NAME

ng_bpf **-** Berkeley packet filter netgraph node type

SYNOPSIS

#include <sys/types.h>
#include <net/bpf.h>
#include <netgraph.h>
#include <netgraph/ng_bpf.h>

DESCRIPTION

The **bpf** node type allows Berkeley Packet Filter (see bpf(4)) filters to be applied to data travelling through a Netgraph network. Each node allows an arbitrary number of connections to arbitrarily named hooks. With each hook is associated a bpf(4) filter program which is applied to incoming data only, a destination hook for matching packets, a destination hook for non-matching packets, and various statistics counters.

A bpf(4) program returns an unsigned integer, which is normally interpreted as the length of the prefix of the packet to return. In the context of this node type, returning zero is considered a non-match, in which case the entire packet is delivered out the non-match destination hook. Returning a value greater than zero causes the packet to be truncated to that length and delivered out the match destination hook. Either or both destination hooks may be the empty string, or may not exist, in which case the packet is dropped.

New hooks are initially configured to drop all packets. A new filter program may be installed using the NGM_BPF_SET_PROGRAM control message.

HOOKS

This node type supports any number of hooks having arbitrary names.

CONTROL MESSAGES

This node type supports the generic control messages, plus the following:

NGM_BPF_SET_PROGRAM (**setprogram**)

This command sets the filter program that will be applied to incoming data on a hook. The following structure must be supplied as an argument:

```
struct ng_bpf_hookprog {
  char          thisHook[NG_HOOKSIZ];     /* name of hook */
  char          ifMatch[NG_HOOKSIZ];      /* match dest hook */
  char          ifNotMatch[NG_HOOKSIZ];   /* !match dest hook */
```

```
      int32_t        bpf_prog_len;           /* #insns in program */
      struct bpf_insn bpf_prog[];            /* bpf program */
    };
```

The hook to be updated is specified in thisHook. The BPF program is the sequence of instructions in the bpf_prog array; there must be bpf_prog_len of them. Matching and non-matching incoming packets are delivered out the hooks named ifMatch and ifNotMatch, respectively. The program must be a valid bpf(4) program or else EINVAL is returned.

NGM_BPF_GET_PROGRAM (**getprogram**)
: This command takes an ASCII string argument, the hook name, and returns the corresponding struct ng_bpf_hookprog as shown above.

NGM_BPF_GET_STATS (**getstats**)
: This command takes an ASCII string argument, the hook name, and returns the statistics associated with the hook as a struct ng_bpf_hookstat.

NGM_BPF_CLR_STATS (**clrstats**)
: This command takes an ASCII string argument, the hook name, and clears the statistics associated with the hook.

NGM_BPF_GETCLR_STATS (**getclrstats**)
: This command is identical to NGM_BPF_GET_STATS, except that the statistics are also atomically cleared.

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, or when all hooks have been disconnected.

EXAMPLES

It is possible to configure a node from the command line, using tcpdump(1) to generate raw BPF instructions which are then transformed into the ASCII form of a NGM_BPF_SET_PROGRAM control message, as demonstrated here:

```
#!/bin/sh

PATTERN="tcp dst port 80"
NODEPATH="my_node:"
INHOOK="hook1"
MATCHHOOK="hook2"
```

```
NOTMATCHHOOK="hook3"

BPFPROG=$( tcpdump -s 8192 -p -ddd ${PATTERN} | \
           ( read len ; \
             echo -n "bpf_prog_len=$len " ; \
             echo -n "bpf_prog=[" ; \
             while read code jt jf k ; do \
                 echo -n " { code=$code jt=$jt jf=$jf k=$k }" ; \
             done ; \
             echo " ]" ) )

ngctl msg ${NODEPATH} setprogram { thisHook=\"${INHOOK}\" \
  ifMatch=\"${MATCHHOOK}\" \
  ifNotMatch=\"${NOTMATCHHOOK}\" \
  ${BPFPROG} }
```

Based on the previous example, it is possible to prevent a jail (or a VM) from spoofing by allowing only traffic that has the expected ethernet and IP addresses:

```
#!/bin/sh

NODEPATH="my_node:"
JAIL_MAC="0a:00:de:ad:be:ef"
JAIL_IP="128.66.1.42"
JAIL_HOOK="jail"
HOST_HOOK="host"
DEBUG_HOOK="nomatch"

bpf_prog() {
    local PATTERN=$1

    tcpdump -s 8192 -p -ddd ${PATTERN} | (
        read len
        echo -n "bpf_prog_len=$len "
        echo -n "bpf_prog=["
        while read code jt jf k ; do
            echo -n " { code=$code jt=$jt jf=$jf k=$k }"
        done
        echo " ]"
    )
}

# Prevent jail from spoofing (filter packets coming from jail)
ngctl msg ${NODEPATH} setprogram {                    \
    thisHook=\"${JAIL_HOOK}\"                         \
    ifMatch=\"${HOST_HOOK}\"                          \
    ifNotMatch=\"${DEBUG_HOOK}\"                      \
    $(bpf_prog "ether src ${JAIL_MAC} && src ${JAIL_IP}") \
}

# Prevent jail from receiving spoofed packets (filter packets
# coming from host)
ngctl msg ${NODEPATH} setprogram {                    \
    thisHook=\"${HOST_HOOK}\"                         \
    ifMatch=\"${JAIL_HOOK}\"                          \
    ifNotMatch=\"${DEBUG_HOOK}\"                      \
    $(bpf_prog "ether dst ${JAIL_MAC} && dst ${JAIL_IP}") \
}
```

SEE ALSO

bpf(4), netgraph(4), ngctl(8)

HISTORY

The **ng_bpf** node type was implemented in FreeBSD 4.0.

AUTHORS

Archie Cobbs <*archie@FreeBSD.org*>

BUGS

When built as a loadable kernel module, this module includes the file *net/bpf_filter.c*. Although loading the module should fail if *net/bpf_filter.c* already exists in the kernel, currently it does not, and the duplicate copies of the file do not interfere. However, this may change in the future.