Linux Packet Filtering Tutorial

Paul 'Rusty' Russell & Michael Neuling

June 30, 1999

A Questionnaire

- Be honest: these are completely anonymous. We want to know where we should pitch this tutorial.
  - I know what an IP address is, and what a netmask is
    [ ] TRUE
    [ ] FALSE
  - I know that a normal host as 2 interfaces, a routing host has 3 or more, and understand what IP forwarding and routing are
    [ ] TRUE
    [ ] FALSE
  - I have set up a packet filter before
    [ ] TRUE
    [ ] FALSE
  - I have set up a Linux packet filter before
    [ ] TRUE
    [ ] FALSE
  - I know what IP spoofing, teardrop Ping of Death are.
    [ ] TRUE
    [ ] FALSE
  - I have read the ipchains HOWTO
    [ ] TRUE
    [ ] FALSE
  - I am :
    [ ] Linus Torvalds[1]
    [ ] David Miller[2]
    [ ] Alan Cox[3]
    [ ] Alexey Kuznetsov[4]
    [ ] Darren Reed[5]
    [ ] Jos Vos[6]
    [ ] W. Richard Stevens[7]
    [ ] None of the above

[1] *Author of AlphaLinux Quake port*
[2] *Linux demigod*
[3] *Linux demigod*
[4] *Linux 2.2 networking guru*
[5] *Author of BSD's ipfilter*
[6] *Author of ipfwadm (2.0's packet filter code)*
[7] *Author of TCP/IP Illustrated vols I, II and III, Advanced Programming in the Unix Environment, and Unix Network Programming.*

1 Introduction

In this section we will show:

- What a Packet is.
- What Packet Filtering is.
- IPv4 and the commonly used protocols within it (ICMP, UDP, TCP).
- How packet filtering differs from firewalling.
- A Linux box as a Packet Filter.

What is packet filtering?

Packets consist of a header, and a body. Packet filtering means allowing or disallowing packets based on their header. (See Figure 1)

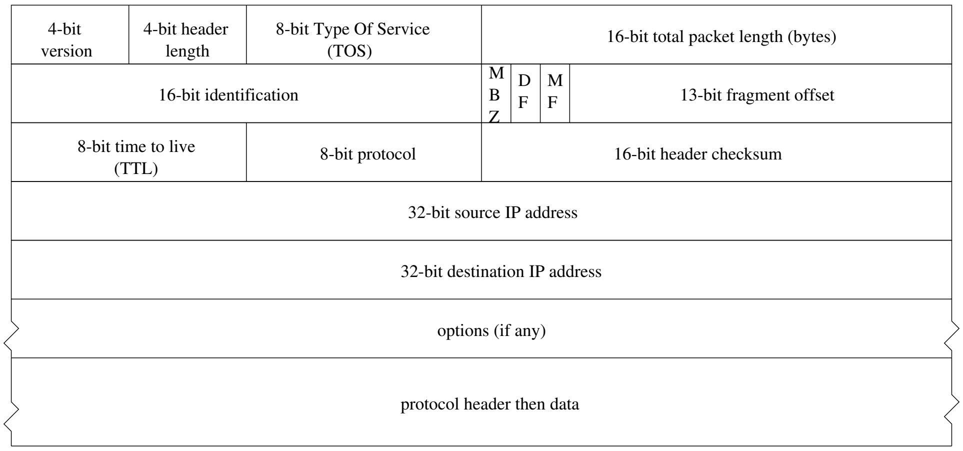


Figure 1: IP Header

- Nobody uses IPv4 “raw”; you use a protocol inside IP. This “inner” protocol similarly has a header, then the actual data.
- The most common protocols are:
  - User Datagram Protocol (UDP): a simple protocol with few features, used for DNS, “productivity apps” (ie. Quake, RealAudio).

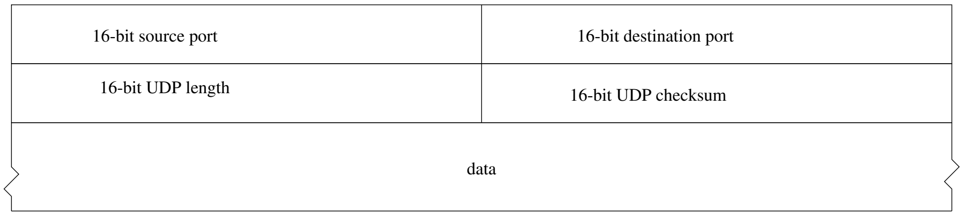


Figure 2: UDP header

  - Transmission Control Protocol (TCP): a more complex protocol with retransmission, acknowledgement, congestion control. Used for WWW, EMail, FTP.

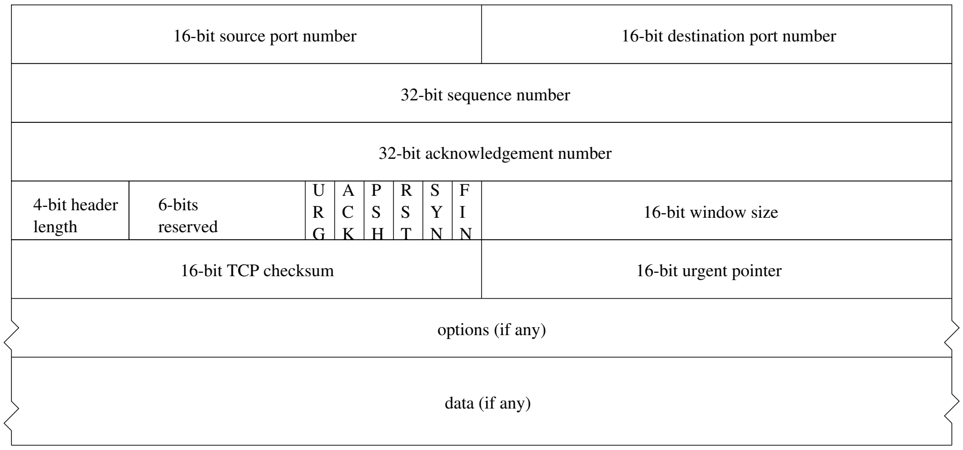


Figure 3: TCP header

  - Internet Control Message Protocol (ICMP): used by the IP layer for reporting errors, and miscelleanous functions (eg. ping).

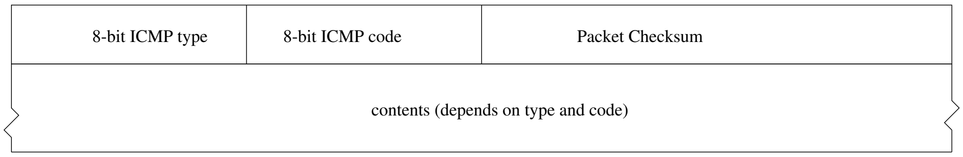


Figure 4: ICMP header

Why packet filter?

- Control: disallowing certain kinds of traffic.
- Security: protecting your boxes from certain kinds of packets.
- Watchfulness: detecting misconfiguration, being nosy.

Packet Filtering is not firewalling

- Packet Filtering can be part of building a firewall.
- Checking each packet's header is not enough to provide real protection.
- Other techniques include proxying (an application actually checks the data inside the packets), and stateful packet filtering (packet flows are tracked, and anomolies detected).

Linux as a Packet Filter

- The Linux Kernel has had packet filtering capability since the 2.0 series.
- It is a configuration option when you compile your kernel
  - But most distributions ship with it enabled.
- A special tool is used to control the tables in the kernel which tell it what packets to filter.
- Linux Packet Filtering is for IPv4 only.
  - But infrastructure exists for other protocols (IPX, AppleTalk, IPv6) to be handled at a future date
- Linux's Packet filtering code understands not only the IP header, but some parts of the TCP, UDP and ICMP protocol headers as well, making it much more useful.

2 Packets Through The Linux Kernel

In this section we will show:

- How a Packet Passes Through A Linux Box
- Where Packet Filtering Occurs
- How Packet Filtering Is Done: Rules and Chains

How A Packet Traverses the Linux Packet Filtering Code

- This has not changed much between 2.0 and 2.2.
  - Circles represent points at which packet filtering can occur.

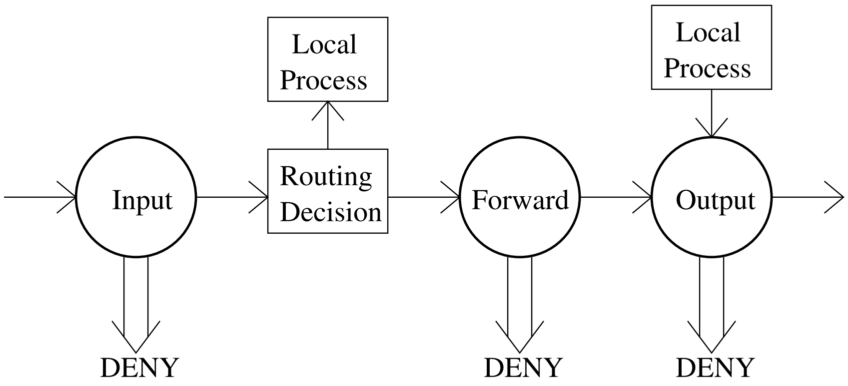


Figure 5: Linux Packet Filtering Traversal For 2.0 and 2.2

- Linux 2.0 has a special accounting point (when compiled with CONFIG_IP_ACCT).

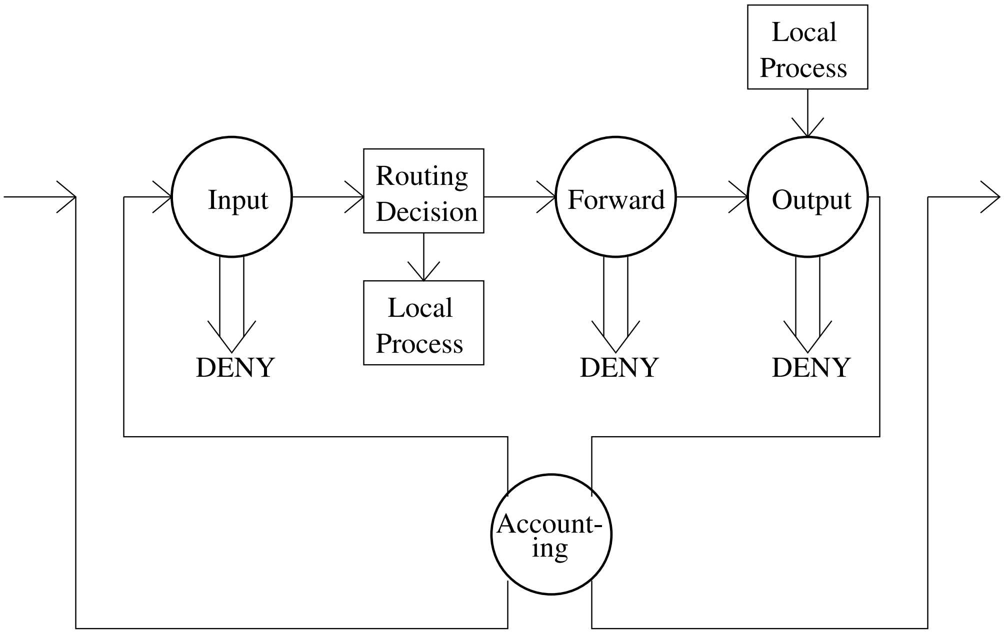


Figure 6: Linux 2.0 With IP Accounting Enabled

What Happens At Each Firewall Point: Chains

- The kernel keeps an ordered list of **rules** for each of these firewall points.
- This list is usually called a **chain**.
- Each rule in the chain is examined in order; if it doesn't tell us what to do with the packet, we look at the next one.

Three Parts of a Rule

Location Which chain the rule is in (eg. input chain).

Conditions What packets the rule matches (eg. source IP 192.168.1.1).

Target What to do when a packet matches the rule (eg. drop the packet).

3 Setting Up Rules

In this section we will show:

- The different commands used for 2.0 and 2.2 kernel chain manipulation.
- How to put rules into chains, and delete them.
- The different conditions rules can specify.
- What targets a rule can specify.

Manipulation of Rules for 2.0 and 2.2

- In Linux v2.1.102, the file net/ipv4/ip_fw.c got replaced with a new version.
- The new version was a derivation (and enhancement) of the original, and required a new tool (ipchains) to replace the old one (ipfwadm).
- The major enhancement is that the user can create new chains, on top of the three built-in chains.
- The decision was made to provide backwards compatibility in userspace, using the ipfwadm-wrapper program.
- This tutorial will concentrate on the 2.2 kernel; see the Appendix for the 2.0 equivalents.
- ipchains versions later than 1.3.5 allow long options (eg. –protocol instead of -p).

Appending a Rule

- You can append a rule to the end of a chain.

```
-A chainname
--append chainname
```

- Example:

```
ipchains -A input
```

Inserting a Rule

- You can insert a rule at any position, starting with 1 (the beginning):

```
-I chainname [position]
--insert chainname [position]
```

- If no position is specified, 1 is assumed.
- Examples:

```
ipchains -I input -j ACCEPT
ipchains -I input 2 -j ACCEPT
```

Deleting a Rule

- You can delete a rule the same way it was inserted; the first matching rule will be deleted.

```
-D chainname [rule-specifiers]
--delete chainname [rule-specifiers]
```

- Example:

```
ipchains -D input -j ACCEPT
```

- You can also delete a rule by its position in the chain.

```
-D chainname number
--delete chainname number
```

- Example:

```
ipchains -D input 2
```

Replacing a Rule

- You can atomically replace a rule by its position in the chain.

```
-R chainname number
--replace chainname number
```

- Example:

```
ipchains -R input 1
```

What a Rule Can Match in a Header

All of the following conditions can be negated by preceding them with a '!' (pronounced "not"). For example, "-p ! TCP".

Protocol

- Case insensitive, also allows numerical values or those from /etc/protocols.

```
-p protocol
--protocol protocol
```

- Example:

```
ipchains -A input -p tcp
```

Source and destination address

- Specify an individual or range of IP addresses
- Can use numerical address or host name:

```
-s source
--source source
--src source
-d destination
--destination destination
--dst destination
```

- Examples:

```
ipchains -A input -s www.linuxhq.com
ipchains -A input -s 199.95.207.0/24 -d 192.168.0.0/16
ipchains -A input -s 199.95.207.0/255.255.255.0
```

Source and destination port

- Used for TCP and UDP rules only
- Can specify an (inclusive) range with ':' representation
- Can use actual port numbers or service names (see /etc/services in Appendix C)
- You can append to the end of the source and destination options...
- ... but the preferred method is to use the long options:

```
--sport port
--source-port port
--dport port
--destination-port port
```

- Examples:

```
ipchains -A input -p udp -s any/0 domain
ipchains -A input -p udp --sport domain
ipchains -A input -p tcp -d 192.168.1.1 6000:6010
ipchains -A input -p tcp -d 192.168.1.1 --dport 6000:6010
```

ICMP type and code

- ICMP packets have a message type which indicates the nature of the packet.

Number	Name	Required By
0	echo-reply	ping
3	destination-unreachable	Any IP Traffic
5	redirect	Simple routing
8	echo-request	ping
11	time-exceeded	traceroute

- Some ICMP types have a subtype called a 'code'.
- You can specify the type after the source option, and the code after the destination option...
- ...or use the –icmp-type option followed by a numerical type or ICMP name.
  - Use 'ipchains -h icmp' to get a list of ICMP names.
- You cannot invert ICMP types when their names are used.
- Examples:

```
ipchains -A input -p icmp -s 192.168.1.0/24 0
ipchains -A input -p icmp -s 192.168.1.0/24 3 -d any/0 0
ipchains -A input -p icmp -s 192.168.1.0/24 --icmp-type network-unreachable
```

Interface

- Network interface the packet came in on, or is going out on.
  - For forward chain the interface used is (arbitrarily) the output interface
- Interface is specified by name:

```
-i interface[+]
```

- If the name ends in a '+', and interface starting with this name matches.
- Interface doesn't need to be "up" to create a rule for it
- Examples:

```
ipchains -A input -i eth0 ipchains -A input -i ppp+
```

TCP SYN packets

- Can filter on whether the SYN bit (and no ACK bit) is set in the TCP header:

```
-y
--syn
```

- SYN bit is set when the a TCP connection is being initialized:
- Examples:

```
ipchains -A input -p tcp -y
```

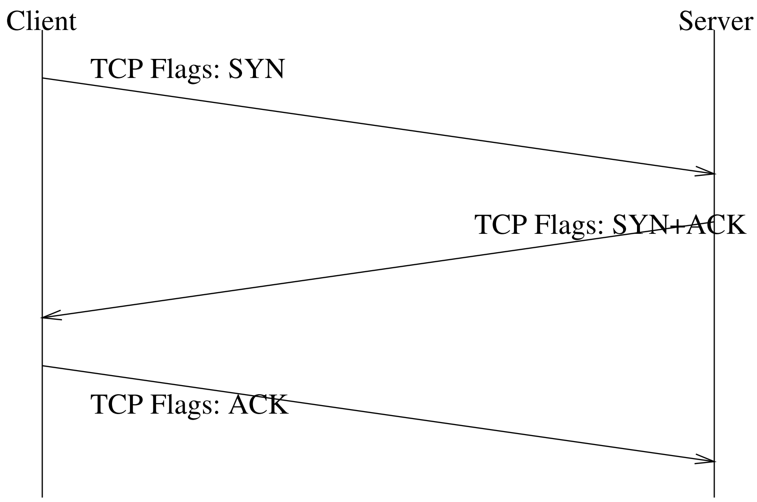


Figure 7: TCP Three-way handshake

What Targets a Rule Can Specify

A rule can specify a target: a fate for the packet:

```
-j TARGET
```

- Simple packet filtering: let the packet through:

  ```
  -j ACCEPT
  ```

- Drop the packet:

  ```
  -j DENY
  ```

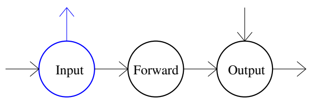
- Drop the packet, and send back an ICMP Port Unreachable message to the sender (unless it's an ICMP error message itself, or a fragment):

  ```
  -j REJECT
  ```

- Transparent proxying: accepting packets headed elsewhere (input only):

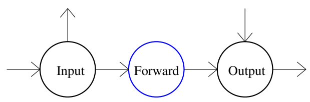
  ```
  -j REJECT
  ```

  – If no portnumber is specified, it's redirected to the same port number as it was destined for originally.

  – 

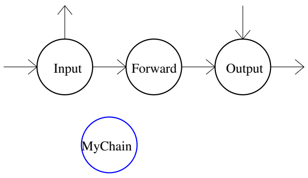
- Masquerading: changing packets to seem to come from this machine (forward only):

  ```
  -j MASQ
  ```

  – 

- Send the packet to a user-defined chain

  ```
  -j chainname
  ```

  - 

  - Major difference between 2.2 and 2.0: Only in 2.2

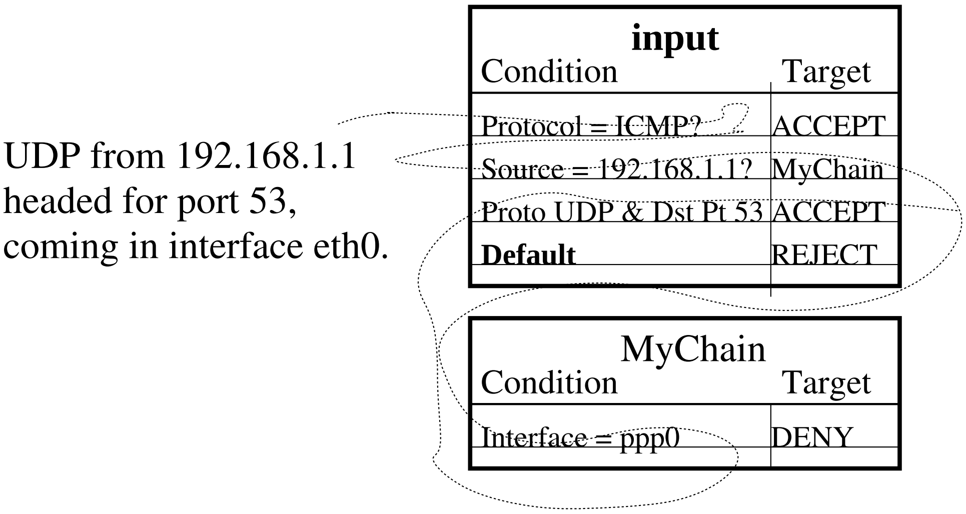


Figure 8: Packet Traversing a User-Defined Chain

- Do nothing: this has no effect on the packet (the next rule in the chain is consulted).
- Change the Type of Service (TOS) field:

  ```
  -t and xor
  ```

  - The TOS field is first ANDed with the and argument, and then XORed with the xor argument.
- Log the packet:

  ```
  -l
  ```

- Copy the packet to userspace:

  ```
  -o [maxlen]
  ```

  - The packet can be read from a special /dev file.
  - maxlen specifies the maximum size to be sent to userspace.

Chain Manipulation

There are three built-in chains, called:

```
input
forward
output
```

- We can flush the rules from an entire chain:

  ```
  -F [chainname]
  ```

  - The chain name is optional: omitting it means "all chains".
  - Example:

  ```
  ipchains -F input
  ```

- We can list the rules:

  ```
  -L [chainname]
  ```

  - Each rule has counters (both packets matched and number of bytes), which can be listed as well, if the "-v" option is given.

- We can set the policy: what happens if no rules match in a chain (only for built-in chains):

  ```
  -P chainname policy
  ```

  - Example:

  ```
  ipchains -P input DENY
  ```

- Create a new user-defined chain:

  ```
  -N chainname
  ```

  - Example:

  ```
  ipchains -N mychain
  ```

- We can also delete user-define chains, if they are empty and no rule refers to them:

  ```
  -X [chainname]
  ```

  - Example:

  ```
  ipchains -X mychain
  ```

4 Questions & Traps

These questions assume you have a simple Linux box with three interfaces. The internal interface is called 'eth0', and the external interface is called 'ppp0'. Use ipchains (ie. 2.2-style filtering) to create the following packet filters. In all cases, assume that the box starts with empty packet filtering chains (ipchains -F; ipchains -X).

1. Allow internal hosts to use WWW only.
2. Allow internal hosts WWW, FTP and DNS.

5 IP Masquerading

In this section we will show:

- What IP Masquerading is.
- Why you'd want to use it.
- Limitations on masquerading.
- How to set it up.

Masquerade: v, To appear or go about in disguise; to pass oneself off under a false character; to have or assume a deceptive appearance.

- The Linux networking code has the capability to alter packets as they pass through, to make them appear to come from the Linux box itself. This involves replacing the source address and source port.
- Used when only the masquerading box has an assigned IP address, eg. PPP dialup connection for a small network.
- The masquerading code remembers which packets have been masqueraded and automatically "demasquerades" replies; replacing the destination address and port.

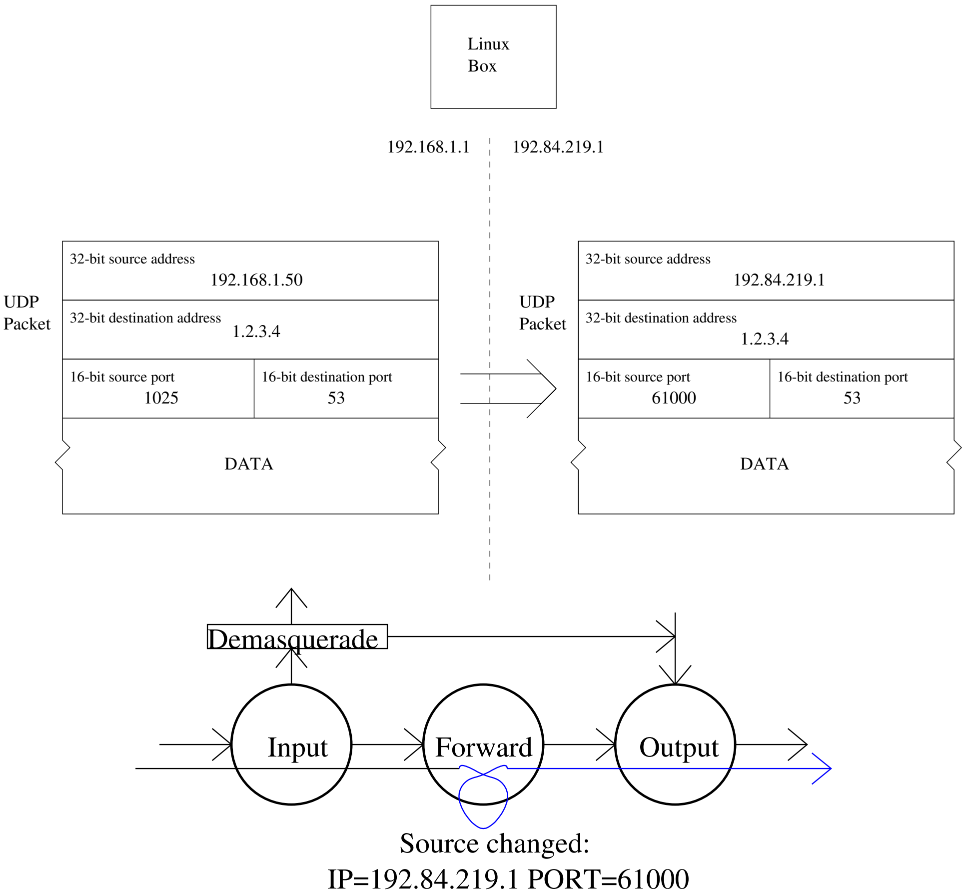


Figure 9: Example of Masquerading a UDP Packet

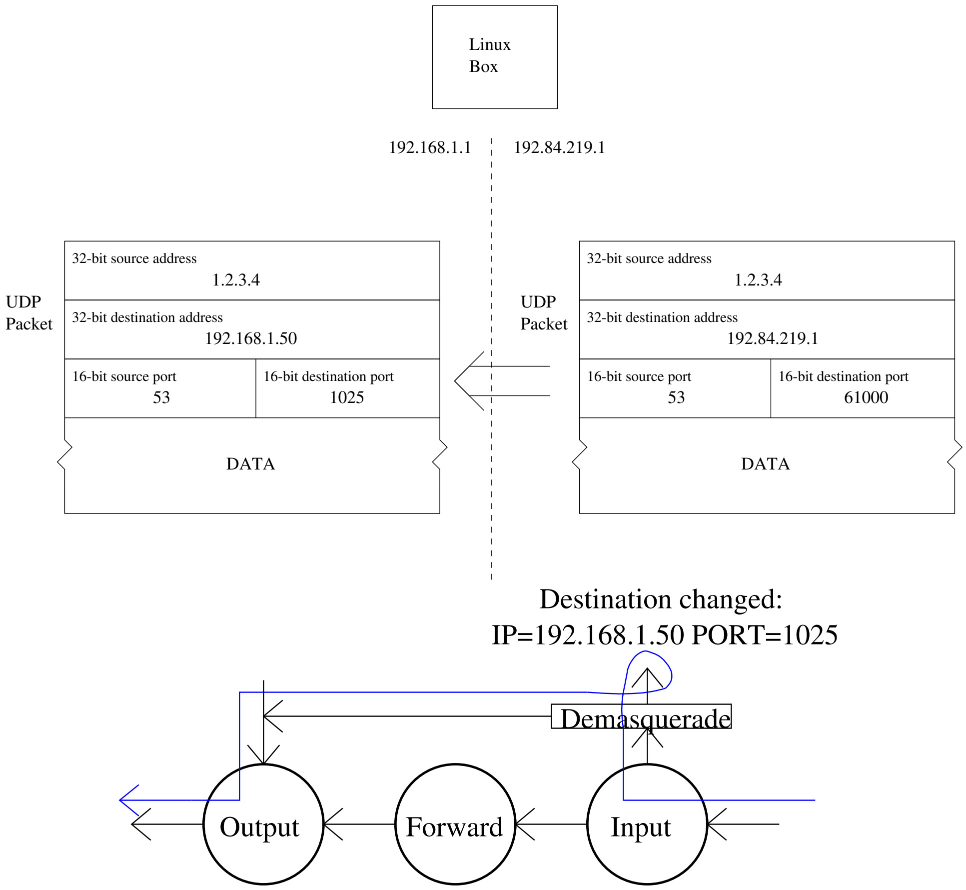


Figure 10: Demasquerading the UDP Reply Packet

Masquerading doesn't really belong with packet filtering, but that was the easiest place to implement it at the time. It has a separate maintainer, HOWTO and web page.

IP Masquerading has limitations:

- It only works for TCP, UDP (and optionally ICMP).
- You must enable CONFIG_IP_ALWAYS_DEFRAG to enable masquerading when building a kernel.
- All packets between the networks must pass through the masquerading box.
- IP addresses of private network must be taken from RFC 1597: 10.0.0.0/8, 172.168.0.0/16, or 192.168.0.0/16.
- Some protocols are intolerant to masquerading, but Linux has special "masquerading modules" for some of these.
- Modules exist for: FTP, IRC, ICQ, quake, raudio, vdolive, cuseeme

Packet Filtering With Masquerading

- Packets to masquerade are chosen by a rule in the forward chain, eg:

  ```
  # ipchains -A input -i ppp0 -j MASQ
  ```

- Best to do packet filtering in forward chain, as you have to have rules there anyway.

- Simplest possible masquerading setup (assuming ppp0 is the external interface):

  ```
  # ipchains -P forward DENY
  # ipchains -I forward -i ppp0 -j MASQ
  # echo 1 > /proc/sys/net/ipv4/ip_forward
  ```

- In the input chain, replies to masqueraded packets look like they are destined for the local machine.

- Masqueraded TCP and UDP connections use ports 61000 to 65095

- In the output chain, outgoing masqueraded packets look like they are coming from the local machine.

- Currently masqueraded connections can be listed:

  ```
  # ipchains -M -L
  ```

- Three Masquerading timeouts can set

  ```
  # ipchains -M -S 30 0 0
  ```

  – Values are in seconds with a zero time retaining the previous state
  – First value sets the time to remember a connected TCP connection. If no packets pass for this length of time, the masquerade information will be lost. Default is 15 minutes.
    * FTP control connections are particularly susceptible to this, as they are idle while large transfers occur.
  – Second value sets the time to remember a closed TCP connection. This allows time to catch any straggling packets. Default is 2 minutes.
  – Third value sets the time to remember UDP connections. Defaults to 5 minutes.

6 IP Spoof Protection

In this section we will show:

- What IP spoofing is.
- Why you'd want to prevent it.
- How to prevent it under 2.0 and 2.2 kernels.

IP spoofing is a technique where a host sends out packets which claim to be from another host

- Since packet filtering often makes decisions based on this source address, IP spoofing is used to fool packet filters.
- It is also used to hide the identity of attackers using SYN attacks, Teardrop, Ping of Death etc.
- The 2.2 kernels automatically reject packets claiming to come from the 127.* addresses (reserved for the local loopback interface lo), and any existing local interface address.
- The best way to protect from other IP spoofing is called Source Address Verification, and can be done by the routing subsystem in 2.2 kernels, and not packet filtering at all.
- Using the file /proc/sys/net/ipv4/conf/*interface*/rp_filter Source Address Verification can be turned on. (Debian 2.1 does this in its boot script).
- Causes routing code the drop a packet if the reply packet would not be routed to the same interface the packet came from; protects from spoofs of local addresses, and local spoofing of non-local addresses.
- If you cannot do this (eg. asymmetric routing), you can manually insert rules to protect every interface
  - This requires that routing information be duplicated in the packet filter rules.
  - For example, say we have three interfaces, eth0, eth1 and ppp0, we can use ifconfig to tell us the address and netmask of the interfaces. Say eth0 was attached to a network 192.168.1.0 with netmask 255.255.255.0 eth1 was attached to a network 10.0.0.0 with netmask 255.0.0.0 ppp0 connected to the Internet (where any address except the reserved private IP addresses are allowed), we would insert the following rules:

    ```
    # ipchains -A input -i eth0 -s ! 192.168.1.0/255.255.255.0 -j DENY
    # ipchains -A input -i ! eth0 -s 192.168.1.0/255.255.255.0 -j DENY
    # ipchains -A input -i eth1 -s ! 10.0.0.0/255.0.0.0 -j DENY
    # ipchains -A input -i ! eth1 -s 10.0.0.0/255.0.0.0 -j DENY
    ```

  - This approach is not as good as the Source Address Verification approach, because if your network changes, you have to change your filtering rules to keep up

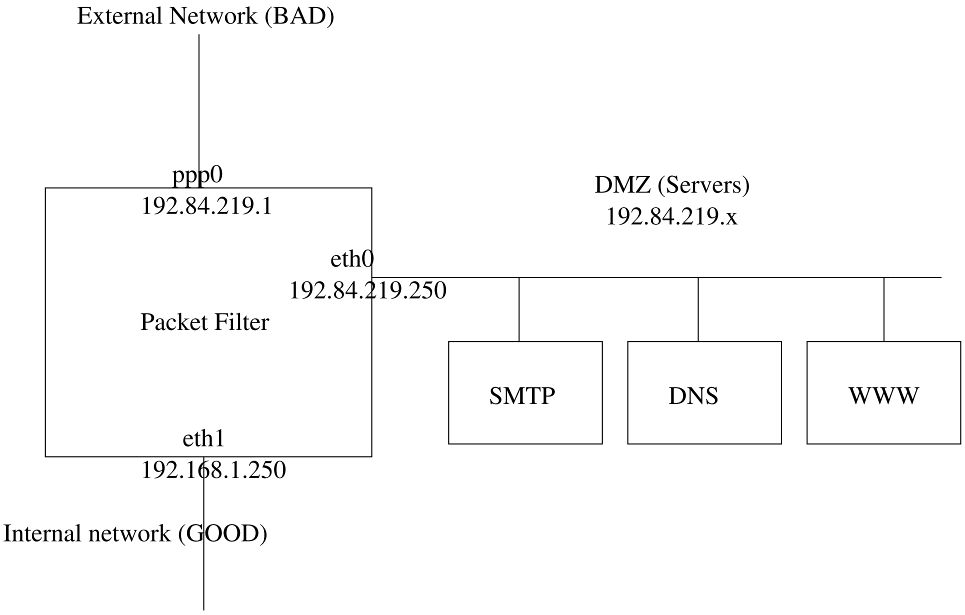
7 A Serious Example

In this section we will show:

- A fairly normal network topography.
- The use of packet filtering to protect such a network.
- Use of user-defined chains to make configuration easier.
- The techniques used when mixing masquerading and packet filtering.
- Use of packet filtering to restrict traffic to and from the packet filtering host itself.

The Arrangement

- Masqueraded internal network (various operating systems).
- Exposed servers in a separate network (DMZ).



Goals

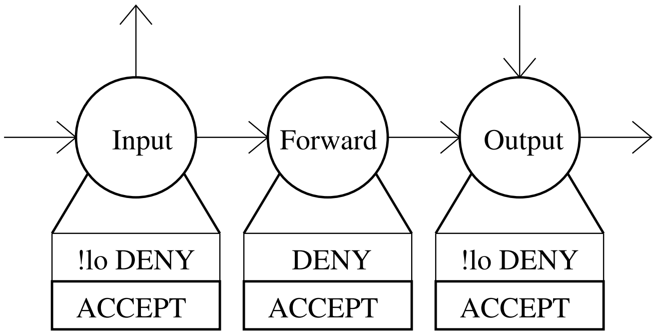
- Packet Filter box:
  - PING any network
  - TRACEROUTE any network
  - Access DNS
- DMZ:
  - Mail server ($MAILSERVER)
    * SMTP to external
    * Accept SMTP from internal and external
    * Accept POP-3 from internal
  - Name server ($NAMESERVER)
    * Send DNS to external
    * Accept DNS from internal, external and packet filter box
  - Web server ($WEBSERVER)
    * Accept HTTP from internal and external
    * Rsync access from internal
- Internal:
  - Allow WWW, ftp, traceroute, ssh to external

- Allow SMTP to Mail server
- Allow POP-3 to Mail server
- Allow DNS to Name server
- Allow rsync to Web server
- Allow WWW to Web server
- Allow ping to packet filter box

Before Packet Filtering

1. Set filtering rules to deny all:

```
# ipchains -A input -i ! lo -j DENY
# ipchains -A output -i ! lo -j DENY
# ipchains -A forward -j DENY
```



1. Prevent IP Spoofing.

```
# echo 1> /proc/sys/net/ipv4/conf/default/rp_filter
```

2. Set up interfaces:

```
# ifconfig eth0 192.84.219.0 netmask 255.255.255.0
# ifconfig eth1 192.168.1.0 netmask 255.255.255.0
# pppd defaultroute
```

3. Insert per-protocol masquerading modules.

```
# insmod ip_masq_ftp
```

Packet filtering

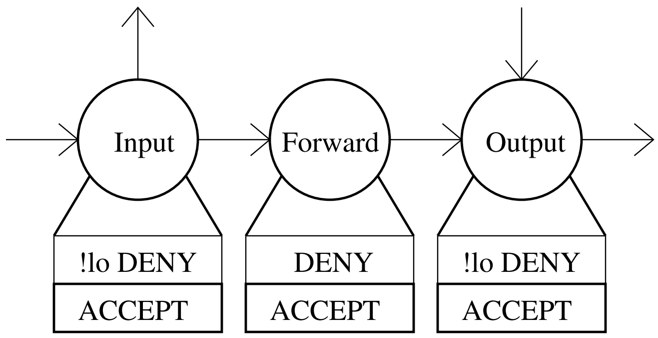
- Since we are masquerading, best to filter in the forward chain.

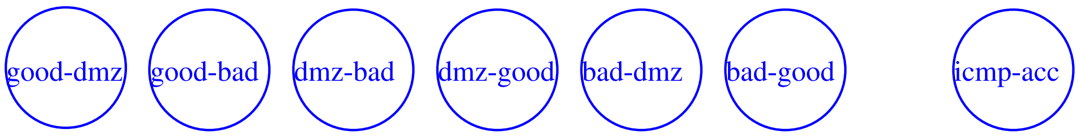
- Split forward chain into various user chains depending on source/dest interfaces.

```
# ipchains -N good-dmz
# ipchains -N bad-dmz
# ipchains -N good-bad
# ipchains -N dmz-good
# ipchains -N dmz-bad
# ipchains -N bad-good
```

- Accepting standard error ICMPs is a common thing to do; create a chain for it.

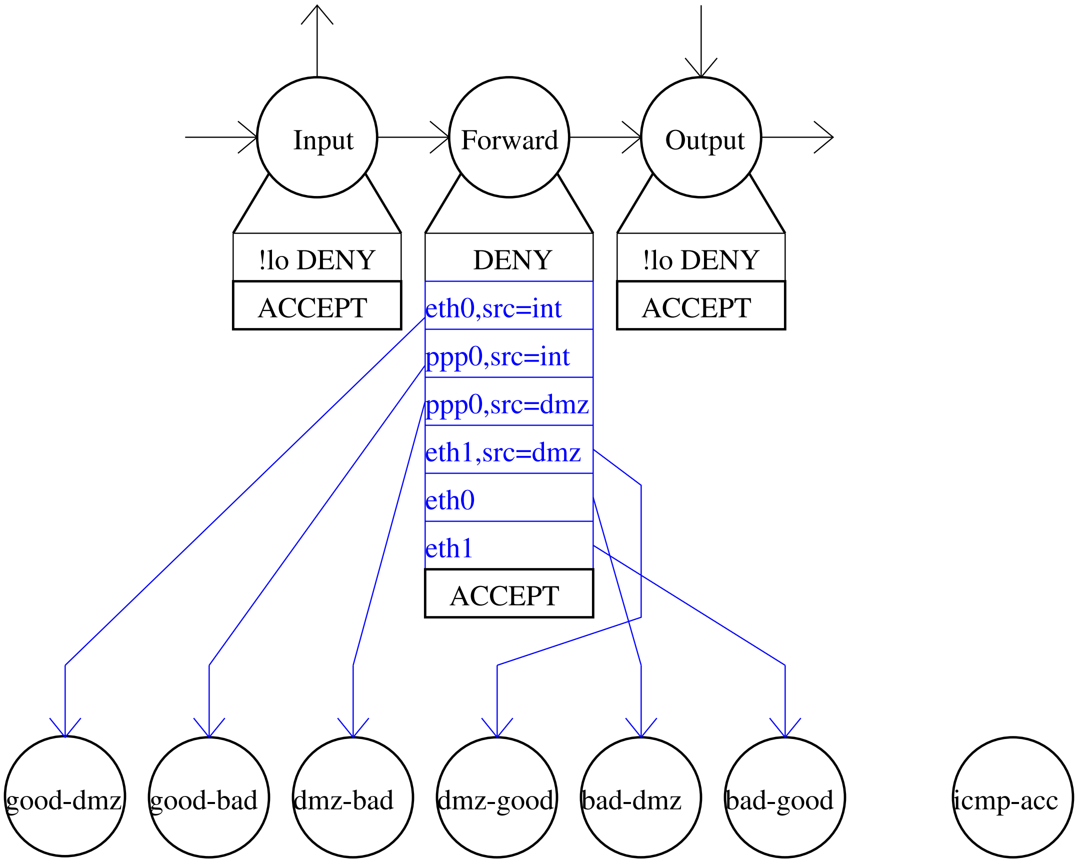
```
# ipchains -N icmp-acc
```





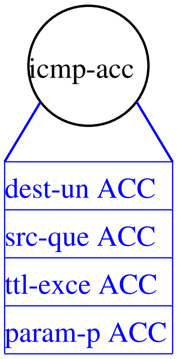
- Set up jumps from forward chain

```
# ipchains -A forward -s 192.168.1.0/24 -i eth0 -j good-dmz
# ipchains -A forward -s 192.168.1.0/24 -i ppp0 -j good-bad
# ipchains -A forward -s 192.84.219.0/24 -i ppp0 -j dmz-bad
# ipchains -A forward -s 192.84.219.0/24 -i eth1 -j dmz-good
# ipchains -A forward -i eth0 -j bad-dmz
# ipchains -A forward -i eth1 -j bad-good
# ipchains -A forward -j DENY -l
```



- Define the icmp-acc chain:

```
# ipchains -A icmp-acc -p icmp --icmp-type destination-unreachable -j ACCEPT
# ipchains -A icmp-acc -p icmp --icmp-type source-quench -j ACCEPT
# ipchains -A icmp-acc -p icmp --icmp-type time-exceeded -j ACCEPT
# ipchains -A icmp-acc -p icmp --icmp-type parameter-problem -j ACCEPT
```



/newpage

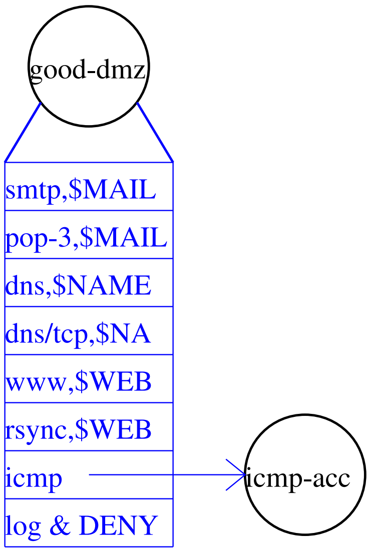
Good (Internal) to DMZ (Servers)

- Internal restrictions:

  - Allow WWW, ftp, traceroute, ssh to external
  - Allow SMTP to Mail server
  - Allow POP-3 to Mail server
  - Allow DNS to Name server
  - Allow rsync to Web server
  - Allow WWW to Web server
  - Allow ping to packet filter box

- Could do masquerading from internal network into DMZ, but here we don't.
- Log violations: they shouldn't happen.

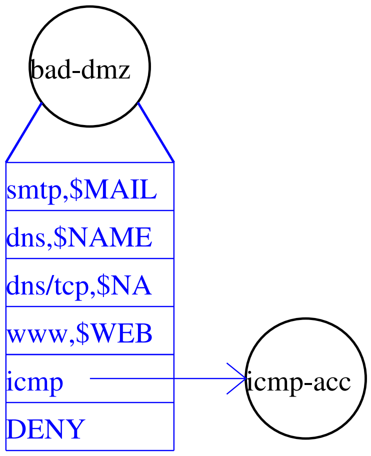
```
# ipchains -A good-dmz -p tcp -d $MAILSERVER smtp -j ACCEPT
# ipchains -A good-dmz -p tcp -d $MAILSERVER pop-3 -j ACCEPT
# ipchains -A good-dmz -p udp -d $NAMESERVER domain -j ACCEPT
# ipchains -A good-dmz -p tcp -d $NAMESERVER domain -j ACCEPT
# ipchains -A good-dmz -p tcp -d $WEBSERVER www -j ACCEPT
# ipchains -A good-dmz -p tcp -d $WEBSERVER rsync -j ACCEPT
# ipchains -A good-dmz -p icmp -j icmp-acc
# ipchains -A good-dmz -j DENY -l
```



BAD (external) to DMZ (servers).

- DMZ restrictions:
  - Mail server
    - SMTP to external
    - Accept SMTP from internal and external
    - Accept POP-3 from internal
  - Name server
    - Send DNS to external
    - Accept DNS from internal, external and packet filter box
  - Web server
    - Accept HTTP from internal and external
    - Rsync access from internal
- Things we allow from external network to DMZ.
- Don't log violations, as they may happen.

```
# ipchains -A bad-dmz -p tcp -d $MAILSERVER smtp -j ACCEPT
# ipchains -A bad-dmz -p udp -d $NAMESERVER domain -j ACCEPT
# ipchains -A bad-dmz -p tcp -d $NAMESERVER domain -j ACCEPT
# ipchains -A bad-dmz -p tcp -d $WEBSERVER www -j ACCEPT
# ipchains -A bad-dmz -p icmp -j icmp-acc
# ipchains -A bad-dmz -j DENY
```



good (internal) to bad (external).

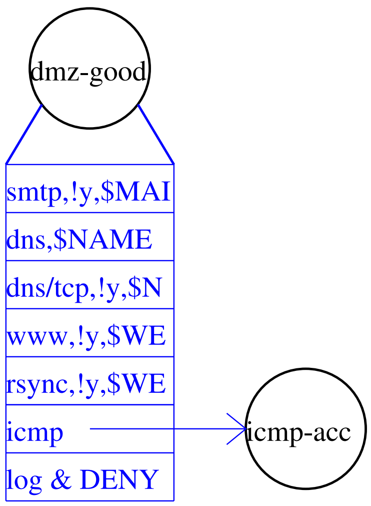
- Internal restrictions:
  - Allow WWW, ftp, traceroute, ssh to external
  - Allow SMTP to Mail server
  - Allow POP-3 to Mail server
  - Allow DNS to Name server
  - Allow rsync to Web server
  - Allow WWW to Web server
  - Allow ping to packet filter box
- Many people allow everything from the internal to external networks, then add restrictions. We're being fascist.
- Log violations.
- Passive FTP handled by masq. module.

```
# ipchains -A good-bad -p tcp --dport www -j MASQ
# ipchains -A good-bad -p tcp --dport ssh -j MASQ
# ipchains -A good-bad -p udp --dport 33434:33500 -j MASQ
# ipchains -A good-bad -p tcp --dport ftp --j MASQ
# ipchains -A good-bad -p icmp --icmp-type ping -j MASQ
# ipchains -A good-bad -j REJECT -l
```

DMZ to good (internal).

- Internal restrictions:
  - Allow WWW, ftp, traceroute, ssh to external
  - Allow SMTP to Mail server
  - Allow POP-3 to Mail server
  - Allow DNS to Name server
  - Allow rsync to Web server
  - Allow WWW to Web server
  - Allow ping to packet filter box
- If we were masquerading from the internal network to the DMZ, simply refuse any packets coming the other way. As it is, only allow packets which might be part of an established connection.

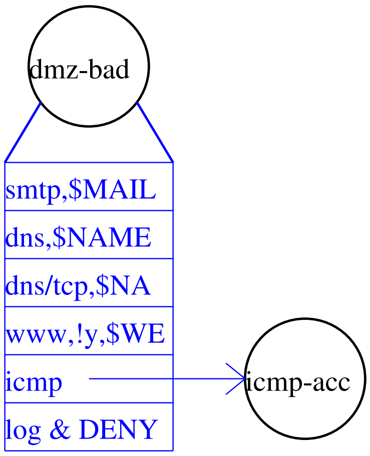
```
# ipchains -A dmz-good -p tcp ! -y -s $MAILSERVER smtp -j ACCEPT
# ipchains -A dmz-good -p udp -s $NAMESERVER domain -j ACCEPT
# ipchains -A dmz-good -p tcp ! -y -s $NAMESERVER domain -j ACCEPT
# ipchains -A dmz-good -p tcp ! -y -s $WEBSERVER www -j ACCEPT
# ipchains -A dmz-good -p tcp ! -y -s $WEBSERVER rsync -j ACCEPT
# ipchains -A dmz-good -p icmp -j icmp-acc
# ipchains -A dmz-good -j DENY -l
```



DMZ to bad (external).

- DMZ restrictions:
  - Mail server
    - SMTP to external
    - Accept SMTP from internal and external
    - Accept POP-3 from internal
  - Name server
    - Send DNS to external
    - Accept DNS from internal, external and packet filter box
  - Web server
    - Accept HTTP from internal and external
    - Rsync access from internal

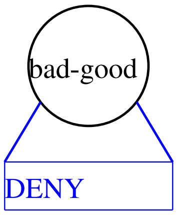
```
# ipchains -A dmz-bad -p tcp -s $MAILSERVER smtp -j ACCEPT
# ipchains -A dmz-bad -p udp -s $NAMESERVER domain -j ACCEPT
# ipchains -A dmz-bad -p tcp -s $NAMESERVER domain -j ACCEPT
# ipchains -A dmz-bad -p tcp ! -y -s $WEBSERVER www -j ACCEPT
# ipchains -A dmz-bad -p icmp -j icmp-acc
# ipchains -A dmz-bad -j DENY -l
```



bad (external) to good (internal).

- We don't allow anything (non-masqueraded) from the external network to the internal network

```
# ipchains -A bad-good -j REJECT
```



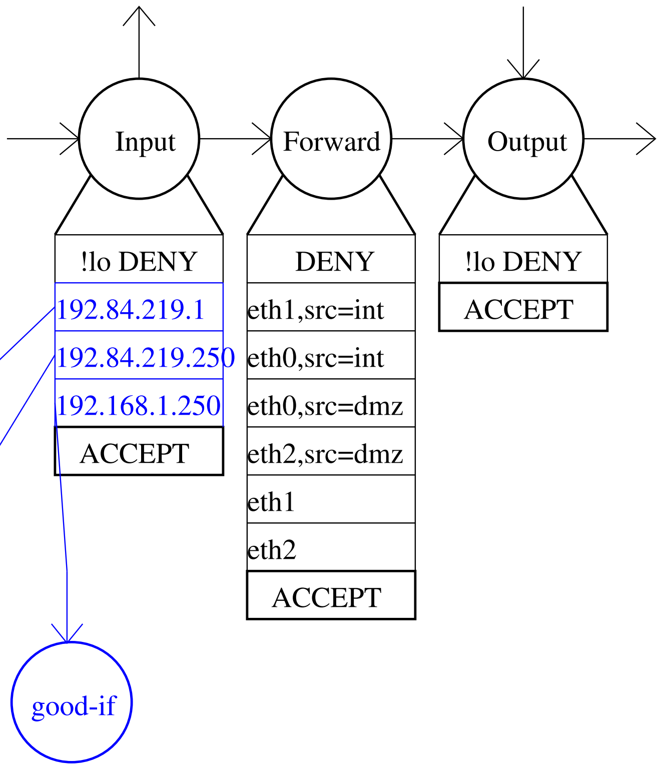
Packet Filtering for the Linux Box Itself

If we want to use packet filtering on packets coming into the box itself, we need to do filtering in the input chain. We create one chain for each destination interface:

```
# ipchains -N bad-if
# ipchains -N dmz-if
# ipchains -N good-if
```

Create jumps to them:

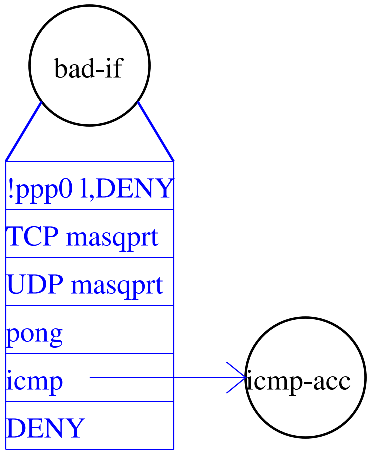
```
# ipchains -A input -d 192.84.219.1 -j bad-if
# ipchains -A input -d 192.84.219.250 -j dmz-if
# ipchains -A input -d 192.168.1.250 -j good-if
```



bad (external) interface

- Packet Filter box:
  - PING any network
  - TRACEROUTE any network
  - Access DNS
- External interface also receives replies to masqueraded packets, and ICMP errors for them and PING replies.

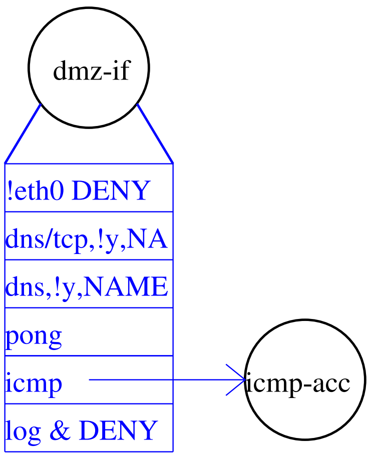
```
# ipchains -A bad-if -i ! ppp0 -j DENY -l
# ipchains -A bad-if -p TCP --dport 61000:65096 -j ACCEPT
# ipchains -A bad-if -p UDP --dport 61000:65096 -j ACCEPT
# ipchains -A bad-if -p ICMP --icmp-type pong -j ACCEPT
# ipchains -A bad-if -j icmp-acc
# ipchains -A bad-if -j DENY
```



DMZ interface

- Packet Filter box restrictions:
  - PING any network
  - TRACEROUTE any network
  - Access DNS
- DMZ interface receives DNS replies, ping replies and ICMP errors.

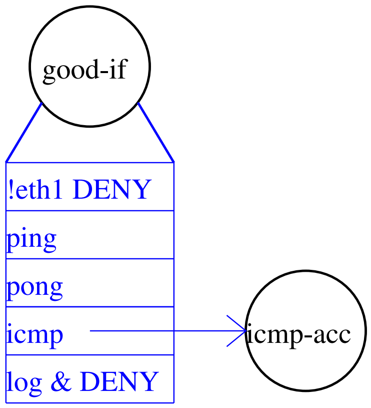
```
# ipchains -A dmz-if -i ! eth0 -j DENY
# ipchains -A dmz-if -p TCP ! -y -s $NAMESERVER 53 -j ACCEPT
# ipchains -A dmz-if -p UDP -s $NAMESERVER 53 -j ACCEPT
# ipchains -A dmz-if -p ICMP --icmp-type pong -j ACCEPT
# ipchains -A dmz-if -j icmp-acc
# ipchains -A dmz-if -j DENY -l
```



good (internal) interface

- Packet Filter box restrictions:
  - PING any network
  - TRACEROUTE any network
  - Access DNS
- Internal restrictions:
  - Allow WWW, ftp, traceroute, ssh to external
  - Allow SMTP to Mail server
  - Allow POP-3 to Mail server
  - Allow DNS to Name server
  - Allow rsync to Web server
  - Allow WWW to Web server
  - Allow ping to packet filter box
- Internal interface receives pings, ping replies and ICMP errors.

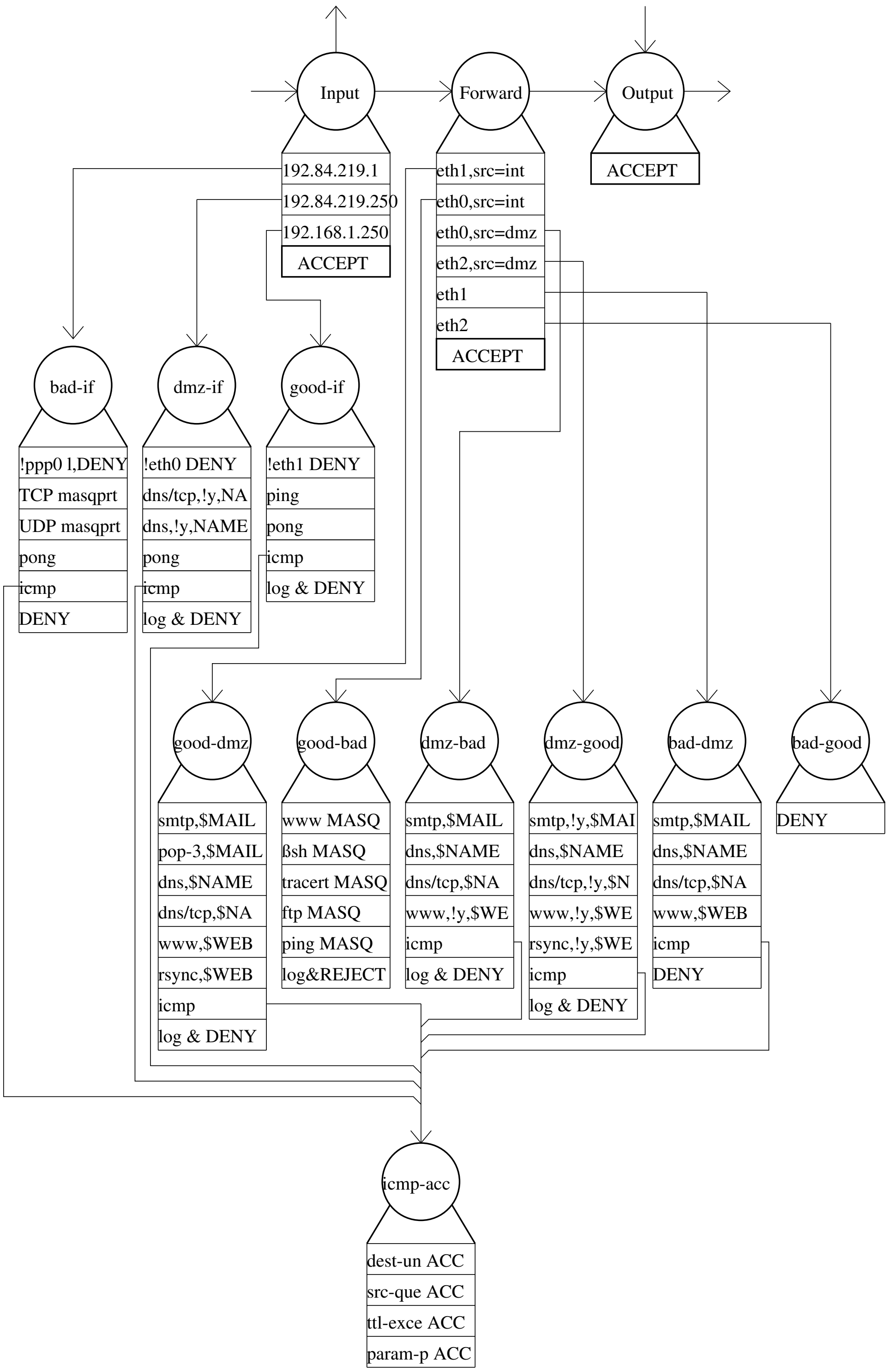
```
# ipchains -A good-if -i ! eth1 -j DENY
# ipchains -A good-if -p ICMP --icmp-type ping -j ACCEPT
# ipchains -A good-if -p ICMP --icmp-type pong -j ACCEPT
# ipchains -A good-if -j icmp-acc
# ipchains -A good-if -j DENY -l
```



Finally

- Delete blocking rules:

```
# ipchains -D input 1
# ipchains -D forward 1
# ipchains -D output 1
```


Input
Forward
Output
192.84.219.1
192.84.219.250
192.168.1.250
ACCEPT
eth1,src=int
eth0,src=int
eth0,src=dmz
eth2,src=dmz
eth1
eth2
ACCEPT
ACCEPT
bad-if
dmz-if
good-if
!ppp0 l,DENY
TCP masqprt
UDP masqprt
pong
icmp
DENY
!eth0 DENY
dns/tcp,!y,NA
dns,!y,NAME
pong
icmp
log & DENY
!eth1 DENY
ping
pong
icmp
log & DENY
good-dmz
good-bad
dmz-bad
dmz-good
bad-dmz
bad-good
smtp,$MAIL
pop-3,$MAIL
dns,$NAME
dns/tcp,$NA
www,$WEB
rsync,$WEB
icmp
log & DENY
www MASQ
ßsh MASQ
tracert MASQ
ftp MASQ
ping MASQ
log&REJECT
smtp,$MAIL
dns,$NAME
dns/tcp,$NA
www,!y,$WE
icmp
log & DENY
smtp,!y,$MAI
dns,$NAME
dns/tcp,!y,$N
www,!y,$WE
rsync,!y,$WE
icmp
log & DENY
smtp,$MAIL
dns,$NAME
dns/tcp,$NA
www,$WEB
icmp
DENY
DENY
icmp-acc
dest-un ACC
src-que ACC
ttl-exce ACC
param-p ACC

8 Other Stuff

In this section we will briefly cover:

- Further Masquerading information
  - Transparent Proxying
  - Auto Forwarding
  - Port Forwarding
- Debugging a Packet Filter
  - Using ipchains 'check' target
  - Using tcpdump
- Setting Up/Changing Filtering Rules
- Other Linux Firewalling techniques
  - Stateful Packet filter (SPF)
  - Advanced Logging
  - Netfilter: The Next Generation

Transparent Proxying

- A rule in the input chain can specify 'REDIRECT' to act as if the packet were destined for the local machine
- Really a routing override, not packet filtering.
- Can either leave the destination port the same or alter it.
- Useful for forcing other hosts to use a proxy without reconfiguring them
  - eg. Redirect all outgoing mail to the mail server.
- Used for "transparent" proxies
  - Patches exist to run Squid as a transparent proxy (see Squid FAQ).
- This kernel config also allows root-processes to bind to non-local IP addresses, completing the illusion of full access (ie. the proxy pretends to be the client, as well as pretending to be the server).
- Slows down packet forwarding somewhat.

Auto Forwarding

- Allows you to demasquerade related connections; sufficient for protocols which do not embed IP addresses and may mean you don't have to write a dedicated module.
- Uses a 'control port': this is the initial connection the client uses to connect to the external machine. You can then specify what protocol and port range (from that external machines) should be sent back to the client. eg.
  Linux 2.2:

  ```
  # ipmasqadm autofw -A -r udp 6970 7170 -c tcp 7070
  ```

  Linux 2.0:

  ```
  # ipautofw -A -r udp 6970 7170 -c tcp 7070
  ```

- This specifies that a masqueraded TCP packet with destination port 7070 should not only cause returning TCP packets from that host to be demasqueraded, but also UDP ports 6970 through 7170 (the ports usually used for the RealAudio audio stream).

Port Forwarding

- Patches available for 2.0 kernels, became standard in 2.2.
- Alters incoming packet's destination IP address and destination port (and then does the reverse for outgoing packets)
- Only works on UDP and TCP packets.
- Used for servers behind firewalls, especially inside masqueraded networks.
- Only conceptual difference between this and masquerading is that translation is set up by an incoming packet, rather than an outgoing packet.
- Can be used for limited load-sharing, by specifying multiple port forwards.
- Limited interface: can only specify a single destination IP and port per rule.

Using the 'check' command

- Both Linux 2.0 and 2.2 allow a user to discover the fate of a hypothetical packet.
- This uses the same function real packets use, thus is very reliable.
- Packet definition specified using complete rule-specification syntax

```
# ipchains -C input -s 192.168.1.1 1025 -d 192.168.2.1 80 -p tcp -i eth0
```

- You can insert into a user-defined chain as well as a builtin chain.

Using tcpdump

- tcpdump is a formidable tool for figuring out what packets are being dropped
- This is the dump of a standard TCP opening; you can see the three-way handshake.

```
23:29:57.608552 arp who-has 192.84.219.17 tell 192.84.219.11
23:29:57.608552 arp reply 192.84.219.17 is-at 0:60:67:8:62:91
23:29:57.608552 192.84.219.11.1030 > 192.84.219.17.telnet:
 S 4009933484:4009933484(0) win 31856
 <mss 1460,sackOK,timestamp 953163[|tcp]>(DF) [tos 0x10]
23:29:57.608552 192.84.219.17.telnet > 192.84.219.11.1030:
 S 17694515:17694515(0) ack 4009933485 win 8760
 <mss 1460> (DF)
23:29:57.608552 192.84.219.11.1030 > 192.84.219.17.telnet:
 . ack 1 win 31856 (DF) [tos 0x10]
```

- This is a failed TCP opening, due to a packet filter with a REJECT rule:

```
01:24:12.329187 192.168.1.121.1040 > 192.168.1.1.telnet:
 S 2768739652:2768739652(0) win 31856
 <mss 1460,sackOK,timestamp 4579135[|tcp]> (DF) [tos 0x10]
01:24:12.329187 192.168.1.1 > 192.168.1.121:
 icmp: 192.168.1.1 tcp port telnet unreachable [tos 0xd0]
```

- A Linux telnet client will display the following in this case:

```
$ telnet 192.168.1.1
Trying 192.168.1.1...
telnet: Unable to connect to remote host: Connection refused
$
```

- This is a failed TCP opening, due to a packet filter with a DENY rule:

```
01:25:50.499187 192.168.1.121.1041 > 192.168.1.1.telnet:
 S 2856939798:2856939798(0) win 31856
 <mss 1460,sackOK,timestamp 4588952[|tcp]> (DF) [tos 0x10]
01:25:53.499187 192.168.1.121.1041 > 192.168.1.1.telnet:
 S 2856939798:2856939798(0) win 31856
 <mss 1460,sackOK,timestamp 4589252[|tcp]> (DF) [tos 0x10]
01:25:59.499187 192.168.1.121.1041 > 192.168.1.1.telnet:
```

```
 S 2856939798:2856939798(0) win 31856
 <mss 1460,sackOK,timestamp 4589852[|tcp]> (DF) [tos 0x10]
01:26:11.499187 192.168.1.121.1041 > 192.168.1.1.telnet:
 S 2856939798:2856939798(0) win 31856
 <mss 1460,sackOK,timestamp 4591052[|tcp]> (DF) [tos 0x10]
01:26:35.499187 192.168.1.121.1041 > 192.168.1.1.telnet:
 S 2856939798:2856939798(0) win 31856
 <mss 1460,sackOK,timestamp 4593452[|tcp]> (DF) [tos 0x10]
```

tcpdump with masquerading

- tcpdump can give unreliable results on a machine which is actually doing the masquerading
- It is usually recommended that the dump be done from a machine not involved in the packet exchange if possible

Setting Up/Changing Filtering Rules

- There is the possibility of packets leaking through the packet filter as rules are being adjusted.
  - It is often advisable to test rules in a separate (unused) chain before inserting them into real chains.
  - Insertion of a set of new rules can be done atomically by creating a new chain and using '-I 1' to insert a jump to the new chain at the beginning of an existing chain.
  - Very large chains can be split into smaller chains, such as one-per-client for an ISP.
- Traversal time is faster if the most commonly matched rules are at the top
  - You can create a reordered chain and then substitute it atomically, for accurate accounting.
  - There are two scripts called "ipchains-save" and "ipchains-restore" which can be useful for packaging setups (eg. boot scripts).
  - There is an "ipfwadm-wrapper" script which accepts ipfwadm commands and translates them into 2.2-style commands.

Stateful Packet Filter (SPF)

- There is a program in beta called SPF (Stateful Packet Filter) for 2.2.??
- Written by Brian J. Murrell, it allows filtering based on connections rather than just packet based.
- It is similar to transparent proxying but has the advantage of being independant of protocol unlike proxying.

Advanced Logging

- You can do advanced logging by using the netlink device for 2.2:
  1. Use "mknod /dev/fwmonitor c 36 3" to create the device.
  2. Use the "-o" option to ipchains to copy packets to the device.
     - For faster processing, you can also use a number after "-o" to restrict the number of bytes to be copied.
  3. Use the "-m" option to set the 32-bit mark value (received by the device).
  4. Read data from the device in the following format (must be read in one go):
     (a) 32 bits: total length
     (b) 32 bits: mark value
     (c) IFNAMSIZ bytes: interface name
     (d) Remainder: IP packet
- You can use the "libfw" library which provides a simple interface and allows simple packet manipulation.

Netfilter: The Next Generation

- Netfilter is the next generation of packet filtering, aimed at 2.4.
- A better infrastructure for packet filtering/mangling for Linux
- Future of Packet Filtering Talk will discuss netfilter in detail

9 Appendix A: 2.0 (ipfwadm) Equivalents

Manipulating Rules

- You can append a rule to the end of a chain:

  ```
  ipfwadm -I -a accept
  ```

- You can insert a rule at the beginning of a chain:

  ```
  ipfwadm -I -i accept
  ```

- You can delete a rule the same way it was inserted; the first matching rule will be deleted.

  ```
  ipfwadm -I -d accept
  ```

- You can't insert or delete by position, or atomically replace a rule.

What a Rule Can Match in a Header

- Protocol
  - You can only specify "tcp", "udp", "icmp" or "any".
    `-P option`
- Source and destination address
  - The same as ipchains, only upper case:
    `-S & -D option`
- Source and destination port
  - Can only be specified by appending to the -S and -D options.
  - Can specify a range using a ':' separator.
  - Or you can specify a list of up to 10 ports (9 if REDIR).
- ICMP type and code
  - ICMP codes cannot be specified.
  - ICMP types can only be specified numerically, after the -S option.
- Interface
  - The interface can be specified by name:
    `-W option`
  - No wildcards are possible: exact matches only.
  - The interface can also be specified by address:
    `-V option`
- TCP SYN packets
  - Same flag as ipchains:
    `-y`
  - The 2.0 kernel matches all non-TCP packets if no '-P tcp' is specified with '-y'.

What Targets a Rule Can Specify

- A rule can specify a target: a fate for the packet. This is specified after the '-a', '-i' or '-d' flags, and you only need the first letter of the target.
  - Simple packet filtering: let the packet through:

    ```
    accept
    ```
  - Drop the packet:

    ```
    deny
    ```
  - Drop the packet, and send back an ICMP Port Unreachable message to the sender (unless it's an ICMP error message itself, or a fragment):

    ```
    reject
    ```
    * This only works in the input chain for 2.0; in other chains it's equivalent to "deny".
  - Transparent proxying: accepting packets headed elsewhere (input only):

    ```
    -r [portnum]
    ```
    * If no portnumber is specified, it's redirected to the same port number as it was destined for originally.
  - Masquerading: changing packets to seem to come from this machine (forward only):

    ```
    -m
    -a masquerade
    ```
  - There are no user-defined chains, so you can't send a packet to one.
  - There is no way to specify a rule which has no effect: in the accounting chain, all rules have no effect, in the other chains, the first rule which matches will control the fate of the packet.
  - Change the Type of Service (TOS) field:

    ```
    -t and xor
    ```
    * The TOS field is first ANDed with the and argument, and then XORed with the xor argument.
    * The least-significant TOS bit can never be altered: attempts to do so are silently ignored.
  - Log the packet:

    ```
    -o
    ```
  - There is no way to copy the packet to userspace.

10 Appendix B: The ipchains man page

IPCHAINS(8)

NAME

ipchains - IP firewall administration

SYNOPSIS

ipchains -[ADC] chain rule-specification [options]
ipchains -[RI] chain rulenum rule-specification [options]
ipchains -D chain rulenum [options]
ipchains -[LFZNX] [chain] [options]
ipchains -P chain target [options]
ipchains -M [-L | -S] [options]

DESCRIPTION

Ipchains is used to set up, maintain, and inspect the IP firewall rules in the Linux kernel. These rules can be divided into 4 different categories: the IP input chain, the IP output chain, the IP forwarding chain, and user defined chains.

For each of these categories, a separate table of rules is maintained, any of which might refer to one of the userdefined chains. See ipfw(4) for more details.

TARGETS

A firewall rule specifies criteria for a packet, and a target. If the packet does not match, the next rule in the chain is the examined; if it does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain, or one of the special values ACCEPT, DENY, REJECT, MASQ, REDIRECT, or RETURN. ACCEPT means to let the packet through. DENY means to drop the packet on the floor. REJECT means the same as drop, but is more polite and easier to debug, since an ICMP message is sent back to the sender indicating that the packet was dropped. (Note that DENY and REJECT are the same for ICMP packets).

MASQ is only legal for the forward and user defined chains, and can only be used when the kernel is compiled with CONFIG_IP_MASQUERADE defined. With this, packets will be masqueraded as if they originated from the local host. Furthermore, reverse packets will be recognized as such and they will be demasqueraded automatically, bypassing the forwarding chain.

REDIRECT is only legal for the input and user-defined chains and can only be used when the Linux kernel is compiled with CONFIG_IP_TRANSPARENT_PROXY defined. With this, packets will be redirected to a local socket, even if they were sent to a remote host. If the specified redirection port is 0, which is the default value, the destination port of a packet will be used as the redirection port. When this target is used, an optional extra argument (the port number) can be supplied.

If the end of a user-defined chain is reached, or a rule with target RETURN is matched, then the next rule in the previous (calling) chain is examined. If the end of a builtin chain is reached, or a rule in a builtin chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

OPTIONS

The options that are recognized by ipchains can be divided into several different groups.

COMMANDS These options specify the specific action to perform; only one of them can be specified on the command line, unless otherwise specified below. For all the long versions of the command and option names, you only need to use enough letters to ensure that ipchains can differentiate it from all other options.

-A, –append Append one or more rules to the end of the selected chain. When the source and/or destination names resolve to more than one address, a rule will be added for each possible address combination.

-D, –delete Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

-R, –replace Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.

-I, –insert Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain.

-L, –list List all rules in the selected chain. If no chain is selected, all chains are listed. It is legal to specify the -Z (zero) option as well, in which case no chain may be specified. The exact output is effected by the other arguments given.

-F, –flush Flush the selected chain. This is equivalent to deleting all the rules one by one.

-Z, –zero Zero the packet and byte counters in all chains. It is legal to specify the -L, –list (list) option as well, to see the counters immediately before they are cleared; if this is done, then no specific chain can be specified (they will all be displayed and cleared.

-N, –new-chain Create a new user-defined chain of the given name. There must be no target of that name already.

-X, –delete-chain Delete the specified user-defined chain. There must be no references to the chain (if there are you must delete or replace the referring rules before the chain can be deleted). If no argument is given, it will attempt to delete every non-builtin chain.

-P, –policy Set the policy for the chain to the given target. See the section TARGETS for the legal targets. Only non-userdefined chains can have policies, and neither built-in nor user-defined chains can be policy targets.

-M, –masquerading This option allows viewing of the currently masqueraded connections (in conjuction with the -L option) or to set the kernel masqerading parameters (with the -S option).

-S, –set tcp tcpfin udp Change the timeout values used for masquerading. This command always takes 3 parameters, representing the timeout values (in seconds) for TCP sessions, TCP sessions after receiving a FIN packet, and UDP packets, respectively. A timeout value 0 means that the current timeout value of the corresponding entry is preserved. This option is only allowed in combination with the -M flag.

-C, –check Check the given packet against the selected chain. This is extremely useful for testing, as the same kernel routines used to check "real" network packets are used to check this packet. It can be used to check user-defined chains as well as the builtin ones. The same arguments used to specify firewall rules are used to construct the packet to be tested. In particular, the -s (source), -d (destination), -p (protocol), and -i (interface) flags are compulsory.

-h Help. Give a (currently very brief) description of the command syntax.

PARAMETERS The following parameters make up a rule specification (as used in the add, delete, replace, append and check commands).

-p, –protocol [!] protocol The protocol of the rule or of the packet to check. The specified protocol can be one of tcp, udp, icmp, or all, or it can be a numeric value, representing one of these protocols or a different one. Also a protocol name from /etc/protocols is allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to all. Protocol all will match with all protocols and is taken as default when this option is omitted. All may not be used in in combination with the check command.

-s, –source [!] address[/mask] [!] [port[:port]] Source specification. Address can be either a hostname, a network name, or a plain IP address. The mask can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of 24 is equivalent to 255.255.255.0. A "!" argument before the address specification inverts the sense of the address. The source may include a port specification or ICMP type. This can either be a service name, a port number, a numeric ICMP type, or one of the ICMP type names shown by the command

```
ipchains -h icmp
```

Note that many of these ICMP names refer to both a type and code, meaning that an ICMP code after the -d flag is illegal. In the rest of this paragraph, a port means either a port specification or an ICMP type. An inclusive range is can also be specified, using the format port:port. If the first port is omitted, "0" is assumed; if the last is omitted, "65535" is assumed.
Ports may only be specified in combination with the tcp, udp, or icmp protocols. A "!" before the port specification inverts the sense. When the check command is specified, exactly one port is required, and if the -f (fragment) flag is specified, no ports are allowed. The flag –src is a convenience alias for this option.

–source-port [!] [port[:port]] This allows separate specifiction of the source port or port range. See the description of the -s flag above for details. The flag –sport is an alias for this option.

-d, –destination [!] address[/mask] [!] [port[:port]] Destination specification. See the desciption of the -s (source) flag for a detailed description of the syntax. For ICMP, which does not have ports, a "destination port" refers to the numeric ICMP code. The flag –dst is a convenience alias for this option.

–destination-port [!] [port[:port]] This allows separate specifiction of the ports. See the description of the -s flag for details. The flag –dport is an alias for this option.

–icmp-type [!] typename This allows specification of the ICMP type (use the -h icmp option to see valid ICMP type names). This is often more convenient to appending it to the destination specification.

-j, –jump target This specifies the target of the rule; ie. what to do if the packet matches it. The target can be a user-defined chain (not the one this rule is in) or one of the special targets which decide the fate of the packet immediately. If this option is omitted in a rule, then matching the rule will have no effect on the packet's fate, but the counters on the rule will be incremented.

-i, –interface [!] name Optional name of an interface via which a packet is received (for packets entering the input chain), or via which is packet is going to be sent (for packets entering the forward or output chains). When this option is omitted, the empty string is assumed, which has a special meaning and will match with any interface name. When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match.

[!] -f, –fragment This means that the rule only refers to second and furthur fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the sense is inverted.

OTHER OPTIONS The following additional options can be specified:

-b, –bidirectional Bidirectional mode. The rule will match with IP packets in both directions; this has the same effect as repeating the rule with the source & destination reversed.

-v, –verbose Verbose output. This option makes the list command show the interface address, the rule options (if any), and the TOS masks. The packet and byte counters are also listed, with the suffix 'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipliers respectively (but see the -x flag to change this). When used in combination with -M, information related to delta sequence numbers will also be listed. For appending, insertion, deletion and replacement, this causes detailed information on the rule or rules to be printed.

-n, –numeric Numeric output. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display them as host names, network names, or services (whenever applicable).

-l, –log Turn on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will print some information of all matching packets (like most IP header fields) via printk().

-o, –output [maxsize] Copy matching packets to the userspace device. This is currently mainly for developers who want to play with firewalling effects in userspace. The optional maxsize argument can be used to limit the maximum number of bytes from the packet which are to be copied. This option is only valid if the kernel has been compiled with CONFIG_IP_FIREWALL_NETLINK set.

-m, –mark markvalue Mark matching packets. Packets can be marked with a 32-bit unsigned value which may (one day) change how they are handled internally. If you are not a kernel hacker you are unlikely to care about this. If the string markvalue begins with a + or -, then this value will be added or subtracted from the current marked value of the packet (which starts at zero).

-t, –TOS andmask xormask Masks used for modifying the TOS field in the IP header. When a packet matches a rule, its TOS field is first bitwise and'ed with first mask and the result of this will be bitwise xor'ed with the second mask. The masks should be specified as hexadecimal 8-bit values. As the LSB of the TOS field must be unaltered (RFC 1349), TOS values which would cause it to be altered are rejected, as are any rules which always set more than TOS bit. Rules which might set multiple TOS bits for certain packets result in warnings (sent to stdout) which can be ignored if you know that packets with those TOS values will never reach that rule. Obviously, manipulating the TOS is a meaningless gesture if the rule's target is DENY or REJECT.

-x, –exact Expand numbers. Display the exact value of the packet and byte counters, instead of only the rounded number in K's (multiples of 1000) M's (multiples of 1000K) or G's (multiples of 1000M). This option is only relevent for the -L command.

[!] -y, –syn Only match TCP packets with the SYN bit set and the ACK and FIN bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. This option is only meaningful when the protocol type is set to TCP. If the "!" flag precedes the "-y", the sense of the option is inverted.

–line-numbers When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in the chain.

–no-warnings Disable all warnings.

FILES

/proc/net/ip_fwchains /proc/net/ip_masquerade

DIAGNOSTICS

Various error messages are printed to standard error. The exit code is 0 for correct functioning. Errors which appear to be caused by invalid or abused command line parameters cause an exit code of 2, and other errors cause an exit code of 1.

BUGS

If input is a terminal, and a rule is inserted in, or appended to, the forward chain, and IP forwarding does not seem to be enabled, and –no-warnings is not specified, a message is printed to standard output, warning that no forwarding will occur until this is rectified. This is to help users unaware of the requirement (which did not exist in the 2.0 kernels).

There is no way to reset the packet and byte counters in one chain only. This is a kernel limitation.

Loop detection is not done in ipchains; packets in a loop get dropped and logged, but that's the first you'll find out about it if you inadvertantly create a loop.

The explanation of what effect marking a packet has is intentionally vague until documentation describing the new 2.1 kernel's packet scheduling routines is released.

There is no way to zero the policy counters (ie. those on the built-in chains).

NOTES

This ipchains is very different from the ipfwadm by Jos Vos, as it uses the new IP firewall trees. Its functionality is a superset of ipfwadm, and there is generally a 1:1 mapping of commands. I believe the new command names are more rational. There are, however, a few changes of which you should be aware.

Fragments are handled differently. All fragments after the first used to be let through (which is usually safe); they can now be filtered. This means that you should probably add an explicit rule to accept fragments if you are converting over. Also, look for old accounting rules which check for source and destination ports of 0xFFFF (0xFF for ICMP packets) which was the old way of doing accounting on fragments.

Accounting rules are now simply integrated into the input and output chains; you can simulate the old behaviour like so:

```
ipchains -N acctin
ipchains -N acctout
ipchains -N acctio
ipchains -I input -j acctio
ipchains -I input -j acctin
ipchains -I output -j acctio
ipchains -I output -j acctout
```

This creates three user-defined chains, acctin, acctout and acctio, which are to contain any accounting rules (these rules should be specified without a -j flag, so that the packets simply pass through them unscathed).

A MASQ or REDIRECT target encountered by the kernel out of place (ie. not during a forward or input rule respectively) will cause a message to the syslog and the packet to be dropped.

The old behaviour of SYN and ACK matching (which was previously ignored for non-TCP packets) has changed; the SYN option is not valid for non-TCP-specific rules.

The ACK matching option (-k) is no longer supported; the combination of ! and -y will give the equivalent).

It is now illegal to specify a TOS mask which will set or alter the least significant TOS bit; previously TOS masks were silently altered by the kernel if they tried to do this.

The -b flag is now handled by simply inserting or deleting a pair of rules, one with the source and destination specifications reversed.

There is no way to specify an interface by address: use its name.

SEE ALSO

ipfw(4)

AUTHOR

Rusty Russell <ipchains@rustcorp.com>

11 Appendix C: Debian's netbase 3.14-3 /etc/services

```
# /etc/services:
# $Id: services,v 1.4 1997/05/20 19:41:21 tobias Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
msp             18/udp                          # message send protocol
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp                          # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp          mail
# 26 - unassigned
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
re-mail-ck      50/tcp                          # Remote Mail Checking Protocol
re-mail-ck      50/udp                          # Remote Mail Checking Protocol
domain          53/tcp          nameserver      # name-domain server
domain          53/udp          nameserver
mtp             57/tcp                          # deprecated
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp                          # Internet Gopher
gopher          70/udp
rje             77/tcp          netrjs
```

```
finger          79/tcp
www             80/tcp          http            # WorldWideWeb HTTP
www             80/udp                          # HyperText Transfer Protocol
link            87/tcp          ttylink
kerberos        88/tcp          kerberos5 krb5  # Kerberos v5
kerberos        88/udp          kerberos5 krb5  # Kerberos v5
supdup          95/tcp
# 100 - reserved
hostnames       101/tcp         hostname        # usually from sri-nic
iso-tsap        102/tcp         tsap            # part of ISODE.
csnet-ns        105/tcp         cso-ns          # also used by CSO name server
csnet-ns        105/udp         cso-ns
# unfortunately the poppassd (Eudora) uses a port which has already
# been assigned to a different service. We list the poppassd as an
# alias here. This should work for programs asking for this service.
# (due to a bug in inetd the 3com-tsmux line is disabled)
#3com-tsmux     106/tcp         poppassd
#3com-tsmux     106/udp         poppassd
rtelnet         107/tcp                         # Remote Telnet
rtelnet         107/udp
pop-2           109/tcp         postoffice      # POP version 2
pop-2           109/udp
pop-3           110/tcp                         # POP version 3
pop-3           110/udp
sunrpc          111/tcp         portmapper      # RPC 4.0 portmapper TCP
sunrpc          111/udp         portmapper      # RPC 4.0 portmapper UDP
auth            113/tcp         authentication tap ident
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp         readnews untp   # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp                         # Network Time Protocol
netbios-ns      137/tcp                         # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/tcp                         # NETBIOS Datagram Service
netbios-dgm     138/udp
netbios-ssn     139/tcp                         # NETBIOS session service
netbios-ssn     139/udp
imap2           143/tcp                         # Interim Mail Access Proto v2
imap2           143/udp
snmp            161/udp                         # Simple Net Mgmt Proto
snmp-trap       162/udp         snmptrap        # Traps for SNMP
cmip-man        163/tcp                         # ISO mgmt over IP (CMOT)
cmip-man        163/udp
cmip-agent      164/tcp
cmip-agent      164/udp
mailq           174/tcp                         # Mailer transport queue for Zmailer
mailq           174/udp                         # Mailer transport queue for Zmailer
xdmcp           177/tcp                         # X Display Mgr. Control Proto
xdmcp           177/udp
nextstep        178/tcp         NeXTStep NextStep       # NeXTStep window
nextstep        178/udp         NeXTStep NextStep       # server
bgp             179/tcp                         # Border Gateway Proto.
bgp             179/udp
```

```
prospero        191/tcp                         # Cliff Neuman's Prospero
prospero        191/udp
irc             194/tcp                         # Internet Relay Chat
irc             194/udp
smux            199/tcp                         # SNMP Unix Multiplexer
smux            199/udp
at-rtmp         201/tcp                         # AppleTalk routing
at-rtmp         201/udp
at-nbp          202/tcp                         # AppleTalk name binding
at-nbp          202/udp
at-echo         204/tcp                         # AppleTalk echo
at-echo         204/udp
at-zis          206/tcp                         # AppleTalk zone information
at-zis          206/udp
qmtp            209/tcp                         # The Quick Mail Transfer Protocol
qmtp            209/udp                         # The Quick Mail Transfer Protocol
z3950           210/tcp         wais            # NISO Z39.50 database
z3950           210/udp         wais
ipx             213/tcp                         # IPX
ipx             213/udp
imap3           220/tcp                         # Interactive Mail Access
imap3           220/udp                         # Protocol v3
rpc2portmap     369/tcp
rpc2portmap     369/udp                         # Coda portmapper
codaauth2       370/tcp
codaauth2       370/udp                         # Coda authentication server
ulistserv       372/tcp                         # UNIX Listserv
ulistserv       372/udp
ldap            389/tcp                         # Lightweight Directory Access Protocol
ldap            389/udp                         # Lightweight Directory Access Protocol
https           443/tcp                         # MCom
https           443/udp                         # MCom
snpp            444/tcp                         # Simple Network Paging Protocol
snpp            444/udp                         # Simple Network Paging Protocol
saft            487/tcp                         # Simple Asynchronous File Transfer
saft            487/udp                         # Simple Asynchronous File Transfer
npmp-local      610/tcp         dqs313_qmaster  # npmp-local / DQS
npmp-local      610/udp         dqs313_qmaster  # npmp-local / DQS
npmp-gui        611/tcp         dqs313_execd    # npmp-gui / DQS
npmp-gui        611/udp         dqs313_execd    # npmp-gui / DQS
hmmp-ind        612/tcp         dqs313_intercell# HMMP Indication / DQS
hmmp-ind        612/udp         dqs313_intercell# HMMP Indication / DQS
#
# UNIX specific services
#
exec            512/tcp
biff            512/udp         comsat
login           513/tcp
who             513/udp         whod
shell           514/tcp         cmd             # no passwords used
syslog          514/udp
printer         515/tcp         spooler         # line printer spooler
talk            517/udp
ntalk           518/udp
```

```
route           520/udp         router routed   # RIP
timed           525/udp         timeserver
tempo           526/tcp         newdate
courier         530/tcp         rpc
conference      531/tcp         chat
netnews         532/tcp         readnews
netwall         533/udp                         # -for emergency broadcasts
gdomap          538/tcp                         # GNUstep distributed objects
gdomap          538/udp                         # GNUstep distributed objects
uucp            540/tcp         uucpd           # uucp daemon
afpovertcp      548/tcp                         # AFP over TCP
afpovertcp      548/udp                         # AFP over TCP
remotefs        556/tcp         rfs_server rfs  # Brunhoff remote filesystem
klogin          543/tcp                         # Kerberized 'rlogin' (v5)
kshell          544/tcp         krcmd           # Kerberized 'rsh' (v5)
nntps           563/tcp         snntp           # NNTP over SSL
nntps           563/udp         snntp           # NNTP over SSL
ldaps           636/tcp                         # LDAP over SSL
ldaps           636/udp                         # LDAP over SSL
kerberos-adm    749/tcp                         # Kerberos 'kadmin' (v5)
#
webster         765/tcp                         # Network dictionary
webster         765/udp
rsync           873/tcp                         # rsync
rsync           873/udp                         # rsync
imaps           993/tcp                         # IMAP over SSL
imaps           993/udp                         # IMAP over SSL
pop3s           995/tcp                         # POP-3 over SSL
pop3s           995/udp                         # POP-3 over SSL
#
# From "Assigned Numbers":
#
#> The Registered Ports are not controlled by the IANA and on most systems
#> can be used by ordinary user processes or programs executed by ordinary
#> users.
#
#> Ports are used in the TCP [45,106] to name the ends of logical
#> connections which carry long term conversations.  For the purpose of
#> providing services to unknown callers, a service contact port is
#> defined.  This list specifies the port used by the server process as its
#> contact port.  While the IANA can not control uses of these ports it
#> does register or list uses of these ports as a convienence to the
#> community.
#
socks           1080/tcp                        # socks proxy server
socks           1080/udp                        # socks proxy server
ingreslock      1524/tcp
ingreslock      1524/udp
prospero-np     1525/tcp                        # Prospero non-privileged
prospero-np     1525/udp
datametrics     1645/tcp        old-radius      # datametrics / old radius entry
datametrics     1645/udp        old-radius      # datametrics / old radius entry
sa-msg-port     1646/tcp        old-radacct     # sa-msg-port / old radacct entry
sa-msg-port     1646/udp        old-radacct     # sa-msg-port / old radacct entry
```

```
radius          1812/tcp                        # Radius
radius          1812/udp                        # Radius
radacct         1813/tcp                        # Radius Accounting
radacct         1813/udp                        # Radius Accounting
cvspserver      2401/tcp                        # CVS client/server operations
cvspserver      2401/udp                        # CVS client/server operations
venus           2430/tcp                        # codacon port
venus           2430/udp                        # Venus callback/wbc interface
venus-se        2431/tcp                        # tcp side effects
venus-se        2431/udp                        # udp sftp side effect
codasrv         2432/tcp                        # not used
codasrv         2432/udp                        # server port
codasrv-se      2433/tcp                        # tcp side effects
codasrv-se      2433/udp                        # udp sftp side effect
dict            2628/tcp                        # Dictionary server
dict            2628/udp                        # Dictionary server
icpv2           3130/tcp        icp             # Internet Cache Protocol (Squid)
icpv2           3130/udp        icp             # Internet Cache Protocol (Squid)
mysql           3306/tcp                        # MySQL
mysql           3306/udp                        # MySQL
rfe             5002/tcp                        # Radio Free Ethernet
rfe             5002/udp                        # Actually uses UDP only
cfengine        5308/tcp                        # CFengine
cfengine        5308/udp                        # CFengine
afs3-fileserver 7000/tcp        bbs             # file server itself
afs3-fileserver 7000/udp        bbs             # file server itself
afs3-callback   7001/tcp                        # callbacks to cache managers
afs3-callback   7001/udp                        # callbacks to cache managers
afs3-prserver   7002/tcp                        # users & groups database
afs3-prserver   7002/udp                        # users & groups database
afs3-vlserver   7003/tcp                        # volume location database
afs3-vlserver   7003/udp                        # volume location database
afs3-kaserver   7004/tcp                        # AFS/Kerberos authentication
afs3-kaserver   7004/udp                        # AFS/Kerberos authentication
afs3-volser     7005/tcp                        # volume managment server
afs3-volser     7005/udp                        # volume managment server
afs3-errors     7006/tcp                        # error interpretation service
afs3-errors     7006/udp                        # error interpretation service
afs3-bos        7007/tcp                        # basic overseer process
afs3-bos        7007/udp                        # basic overseer process
afs3-update     7008/tcp                        # server-to-server updater
afs3-update     7008/udp                        # server-to-server updater
afs3-rmtsys     7009/tcp                        # remote cache manager service
afs3-rmtsys     7009/udp                        # remote cache manager service
wnn6            22273/tcp                       # wnn6
wnn6            22273/udp                       # wnn6
#
#=========================================================================
# The remaining port numbers are not as allocated by IANA.
#
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4, and are unofficial.  Sites running
# v4 should uncomment these and comment out the v5 entries above.
#
```

```
kerberos4       750/udp         kerberos-iv kdc # Kerberos (server) udp
kerberos4       750/tcp         kerberos-iv kdc # Kerberos (server) tcp
kerberos_master 751/udp                         # Kerberos authentication
kerberos_master 751/tcp                         # Kerberos authentication
passwd_server   752/udp                         # Kerberos passwd server
krb_prop        754/tcp                         # Kerberos slave propagation
krbupdate       760/tcp         kreg            # Kerberos registration
kpasswd         761/tcp         kpwd            # Kerberos "passwd"
kpop            1109/tcp                        # Pop with Kerberos
knetd           2053/tcp                        # Kerberos de-multiplexor
zephyr-srv      2102/udp                        # Zephyr server
zephyr-clt      2103/udp                        # Zephyr serv-hm connection
zephyr-hm       2104/udp                        # Zephyr hostmanager
eklogin         2105/tcp                        # Kerberos encrypted rlogin
#
# Unofficial but necessary (for NetBSD) services
#
supfilesrv      871/tcp                         # SUP server
supfiledbg      1127/tcp                        # SUP debugging
#
# Datagram Delivery Protocol services
#
rtmp            1/ddp                           # Routing Table Maintenance Protocol
nbp             2/ddp                           # Name Binding Protocol
echo            4/ddp                           # AppleTalk Echo Protocol
zip             6/ddp                           # Zone Information Protocol
#
# Services added for the Debian GNU/Linux distribution
#
linuxconf       98/tcp                          # LinuxConf
poppassd        106/tcp                         # Eudora
poppassd        106/udp                         # Eudora
ssmtp           465/tcp                         # SMTP over SSL
moira_db        775/tcp                         # Moira database
moira_update    777/tcp                         # Moira update protocol.
moira_ureg      779/udp                         # Moira user registration.
omirr           808/tcp         omirrd          # online mirror
omirr           808/udp         omirrd          # online mirror
swat            901/tcp                         # Samba configuration tool
skkserv         1178/tcp                        # skk jisho server port
rmtcfg          1236/tcp                        # Gracilis Packeten remote config server
xtel            1313/tcp                        # french minitel
support         1529/tcp                        # GNATS
cfinger         2003/tcp                        # GNU Finger
ninstall        2150/tcp                        # ninstall service
ninstall        2150/udp                        # ninstall service
afbackup        2988/tcp                        # Afbackup system
afbackup        2988/udp                        # Afbackup system
fax             4557/tcp                        # FAX transmission service        (old)
hylafax         4559/tcp                        # HylaFAX client-server protocol  (new)
noclog          5354/tcp                        # noclogd with TCP (nocol)
noclog          5354/udp                        # noclogd with UDP (nocol)
hostmon         5355/tcp                        # hostmon uses TCP (nocol)
hostmon         5355/udp                        # hostmon uses TCP (nocol)
```

```
postgres        5432/tcp                        # POSTGRES
postgres        5432/udp                        # POSTGRES
canna           5680/tcp                        # cannaserver
saned           6566/tcp                        # SANE network scanner daemon
ircd            6667/tcp                        # Internet Relay Chat
ircd            6667/udp                        # Internet Relay Chat
webcache        8080/tcp                        # WWW caching service
webcache        8080/udp                        # WWW caching service
tproxy          8081/tcp                        # Transparent Proxy
tproxy          8081/udp                        # Transparent Proxy
mandelspawn     9359/udp        mandelbrot      # network mandelbrot
amanda          10080/udp                       # amanda backup services
kamanda         10081/tcp                       # amanda backup services (Kerberos)
kamanda         10081/udp                       # amanda backup services (Kerberos)
amandaidx       10082/tcp                       # amanda backup services
amidxtape       10083/tcp                       # amanda backup services
isdnlog         20011/tcp                       # isdn logging system
isdnlog         20011/udp                       # isdn logging system
vboxd           20012/tcp                       # voice box system
vboxd           20012/udp                       # voice box system
binkp           24554/tcp                       # Binkley
binkp           24554/udp                       # Binkley
asp             27374/tcp                       # Address Search Protocol
asp             27374/udp                       # Address Search Protocol
tfido           60177/tcp                       # Ifmail
tfido           60177/udp                       # Ifmail
fido            60179/tcp                       # Ifmail
fido            60179/udp                       # Ifmail

# Local services
```

12 Appendix D: Question Answers

Question 1 Answer

A simple approach:

```
ipchains -A forward -p tcp -i ppp0 --dport www -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport www ! -y -j ACCEPT
ipchains -P forward DENY
echo 1 > /proc/sys/net/ipv4/ip_forward
```

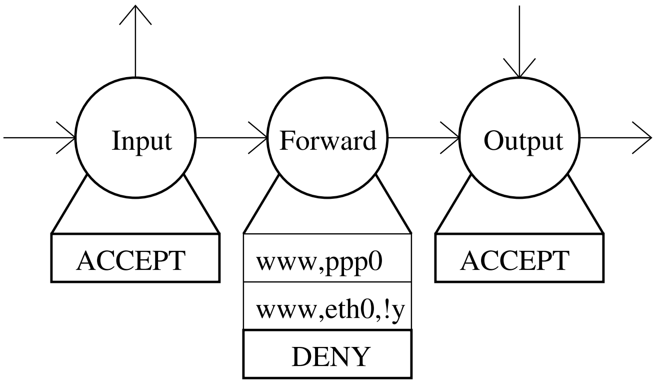


Figure 11: Question 1 Simplistic Answer

Problem with the above:

- ICMP is the message protocol of IP networks
  - Part of its role is to indicate the failure of other protocols
- Blocking all ICMP results in never receiving 'Destination Unreachable' errors
  - This can be irritating but rarely fatal
- Worse problem is in the role of ICMP in MTU discovery
  - All good TCP implementations use MTU discovery to try to figure the largest packet that can be sent to a destination without being fragmented
  - This involves sending a packets with the "Don't Fragment" bit set and then sending smaller packets if it receives an ICMP packet indicating "Fragment needed but DF set"
  - If all ICMP packets are blocked then MTU will remain too high, and no packets will get through.

The correct approach:

```
ipchains -A forward -p icmp --icmp-type destination-unreachable -j ACCEPT
ipchains -A forward -p tcp -i ppp0 --dport www -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport www ! -y -j ACCEPT
ipchains -P forward DENY
echo 1 > /proc/sys/net/ipv4/ip_forward
```

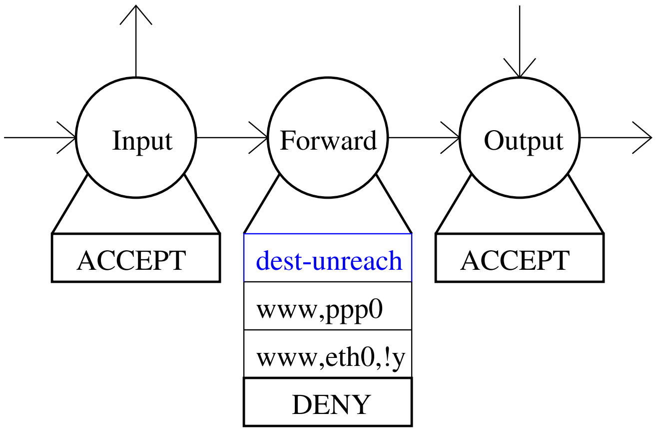


Figure 12: Question 1 Real Answer

Question 2 Answer

The simple approach:

```
# Allow ICMP errors
ipchains -A forward -p icmp --icmp-type destination-unreachable -j ACCEPT
ipchains -A forward -p tcp -i ppp0 --dport www -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport www ! -y -j ACCEPT
# Allow FTP connections
ipchains -A forward -p tcp -i ppp0 --dport ftp -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport ftp ! -y -j ACCEPT
# Allow FTP data connections
ipchains -A forward -p tcp -i ppp0 --dport 1024: -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport 1024: ! -y -j ACCEPT
# Allow DNS queries and responses
ipchains -A forward -p udp -i ppp0 --dport domain -j ACCEPT
ipchains -A forward -p udp -i eth0 --sport domain -j ACCEPT
# Everything else
ipchains -P forward DENY
```

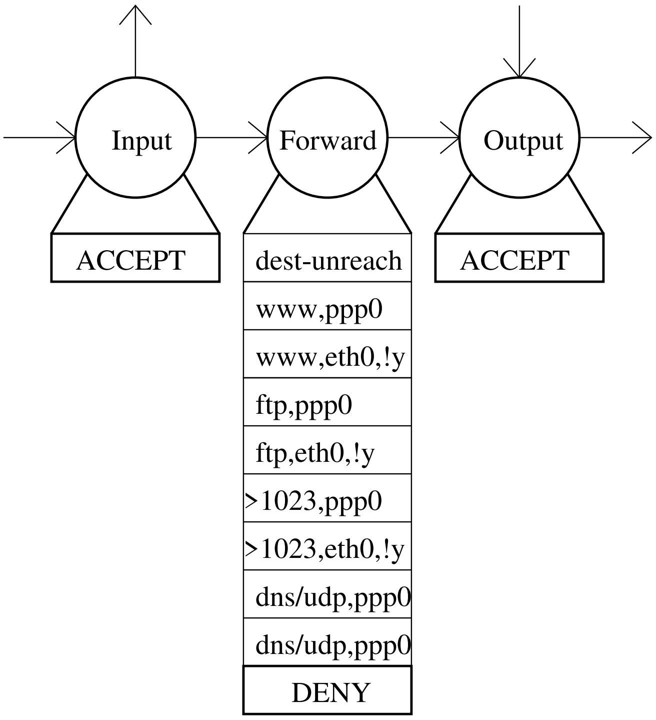


Figure 13: Question 2 Simplistic Answer

DNS has a problem: normally UDP is used for DNS lookups (port 53)...

- ...but when the server reply exceeds 512 bytes the client uses TCP instead

FTP has a problem: it has two modes of operation: Active and passive

- Active is normally used by command line FTP clients while Web browsers normally use passive mode.
- In active mode when the server wants to send a file (or even the result of an ls or dir command) the server tries to make a TCP connection back to the client
- This means that you cannot filter out all TCP connections if you want to use active FTP
- Passive mode makes the data connection from the client to the server in all cases
- If active mode FTP is required then allowing all TCP connections above port 1024 (excluding 6000 to 6010 used for X-Windows) is recommended

So the revised solution:

```
# Allow ICMP errors
ipchains -A forward -p icmp --icmp-type destination-unreachable -j ACCEPT
# WWW
ipchains -A forward -p tcp -i ppp0 --dport www -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport www ! -y -j ACCEPT
# Allow FTP connections
ipchains -A forward -p tcp -i ppp0 --dport ftp -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport ftp ! -y -j ACCEPT
# Allow PASSIVE FTP data connections
ipchains -A forward -p tcp -i ppp0 --dport 1024: -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport 1024: ! -y -j ACCEPT
# Allow ACTIVE FTP data connections
ipchains -A forward -p tcp -i eth0 --sport ftp-data --dport 1024:5999 -j ACCEPT
ipchains -A forward -p tcp -i eth0 --sport ftp-data --dport 6010: -j ACCEPT
ipchains -A forward -p tcp -i ppp0 --sport 1024:5999 --dport ftp-data ! -y -j ACCEPT
ipchains -A forward -p tcp -i ppp0 --sport 6010: --dport ftp-data ! -y -j ACCEPT
# Allow DNS queries and responses
ipchains -A forward -p udp -i ppp0 --dport domain -j ACCEPT
ipchains -A forward -p udp -i eth0 --sport domain -j ACCEPT
# Allow TCP DNS queries
ipchains -A forward -p tcp -i ppp0 --dport domain -j ACCEPT
ipchains -A forward -p tcp -i ppp0 --sport domain ! -y -j ACCEPT
# Everything else
ipchains -P forward DENY
```

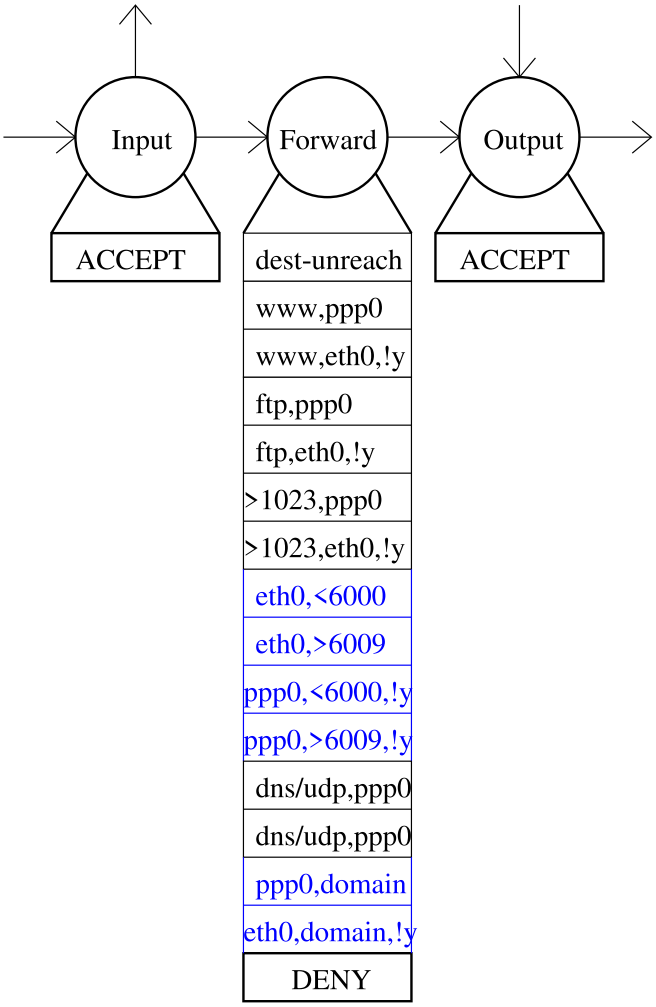


Figure 14: Question 2 Real Answer

13 References

RFC archive at http://mirror.aarnet.edu.au/ietf/rfc/

Where's the tutorial?
On the CDROM (if we got them cut in time).
http://www.rustcorp.com/talks/CALU-07-1999/

The definition of an IP Packet
RFC 791: INTERNET PROTOCOL

The definition of a UDP Packet
RFC 768: User Datagram Protocol

The definition of a TCP Packet
RFC 793: TRANSMISSION CONTROL PROTOCOL

The definition of an ICMP Packet
RFC 792: INTERNET CONTROL MESSAGE PROTOCOL

Compiling The Linux Kernel
The Linux Kernel HOWTO

The Linux Masquerading HOWTO
http://ipmasq.cjb.net/

Definitions of PORT_MASQ_BEGIN and PORT_MASQ_END
Linux Kernel source code: include/net/ip_masq.h

FTP site for SPF for 2.2 kernels:
ftp://ftp.interlinx.bc.ca/pub/spf