NAME

divert - kernel packet diversion mechanism

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
socket(*PF_DIVERT*, *SOCK_RAW*, *0*);

To enable support for divert sockets, place the following lines in the kernel configuration file:

options IPFIREWALL
options IPDIVERT

Alternatively, to load the driver as a module at boot time, add the following lines into the loader.conf(5) file:

ipfw_load="YES"
ipdivert_load="YES"

DESCRIPTION

Divert sockets allow to intercept and re-inject packets flowing through the ipfw(4) firewall. A divert socket can be bound to a specific **divert** port via the bind(2) system call. The sockaddr argument shall be sockaddr_in with sin_port set to the desired value. Note that the **divert** port has nothing to do with TCP/UDP ports. It is just a cookie number, that allows to differentiate between different divert points in the ipfw(4) ruleset. A divert socket bound to a divert port will receive all packets diverted to that port by ipfw(4). Packets may also be written to a divert port, in which case they re-enter firewall processing at the next rule.

By reading from and writing to a divert socket, matching packets can be passed through an arbitrary ‘‘filter’’ as they travel through the host machine, special routing tricks can be done, etc.

READING PACKETS

Packets are diverted either as they are ‘‘incoming’’ or ‘‘outgoing.’’ Incoming packets are diverted after reception on an IP interface, whereas outgoing packets are diverted before next hop forwarding.

Diverted packets may be read unaltered via read(2), recv(2), or recvfrom(2). In the latter case, the address returned will have its port set to some tag supplied by the packet diverter, (usually the ipfw rule

number) and the IP address set to the (first) address of the interface on which the packet was received (if the packet was incoming) or INADDR_ANY (if the packet was outgoing). The interface name (if defined for the packet) will be placed in the 8 bytes following the address, if it fits.

WRITING PACKETS

Writing to a divert socket is similar to writing to a raw IP socket; the packet is injected ‘‘as is’’ into the normal kernel IP packet processing using sendto(2) and minimal error checking is done. Packets are distinguished as either incoming or outgoing. If sendto(2) is used with a destination IP address of INADDR_ANY, then the packet is treated as if it were outgoing, i.e., destined for a non-local address. Otherwise, the packet is assumed to be incoming and full packet routing is done.

In the latter case, the IP address specified must match the address of some local interface, or an interface name must be found after the IP address. If an interface name is found, that interface will be used and the value of the IP address will be ignored (other than the fact that it is not INADDR_ANY). This is to indicate on which interface the packet "arrived".

Normally, packets read as incoming should be written as incoming; similarly for outgoing packets. When reading and then writing back packets, passing the same socket address supplied by recvfrom(2) unmodified to sendto(2) simplifies things (see below).

The port part of the socket address passed to the sendto(2) contains a tag that should be meaningful to the diversion module. In the case of ipfw(8) the tag is interpreted as the rule number *after which* rule processing should restart.

LOOP AVOIDANCE

Packets written into a divert socket (using sendto(2)) re-enter the packet filter at the rule number following the tag given in the port part of the socket address, which is usually already set at the rule number that caused the diversion (not the next rule if there are several at the same number). If the ’tag’ is altered to indicate an alternative re-entry point, care should be taken to avoid loops, where the same packet is diverted more than once at the same rule.

DETAILS

If a packet is diverted but no socket is bound to the port, or if IPDIVERT is not enabled or loaded in the kernel, the packet is dropped.

Incoming packet fragments which get diverted are fully reassembled before delivery; the diversion of any one fragment causes the entire packet to get diverted. If different fragments divert to different ports, then which port ultimately gets chosen is unpredictable.

Note that packets arriving on the divert socket by the ipfw(8) **tee** action are delivered as-is and packet

fragments do not get reassembled in this case.

Packets are received and sent unchanged, except that packets read as outgoing have invalid IP header checksums, and packets written as outgoing have their IP header checksums overwritten with the correct value. Packets written as incoming and having incorrect checksums will be dropped. Otherwise, all header fields are unchanged (and therefore in network order).

Creating a **divert** socket requires super-user access.

ERRORS

Writing to a divert socket can return these errors, along with the usual errors possible when writing raw packets:

[EINVAL] The packet had an invalid header, or the IP options in the packet and the socket options set were incompatible.

[EADDRNOTAVAIL] The destination address contained an IP address not equal to INADDR_ANY that was not associated with any interface.

SEE ALSO

bind(2), recvfrom(2), sendto(2), socket(2), ipfw(4), ipfw(8)

AUTHORS

Archie Cobbs <*archie@FreeBSD.org*>, Whistle Communications Corp.

BUGS

This is an attempt to provide a clean way for user mode processes to implement various IP tricks like address translation, but it could be cleaner, and it is too dependent on ipfw(8).

It is questionable whether incoming fragments should be reassembled before being diverted. For example, if only some fragments of a packet destined for another machine do not get routed through the local machine, the packet is lost. This should probably be a settable socket option in any case.