# NETWORK SECURITY CSE 508

Aditya Tomer
111491409

---

## Header Files Used

**Standard c header files**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <stdint.h>
```

**Header files related to packet sniffing**

```
#include <pcap.h> // sniffing packets from network
#include <netinet/if_ether.h> // fetch ethernet header from the captured packets
#include <netinet/ip.h> // fetch ip header from the captured packets
#include <netinet/tcp.h> // fetch tcp header from the captured packets
#include <netinet/udp.h> // fetch udp header from the captured packets
#include <arpa/inet.h> // various conversions like ntoa, ntohs …
#include <netinet/in.h>
#include <unistd.h>
#include <time.h>
#include <netinet/ether.h>
```

## Flow of program using Functions

**getopt**() function is used for parsing command line arguments.
**-r** : look for a file at location
**-i** : device interface lookup
**-s** : display only IP packets whose payload contains the pattern string provided after -s.
NOTE: Incase -i and -s both parameters are provided, preference is given to -r parameter for reading from the offline file.
After creating session for the packets sniffing, **pcap_lookupnet**() is used to look up machine ip address and mask. Once we have Ip address and Mask. We use

**pcap_open_live() :** for sniffing live packets from the network using the provided network interface by **-i** parameter. Or by using
**or**
**pcap_open_offline() :** for sniffing packets from a **pcap file through offline** medium.

**pcap_datalink() :** is used for figuring out if the device supports Ethernet headers.

In the next step bpf filter expression is provided to sniff particular protocols using
**pcap_compile() :** which, compiles the chosen bpf filter and makes it ready for use.

**pcap_setfilter() :** applies the compiled bpf filter and filters the traffic.

Once filter expression is ready to use.
**pcap_loop() : is** used for continuously sniffing and capturing the packet. Captures as many packets as indicated by the second argument **num_packets.** If it's value is set to -1, it continues capturing until an error is occurred, else sniffs as many packets as mentioned by the parameter.

// A network packet contains **ethernet** header at the top, Ip packet after it and the underlying header UDP/TCP/ICMP/Others inside it, which also contains the data payload.
**handle_packets_callback(u_char** ***args, const struct pcap_pkthdr** ***packetHeader, const u_char** ***packet) :**

1. function is the callback function which is called every time a new packet is captured and sequence of steps as mentioned inside it is performed for each packet.
2. **pcap_pkthdr :** Timestamp of the packet is fetched from it.
3. **packet :**
    3a. **(struct ether_header *)** contains source Mac, destination Mac, ethernet type
    3b. After extracting the Mac related details, **IP header** is extracted. Whose length is given by the parameter "**ip_hl**". Using **"ip_src" source IP** of the packet is extracted. **"Ip_dst" destination IP** of the packet is extracted.
    **3c. "ip->ip_p"** parameter specifies which protocol packet is contained inside. Currently handled packet types are TCP, UDP, ICMP, Others.

**createPayloadString()** function returns the packet data payload in a string, it contains only printable characters and non printable characters are represented by a dot(.).
**-r** parameter used earlier represents the pattern string which is searched inside the payload data string for filtering the packets.

A payload is represented in rows of 32 bytes. First 16 bytes provides the hex addresses of the payload. Last 16 bytes represents the characters inside it.

**print_hex_ascii_line() :** prints a line first 16 characters print the hex value of payload. Last 16 characters print the character itself. If the character cannot be shown on screen or not printed a dot "." is used.

Finally after the packet information is printed, all cleanups are performed like -
**pcap_freecode() :** Free up/Clean up bpf filter

**pcap_close():** Closing the session used for sniffing.

**OUTPUT:**

Here Offline file is provided by -r parameter and -s is used for searching google inside the payload of packets.

./2hw -r hw1.pcap -s google udp

```
aditya@aditya-VirtualBox:~/Desktop/Masters/1sem/NetSec/2Hw$ make
gcc -c  2hw.c
gcc 2hw.o -o 2hw -lpcap
aditya@aditya-VirtualBox:~/Desktop/Masters/1sem/NetSec/2Hw$ ./2hw -r hw1.pcap -s google udp
2013-01-13 05:36:11.638926  00:0c:29:e9:94:8e ->  c4:3d:c7:17:6f:9b type 0x0800 len 74 192.168.0.200:41133 -> 194.168.4.100:53 UDP
00000   4f 19 01 00 00 01 00 00  00 00 00 00 03 77 77 77    O............www
00016   06 67 6f 6f 67 6c 65 03  63 6f 6d 00 00 01 00 01    .google.com.....
2013-01-13 05:36:11.649312  c4:3d:c7:17:6f:9b ->  00:0c:29:e9:94:8e type 0x0800 len 170 194.168.4.100:53 -> 192.168.0.200:41133 UDP
00000   4f 19 81 80 00 01 00 06  00 00 00 00 03 77 77 77    O............www
00016   06 67 6f 6f 67 6c 65 03  63 6f 6d 00 00 01 00 01    .google.com.....
00032   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 68    ..............Ch
00048   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 93    ..............C.
00064   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 67    ..............Cg
00080   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 69    ..............Ci
00096   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 6a    ..............Cj
00112   c0 0c 00 01 00 01 00 00  01 02 00 04 ad c2 43 63    ..............Cc
```

Here the program is listening to device enp0s3 using -i parameter and tcp is used as a bfp filter.

./2hw -i enp0s3 tcp

```
2017-10-14 05:39:56.372134  08:00:27:75:9d:5a ->  52:54:00:12:35:02 type 0x0800 len 74 10.0.2.15:35244 -> 172.86.179.14:80 TCP
00000   5d 3c e9 84 00 00 00 00  a0 02 72 10 6b a2 00 00    ]<........r.k...
00016   02 04 05 b4 04 02 08 0a  06 85 b6 49 00 00 00 00    ...........I....
00032   01 03 03 07                                        ....
2017-10-14 05:39:56.407973  52:54:00:12:35:02 ->  08:00:27:75:9d:5a type 0x0800 len 60 172.86.179.14:80 -> 10.0.2.15:35244 TCP
00000
2017-10-14 05:39:56.408015  08:00:27:75:9d:5a ->  52:54:00:12:35:02 type 0x0800 len 54 10.0.2.15:35244 -> 172.86.179.14:80 TCP
00000   c3 7a f8 02 50 10 72 10  6b 8e 00 00                .z..P.r.k...
```

REFERENCES Used for the assignment


1. http://www.tcpdump.org/pcap.html
2. http://unix.superglobalmegacorp.com/Net2/newsrc/netinet/if_ether.h.html
3. http://unix.superglobalmegacorp.com/Net2/newsrc/netinet/ip.h.html
4. http://unix.superglobalmegacorp.com/BSD4.4/newsrc/netinet/tcp.h.html