Firewalls are widely deployed technologies for protecting networks from unauthorized access while permitting legitimate communications to pass. In this lab, you will learn about how to build a firewall of your own using the power of Linux kernel modules and Netfilter, which is a packet filtering subsystem in the Linux kernel stack. The techniques you learn on kernel module and filter/manipulating network packets can be an extremely useful tool for developing other cool network applications. Anyway, writing code in the kernel means you are writing code in a place where you are limited only by your imagination.

1 Initial Setup

This section will help you get your environment setup for writing your own kernel module in vSphere. The procedures for accessing the virtual machine is the same as what we did in the Lab 3.

1.1 Accessing the Virtual Machine

We have access to a service called vSphere, which will provide us with a number of local Virtual Machines for you to use (easy to recover from kernel panic). It is recommended to use the vSphere for your convenience, since the lab has been tested and will be graded by running your program in the Ubuntu VM in the vSphere. If you have your own VM setup, or even a personal computer with the same kernel version, you may use that as well. To access your VM that we have setup for you, please follow these instructions.

- Speak with the TA to get permissions to a VM setup.
- If you are on a Windows machine, and you are on campus, you can download the vSphere client from https://csil-vcenter.ad.uiuc.edu/.
- If you are not on a Windows machine, you can still use vSphere. The campus Windows Terminal servers already have vSphere installed. The remote desktop host is ews-windows-ts.ews.illinois.edu. Your username will be UofI\NetID. Be careful of the UofI domain name...it is the only one that will work. (I have no idea why...) If you are using Linux, you can use the rdesktop command from the command line to remote into the windows terminal server.
  
  rdesktop ews-windows-ts.ews.illinois.edu.
- Launch the vSphere client and connect to csil-vcenter.ad.uiuc.edu. You username and password should be your Netid and AD password.
- If you see a 'home' screen, click the 'VMs and Templates' icon. This will show you all the VMs you have access to.
- Start your VM with the green arrow at the top.
- You can pull up an X-console using the icon that looks like a computer with an arrow pointing out of it.
• The default user account if cs498class and the password is cs498class. Please change this password **Immediately**! This account has sudo access.

• From this console, you can use this machine as you would any other. Please keep in mind that vSphere and your VM will only be accessible inside the UIUC network, or via VPN.

### 1.2 Setting Up the Ubuntu Build Environment

In order to build kernel modules, we need to setting up the Ubuntu build environment using module assistant.

```
sudo apt-get install module-assistant
sudo m-a prepare
```

### 2 The Netfilter

Netfilter is the packet filtering framework inside the Linux 2.4.x and 2.6.x kernel series. Netfilter essentially defines a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack. Although this lab deals with IPv4 packets, most of the IPv4 content can be applied to the other protocols as well.

#### 2.1 The Netfilter hooks for IPv4

Netfilter defines five hooks for IPv4. The declaration of the symbols for these can be found in `linux/netfilter_ipv4.h`. These hooks are displayed in the Table 6.1

<table>
<thead>
<tr>
<th>Hook</th>
<th>Called</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF_INET_PRE_ROUTING</td>
<td>After sanity checks, before routing decisions</td>
</tr>
<tr>
<td>NF_INET_LOCAL_IN</td>
<td>After routing decisions if packet is for this host</td>
</tr>
<tr>
<td>NF_INET_FORWARD</td>
<td>If the packet is destined for another interface</td>
</tr>
<tr>
<td>NF_INET_LOCAL_OUT</td>
<td>For packets coming from local processes on their way out</td>
</tr>
<tr>
<td>NF_INET_POST_ROUTING</td>
<td>Just before outbound packets &quot;hit the wire&quot;</td>
</tr>
</tbody>
</table>

Table 6.1: IPv4 hooks

After hook functions have done whatever processing they need to do with a packet, they must return one of the predefined Netfilter return codes as shown in Table 6.2.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF_DROP</td>
<td>Discard the packet</td>
</tr>
<tr>
<td>NF_ACCEPT</td>
<td>Keep the packet, and execute the remaining hooks</td>
</tr>
<tr>
<td>NF_STOLEN</td>
<td>Forget about the packet</td>
</tr>
<tr>
<td>NF_QUEUE</td>
<td>Queue packet for userspace</td>
</tr>
<tr>
<td>NF_REPEAT</td>
<td>Call this hook function again</td>
</tr>
<tr>
<td>NF_STOP</td>
<td>Keep the packet, and skip the remaining hooks</td>
</tr>
</tbody>
</table>

Table 6.2: Netfilter return codes
The NF_DROP return code means that this packet should be dropped completely and any resources allocated for it should be released. NF_ACCEPT tells Netfilter that so far the packet is still acceptable and that it should move to the next stage of the network stack. These two return codes are used in this lab.

### 2.2 Registering and unregistering Netfilter hooks

Registration of a hook function is a simple process with the use of the `nf_hook_ops` structure as defined in `linux/netfilter.h`. The definition of this structure is as follows:

```c
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;
};
```

Let us explain some data fields that will be used for this lab. `hook` is a pointer to a `nf_hookfn` function. This is the function that will be called for the hook. `nf_hookfn` is defined in `linux/netfilter.h` as well. The `pf` field specifies a protocol family. Valid protocol families are available from `linux/socket.h`, but for IPv4 we want to use `PF_INET`. The `hooknum` field specifies the particular hook to install this function for and is one of the values listed in Table 6.1. Finally, the `priority` field specifies where in the order of execution this hook function should be placed. For IPv4, acceptable values are defined in `linux/netfilter_ipv4.h` in the `nf_ip_hook_priorities` enumeration. In this lab, we will be using `NF_IP_PRI_FIRST`.

### 2.3 The Very First Firewall Kernel Module

Now we are ready to write our first netfilter kernel module. The module simply drops all the incoming packets like what is configured as the default rule in many modern firewalls.

#### 2.3.1 Writing the Netfilter Hook

To create a simple Netfilter hook requires several steps. We first write a function named `hook_func_incoming()`, which is called whenever a packet comes in that meets the conditions of `nf_hook_ops`. This function performs the appropriate logic, which in our case is to drop all incoming packets. We fill in the `nf_hook_ops` structure which defines the particular Netfilter hook to target, the priority of it, and gives the type of packet to target (in this case IPv4), and bind `hook_func_incoming()` to it.

We also need to define the methods `init_module()` and `cleanup_module()`. For the Netfilter module, `init_module()` should now contain the code to set up the `nf_hook_ops` structure and register the hook with Netfilter. In `cleanup_module()` we put any code required to clean up. In this case, it corresponds to the code to unregister the hook from Netfilter.

Below is the sample code of the drop-all-packet firewall.
```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>

static struct nf_hook_ops nfho; //struct holding set of hook function options

//function to be called by hook
unsigned int hook_func_incoming( unsigned int hooknum,
                                 struct sk_buff *skb,
                                 const struct net_device *in,
                                 const struct net_device *out,
                                 int (*okfn)(struct sk_buff *))
{
    return NF_DROP; /* Drop ALL packets */
}

//Called when module loaded using 'insmod'
int init_module()
{
    //function to call when conditions below met
    nfho.hook = hook_func_incoming;

    //called right after packet recieved, first hook in Netfilter
    nfho.hooknum = NF_INET_PRE_ROUTING;

    //IPV4 packets
    nfho.pf = PF_INET;

    //set to highest priority over all other hook functions
    nfho.priority = NF_IP_PRI_FIRST;

    //register hook
    nf_register_hook(&nfho);

    printk(KERN_INFO "simple firewall loaded\n");
    return 0;
}

//Called when module unloaded using 'rmmod'
void cleanup_module()
{
    printk("simple firewall unloaded\n");
    nf_unregister_hook(&nfho); //cleanup and unregister hook
}

2.3.2 Writing the Makefile

In order to compile the module, we create an associated Makefile, like the following:
Lab Matthew Caesar

obj-m += simple_firewall.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

clean:
  $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
default:
  $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

2.3.3  Loading and Unloading a basic Netfilter Kernel Module

To install the module, type

```
sudo insmod simple_firewall.ko
```

To remove the module, type:

```
sudo rmmod simple_firewall
```

To verify the module is actually loaded or unloaded, type:

```
dmesg | tail
```

Now the system will drop any incoming packets. You can test the result by pinging the machine or pinging outside world, both should not work. If you are using ssh to remotely log into the machine, your ssh session will be broken. :)

**Note**

- You can download the template of the deny-all firewall as well as the makefile from Piassa. The template can serve as a good starting point for the rest of your work in this lab.
- `printk` is a very useful tool for debugging purpose

3  Building your own Light-weighted Firewall

The light-weighted firewall provides three basic options for dropping packets. These are, in the order of processing:

- Source interface
- Source IP address
- Destination TCP port

First let us look at what data gets passed into hook functions (e.g. the `hook_func_incoming()` in the simple_firewall.c) and how that data can be used to make filtering decisions.

The first argument to `nf_hookfn` functions is a value specifying one of the hook types given in Table 6.1. The second argument is a pointer to a pointer to a `sk_buff` structure, the structure used by the network stack to describe packets,
such as network header, transport headers, packet length, data, etc. You will need to know this struct for accessing the necessary information in a packet for this lab. This structure is defined in `linux/skbuff.h`.

The two arguments that come after skb are pointers to `net_device` structures. `net_device` structures are what the Linux kernel uses to describe network interfaces of all sorts. The first of these structures, `in`, is used to describe the interface the packet arrived on. Not surprisingly, the `out` structure describes the interface the packet is leaving on. It is important to realise that usually only one of the two structures will be provided. For instance, `in` will only be provided for the `NF_INET_PRE_ROUTING` and `NF_INET_LOCAL_IN` hooks. `out` will only be provided for the `NF_IP_LOCAL_OUT` and `NF_IP_POST_ROUTING` hooks.

### 3.1 Filtering by Interface

First, we would like to filter any **INCOMING** packets to the interface `lo`. Remember those `net_device` structures our hook function received? Using the name field from the relevant `net_device` structure allows us to drop packets depending on their source interface or destination interface. To drop all packets that arrive on interface `lo` all one has to do is compare the value of `in->name` with `lo`. If the names match then the hook function simply returns `NF_DROP` and the packet is destroyed.

You can test your program by pinging the localhost and pinging some well-known website, such as google.com

### 3.2 Filtering by IP Address

Now we want to filter any **OUTGOING** packet with the destination IP address 130.126.31.10. This time we are interested in the `sk_buff` structure. Now remember that the skb argument is a pointer to a pointer to a `sk_buff` structure. Obtaining the IP header for a packet is done using the network layer header from the the `sk_buff` structure.

Note that the value of the destination IP should be stored in Network Byte Order (Big-endian, opposite of Intel). Also it is a good practice to check if skb and ip header you get is NULL pointer though it is unlikely the case, and return `NF_ACCEPT` if so.

You can test your program with tools like ping, ssh, iperf or anything at your choice.

### 3.3 Filtering by Destination TCP Port

Now we want to filter any **INCOMING** tcp packets with the destination port 8888. This is only one more step more than checking IP addresses because we need to create a pointer to the TCP header ourselves. Getting pointer to the TCP header is about of allocating a pointer to a struct `tcphdr` (define in linux/tcp.h) and pointing after the IP header in our packet data. Don’t forget that for this function to work, the tcp destination port to deny should be in network byte order.

You can test your program by opening a iperf tcp server listening to port 8888 at host running the firewall kernel module and try to connect it from another machine.

*Note: The Linux cross reference is a very handy tool when dealing with code base in such scale.

http://lxr.free-electrons.com/
4 The Report

Please submit the following materials through the compass.

1. Copy your code done in the Section 3 as well as the makefile. Make sure the makefile is runnable, since your generated executable will be used for testing and grading. Please name the executable as firewall in your makefile.

2. Create a NetFilter kernel module based on (beyond) what we learn in this lab. (e.g. more complicate firewall filtering rules; password sniffer; selectively dropping tcp packets and measure change in congestion window size ... anything at your choice).
   Please include the following in the report
   (1) Description of your module
   (2) Well-commented code body of your hook(s)
   (3) Runnable code and the make file
   (4) The instruction of how to run your code or any experimental results you collected from your code if it is hard to demo or for TA to reproduce your testing scenarios

3. Please submit the lab report in pdf or MS Word format