ADD

Packet Filter patch
which allow to dynamically add IP address
into tables

This document describe the modifications made to pf and pfctl by the patch ADD

Made by Mr Quentin Narvor with the help of Mr Nicolas Greneche

Table of Contents

I- DESCRIPTION :

The idea behind this patch is to make pf able to dynamically add ip address into tables when it's wanted.
Basically, this will work the same as the "overload" option which already exist in pf, except there will be no conditions to add addresses into desired tables.
It can be used for making pf rules become more accurate : specific rules are applied on packets from ip addresses in some tables, and pf can fill in these table if a specific event happen.
In this patch, only pf and pfctl have been modified.

I-1 How it works :

A new parameter is now possible for rules : add.
If this parameter is used, it have to be added at the end of the normal rule, after all normal and optionals parameters.

It have to be followed by at least one of these options:
- ipsrc <src_tblname>
- ipdst <dst_tblname>

The order these options are written doesn't matter but it's not possible to use each twice in a rule.

If the option "ipsrc <src_tblname>" is present, source IP addresses of all packets which match the rule will be added to the table named src_tblname.
If the option "ipdst <dst_tblname>" is present, destination IP addresses of all packets which match the rule will be added to the table named dst_tblname.
Tables in parameters of ipsrc and ipdst can be the same.

I-2 Examples :

(1) pass in on em0 from 192.168.0.0/16 to any add ipdst <table>

In this rule, all destinations addresses of all packets coming on em0 from the subnet 192.168.0.0/16 will be added to the table named table.

(2) pass in on em0 from 192.168.1.0/24 to 192.168.2.0/24 add ipsrc <table> ipdst <other_table>

In this rule, sources ip addresses of all packets coming on em0 from the subnet 192.168.1.0/24 to the subnet 192.168.2.0/24 will be added to the table named table. In the same way, destinations ip addresses will be added to the table named other_table.

I-3 Installation :

patch the source with the file add.patch :

```
patch -p1 -d /usr/src < add.patch
```

Don't forget to copy the patched pfvar.h from the pf source directory in the /usr/include/net/ directory : it will be useful to compile the new pfctl.

II- Changes on PF :

Modified files for pf :

- pfvar.h
- pf.c
- pf_ioctl.c

II-1 File pfvar.h :

File situated in two places :
 - in the pf sources directory
 - in the include directory : /usr/include/net/

The second one is used to compile pfctl.

In this file, the only modification is the addition of a new structure named add in the current structure pf_rule :

```
struct add{
        u_int8_t check_add;
        struct {
                u_int8_t                check_src;
                char                    src_tblname[PF_TABLE_NAME_SIZE];
                struct pfr_ktable       *src_tbl;
        } src;
        struct {
                u_int8_t                check_dst;
                char                    dst_tblname[PF_TABLE_NAME_SIZE];
                struct pfr_ktable       *dst_tbl;
        } dst;
} add;
```

This structure is used to save data that the parameter add need.
It has "int check_add" which is used to check whether add parameter is in the rule or not and two similar s structures src and dst.
These structure contain respectively one int to check whether ipsrc/ipdst is in the rule or not, the name of the table and the table itself.

II-2 File pf.c:

In this file, one function have been added, and four calls to it.
Calls have been added in functions pf_test_tcp(), pf_test_udp(), pf_test_icmp() and pf_test_other().

The reason why calls are made in these functions and not in pf_test() and pf_test6() is because pf tracks connections.
The four functions above are just called with the first packet of a connection. After, with connection tracking, all the others packets of the same connection will be processed just by functions pf_test() or pf_test6().

Because of that, a packet of the same connection will pass through pf faster than the first packet.
In addition, one single rule "pass in" is enough to allow traffic of the same connection in both direction (in and out) instead of one rule "pass in" and one rule "pass out" unless the presence of the parameter no state.

But, if the parameter add is in the rule, both source address and destination address will be added in the tables because one single rule will match with packets of the same connection in both directions.

Here is the call which is made in the fours functions :

```
if (r->add.check_add != 0) {
        pf_save_addrs (&r, pd);
}
```

Here is the function which have been added.
It takes as parameters one pointer on a pf_rule structure and a pointer on a pf_pdesc structure :

```
void pf_save_addrs( struct pf_rule **rule, struct pf_pdesc *pd)
{
        if ((*rule)->add.check_add != 0) {

                struct pf_src_node sn;
                struct pf_src_node sn2;

                PF_ACPY(&sn.addr, pd->src, pd->af);
                PF_ACPY(&sn2.addr, pd->dst, pd->af);
                sn.af = pd->af;
                sn.af = pd->af;

                struct pfr_addr p;
                struct pfr_addr p2;
                bzero(&p, sizeof(p));
                bzero(&p2, sizeof(p2));
                p.pfra_af = pd->af;
                p2.pfra_af = pd->af;

                switch (pd->af) {

#ifdef INET
                case AF_INET:
                        p.pfra_net=32;
                        p2.pfra_net=32;
                        p.pfra_ip4addr = sn.addr.v4;
                        p2.pfra_ip4addr = sn2.addr.v4;
                        break;
#endif

#ifdef INET6
                case AF_INET6:
                        p.pfra_net=128;
                        p2.pfra_net=128;
                        p.pfra_ip6addr = sn.addr.v6;
                        p2.pfra_ip6addr = sn2.addr.v6;
```

```
                                        break;
#endif
                        }

                        if ((*rule)->add.src.check_src != 0) {
                                pfr_insert_kentry((*rule)->add.src.src_tbl, &p, time_second);
                        }

                        if ((*rule)->add.dst.check_dst != 0) {
                                pfr_insert_kentry((*rule)->add.dst.dst_tbl, &p2, time_second);
                        }
            }
}
```

This function extracts the source and the destination addresses of the packet from the pf_pdesc structure and add them into tables specified on the pf_rule structure.

II-3 File pf_ioctl.c :

In this file, table management for the new tables have been by adapting the current code used by the overload parameter.

Here is the code which have been added :
In the function pf_rm_rule :

```
if (rule->add.src.src_tbl)
            pfr_detach_table(rule->add.src.src_tbl);
if (rule->add.dst.dst_tbl)
            pfr_detach_table(rule->add.dst.dst_tbl);

and

if (rule->add.src.src_tbl)
            pfr_detach_table(rule->add.src.src_tbl);
if (rule->add.dst.dst_tbl)
            pfr_detach_table(rule->add.dst.dst_tbl);


In the pfioctl() function case DIOCADDRULE :

if (rule->add.src.check_src != 0) {
            if ((rule->add.src.src_tbl = pfr_attach_table(ruleset,
                rule->add.src.src_tblname)) == NULL) {
                        error = EINVAL;
            }
            else {
                        rule->add.src.src_tbl->pfrkt_flags |=
                            PFR_TFLAG_ACTIVE;
            }
}
```

```
if (rule->add.dst.check_dst != 0) {
        if ((rule->add.dst.dst_tbl = pfr_attach_table(ruleset,
            rule->add.dst.dst_tblname)) == NULL)
                error = EINVAL;
        else
                rule->add.dst.dst_tbl->pfrkt_flags |=
                    PFR_TFLAG_ACTIVE;
}
```

In the pfioctl() function case DIOCCHANGERULE :

```
if (newrule->add.src.check_src != 0) {
        if ((newrule->add.src.src_tbl = pfr_attach_table(
            ruleset, newrule->add.src.src_tblname)) ==
            NULL) {
                error = EINVAL;
        }
        else {
                newrule->add.src.src_tbl->pfrkt_flags |=
                    PFR_TFLAG_ACTIVE;
        }
}

if (newrule->add.dst.check_dst != 0) {
        if ((newrule->add.dst.dst_tbl = pfr_attach_table(
            ruleset, newrule->add.dst.dst_tblname)) ==
            NULL)
                error = EINVAL;
        else
                newrule->add.dst.dst_tbl->pfrkt_flags |=
                    PFR_TFLAG_ACTIVE;
}
```

III- Changes on PFCTL :

Modified files for pfctl :

- parse.y
- pfctl_optimize.c
- pfctl_parser.c

III-1 File parse.y :

The first change on this file is the addition of a new structure node_add_opt, which contains the same data as the structure "add" in the structure "pf_rule" in the file pfvar.h except tables themselves.

```
struct node_add_opt {
        u_int8_t                check_add;
        struct {
           u_int8_t     check_src;
           char  src_tblname[PF_TABLE_NAME_SIZE];
        } src;
        struct {
           u_int8_t     check_dst;
           char  dst_tblname[PF_TABLE_NAME_SIZE];
        } dst;
        struct node_add_opt     *next;
        struct node_add_opt     *tail;
};
```

This structure is added in the structure "v" used by yacc :

```
struct node_add_opt     *add_opt;
```

Tokens ADD, IPSRC and IPDST are added to the yacc configuration file as well as types add_opt, add_list, and add_item, related to the structure v.add_opt.

Here is the definition of the three types mentioned above:

```
add_opt         :       /* empty */ {
                $$ = calloc(1, sizeof(struct node_add_opt));
                $$->check_add = 0;
                $$->src.check_src = 0;
                $$->dst.check_dst = 0;
                $$->next = NULL;
                $$->tail = $$;
                }
                | ADD add_list { $$ = $2; }
                ;
```

Type add_opt is at the root of the parameter in the rule.
It is either empty : the add parameter isn't in the rule,
or the token ADD is in the rule : then are following add options (ipsrc and ipdst), symbolized by the type

add_list.

```
add_list        : add_item { $$ = $1; }
                | add_list add_item {
                        $1->tail->next = $2;
                        $1->tail = $2;
                        $$ = $1;
                }
                ;
```

Type add_list show how are written the options of the parameter add : either just one option or a list of them, these part is very similar to the overload one.

```
add_item        : IPSRC '<'STRING'>' {
                if (strlen($3) >= PF_TABLE_NAME_SIZE) {
                        yyerror("table name '%s' too long", $3);
                        free($3);
                        YYERROR;
                }
                $$=calloc(1, sizeof(struct node_add_opt));
                if ($$ == NULL) {
                        err(1, "state_opt_item: calloc");
                }
                if (strlcpy($$->src.src_tblname, $3,
                    PF_TABLE_NAME_SIZE) >= PF_TABLE_NAME_SIZE) {
                        yyerror("state option: "
                            "strlcpy");
                        YYERROR;
                }
                free($3);
                $$->check_add = 1;
                $$->src.check_src = 1;
                $$->next = NULL;
                $$->tail = $$;
                }
                | IPDST '<'STRING'>' {
                if (strlen($3) >= PF_TABLE_NAME_SIZE) {
                        yyerror("table name '%s' too long", $3);
                        free($3);
                        YYERROR;
                }
                $$=calloc(1, sizeof(struct node_add_opt));
                if ($$ == NULL) {
                        err(1, "state_opt_item: calloc");
                }
                if (strlcpy($$->dst.dst_tblname, $3,
                    PF_TABLE_NAME_SIZE) >= PF_TABLE_NAME_SIZE) {
                        yyerror("state option: "
                            "strlcpy");
                        YYERROR;
                }
                free($3);
                $$->check_add = 2;
                $$->dst.check_dst = 1;
                $$->next = NULL;
                $$->tail = $$;
                }
                ;
```

Type add_item contains the list of the possible options of the add parameter.
There is just two of them :
 - IPSRC '<'STRING'>' which match to ipsrc <src_tblname> in pf.conf.
 - IPDST '<'STRING'>' which match to ipdst <dst_tblname> in pf.conf.

The current type pfrule as been changed to make pfctl understand the new possible parameter in the rule :

```
pfrule          : action dir logquick interface route af proto fromto
                  filter_opts add_opt
```

Plus some code have been added :

```
a = $10;
while (a) {
        switch (a->check_add) {
        case 0 :
                r.add.check_add = 0;
                r.add.src.check_src = 0;
                r.add.dst.check_dst = 0;
                break;

        case 1 :
                r.add.check_add = 1;
                r.add.src.check_src = 1;
                if (r.add.src.src_tblname[0]) {
                yyerror("multiple 'src' "
                    "table definitions");
                YYERROR;
                }
                if (strlcpy(r.add.src.src_tblname,
                    a->src.src_tblname,
                    PF_TABLE_NAME_SIZE) >=
                    PF_TABLE_NAME_SIZE) {
                        yyerror("state option: "
                            "strlcpy");
                        YYERROR;
                 }
                break;

        case 2 :
                r.add.check_add = 1;
                r.add.dst.check_dst = 1;
                if (r.add.dst.dst_tblname[0]) {
                yyerror("multiple 'dst' "
                    "table definitions");
                YYERROR;
                }
                if (strlcpy(r.add.dst.dst_tblname,
                    a->dst.dst_tblname,
                    PF_TABLE_NAME_SIZE) >=
                    PF_TABLE_NAME_SIZE) {
                        yyerror("state option: "
                            "strlcpy");
                        YYERROR;
                 }
                break;
        }
        a = a->next;
}
```

In this code, the structure pf_rule used by pf is filled in by the data in the node_add_opt structure allocated by the type described above.

These keywords are also added to the yacc keywords list :

```
{ "add",                ADD},
{ "ipdst",              IPDST},
{ "ipsrc",              IPSRC},
```

III-2 File pfctl_optimize.c :

Only types PF_RULE_FIELD related to the add parameter have been added to this file :

```
PF_RULE_FIELD(add.src.src_tblname,          NOMERGE),
PF_RULE_FIELD(add.dst.dst_tblname,          NOMERGE),

PF_RULE_FIELD(add.src.src_tbl,      DC),
PF_RULE_FIELD(add.dst.dst_tbl,      DC),
```

III-3 File pfctl_parser.c :

The writing rule of the add parameter has been added in the function print_rule() :

```
if (r->add.check_add != 0) {
                    if (r->add.src.check_src != 0 && r->add.dst.check_dst == 0)
                            printf("add ipsrc <%s>", r->add.src.src_tblname);
                    if (r->add.src.check_src == 0 && r->add.dst.check_dst != 0)
                            printf("add ipdst <%s>", r->add.dst.dst_tblname);
                    if (r->add.src.check_src != 0 && r->add.dst.check_dst != 0)
                            printf("add ipsrc <%s> ipdst <%s>", r->add.src.src_tblname, r->add.dst.dst_tblname);
}
```