

## Netfilter 를 이용한 Packet Capturing 과 Firewall 모듈 제작

양희철 / 이화경  
부산대학교 정보컴퓨터공학부 정보보호동아리 Keeper

Heecheol, Yang / Hhakyung, Lee  
Keeper. Dept. of CSE, Pusan National University

shrtngo@gmail.com / pongpong21@naver.com

2010 년 5 월 27 일

### 요 약

이 문서는 Linux의 Netfilter Framework를 이용하여 Network 상의 Packet를 분석하고 Accept & Drop 하는 방법을 담고 있다. Netfilter Framework는 Linux Network Stack 사이에서 움직이는 Packet를 Hooking하는 기능을 제공하며 이를 이용하면 다양한 네트워크 조작성이 가능하다. Netfilter는 Kernel-Level에서 동작하므로 Kernel에 포함되거나 Module로서 동작 가능하며, 이 문서에서는 Module로서의 사용법을 다룬다. 나아가 이것들을 이용하여 간단한 Firewall Module을 제작하여 Packet을 분석 후 조건에 맞게 Filtering 해 본다.

주제어: Netfilter, Packet, Filter, Module, Linux, Firewall

## 1. Introduction

Firewall(방화벽)은 네트워크의 특정 계층 사이에 존재하여 정책에 따라 계층과 계층 사이의 데이터 통신을 감시하고, 필요시에는 이것을 차단하거나 허용하는 역할을 하는 모듈(Module)이다. 이 모듈은 하드웨어가 될 수도 있고, 소프트웨어도 될 수 있다.

일반적으로 방화벽이 네트워크를 감시하는 방식은 계층과 계층 사이의 패킷을 분석하는 방식이다. 자신을 통과하는 패킷들에 포함되어 있는 각종 계층의 헤더 파일을 분석하여 IP주소, Port등을 알아내는 것뿐만 아니라 헤더 외에 실제 데이터의 내용도 분석하여 위험한 데이터는 차단하는 등의 방식도 가능하다.

이렇게 방화벽은 계층 사이의 모든 계층을 감시해야 하기 때문에, 물리적, 논리적으로 모든 패킷이 공통적으로 반드시 지나가게 되는 위치에 설치를 해야 한다.

Linux의 대표적인 방화벽 모듈로는 iptables가 있다.

## 2. Netfilter

Netfilter 는 표준 Berkeley socket interface 의 외부에 존재하는 packet mangling에 대한 framework 로, 크게 네 부분으로 구성된다.

먼저 각각의 프로토콜은 “hooks”라는 것을 정의하며, 이는 패킷 프로토콜 스택의 packet’s traversal에 있는 잘 정의된 포인터를 의미한다. 이러한 포인터에서, 각각의 프로토콜은 packet과 hook number를 이용하여 netfilter framework 을 호출하게 된다.

두 번째로, 커널의 일부분은 각 프로토콜에 대하여 다른 hook을 감시하도록 등록할 수 있다. 따라서 패킷이 넷필터 프레임워크를 통과할 때, 누가 그 프로토콜과 hook을 등록했는지 확인하게 된다. 이러한 것이 등록되어 있다면, 등록된 순서대로 패킷을 검사하고, 패킷을 무시하거나(NF\_DROP), 통과시키고(NF\_ACCEPT), 또는 패킷에 대한 것을 잊어버리도록 넷필터에게 지시하거나(NF\_STOLEN), 사용자 공간에 패킷을 대기시키도록(queuing) 넷필터에게 요청한다(NF\_QUEUE).

세 번째 부분은 대기된 패킷을 사용자 공간으로 보내기 위해 제어하는 것으로 이러한 패킷은 비동기방식으로 처리된다.

Netfilter 는 이러한 저 수준 framework 와 더불어, 다양한 모듈이 작성되었으며, 이는 이전 버전의 커널에 대하여 유사한 기능, 확장 가능한 NAT시스템 그리고 확장 가능한 패킷 필터링 시스템을 제공한다.

## 3. Packet Capture / Drop

대부분의 모듈이 그렇듯이, Netfilter 모듈은 자신을 처리하기 위한 기본 정보를 구조체에 채워 넣은 다음 커널에 등록하는 방식으로 동작하게 된다. Netfilter의 경우, nf\_hook\_ops 구조체에 이 정보가 저장되는데, linux/netfilter.h에 정의되어 있다.

```
struct nf_hook_ops {
```

```

struct list_head list;

/* User fills in from here down. */
nf_hookfn *hook;
struct module *owner;
u_int8_t pf;
unsigned int hooknum;
/* Hooks are ordered in ascending priority. */
int priority;
};

```

그림 1 linux/netfilter.h : struct\_netfilter\_ops

여기서 주의깊게 봐야 할 것은 **nf\_hook \*hook** 멤버이다. 이 멤버는 패킷을 받았 을 시 실행할 hooking function의 function pointer를 지정한다. nf\_hook 타입은 그림 2와 같다.

```

typedef unsigned int nf_hookfn(unsigned int hooknum,
                               struct sk_buff *skb,
                               const struct net_device *in,
                               const struct net_device *out,
                               int (*okfn)(struct sk_buff *));

```

그림 2 nf\_hookfn

이 함수가 호출되면 인자로 여러 가지 정보들이 넘겨지는데, 이 중 특히 중요한 sk\_buff는 실제 NIC에 전달될 패킷을 가리키는 포인터가 저장되어 있다.

이 함수가 어떤 값을 리턴하느냐에 따라 패킷의 이동 여부가 결정된다. 예를 들어, NF\_DROP를 리턴하면 이 패킷은 다음 Layer로 가지 못하고 Drop되며, NF\_ACCEPT를 리턴하면 이 패킷은 다음 Layer로 이동한다. 즉 어떤식으로 리턴하느냐를 조정하여 Packet Filtering을 할 수 있다.

## 4. Important Fields in IP & TCP Header

단순히 Packet 만을 filtering 하는 것은 크게 의미 없는 행위이다. 그러나 Packet 내부의 어떤

정보를 이용하면 원하는 Packet을 filtering 할 수 있다. TCP/IP의 경우, 일반적으로 Packet filtering에 사용될 수 있는 정보는 protocol, source & destination IP, source & destination Port가 있을 수 있는데, 이는 IP Header, TCP Header에서 찾아볼 수 있다.

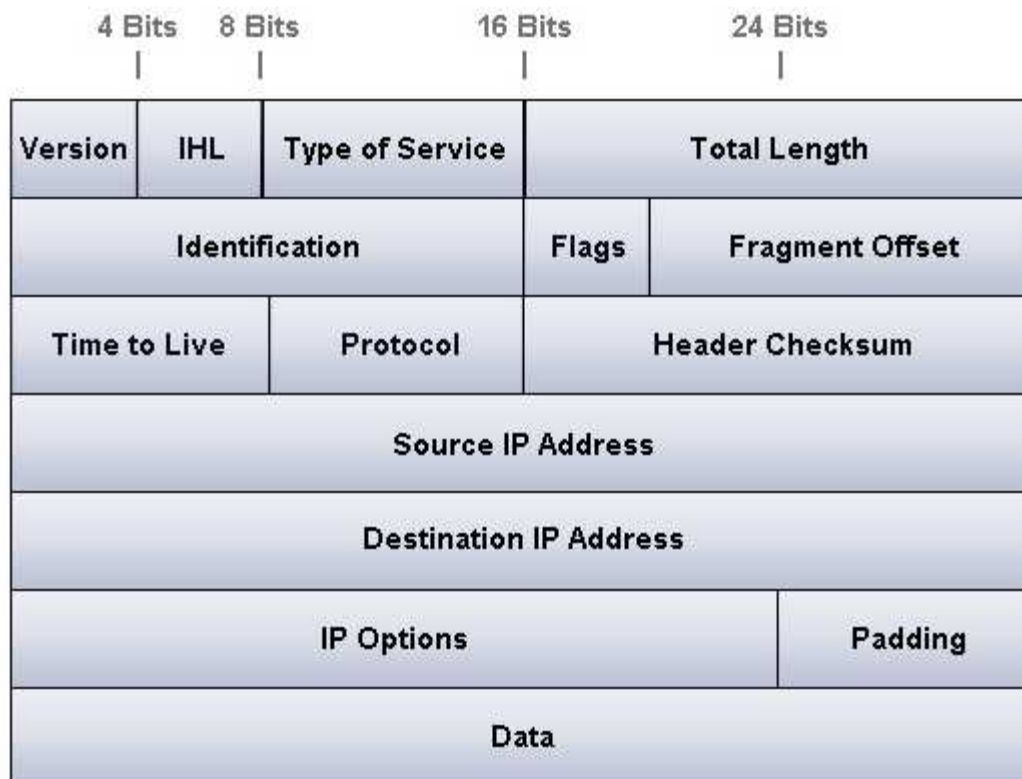


그림 3 IP Header

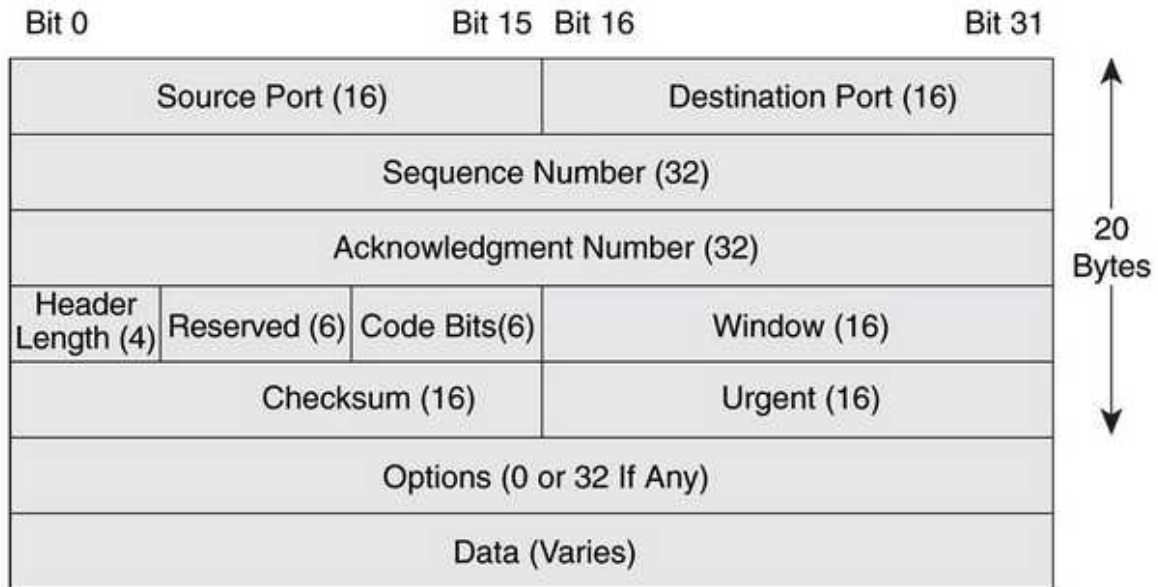


그림 4 TCP Header

IP, TCP Header는 그림4, 그림 5와 같이 구성되어 있다. Linux 에서는 위와 같은 Header 를 <linux/ip.h>, <linux/tcp.h>에 정의 해 두고 있다.

```

struct iphdr {
#ifdef __LITTLE_ENDIAN_BITFIELD
    __u8    ihl:4,
           version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
    __u8    version:4,
           ihl:4;
#else
#error "Please fix <asm/byteorder.h>"
#endif
    __u8    tos;
    __be16  tot_len;
    __be16  id;
    __be16  frag_off;
    __u8    ttl;
    __u8    protocol;

```

```
    __sum16 check;  
    __be32  saddr;  
    __be32  daddr;  
    /*The options start here. */  
};
```

그림 5 IP Header - linux/ip.h : struct iphdr

```
struct tcphdr {  
    __be16  source;  
    __be16  dest;  
    __be32  seq;  
    __be32  ack_seq;  
#if defined(__LITTLE_ENDIAN_BITFIELD)  
    __u16   res1:4,  
          doff:4,  
          fin:1,  
          syn:1,  
          rst:1,  
          psh:1,  
          ack:1,  
          urg:1,  
          ece:1,  
          cwr:1;  
#elif defined(__BIG_ENDIAN_BITFIELD)  
    __u16   doff:4,  
          res1:4,  
          cwr:1,  
          ece:1,  
          urg:1,  
          ack:1,  
          psh:1,  
          rst:1,  
          syn:1,  
          fin:1;  
#else
```

```

#error "Adjust your <asm/byteorder.h>
defines"
#endif

    __be16  window;
    __sum16 check;
    __be16  urg_ptr;
};

```

그림 6 TCP Header - linux/tcp.h : struct tcphdr

그림 3,4의 각 Field는 각각 그림 5,6의 각 멤버에 mapping 되어 있다. 예를 들어, 패킷의 목적지 포트를 알고 싶으면 **iphdr.daddr**를 조사하면 된다.

전송된 패킷에서 IP Header를 추출하려면 ip\_hdr()함수의 인자로 skb를 넘겨 주면 IP Header의 포인터를 리턴한다. 그리고 IP Header의 다음에 TCP Header가 위치하여 있으므로 그림 7과 같은 방식으로 IP Header와 TCP Header의 포인터를 얻을 수 있다.

```

struct iphdr *iph = ip_hdr(skb);
struct tcphdr *tcph = (struct tcphdr*)((char*)iph + sizeof(struct iphdr));
char *data = (char*)tcph + sizeof(*tcph);

```

그림 7 skb로부터 IP Header와 TCP Header, Original Data를 얻는 코드

그림 7에서 data는 Application Layer에서 생성한 Original Data를 가리키는 포인터이다.

iph에서는 Source / Destination의 IP Address를 얻을 수 있는데, **iph->saddr** / **iph ->daddr** 멤버에서 알 수 있다. 마찬가지로 tcph에는 Port Number를 알 수 있으며, **tcph->saddr** / **tcph -> daddr** 멤버에서 얻을 수 있다. 단 이 때에는 **Byte Ordering**에 주의하자.

## 5. Device Driver Module - Register to Kernel

Netfilter는 Kernel Level에서 동작하는 만큼 Module로써 움직인다. 즉 Netfilter Module을 작성 후 등록해야 동작한다.

Module의 **init\_module()** 함수는 Module이 Insert 될 때 가장 먼저 실행되는 함수로, 여기서 등록 작업을 할 수 있다.

```

int init_module()
{
    netfilter_ops.hook = main_hook;
    netfilter_ops.pf = PF_INET;
    netfilter_ops.hooknum = NF_INET_PRE_ROUTING;
    netfilter_ops.priority = 1;
    nf_register_hook(&netfilter_ops);    //모듈등록
    return 0;
}

```

그림 8 Netfilter 모듈 등록

**struct nf\_hook\_ops netfilter\_ops**에 각 초기값을 지정한 후 **nf\_register\_hook()** 함수로 등록하면 모듈로서 동작이 가능하다.

Compile 한 Module은 **insmod** 명령으로 삽입 가능하며, **lsmod**로 확인 가능하고, **rmmod**로 제거 할 수 있다.

```

[root@localhost dd]# make
make -C /lib/modules/2.6.31.5-127.fc12.i686.PAE/build M=/home/shrtngo/dd modules
make[1]: Entering directory `/usr/src/kernels/2.6.31.5-127.fc12.i686.PAE'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/kernels/2.6.31.5-127.fc12.i686.PAE'
[root@localhost dd]# ls
Makefile Module.markers Module.symvers drop.c drop.ko drop.mod.c drop.mod.o drop.o modules.order
[root@localhost dd]# insmod drop.ko
[root@localhost dd]# lsmod | more
Module                Size Used by
drop                   1652  0
fuse                   52712  2
sunrpc                 158388  1
ip6t_REJECT           4620  2

```

그림 9 모듈 컴파일과 삽입



## 6. Example Source Code - Simple Firewall Module

2~5 절에서 설명한 내용을 바탕으로 간단한 방화벽을 제작하여 보자. 방화벽 프로그램은 은 그림 10과 같이 구성되어 있다.

파일	설명
<b>drop.c</b>	Netfilter Firewall Module
<b>app.c</b>	Application that controls module
<b>myprotocol.h</b>	Command Protocol
<b>Makefile</b>	Makefile

그림 10 방화벽 소스 파일의 구성

drop.c 는 실제 방화벽 Module Source 이다. app.c의 Application은 Rule를 등록하는 명령을 Module에게 내릴 수 있으며, 이 예제에서는 차단할 IP 주소를 추가하는 명령만 넣었다. 필요한 명령은 myprotocol.h에 추가해서 넣을 수 있다. Makefile은 Module을 Compile 하기 위한 Makefile이다.

그림 11,12,13,14,15는 Simple Firewall의 전체 소스 코드이다.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/fcntl.h>

#include <linux/module.h>
#include <linux/skbuff.h>
#include <linux/netdevice.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/in.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/ip.h>
#include "myprotocol.h"
```

```
#define DROP_NAME      "drop"
#define DROP_MAJOR    240

unsigned long filter_addr[10]; /* array that contains ip addresses to drop */
unsigned int filter_addr_cnt = 0;

static struct nf_hook_ops netfilter_ops;
struct sk_buff *sock_buff;

struct file_operations drop_fops ;

void add_addr(const char*);

/* open() system call */
int drop_open(struct inode *inode, struct file *filp)
{
    return 0;
}

/* close() system call */
int drop_release(struct inode *inode, struct file *filp)
{
    return 0;
}

/* write() system call */
ssize_t drop_write(struct file *filp, const char *buf, size_t count, loff_t *fpos)
{
    struct myprotocol* pmsg = (struct myprotocol*)buf;
    if(pmsg->cmd == ADD_IP)
    {
        char *addr = pmsg->addr;
        add_addr(addr);
    }
    return 0;
}
```

```
}

unsigned long inet_aton(const char*);
unsigned int main_hook(unsigned int hooknum,
                        struct sk_buff *skb,
                        const struct net_device *in,
                        const struct net_device *out,
                        int (*okfn)(struct sk_buff*))
{
    int i;
    char *data;
    struct iphdr *iph = ip_hdr(skb);    /* ip header*/
    unsigned long saddr = 0, daddr = 0; /* ip address */
    unsigned short source = 0, dest = 0; /* port number */

    /*tcp, udp is apart from iph 20byte(sizeof(struct iphdr));
    struct tcphdr *tcph = (struct tcphdr*)((char*)iph + sizeof(struct iphdr));
    struct udphdr *udph = (struct udphdr*)((char*)iph + sizeof(struct iphdr));
    unsigned short http = 80;           /*http port number : 80

    /*get source and dest ip from iph
    saddr = iph->saddr;
    daddr = iph->daddr;

    /*get source and dest port from tcph
    source = htons(tcph->source);
    dest = htons(tcph->dest);

    if(iph->protocol == IPPROTO_UDP)
    {
        data = (char*)udph + sizeof(*udph);
        /* print udp data */
        printk("<1>source : %u Wt dest : %uWn %sWn",source,dest,data);
    }
    for(i = 0 ; i < filter_addr_cnt ; i++){
        if(saddr == filter_addr[i]){
```

```
        /* ip drop */
        return NF_DROP;
    }
}
if(source == http){return NF_DROP;} //http port Drop

return NF_ACCEPT;
}

int init_module()
{
    drop_fops.owner = THIS_MODULE;
    drop_fops.open = drop_open;
    drop_fops.release = drop_release;
    drop_fops.write = drop_write;

    netfilter_ops.hook = main_hook;
    netfilter_ops.pf = PF_INET;
    netfilter_ops.hooknum = NF_INET_PRE_ROUTING;
    netfilter_ops.priority = 1;
    nf_register_hook(&netfilter_ops); //모듈등록
    register_chrdev(DROP_MAJOR,DROP_NAME,&drop_fops);
    return 0;
}

void cleanup_module()
{
    nf_unregister_hook(&netfilter_ops); //모듈해제
    unregister_chrdev(DROP_MAJOR,DROP_NAME);
}

unsigned long inet_aton(const char * str)
{
    unsigned long result = 0;
    unsigned int iaddr[4] = {0,};
    unsigned char addr[4] = {0,};
```

```
    int i;
    sscanf(str, "%d.%d.%d.%d ", iaddr, iaddr+1, iaddr+2, iaddr+3);

    for(i = 0 ; i < 4 ; i++)
    {
        addr[i] = (char) iaddr[i];
    }
    for(i = 3 ; i > 0 ; i--)
    {
        result |= addr[i];
        result <<= 8;
    }
    result |= addr[0];

    return result;
}

/* add address to drop */
void add_addr(const char *addr)
{
    filter_addr[filter_addr_cnt++] = inet_aton(addr);
    printf("<1> %s(%lu) drop added!\n", addr, filter_addr[filter_addr_cnt]);
}
```

그림 11 drop.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#include "myprotocol.h"
```

```
int main()
{
    int dev;
    printf("open start\n");
    dev = open("/dev/drop", O_RDWR|O_NDELAY);
    printf("open end\n");
    if( dev < 0 )
    {
        printf("open error!\n");
        return -1;
    }
    struct myprotocol msg;
    msg.cmd = ADD_IP;
    strcpy(msg.addr, "192.168.0.4");
    write(dev, (char*)&msg, sizeof(msg));
    close(dev);
}
```

그림 12 app.c

```
#define ADD_IP 1

struct myprotocol
{
    unsigned int cmd;
    char addr[256];
};
```

그림 13 myprotocol.h

```
obj-m := drop.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

그림 14 Makefile

Module은 HTTP Port인 80번 막도록 하였고, app.c에서 192.168.0.4 IP를 차단할 IP 목록에 추가하여 Firewall Module이 차단하도록 하였다. 앞서 관련 내용들을 설명하였고, 코드상 특별히 어려

운 부분은 없기에 drop.c의 설명은 주석 정도로 마무리짓겠다. Kernel-Level에서는 `inet_aton()` 함수가 제공되지 않아서 직접 구현하였다는 것과 `add_addr()` 함수가 차단할 IP를 추가한다는 것 정도만 염두해 두자.

Module은 Application과 연동하기 위해 **Character Device Driver**를 이용하여 I/O를 수행하도록 하였다. Character Device Driver의 설명은 이 문서의 범주를 벗어나므로 생략하겠다.

그림 15는 Firewall Module 실행 후 **HTTP Port(80)**을 차단하여 웹 브라우저가 접속을 하지 못하는 것을 보여준다.

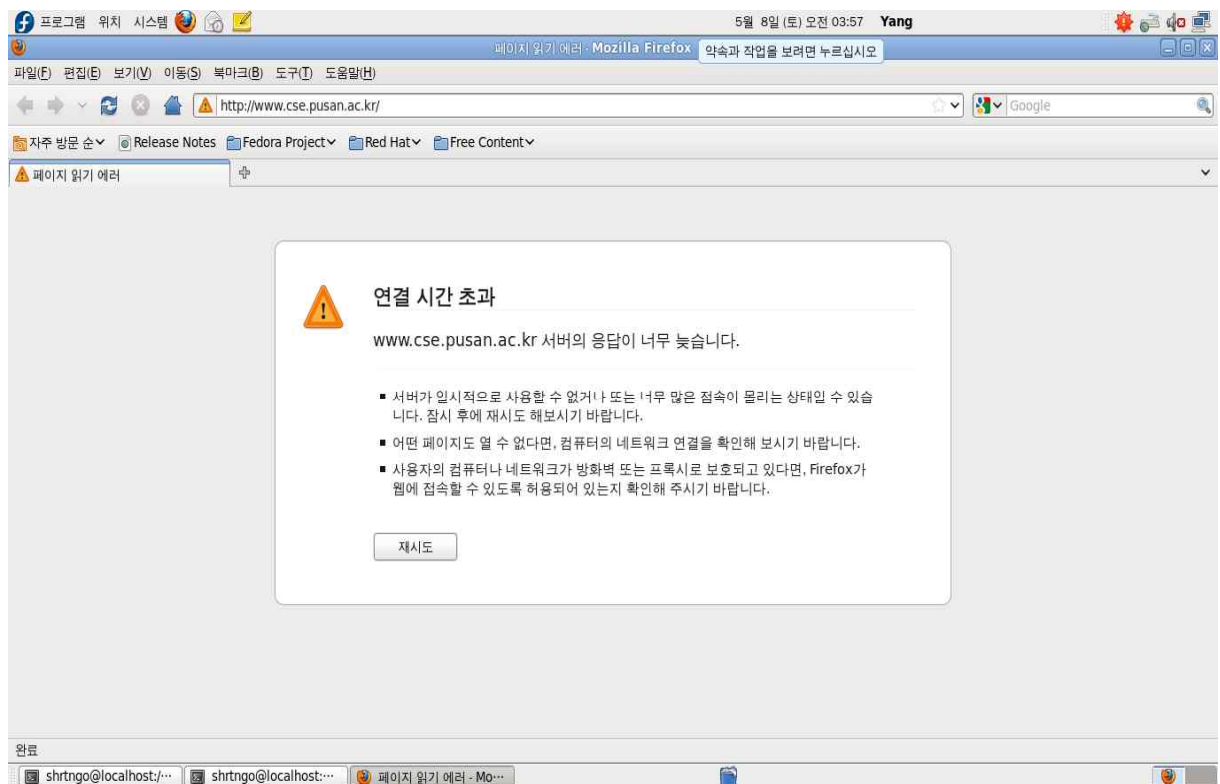
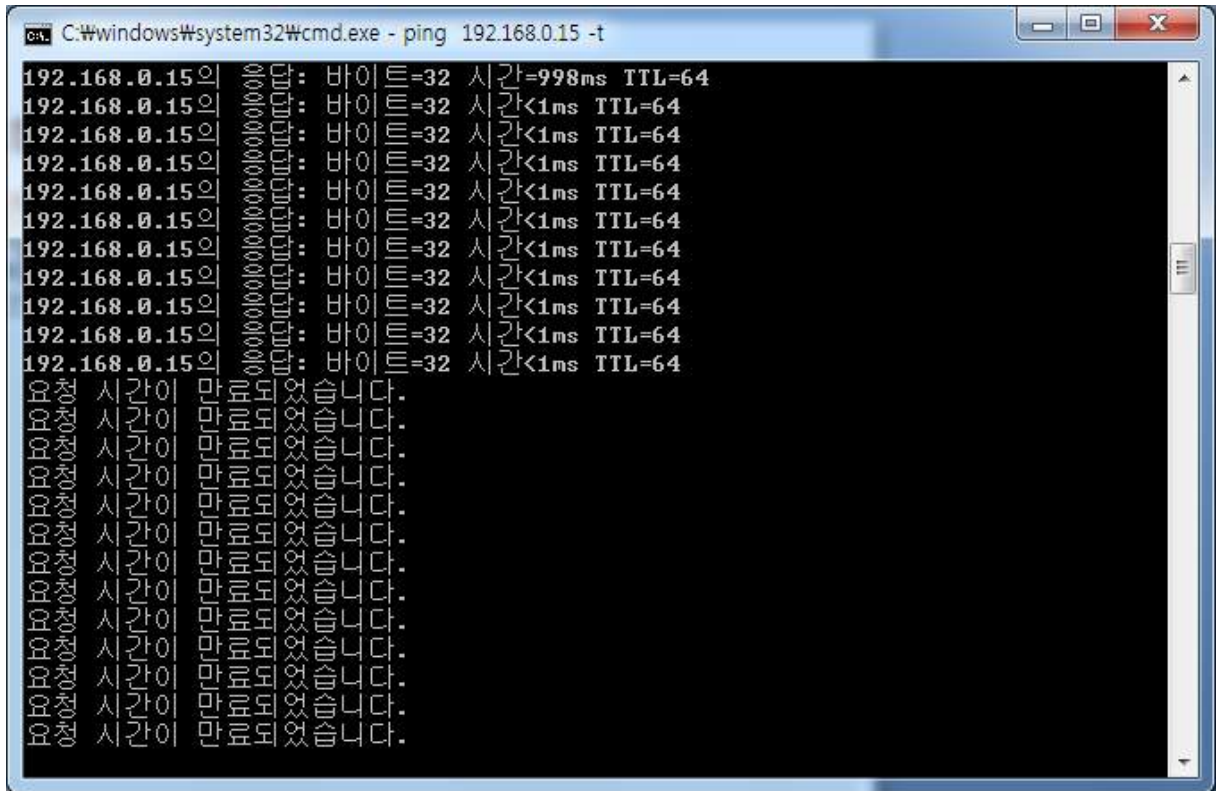


그림 15 HTTP Port를 차단한 모습

app.c는 192.168.0.4라는 IP를 차단하게 하는데, 이 IP 주소는 필자의 테스트 당시 IP 주소이다. 이 Host에서 Firewall이 Load되어 있는 Host로 **Ping Test**를 하였는데, Application을 실행시키기 전에는 IP 주소가 등록되어 있지 않으므로 Ping Test가 성공하지만 실행 후에는 실패하게 된다. 그림 16은 이것을 보여준다.



```
C:\windows\system32\cmd.exe - ping 192.168.0.15 -t
192.168.0.15의 응답: 바이트=32 시간=998ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
192.168.0.15의 응답: 바이트=32 시간<1ms TTL=64
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
요청 시간이 만료되었습니다.
```

그림 16 Application 실행 결과

마지막으로, 이 방화벽은 UDP 기반의 Echo Server와 Client가 서로 데이터를 주고 받으면 UDP Packet을 Capture하여 그 내용을 보여주는 기능도 수행한다. TCP의 경우에는 **Byte Stream** 단위로 전송되기 때문에 그 내용을 알기 어렵지만 UDP는 **Message** 단위로 전송되기 때문에 그 내용을 쉽게 알 수 있다. 그림 17은 그 결과를 보여준다.



```
[root@localhost dd]# dmesg | grep keeper
keeper!
keeper!
keeper!!!
keeper!!!
[root@localhost dd]#
```



```
[shrtngo@localhost udpecho]$ ./uecho_client 127.0.0.1 1234
Insert message(q to quit): keeper!
Message from server: keeper!
Insert message(q to quit): keeper!!!
Message from server: keeper!!!
Insert message(q to quit): █
```

그림 17 UDP Packet Capture

## 7. Conclusion

Netfilter Framework를 이용하여 Packet를 Capture하는 방법을 알아보고 TCP/IP의 주요 Field들을 확인하여 보았다.

Netfilter는 Packet 전송시 중간에서 Hook하여 이를 처리 할 수 있게 해 주는 Framework로, 어떻게 사용하느냐에 따라 다양한 응용이 가능하다. 예를 들어 Hooking Function의 Return을 어떻게 해 주느냐에 따라 패킷을 Accept/Drop할 수 있고, 패킷의 내용을 알 수 있으므로 패킷 내용의 조작도 가능하다.

Netfilter도 결국 Kernel Level Module이기 때문에 Module로 제작하여 동작한다.

이 Netfilter를 이용하여서 예제로 간단한 방화벽을 제작해 보았다. IP와 TCP의 Header를 확인하여 특정 IP와 Port를 차단할 수 있게 하였고, UDP Packet인 경우 Original Data도 확인하여 출력하여 보았다.

User Level이 아닌 Kernel Level에서 하는 작업은 무궁무진하다. Netfilter도 그 예이다. 하지만 이렇게 Low Level의 조작은 항상 시스템을 불안하게 하고 심하게는 망가뜨리게 할 수도 있으므로 항상 사용에 신중을 기해야 할 것이다.

## Refernces.

- [1] Rusty Russel, kenji, KLDLP Wiki : Netfilter-Hacking HOWTO, <http://wiki.kldp.org/wiki.php/DocbookSgml/Netfilter-hacking-TRANS#NETFILTER>
- [2] Netfilter Project, <http://netfilter.org>
- [3] 유영창, 리눅스 디바이스 드라이버 , 한빛미디어, 2008
- [4] Alessandro Rubini & Jonathan Corbet, Linux Device Driver, O'Reilly Media, 2001  
<http://www.xml.com/ldd/chapter/book/>
- [5] 부산대학교 이동통신연구실, <http://mobile.cse.pusan.ac.kr>

### 양희철 / Hee-cheol, Yang



2010년 부산대학교 정보컴퓨터공학부 2학년에 재학중임. 현재 정보컴퓨터공학부 산하 보안동아리 Keeper에서 회장을 담당하고 있음. 시스템 소프트웨어와 컴퓨터 네트워크, 임베디드 시스템, 전산학 이론에 관심이 있음. 작성한 주요 프로그램으로 Windows Shell, Network Manager, File Manager, Internet Messenger 등이 있음

Blog : <http://studio.tistory.com>

### 이화경 / Hwa-kyung, Lee



2010년 부산대학교 정보컴퓨터공학부 4학년에 재학중임. 현재 정보컴퓨터공학부 산하 보안동아리 Keeper에서 기술고문을 담당하고 있음. 컴퓨터네트워크와 컴퓨터보안, 임베디드 공부에 관심이 있음. 작성한 주요 프로그램으로 Firewall on Windows XP가 있음