Lecture 18: Packet Filtering Firewalls (Linux)

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

March 30, 2011

2:10pm

Goals:

- Packet-filtering vs. proxy-server firewalls

- The four iptables supported by the Linux kernel: **filter**, **nat**, **mangle**, and **raw**

- Creating and installing new firewall rules

- Structure of the **filter** table

- Connection tracking and extension modules

- Designing your own filtering firewall

18.1:   Firewalls in General

- Two primary types of firewalls are

  - packet filtering firewalls

  - proxy-server firewalls

  Sometimes both are employed to protect a network. A single computer may serve both roles.

- A proxy-server firewall handles various network services itself rather then passing them straight through. What exactly that means will be explained in the lecture on proxy server firewalls.

- In Linux, a packet filtering firewall is configured with the iptables modules.

- To use iptables for a packet filtering firewall, you need a Linux kernel which has the **netfilter infrastructure** in it: **netfilter** is a general framework inside the Linux kernel which other things (such as the **iptables module**) can plug into. This means you need kernel 2.3.15 or above, and answer 'Y' to **CON-**

FIG_NETFILTER in the kernel configuration. (The tool **iptables** talks to the kernel and tells it what packets to filter.)

- The **iptables** tool inserts and deletes rules from the kernel's packet filtering table. Ordinarily, these rules created by the **iptables** command would be lost on reboot. However, you can make the rules permanent with the commands **iptables-save** and **iptables-restore**. The other way is to put the commands required to set up your rules in an initialization script.

- Note that **iptables** replaces the older **ipfwadm** and **ipchains**, and that the **iptables** based firewalls are very similar to the previous **ipchains** based firewalls.

- Rusty Russell wrote **iptables**. He is also the author of **ipchains** that was incorporated in version 2.2 of the kernel and that was replaced by **iptables** in version 2.4.

18.2:   A "Demo" to Motivate You to Use Iptables

- The **iptables** command with all its options can appear at first sight to be daunting to use. The "demo" presented in this section illustrates how easy it is to use this command. Basically, I will show how you can create a **single-rule firewall** to achieve some pretty amazing protection for your computer.

- If you do not need this sort of a motivation, proceed directly to Section 18.3.

- The "demo" will consist of showing the following:

  - **Demo Goal 1:** How you can prevent anyone from "pinging" your machine.

  - **Demo Goal 2:** How you can allow others to ssh into your machine, but block it for every other access.

  - **Demo Goal 3:** How you can prevent others from sending connection-initiation packets to your machine.

- **ASSUMPTIONS:** For this "demo" I will assume that you are sitting in front of two machines, of which at least one is running the Ubuntu distribution of Linux. Obviously, I am also assuming that both machines are connected to the network. The machine that needs to be protected with a firewall will be referred to as the Ubuntu laptop.

- **When you installed Ubuntu on your laptop, that automatically activated the `iptables` firewall** — although with rules that do **not** carry out any packet filtering. To see this, when you execute the following command **as root** on your Ubuntu laptop:

      iptables -L

  you will see the following sort of output in the terminal window:

      Chain INPUT (policy ACCEPT)
      target   prot   opt   source   destination

      Chain FORWARD (policy ACCEPT)
      target   prot   opt   source   destination

      Chain OUTPUT (policy ACCEPT)
      target   prot   opt   source   destination

  This output tells that **iptables** is on and running, but there are no rules in the firewall at this time. As to what is meant by `target`, `prot`, `opt`, etc., in the output shown above will be explained in Section 18.6.

- **More precisely speaking**, the above output tells us that there are currently no rules in the **filter table** of the firewall. So, as far as the firewall is concerned, every packet will be subject to the policy `ACCEPT`. That is, every packet will get to its destination, coming in or going out, unhindered.

- Later in this lecture, I will talk about the fact the **iptables** supports four tables: **filter**, **mangle**, **nat**, and **raw**. I will also mention later that the command '`iptables -L`' is really a short form for the more table-specific command '`iptables -L -t filter`' for examining the contents of the **filter** table. [So the output shown previously tells us that there is currently nothing in only the **filter** table. But note that the packets may still be subject to filtering by the rules in the other tables. Later in this demo I will show an example in which the packets of a certain kind will be denied entry into the Ubuntu laptop even when the **filter** table has nothing in it.]

- If the output you see for the '`iptables -L`' command is different from what I have shown on the previous slide, please flush the **filter** table (meaning get rid of the rules in the **filter** table) by

        iptables -F

  For this demo to work as I will present it, ideally you should be flushing (after you have saved the rules by `iptables-save` using the syntax I will show later) all of the rules in all of the tables by

        iptables -t filter -F
        iptables -t filter -X
        iptables -t mangle -F

```
iptables -t mangle -X
iptables -t nat    -F
iptables -t nat    -X
iptables -t raw    -F
iptables -t raw    -X
```

The '-X' option is for deleting user-defined chains. I will explain later what that means.

- **Achieving Demo Goal 1:**

- Now let's go to the **first goal of this demo**: You don't want others to be able to **ping** your Ubuntu laptop.

- As root, execute the following in the command line

```
iptables -A INPUT  -p icmp --icmp-type echo-request -j DROP
```

where the '`-A INPUT`' option says to append a new rule to the **INPUT** chain of the **filter** table. The '`-p icmp`' option specifies that the rule is to be applied to ICMP packets only. The next option mentions what specific subtype of the ICMP packets this rule applies to. Finally, '`-j DROP`' specifies the action to be taken for such packets. [As I will explain later, the above command enters a rule in the **INPUT** chain of the **filter** table. This rule says to **drop** all incoming `icmp` packets that are of the type `echo-request`. As stated in Section 18.10 of this lecture, that is the type of ping ICMP packets.]

- Now use the other machine to ping the Ubuntu laptop by using either the 'ping hostname' syntax or the 'ping xxx.xxx.xxx.xxx' syntax where the argument to ping is the IP address. You will notice that you will **not** get back any echos from the Ubuntu machine. If you had pinged the Ubuntu machine prior to the entry of the above firewall rule, you would have received a normal echo from that machine.

- To get ready for our second demo goal, now delete the rule you entered above by

  ```
  iptables -F
  ```

  Subsequently, if you execute 'iptables -L' again, you will see again the empty chains of the **filter** table.

- **Achieving Demo Goal 2:**

- Recall that the objective now is to allow others to ssh into our Ubuntu laptop, but we we do not want the Ubuntu laptop to respond to any other service request coming from other computers. I am assuming that the SSH server **sshd** is running on the Ubuntu laptop.

- As root, execute the following two lines in your Ubuntu laptop:

```
iptables -A INPUT -p tcp --destination-port 22 -j ACCEPT

iptables -A INPUT -j REJECT
```

where the '`-A INPUT`' option says to append the rules to the **INPUT** chain of the `filter` table. The '`-p tcp`' option says the rule is to be applied to TCP packets. The next option mentions the destination port on the local machine for these incoming packets. Finally, the option '`-j ACCEPT`' says to accept all such packets. Recall that 22 is the port registered for the SSH service.

- To see that you have entered two new rules in the **INPUT** chain of the **filter** table, execute the '`iptables -L`' command as root. You should see the following:

```
Chain INPUT (policy ACCEPT)
target     prot opt  source          destination

ACCEPT     tcp  --   anywhere        anywhere        tcp dpt:ssh
REJECT     0    --   anywhere        anywhere        reject-with icmp-port-unreachable


Chain FORWARD (policy ACCEPT)
target     prot opt  source          destination


Chain OUTPUT (policy ACCEPT)
target     prot opt  source          destination
```

- Now when you use the other laptop to ssh into the Ubuntu laptop with its firewall set as above, you should experience no prob-

lems. However, if the other laptop makes any other type of access (such as by ping) to the Ubuntu laptop, you will receive "Port Unreachable" error message. If we had used **DROP** instead of **REJECT** in the second rule we entered with the `iptables` command, when the other laptop makes any access other than ssh to the Ubuntu laptop, the other laptop would not receive back any error messages. [When we entered the second `iptables` command line, we did **not** specify the `-reject-with` option, yet it shows up in the second rule in the `filter` table. Note that, as opposed to **DROP**, the job of **REJECT** is to send back an error message. If you don't specify what this error message should be, **iptables** will by default use the `icmp-port-unreachable` option that sends back the `Dest Unreachable` message.]

- To see the effect of the second rule — the `REJECT` rule — try pinging the Ubuntu laptop and see what happens. The machine that is doing the pinging will receive and display a 'Destination Port Unreachable' message.

- To get ready for our third demo goal, now delete the two rules you entered above by

      iptables -F

  Subsequently, if you execute 'iptables -L' again, you will see again the empty chains of the **filter** table.

- **Achieving Demo Goal 3:**

- Recall that the goal of this part of the demo is to reject all requests for new connections coming from other hosts in the network. As mentioned in Lecture 16, when a host wants to make a new connection with your machine, it sends your machine a **SYN** packet. To block all such packets, we could use a rule very similar to what we have shown so far. But, just to add an interesting twist to the demo, we will use the **mangle** table for the purpose. So go ahead and execute the following command line as root:

  ```
  iptables -t mangle -A PREROUTING -p tcp -m tcp --tcp-flags SYN NONE -j DROP
  ```

  The '-t' option says that the new rule is meant for the **mangle** table. We want the rule to be appended to the **PREROUTING** chain (assuming that this chain was empty previously). You can check that the rule is in the **mangle** table by executing the command

  ```
  iptables -t mangle -L
  ```

- With the above rule in place in the **mangle** table, use the other laptop to try to make any sort of connection with the Ubuntu laptop. You could, for example, try to SSH into the Ubuntu laptop. You will not be able to do. (You will still be able the ping the Ubuntu laptop since ping packets do not have the **SYN** flag set.)

11

- Finally, restore the Ubuntu laptop's firewall to its original all-accepting condition by deleting the rule you just entered in the **mangle** table:

```
iptables -t mangle -F
```

18.3: The Four Tables Maintained by the Linux Kernel for Packet Processing

- Linux kernel uses the following **four tables**, each consisting of rule chains, for processing the incoming and outgoing packets:

  - the **filter** table

  - the **nat** table

  - the **mangle** table

  - the **raw** table

- Each table consists of **chains of rules**.

- Each packet is subject to each of the rules in a table and the fate of the packet is decided by the first matching rule.

- The **filter** table contains at least three rule chains: **INPUT** for processing all incoming packets, **OUTPUT** for processing all

outgoing packets, and **FORWARD** for processing all packets being routed through the machine. The **INPUT**, **OUTPUT**, and **FORWARD** chains of the **filter** table are also referred to as the **built-in chains** since they cannot be deleted (unlike the user-defined chains we will talk about later).

- The **nat** table is consulted when a packet that creates a new connection is encountered. **nat** stands for Network Address Translation. When your machine acts as a router, it would need to alter either the source IP address in the packet passing through, or the destination IP address, or both. That is where the **nat** table is useful. The **nat** table also consists of three built-in chains: **PREROUTING** for altering packets as soon as they come in, **OUTPUT** for altering locally-generated packets before routing, and **POSTROUTING** for altering packets as they are about to go out.

- The **mangle** table is used for specialized packet alteration. (Demo 3 in Section 18.2 inserted a new rule in the **mangle** table.) The **mangle** table has five rule chains: **PREROUTING** for altering incoming packets before a routing decision is made concerning the packet, **OUTPUT** for altering locally generated outgoing packets, **INPUT** for altering packets coming into the machine itself, **FORWARD** for altering packets being routed through the machine, and **POSTROUTING** for altering packets im-

mediately after the routing decision.

- The **raw** table is used for configuring exemptions to connection tracking. As to what is meant by connection tracking will become clear later. **When a `raw` table is present, it takes priority over all other tables.**

- We will focus most of our attention on the **filter** table since that is usually the most important table for firewall security — particularly if your focus is on protecting your laptop with a firewall of your own design.

18.4:   How Packets are Processed by the `filter` Table

- As mentioned already, the **filter** table contains the following built-in rule chains: **INPUT**, **OUTPUT**, and **FORWARD**.

- The figure below shows how a packet is subject to these rule chains:
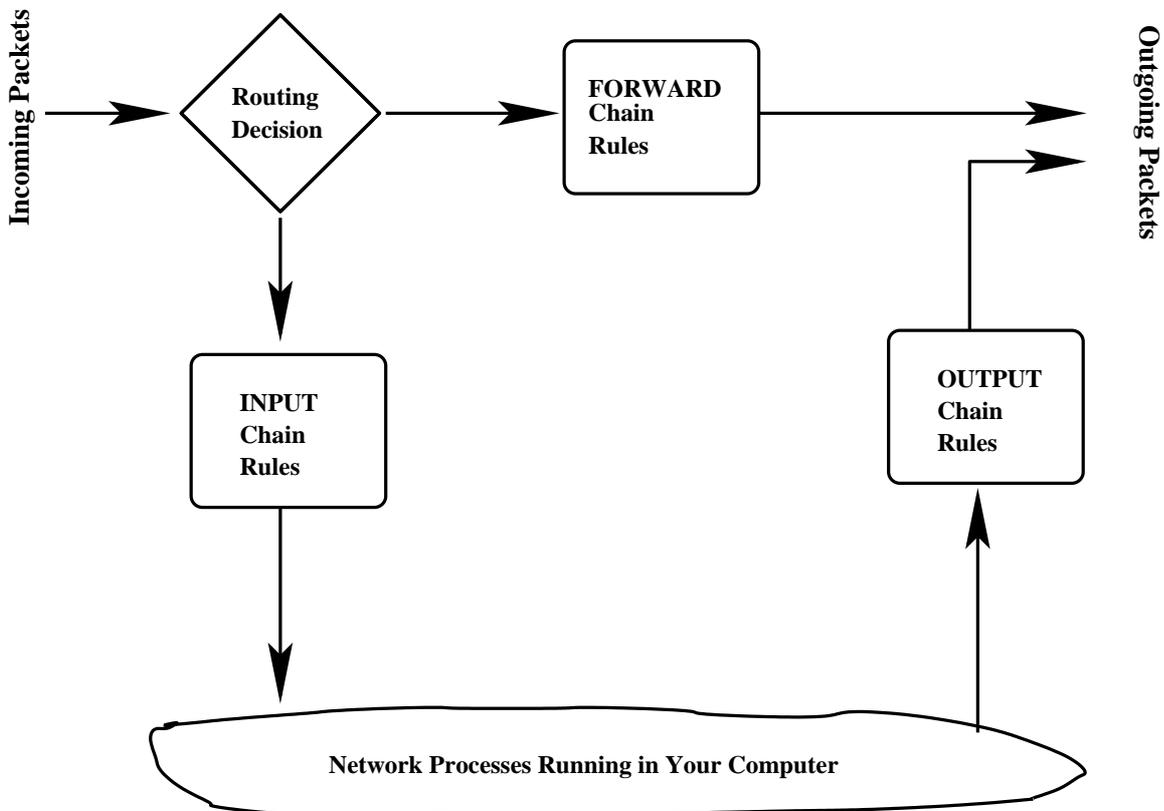
- When a packet comes in (say, through the Ethernet interface) the kernel first looks at the destination of the packet. This step is labeled 'routing' in the figure.

- If the routing decision is that the packet is intended for the machine in which the packet is being processed, the packet passes downwards in the diagram to the **INPUT** chain.

- If the incoming packet is destined for another network interface on the machine, then the packet goes rightward in our diagram to the **FORWARD** chain. If accepted by the **FORWARD** chain, the packet is sent to the other interface. (But note that if the kernel does not have forwarding enabled or if the kernel does not know how to forward the packet, the packet would be dropped.)

- If a program running on the computer wants to send a packet out of the machine, the packet must traverse through the **OUTPUT** chain of rules. If it is accepted by any of the rules, it is sent to whatever interface the packet is intended for.

- Each rule in a chain will, in general, examine the packet header. If the condition part of the rule matches the packet header, the action specified by the rule is taken. Otherwise, the packet moves

on to the next rule.

- If a packet reaches the end of a chain, then the Linux kernel looks at what is known as the **chain policy** to determine the fate of the packet. **In a security-conscious system, this policy usually tells the kernel to DROP the packet.**

18.5:   To See if `iptables` is Installed and Running

- Execute as root

  ```
  lsmod | grep ip
  ```

  where `lsmod` shows you what kernel modules are currently loaded in. On my laptop running Ubuntu Linux, this returns

  ```
  iptable_raw            3328  0
  ipt_REJECT             5760  0
  iptable_mangle         3840  0
  iptable_nat            8708  0
  nf_nat                20140  1 iptable_nat
  nf_conntrack_ipv4     19724  2 iptable_nat
  nf_conntrack          65288  4 xt_state,iptable_nat,nf_nat,nf_conntrack_ipv4
  nfnetlink              6936  3 nf_nat,nf_conntrack_ipv4,nf_conntrack
  iptable_filter         3968  1
  ip_tables             13924  4 iptable_raw,iptable_mangle,iptable_nat,iptable_filter
  x_tables              16260  5 ipt_REJECT,xt_state,xt_tcpudp,iptable_nat,ip_tables
  ipv6                 273892 21
  ```

  If you do not see all these modules, that does **not** mean that **iptables** is not installed and running on your machine. Many of the kernel modules are loaded in dynamically as they are needed by the application programs. [For Red Hat users, If you see the `ipchains` module running, as you might with older installations of Linux, unload that module. You can do that by first stopping `ipchains`:

  ```
  /etc/init.d/ipchains stop
  ```

  followed by
  ```
  modprobe -r ipchains
  ```

19

]

- Another way to see if `iptables` is installed and running, execute as root

      iptables -L

  On my Ubuntu laptop, this command line returns (assuming this is your very first invocation of the **iptables** command):

  ```
  Chain INPUT (policy ACCEPT)
  target     prot opt source              destination


  Chain FORWARD (policy ACCEPT)
  target     prot opt source              destination


  Chain OUTPUT (policy ACCEPT)
  target     prot opt source              destination
  ```

  This output means that **iptables** is up and running, although at this time it has **no** rules in it for packet filtering. When **iptables** is loaded with firewall rules, they would show up under one or more of the **chains** listed above.

- The invocation `iptables -L` shows only the **filter** table. In general, if you want to see the rules in a particular table, you would call

```
iptables -t filter -L          (to see the filter table)
iptables -t nat    -L          (to see the nat table)
iptables -t mangle -L          (to see the mangle table)
iptables -t raw    -L          (to see the raw table)
```

Note that these are the only four tables recognized by the kernel. (Unlike user-defined chains in the tables, there are **no** user-defined tables.)

- For the **filter** table shown on the previous slide, note the policy shown for each built-in chain right next to the name of the chain. As mentioned earlier, only built-in chains have policies. Policy is what is applied to a packet if it is not trapped by any of the rules in a chain.

18.6:   Structure of the `filter` Table

- To explain the structure of the **filter** table, let's first create a new **filter** table for your firewall. I am assuming that this is the first time you are playing with the `iptables` command on your Ubuntu laptop.

- Go ahead and create the following shell script anywhere in your personal directory. The name of the script file is **myfirewall.sh**. *At this point, do not worry about the exact syntax I have used for the iptables commands — the syntax will become clear later in the lecture.*

```
#!/bin/sh

# myfirewall.sh

# A minimalist sort of a firewall for your laptop:

# Create a new user-defined chain for the filter table:
# Make sure you first flush the previous rules by
# 'iptables -t filter F' and delete the previous chains
# by 'iptables -t filter -X'
iptables -t filter -N myfirewall.rules

# Accept all packets generated locally:
iptables -A myfirewall.rules -p all -i lo -j ACCEPT

# Accept all ICMP packets regardless of source:
iptables -A myfirewall.rules -p icmp --icmp-type any -j ACCEPT

# You must not block packets that correspond to TCP/IP protocol
# numbers 50 (ESP) and 51 (AH) for VPN to work. (See Lecture 20
```

```
# for ESP and AH.). VPN also needs the UDP ports 500 (for IKE),
# UDP port 10000 (for IPSec encapsulated in UDP) and TCP
# port 443 (for IPSec encapsulated in TCP). [Note that if you
# are behind a NAT device, make sure it does not change the source
# port on the IKE (Internet Key Exchange) packets.  If the
# NAT device is a Linksys router, just enable "IPSec Passthrough":
iptables -A myfirewall.rules -p 50 -j ACCEPT
iptables -A myfirewall.rules -p 51 -j ACCEPT
iptables -A myfirewall.rules -p udp --dport 500 -j ACCEPT
iptables -A myfirewall.rules -p udp --dport 10000 -j ACCEPT

# The destination port 443 is needed both by VPN and by HTTPS:
iptables -A myfirewall.rules -p tcp --dport 443 -j ACCEPT

# For multicast DNS (mDNS) --- allows a network device to choose a
# domain name in the .local namespace and announce it using multicast.
# Used by many Apple products.  mDNS works differently from the
# unicast DNS we discussed in Lecture 17.  In mDNS, each host stores
# its own information (for example its own IP address).  If your
# machine wants to get the IP address of such a host, it sends out
# a multicast query to the multicast address 224.0.0.251.
iptables -A myfirewall.rules -p udp --dport 5353 -d 224.0.0.251 -j ACCEPT

# for the Internet Printing Protocol (IPP):
iptables -A myfirewall.rules -p udp -m udp --dport 631 -j ACCEPT

# Accept all packets that are in the states ESTABLISHED and
# RELATED (See Section 18.11 for packet states):
iptables -A myfirewall.rules -p all -m state --state ESTABLISHED,RELATED -j ACCEPT

# I will run SSH server on my laptop.  Accept incoming connection requets:
iptables -A myfirewall.rules -p tcp --destination-port 22 -j ACCEPT

# Drop all other incoming packets.  Do not send back any ICMP messages
# for the dropped packets:
iptables -A myfirewall.rules -p all -j REJECT --reject-with icmp-host-prohibited

iptables -I INPUT -j myfirewall.rules
iptables -I FORWARD -j myfirewall.rules
```

- Now, make the shell script executable by

```
chmod +x myfirewall.sh
```

and execute the file as root.

- To see the rule structure created by the above shell script, execute the following command

    `iptables -L -n -v --line-numbers`

  where the option '-n' will display all IP address in the decimal-dot notation and the option '–line-numbers' displays a line number at the beginning of each line in a rule chain. The option '-v' is for the verbose mode. This command will generate the following display for the **filter** table in your terminal window:

```
Chain INPUT (policy ACCEPT 53204 packets, 9375K bytes)
num  pkts bytes  target           prot opt  in  out source     destination
1    568  74832  myfirewall.rules 0    --   *   *   0.0.0.0/0 0.0.0.0/0


Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts bytes  target           prot opt in  out  source     destination
1    0    0      myfirewall.rules 0    --  *   *    0.0.0.0/0 0.0.0.0/0


Chain OUTPUT (policy ACCEPT 76567 packets, 9440K bytes)
num  pkts bytes  target       prot  opt in  out  source     destination


Chain myfirewall.rules (2 references)
num pkts  bytes  target     prot  opt in  out  source      destination
1   327   34807  ACCEPT     0     --  lo  *    0.0.0.0/0   0.0.0.0/0
2   0     0      ACCEPT     icmp  --  *   *    0.0.0.0/0   0.0.0.0/0      icmp type 255
3   0     0      ACCEPT     esp   --  *   *    0.0.0.0/0   0.0.0.0/0
4   0     0      ACCEPT     ah    --  *   *    0.0.0.0/0   0.0.0.0/0
5   0     0      ACCEPT     udp   --  *   *    0.0.0.0/0   0.0.0.0/0      udp dpt:500
6   0     0      ACCEPT     udp   --  *   *    0.0.0.0/0   0.0.0.0/0      udp dpt:10000
7   0     0      ACCEPT     tcp   --  *   *    0.0.0.0/0   0.0.0.0/0      tcp dpt:443
8   6     426    ACCEPT     udp   --  *   *    0.0.0.0/0   224.0.0.251    udp dpt:5353
9   0     0      ACCEPT     udp   --  *   *    0.0.0.0/0   0.0.0.0/0      udp dpt:631
10  228   38248  ACCEPT     0     --  *   *    0.0.0.0/0   0.0.0.0/0      state RELATED,ESTABLISHED
```

```
11    1    48   ACCEPT    tcp   -- *  *   0.0.0.0/0   0.0.0.0/0   tcp dpt:22
12    6 1303 REJECT        0     -- *  *   0.0.0.0/0   0.0.0.0/0   reject-with icmp-host-prohibi
```

- In the output shown above, note that the last column, with no heading, contains ancillary information related to a rule. It may mention a port (as in `tcp dpt:443`, where `dpt` stands for "destination port"), the state of a packet, etc.

- Here are the meanings to be associated with the various column headers shown in the display produced by executing the command '`iptables -L -n -v --line-numbers`':

num : The rule number in a chain.

pkts : The packet count processed by a rule so far.

bytes : The byte count processed by a rule so far.

target :
  The *action* part of a rule. The target can be one of the following: **ACCEPT**, **DROP**, **REJECT**, **REDIRECT**, **RETURN**, or **the name of the chain to jump to**. **DROP** means to drop the packet without sending an error message to the originator of that packet. **REJECT** has the same effect as **DROP**, except that the sender is sent an error message that depends on the argument supplied to this target. **REDIRECT** means to send the packet to a new destination (used with **NAT**). **Return** means to return from this chain to the calling chain and to continue examining rules in the calling chain where you left off. When **RETURN** is encountered in a user-defined chain, the policy associated with the chain is executed.

proto :

The protocol associated with the packet to be trapped by this rule. The protocol may be either named symbolically or specified by a number. Each standard protocol has a number associated with it. The protocol numbers are assigned by Internet Assigned Numbers Authority (IANA).

opt : optional

in : The input interface to which the rule applies.

out : The output interface to which the rule applies.

source : The source address(es) to which the rule applies.

destination : The destination address(es) to which the rule applies.

Note that when the fifth column mentions a user-defined service, then the last column (without a title) must mention the port specifically. On the other hand, for packets corresponding to standard services, the system can figure out the ports from the entries in the file `/etc/services`.

- In the display produced by executing the command '`iptables -L -n -v --line-numbers`', note the three rule chains in the `filter` table: **INPUT**, **FORWARD**, and **OUTPUT**. Most importantly, note how the **INPUT** chain jumps to the user-defined **myfirewall.rules** chain. The built-in **FORWARD** chain also jumps to the same user-defined chain.

- Note the **policy** declaration associated with each chain. It is

ACCEPT. As mentioned previously, the policy sets the fate of a packet it it is not trapped by any of the rules in a chain.

- Since both the built-in **INPUT** and the built-in **FORWARD** chains jump to the user-defined **myfirewall.rules** chain, let's look at the first rule in this user-defined chain in some detail. This rule is:

```
num pkts  bytes   target   prot  opt  in   out  source      destination
1   327   34807   ACCEPT   0     --   lo   *    0.0.0.0/0   0.0.0.0/0
```

The `IP/port` address `0.0.0.0/0` means all addresses. [The forward slash in the **source** and the **destination** IP addresses is explained on Section 18.10.] Since the input interface mentioned is **lo** and since no ports are mentioned (that is, there is no entry in the unlabeled column at the very end), this means that this rule applies only to the packets generated by the applications running on the local system. (That is, this rule allows the loopback driver to work.) Therefore, with this rule, you can request any service from your local system without the packets being denied.

- Let's now examine the rule in line 2 for the user-defined chain **myfirewall.rules** shown in the display produced by the command 'iptables -L -n -v --line-numbers' command:

```
num pkts  bytes   target   prot  opt  in   out  source      destination
2    0     0      ACCEPT   icmp  --   *    *    0.0.0.0/0   0.0.0.0/0   icmp type 255
```

As mentioned in Lecture16, ICMP messages are used for error reporting between host to host, or host to gateway in the internet. (Between gateway to gateway, a protocol called Gateway to Gateway protocol (GGP) should normally be used for error reporting.) [The three types of commonly used ICMP headers are `type 0`, `type 8`, and `type 11`. ICMP echo requests coming to your machine when it is pinged by some other host elsewhere in a network are of type 8. If your machine responds to such a request, it echos back with an ICMP packet of type 0. Therefore, when a host receives a type 8 ICMP message, it replies with a type 0 ICMP message. Type 11 service relates to packets whose 'time to live' (TTL) was exceeded in transit and for which you as sender is accepting a 'Time Exceeded' message that is being returned to you. You need to accept type 11 ICMP protocol messages if you want to use the 'traceroute' command to find broken routes to hosts you want to reach. ICMP type 255 is unassigned by IANA (Internet Assigned Numbers Authority); it is used internally by **iptables** to mean all ICMP types. **See Section 18.10 for additional ICMP types.**]

- With regard to the other rules in the **myfilter.rules** chain, their purpose should be clear from the comments in the **myfilter.sh** shell script.

- Let's now examine the **OUTPUT** chain in the **filter** table. [(See the output shown earlier in this section that was produced by the command '`iptables -L -n -v --line-numbers`' command.) There are no rules in this chain. Therefore, for all outbound packets, the policy asso-

ciated with the **OUTPUT** chain will be used. This policy says
`ACCEPT`, implying that all outbound packets will be sent directly,
without further examination, to their intended destinations.

- About the **FORWARD** chain, note that packet forwarding only
  occurs when the machine is configured as a router. (For IP
  packet forwarding to work, you also have to change the value of
  `net.ipv4.ip_forward` to 1 in the `/etc/sysctl.conf file`.)

18.7: Structure of the `nat` Table

- Let's now examine the output produced by the command line

  ```
  iptables -t nat -n -L
  ```

  we get

  ```
      Chain PREROUTING (policy ACCEPT)
      target     prot opt source              destination


      Chain POSTROUTING (policy ACCEPT)
      target     prot opt source              destination


      Chain OUTPUT (policy ACCEPT)
      target     prot opt source              destination
  ```

- The **nat** table is used only for translating either the packet's source address field or its destination address field.

- NAT (which stands for Network Address Translation) allows a host or several hosts to share the same IP address. For example, let's say we have a local network consisting of 5-10 clients. We set their default gateways to point through the NAT server.

The NAT server receives the packet, rewrites the source and/or destination address and then recalculates the checksum of the packet.

- Only the first packet in a stream of packets hits this table. After that, the rest of the packets in the stream will have this network address translation carried out on them automatically.

- The 'targets' for the nat table (meaning, the actions that are permitted for the rules) are

  ```
  DNAT
  SNAT
  MASQUERADE
  REDIRECT
  ```

- The **DNAT** target is mainly used in cases where you have a **single** public IP for a local network in which different machines are being used for different servers. When a remote client wants to make a connection with a local server using the publicly available IP address, you'd want your firewall to rewrite the destination IP address on those packets to the local address of the machine where the server actually resides.

- **SNAT** is mainly used for changing the source address of packets. Using the same example as above, when a server residing on one of the local machines responds back to the client, initially the packets emanating from the server will bear the source address of the local machine that houses the server. But as these packets pass through the firewall, you'd want to change the source IP address in these packets to the single public IP address for the local network.

- The **MASQUERADE** target is used in exactly the same way as **SNAT**, but the **MASQUERADE** target takes a little bit more overhead to compute. Whereas **SNAT** will substitute a single previously specified IP address for the source address in the outgoing packets, **MASQUERADE** can substitute a DHCP IP address (that may vary from connection to connection).

- Note that in the output of `iptables -t nat -n -L` shown at the beginning of this section, we did not have any targets in the `nat` table. That is because my laptop is not configured to serve as a router.

18.8:   Structure of the `mangle` Table

- The mangle table is used for specialized packet alteration, such as for changing the TOS (Type of Service) field, the TTL (Time to Live) field, etc., in a packet header.

On my Linux laptop, the command

```
iptables -t mangle -n -L
```

returns

```
Chain PREROUTING (policy ACCEPT)
target     prot  opt  source  destination

Chain INPUT (policy ACCEPT)
target     prot  opt  source  destination

Chain FORWARD (policy ACCEPT)
target     prot  opt  source  destination

Chain OUTPUT (policy ACCEPT)
target     prot  opt  source  destination

Chain POSTROUTING (policy ACCEPT)
target     prot  opt  source  destination
```

- Earlier, toward the end of Section 18.2, I showed an example of a rule for the **PREROUTING** chain of the **mangle** table that

used the **DROP** target. The rules in the **PREROUTING** chain are applied before the operating system applies a routing decision to a packet.

- The following targets can only be used in the **mangle** table.

  1. **TOS** — Used to change the TOS (Type of Service) field in a packet. (This is the second byte in the IP header) Not understood by all the routers.

  2. **TTL** — The TTL target is used to change the TTL (Time To Live) field of the packet.

  3. **MARK** — This target is used to give a special mark value to the packet. Such marks are recognized by the iproute2 program for routing decisions.

  4. **SECMARK** — This target sets up a security-related mark in the packet. Such marks can be used by SELinux fine-grained security processing of the packets.

  5. **CONNSECMARK** — This target places a connection-level mark on a packet for security processing.

18.9:   Structure of the `raw` Table

- If you execute the following command

  ```
  iptables -t raw -L
  ```

  you will see the following output:

  ```
  Chain PREROUTING (policy ACCEPT)
  target     prot   opt    source      destination


  Chain OUTPUT (policy ACCEPT)
  target     prot   opt    source      destination
  ```

  This output shows that the **raw** table supports only two chains:
  PREROUTING and **OUTPUT**.

- As mentioned earlier, the **raw** table is used for specifying the exemptions from connection tracking that we will talk about later.
  When rules are specified for the `raw` table, the table takes priority over the other tables.

18.10:   How the Tables are Actually Created

- The iptables are created by the **iptables** command that is run as root with different options. To see all the option, say

      iptables -h

- Here are some other optional flags for the iptables command and a brief statement of what is achieved by each flag:

```
    iptables -N chainName           Create a new user-defined chain

    iptables -X chainName           Delete a user-defined chain; must have
                                    been previoiusly emptied of rules by
                                    either the '-D' flag or the '-F' flag.

    iptables -P chainName ....      Change the policy for a built-in chain

    iptables -L chainName           List the rules in a chain.  If no
                                    chain specified, it lists rules in all
                                    the chains in the filter table.  Without
                                    the '-t' flag, the filter table is the
                                    default.

    iptables -F chainName           Flush the rules out of a chain
                                    When no chain-name is supplied as the
                                    argument to '-F', all chains are flushed.

    iptables -Z chainName           Zero the packet and byte counters
                                    on all rules in the chain

    iptables -A chainName ....      Append a new rule to the chain
```

```
iptables -I chainName pos ....    Insert a new rule at position 'pos'
                                  in the specified chain)
                                  See an example of rule insertion in
                                  my 'experiment' file

iptables -R chainName ....        Replace a rule at some position in
                                  the specified chain

iptables -D chainName ....        Delete a rule at some position in
                                  the specified chain, or the first that
                                  matches
                                  See my 'experiment' file for the
                                  two different ways in which the '-D'
                                  flag can be used

iptables -P chainName target      Specify a target policy for the chain.
                                  This can only be done for built-in
                                  chains.
```

- After the first level flags shown above that name a chain, if this flag calls for a new rule to be specified (such as for '-A' flag) you can have additional flags that specify the state of the packet that must be true for the rule to apply and specify the action part of the rule. We say that these additional flags describe the **filtering specifications** for each rule.

- Here are the rule specification flags:

```
-p  args
        for specifying the protocol (tcp, udp,
        icmp, etc)  You can also specify a protocol
        by number if you know the numeric protocol
        values for IP.
```

```
-s  args
        for specifying source address(es)


--sport args
        for specifying source port(s)


-d  args
        for specifying destination address(es)


--dport args
        for specifying destination port(s)
        (For the port specifications, you can supply
        a port argument by name, as by 'www', as
        listed in /etc/services.)


--icmp-type typemane
        (  for spcifying the type of ICMP packet.  The
         icmp type names can be found by the comamnd

            iptables -p icmp --help

         it returns the following for the icmp types

         Valid ICMP Types:
                any
                echo-reply (pong)       (type 0)
                destination-unreachable (type 3)
                    network-unreachable     (code 0)
                    host-unreachable        (code 1)
                    protocol-unreachable    (code 2)
                    port-unreachable        (code 3)
                    fragmentation-needed    (code 4)
                    source-route-failed     (code 5)
                    network-unknown         (code 6)
                    host-unknown            (code 7)
                    network-prohibited      (code 8)
                    host-prohibited         (code 9)
                    TOS-network-unreachable (code 10)
                    TOS-host-unreachable    (code 11)
                    communication-prohibited (code 12)
                    host-precedence-violation
                    precedence-cutoff
                source-quench               (type 4)
```

```
                    redirect              (type 5)
                       network-redirect
                       host-redirect
                       TOS-network-redirect
                       TOS-host-redirect
                    echo-request (ping)    (type 8)
                    router-advertisement   (type 9)
                    router-solicitation    (type 10)
                    time-exceeded (ttl-exceeded)(type 11)
                       ttl-zero-during-transit    (code 0)
                       ttl-zero-during-reassembly (code 1)
                    parameter-problem      (type 12)
                       ip-header-bad
                       required-option-missing
                    timestamp-request      (type 13)
                    timestamp-reply        (type 14)
                    address-mask-request   (type 17)
                    address-mask-reply     (type 18)  )
```

-j args
        the name of the target to execute when
        the rule matches; 'j' stands for 'jump to'

-i args
        for naming the input interface (when an
        interface is not named, that means all
        interfaces)

-o args
        for specifying an output interface

        (Note that an interface is the physical
         device a packet came in on or is going
         out on. You can use the ifconfig command
         to see which interfaces are up.)

        (Also note that only the packets traversing the
         FORWARD chain have both input and output
         interfaces.)

        (It is legal to specify an interface that
         currently does not exist. Obvously, the
         rule would not match until the interface
         comes up.)

(When the argument for interface is followed
 by '+', as in 'eth+', that means all
 interfaces whose names begin with the
 string 'eth'.)

(So an interface specified as
      -i ! eth+
 means none of the ethernet interfaces.)

-f   (For specifying that a packet is
             a second or a further fragment. As mentioned
             in Lecture 16 notes, sometimes, in order
             to meet the en-route or destination
             hardware constraints, a packet may have
             to be fragmented and sent as multiple
             packets. This can create a problem for
             packet-level filtering since only the first
             fragment packet may carry all of the headers,
             meaning the IP header and the enveloped
             higher-level protocol header such as the TCP,
             or UDP, etc., header. The subsequent fragments
             may only carry the IP header and not mention
             the higher level protocol headers. Obviously,
             such packets cannot be processed by rules that
             mention higher level protocols. Thus a rule
             that carries the specification
                  '-p TCP --sport www'
             will never match a fragment (other than the
             first fragment). Neither will the opposite
             rule
                  '-p TCP --sport ! www'

             However, you can specify a rule specifically
             for the second and further fragments, using
             the '-f' flag. It is also legal to specify
             that a rule does not apply to second and
             further fragments, by preceding the '-f' with
             ' ! '.

             Usually it is regarded as safe to let second
             and further fragments through, since filtering
             will effect the first fragment, and thus
             prevent reassembly on the target host;

```
                        however, bugs have been known to allow
                        crashing of machines simply by sending
                        fragments.)

                        (Note that the '-f' flag does not take any
                         arguments.)

    --syn               (To indicate that this rule is meant for
                         a SYN packet. It is sometimes useful to allow
                         TCP connections in one direction, but not
                         in the other. As explained in Lecture 16,
                         SYN packets are for requesting new connetions.
                         These are packets with the SYN flag set, and
                         the RST and ACK flags cleared.  By
                         disallowing only the SYN packets, we can
                         stop attempted connections in their
                         tracks. The '-syn' flag is only valid for
                         rules which specify TCP as their protocol.
                         For example, to specify TCP connection
                         attempts from 192.168.1.1:

                             -p TCP -s 192.168.1.1 --syn

                         This flag can be inverted by preceding it
                         with a '!', which means every packet other
                         than the connection initiation.)

    -m match            (This is referred to as a rule seeking an
                         'extended match'. This may load extensions
                         to iptables.)

    -n                  (This forces the output produced by the
                         '-L' flag to show numeric values for
                         the IP addresses and ports.)
```

- Many rule specification flags (such as '-p', '-s', '-d', '-f' '–syn', etc.)
  can have their arguments preceded by '!' (that is pronounced
  'not') to match values not equal to the ones given. This is referred

to as **specification by inversion**. For example, to indicate all sources addresses but a specific address, you would have

```
-s ! ip_address
```

- For the '-f' option flags, the inversion is done by placing '!' before the flag, as in

```
! -f
```

The rule containing the above can only be matched with the first fragment of a fragmented packet.

- Also note that the '–syn' second-level option is a shorthand for

```
--tcp-flags SYN,RST,ACK SYN
```

where `--tcp-flags` is an example of a TCP extension flag. Note that '-d', and '-s' are also TCP extension flags. These flags work only when the argument for the protocol flag '-p' is 'tcp'.

- The source ('-s', '–source' or '–src') and destination ('-d', '–destination' or '–dst') IP addresses can be specified in four ways:

  1. The most common way is to use the full name, such as `localhost` or `www.linuxhq.com`.

  2. The second way is to specify the IP address such as 127.0.0.1.

  3. The third way allows specification of a group of IP addresses with the notation 199.95.207.0/24 where the number after the forward slash indicates the number of leftmost bits in the 32 bit address that must remain fixed. Therefore, 199.95.207.0/24 means all IP addresses between 199.95.207.0 and 199.95.207.255.

  4. The fourth way uses the net mask directly to specify a group of IP addresses. What was accomplished by 199.95.207.0/24 above is now accomplished by 199.95.207.0/255.255.255.0.

- If nothing comes after the forward slash in the prefix notation for an IP address range, the default of /32 (which is the same as writing down the net mask as /255.255.255.255) is assumed. Both of these imply that all 32 bits must match, implying that only one IP address can be matched. Obviously, the opposite of the default /32 is /0. This means all 32 address bits can be

anything. Therefore, /0 means every IP address. The same is meant by the specifying the IP address range as 0/0 as in

```
iptables -A INPUT -s 0/0 -j DROP
```

which will cause all incoming packets to be dropped. But note that `-s 0/0` is redundant here because not specifying the '-s' flag is the same as specifying '-s 0/0' since the former means all possible IP addresses.

18.11: Connection Tracking by `iptables` and the Extension Modules

- A modern iptables-based firewall understands the notion of a stream. This is done by what is referred to as connection tracking.

- Connection tracking is based on the notion of '**the state of a packet**'.

- If a packet is the first that the firewall sees or knows about, it is considered to be in state `NEW` [as would be the case for, say, a SYN packet in a TCP connection (see Lecture 16)], or if it is part of an already established connection or stream that the firewall knows about, it is considered to be in state `ESTABLISHED`.

- States are known through the connection tracking system, **which keeps track of all the sessions.**

- *It is because of the connection-tracking made possible by the rule in line 10 of the* **myfirewall.rules** *chain in the display produced by executing* `iptables -L -n -v --line-numbers` *in*

Section 18.6 that when I make a connection with a remote host such as `www.nyt.com` that I am able to receive all the incoming packets. That rule tells the kernel that the incoming packets are of state **ESTABLISHED**, *meaning that they belong to a connection that was established and accepted previously.*

- Connection tracking is also used by the **nat** table and by its **MASQUERADE** target in the tables.

- Let's now talk about extension modules since it is one of those extensions to **iptables** that makes it possible to carry out connection tracking.

- When invoking `iptables`, an extension module can be loaded into the kernel for additional match options for the rules. **An extension module is specified by the '-m' option** as in the following rule we used in the shell executable file Section 18.6 of this lecture:

```
iptables -A myfirewall.rules -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- As the above rule should indicate, a most useful extension module is **state**. This extension tries to interpret the connection-tracking analysis produced by the `ip_conntrack` module.

- As to how exactly the interpretation of the results on a packet produced by the `ip_conntrack` module should be carried out is specified by the additional '`--state`' option supplied to the '`state`' extension module. See the rule example shown above that uses both the '`-m state`' option and the '`--state`' suboption.

- The '`--state`' suboption supplies a comma-separated list of states of the packet that must be found to be true for the rule to apply, and as before, the '!' flag indicates not to match those states. These states that can be supplied as arguments to the '`--state`' option are:

```
NEW                 A packet which creates a new connection.

ESTABLISHED         A packet which belongs to an existing
                    connection (i.e., a reply packet, or
                    outgoing packet on a connection which
                    has seen replies).

RELATED             A packet which is related to, but not
                    part of, an existing connection, such as
                    an ICMP error, or (with the FTP module
                    inserted), a packet establishing an ftp
```

```
                            data connection.

    INVALID                 A packet which could not be identified
                            for some reason: this includes running
                            out of memory and ICMP errors which
                            don't correspond to any known connection.
                            Generally these packets should be dropped.
```

- Another example of a rule that uses the 'state' extension for stating the rule matching conditions:

```
iptables -A FORWARD -i ppp0 -m state ! --state NEW -j DROP
```

This says to append to the FORWARD chain a rule that applies to all packets being forwarded through the ppp0 interface. If such a packet is NOT requesting a new connection, it should be dropped.

- Another extension module is the 'mac' module that can be used for matching an incoming packet's source Ethernet (MAC) address. This only works for for packets traversing the PREROUT-ING and INPUT chains. It provides only one option '--mac-source' as in

```
iptables  -A INPUT  -m mac --mac-source 00:60:08:91:CC:B7  ACCEPT
```

or as in

```
iptables  -A INPUT  -m mac --mac-source ! 00:60:08:91:CC:B7  DROP
```

The second rule will drop all incoming packets unless they are from the specific machine with the MAC address shown.

- Yet another useful extension module is the 'limit' module that is useful in warding off Denial of Service (DoS) attacks. This module is loaded into the kernel with the '`-m limit`' option. What the module does can be controlled by the subsequent option flags '`--limit`' and '`--limit-burst`'. The following rule will limit a request for a new connection to one a second. Therefore, if DoS attack consists of bombarding your machine with SYN packets, this will get rid of most of them. This is referred to as "**SYN-flood protection**".

```
iptables -A FORWARD -p tcp --syn -m limit --limit 1/s -j ACCEPT
```

- The next rule is a protection against indiscriminate and nonstop scanning of the ports on your machine. This is referred to as protection against a "furtive port scanner":

```
iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST \
                        -m limit --limit 1/s -j ACCEPT
```

- The next rule is a protection against what is called as the **"ping of death"** where someone tries to ping your machine in a non-stop fashion:

```
iptables -A FORWARD -p icmp --icmp-type echo-request \
                    -m limit --limit 1/s -j ACCEPT
```

18.12: Using `iptables` to do Port Forwarding

- Let's say that you have a firewall computer protecting a LAN. Let's also say that you are providing a web server on one of the LAN computers that is physically different from the firewall computer. Further, let's assume that there is a single IP address available for the whole LAN, this address being assigned to the firewall computer. Let's assume that this IP address is 123.45.67.89.

- So when a HTTP request comes in from the internet, it will typically be received on port 80 that is assigned to HTTP in `/etc/services`. The firewall would need to forward this request to the LAN machine that is actually hosting the web server.

- This is done by adding a rule to the PREROUTING chain of the NAT table:

```
iptables -t nat  -A PREROUTING  -p tcp  -d 123.45.67.89 \
             -dport 80  -j DNAT  --to-destination 10.0.0.25
```

  where the jump target DNAT stands for **Dynamic Network Address Translation**. We are also assuming that the LAN address of the machine hosting the HTTP server is 10.0.0.25 in a Class A private network 10.0.0.0/8.

- If multiple LAN machines are simultaneously hosting the same HTTP server for reasons of high traffic to the server, you can spread the load of the service by providing a range of addresses for the '–to-destination' option, as by

  ```
  --to-destination 10.0.0.1-10.0.0.25
  ```

- This will now spread the load of the service over 25 machines, including the gateway machine if its LAN address is 10.0.0.1.

- So the basic idea in port forwarding is that you forward all the traffic received at a given port on our firewall computer to the designated machines in the LAN that is protected by the firewall.

18.13: Using Logging with `iptables`

- So far we have only talked about the following targets: AC-CEPT, DENY, DROP, REJECT, REDIRECT, RETURN, and chain_name_to_jump_to for the `filter` table, and SNAT and DNAT for the `nat` table.

- One can also use LOG as a target. So if you did not want to drop a packet for some reason, you could go ahead and accept it but at the same time log it to decide later if your current rule for such packets is a good rule. Here is an example of a LOG target in a rule for the FORWARD chain:

  ```
  iptables  -A FORWARD  -p tcp  -j LOG  --log-level info
  ```

- Here are all the possibilities for the '–log-level' argument:

  ```
  emerg
  alert
  crit
  err
  warning
  notice
  info
  ```

```
debug
```

- You can also supply a '–log-prefix' option to add further information to the front of all messages produced by the logging action:

```
iptables  -A FORWARD  -p tcp  -j LOG  --log-level info  \
                             --log-prefix "Forward INFO "
```

18.14:   Saving and Restoring Your Firewall

- As I showed in Section 18.6, you can write a shell script with the `iptables` commands in it for creating the different rules for your firewall. You can load in the firewall rules simply by executing the shell script. *If this is the approach you use, make sure you invoke `iptables -F` and `iptables -X` for each of the tables* **before** *executing the script.*

- A better way to save your firewall rules is by invoking the `iptables-save` command:

  ```
  iptables-save  >  MyFirewall.bk
  ```

  Subsequently, when you reboot the machine, you can restore the firewall by using the command `iptables-restore` as root:

  ```
  iptables-restore < MyFirewall.bk
  ```

- When you save a firewall with the `iptables-save` command, the text file that is generated is visually different from the output produced by the `iptables -L` command. If I use `iptables-save` to save the firewall I created with the shell script in Section 18.6, here is what is placed in the **MyFirewall.bk** file:

```
# Generated by iptables-save v1.3.6 on Fri Apr  4 18:23:31 2008
*raw
:PREROUTING ACCEPT [96159:19721033]
:OUTPUT ACCEPT [91367:10876335]
COMMIT
# Completed on Fri Apr  4 18:23:31 2008
# Generated by iptables-save v1.3.6 on Fri Apr  4 18:23:31 2008
*mangle
:PREROUTING ACCEPT [173308:40992127]
:INPUT ACCEPT [173282:40986202]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [160626:19665486]
:POSTROUTING ACCEPT [160845:19695621]
COMMIT
# Completed on Fri Apr  4 18:23:31 2008
# Generated by iptables-save v1.3.6 on Fri Apr  4 18:23:31 2008
*nat
:PREROUTING ACCEPT [6231:908393]
:POSTROUTING ACCEPT [10970:640894]
:OUTPUT ACCEPT [10970:640894]
COMMIT
# Completed on Fri Apr  4 18:23:31 2008
# Generated by iptables-save v1.3.6 on Fri Apr  4 18:23:31 2008
*filter
:INPUT ACCEPT [53204:9375108]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [112422:12698834]
:myfirewall.rules - [0:0]
-A INPUT -j myfirewall.rules
-A FORWARD -j myfirewall.rules
-A myfirewall.rules -i lo -j ACCEPT
-A myfirewall.rules -p icmp -m icmp --icmp-type any -j ACCEPT
-A myfirewall.rules -p esp -j ACCEPT
-A myfirewall.rules -p ah -j ACCEPT
-A myfirewall.rules -p udp -m udp --dport 500 -j ACCEPT
-A myfirewall.rules -p udp -m udp --dport 10000 -j ACCEPT
-A myfirewall.rules -p tcp -m tcp --dport 443 -j ACCEPT
-A myfirewall.rules -d 224.0.0.251 -p udp -m udp --dport 5353 -j ACCEPT
-A myfirewall.rules -p udp -m udp --dport 631 -j ACCEPT
-A myfirewall.rules -m state --state RELATED,ESTABLISHED -j ACCEPT
-A myfirewall.rules -p tcp -m tcp --dport 22 -j ACCEPT
-A myfirewall.rules -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Fri Apr  4 18:23:31 2008
```

This format is obviously still readable and still directly editable. On Red Hat machines, what is produced by `iptables-save` is

directly accessible from the file `/etc/sysconfig/iptables`. So to restore a previously created firewall on a Red Hat machine, all you have to do is to invoke `iptables-restore` and direct into it the contents of `/etc/sysconfig/iptables`.

- Note that when a system is rebooted, the firewall rules are automatically flushed and reset — in most cases to empty tables (implying really no firewall protection).

- For Ubuntu Linux, if you want the system to automatically save the latest firewall on shutdown and then also automatically restore the firewall at startup, you would need to edit your

    `/etc/network/interfaces`

    network configuration file and enter in it the appropriate **pre-up** and **post-down** commands for each of the interfaces that are meant to be protected by the firewall.

- If, say, **eth0**, is the ethernet interface on your Ubuntu laptop, you'd need to enter the following `pre-up` and `post-down` lines in the `/etc/network/interfaces` file so that its **eth0** entry looks like:

```
auto eth0
iface eth0 inet dhcp
pre-up iptables-restore < /etc/iptables.rules
post-down iptables-save > /etc/iptables.rules
```

The `iptables.rules` file, initially created manually with the **iptables-save** command, must already exist in the **/etc/** folder before the automatic save and reload procedure can work. [The file `/etc/network/interfaces` contains the network interface configuration information for the Ubuntu distribution of Linux. Do 'man interfaces' to see how to configure this file for static and DHCP-provided IP addresses. In this file, lines beginning with `auto` are used to identify the physical interfaces to be brought up at system startup. With respect to each interface, a line beginning with `pre-up` specifies the command that must be executed before the interface is brought up. By the same token, a line beginning with `post-down` specifies the command that must be executed after the interface is taken down.]

- Note that on Red Hat Linux and its variants, you can start and stop `iptables` by

```
/etc/init.d/iptables start
/etc/init.d/iptables stop
/etc/init.d/iptables restart
```

Also on Red Hat Linux, if you are doing NAT, make sure you turn on IP packet forwarding by setting

```
net.ipv4.ip_forward = 1
```

in the `/etc/sysctl.conf` file.

- Some other points to remember:

  - Note that the names of built-in chains, INPUT, OUTPUT, and FOR-WARD, must always be in uppercase.

  - The '`-p tcp`' and '`-p udp`' options load into the kernel the TCP and UDP extension modules.

  - Chain names for user-defined chains can only be up to 31 characters.

  - User-defined chain names are by convention in lower-case.

  - When a packet matches a rule whose target is a user-defined chain, the packet begins traversing the rules in that user-defined chain. If that chain doesn't decide the fate of the packet, then once traversal on that chain has finished, traversal resumes on the next rule in the current chain.

  - Even if the condition part of a rule is matched, if the rule does not specify a target, the next rule will be considered.

  - User-defined chains can jump to other user-defined chains (but don't make loops: your packets will be dropped if they're found to be in a loop).

18.15:   Case Study: Designing Iptables for a New LAN

Let's say that you want to create a firewall to protect a Class C 192.168.1.0/24 private LAN that is connected to the rest of the internet by a router and a gateway machine as shown.

```
local_machine_1   local_machine_2   .....   local_machine_N

        |                 |                   |     LAN addresses:
        \                 |                   /       192.168.1.0/24
         \                |                  /
          \               |                 /
          ---------------------------------
          |             ROUTER             |
          ---------------------------------
                         |
                         |
                         |
                         |   interface eth1, IP adress: 192.168.1.1
            ----------------------
            |                    |
            |     Gateway        |  loopback: localhost
            |     Machine        |  addess:   127.0.0.1
            |  (firewall computer)|  interface: lo
            |                    |
            ----------------------
                     |   interface eth0, IP address: 123.45.67.89
                     |
                     |
                internet
```

We will also assume that the gateway machine has its IP address

assigned dynamically (DHCP) by some ISP. We will assume that the gateway machine is using Linux as its OS and that **iptables** based packet filtering software is installed. We want the firewall installed in the gateway machine to allow for the following:

- It should allow for unrestricted internet access from all the machines in the LAN.

- Allow for SSH access (port 22) to the firewall machine from outside the LAN for external maintenance of this machine.

- Permit `Auth/Ident` (port 113) that is used by some services like SMTP and IRC. (Note that port 113 is for Auth (authentication). Some old servers try to identify a client by connecting back to the client machine on this port and waiting for the IDENTD server on the client machine to report back. But this port is now considered to be a security hole. See http://www.grc.com/port_113.htm. So, for this port, 'ACCEPT' should probably be changed to DROP.)

- Let's say that the LAN is hosting a web server (on behalf of the whole LAN) and that this HTTPD server is running on the machine 192.168.1.100 of the LAN. So the firewall must use NAT to redirect the incoming TCP port 80 requests to 192.168.1.100.

- We also want the firewall to accept the ICMP Echo requests (as used by ping) coming from the outside.

- The firewall must log the filter statistics on the external interface of the firewall machine.

- We want the firewall to respond back with TCP RST or ICMP Unreachable for incoming requests for blocked ports.

- Shown below is Rusty Russell's recommended firewall that has the above mentioned features:

```sh
#! /bin/sh

# macro for external interface:
ext_if = "eth0"
# macro for internal interface:
int_if = "ath0"

tcp_services = "22,113"
icmp_types = "ping"

comp_httpd = "192.168.1.100"

# NAT/Redirect
modprobe ip_nat_ftp
iptables -t nat -A POSTROUTING -o $ext_if -j MASQUERADE
iptables -t nat -i -A PREROUTING $ext_if -p tcp --dport 80 -j DNAT --to-destination $c

# filter table rules
# Forward only from external to webserver:
iptables -A FORWARD -m state --state=ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i $ext_if -p tcp -d $comp_httpd --dport 80 --syn -j ACCEPT
```

```
# From internal is fine, rest rejected
iptables -A FORWARD -i $int_if -j ACCEPT
iptables -A FORWARD -j REJECT

# External can only come in to $tcp_services and $icmp_types
iptables -A INPUT -m state --state=ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i $ext_if -p tcp --dport $tcp_services --syn -j ACCEPT
for icmp in $icmp_types; do
    iptables -A INPUT -p icmp --icmp-type $icmp -j ACCEPT
done

# Internal and loopback are allowed to send anything:
iptables -A INPUT -i $int_if -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -j REJECT

# logging
echo "1" > /proc/sys/net/ipv4/ip_forward
```

HOMEWORK PROBLEMS

1. What is the difference between a packet-filtering firewall and a proxy-server firewall? Can the two be used together?

2. How can you figure out if `iptables` is installed and running in your machine?

3. What are the four tables maintained by the Linux kernel for processing incoming and outgoing packets?

4. How does an `iptables` based firewall decide as to which packets to subject to the `INPUT` chain of rules, which to the `FORWARD` chain of rules, and which to the `OUTPUT` chain of rules? Additionally, which part of a packet is examined in order to figure out whether or not the condition part of a rule is satisfied?

5. As a packet is being processed by a chain of rules, what happens to the packet if it does not satisfy the conditions in any of the rules? What is meant by a chain policy?

6. Show how you would use the `iptables` command to reject all incoming SYN packets that seek to open a new connection with

your machine?

7. What is the option given to the `iptables` command to flush all the user-defined *chains* in a table? How do you flush all the rules in a table?

8. If you see the string '`icmp type 255`' at the end of a line of the output produced by the '`iptables -L`' command, what does that mean?

9. What are the `icmp-types` associated with the echo-request (ping) and with the echo-reply (pong) packets?

10. The `raw` table is used for specifying exemptions to connection tracking. What does that mean?

11. What is the `iptables` command if you want your machine to accept only the incoming connection requests for the SSHD server you are running on your machine? (You want your machine to drop all other connection request packets from remote clients.)

12. What is connection tracking? How does an `iptables`-based firewall know that the incoming packets all belong to the same ongoing connection?

13. What are the different packet states recognized by the connection tracking `iptables` extension module `state`?

14. **Programming Assignment:**

Design a firewall for your Linux machine using the `iptables` packet filtering modules. Your homework consists of writing iptables rules to do the following:

- Place no restriction on outbound packets.
- Allow for SSH access (port22) to your machine from only the `purdue.edu` domain.
- Assuming you are running an HTTPD server on your machine that can make available your entire home directory to the outside world, write a rule that allows only a single IP address in the internet to access your machine for the HTTP service.
- Permit Auth/Ident (port 113) that is used by some services like SMTP and IRC.
- Accept the ICMP Echo requests (as used by ping) coming from the outside.
- Respond back with TCP RST or ICMP unreachable for incoming requests for blocked ports.