Firewall

Lecture 6

Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu

Introduction

- ***firewall***: stop unauthorized traffic flowing from one network to another
  - deployed between trusted and untrusted components
  - separating networks within a trusted network
    - differentiating networks
- firewall implementation: *hardware*, *software*, or *combination*
- firewall's main functionalities:
  - filtering data
  - redirecting traffic
  - protecting against network attacks

Firewall Requirements

- a well-designed firewall meets following requirements
  1. all traffic between two trust zones should pass through
  2. only authorized traffic should be allowed to pass through
  3. immune to penetration

Firewall Policy

- firewall policy: rules that should be enforced
  - rule: provide controls for traffic on network

1. *user control*: controls access to the data based on the role of the user who is attempting to access it
   - applied to user inside firewall perimeter
2. *service control*: access is controlled by the type of service offered by the host that is being protected by firewall
   - enforced on network address, port number, protocol
3. *direction control*: determines the direction in which requests may be initiated and are allowed to flow through the firewall

Firewall Actions

- three actions:
  - ***accepted***: allowed to enter through firewall
  - ***denied***: not permitted to enter through firewall
  - ***rejected***: similar to denied, but notifying the source of packet about decision

ingress filtering: inspects the incoming traffic to safeguard an internal network and prevent attacks from outside.

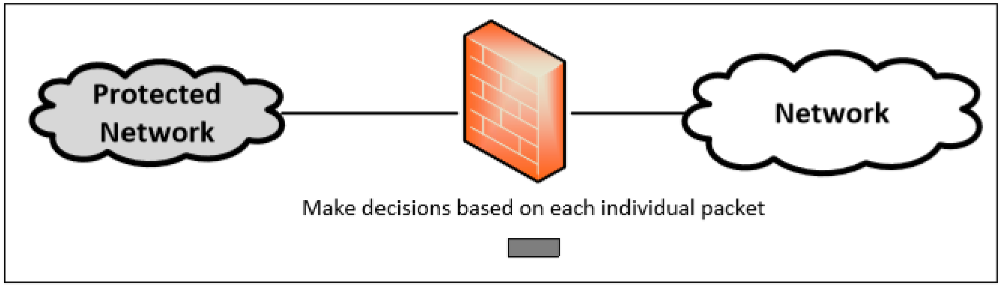
egress filtering: inspects the outgoing network traffic and prevent the users in the internal network to reach out to the outside network.

- *for example:*
  - *blocking social networking sites in school*
  - *Great Firewall of China, blocking access to many sites (YouTube, etc.)*

Types of Firewalls

- depending on the mode of operation, there are three types of firewalls
  - packet filter firewall
  - stateful firewall
  - application/proxy firewall

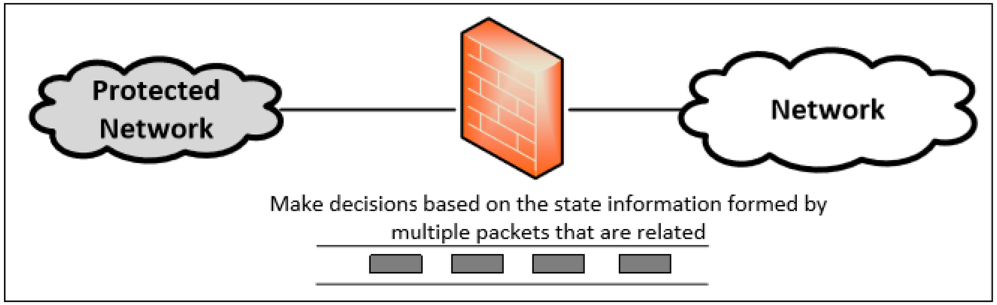
Packet Filter Firewall



controls traffic based on the information in packet headers, without looking into the payload that contains application data

- inspects each packet and make decision based on information in the packet header
- doesn't pay attention to if the packet is a part of existing stream or traffic
- advantages:
  - speed; doesn't maintain the states about packets
    - also called stateless firewall

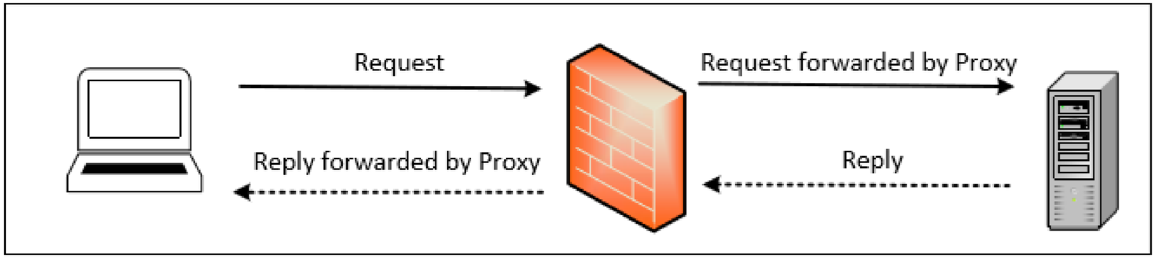
Stateful Firewall


Protected Network
Network
Make decisions based on the state information formed by multiple packets that are related

- tracks the state of traffic by monitoring all the connection interactions until is closed
  - retrains packets until a decision can be made
- connection state table is maintained to understand the context of packets

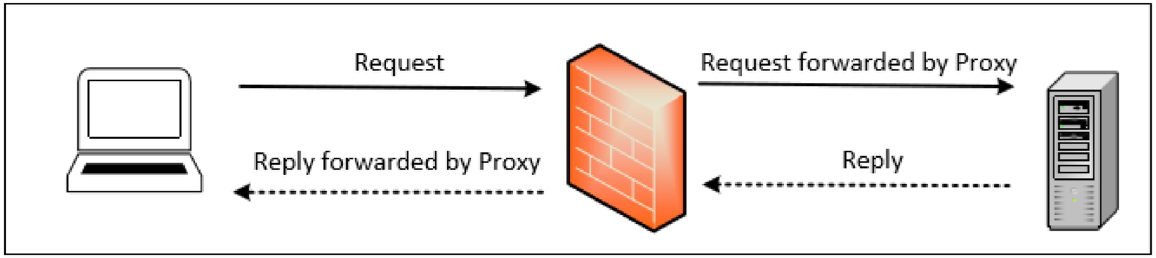
- advantages:
  - allowing through traffic that belong to existing connection

Application/Proxy Firewall



- controls input, output, and access from/to application or service
- unlike packet/stateful firewalls, inspects network traffic up to application layer
- typical application: proxy (application proxy firewall)
  - impersonating the intended recipient
    - client's connection terminates at proxy
    - a new connection initiated from proxy to destination
    - data is analyzed up to application layer to determine if the packet should be allowed or rejected
      - protecting internal from risk of direct interaction

Application/Proxy Firewall



- limitation:
  - implementing new proxies for new protocols
  - slower (reading the entire packet)
- advantages:
  - authenticate user directly rather than depending on network address of system

Building Firewall using Netfilter

- packet filter firewall implementation in Linux
  - packet filtering can be done inside the kernel
  - need to modify the kernel
  - Linux provides two mechanisms (no need to recompile kernel)

- Linux provides two mechanisms

Netfilter: provides hooks at critical points on the packet traversal path inside Linux kernel

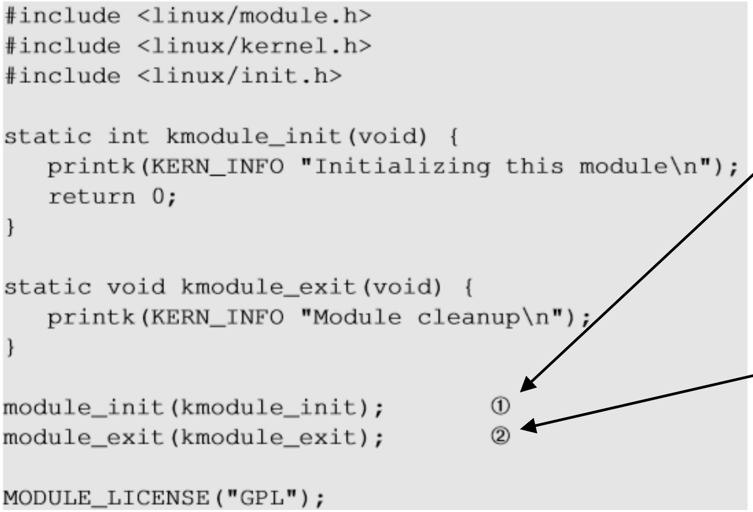
- *allow packets to go through additional program logics (e.g., packet filer)*

Loadable Kernel Modules: allow privileged users to dynamically add/remove modules to the kernel, so there is no need to recompile the entire kernel

Writing Loadable Kernel Modules

- modular Linux kernel: a minimal part of kernel is loaded into memory
- additional features can be implemented as kernel modules, and be loaded into kernel dynamically
  - e.g., a new kernel module supporting a new hardware
- kernel module: pieces of code that can be loaded and unloaded on-demand at runtime
  - they don't run as specific processes but are executed in kernel on behalf of current process
  - need root privilege or CAP_SYS_MODULE capability to be able to insert or remove kernel modules

Loadable Kernel Modules (cont.)



```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int kmodule_init(void) {
   printk(KERN_INFO "Initializing this module\n");
   return 0;
}

static void kmodule_exit(void) {
   printk(KERN_INFO "Module cleanup\n");
}

module_init(kmodule_init);          ①
module_exit(kmodule_exit);          ②

MODULE_LICENSE("GPL");
```

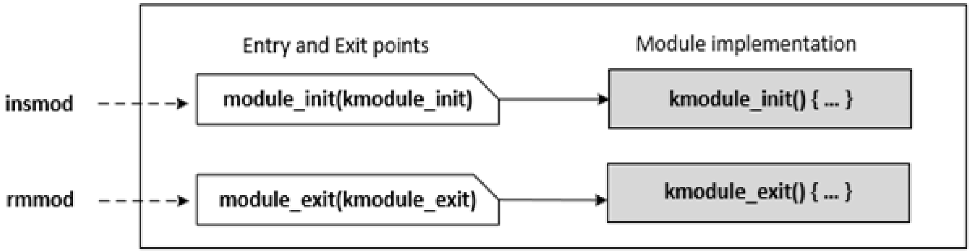
defining function

setup entry point

cleanup entry point

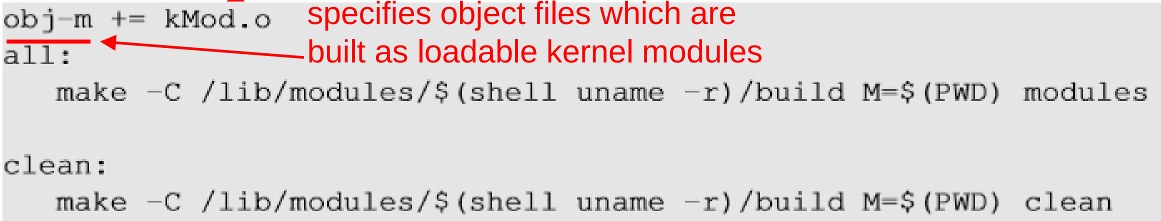
specify an initialization function that will be invoked when the kernel module is inserted.

specify a cleanup function that will be invoked when the kernel module is removed.



Compiling Kernel Modules

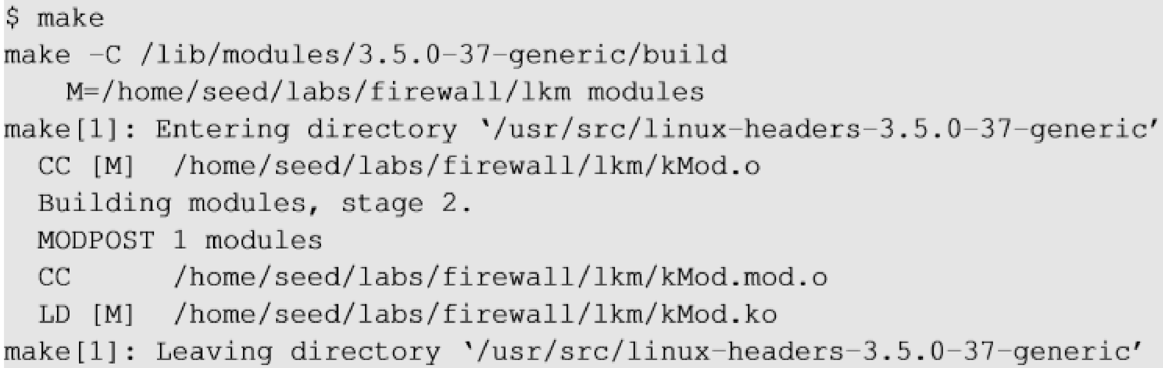
object file to be built



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

clean:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

specifies object files which are built as loadable kernel modules



```
$ make
make -C /lib/modules/3.5.0-37-generic/build
   M=/home/seed/labs/firewall/lkm modules
make[1]: Entering directory ‘/usr/src/linux-headers-3.5.0-37-generic’
  CC [M]  /home/seed/labs/firewall/lkm/kMod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/labs/firewall/lkm/kMod.mod.o
  LD [M]  /home/seed/labs/firewall/lkm/kMod.ko
make[1]: Leaving directory ‘/usr/src/linux-headers-3.5.0-37-generic’
```

Makefile

M: signifies that an external module is being built and tells the build environment where to place the built module file

-C: specify the directory of the library files for the kernel source

Installing Kernel Modules

insert modules into the kernel

```
// Insert the kernel module into the running kernel.
$ sudo insmod kMod.ko

// List kernel modules
$ lsmod | grep kMod
kMod                        12453   0

// Remove the specified module from the kernel.
$ sudo rmmod kMod
```

display the status of modules in the Linux kernel

filter the output with grep

remove a module from the kernel

```
$ dmesg
......
[65368.235725] Initializing this module
[65499.594389] Module cleanup
```

examine the kernel ring buffer and print the message buffer of kernel

in the sample code, we use printk() to print out messages to the kernel buffer
we can view the buffer using dmesg

Netfilter

- netfilter hooks in Linux: packet processing and filtering framework
- in Linux,
  - each protocol stack defines hooks along the packet's traversal path
    - hook is a location in the kernel that calls out of the kernel to a kernel module routine
  - developers use kernel modules to register callback functions to hooks
  - when packet arrives at a hook, the protocol stack call netfilter framework with the packet and hook number
  - netfiler checks if any kernel module has registered a call back function at this hook
  - each registered module will be called to analyze or manipulate packet, and return their verdict on packet

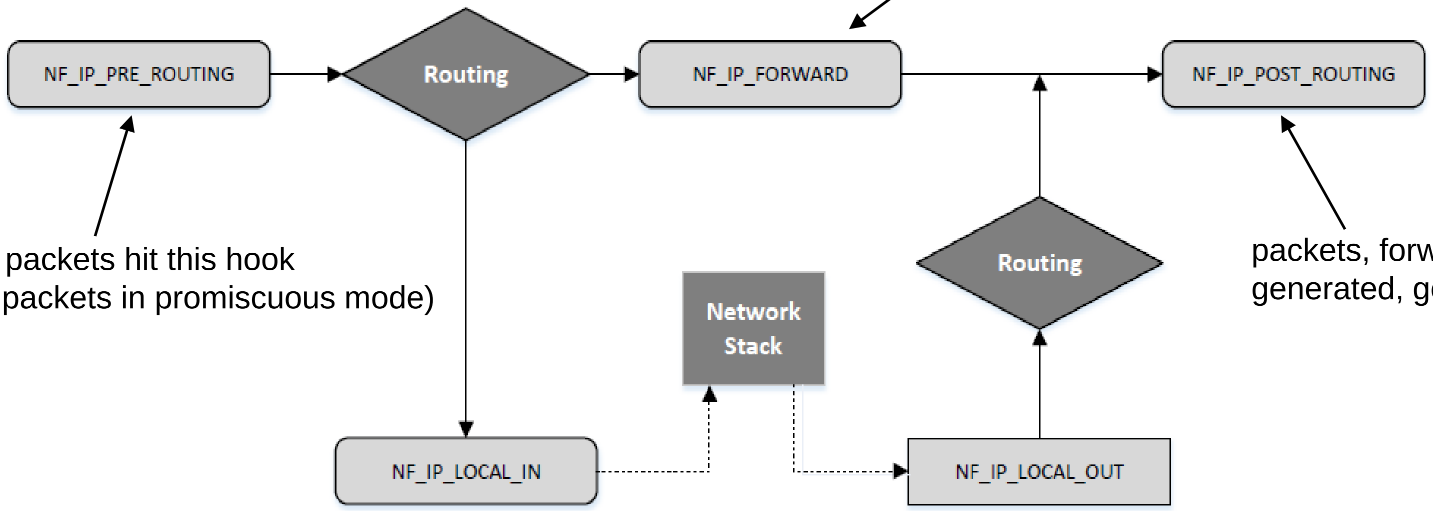
Netfilter (cont.)

- five return values of modules:
  - NF_ACCEPT: let the packet flow through the stack
  - NF_DROP: discard the packet
  - NF_QUEUE: pass the packet to the user space via nf_queue facility
    - perform packet handling in user space (asynchronous operation)
  - NF_STOLEN: inform the netfilter to forget about this packet, the packet is further processed by the module
  - NF_REPEAT: request the netfilter to call this module again

reference: https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-4.html
(Writing New Netfilter Modules)

Netfiler Hooks for IPv4

- Netfiler defines five hooks for IPv4:



move packets to other hosts (useful for firewall)

all incoming packets hit this hook
(not include packets in promiscuous mode)

packets, forwarded or
generated, going out of host

incoming packets will go through routing, which decides whether the packet is for

- other machines or
  - will go through forwarding path
- host itself
  - will go to next hook

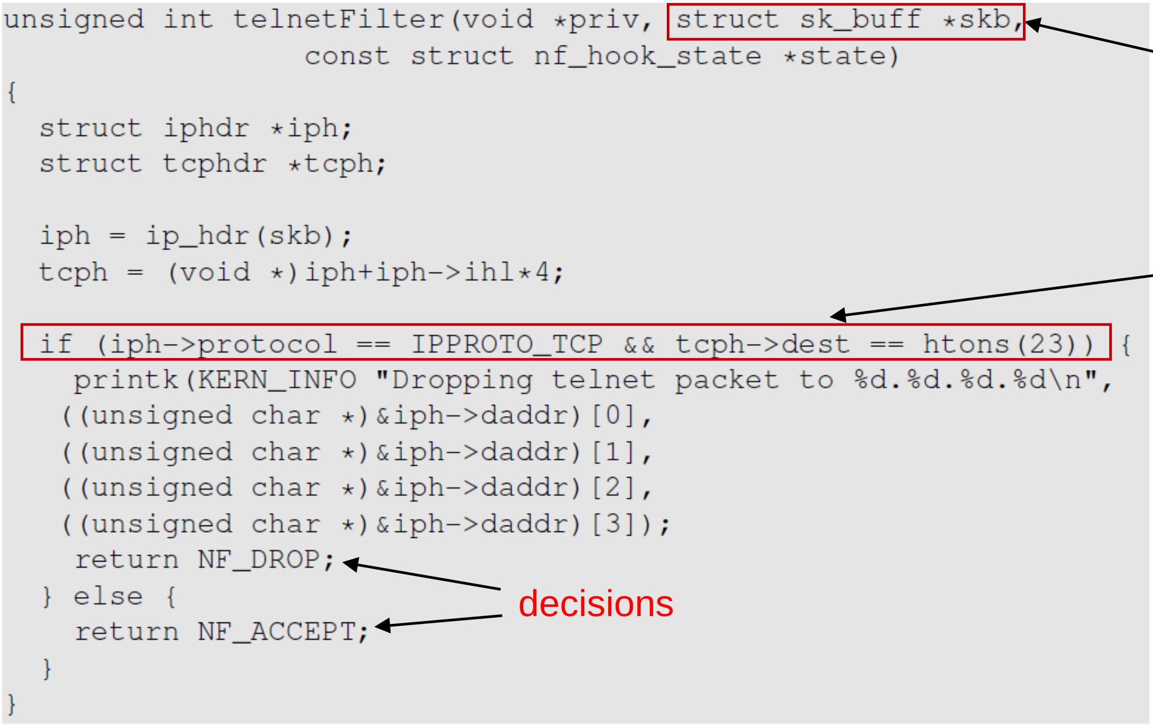
packets generated by local host reach this hook (the first hook on their way out of host)

Implementing Simple Packet Filter Firewall

- implementing a packet filter using netfilter framework and loadable kernel module
  - goals:
    - blocking all packets that are going out to port number 23
    - preventing users from using telnet to connect to other machines

Implementing Simple Packet Filter Firewall

- implementing a callback function, *telnetFilter*, for actual filtering
  - inspect packet (TCP header, port number)
    - if port # is 23, drop packet
    - otherwise, allow to pass



```
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                 const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
   ((unsigned char *)&iph->daddr)[0],
   ((unsigned char *)&iph->daddr)[1],
   ((unsigned char *)&iph->daddr)[2],
   ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}
```

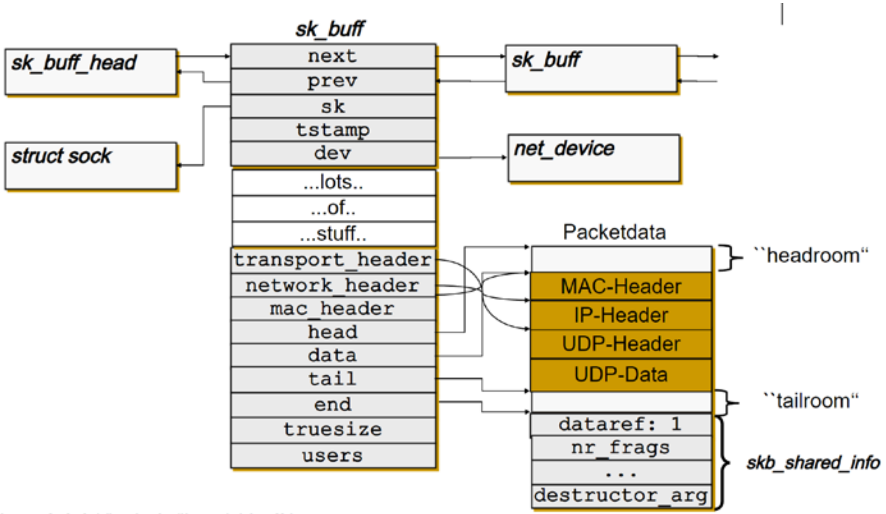
decisions

the entire packet is provided here.

the filtering logic is hardcoded here. Drop the packet if the destination TCP port is 23 (telnet)

Implementing Simple Packet Filter Firewall

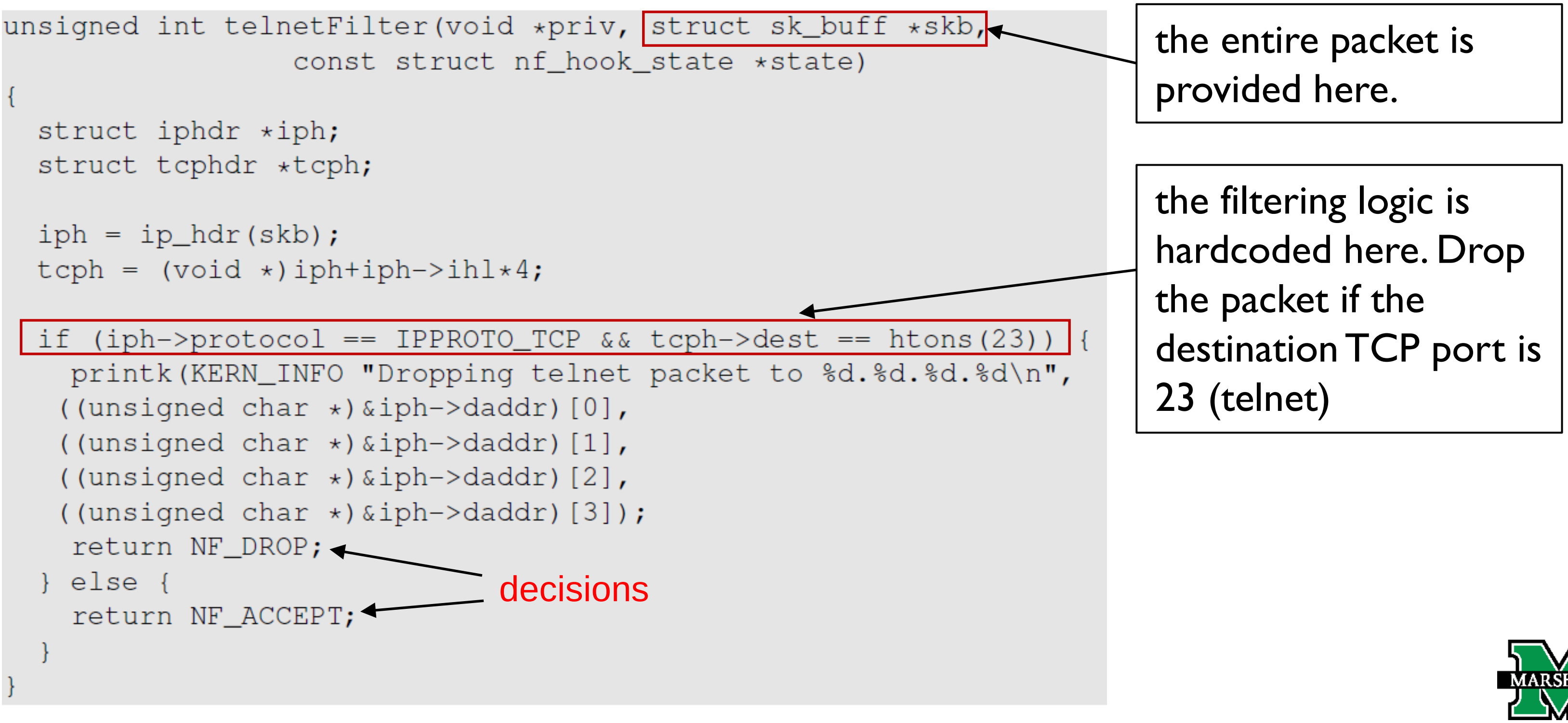
- struct sk_buff (means socket buffers)
  - core structure in Linux networking
  - socket buffers are the buffers where the Linux kernel handles network packets
    - packets are received by network card
    - put into a socket buffer
    - passed to network stack for processing



linux-2.6.31/include/linux/skbuff.h

Implementing Simple Packet Filter Firewall

- implementing a callback function, *telnetFilter*, for actual filtering
    - inspect packet (TCP header, port number)
        - if port # is 23, drop packet
        - otherwise, allow to pass



```
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                 const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
   ((unsigned char *)&iph->daddr)[0],
   ((unsigned char *)&iph->daddr)[1],
   ((unsigned char *)&iph->daddr)[2],
   ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}
```

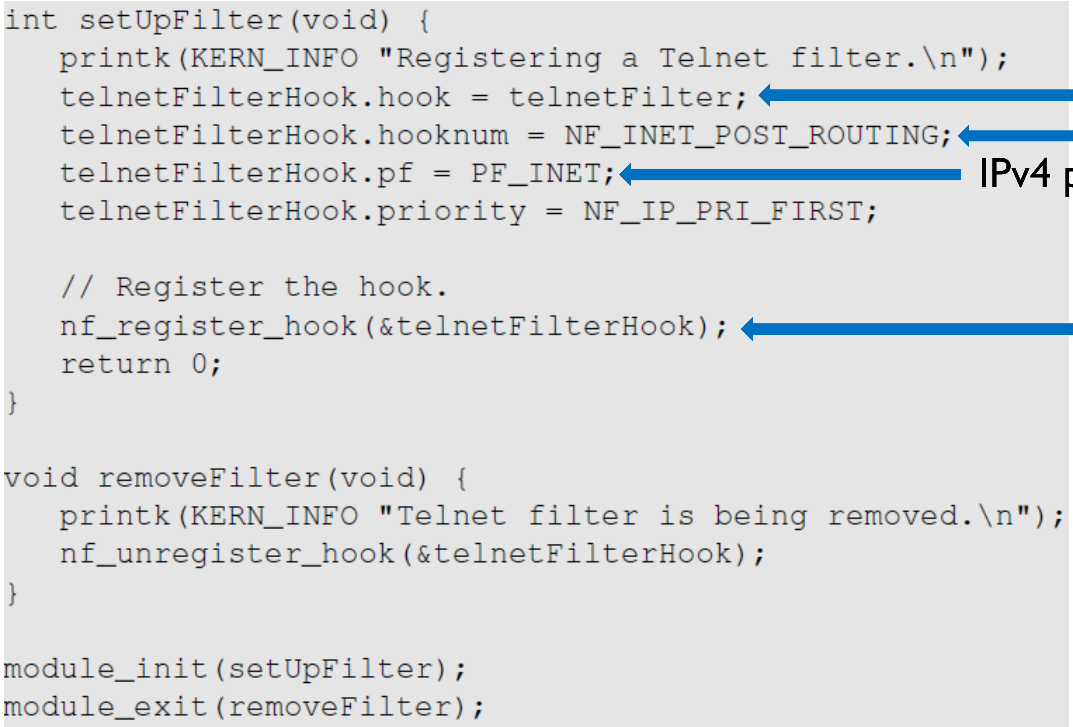
the entire packet is provided here.

the filtering logic is hardcoded here. Drop the packet if the destination TCP port is 23 (telnet)

decisions

Implementing Simple Packet Filter Firewall (cont.)

- hook previous function to one netfilter hook
  - use either NF_IP_LOCAL_OUT or NF_IP_POST_ROUTING



```
int setUpFilter(void) {
   printk(KERN_INFO "Registering a Telnet filter.\n");
   telnetFilterHook.hook = telnetFilter;
   telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
   telnetFilterHook.pf = PF_INET;
   telnetFilterHook.priority = NF_IP_PRI_FIRST;

   // Register the hook.
   nf_register_hook(&telnetFilterHook);
   return 0;
}

void removeFilter(void) {
   printk(KERN_INFO "Telnet filter is being removed.\n");
   nf_unregister_hook(&telnetFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);
```

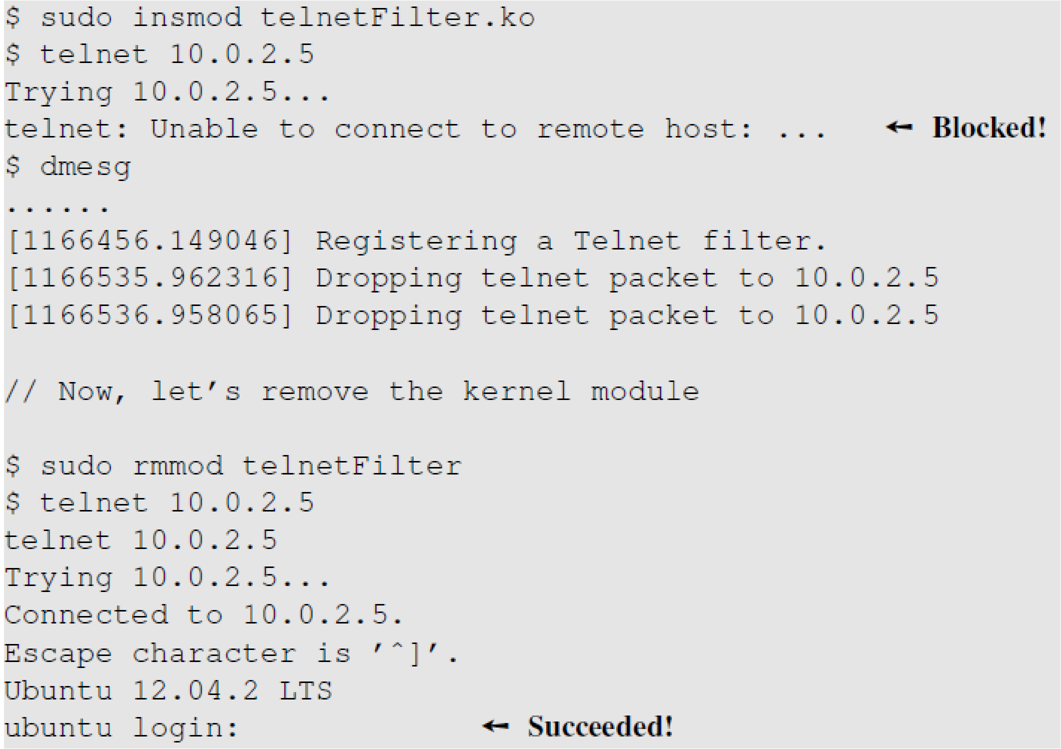
hook this callback function

use this netfilter hook

IPv4 packet family

register the hook

Testing Our Firewall



```
$ sudo insmod telnetFilter.ko
$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: ...    ← Blocked!
$ dmesg
......
[1166456.149046] Registering a Telnet filter.
[1166535.962316] Dropping telnet packet to 10.0.2.5
[1166536.958065] Dropping telnet packet to 10.0.2.5

// Now, let’s remove the kernel module

$ sudo rmmod telnetFilter
$ telnet 10.0.2.5
telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is ’^]’.
Ubuntu 12.04.2 LTS
ubuntu login:              ← Succeeded!
```