

Tutorial for Lab 6

Building your own firewall with Linux Kernel Module

CS 498: Networking Lab

Kevin Jin

April 6, 2012

Lab 6

Building Your Own Firewall

- Build a firewall of your own with
 - Linux kernel modules
 - Netfilter
- Use vSphere (similar to lab 3)
 - Ubuntu 10, kernel version 2.6.38.8
 - Easy to recover from kernel panic
- 1 or 2 people per group, google doc sign up sheet
- Grading (partly) by running your code
 - Hand in code, makefile (compilable and runnable)
 - Please name the executable as **firewall** in your makefile

Loadable Kernel Module

- An object file that contains code to extend the running kernel of an operating system
 - Support new hardware/file system
 - Add system calls
- Kernel can be modified while it's running
 - No need to recompile and then reboot
- Unload when no longer needed (free resources)
 - Otherwise, all possible functionalities need to be compiled directly into the base kernel (waster memory, rebuilding time...)
- But inherently unsafe: any 'bug' can cause a system malfunction -- or complete crash!

Netfilter

- A packet filtering framework
- Linux 2.4.x and 2.6.x kernel series
- Essentially defines a set of hooks
 - 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.

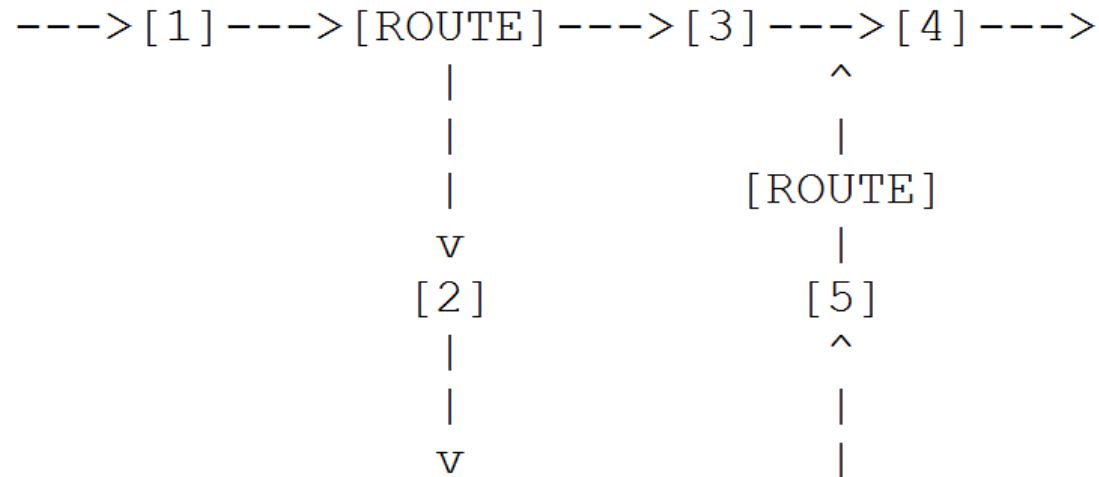
Netfilter hooks for IPv4

- Netfilter defines five hooks for IPv4
 - The declaration of the symbols in linux/netfilter.h

Hook	Called
NF_INET_PRE_ROUTING	After sanity checks, before routing decisions
NF_INET_LOCAL_IN	After routing decisions if packet is for this host
NF_INET_FORWARD	If the packet is destined for another interface
NF_INET_LOCAL_OUT	For packets coming from local processes on their way out
NF_INET_POST_ROUTING	Just before outbound packets "hit the wire"

Netfilter hooks for IPv4

A Packet Traversing the Netfilter System



Hook	Called
[1] NF_INET_PRE_ROUTING	After sanity checks, before routing decisions
[2] NF_INET_LOCAL_IN	After routing decisions if packet is for this host
[3] NF_INET_FORWARD	If the packet is destined for another interface
[5] NF_INET_LOCAL_OUT	For packets coming from local processes on their way out
[4] NF_INET_POST_ROUTING	Just before outbound packets "hit the wire"

Registering/unregistering Netfilter hooks

- Kernel modules can register to listen at any of these hooks
- A module that registers a function must specify the priority of the function within the hook
- Each module registered at that point is called in the order of priorities
- Now your code is free to manipulate the packet

```
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;  
};
```

Protocol family
linux/socket.h

Type of
NetFilter hook

the order of
execution

nf hookfn is defined in linux/netfilter.h

Netfilter hooks for IPv4

– return code

Return Code	Meaning
NF_DROP	Discard the packet
NF_ACCEPT	Keep the packet, and execute the remaining hooks
NF_STOLEN	Forget about the packet
NF_QUEUE	Queue packet for userspace
NF_REPEAT	Call this hook function again
NF_STOP	Keep the packet, and skip the remaining hooks

Useful Tools Before Writing the First Kernel Module

- Linux Cross Reference

<http://lxr.free-electrons.com/>

- Cscope

<http://cscope.sourceforge.net>

–a text-based code browser that facilitates searching complex source trees for definitions, references, global symbols, etc

The Very First Firewall Kernel Module

- Drop all incoming packets
- Three functions

```
//Called when module loaded using 'insmod'  
int init_module()
```

```
//Called when module unloaded using 'rmmod'  
void cleanup_module()
```

```
//function to be called by hook  
unsigned int hook_func_incoming
```

The Very First Firewall Kernel Module – `init_module()`

- `init_module()` contains the code to set up the `nf_hook_ops` structure and register the hook with Netfilter

```
//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;
}
```

Function to process
the packet

printk: extremely
useful tool

The Very First Firewall Kernel Module – `clean_module()`

- Put any code required to clean up, unregister the hook from Netfilter in this example

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

The Very First Firewall Kernel Module – the hook function

- Called whenever a packet comes in that meets the conditions of `nf_hook_ops`
- Code body of how you want to process the packets

```
//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 */
}
```

Building your own Light-weighted Firewall

- Provide three basic options for dropping packets
 - Source interface
 - Source IP address
 - Destination TCP port

```
//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 *)
```

the structure used by
the network stack to
describe packets,
defined in linux/skbuff.h

the interface that
a packet is arrived on

the interface that
a packet is leaving on

Demo

More you can do...

- Traffic sniffer
- Packet analysis, possibly extract username/passwd
- Send out data (secretly) to a remote host when that host sends something (e.g. a PING request)
 - Remote access key-logger
 - Getting lists of who is currently logged onto the machine or obtaining information about open network connections
- Blocks unwanted traffic (trojans, ICMP covert channels or file sharing tools)