

시스템 프로그래밍

3차과제 보고서

- Netfilter를 이용한 Firewall 모듈 -

제출일	2020.12.19.	학과	컴퓨터학과
과목	시스템 프로그래밍	팀	4
Freeday	2일 사용	팀원	홍성윤(2016320187) 윤상준(2016320186)

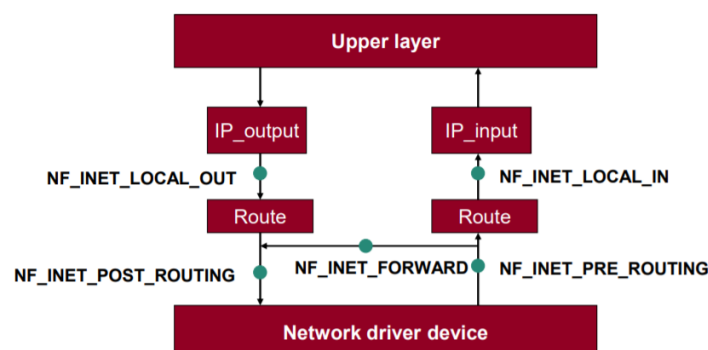
1. Netfilter 및 Hooking에 대한 설명

● 개념

- Netfilter란 ip layer에서 패킷에 대한 추가적인 처리를 도와주는 리눅스 커널 내부의 프레임워크이다.
- Netfilter를 사용함으로써 패킷에 대한 처리를 위해 커널 내부의 코드를 직접 수정할 필요 없다.
- Netfilter는 hooking 형태로 작동한다. Hooking이란 callback 함수를 등록해 특정 조건이 참이면 callback 함수를 호출하는 방식이다.
- Netfilter에서의 hooking point(hook 지점)는 패킷이 네트워크 경로를 통과할 때 패킷 경로에 정의된 지점을 의미한다. 패킷이 hook 지점에 도달하면 해당 지점에 등록된 hook함수가 호출되게 된다. Hook 함수 처리가 끝난 뒤에, 나머지 네트워크 스택을 이어서 진행하게 된다.
- 이러한 특징을 이용해 커널 코드의 수정 없이 패킷의 대한 추가적인 처리나 조작이 용이하다.

● 구현

- nf_hook_ops : 원하는 hooking 포인트에 등록하는 구조체이다. *hook를 통해 등록하고자 하는 함수를 가리킨다. hook 포인트, 우선순위, 네트워크 family 등의 정보를 포함한다. Hook 포인트는 아래 그림과 같이 총 다섯 종류가 있다. 다섯가지 hook 포인트를 가지고 어떠한 위치를 통과하는 패킷을 처리할 것인지 정할 수 있다.



<그림 1>

- `nf_register_hook()`, `nf_unregister_hook()` : `nf_hook_ops`를 해당 hooking 포인트에 등록/해제하는 함수이다.
- Hook 함수를 통해서 패킷에 대한 원하는 처리를 할 수 있다. `return` 값은 `NF_DROP`, `NF_ACCEPT`, `NF_STOLEN`, `NF_QUEUE`, `NF_REPEAT` 중에 가능하다. 이러한 `return` 값들을 통해 특정 패킷은 drop 하는 등의 처리가 가능하다.
- Hook 함수는 다음과 같은 구조체 타입을 가진다: `typedef unsigned int nf_hookfn(void *priv, struct sk_buff *skb, const struct nf_hook_state *state);` 따라서 Hook 함수를 선언할 때에도 `unsigned int` (함수 이름) `(void *priv, struct sk_buff *skb, const struct nf_hook_state *state);` 의 형식으로 해야한다.

2. 작성한 소스코드에 대한 설명

● group4.c

- 커널 모듈을 작성한 c 파일. `proc` 파일시스템을 통해 사용자는 모듈과 상호 작용한다.
- Makefile : 커널 모듈을 만들기 위해서는 소스코드 뿐만 아니라 Makefile 이 필요하다. Makefile이 있어야 `make`로 파일을 컴파일 할 수 있다. Makefile과 `group4.c`는 같은 디렉토리에 위치한다.
- 커널 모듈을 사용한 이유 : netfilter는 등록된 hook 함수가 호출되는 방식으로 작동한다. 모듈이 올라가면 netfilter hook 함수를 등록하고, 모듈이 내려가면 hook 함수를 해제하도록 하였다. 커널 모듈은 동적으로 커널의 기능을 넣었다 뺐다 할 수 있기 때문에 필요한 커널 기능을 확장하기 용이하다. 즉, 커널을 새로 컴파일할 필요 없이 netfilter hook 함수를 동적으로 등록했다 해제할 수 있다.
- Proc 파일시스템을 사용한 이유 : `proc` 파일시스템을 사용하면 커널 모듈 등 운영체제의 각종 정보를 유저 모드에서 쉽게 접근할 수 있다. 커널 모듈의 작동상황이나 정보 등을 쉽게 알아올 수 있다. 본 과제에서는 `proc` 파일시스템을 통해 사용자가 쉽게 방화벽 rule을 수정하고 확인할 수 있다. 또한 `proc`

파일 시스템은 커널이 직접 파일시스템을 관리하는 방식이므로 파일시스템 오버헤드를 줄일 수 있다.

- group4.c 소스코드는 내용에 따라 크게 두 부분으로 나눌 수 있다. proc 파일시스템을 이용해 rule 수정 및 출력하는 부분과 netfilter hook함수를 등록하고 rule에 따라 패킷 처리하는 부분으로 나뉘서 설명하겠다.

● Proc 파일시스템을 통한 firewall rule 수정 및 출력

- 모듈이 처음 올라가면 실행되는 함수(module_init)으로 simple_init 함수를 등록했다. simple_init은 proc 디렉토리 및 파일을 생성하고 hooking 포인트에 nf_hook_ops를 등록한다. 즉, 커널 모듈이 적재되면 proc 파일시스템에 group4 디렉토리 및 add, del, show 3개의 파일이 생성되고 작성한 hook 함수가 등록된다. 또한, rule 구조체 배열에 사용하기 위한 index 인 idx 값을 0으로 초기화한다.
- add, del, show 세개의 파일은 각각 add_fops, del_fops, show_fops가 file operation으로 등록되었다. 각 파일에서 사용하는 operation을 정의한다. 세 파일 모두 권한은 '0777'로, 모든 접근이 허용된다.
- my_open은 세 파일 중 하나가 open 되면 실행된다. 파일이 open 됨을 확인하기 위한 커널 메시지를 출력한다.
- struct my_rules : firewall rule을 저장하기 위한 구조체이다. 구조체 배열로써 사용했다. Type은 rule의 네가지 타입 중 하나를 나타내고 port_num은 포트 번호다. Firewall rule은 블랙리스트이므로 my_rules 구조체 배열에 존재하는 경우, 해당 패킷을 drop하게 된다.
- add : 새로운 firewall rule을 array에 저장하기 위한 함수이다. add 함수는 add proc 파일에 대해 write를 실행했을 때 호출된다. 사용자는 echo 명령어로 add에 rule type과 port 번호를 입력한다. 이러한 입력정보와 size는 add 함수의 인자인 ubuf, size 로 들어온다. copy_from_user 함수를 통해 ubuf의 데이터를 size만큼 커널 버퍼로 복사한다. sscanf 함수를 통해 복사된 데이터를 형식에 맞게 r, p 변수에 저장하고 rule 구조체 배열의 현재 index 위치에 rule type과 port 번호를 추가한다. 이때, 입력된 rule type이 유효하지 않은

값이라면 커널 메시지를 출력한다. idx 값을 증가시키고 복사된 data의 길이를 반환한다.

- del : 특정 firewall rule을 제거하기 위한 함수이다. del 함수는 del proc 파일에 대해 write를 실행했을 때 호출된다. 사용자는 echo 명령어로 del에 지우고자 하는 rule의 index를 입력한다. 유저 데이터를 커널 영역으로 복사해 오는 과정은 add 함수와 동일하다. 입력된 데이터를 int i 변수에 저장한다. i 값이 유효하지 않다면 커널 메시지를 출력한다. 그렇지 않다면 memmove 함수를 사용해 구조체 배열을 한 칸씩 앞으로 이동함으로써 rule 정보를 삭제한다. idx 값을 1 감소시키고 복사된 data의 길이를 반환한다.
- show : 현재 firewall rule을 출력해 사용자에게 보여주기 위한 함수이다. show 함수는 show proc 파일에 대해 read를 실행했을 때 호출된다. 사용자는 cat 명령어로 show 파일을 읽는다. 이를 통해 사용자는 현재 firewall rule 정보를 확인하게 된다. 데이터를 출력하기 위해서 ubuf로 데이터를 복사해야 한다. 복사할 데이터는 sprintf 함수를 사용해 buf에 형식에 맞게 저장했다. 모든 rule에 대한 데이터를 buf에 저장한다. Show 함수가 처음 실행될 때, *ppos 값은 0이다. 출력이 한 번만 되게 하기 위해서 마지막에 *ppos 값을 복사하는 데이터의 길이 값으로 저장하고 *ppos 값이 0일 때만 user buffer로 데이터를 복사하도록 했다. 그렇지 않으면 del 파일에 write를 한 순간 이후로 계속해서 데이터를 출력하게 된다. 복사된 데이터의 길이를 반환한다.

● Netfilter hook 함수 작성, 등록 및 해제

- nf_hook_ops : hooking 포인트에 hook 함수를 등록하는 구조체이다. Rule type 'P'는 'I'와 같이 Inbound 패킷에 대한 rule이다. 해당하는 hook 포인트는 'NF_INET_PRE_ROUTING'이다. Rule 'O', 'F'에 해당하는 hook 포인트는 각각 'NF_INET_POST_ROUTING', 'NF_INET_FORWARD'이다. Hook 함수 등록이 필요한 지점이 세 곳이므로 nf_hook_ops 구조체를 세 개 사용했다. inbound_ops, outbound_ops, forward_ops는 각각 inbound_book, outbound_hook, forward_hook 함수가 hook 함수로 등록되어 있다. 세 구조체 모두 pf 필드는 'PF_INET'이고 priority 필드는 'NF_IP_PRI_FIRST'이다.
- Priority를 'NF_IP_PRI_FIRST'로 해준 이유는 Rule type 'P'를 제외한 모든

type이 해당 경우에 부합한다면 drop하는 것이기 때문이다. 애초에 drop할 패킷이라면 다른 hook 함수를 거치기 전에 확인하여 drop시켜 주는 것이 훨씬 효율 적일 것이다. Rule type 'P'은 부합하는 port number에 대한 패킷의 ip address와 port number를 바꿔 준다. 이 또한, hook 함수로 다른 처리를 해 주기 전에 바꿔주면 발생 가능한 에러를 줄일 수 있으므로 이점이 있다고 생각하였다. 예를 들면 어떤 hook 함수가 특정 ip address로 들어가는 패킷에 대해서만 특별한 처리를 해준다고 하자. 그리고 한 패킷이 들어가는데, 이는 Rule type 'P'에 해당한다고 가정하자. 앞에서 언급한 특별한 처리를 Rule type 'P'에 대한 처리 이전에 해준다면, 이것은 처리를 잘못된 경우가 될 것이다. 왜냐하면 처음에 언급한 대로, 특정 ip address로 들어가는 패킷에만 처리를 해 주고 싶었는데, 뒤에서 ip address가 바뀌어 그것이 아니게 되어 버렸기 때문이다. 따라서 priority를 최우선으로 설정해 두면 이러한 경우를 미연에 방지할 수 있다.

- inbound_hook : inbound 패킷과 proxy 패킷에 대한 처리를 해주기 위한 hook 함수이다. inbound_hook 의 hook 지점은 'NF_INET_PRE_ROUTING'이다. 해당 hook 지점에서 추가해 줄 수 있는 Rule type 은 'I'와 'P'이다. 그러므로 이 두 가지 경우를 한 함수에서 체크하도록 하였다. if 문을 통해 우선은 'I' type 에 대해 확인하고 'P' type 에 대해 확인한다. 그 이유는, 만약 둘 다에 해당하는 경우에는 무엇을 우선 시 하던지 간에 결국 패킷은 drop 이 되기 때문에 패킷을 drop 시켜 줄 수 있는 'I' type 부터 먼저 검사한다.
- outbound_hook : outbound 패킷에 대한 처리를 하기위한 hook 함수이다. outbound_hook 의 hook 지점은 'NF_INET_POST_ROUTING'이다. 패킷이 나갈 때 적용이 된다. 해당 hook 지점에서 추가해 줄 수 있는 Rule type 은 'O'이다. 이에 해당하는 경우 패킷을 drop 시켜준다.
- forward_hook : forward 패킷에 대한 처리를 하기위한 hook 함수이다. forward_hook 의 hook 지점은 'NF_INET_forwarding'이다. 패킷이 forward 될 때 적용이 된다. 해당 hook 지점에서 추가해 줄 수 있는 Rule type 은 'F'이다. 이에 해당하는 경우 패킷을 drop 시켜준다.
- 위 세 hook 함수 모두 return 을 하기전에 해당 패킷에 대한 정보를 커널

로그에 출력한다. 제일 먼저 drop/accept 여부와 패킷의 종류(ex.INBOUND)를 출력한다. 패킷이 drop 된다면 DROP 을 패킷 종류 앞에 출력하고 accept 되었다면 패킷 종류만 출력한다. 이외에도 protocol, source 포트 번호, destination 포트 번호, source ip 주소, destination ip 주소, syn fin ack rst 에 대한 flag 정보를 커널 로그에 출력한다.

- Port number 나 ip address 변환에 사용했던 함수들: 네트워크의 경우 우리가 쓰는 cpu 와 다르게 big endian 을 사용한다. 그러므로 이들을 우리가 활용하기 위해서는 little endian 으로 변환해주어야 한다. 또한, ip address 값을 편리하게 다루기 위해 as_addr_to_net 이나 as_net_to_addr 함수를 활용하였다.

endian을 변환 시켜주는 함수는 다음과 같다 : be16_to_cpu, cpu_to_be16.

be16_to_cpu은 big endian 16비트 값을 little endian으로 변환 시켜주고, cpu_to_be16은 그 반대이다. 이를 통해 헤더 안에 있는 port number과 본인이 직접 코드에 입력한 port number 을 비교할 수 있었다. 출력 또한 이 변환 과정을 거친 후에 실시해 주었다. as_net_to_addr 함수는 ip address를 string 형식으로 변환시켜 주므로 패킷의 ip address를 출력할 때 유용하였고, as_addr_net 함수는 그 반대로 작용하여 'P' type에서 패킷의 ip address를 바꿔줄 때 유용하였다.

3. 실행 방법에 대한 설명 및 로그파일 결과 분석

● 실행 방법 설명

- 커널 모듈 적재 : 위에서 커널 모듈 소스파일과 Makefile 에 대해서 설명하였다. 두 파일이 있는 디렉토리에서 make 명령어를 실행하면 *.ko 확장자를 가지는 LKM 파일이 생성된다. insmod 명령어를 통해 원하는 커널 모듈을 적재할 수 있다. 이때 반드시 'sudo'를 앞에 붙여야 한다. (실제 예: sudo insmod group4.ko) group4.ko 모듈이 적재되면 /proc 에 group4 디렉토리가 생성되며 /proc/group4 에 add, del, show proc 파일이 생성된다.
- Firewall Rule 추가 : add proc 파일을 통해 firewall rule을 추가할 수 있다. echo 명령어로 추가하고자 하는 rule을 add 파일에 쓰면 해당 rule이 커널 모듈의 my_rules 구조체 배열에 저장된다. 입력하는 rule은 'P 1111' 과 같이 'rule type 포트 번호' 형식이어야 한다. (실제 예: echo P 1111 > add)
- Firewall Rule 삭제 : del proc 파일을 통해 기존의 firewall rule을 삭제할 수 있다. echo 명령어로 삭제하고자 하는 rule의 인덱스를 del 파일에 쓰면 해당 index의 rule 구조체가 구조체 배열에서 제거된다. 입력하는 인덱스는 int 값이어야 하고, 현재 rule 배열에 없는 인덱스를 입력하면 안된다. (실제 예: echo 0 > del)
- Firewall Rule 출력 : show proc 파일을 통해 현재 firewall rule을 출력하고 확인할 수 있다. cat 명령어로 show 파일을 읽으면 현재 my_rules 구조체 배열에 있는 rule들을 출력하게 된다. 이때 출력되는 형식은 '인덱스(rule type): 포트번호' 이다. (실제 예: cat show)
- 실험에 사용되는 rule : custom firewall 실험을 위해 사용한 firewall rule은 다음과 같다.

0(O): 8888 1(I): 4444 2(P): 5555 3(F): 5555

<그림 2>

```
sungyoon@sungyoon-VirtualBox:/proc/group4$ sudo echo O 8888 > add
sungyoon@sungyoon-VirtualBox:/proc/group4$ sudo echo I 4444 > add
sungyoon@sungyoon-VirtualBox:/proc/group4$ sudo echo P 5555 > add
sungyoon@sungyoon-VirtualBox:/proc/group4$ sudo echo F 5555 > add
sungyoon@sungyoon-VirtualBox:/proc/group4$ cat show
0(O): 8888
1(I): 4444
2(P): 5555
3(F): 5555
```


- Forward가 되는 패킷을 만들기 위해, 라우팅 테이블에 새로 ip를 추가하였다. 그 ip는 proxy packet이 향하게 될 “131.1.1.1”이다. 만약 어떤 port에 대한 패킷에 대해 “P”와 “F” rule을 동시에 적용시켜 준다면, 그 패킷은 필시 “F” rule에 의하여 ‘NF_INET_forwarding’ 지점에서 drop될 것이다.

<그림 3>

```
user@user-VirtualBox:~/net$ sudo route add -host 131.1.1.1 dev enp0s3
user@user-VirtualBox:~/net$ route
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
131.1.1.1        *               255.255.255.255 UH      0      0      0 enp0s3
link-local        *               255.255.0.0     U       1000   0      0 enp0s3
192.168.56.0      *               255.255.255.0   U       100    0      0 enp0s3
```

다음 system call은 forwarding 기술을 사용하기 위한 옵션 설정이다.

이 또한 실험하기 전에 적용시켜 주어야한다.

```
user@user-VirtualBox:~/net$ sudo sysctl -w net.ipv4.ip_forward=1
```

- 실험을 위한 서버와의 통신 : 등록된 firewall rule이 잘 작동하는지 확인하기 위해서 서버와의 통신을 한다. 클라이언트 어플리케이션은 2차과제의 프로그램을 사용한다. 먼저, 다음 그림과 같이 server 프로그램에서 ‘4444, 5555, 6666, 7777, 8888’ 포트 번호를 입력해서 포트 번호에 대한 소켓을 각각 만든다. 이것이 클라이언트와 커넥션을 수립하기 위한 첫 번째 준비과정이다.

```
Ubuntu 16.04.1 LTS oslab tty1
oslab login: syspro
Password:
Last login: Sat Dec 19 12:31:32 KST 2020 on tty1
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-190-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

134 packages can be updated.
0 updates are security updates.

syspro@oslab:~$ ./server 4444 5555 6666 7777 8888
Open: 4444 5555 6666 7777 8888
Bind: 4444 5555 6666 7777 8888
Listen: 4444 5555 6666 7777 8888
Start worker: 4444 5555 6666 7777 8888
accept port: 7777-53666
accept port: 6666-48298
close port: 6666-48298
close port: 7777-53666
```

<그림 4>

- 다음과 같이 Client 프로그램에 입력할 포트번호 개수와 ‘4444, 5555, 6666, 7777, 8888’ 포트를 입력하면 server와 통신을 시작하게 된다.

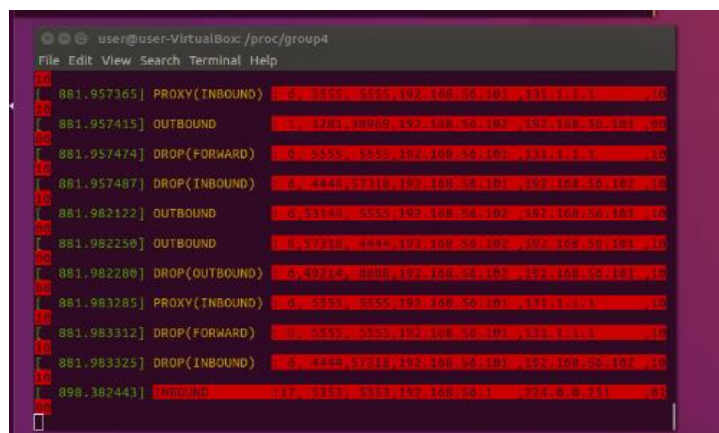
```

user@user-VirtualBox:~/net$ ./a.out
5 4444 5555 6666 7777 8888
open: 4444 5555 6666 8888 7777
Connection Successful
Connection Successful
6666-5.txt write completed!
7777-7.txt write completed!
Connection failed for port 4444

```

<그림 5>

- 통신이 진행되고 난 뒤, dmesg 명령어를 통해 커널 로그를 확인한다. 로그 파일을 통해 구현한 firewall이 잘 작동하는지 확인할 수 있다.



```

user@user-VirtualBox: /proc/group4
File Edit View Search Terminal Help
[ 881.957365] PROXY(INBOUND) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.957415] OUTBOUND : 1, 1281,38969,192.168.56.102,192.168.56.101,0000
[ 881.957474] DROP(FORWARD) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.957487] DROP(INBOUND) : 6, 4444,57318,192.168.56.101,192.168.56.102,1010
[ 881.982122] OUTBOUND : 6,53144, 5555,192.168.56.102,192.168.56.101,1000
[ 881.982250] OUTBOUND : 6,57318, 4444,192.168.56.102,192.168.56.101,1000
[ 881.982280] DROP(OUTBOUND) : 6,49214, 8888,192.168.56.102,192.168.56.101,1000
[ 881.983285] PROXY(INBOUND) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.983312] DROP(FORWARD) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.983325] DROP(INBOUND) : 6, 4444,57318,192.168.56.101,192.168.56.102,1010
[ 898.382443] dmesg : 17, 3157, 3157,192.168.56.1,224.0.0.251,00

```

<그림 6>

● 로그파일 결과 분석

- 로그파일 중 마지막 14개의 line에 대해서 결과를 분석해보자.

```

[ 865.957459] DROP(OUTBOUND) : 6,49214, 8888,192.168.56.102,192.168.56.101,1000
[ 865.958320] PROXY(INBOUND) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 865.958392] DROP(FORWARD) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 865.958423] DROP(INBOUND) : 6, 4444,57318,192.168.56.101,192.168.56.102,1010
[ 881.957365] PROXY(INBOUND) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.957415] OUTBOUND : 1, 1281,38969,192.168.56.102,192.168.56.101,0000
[ 881.957474] DROP(FORWARD) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.957487] DROP(INBOUND) : 6, 4444,57318,192.168.56.101,192.168.56.102,1010
[ 881.982122] OUTBOUND : 6,53144, 5555,192.168.56.102,192.168.56.101,1000
[ 881.982250] OUTBOUND : 6,57318, 4444,192.168.56.102,192.168.56.101,1000
[ 881.982280] DROP(OUTBOUND) : 6,49214, 8888,192.168.56.102,192.168.56.101,1000
[ 881.983285] PROXY(INBOUND) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.983312] DROP(FORWARD) : 6, 5555, 5555,192.168.56.101,131.1.1.1,1010
[ 881.983325] DROP(INBOUND) : 6, 4444,57318,192.168.56.101,192.168.56.102,1010

```

<그림 7>

- 우선은 모든 패킷은 크게 3가지 방향으로 흐른다. 서버(192.168.56.101)에서 client(192.168.56.102), client에서 서버, 그리고 client에서 131.1.1.1. 마지막 경우는 Netfilter를 통해 패킷을 proxy 패킷으로 만든 경우이다.

- 각 메시지 별 분석 (1)~(15) : 중복되는 내용은 생략한다.

(1) Outbound 패킷이 drop 되었음을 의미한다. Protocol type은 6으로 tcp를 의미한다. 발신지포트는 49214(client), 목적지포트는 8888(server) 다. 0번 rule에 의해서 8888포트로의 outbound 패킷이 잘 drop 되었다. flag가 1000이므로 syn 패킷이다.

(2) Inbound 패킷이 rule 타입 'P' 에 의해 destination이 변경된 것을 알 수 있다. Protocol은 tcp이고, 발신지포트는 5555(server) 이다. 2번 rule에 의해서 5555 포트의 패킷이 특수케이스로 처리되었다. 목적지아이피는 131.1.1.1로, 목적지포트는 발신지 포트로 바뀐 것을 확인할 수 있다. Flag가 1010이므로 syn과 ack 정보를 포함한 패킷이다.

(3) Forward 패킷이 drop 되었음을 의미한다. Protocol은 tcp이고, 발신지포트와 목적지포트는 5555 이다. 3번 rule에 의해서 5555 포트에서 온 forward 패킷이 drop 되었다. 목적지 아이피 131.1.1.1인 점을 보아 forward 패킷인 것을 알 수 있다. Flag가 1010이므로 syn과 ack 정보를 포함한 패킷이다.

(4) Inbound 패킷이 drop 되었음을 의미한다. Protocol은 tcp이고, 발신지포트는 4444이다. 1번 rule에 의해서 4444 포트에서 들어오는 패킷이 drop 된다. Flag가 1010이므로 syn과 ack 정보를 포함한 패킷이다.

(5) (2) 와 동일

(6) Outbound 패킷이 drop 되지 않고 잘 처리되었음을 의미한다. Protocol number가 1이므로 ICMP 패킷임을 알 수 있다. 즉, 에러 발생 원인을 알려주거나, 네트워크 상태 진단 등의 기능을 가지는 패킷이다. 발신지포트는 1281, 목적지포트는 38969 다. Rule에 있는 포트번호가 아니므로 drop 되거나 특수케이스 처리되지 않는다.

(7) (3)과 동일

(8) (4)와 동일

(9) Outbound 패킷이 drop 되지 않고 잘 처리되었음을 의미한다. Protocol은 tcp다. 발신지포트는 53144(client), 목적지포트는 5555(server) 다. 5555번 포트로 outbound 되는 패킷에 대해서는 rule 이 존재하지 않으므로 drop 되지 않는다. flag가 1000이므로 syn 패킷이다.

(10) Outbound 패킷이 drop 되지 않고 잘 처리되었음을 의미한다. Protocol 은 tcp다. 발신지포트는 57318(client), 목적지포트는 4444(server) 다. 4444번 포트로 outbound 되는 패킷에 대해서는 rule 이 존재하지 않으므로 drop 되지 않는다. flag가 1000이므로 syn 패킷이다.

(11)(1)과 동일

(12)(2)과 동일

(13)(3)과 동일

(14)(4)과 동일

- 정리 : 위의 <그림 5>를 보면 6666, 7777 포트에 대한 수신은 끝마쳤는데, 다른 포트에 대한 수신은 아직 완료되지 않은 상태인 것을 알 수 있다. 하나 하나씩 살펴보자.
- 4444 포트에 대한 패킷은 'NF_INET_PRE_ROUTING' 에서 바로 drop 되기 때문이다. 그래서 <그림 4>을 보면 아직 connection setup조차 되지 않은 것을 볼 수 있다. 위의 분석을 보면 4444 포트에 대한 패킷은 항상 SYN+ACK 이며, 이것은 SYN에 대한 ACK 패킷을 보내는 것이다. 이 패킷이 계속해서 drop 되기 때문에 connection setup은 이루어지지 않고, 수신 또한 되지 않는다.
- 5555 포트에 대한 패킷의 경우는 위의 분석을 보면 ip address가 Rule 'P'에 의해 "131.1.1.1"로 바뀌고, forward되어 결과적으로 Rule "F"에 의해 drop된다. 4444포트에 대한 패킷은 모두 drop된다고 보면 된다. 따라서 이 경우도 4444 포트와 마찬가지로 SYN와 대한 ACK을 받지 못해 connection setup이 되지 않는 것이고, 수신 또한 되지 않는다.
- 8888 포트에 대한 패킷은 'NF_INET_POST_ROUTING'에서 drop 되기 때문에 client가 SYN 패킷조차 송신을 완료하지 못한다. 그러므로 connection setup이 되지 않고, 수신도 되지 않는다.
- 서버는 일정 시간이 되면 connection이 이루어지지 않은 port에 대한 setup을 중단한다
- 결과적으로 6666, 7777포트에 대한 패킷들은 아무런 Rule에 적용 받지 않고 성공적으로 패킷을 client가 서버로부터 받을 수 있었다.

4. 과제 수행 시 어려웠던 부분과 해결 방법

● Proc 파일 출력

- 커널 모듈에서 정의한 rule 구조체 배열을 통해 rule을 관리한다. 사용자가 proc 파일시스템을 통해 커널 모듈과 상호작용을 하면서 rule을 직접 관리할 수 있도록 구현하는 부분에서 크고 작은 어려움이 있었다.
- 특히 show 함수를 통해 rule을 출력하는 부분이 어려웠다. 계속해서 같은 내용이 출력 되었기 때문이다. 이는 cat 명령어로 show파일을 읽은 후부터 show 함수가 계속 실행되는 것이 원인이라고 판단했다. 그리고 ppt 슬라이드의 예시를 참고해서 *ppos = len; 문장의 의미를 파악하려 했다. 함수가 계속해서 호출된다고 가정했을 때, len과 같은 변수는 계속 0으로 초기화 되지만 ppos 는 포인터 변수이므로 함수가 호출될 때 마다 변하지 않는다. *ppos의 초기값은 0이었다. 이 점을 이용해 user 영역에 data를 복사하는 copy_to_user함수를 *ppos 값이 0일 때만 호출하게 해서 user buffer로의 복사가 한 번만 일어나도록 하였다. 함수 종료 직전에 *ppos 값에 len을 저장함으로써 가능했다. Len은 총 복사한 데이터의 길이이다.
- 한 번의 copy_to_user로 rule 배열의 모든 정보를 복사하기 위해 sprintf 함수로 buffer에 모든 내용을 저장했다. sprintf 함수는 서식에 맞게 내용을 buf에 저장하고 데이터의 길이를 반환한다. 이 점을 이용해 rule 배열의 각 인덱스마다 buf에 저장할 때, 저장할 때마다 데이터의 길이에 맞게 저장할 위치를 (buf+len) 옮겨 주었다. buf 에는 복사하고자 하는 모든 데이터가 들어가게 되고 copy_to_user 함수를 통해 user buffer에 복사되면서 사용자는 현재 rule의 내용을 확인할 수 있게 되었다.

● Hooking 함수 작성

- 전반적으로 크게 어려움은 없었으나, 실습 설명 ppt에서 제공해주신 unsigned int as_addr_to_net 함수의 코드가 잘못되어 초반 에러를 찾는데 조금 고생을 하였다. 제공해 주신 코드라 틀릴 리가 없다고 생각해 코드를 헤집었지만 문제가 해결되지 않아 as_addr_to_net을 적절히 수정하였더니 해결되었다.