FreeDay 사용일수 0일

2019F 시스템 프로그래밍

과제2: netfilter를 이용한 packet forwarding / monitoring

Group 25

2014210042 한 윤
2014210055 김 민섭

제출일자 2019년 12월 14일

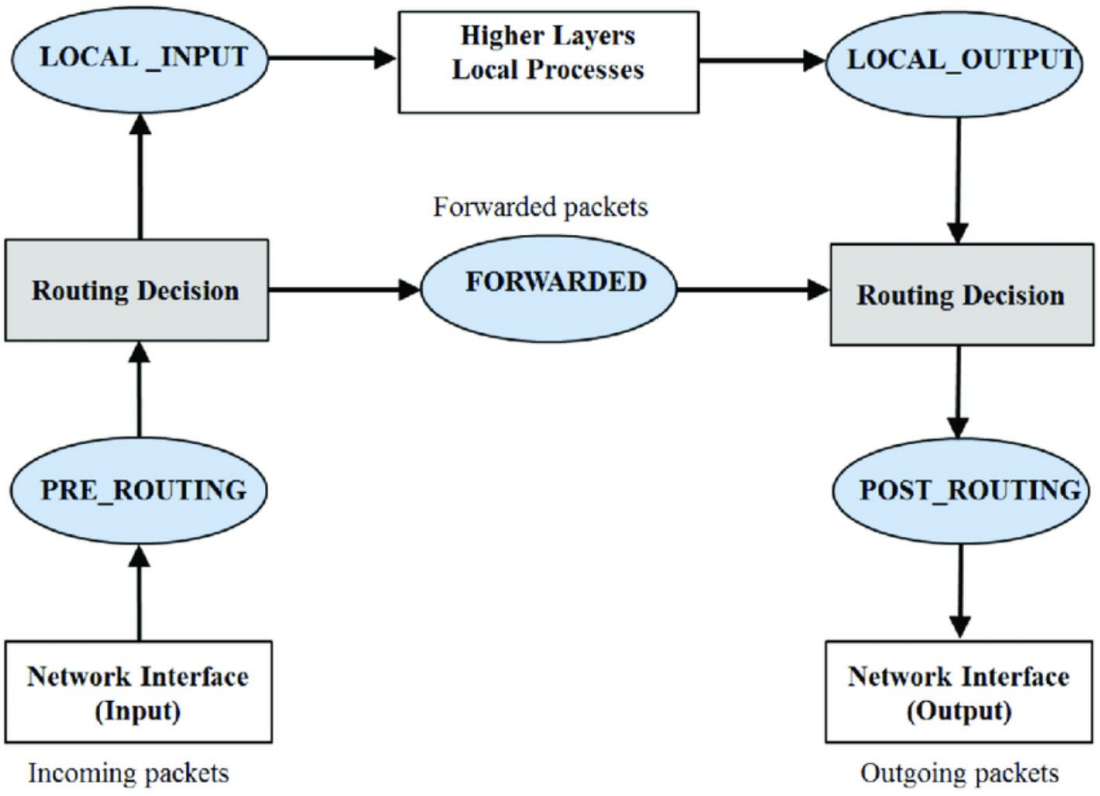
목차

1. Netfilter 및 Hooking에 대한 설명

1.1 Netfilter

Netfilter는 리눅스 커널에 있는 네트워크 관련 프레임워크로 네트워크 주소 변환, 패킷 필터링, 방화벽 설정, 패킷 mangling(패킷을 토막내는 일)을 용이하게 해준다. 또 네트워크 관련 연산을 핸들러 형태로 구현할 수 있도록 Hook을 제공한다. 이 훅은 패킷이 네트워크를 순회할때 그 경로에 있는 정의된 포인터를 의미한다. 또 리눅스에서 방화벽으로 사용되는 iptables도 넷필터 훅과 연동해서 동작을한다.

1.2 Hooking



넷필터 훅 과 패킷 흐름

후킹은 함수호출, 메세지, 이벤트 등을 중간에서 바꾸거나 가로채는 것을 말한다. 그리고 후킹을 처리하는 코드를 훅이라고 한다. 넷필터에서도 이러한 훅을 5개 가지고 있다. 패킷들이 들어오면 훅에 등록된 커널 모듈들이 Trigger된다.

NF_INET_PRE_ROUTING: 이 훅은 패킷이 맨 처음에 들어온뒤 라우팅되기 이전에 트리거 된다.
NF_INET_LOCAL_IN: 패킷의 목적지가 로컬일 경우 라우팅되고 트리거 된다.
NF_INET_FORWARD: 패킷의 목적지가 다른 호스트일 경우 라우팅 되고 트리거 된다.
NF_INET_LOCAL_OUT: 로컬에서 생성된 패킷에 의해 트리거 된다.
NF_INET_POST_ROUTING: 실제 네트워크에 전송전에 라우팅된 패킷에 의해 트리거 된다.

2. 커널레벨 네트워킹 코드 분석

ip_rcv(): ip header의 사이즈, 버전, checksum 등을 체크한다. 그런뒤에 NF_INET_PRE_ROUTING 훅과 ip_rcv_finish를 호출한다.

ip_rcv_finish(): 패킷을 리눅스 네트워크상에서 처리하기위해 virtual path cache를 초기화한다. 최종 목적지가 자기 자신이면 **ip_local_deliver()** 가 호출된다 **ip_local_deliver()**가 실행되면 ip_is_fragment에 의해 패킷이 fragment 상태라면 NF_INET_LOCAL_IN 훅과 **ip_local_deliver_finish()**가 호출된다. **ip_local_deliver()**가 호출되면 tcp 헤더정보를 찾아서 상위 프로토콜로 넘겨준다.

만약 ip_rcv_finish()에서 패킷의 최종목적지가 자신이 아니라 자신을 거쳐 다른 곳에 보내지면 **ip_forward()**가 호출된다.
ip_forward()가 호출되면 패킷의 목적지가 호스트인지 등 유효성검사를 하고 TTL을 감소시킨다. 그런뒤 NF_INET_FORWARD 훅과 **ip_forward_finish()**함수를 호출한다.**ip_forward_finish()**에서는 ip options이 있으면 **ip_forward_options()**를 호출해 처리한다.

ip_queue_xmit(): 패킷이 이미 라우트되어있으면 이 함수를 스킵하고 아니라면 목적지를 할당하고 헤더를 빌드한다. 그 뒤 ip_local_out()을 호출한다.

ip_local_out(): ip_send_check을 통해 checksum을 생성하고 NF_INET_LOCAL_OUTPUT 훅을 호출한다. 그리고 이때 패킷이

loopback이라면 ip_local_deliver(), 아니면 ip_output()을 호출한다.

ip_output(): sk_buff를 업데이트하고 NF_INET_POST_ROUTING 훅을 호출하고 ip_finish_output을 호출한다.

3. 작성한 소스코드에 대한 설명

이번 과제에서 새로 작성한 소스코드는 한 가지이다. LKM 소스코드만 새로 작성하였고 서버와 통신하는 클라이언트 코드는 warmup 과제 에서 사용했던 코드를 그대로 사용하였다.

LKM module 이 하는 일은 다음과 같다.

1) Module init 시에 nf_hook_ops 를 등록한다.

```
static struct nf_hook_ops pkt_changing_hook = {
    .hooknum = NF_INET_PRE_ROUTING,
    .pf = PF_INET,
    .hook = &pkt_changing,
    .priority = NF_IP_PRI_FIRST,
};

static struct nf_hook_ops pkt_forwarding_hook = {
    .hooknum = NF_INET_FORWARD,
    .pf = PF_INET,
    .hook = &pkt_monitoring,
    .priority = NF_IP_PRI_FIRST,
};

static struct nf_hook_ops pkt_output_hook = {
    .hooknum = NF_INET_POST_ROUTING,
    .pf = PF_INET,
    .hook = &pkt_monitoring,
    .priority = NF_IP_PRI_FIRST,
};
```

pkt_changing_hook 은 PRE_ROUTING hooking 포인트에 pkt_changing callback 함수를 등록하는 구조체이다. PRE_ROUTING hooking 포인트를 **사용하는 이유는 이 지점에서 패킷이 local 로 들어갈지, 다른 곳으로 forward 될지 결정할 수 있는 지점이기 때문**이다.

pkt_forwarding_hook 은 FORWARD hooking 포인트에 pkt_monitoring callback 함수를 등록하는 구조체이다. 이 hooking 포인트에서는 forward 된 패킷을 감지할 수 있다.

pkt_output_hook 은 POST ROUTING hooking 포인트에 pkt_monitoring callback 함수를 등록하는 구조체이다. 이 hooking 포인트에서는 routing 처리가 끝나고 local 에서 나가는 패킷을 감지할 수 있다.

- Hooking Callback 함수 설명 -

pkt_changing 함수의 코드는 다음과 같다.

```
unsigned int pkt_changing(void* priv, struct sk_buff *skb, const struct nf_hook_state *state)
{
    struct iphdr *iph = ip_hdr(skb);
    struct tcphdr *tcph = tcp_hdr(skb);
    unsigned int sport;

    if (!skb) return NF_ACCEPT;

    iph = ip_hdr(skb);
    tcph = tcp_hdr(skb);

    // ntohs is for tr wansforming network byte stream to host byte stream
    printk(KERN_INFO "PRE_ROUTING packet(%u; %d; %d; %d.%d.%d.%d; %d.%d.%d.%d)\n",
        iph->protocol,
        ntohs(tcph->source),
        ntohs(tcph->dest),
        NIPQUAD(iph->saddr),
        NIPQUAD(iph->daddr));

    // change sport number and dport number to 7777
    // if source port number is 3333.
    sport = ntohs(tcph->source);
    if (sport == 33333) {
        tcph->source = htons(7777);
        tcph->dest = htons(7777);
        iph->daddr = inet_addr("10.37.129.5");
        printk(KERN_INFO "[After changing]: %d %d", ntohs(tcph->source), ntohs(tcph->dest));
    }

    return NF_ACCEPT;   // let the packet go through the rest of process.
}
```

ip_hdr와 tcp_hdr 함수를 사용해서 **socket buffer 로부터 각각 ip header 와 tcp header 정보를 가져온다.** 그 다음에 ROUTING 단계로 들어가기 전에 로그를 한 번 출력해주고나서, source port 번호가 33333인 패킷에 한정하여 tcp source port number, destination port number 그리고 ip destination address 를 바꿔준다.

tcp header 에서 source와 dest 멤버 변수를 바꿔준 이유는 tcp header 가 socket의 포트 정보를 담고 있고, 이 포트 번호를 원하는 forwarding 동작에 맞게 바꿔줄 필요가 있기 때문이다. 마찬가지로 **ip header 가 가지고 있는 destination addr(daddr)을 원하는 목적지 주소로 치환**해 줌으로써 패킷이 적절한 주소로 forwarding 되도록 설정할 수 있다.

forwarding 에 필요한 정보를 다 바꿔준 후에는 헤더 정보가 잘 바뀌었는지 디버깅하는 printk 코드를 삽입했다.
pkt_monotoring 함수의 코드는 다음과 같다.

```
unsigned int pkt_monitoring(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);
    tcph = tcp_hdr(skb);

    if (state->hook == 2) {  // NF_INET_FORWARD
        printk(KERN_INFO "FORWARD packet(%u; %d; %d; %d.%d.%d.%d; %d.%d.%d.%d)\n",
            iph->protocol,
            ntohs(tcph->source),
            ntohs(tcph->dest),
            NIPQUAD(iph->saddr),
            NIPQUAD(iph->daddr));
    } else if (state->hook == 4) {   // NF_INET_POST_ROUTING
        printk(KERN_INFO "POST_ROUTING packet(%u; %d; %d; %d.%d.%d.%d; %d.%d.%d.%d)\n",
            iph->protocol,
            ntohs(tcph->source),
            ntohs(tcph->dest),
            NIPQUAD(iph->saddr),
            NIPQUAD(iph->daddr));
    }

    return NF_ACCEPT;
}
```

이 callback 함수는 FORWARD hooking 포인트와 POST ROUTING hooking 포인트 두 곳 모두에 등록되어있다. FORWARD hooking 포인트와 POST ROUTING hooking 포인트에서 수행해야할 로직은 커널에 로그 메시지를 보여주는 것으로 동일하므로, state->hook 에 따라 판단하여 로그 메시지를 분기하는 식으로 처리해주었다. 코드 중복 방지를 위해서이다.

2) 원하는 포트에서 들어온 패킷의 ip header와 tcp header를 nf_hook_ops에 등록된 callback 함수 안에서 변조하여 forward 시킨다.

4. 실험 방법에 대한 설명 및 로그파일 결과 분석

패킷 forwarding 이 잘 진행되는지 확인하기 위해서 다음과 같은 과정으로 실험을 진행했다.

1) Routing table 에 새로운 ip 주소에 대한 rule 을 설정해준다. 이 때의 주소는 임의로 정하였다(10.37.129.5). 클라이언트의 ip는 10.37.129.3 이었고 서버의 ip는 10.37.129.4 였다.

```
yoonhan@yoonhan-Parallels-Virtual-Platform:~/lkm/pktfwd$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.37.129.0     *               255.255.255.0   U     100    0        0 enp0s5
10.37.129.5     *               255.255.255.255 UH    0      0        0 enp0s5
link-local      *               255.255.0.0     U     1000   0        0 enp0s5
```

2) kernel 에서 ip forward 가 가능하도록 해주기 위해서 **/proc/sys/net/ipv4/ip_forward 의 값을 1로** 바꾸어 준다.

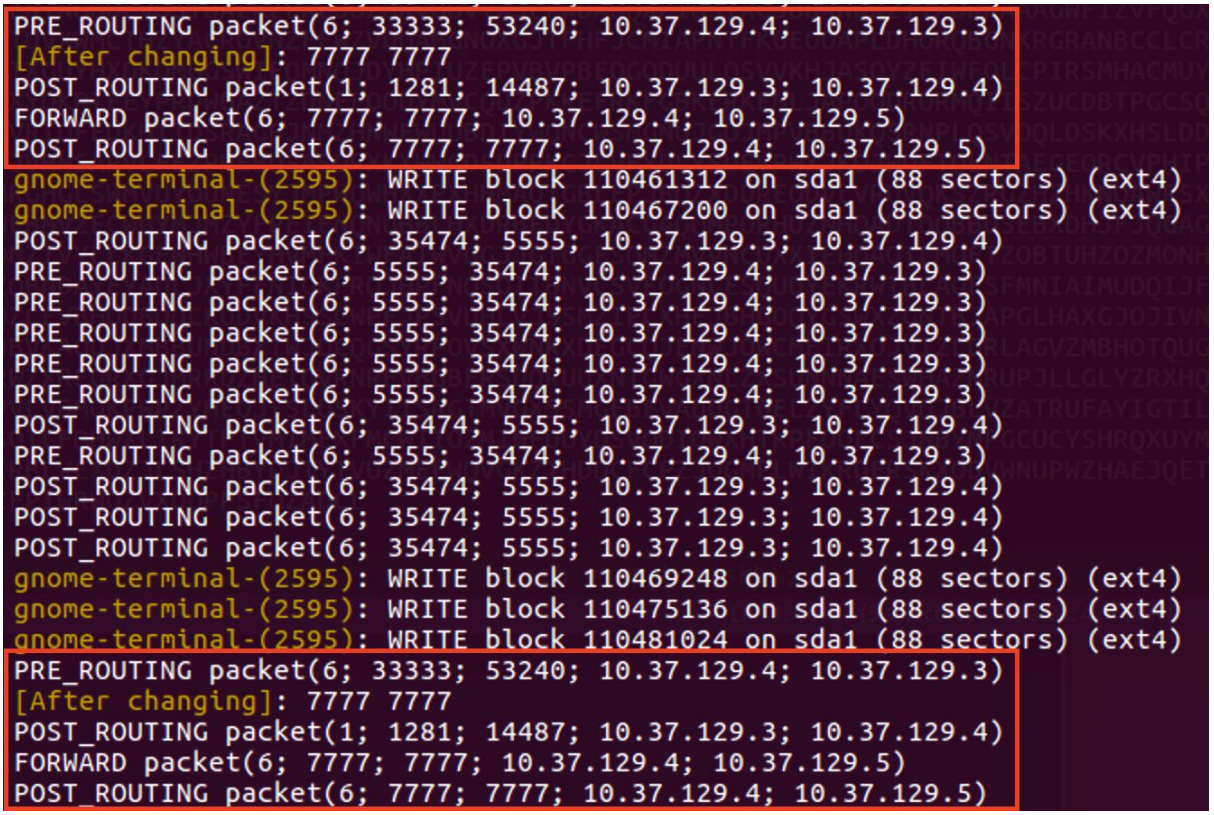
```
yoonhan@yoonhan-Parallels-Virtual-Platform:~/lkm/pktfwd$ cat /proc/sys/net/ipv4/ip_forward
1
```

3) 작성한 LKM 을 컴파일하고 load 한다.

```
yoonhan@yoonhan-Parallels-Virtual-Platform:~/lkm/pktfwd$ lm
mymodule               2771  0
```

4) 서버와 클라이언트를 포트 번호 33333, 4444, 5555, 6666, 7777을 이용하여 연결하고, 이후에 클라이언트는 서버로부터 메시지를 수신받도록 한다.

5) dmesg 명령어로 커널 로그 메시지를 확인한다. 확인한 이후에는 *dmesg > log.txt* 로 로그를 텍스트 파일 형식으로 저장한다.



커널 메시지 로그를 보면, PRE_ROUTING packet 메시지에 서버의 33333번 포트로부터 온 패킷이 forward 되어 FORWARD hooking 포인트로 넘어가서 FORWARD packet 메시지가 출력된 것을 볼 수 있다. 이 과정에서 destination address 가 10.37.129.3(클라이언트) 에서 10.37.129.5(임의의 주소값)으로 변경된 것을 POST_ROUTING hooking 포인트에서 확인할 수 있다.

5. 과제 수행 시의 Trouble과 Troubleshooting 과정

- 패킷 헤더를 코드상에서 어떻게 변조하는지 다음의 블로그를 참조하였다.
https://richong.tistory.com/257?category=707815

- iphdr struct 와 tcphdr struct 의 멤버(source, dest, saddr, dad)는 elixir 의 linux 커널 코드를 참조했다.

- String 형식으로된 ip 주소(ex. “10.37.129.4”)를 네트워크가 요구하는 형식에 맞도록 변환하기 위해 custom inet_addr 함수 코드를 stack overflow 에서 참조하였다. 기존의 inet_addr 함수는 커널 영역에서는 사용이 불가능하기 때문이다.

https://stackoverflow.com/questions/22530089/passing-ip-in-string-to-htonl