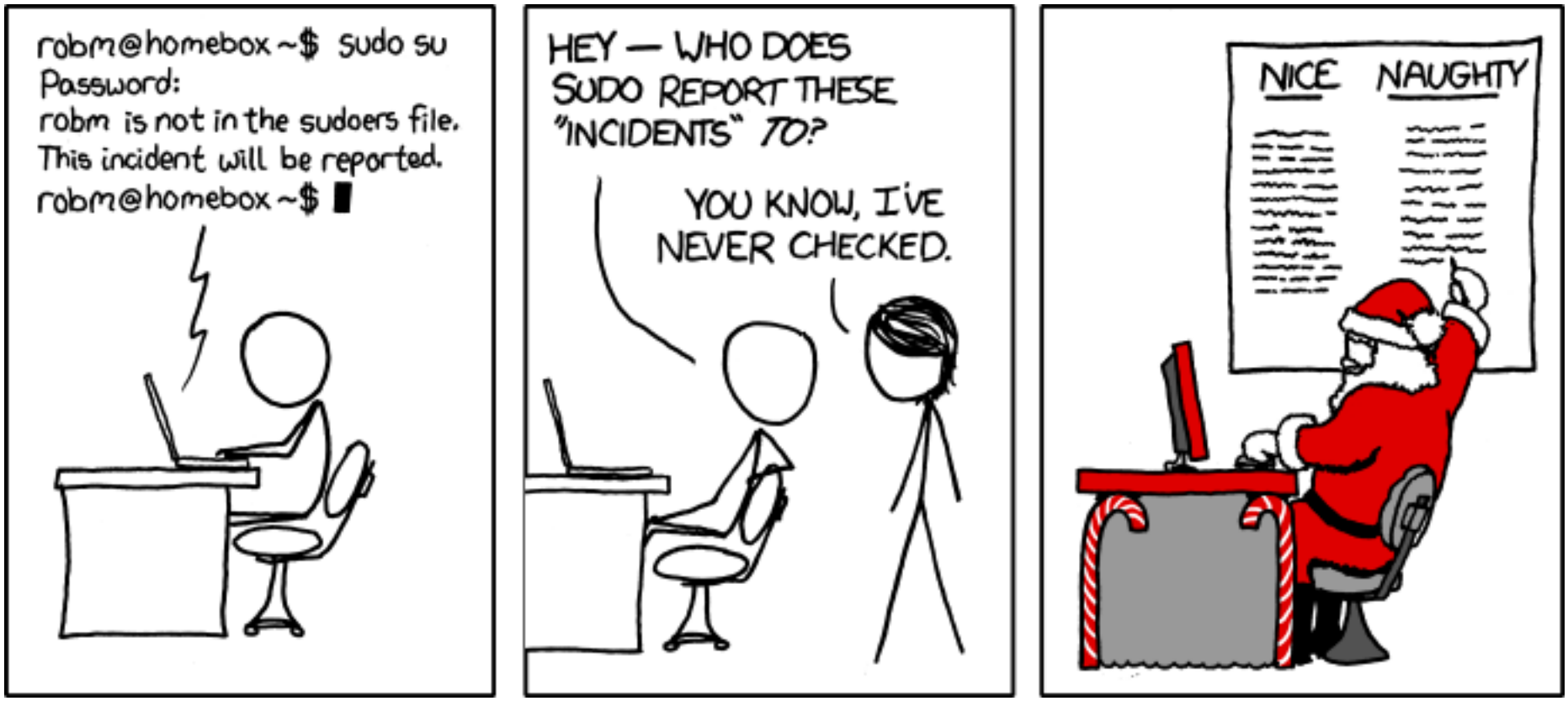
CS 55:
Security and Privacy

Firewalls

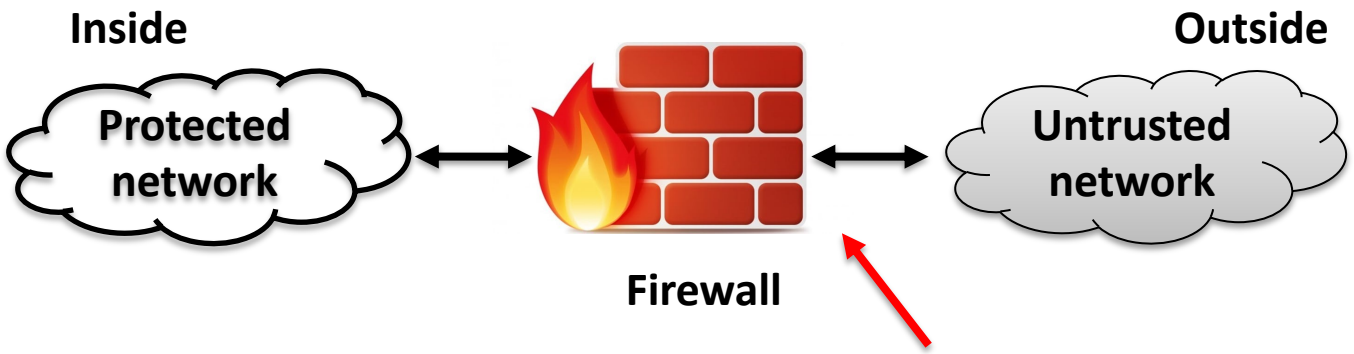


https://xkcd.com/838

Agenda

1. What are firewalls?
2. Building a simple firewall using Netfilter
3. Using iptables firewall
4. Stateful firewall

Firewalls are designed to stop unauthorized network traffic



Firewall separates trusted and untrusted network components

Often sold as a hardware appliance, but it is just a computer

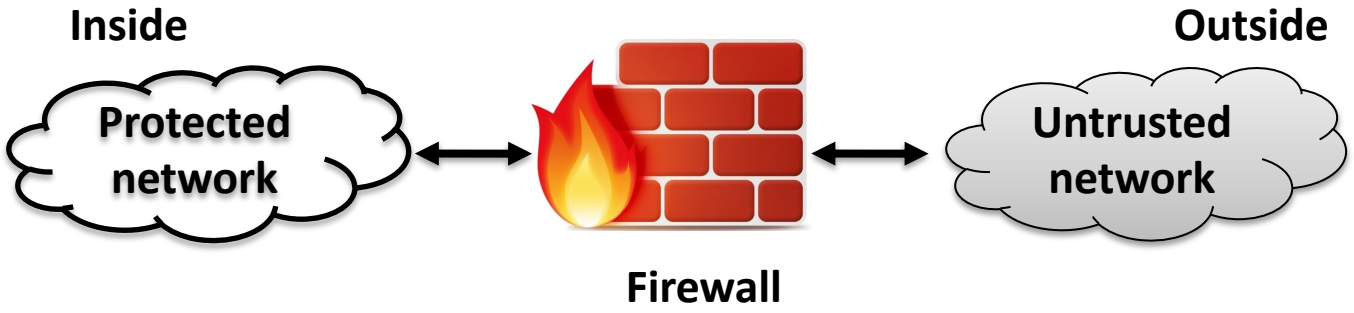
Main functions:

- Filtering packets
- Redirecting traffic
- Protecting against network attacks

Requirements

- Traffic between trust zones should flow through firewall
- Only authorized traffic passes
- Firewall itself must be hardened against attack

Firewalls are designed to stop unauthorized network traffic

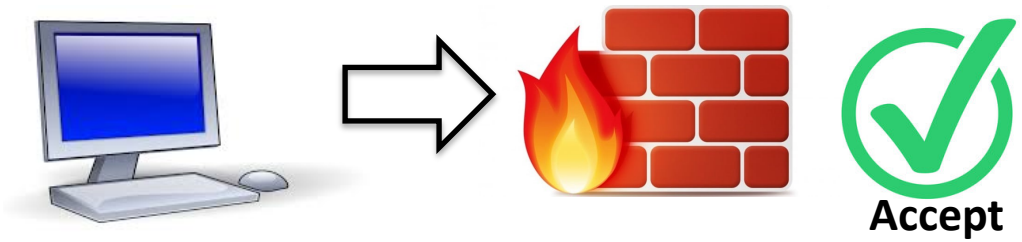


Firewall separates trusted and untrusted network components

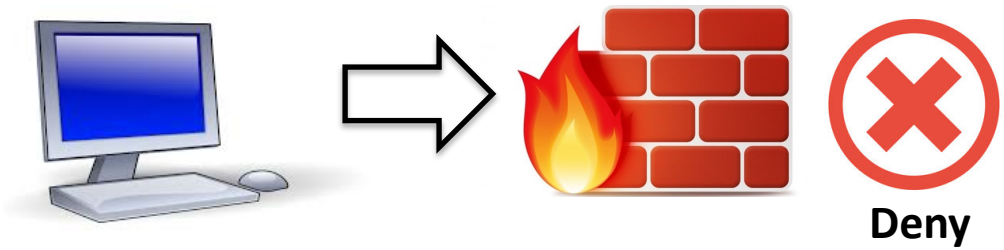
Firewall policy

- User control: Controls access based on the role of the user. Applied to users inside the firewall perimeter
- Service control: Controls access by the type of service offered by the host. Applied on the basis of network address, protocol of connection and port numbers
- Direction control: Determines the direction in which requests may be initiated and are allowed to flow through the firewall. It tells whether the traffic is "inbound" (from the outside to firewall) or "outbound" (from the inside to the firewall)

Firewalls can take one of three actions on a packet: accept, deny, or reject

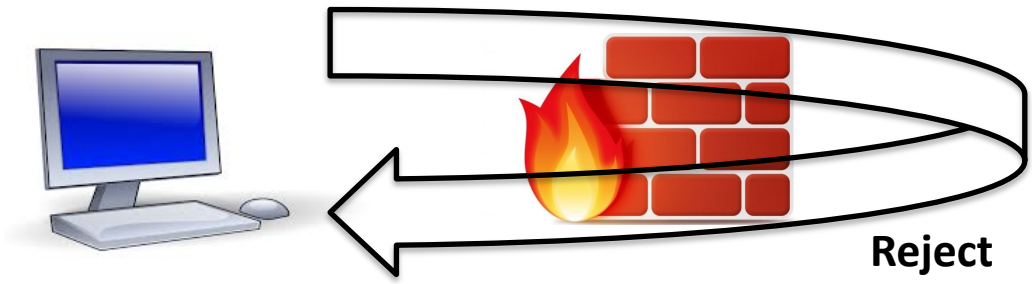


Accept
Traffic allowed through firewall



Deny
Traffic stopped at firewall

Filtering can be done on ingress or egress

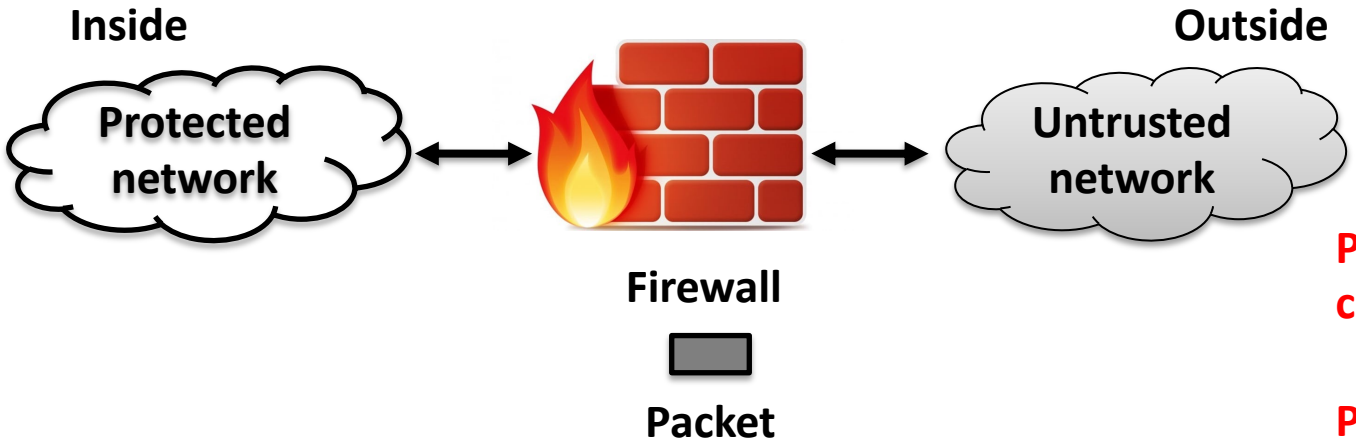


Reject
Traffic stopped at firewall and sender notified

There are three main types of firewalls

1. Packet filters
2. Stateful inspection
3. Application proxy

1) Packet filter firewalls make decisions based on a packet-by-packet basis

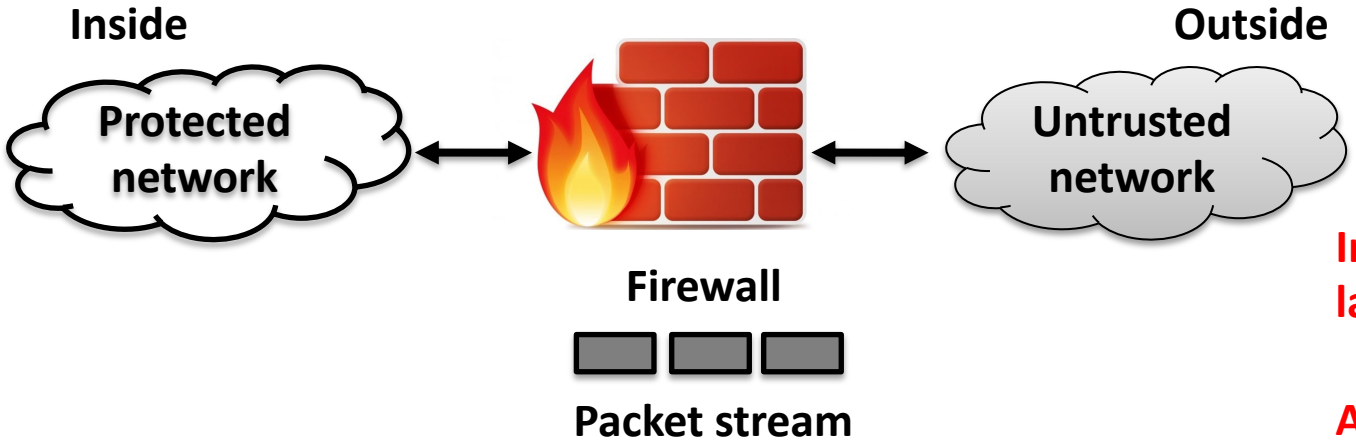


Packet filter firewalls also called Stateless Firewalls

Primary advantages: speed and cost

- Decisions made based on a *single* packet
- Controls traffic based on the information in packet headers up to layer 4, without looking into the payload that contains application data
- Does not consider if the packet is a part of existing stream of traffic
- Does not maintain info on connection state

2) Stateful firewalls make decisions based on a stream of packets



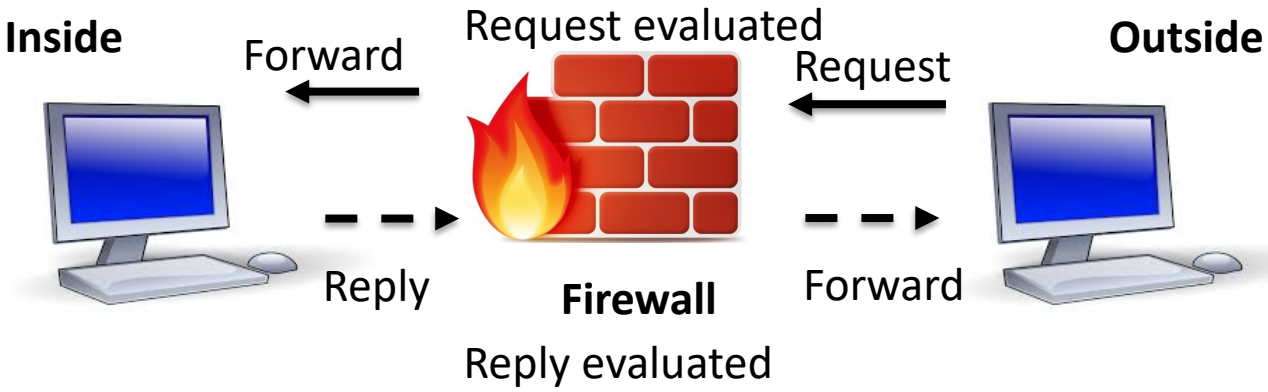
Inspects up to transport layer (layer 4)

Able to prevent many kinds of spoofing attacks

- Decisions based on a *stream* of packets
- Tracks the state of traffic by monitoring all the connection interactions until is closed
- Connection state table is maintained to understand the context of packets
- Example: connection are only allowed through the ports that hold open connections

Usually more expensive than packet filer firewalls

3) Application proxy firewalls control access to/from a service



Limitations:

- **Must implement proxy rules for each app**
- **Slower than other firewalls**

Nextgen firewalls also do intrusion detection/prevention, malware prevention, URL filtering, QoS

- Firewall controls I/O to/from application or service
- Acts as intermediary (no direct contact to app/service)
- Client connection terminates at firewall, separate connection initiated to application/service
- Data on connection analyzed up to application layer to determine if packet allow or denied/rejected
- Can prevent sensitive information leaks

Agenda

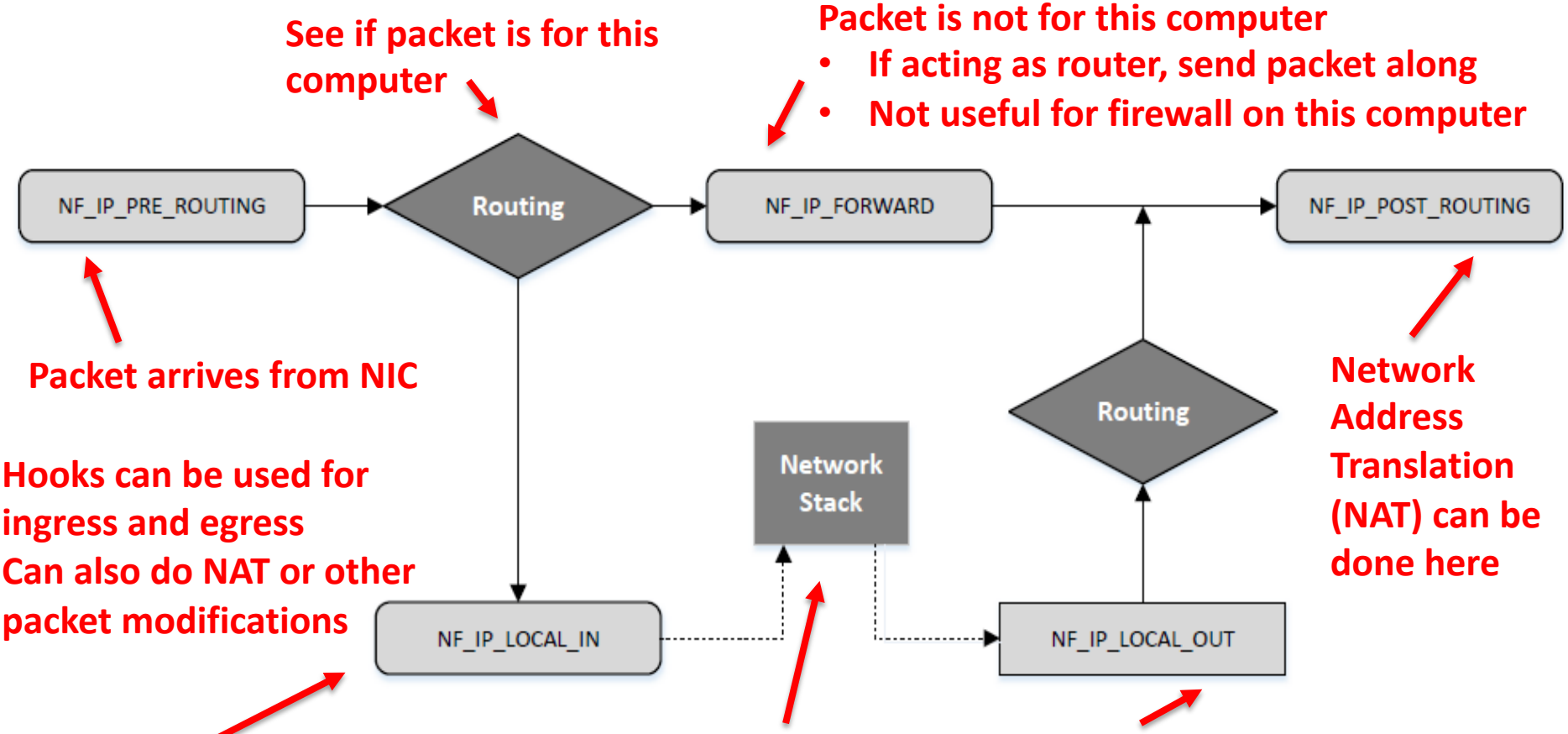
1. What are firewalls?
2. Building a simple firewall using Netfilter
3. Using iptables firewall
4. Stateful firewall

Linux provides two things useful for building a simple packet filter firewall

Packet filter firewall implementation in Linux

- Packet filtering must be done inside the kernel, user space will not be able to control packet flow
- Need changes in the kernel, two way to do this:
  - Netfilter: Provides hooks at critical points on the packet traversal path inside Linux Kernel
  - Loadable Kernel Modules: Allow privileged users to dynamically add/remove modules to the kernel, so there is no need to recompile the entire kernel

Netfilter hooks can call our code when a packet arrives at the hook



OS looks to see if code registered for callback, if so, call code when packet reaches hook

Code connected to a hook can render one of five decisions on a packet

1. `NF_ACCEPT`: Let the packet continue
2. `NF_DROP`: Discard the packet
3. `NF_QUEUE`: Pass the packet to the user space via nf_queue facility
4. `NF_STOLEN`: typically used to store fragmented packets so related packets can be analyzed together
5. `NF_REPEAT`: Request the netfilter to call this module again

Our code must run in the kernel; use Loadable Kernel Modules (LKMs)

- Loadable Kernel Modules allow us to change the kernel without the need to recompile the entire kernel
- Developers can use LKMs to register callback functions to these hooks
- When a packet arrives at a hook
  - Netfilter checks if any kernel module has registered a callback function at this hook
  - Registered modules will be called
  - Modules are free to analyze or manipulate the packet and return the verdict on the packet

LKMs have *init* and *exit* functions that fire when the modules is installed or removed

kMod.c

```
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);
module_exit(kmodule_exit);

MODULE_LICENSE("GPL");
```

Invoked when kernel module is loaded (sudo insmod kMod.ko)

Invoked when kernel module is removed (sudo rmmod kMod)

Note: printk, not printf!

Set init and exit modules

```
//insert our kernel module
$ make
$ sudo insmod kMod.ko

//list installed kernel modules
$ lsmod | grep kMod
kMod            16384     0

//remove our kernel module
$ sudo rmmod kMod

//check dmesg
$ dmesg | tail
[ 2969.190306] Initializing this module
[ 3005.730520] Module cleanup
```

Makefile

```
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
```

-C specifies directory of library files for kernel

M indicates external kernel module

We can use netfilter and LKMs to block outgoing telnet traffic

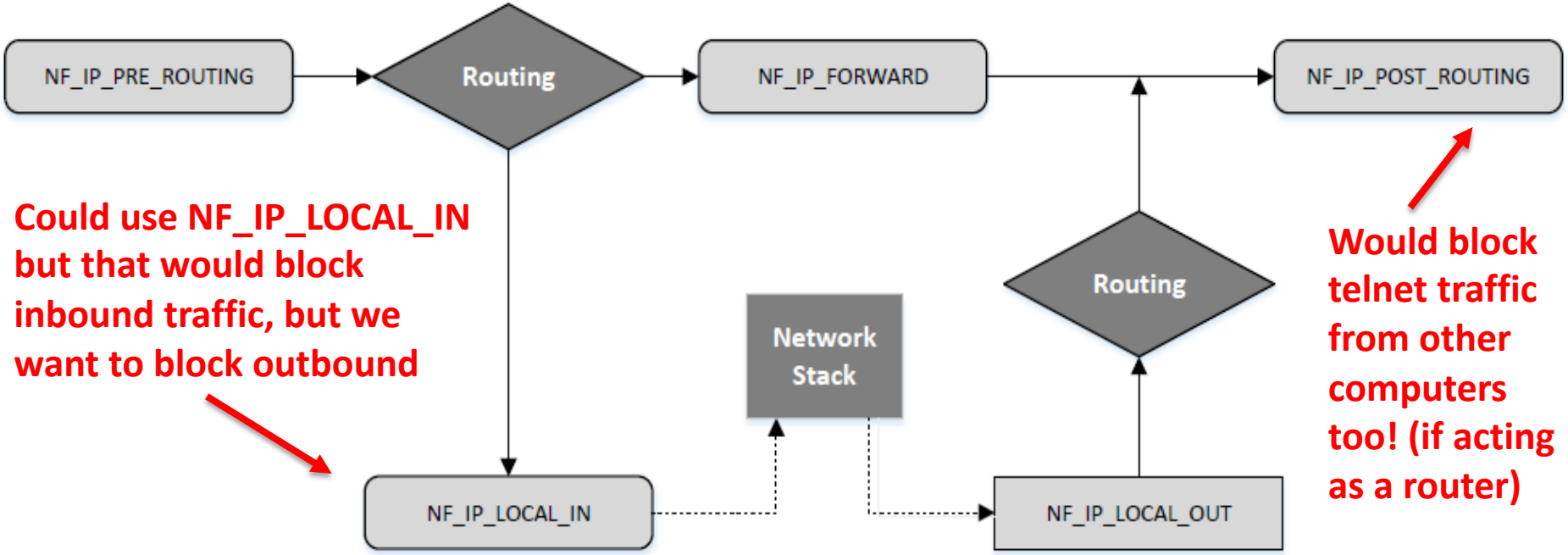
Goal: block outgoing *telnet* traffic

- TCP
- Uses port 23

We can use hooks with our LKMs to block outbound telnet traffic, but which one?

Goal: block outgoing telnet traffic (TCP on port 23)

Which hook should we use?



Packets traverse stages

Kernel looks for hooks registered at each stage

Make callback if registered

Callback returns NF_ACCEPT, NF_DROP (or other)

Implement a telnet filter with LKM

telnetFilter.c

```
static struct nf_hook_ops telnetFilterHook;

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);
```

Will fill this struct with needed data

Identify callback function (shown on next slide)

Identify hook to use (use NF_INET_LOCAL_IN to block inbound traffic)

Register callback

Remove callback

Set init and exit modules

Implement a telnet filter with LKM

telnetFilter.c

Hook from previous slide

```
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;
  }
}
```

Entire packet is here

Get IP and TCP headers

Drop if TCP and port 23, otherwise accept

Netfilter handles destroying packet (we don't have to do it ourselves)

DEMO

Primary computer (10.0.2.15)		Secondary computer (10.0.2.4)
		$ telnet 10.0.2.15 Log in with seed and dees (works) **$ exit**
# make and install firewall rule **$ cd ~/src/firewall/packet_filter/** **$ make** <builds> **$ sudo insmod telnetFilter.ko**	**Note: hook set to NF_INET_LOCAL_IN (blocking inbound traffic)**	
		$ telnet 10.0.2.15 Blocked
$ dmesg | tail Initializing this module Module cleanup Registering a Telnet filter. Dropping telnet packet to 10.0.2.15 # remove firewall rule **$ sudo rmmod telnetFilter**		
		$ telnet 10.0.2.15 Log in with seed and dees (works) **$ exit**

Agenda

1. What are firewalls?
2. Building a simple firewall using Netfilter
3. Using iptables firewall
4. Stateful firewall

Linux has a built-in firewall called *iptables* built on top of Netfilter

Netfilter

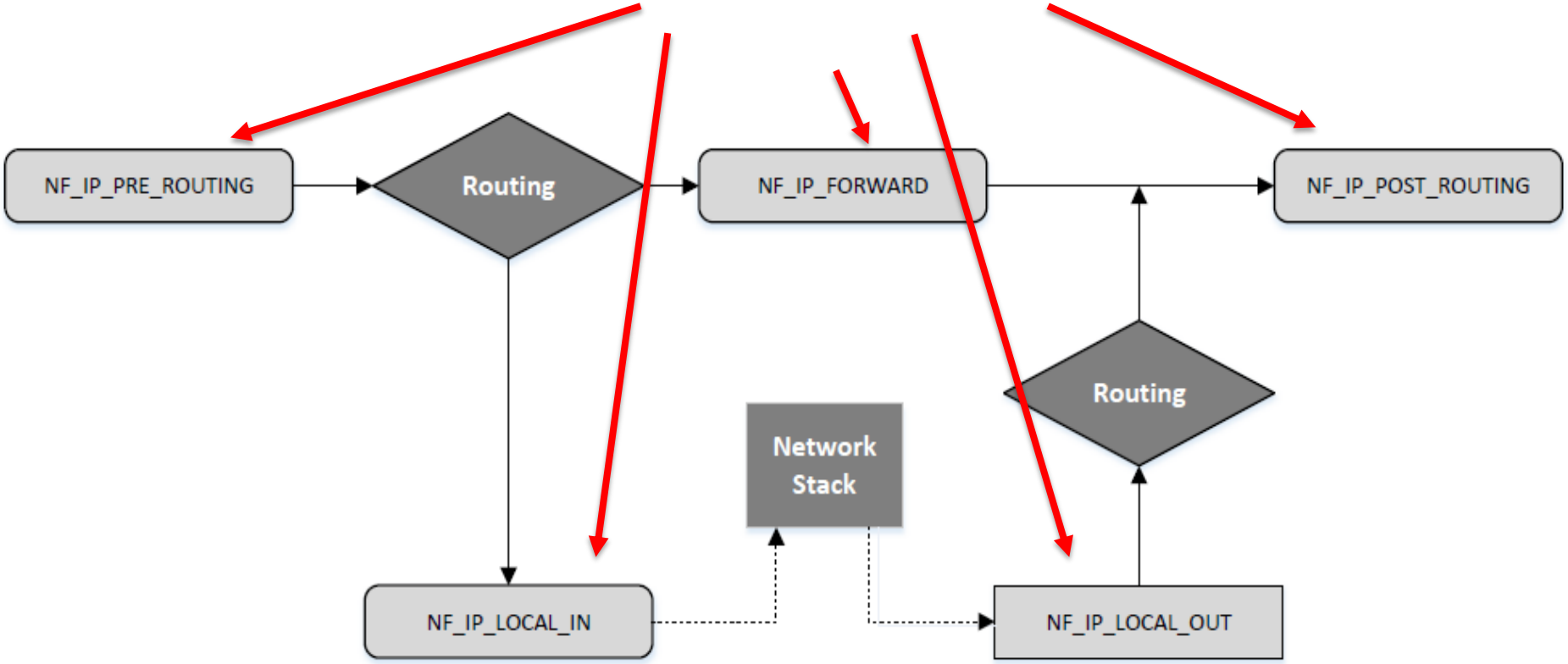
User space: iptables

Kernel space: xtables

Users enter commands into iptables, iptables inserts kernel code via xtables

No need to write and compile kernel modules

xtables has access to netfilter's 5 hooks



Iptables organizes functionality into tables and chains based on needed functionality

Chains correspond to netfilter hooks

INPUT = NF_IP_LOCAL_IN
FORWARD = NF_IP_FORWARD
OUTPUT = NF_IP_LOCAL_OUT

Table	Chain	Functionality
filter	INPUT FORWARD OUTPUT	Packet filtering
nat	PREROUTING INPUT OUTPUT POSTROUTING	Modifying source or destination network addresses
mangle	PREROUTING INPUT FORWARD OUTPUT POSTROUTING	Packet content modification

To do port forwarding (send packets to a different port) add hook to INPUT

There are a total of 5 hooks
But only 3 are meaningful for filtering applications
All 5 are available for changing (mangling) packets

If you want to implement a packet filter, put your rules in the filter table

If you want to change a packet, put your rules in the mangle table

If you want to do NAT….

Filtering applications only need 3 hooks, other applications may need more

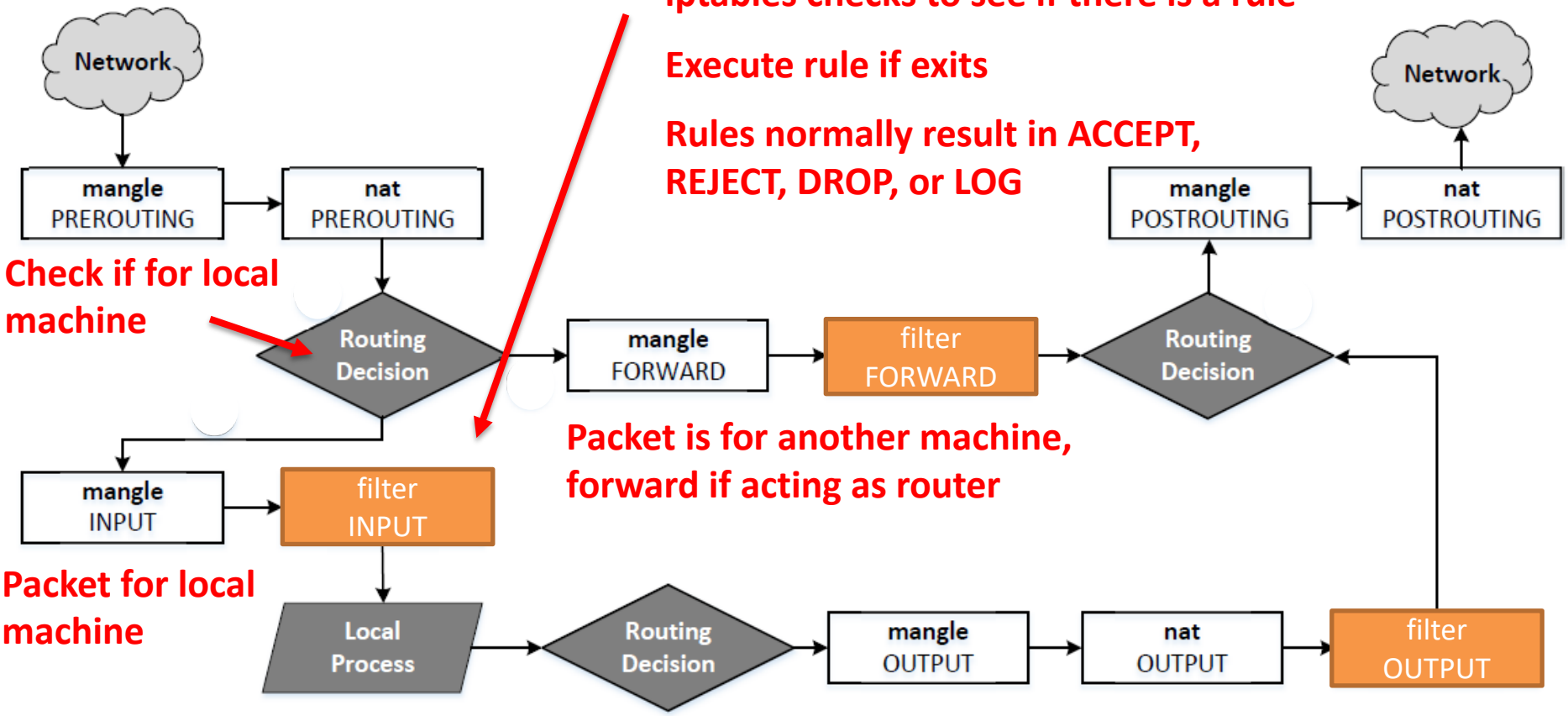
Iptables

Multiple tables per netfilter hook

As packets move through each chain iptables checks to see if there is a rule

Execute rule if exits

Rules normally result in ACCEPT, REJECT, DROP, or LOG



Iptables is powerful, but the commands to create rules initially look difficult

Iptables

Matching rule (most complicated part)

$ iptables [-t filter] –A CHAIN <rule> -j <action>

Command

Specify table
- **filter (default)**
- **nat**
- **mangle**

Which chain and which action apply
- **INPUT**
- **FORWARD**
- **OUTPUT**
- **PRE/POST ROUTING**

What to do with packets that match this rule

Flags
- **-A = append rule**
- **-C = check if rule exists**
- **-D = delete rule**
- **-P = change policy**
- **Others**

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Iptables is powerful, but the commands to create rules initially look difficult

Iptables

Matching rule (most complicated part)

$ iptables [-t filter] –A CHAIN <rule> -j <action>

Specifying rule:

-i incoming interface Layer 2
-o outgoing interface

-s source IP (/mask)
-d destination IP (/mask) Layer 3

-p protocol
    -p tcp –dport 22 Layer 4 Get help: iptables -p tcp -h

-m match extension
    -m owner –uid-owner bob

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Iptables is powerful, but the commands to create rules initially look difficult

Iptables

$ iptables [-t filter] –A CHAIN <rule> -j <action>

Specifying target action:

ACCEPT
REJECT
DROP
LOG
Target extension (e.g., NAT, TOS, TTL)

What to do with packets that match this rule

Example:
-j ACCEPT

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Example: block a specific IP address from communicating with a network

Block IP address

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –A INPUT –s 192.168.30.6 –d 192.1.0/16 -j DROP

Filter table used by default (if no –t)

Apply to INPUT chain

Look at packets coming from here

Going here (any address that starts with 192.1)

Drop packets like they're hot

Chain choices:
PREROUTING
INPUT
FORWARD
OUTPUT
POSTROUTING

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Example: Open SSH (port 22) and HTTP (port 80)

Open port 22 and 80

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –A INPUT –p tcp --dport 22 –j ACCEPT
$ sudo iptables –A INPUT –p tcp --dport 80 –j ACCEPT

Accept matching packets

Apply to transport layer TCP packets

Look at ports 22 and 80 (notice double dash)

All other traffic is not necessarily rejected, packets continue on path

- **Rules evaluated in order**
- **Once a packet matches a rule and is accepted, it proceeds on**
- **Packets matching this rule will be accepted and proceed on, even if later rules would drop them**

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Example: block user bob from sending any outbound packets

Allow all outgoing TCP traffic

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –A OUTPUT –m owner –uid-owner bob –j DROP

Apply to outbound traffic
Rules based on user only
Works for outbound traffic

Set owner to bob

Drop any of bob's outbound packets

Other users unaffected

Can we drop packets inbound for bob?
No! User is not part of packet;
firewall doesn't know who owns it

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

We can set the default behavior to ACCEPT or DROP packets with -P

Allow all outgoing TCP traffic

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –P INPUT DROP
$ sudo iptables –P OUTPUT DROP
$ sudo iptables –P FORWARD DROP

Default now is to drop all traffic unless allowed by other rules

Note: no –j on action

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Example: allow all outgoing TCP traffic

Allow all outgoing TCP traffic

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –A OUTPUT –p tcp –j ACCEPT

Apply to outbound traffic

Look TCP protocol

Accept matching packets

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

Example: add five to time-to-live (ttl)

Allow all outgoing TCP traffic

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables –t mangle –A PREOUTING –j TTL –ttl-inc 5

We are going to change packet, so use mangle table instead of filter table

Use PREROUTING so all packets are changed before moving on

Action is to add increase time to live by 5

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

We can flush all rules with -F

Allow all outgoing TCP traffic

$ iptables [-t filter] –A CHAIN <rule> -j <action>

$ sudo iptables -F

Removes rules (keeps default settings)
Can be done on per table basis

https://help.ubuntu.com/community/IptablesHowTo?action=show&redirect=Iptables

We can list and remove rules

list all the rules for a table

$ sudo iptables –t filter –L

$ sudo iptables –t filter –L --line-numbers

flash (remove) all rules for a table

$ sudo iptables –t nat –F

remove a single rule

$ sudo iptables –t filter –L –line-numbers

```
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:http

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
```

remove rule 1 from input chain on filter table

$ iptables –D INPUT 1

Can put commands into a shell script for convenience

Iptables is extremely powerful, we've just barely scratched the surface today

To reset to default, run this shell script

clean_up.sh

```
#!/bin/sh

# Set up all the default policies to ACCEPT packets
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT

#Flush all existing configurations.
iptables -F
```


https://github.com/kevin-w-du/BookCode/blob/master/Firewall/iptables/cleanup.sh

Agenda

1. What are firewalls?
2. Building a simple firewall using Netfilter
3. Using iptables firewall
4. Stateful firewall

Looking at the context of a packet can provide better control than just filtering

```
//open port 22 (ssh) and 80 (http)
$ sudo iptables –A INPUT –p tcp --dport 22 –j ACCEPT
$ sudo iptables –A INPUT –p tcp --dport 80 –j ACCEPT
```

Our previous example allowed all communication over port 22 and 80
An adversary can send TCP packets out port 80, but to clients other than one that has established a connection
Add stateful inspect to prevent communication to clients that have not completed 3-way handshake
States are:

- NEW: Connection starting 3-way TCP handshake
- ESTABLISHED: Connection established
- RELATED: Establishes relationship between connections
- INVALID: Used for packets that do not follow protocol

Looking at the context of a packet can provide better control than just filtering

```
//open port 22 (ssh) and 80 (http)
$ sudo iptables –A INPUT –p tcp --dport 22 –j ACCEPT
$ sudo iptables –A INPUT –p tcp --dport 80 –j ACCEPT
$ sudo iptables –A INPUT –p tcp –m conntrack
                        --ctstate ESTABLISHED, RELATED – j ACCEPT
```

Our previous example allowed all communication over port 22 and 80

An adversary can send TCP packets out port 80, but to clients other than one that has established a connection

Add stateful inspect to prevent communication to clients that have not completed 3-way handshake

States are:

- NEW: Connection starting 3-way TCP handshake
- ESTABLISHED: Connection established
- RELATED: Establishes relationship between connections
- INVALID: Used for packets that do not follow protocol

Conntrack package for iptables allow state tracking