시스템 프로그래밍

2차과제

Netfilter를 이용한 packet forwarding / monitoring

15조

2016320272 권영진

2016320265 박현종

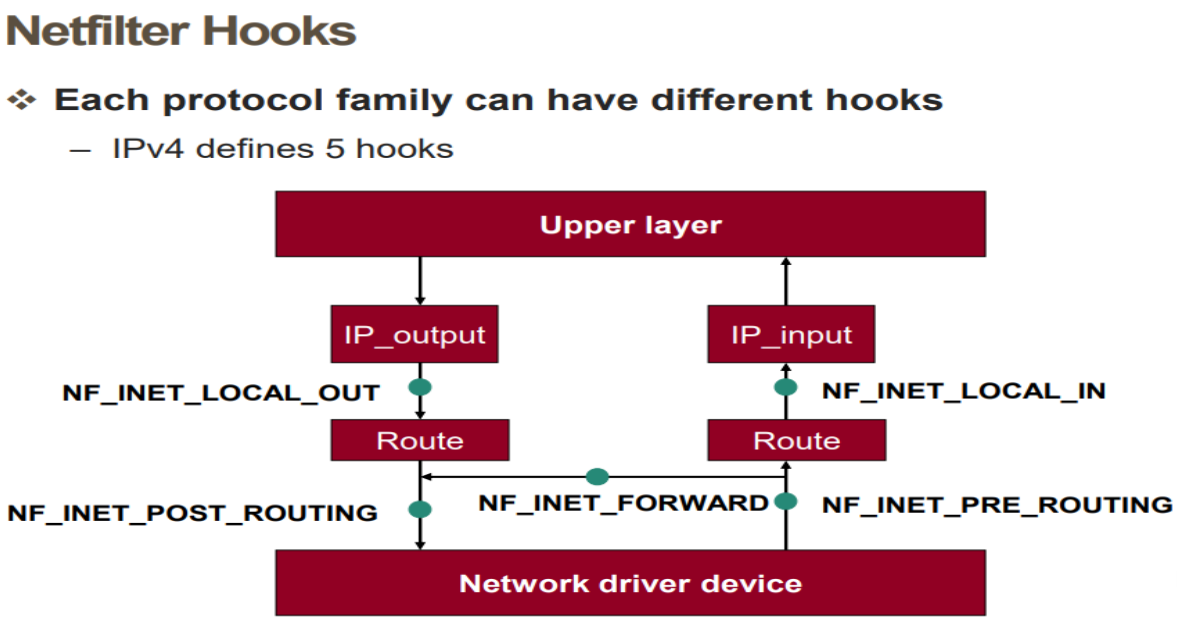
Free Day 사용일수 : 2일

목차

1. 배경지식

Netfilter는 리눅스 커널에서의 네트워크 프레임워크다. 기본적인 역할은 패킷 필터링, 네트워크 주소 변환 및 포트 변환에 대한 다양한 기능과 작업을 제공하여 네트워크를 통해 패킷을 전달하고 패킷이 네트워크 내 특정 위치에 도달하지 못하도록 하는 기능을 제공한다. 즉 일반적인 소켓 프로그래밍으로는 패킷을 감시(monitoring)할 수는 있지만, 패킷을 변조하거나 버리는 것은 할 수 없기 때문에 넷필터를 이용한다.

Hooking이란 이미 작성되어 있는 코드의 특정 지점을 가로채서 동작 방식에 변화를 주는 기술이다. Hook의 원래 의미는 낚시바늘과 같은 갈고리로서 코드의 중간 부분을 낚아채는 의미라고 생각하면 된다. 즉 hook이란 메시지가 목표 지점에 전달되기 전에 가로채는 특수한 덫, 프로시저이다. netfilter는 앞서 말한 네트워크 관련 연산들을 핸들러의 형태로 구현할 수 있도록 Hook을 제공하는데 크게 다섯 가지 hook을 정의한다. hook이란 패킷 경로에 정의된 포인트로서 네크워크 스택을 통과할 때 특정 hook point에서 함수가 호출되어 패킷을 드랍 하거나 변조할 수 있다. Hook들의 이름과 시점은 다음과 같다.



- NF_IP_PRE_ROUTING : 패킷이 네트워크 스택에 들어온 바로 직후, 패킷을 보낼 위치와 관련하여 라우팅 결정하기 이전에 처리 된다. 즉 패킷이 들어오고 라우팅을 해야 하는데 그 전 시점에 이 Hook이 호출 되어서 결정을 하게 된다.

- NF_IP_LOCAL_IN : 만약 패킷이 로컬 시스템으로 목적지를 가지게 될 경우 패킷이 해당 목적지로 라우팅 되고 난 이후에 트리거 된다.

- NF_IP_FORWARD : 만약 패킷이 다른 호스트로 포워딩 될 경우에 패킷이 해당 호스트로 라우팅 되고 난 이후 트리거 된다.

- NF_IP_LOCAL_OUT : 네트워크 스택에 도달하자 마자, 로컬에 생성되어진 아웃 바운드 트래픽에 의해 트리거 된다.

- NF_IP_POST_ROUTING : 라우팅이 수행된 후 실제 네트워크에 전송되기 직전에 나가거나 포워딩 된 트래픽에 의해 트리거 된다.

위와 같은 각 hook point에 nf_hook_ops를 등록,해제 할 때 Hook point에는 여러 함수들이 동시에 등록될 수 있는데 그렇기 때문에 함수의 우선순위를 정해 주어야한다. 또한 각 hook function에 리턴 값을 지정해 줌으로써 패킷을 drop하거나 accept할 수 있다. 이후 작성한 소스코드는 컴파일과 함께 작성한 모듈을 실행 하면 된다. 실제 소스는 다음과 같다.

Netfilter implementation

❖ 후킹 함수 구조

include/linux/netfilter.h

```
83 typedef unsigned int nf_hookfn(void *priv,
84                               struct sk_buff *skb,
85                               const struct nf_hook_state *state);
```

– skb : a packet

❖ 후킹 함수의 return value

include/uapi/linux/netfilter.h

```
10 /* Responses from hook functions. */
11 #define NF_DROP 0
12 #define NF_ACCEPT 1
13 #define NF_STOLEN 2
14 #define NF_QUEUE 3
15 #define NF_REPEAT 4
16 #define NF_STOP 5
```

– NF_DROP : 현재 패킷을 Drop

– NF_ACCEPT : 현재 패킷을 다음 루틴으로 넘김

Netfilter implementation

❖ 후킹 포인트들 (후킹 함수를 등록하는 지점)

include/uapi/linux/netfilter.h

```
46 enum nf_inet_hooks {
47         NF_INET_PRE_ROUTING,
48         NF_INET_LOCAL_IN,
49         NF_INET_FORWARD,
50         NF_INET_LOCAL_OUT,
51         NF_INET_POST_ROUTING,
52         NF_INET_NUMHOOKS          /* NF_INET_NUMHOOKS는 enum의 마지막 인덱
                                     스로, 후킹 포인트 개수를 나타내기 위함 */
53 };
```

nf_hook_ops 구조체의 hooknum 필드값

❖ Hooking priorities

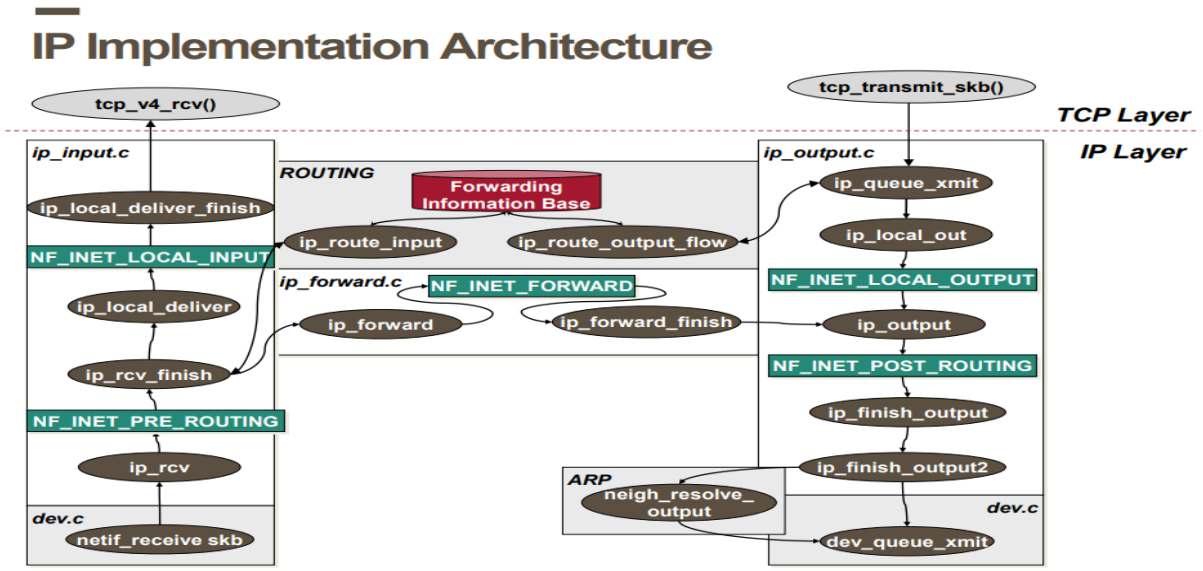
include/uapi/linux/netfilter_ipv4.h

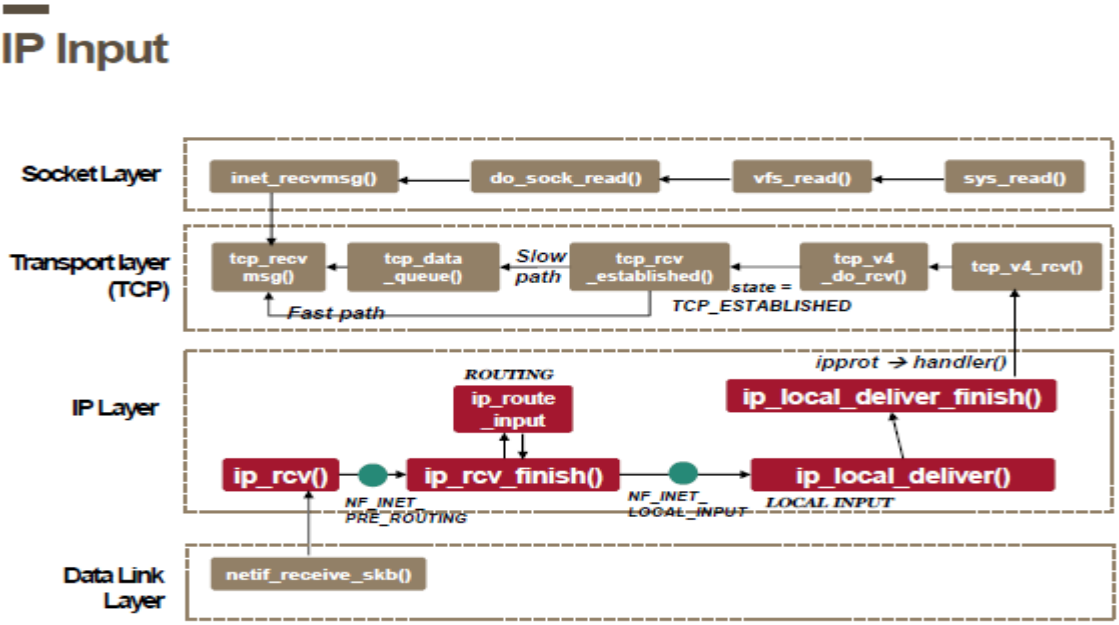
```
57 enum nf_ip_hook_priorities {
58         NF_IP_PRI_FIRST = INT_MIN,
59         NF_IP_PRI_CONNTRACK_DEFRAG = -400,
...
...
71         NF_IP_PRI_LAST = INT_MAX,
72 };
```

nf_hook_ops 구조체의 priority필드값

참고 : ditialocean community <A Deep Dive into Iptables and Netfilter Architecture>

2. 커널 레벨 네트워킹 분석





- ip_rcv()

무결성(sanity)을 확인한다. 헤더와 데이터를 구분하여 header size, length, check sum등을 확인한다. 무결성 확인 후 NF_INET_PRE_ROUTING 훅 포인트를 지나가면서 함수콜이 될 수 있으며 이후 ip_rcv_finish() 를 호출한다.

- ip_rcv_finish()

라우팅 캐시로부터 최종 목적지를 찾는다. 이 목적지를 비교하여 로컬(자기 자신의 주소)인 경우에는 ip_local_deliver()가 호출된다. 이외에도 유니캐스트 포워딩일 경우에는 ip_forward(), 멀티

캐스트 포워딩일 경우에는 ip_mr_input()을 호출한다.

- ip_local_deliver()

ip_rcv_finish()에서 목적지가 로컬 주소였을 때 실행되는 함수이다. 만일 들어온 IP 헤더가 패킷이 파편화(fragment)된 상태라면 이들을 합친다.(reassemble). 이후 NF_INET_LOCAL_INPUT 훅 포인트를 거쳐 ip_local_deliver_finish()를 호출한다.

- ip_local_deliver_finish()

조각화되지 않은 패킷이 수신되었거나 재조립이 완료되었을 때 ip_local_deliver()에 의해 간접적으로 호출되는 확인용 함수이다. 소켓버퍼로부터 IP 헤더를 제거하고 IP 헤더의 프로토콜 필드(iphdr->protocol)에 맞게 TCP와 같은 상위 계층의 핸들러를 호출한다. 만일 일치하는 프로토콜이 없다면 드랍 하고 Destination Unreachable 옵션으로 ICMP 메시지를 보낸다.

- ip_forward()

ip_rev_finish()에서 패킷의 최종목적지가 자신의 주소가 아닐 경우에 포워딩을 위하여 호출되는 함수이다. 우선 최대 길이가 MTU(Maximum Transfer Unit)을 초과하였거나 수명이 종료 되었는지 확인을 하고 L2 layer를 위한 공간을 만든다. 이후 NF_INET_FORWARDING 훅 포인트를 거쳐 ip_forward_finish() 함수를 호출한다.

- ip_forward_finish()

IP 옵션이 있으면 옵션들을 처리해주고(ip_foward_options) ip_output()을 호출한다.

- ip_queue_xmit()

전달 받은 패킷이 이미 라우팅 되있지 않다면 이 함수를 통해서 소켓에 헤더를 위한 공간을 만들고 헤더를 초기화한다. 이후 ip_local_out()을 호출한다.

- ip_local_out()

체크섬을 계산하고 NF_INET_LOCAL_OUTPUT 훅을 호출한다. 목적지가 자신의 host일 경우에는 ip_local_deliver()를 호출하고 아니라면 ip_output()을 호출한다.

- ip_output()

이 패킷 전송에 사용할 장치를 지정 하는 등 소켓 버퍼를 업데이트를 한다. 이후 NF_INET_POST_ROUTING을 호출하고 완료되면 ip_finish_output()을 호출한다.

3. 소스 코드 분석

서버와 통신하는 코드는 warmup에 있는 **clinet.c**를 그대로 사용하였고 추가적으로 lkm module 인 **hw.c**를 작성하였다.

```
unsigned int server_port[5]={33333,4444,5555,6666,7777};



unsigned int inet_addr(char *str)
{
    int i,j,k,l;
    char arr[4];
    sscanf(str,"%d.%d.%d.%d",&i,&j,&k,&l);
    arr[0]=i;
    arr[1]=j;
    arr[2]=k;
    arr[3]=l;

    return *(unsigned int *)arr;
}

static unsigned int pre_my_hook_fn(void *priv,
                                   struct sk_buff *skb,
                                   const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    if(!skb) return NF_ACCEPT;

    iph=ip_hdr(skb);

    unsigned int sp,dp;
    char saddr[16],daddr[16];
```

```
    tcph=tcp_hdr(skb);
    sp=htons((unsigned short int)tcph->source);
    dp=htons((unsigned short int)tcph->dest);
    snprintf(saddr,16,"%pI4",&iph->saddr);
    snprintf(daddr,16,"%pI4",&iph->daddr);

    printk(KERN_ALERT "PRE_ROUTING : %u,%5d,%5d,%s,%s\n",
                                    iph->protocol,sp,dp,saddr,daddr);

    if(sp==PORT)
    {
        iph->daddr=inet_addr("192.168.101.10");
        tcph->source=ntohs((unsigned short int) 7777);
        tcph->dest = ntohs((unsigned short int) 7777);
    }


    return NF_ACCEPT;
}
```

pre_my_hook_fn 함수는 hooking point에 등록하는 콜백함수이다. 이 함수에서는 ip_hdr 과 tcp_hdr 함수를 사용해서 ip header와 tcp header의 정보를 가져온다. 패킷의 **ip protocol , 출발포트 , 도착포트 , 출발주소 , 도착주소**를 순서대로 출력을 해주고 포워딩하려는 포트인 **33333**인 포트를 만나면 포트번호를 **7777**로 바꿔주고 주소를 포워딩하려는 주소인 **"192.168.101.10"**로 바꿔준다.

```
static unsigned int post_my_hook_fn(void *priv,
        struct sk_buff *skb,const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    if(!skb) return NF_ACCEPT;

    iph=ip_hdr(skb);


    unsigned int sp,dp;
    char saddr[16],daddr[16];

    tcph=tcp_hdr(skb);
    sp=htons((unsigned short int) tcph->source);
    dp=htons((unsigned short int) tcph->dest);
    snprintf(saddr,16,"%pI4",&iph->saddr);
    snprintf(daddr,16,"%pI4",&iph->daddr);

    printk(KERN_ALERT "FORWARD: %u,%5d,%5d,%s,%s\n",iph->protocol,
                                sp,dp,saddr,daddr);


    return NF_ACCEPT;
}
```

post_my_hook_fn 함수도 hooking point에 등록하는 콜백함수이다. 이 함수에서는 단순히 패킷의 정보를 출력해주기만 한다.

```
static struct nf_hook_ops pre_my_nf_ops={
    .hook=pre_my_hook_fn,
    .hooknum = NF_INET_PRE_ROUTING,
    .pf=PF_INET,
    .priority=NF_IP_PRI_FIRST,

};

static struct nf_hook_ops post_my_nf_ops={
    .hook=post_my_hook_fn,
    .hooknum=NF_INET_FORWARD,
    .pf=PF_INET,
    .priority=NF_IP_PRI_FIRST,
};




static const struct file_operations proc_fops={
    .owner = THIS_MODULE,

};
```

앞에서 작성한 두 개의 함수는 각각 **NF_INET_PRE_ROUTING** 과 **NF_INET_FORWARD**인 hooking point에 등록된다. **NF_INET_PRE_ROUTING**에 **pre_my_hook_fn** 함수를 등록하는 이유는 forward 할지를 결정하는 부분이기 때문이고 **NF_INET_FORWARD**에서는 포워딩된 패킷을 확인할수있기때문에 이곳이 **post_my_hook_fn** 함수를 등록한다.

4. 실행방법 및 결과 분석

4-1. 실행방법

첫번째로 포워딩할 곳인 **"192.168.101.10"**을 라우팅테이블에 등록한다.

Route add –net 192.168.101.10 netmask 255.255.255.255 dev enp0s3

```
hyun@hyun-VirtualBox:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.101.10  *               255.255.255.255 UH    0      0        0 enp0s3
hyun@hyun-VirtualBox:~$
```

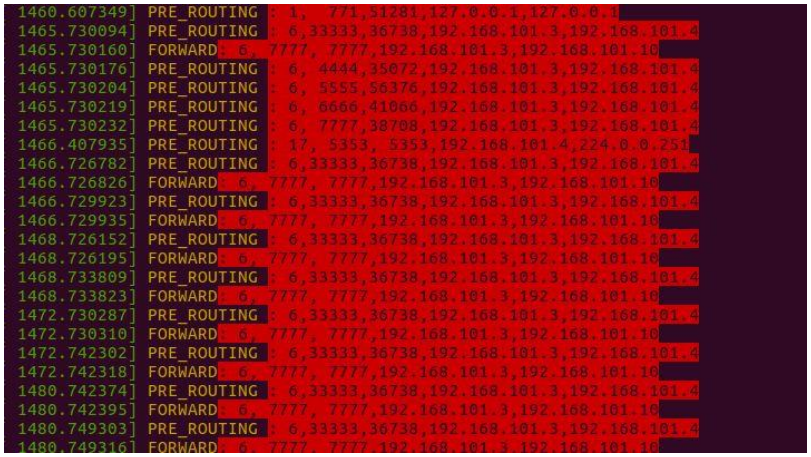
두번째로 포워딩하기위해서는 **/proc/sys/net/ipv4/ip_forward** 에 써있는 값인 0을 1로 바꿔서 포워딩을 활성화시켜야한다.

세번째로 위에서 언급한 lkm 모듈인 **hw2.ko**를 **insmod** 하고 warmup에서 진행했었던 스레드를 이용한 소켓통신을 한다. 이때 포트 33333 , 4444 , 5555 , 6666 , 7777 를 사용한다.

4-2. 결과분석

dmesg로 lkm 모듈에서 사용한 **KERN_ALERT** 메시지를 확인한다.

pre_routing 부분에서 **33333**포트를 확인할수있고 forward 부분에서 **7777**로 바뀐 것을 확인할수 있다.



5. 과제 수행 시의 Trouble과 Troubleshooting 과정

- 커널을 부팅하는 과정에서 **grub**에서 error가 발생해서 커널을 다시 깔아야했다.

- **/proc/sys/net/ipv4/ip_forward** 파일을 0에서 1로 바꾸는 과정에서 처음에 chown과 chmod를 사용했는데 파일을 수정할수없어서 root로 login해서 바꾸는 방법을 찾았다.

- string 형식으로 된 ip주소를 네트워크가 요구하는 형식으로 바꾸는 과정에서 다른 사람들의 코드를 참조해서 바꾸었다.