# rp\_filter：反向过滤技术

reverse-path filtering，反向过滤技术，系统在接收到一个IP包后，检查该IP是不是合乎要求，不合要求的IP包会被系统丢弃。该技术就称为rp filter。怎么样的包才算不合要求呢？例如，用户在A网口上收到一个IP包，检查其IP为B。然后考查：对于B这个IP，在发送时应该用哪个网口，“如果在不应该接收到该包的网口上接收到该IP包，则认为该IP包是hacker行为”。

例如：

A: 192.168.8.100

B: (IGMP Query) 10.0.0.1 来自路由器

查找路由表

网卡1为默认路由: 172.17.5.100 172.17.5.1

网卡2 192.168.8.100 192.168.8.1

系统根据路由表，认为10.0.0.1这个IP应该在第一个网卡172.17.5.100上收到，现实的情况是在第二张网卡192.168.8.100上收到了。认为这是不合理的，丢弃该包。致命的问题的，该包是来自路由器的IGMP Query包。

The rp\_filter can reject incoming packets if their source address doesn’t match the network interface that they’re arriving on, which helps to prevent IP spoofing. Turning this on, however, has its consequences: If your host has several IP addresses on different interfaces, or if your single interface has multiple IP addresses on it, you’ll find that your kernel may end up rejecting valid traffic. It’s also important to note that even if you do not enable the rp\_filter, protection against broadcast spoofing is always on. Also, the protection it provides is only against spoofed internal addresses; external addresses can still be spoofed.. By default, it is disabled.

解决方法：

系统配置文件

1. /etc/sysctl.conf

把 net.ipv4.conf.all.rp\_filter和 net.ipv4.conf.default.rp\_filter设为0即可

net.ipv4.conf.default.rp\_filter = 0

net.ipv4.conf.all.rp\_filter = 0

系统启动后，会自动加载这个配置文件，内核会使用这个变量

2. 命令行

显示一个内核变量 sysctl net.ipv4.conf.all.rp\_filter

设置一个内核变量 sysctl -w net.ipv4.conf.all.rp\_filter=0

设置完后，会更新内核（实时的内存）中的变量的值，但不会修改sysctl.conf的值

3. 使用/proc文件系统

查看 cat /proc/sys/net/ipv4/conf/all/rp\_filter

设置 echo "0" >/proc/sys/net/ipv4/conf/all/rp\_filter

# TCP

## Nagle算法

#include<netinet/tcp.h>

int flag = 1;

int res = setsockopt(fGS->socketNum(), IPPROTO\_TCP, TCP\_NODELAY, &flag, sizeof(flag));

Nagle算法用于对缓冲区内的一定数量的消息进行自动连接。该处理过程(称为Nagling)，通过减少必须发送的封包的数量，提高了网络应用 程序系统的效率。（Nagle虽然解决了小封包问题，但也导致了较高的不可预测的延迟，同时降低了吞吐量。）

vc下面socket编程，使用阻塞方式的时候，会自动使用Nagle算法，如：当pc不断发送32Bytes的数据的时候，会将这些包合并起来一起发送。如果另一头使用的时候一个tcp包一条命令处理的话，会出问题的。

# Setsockopt

#include <sys/types.h>

#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname,void \*optval, socklen\_t \*optlen);

int setsockopt(int sockfd, int level, int optname,const void \*optval, socklen\_t optlen);

**协议层        选项名字  
SOL\_SOCKET    SO\_REUSEADDR  
SOL\_SOCKET    SO\_KKEPALIVE  
SOL\_SOCKET    SO\_LINGER  
SOL\_SOCKET    SO\_BROADCAST  
SOL\_SOCKET    SO\_OOBINLINE  
SOL\_SOCKET    SO\_SNDBUF  
SOL\_SOCKET    SO\_RCVBUF  
SOL\_SOCKET    SO\_TYPE  
SOL\_SOCKET    SO\_ERROR  
SOL\_TCP        SO\_NODELAY**

**SO\_ACCEPTCONN**

Returns a value indicating whether or not this socket has been marked to accept connections with listen(2). The value 0 indicates that this is not a listening socket, the value 1 indicates that this is a listeningsocket. This socket option is read-only.

**SO\_BINDTODEVICE**

Bind this socket to a particular device like "eth0", as specified in the passed interface name. If the name is an empty string or the option length is zero, the socket device binding is removed. The passed option is a variable-length null-terminated interface name string with the maximum size of IFNAMSIZ. If a socket is bound to an interface, only packets received from that particular interface are processed by the socket.

Note that this works only for some socket types, particularly AF\_INET sockets. It is not supported for packet sockets (use normal bind(2) there).

Before Linux 3.8, this socket option could be set, but could not retrieved with getsockopt(2). Since Linux 3.8, it is readable. The optlen argument should contain the buffer size available to receive the device name and is recommended to be IFNAMSZ bytes. The real device name length is reported back in the optlen argument.

**SO\_BROADCAST**

Set or get the broadcast flag. When enabled, datagram sockets are allowed to send packets to a broadcast address. This option has no effect on stream-oriented sockets.

一般在发送UDP数据报的时候，希望该socket发送的数据具有广播特性：

BOOL bBroadcast=TRUE;

setsockopt(s,SOL\_SOCKET,SO\_BROADCAST,(const char\*)&bBroadcast,sizeof(BOOL));

**SO\_BSDCOMPAT**

Enable BSD bug-to-bug compatibility. This is used by the UDP protocol module in Linux 2.0 and 2.2. If enabled ICMP errors received for a UDP socket will not be passed to the user program. In later kernel versions, support for this option has been phased out: Linux 2.4 silently ignores it, and Linux 2.6 generates a kernel warning (printk()) if a program uses this option. Linux 2.0 also enabled BSD bug-to-bug compatibility options (random header changing, skipping of the broadcast flag) for raw sockets with this option, but that was removed in Linux 2.2.

**SO\_DEBUG**

Enable socket debugging. Only allowed for processes with the CAP\_NET\_ADMIN capability or an effective user ID of 0.

**SO\_DOMAIN (since Linux 2.6.32)**

Retrieves the socket domain as an integer, returning a value such as AF\_INET6. See socket(2) for details. This socket option is read-only.

**SO\_ERROR**

Get and clear the pending socket error. This socket option is read-only. Expects an integer.

**SO\_DONTROUTE**

Don't send via a gateway, send only to directly connected hosts. The same effect can be achieved by setting the MSG\_DONTROUTE flag on a socket send(2) operation. Expects an integer boolean flag.

**SO\_KEEPALIVE**

Enable sending of keep-alive messages on connection-oriented sockets. Expects an integer boolean flag.

**SO\_LINGER**

When enabled, a close(2) or shutdown(2) will not return until all queued messages for the socket have been successfully sent or the linger timeout has been reached. Otherwise, the call returns immediately and the closing is done in the background. When the socket is closed as part of exit(2), it always lingers in the background.

如果要已经处于连接状态的soket在调用closesocket后强制关闭，不经历TIME\_WAIT的过程：

struct linger lig;

int iLen;

lig.l\_onoff=1; //(在closesocket()调用,但是还有数据没发送完毕的时候容许逗留)

lig.l\_linger=0; //(容许逗留的时间为0秒)

iLen=sizeof(struct linger);

setsockopt(conn,SOL\_SOCKET,SO\_LINGER,(char \*)&lig,iLen);

如果在发送数据的过程中(send()没有完成，还有数据没发送)而调用了closesocket(),以前我们一般采取的措施是"从容关闭"shutdown(s,SD\_BOTH),但是数据是肯定丢失了，如何设置让程序满足具体应用的要求(即让没发完的数据发送出去后在关闭socket)？

struct linger {

u\_short l\_onoff;

u\_short l\_linger;

};

linger m\_sLinger;

m\_sLinger.l\_onoff=1;//(在closesocket()调用,但是还有数据没发送完毕的时候容许逗留)

// 如果m\_sLinger.l\_onoff=0;则功能和2.)作用相同;

m\_sLinger.l\_linger=5;//(容许逗留的时间为5秒)

setsockopt(s,SOL\_SOCKET,SO\_LINGER,(const char\*)&m\_sLinger,sizeof(linger));

**SO\_MARK (since Linux 2.6.25)**

Set the mark for each packet sent through this socket (similar to the netfilter MARK target but socket-based). Changing the mark can be used for mark-based routing without netfilter or for packet filtering. Setting this option requires the CAP\_NET\_ADMIN capability.

**SO\_OOBINLINE**

If this option is enabled, out-of-band data is directly placed into the receive data stream. Otherwise out-of-band data is passed only when the MSG\_OOB flag is set during receiving.

**SO\_PASSCRED**

Enable or disable the receiving of the SCM\_CREDENTIALS control message. For more information see unix(7).

**SO\_PEEK\_OFF (since Linux 3.4)**

This option, which is currently supported only for unix(7) sockets, sets the value of the "peek offset" for the recv(2) system call when used with MSG\_PEEK flag.

When this option is set to a negative value (it is set to -1 for all new sockets), traditional behavior is provided: recv(2) with the MSG\_PEEK flag will peek data from the front of the queue.

When the option is set to a value greater than or equal to zero, then the next peek at data queued in the socket will occur at the byte offset specified by the option value. At the same time, the "peek offset" will be incremented by the number of bytes that were peeked from the queue, so that a subsequent peek will return the next data in the queue.

If data is removed from the front of the queue via a call to recv(2) (or similar) without the MSG\_PEEK flag, the "peek offset" will be decreased by the number of bytes removed. In other words, receiving data without the MSG\_PEEK flag will cause the "peek offset" to be adjusted to maintain the correct relative position in the queued data, so that a subsequent peek will retrieve the data that would have been retrieved had the data not been removed.

For datagram sockets, if the "peek offset" points to the middle of a packet, the data returned will be marked with the MSG\_TRUNC flag.

The following example serves to illustrate the use of SO\_PEEK\_OFF. Suppose a stream socket has the following queued input data:

aabbccddeeff

The following sequence of recv(2) calls would have the effect noted in the comments:

int ov = 4; // Set peek offset to 4

setsockopt(fd, SOL\_SOCKET, SO\_PEEK\_OFF, &ov, sizeof(ov));

recv(fd, buf, 2, MSG\_PEEK); // Peeks "cc"; offset set to 6

recv(fd, buf, 2, MSG\_PEEK); // Peeks "dd"; offset set to 8

recv(fd, buf, 2, 0); // Reads "aa"; offset set to 6

recv(fd, buf, 2, MSG\_PEEK); // Peeks "ee"; offset set to 8

**SO\_PEERCRED**

Return the credentials of the foreign process connected to this socket. This is possible only for connected AF\_UNIX stream sockets and AF\_UNIX stream and datagram socket pairs created using socketpair(2); see unix(7). The returned credentials are those that were in effect at the time of the call to connect(2) or socketpair(2). The argument is a ucred structure; define the GNU\_SOURCE feature test macro to obtain the definition of that structure from <sys/socket.h>. This socket option is read-only.

recv(fd, buf, 2, 0); // Reads "aa"; offset set to 6

recv(fd, buf, 2, MSG\_PEEK); // Peeks "ee"; offset set to 8

**SO\_PEERCRED**

Return the credentials of the foreign process connected to this socket. This is possible only for connected AF\_UNIX stream sockets and AF\_UNIX stream and datagram socket pairs created using socketpair(2); see unix(7). The returned credentials are those that were in effect at the time of the call to connect(2) or socketpair(2). The argument is a ucred structure; define the GNU\_SOURCE feature test macro to obtain the definition of that structure from <sys/socket.h>. This socket option is read-only.

**SO\_PRIORITY**

Set the protocol-defined priority for all packets to be sent on this socket. Linux uses this value to order the networking queues: packets with a higher priority may be processed first depending on the selected device queueing discipline. For ip(7), this also sets the IP type-of-service (TOS) field for outgoing packets. Setting a priority outside the range 0 to 6 requires the CAP\_NET\_ADMIN capability.

**SO\_PROTOCOL (since Linux 2.6.32)**

Retrieves the socket protocol as an integer, returning a value such as IPPROTO\_SCTP. See socket(2) for details. This socket option is read-only.

**SO\_RCVBUFFORCE (since Linux 2.6.14)**

Using this socket option, a privileged (CAP\_NET\_ADMIN) process can perform the same task as SO\_RCVBUF, but the rmem\_max limit can be overridden.

**SO\_RCVLOWAT and SO\_SNDLOWAT**

Specify the minimum number of bytes in the buffer until the socket layer will pass the data to the protocol (SO\_SNDLOWAT) or the user on receiving (SO\_RCVLOWAT). These two values are initialized to 1. SO\_SNDLOWAT is not changeable on Linux (setsockopt(2) fails with the error ENOPROTOOPT). SO\_RCVLOWAT is changeable only since Linux 2.4. The select(2) and poll(2) system calls currently do not respect the SO\_RCVLOWAT setting on Linux, and mark a socket readable when even a single byte of data is available. A subsequent read from the socket will block until SO\_RCVLOWAT bytes are available.

**SO\_RCVTIMEO** and **SO\_SNDTIMEO**

Specify the receiving or sending timeouts until reporting an error. The argument is a struct timeval. If an input or output function blocks for this period of time, and data has been sent or received, the return value of that function will be the amount of data transferred; if no data has been transferred and the timeout has been reached then -1 is returned with errno set to EAGAIN or EWOULDBLOCK, or EINPROGRESS (for connect(2)) just as if the socket was specified to be nonblocking. If the timeout is set to zero (the default) then the operation will never timeout. Timeouts only have effect for system calls that perform socket I/O (e.g., read(2), recvmsg(2), send(2), sendmsg(2)); timeouts have no effect for select(2), poll(2), epoll\_wait(2), and so on.

在send(),recv()过程中有时由于网络状况等原因，发收不能预期进行,而设置收发时限：

int nNetTimeout=1000;//1秒

//发送时限

setsockopt(socket，SOL\_S0CKET,SO\_SNDTIMEO，(char \*)&nNetTimeout,sizeof(int));

//接收时限

setsockopt(socket，SOL\_S0CKET,SO\_RCVTIMEO，(char \*)&nNetTimeout,sizeof(int));

这样做在Linux环境下是不会产生效果的，须如下定义：

struct timeval timeout = {3,0};

//设置发送超时

setsockopt(socket，SOL\_SOCKET,SO\_SNDTIMEO，(char \*)&timeout,sizeof(struct timeval));

//设置接收超时

setsockopt(socket，SOL\_SOCKET,SO\_RCVTIMEO，(char \*)&timeout,sizeof(struct timeval));

如果ret==0 则为成功,-1为失败,这时可以查看errno来判断失败原因

int recvd=recv(sock\_fd,buf,1024,0);

if(recvd==-1&&errno==EAGAIN) printf("timeout\n");

有两点注意就是：

1）recv ()的第四个参数需为MSG\_WAITALL，在阻塞模式下不等到指定数目的数据不会返回，除非超时时间到。还要注意的是只要设置了接收超时，在没有MSG\_WAITALL时也是有效的。说到底超时就是不让你的程序老在那儿等，到一定时间进行一次返回而已。

2）即使等待超时时间值未到，但对方已经关闭了socket， 则此时recv()会立即返回，并收到多少数据返回多少数据。

补充：

同样可以设置连接(connect)超时：即通过SO\_SNDTIMO套节字参数让超时操作跳过select。

原因是：Linux内核源码中connect的超时参数和SO\_SNDTIMO操作的参数一致。

因此，在linux平台下，可以通过connect之前设置SO\_SNDTIMO来达到控制连接超时的目的。

**SO\_REUSEADDR**

Indicates that the rules used in validating addresses supplied in a bind(2) call should allow reuse of local addresses. For AF\_INET sockets this means that a socket may bind, except when there is an active listening socket bound to the address. When the listening socket is bound to INADDR\_ANY with a specific port then it is not possible to bind to this port for any local address. Argument is an integer boolean flag.

closesocket（一般不会立即关闭而经历TIME\_WAIT的过程）后想继续重用该socket：

BOOL bReuseaddr=TRUE;

setsockopt(s,SOL\_SOCKET ,SO\_REUSEADDR,(const char\*)&bReuseaddr,sizeof(BOOL));

**SO\_SNDBUF && SO\_RCVBUF**

Sets or gets the maximum socket send buffer in bytes. The kernel doubles this value (to allow space for bookkeeping overhead) when it is set using setsockopt(2), and this doubled value is returned by getsockopt(2). The default value is set by the /proc/sys/net/core/wmem\_default file and the maximum allowed value is set by the /proc/sys/net/core/wmem\_max file. The minimum (doubled) value for this option is 2048.

如果在发送数据的时，希望不经历由系统缓冲区到socket缓冲区的拷贝而影响程序的性能：

int nZero=0;

setsockopt(socket，SOL\_S0CKET,SO\_SNDBUF，(char \*)&nZero,sizeof(nZero));

setsockopt(socket，SOL\_S0CKET,SO\_RCVBUF，(char \*)&nZero,sizeof(int));

在send()的时候，返回的是实际发送出去的字节(同步)或发送到socket缓冲区的字节(异步);系统默认的状态发送和接收一次为8688字节(约为8.5K)；在实际的过程中发送数据和接收数据量比较大，可以设置socket缓冲区，而避免了send(),recv()不断的循环收发：

// 接收缓冲区

int nRecvBuf=32\*1024;//设置为32K

setsockopt(s,SOL\_SOCKET,SO\_RCVBUF,(const char\*)&nRecvBuf,sizeof(int));

//发送缓冲区

int nSendBuf=32\*1024;//设置为32K

setsockopt(s,SOL\_SOCKET,SO\_SNDBUF,(const char\*)&nSendBuf,sizeof(int));

**SO\_SNDBUFFORCE** (since Linux 2.6.14)

Using this socket option, a privileged (CAP\_NET\_ADMIN) process can perform the same task as SO\_SNDBUF, but the wmem\_max limit can be overridden.

**SO\_TIMESTAMP**

Enable or disable the receiving of the SO\_TIMESTAMP control message. The timestamp control message is sent with level SOL\_SOCKET and the cmsg\_data field is a struct timeval indicating the reception time of the last packet passed to the user in this call. See cmsg(3) for details on control messages.

**SO\_TYPE**

Gets the socket type as an integer (e.g., SOCK\_STREAM). This socket option is read-only.

**SO\_CONDITIONAL\_ACCEPT**

在client连接服务器过程中，如果处于非阻塞模式下的socket在connect()的过程中可以设置connect()延时,直到accpet()被呼叫(本函数设置只有在非阻塞的过程中有显著的作用，在阻塞的函数调用中作用不大)

BOOL bConditionalAccept=TRUE;

setsockopt(s, SOL\_SOCKET, SO\_CONDITIONAL\_ACCEPT, (const char\*)&bConditionalAccept, sizeof(BOOL) );

**SO\_DONTLINGER**

如果要已经处于连接状态的soket在调用closesocket后强制关闭，不经历TIME\_WAIT的过程：

BOOL bDontLinger = FALSE;

setsockopt(s,SOL\_SOCKET, SO\_DONTLINGER, (const char\*)&bDontLinger,sizeof(BOOL));

TCP\_NODELAY BOOL 禁止发送合并的Nagle算法。

# MTU测试

Header(28) + Data = MTU

**Windows:**

ping -f -l size IP

**linux:**

ping -M do -s 1500 202.96.128.68

# UDP

/proc/sys/net/ipv4/

udp\_mem (since Linux 2.6.25)

This is a vector of three integers governing the number of pages allowed for queueing by all UDP sockets.

min Below this number of pages, UDP is not bothered about its memory appetite. When the amount of memory allocated by

UDP exceeds this number, UDP starts to moderate memory usage.

pressure This value was introduced to follow the format of tcp\_mem (see tcp(7)).

max Number of pages allowed for queueing by all UDP sockets.

Defaults values for these three items are calculated at boot time from the amount of available memory.

===============================================================================

1. 修改SO\_SNDBUF

int buf\_size = 10 \* 1024;

int send\_buf\_size = buf\_size \* 2 + 1024 \* 32;

setsockopt(msocket, SOL\_SOCKET, SO\_SNDBUF, (char\*)&send\_buf\_size, sizeof(send\_buf\_size));

2. 修改wmem\_max

cat /proc/sys/net/core/wmem\_max

vi /etc/sysctl.conf

wmem\_max=MAX

sysctl -a | grep wmen\_max

# End