IP-Address Reflection Scheme Implemetation for Linux

Mike Krinkin
St Petersburg Academic University
Saint-Petersburg, Russia
krinkin.m.u@gmail.com

Kirill Krinkin
Open Source and Linux Lab
Saint-Petersburg, Russia
kirill.krinkin@fruct.org

Abstract—**In storage systems, designed to provide required quality of service, hight performance and data flows isolation, abilities of existing routing techniques are not enough. For example, Linux host with number of network interfaces makes decision about which one would be used to reply clients based only on routing table, so there is no obvious way to hold a specific gateway per client. So aim of our study is to design and implement gateway hold technique in Linux kernel.**

Solution called IPREFLECT implemented as Linux kernel loadable module and uses Netfilter API to embed in Linux network stack. Netfilter API provides five points of embedding:

- `NF_INET_PRE_ROUTING` ,
- `NF_INET_LOCAL_IN` ,
- `NF_INET_FORWARD` ,
- `NF_INET_LOCAL_OUT` ,
- `NF_INET_POST_ROUTING` .

Netfilter API allow to register custom callback function for any of this points. IPREFLECT uses two - `NF_INET_LOCAL_IN` **and** `PNF_INET_LOCAL_OUT` **, because only localy delivered traffic does matter for IPREFLECT. Callabck signature is next:**

```
unsigned int callback(
    struct nf_hook_ops const *ops,
    struct sk_buff *skb,
    struct net_device const *in,
    struct net_device const *out,
    int (*okfn)(struct sk_buff *)
);
```

Most important parameters are: `skb` , `in` **and** `out` . `struct sk_buff` **represents network packet and holds almost all related information (socket, refcounters and so on) in Linux.** `struct net_device` **represents network interface in Linux kernel,** `in` **or** `out` **may be** `NULL` **, for output or input packets, respectively.**

To register callback it is needed to fill `struct nf_hook_ops` **and call to** `nf_register_hook` **with pointer to the struct. In IPREFLECT we use next code to fill structures:**

```
static struct nf_hook_ops hwaddr_in_hook = {
    .hook = hwaddr_in_hook_fn,
    .owner = THIS_MODULE,
    .pf = NFPROTO_IPV4,
    .hooknum = NF_INET_LOCAL_IN,
    .priority = NF_IP_PRI_LAST
};
```

```
static struct nf_hook_ops hwaddr_out_hook = {
    .hook = hwaddr_out_hook_fn,
    .owner = THIS_MODULE,
    .pf = NFPROTO_IPV4,
    .hooknum = NF_INET_LOCAL_OUT,
    .priority = NF_IP_PRI_LAST
};
```

In `NF_INET_LOCAL_IN` **, after performing some sanity checks, IPREFLECT extracts next information from packet:**

- **client IP,**
- **gateway MAC address.**

Extracted information saved in hashtable for later usage. Hashtable entry is shown below.

```
struct hwaddr_entry
{
    struct hlist_node  node;
    atomic_t           refcnt;
    __be32             remote;
    rwlock_t           lock;
    unsigned int       ha_len;
    unsigned char      ha[MAX_ADDR_LEN];
};
```

It is worth to note, that Linux kernel has IP address spoofing protection mechanism called reverse path filtering. If reverse path filtering enabled kernel drops all packets from unknown hosts or packets received through wrong interface, so it is necessary to disable filtering.

In `NF_INET_LOCAL_OUT` **IPREFLECT checks packet route, and looks for new route through appropriate network interface if needed.**

Then it checks if arp entry for specified client IP doesn't exists. If so IPREFLECT creates fake arp entry with client IP address. Then it changes new arp entry state to `NUD_NOARP` **and sets entry MAC address to saved gateway address to prevent MAC address resolving.**

Control reaches `NF_INET_LOCAL_OUT` **point only if network stack found some route for packet. So some kind of default route should exist to guarantee correct work.**

Thus service host can reply clients through appropriate network interface and gateway without any defined routes to client. So in case of infrastructure changes service host need no reconfiguration.