuous mode
$ ifconfig eth0 promisc
```

Network Sniffing

The Act of capturing packets that are not necessarily meant for public viewing is called **network sniffing**.

- Sniffing packets is **promiscuous mode** in **unswitched network** can turn up useful information.

Example: tcpdump Program Output

- tcpdump is a packet analyzer program that allows the user to display TCP/IP and other packets being transmitted over a network to which the computer is attached.
- User “leech” is seen (in the data portion of transmitted packet) logging into an FTP server using the password “18&nite”.
- Data transmitted by **TELNET**, **FTP** and **POP3** are unencrypted.

```
                                                          ...x
21:27:52.406177 192.168.0.193.32778 > 192.168.0.118.ftp: P 1:13(12) ack 42 win
5840 <nop,nop,timestamp 921434 466808> (DF) [tos 0x10]
0x0000   4510 0040 9670 4000 4006 21b0 c0a8 00c1        E..@.p@.@.!.....
0x0010   c0a8 0076 800a 0015 5ed4 9ce8 292e 8a9c        ...v....^...)...
0x0020   8018 16d0 edd9 0000 0101 080a 000e 0f5a        ...............Z
0x0030   0007 1f78 5553 4552 206c 6565 6368 0d0a        ...xUSER.leech..
21:27:52.415487 192.168.0.118.ftp > 192.168.0.193.32778: P 42:76(34) ack 13
win 17304 <nop,nop,timestamp 466885 921434> (DF)
0x0000   4500 0056 e0ac 4000 8006 976d c0a8 0076        E..V..@....m...v
0x0010   c0a8 00c1 0015 800a 292e 8a9c 5ed4 9cf4        ........)...^...
0x0020   8018 4398 4e2c 0000 0101 080a 0007 1fc5        ..C.N,..........
0x0030   000e 0f5a 3333 3120 5061 7373 776f 7264        ...Z331.Password
0x0040   2072 6571 7569 7265 6420 666f 7220 6c65        .required.for.le
0x0050   6563                                           ec
21:27:52.415832 192.168.0.193.32778 > 192.168.0.118.ftp: . ack 76 win 5840
<nop,nop,timestamp 921435 466885> (DF) [tos 0x10]
0x0000   4510 0034 9671 4000 4006 21bb c0a8 00c1        E..4.q@.@.!.....
0x0010   c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe        ...v....^...)...
0x0020   8010 16d0 7e5b 0000 0101 080a 000e 0f5b        ....~[.........[
0x0030   0007 1fc5                                      ....
21:27:56.155458 192.168.0.193.32778 > 192.168.0.118.ftp: P 13:27(14) ack 76
win 5840 <nop,nop,timestamp 921809 466885> (DF) [tos 0x10]
0x0000   4510 0042 9672 4000 4006 21ac c0a8 00c1        E..B.r@.@.!.....
0x0010   c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe        ...v....^...)...
0x0020   8018 16d0 90b5 0000 0101 080a 000e 10d1        ................
0x0030   0007 1fc5 5041 5353 206c 3840 6e69 7465        ....PASS.l8@nite
0x0040   0d0a
```

Example: dsniff Program Output

- *tcpdump* is a **general-purpose** packet sniffer, but there are **specialized** sniffing tools designed specifically to search for usernames and passwords.
- One example is Dung Song's program, dsniff, which is smart enough to pars out data that looks important.

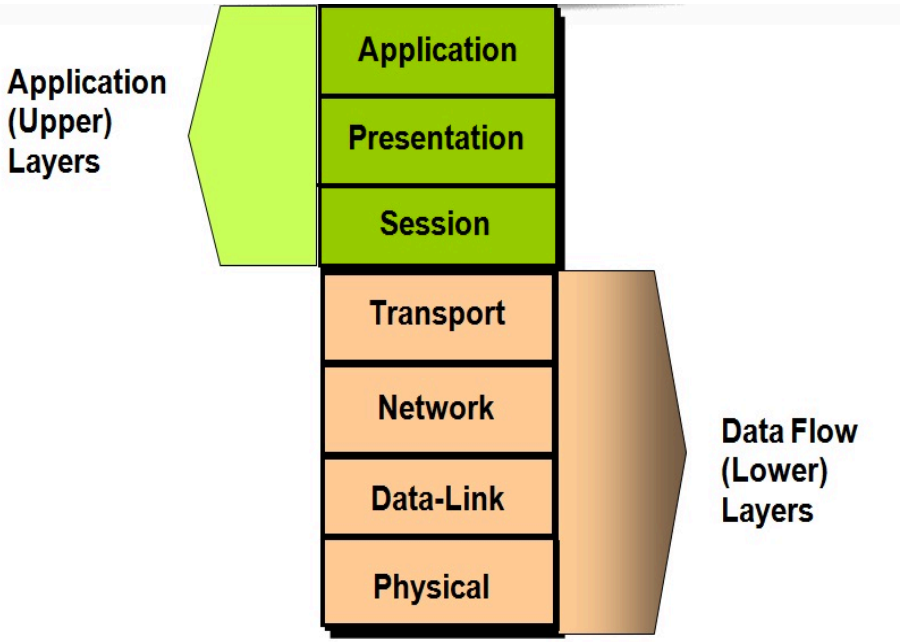
```
dsniff: listening on eth0
-----------------
12/10/02       21:43:21 tcp 192.168.0.193.32782 -> 192.168.0.118.21 (ftp)
USER leech
PASS l8@nite

------------------
12/10/2 21:47:49 tcp 192.168.0.193.32785 -> 192.168.0.120.23 (telnet)
USER root
PASS 5eCr3t
```

Raw Socket Sniffer

OSI upper levels vs lower levels

- At upper layers, operating system takes care of details of transmission and routing
- At lower layers, all details are exposed and must be handled explicitly by programmer.

- So far the examples were at upper levels and the data was neatly wrapped in a TCP/IP connection.
- It is possible to access the network at lower layers using raw sockets.



Example: TCP Sniffing Using Raw Socket I

```
 # include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <arpa/inet.h>


# include “hacking.h”


int main(void) {
  int I, recv_length, sockfd;
  u_char buffer[9000];                                  # u_char  is a type definition from sys/socket.h that expands to “unsigned char”
  If ((socket_fd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) == -1)                  # open raw TCP socket
                fatal(“in socket”);
```

Example: TCP Sniffing Using Raw Socket II

```
For(i=0; i<3; i++){      # Listen to three packets

        recv_length= recv(sockfd, buffer, 8000, 0);
        printf("Got a %d byte packet \n", recv_length);
        dump(buffer, recv_length);          # printing the raw data

            }

}
```

- The program needs to be run with root privilege, since the use of raw sockets require root access.

Notes about Raw TCP Sniffing Program

- The raw_tcpsniff.c program:
  - Opens a raw TCP socket and Listens for three packets
  - Print the raw data of each packet with dump function

- I raw_tcpsniff.c program:
  - Raw sockets are specified by using *SOCKET_RAW* as the type
  - In this case the protocol matters since there are multiple options. The protocol can be `IPPROTO_TCP`, `IPPROTO_UDP`, or `IP_PROTOICMP`
  - The buffer is declared as `u_char` variable. This is just a convenience type definition from `sys/socket.h` library that expands to "unsigned char"

Example: TCP Sniffing Program Output

- The program needs to be run as root, since the use of raw sockets requires root access.

```
reader@hacking:~/booksrc $ gcc -o decode_sniff decode_sniff.c -lpcap
reader@hacking:~/booksrc $ sudo ./decode_sniff
Sniffing on device eth0
    Got a 75 byte packet
[[  Layer 2 :: Ethernet Header  ]]
[ Source: 00:01:29:15:65:b6     Dest: 00:01:6c:eb:1d:50 Type: 8 ]
       ((  Layer 3 ::: IP Header  ))
       ( Source: 192.168.42.1  Dest: 192.168.42.249 )
       ( Type: 6       ID: 7755        Length: 61 )
              {{  Layer 4 :::: TCP Header  }}
              { Src Port: 35602       Dest Port: 7890 }
              { Seq #: 2887045274     Ack #: 3843058889 }
              { Header Size: 32       Flags: PUSH ACK  }
                     9 bytes of packet data
74 65 73 74 69 6e 67 0d 0a                      | testing..
==== Got a 66 byte packet ====
[[  Layer 2 :: Ethernet Header  ]]
[ Source: 00:01:6c:eb:1d:50     Dest: 00:01:29:15:65:b6 Type: 8 ]
       ((  Layer 3 ::: IP Header  ))
       ( Source: 192.168.42.249        Dest: 192.168.42.1 )
       ( Type: 6       ID: 15678       Length: 52 )
              {{  Layer 4 :::: TCP Header  }}
              { Src Port: 7890        Dest Port: 35602 }
              { Seq #: 3843058889     Ack #: 2887045283 }
              { Header Size: 32       Flags: ACK  }
                     No Packet Data
==== Got a 82 byte packet ====
[[  Layer 2 :: Ethernet Header  ]]
[ Source: 00:01:29:15:65:b6     Dest: 00:01:6c:eb:1d:50 Type: 8 ]
       ((  Layer 3 ::: IP Header  ))
       ( Source: 192.168.42.1  Dest: 192.168.42.249 )
       ( Type: 6       ID: 7756        Length: 68 )
              {{  Layer 4 :::: TCP Header  }}
              { Src Port: 35602       Dest Port: 7890 }
              { Seq #: 2887045283     Ack #: 3843058889 }
              { Header Size: 32       Flags: PUSH ACK  }
                     16 bytes of packet data
74 68 69 73 20 69 73 20 61 20 74 65 73 74 0d 0a | this is a test..
reader@hacking:~/booksrc $
```

Problems with the Raw Socket Sniffer

- While the example TCP sniffing program will capture packets, it will **miss some packets** when there is a lot of traffic.

- The example TCP sniffing program **only captures TCP packets**.
  - To capture UDP or ICMP packets, additional raw sockets need to be opened.

- **Multiplatform programming with raw sockets is nearly impossible.**
  - Raw sockets are notoriously inconsistent between systems.
  - Example: raw socket code for Linux most likely won't work on BSD or Solaris.
  - libpcap library can be used to smooth out inconsistencies.

libpcap Sniffer

- A standardized programming library.

- Used to smooth out inconsistencies of raw sockets.
  - Popular sniffing programs such as *tcpdump* and *dsniff* use *libpcap*.

- Function in the *libpcap* library use raw sockets to do their magic, But the library knows how to correctly work with raw sockets on multi architectures.

Example: Raw Packet Sniffer Using *libcap*

pcap_fatal function for displaying fatal errors

```
#include <pcap.h>

#include “hacking.h”
void pcap_fatal(const char *failed_in, const char *errbuf){

        printf(“Fatal error in %s: %s\n”,failed_in, err_buff);
        exit(1);
}
```

Notes about pcap_fatal funtion

- *pcap.h* is included to provide various structures and defines for pcap functions.
- pcap functions use an error buffer to return errors and status messages.
- *pcap_fatal* function is designed to display the error buffer to the user.

Example: Raw Packet Sniffer Using *libcap*

Main function for sniffing raw packets (part 1)

```
int main() {
        struct pcap_pkhdr header;
        construct u_char *packet;
        char errbuff[PCAP_ERRBUF_SIZE];        #pcap error buffer
        char *device;
        pcap_t *pcap_handle;                   #reference packet capturing object
        int i;
```

Notes about libcap Sniffer

Main function for sniffing raw packets (part 1)

- Part 1 contains variable declarations.
- The *errbuff* variable is is the *pcap* error buffer, its size is defined (in *pcap.h*) to be 256.
- *header* variable is a *pcap_pkthdr* structure containing extra capture information about the packet, such as when it was captured and its length.
- *pcap_handle* is used to reference a packet capturing object.

Example: Raw Packet Sniffer Using *libcap*

Main function for sniffing raw packets (part 2)

```
device= pcap_lookupdev(errbuf);              #look for a suitable device to sniff on

If(device==NULL)
            pcap_fatal("pcap_lookupdev",errbuf);

printf("sniffing on device %s \n", device);
```

- pcap_lookupdev returns a string pointer referring static function memory.
  - e.g. *dev/eth0*
  - If function can not find a suitable interface, It will return *NULL*.

Example: Raw Packet Sniffer Using *libcap*

Main function for sniffing raw packets (part 3)

```
pcap_handle = pcap_open_live(device,49=096,1,0, errbuf); # open a packet capturing device
if (pcap_handle == NULL)
        pcap_fatal("pcap_open_live",errbuf);
```

Notes About libcap Sniffer

Main function for sniffing raw packets (part 3)

- *pcap_open_live()* function opens a packet capturing device and returns a handle to it.

- *pcap_open_live () function* arguments are:
  - A packet capturing device
  - Maximum packet size
  - Promiscuous flag
  - Timeout Value
  - Pointer to error buffer

Example: Raw Packet Sniffer Using *libcap*

Main function for sniffing raw packets (part 4)

```
for(i=0; i<3; i++){
                packet=packet_next(pcap_handle, &header); # grab the next packet
                printf("Got a %d byte packet \n", header.len);
                dump(buffer, header.len);
                }
   pcap_close(pcap_handle); # closes the capture interface

   }
```

- *pcap_next()* function arguments are:
  - *pcap_handle*
  - *A pointer to pcap_pkthdr* structure (to be filled with details of the capture)

Example: Output of *libcap* sniffer

```
reader@hacking:~/booksrc $ gcc -o pcap_sniff pcap_sniff.c
/tmp/ccYgieqx.o: In function `main':
pcap_sniff.c:(.text+0x1c8): undefined reference to `pcap_lookupdev'
pcap_sniff.c:(.text+0x233): undefined reference to `pcap_open_live'
pcap_sniff.c:(.text+0x282): undefined reference to `pcap_next'
pcap_sniff.c:(.text+0x2c2): undefined reference to `pcap_close'
collect2: ld returned 1 exit status
reader@hacking:~/booksrc $ gcc -o pcap_sniff pcap_sniff.c -l pcap
reader@hacking:~/booksrc $ ./pcap_sniff
Fatal Error in pcap_lookupdev: no suitable device found
reader@hacking:~/booksrc $ sudo ./pcap_sniff
Sniffing on device eth0
Got a 82 byte packet
00 01 6c eb 1d 50 00 01 29 15 65 b6 08 00 45 10 | ..l..P..).e...E.
00 44 1e 39 40 00 40 06 46 20 c0 a8 2a 01 c0 a8 | .D.9@.@.F ..*...
2a f9 8b 12 1e d2 ac 14 cf c7 e5 10 6c c9 80 18 | *...........l...
05 b4 54 1a 00 00 01 01 08 0a 26 b6 a7 76 02 3c | ..T.......&..v.<
37 1e 74 68 69 73 20 69 73 20 61 20 74 65 73 74 | 7.this is a test
0d 0a                                           | ..
Got a 66 byte packet
00 01 29 15 65 b6 00 01 6c eb 1d 50 08 00 45 00 | ..).e...l..P..E.
00 34 3d 2c 40 00 40 06 27 4d c0 a8 2a f9 c0 a8 | .4=,@.@.'M..*...
2a 01 1e d2 8b 12 e5 10 6c c9 ac 14 cf d7 80 10 | *.......l.......
```

Notes about Output of *libcap* sniffer

- The *libcap* sniffer is compiled using –l flag to link *pcap* library.

```
$ gcc -o pcap_sniff packet_sniff.c -l pcap
```

- There are many bytes preceding the sample text in the packet.
- Since the output is raw packet captures, it contains layers of header information for Ethernet, IP and TCP.

Decoding Layers

Ethernet Layer

- In the raw packet captured the **outermost layer** is **Ethernet**.
- Ethernet is used to send data between Ethernet end-points with **MAC addresses**.

Header of Ethernet Layer Packers:

- Source MAC address
- Destination MAC address
- The type of Ethernet packet (a16 bit value)

Structure of headers in linux

- The structure of this layer is defined in */usr/include/linux/if_ethernet.h*
- The structure for IP header are located in *user/include/netinet/ip.h*
- The structure of TCP layer is located in *user/include/netinet/tcp.h*

To gain better understanding of header structures:

1. We look at existing definitions of the Ethernet header
2. We create our own header structure

The Definition of Ethernet Header

From */usr/include/linux/if_ethernet.h*

```
# define ETH_ALEN 6                /* Octets in one Ethernet addr */
# define ETH_HLEN 14               /* Octets in one Ethernet addr */

/*
* This is an Ethernet frame header
*/

struct ethhdr {
unsigned_char h_dest[ETH_ALEN];    /* Destination eth addr */
unsigned_char h_src[ETH_ALEN];     /* Source eth addr */
__be16    h_proto;                 /* Packet type ID field */

}__attribute__((packed));
```

More about the definition of Ethernet header

The structure contains three elements of an Ethernet header

- Definition of destination/source MAC address length in ETH_ALEN as 6 bytes.
- Definition of Ethernet header length in ETH_HLEN as 14 bytes.
- A type definition for a 16-bit (2 bytes) unsigned short integer.

More about the definition of Ethernet header

The structure contains three elements of an Ethernet header

- Definition of destination/source MAC address length in ETH_ALEN as **6 bytes**.
- Definition of Ethernet header length in ETH_HLEN as **14 bytes**.
- A type definition for a 16-bit (**2 bytes**) unsigned short integer.

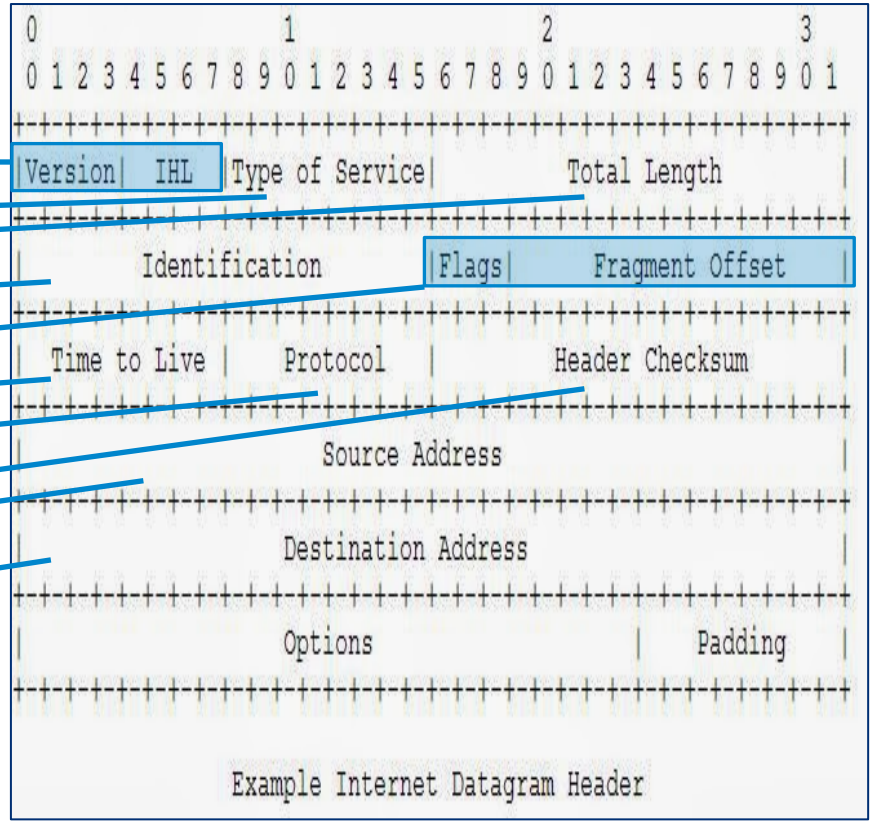
14 bytes= **2 bytes** + 2* (**6 bytes**)

Create our own Ethernet header structure

```
#define ETHER_ADDR_LEN 6
#define ETHER_HDR_LEN 14

struct ether_hdr{
unsigned char ether_dest_addr[ETHER_ADDR_LEN];      // Dest MAC address
unsigned char ether_src_addr[ETHER_ADDR_LEN];       // Source MAC address
unsigned short ether_type;                          // Type of Ethernet Packet
};
```

Create our own IP header structure

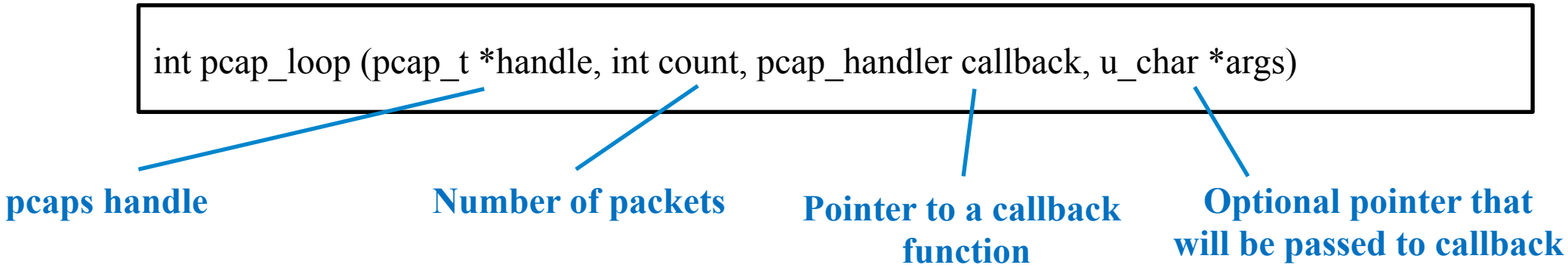
```
struct ip_hdr{
unsigned char ip_version_and_header_length;
unsigned char ip_tos;
unsigned short ip_len;
unsigned short ip_id;
unsigned short ip_frag_offset;
unsigned char ip_ttl;
unsigned char ip_type;
usigned_short_ip_checksum;
unsigned int ip_src_addr;
unsigned int ip_dest_addr;
};
```



Decode the layers I

- Having the headers defined as structures, we can write a program to decode the layered headers of each packet.

A necessary libcap function



Decode the layers II

A necessary libcap function

```
int pcap_loop (pcap_t *handle, int count, pcap_handler callback, u_char *args)
```

- the callback function needs to follow a certain prototype.

```
void callback (u_char *args, const struct pcap_pkthdr *cap_header, const u_char *packet);
```

The optional pointer from pcap_loop arguments

A pointer to the capture header structure

A pointer to the Packet itself

Example: A program that decode packets

Example: Main Function

decode_sniff.c

```
#include <pcap.h>
#include "hacking.h"
#include "hacking-network.h"

void pcap_fatal(const char *, const char *);
void decode_ethernet(const u_char *);
void decode_ip(const u_char *);
u_int decode_tcp(const u_char *);

void caught_packet(u_char *, const struct pcap_pkthdr *, const u_char *);

int main() {
   struct pcap_pkthdr cap_header;
   const u_char *packet, *pkt_data;
   char errbuf[PCAP_ERRBUF_SIZE];
   char *device;

   pcap_t *pcap_handle;

   device = pcap_lookupdev(errbuf);
   if(device == NULL)
      pcap_fatal("pcap_lookupdev", errbuf);

   printf("Sniffing on device %s\n", device);

   pcap_handle = pcap_open_live(device, 4096, 1, 0, errbuf);
   if(pcap_handle == NULL)
      pcap_fatal("pcap_open_live", errbuf);

   pcap_loop(pcap_handle, 3, caught_packet, NULL);

   pcap_close(pcap_handle);
}
```

Example: Callback function

- The callback function gets called when ever pcap_loop() function captures a packet.

```
void caught_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char
*packet) {
   int tcp_header_length, total_header_size, pkt_data_len;
   u_char *pkt_data;

   printf("==== Got a %d byte packet ====\n", cap_header->len);


   decode_ethernet(packet);
   decode_ip(packet+ETHER_HDR_LEN);
   tcp_header_length = decode_tcp(packet+ETHER_HDR_LEN+sizeof(struct ip_hdr));

   total_header_size = ETHER_HDR_LEN+sizeof(struct ip_hdr)+tcp_header_length;
   pkt_data = (u_char *)packet + total_header_size;  // pkt_data points to the data portion.
   pkt_data_len = cap_header->len - total_header_size;
   if(pkt_data_len > 0) {
      printf("\t\t\t%u bytes of packet data\n", pkt_data_len);
      dump(pkt_data, pkt_data_len);
   } else
      printf("\t\t\tNo Packet Data\n");
}

void pcap_fatal(const char *failed_in, const char *errbuf) {
   printf("Fatal Error in %s: %s\n", failed_in, errbuf);
   exit(1);
}
```

Example: decode_ethernet function

```
void decode ethernet(const u char *header start) {
   int i;
   const struct ether hdr *ethernet header;

   ethernet header   (const struct ether hdr *)header start;
   printf("[[  Layer 2 :: Ethernet Header  ]]\n");
   printf("[ Source: %02x", ethernet header->ether src addr[0]);
   for(i 1; i < ETHER ADDR LEN; i++)
      printf(":%02x", ethernet header->ether src addr[i]);

   printf("\tDest: %02x", ethernet header->ether dest addr[0]);
   for(i=1; i < ETHER ADDR LEN; i++)
      printf(":%02x", ethernet header->ether dest addr[i]);
   printf("\tType: %hu ]\n", ethernet header->ether type);
}
```

Example: decode_ip function

```
void decode_ip(const u_char *header_start) {
   const struct ip_hdr *ip_header;

   ip_header = (const struct ip_hdr *)header_start;
   printf("\t((  Layer 3 ::: IP Header  ))\n");
   printf("\t( Source: %s\t", inet_ntoa(ip_header->ip_src_addr));
   printf("Dest: %s )\n", inet_ntoa(ip_header->ip_dest_addr));
   printf("\t( Type: %u\t", (u_int) ip_header->ip_type);
   printf("ID: %hu\tlength: %hu )\n", ntohs(ip_header->ip_id), ntohs(ip_header->ip_len));
}
```

Reading Materials

Textbook

- Hacking: The art of exploitation, Jon Erickson
  - Chapter 4 - From 0x440 to 0x444

Programming Library Documentations

- libpcap library documentation: https://www.tcpdump.org/pcap.html
- libnet library documentation: http://libnet.sourceforge.net/libnet.html