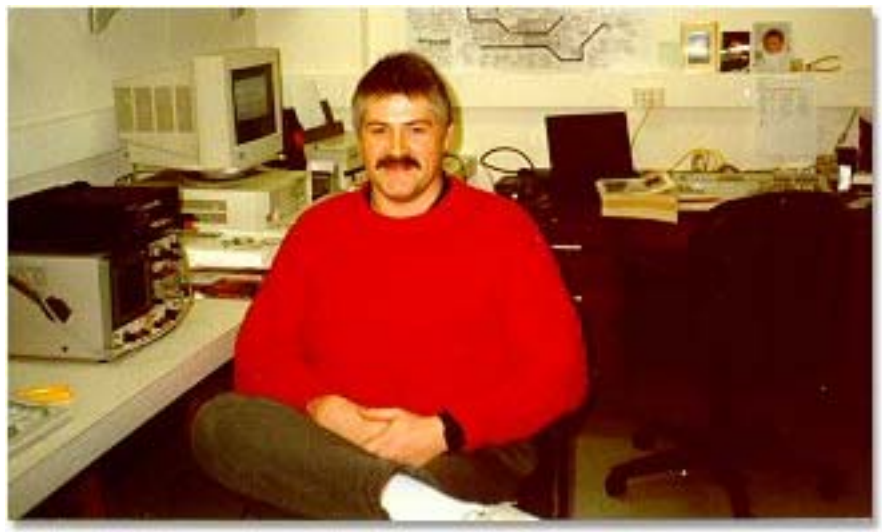
ipchains and iptables for Firewalling and Routing

Jeff Muday

Instructional Technology Consultant

Department of Biology, Wake Forest University

WAKE FOREST
UNIVERSITY



The ipchains utility

- Used to filter packets at the Kernel level
- Best firewall/routing utility for the 2.2 Kernel
- Replaces *ipfwadm* command from 2.0 Kernel
- Works with TCP/IP packets only
  - Protocols: TCP, ICMP, UDP
- Must understand TCP/IP thoroughly to take full advantage to the utility.
- Used in concert with *ipmasqadm* to provide masqueraded port forwarding.

Kernel Options

- CONFIG_FIREWALL
- CONFIG_IP_FIREWALL
- CONFIG_IP_MASQUERADE
- CONFIG_IP_MASQUERADE_ICMP
- CONFIG_IP_ALWAYS_DEFRAG
- CONFIG_IP_ADVANCED_ROUTER
- CONFIG_IP_ROUTE_VERBOSE
- CONFIG_IP_TRANSPARENT_PROXY

Problems…

- Not as flexible as iptables of Linux 2.4 Kernel
- Cannot resolve other network protocols
  - Such as:
    - AppleTalk
    - IPX/SPX
    - NetBeui

  Security Advice: probably should only run non TCP protocols on private internal network isolated by a separate NIC

Using ipchains

- 3 Basic chains:
  - Input

    Handles all inbound packets
  - Forward

    Handles packets destined for other computers
  - Output

    Handles packets processed and sent out by local machine

Using ipchains

- Each “chain” is a set of rules
- Rules are processed sequentially from first to last
- User can create define “custom” chains, which must be called by the user.
  - In practice this is rarely used…

Understanding the “rules”

- Each chain has a Default policy target of ACCEPT, DENY, REJECT, MASQ, REDIRECT, or RETURN
- The “standard” policy is ACCEPT
- What does this mean?
- That any packet that goes through the chain is interrogated by each “rule” and if the packet reaches the end of the chain (passed by all rules) the “default” policy is invoked on the packet

Targets defined

If a packet “matches” the rule it encounters it then has a target action:

- ACCEPT – it is allowed to be processed
- DENY – it is filtered out and forgotten
- REJECT – it is filtered out and responds with an ICMP message back to the sender of its rejection
- MASQ – the packet is allowed to be masqueraded for routing to internal or external networks
- REDIRECT – the packet can be redirected to another port or user defined chain
- RETURN – returns processing to the calling chain, much like a subroutine return

Syntax: ipchains

ipchains CMD [chain] [rule-spec|num] [options]

Commands:

-A (append) appends another rule to the end of the specified chain

-D (delete) deletes rule in a list. If n is not supplied, then the first rule is assumed.

-R (replace) replaces rule in a list.

-I (insert) inserts a rule at position n. If n is not supplied, then the rule is placed at the start of the chain.

Commands cont'd

-L (list) Lists all the rules in a particular chain. If *chainname* is omitted, it lists all rules in all chains.

-F (flush) Clears all rules in the named chain, or it *chainname* omitted, flushes all rules in all chains.

-N (create) creates a user specified chain

-X (delete user defined chain)

-P *policy* (set the default policy)

Common ipchains options

- -s [!] addr [[!] port] *specifies source address*
- -d [!] addr [[!] port] *specifies destination address*
- -p [!] {tcp | udp | icmp} *specifies protocol*
- -i [!] interface[+] *specify interface + means to use all interfaces matching a particular type*
- -j target *if a rule is matched, jump to a particular target action such as ACCEPT, DENY, REJECT, etc.*
- -l *log the packet by klogd, often into /var/log/messages*
- [!] –y *only match TCP packets that are "initiating a connection.*

Recall TCP/IP nomenclature

- IP specifications are in the:

  0.0.0.0/0 format

  Examples:

  192.168.1.0/24

  (match any 192.168.1.1 through 192.168.1.254)

  10.1.0.0/16

  (matches a class B network 10.1.0.1 to 10.1.255.254)

Really simple firewall script

```
#!/bin/sh
MYNET=“10.100.1.0/24”
LOCAL=“127.0.0.1/32”
# flush existing
/sbin/ipchains –F
# allow all traffic on my network to be processed
/sbin/ipchains –A input –s $MYNET –d 0/0 –j ACCEPT
# allow all loopback traffic
/sbin/ipchains –A input –s $LOCAL –d 0/0 –j ACCEPT
# deny everything else
/sbin/ipchains –A input –s 0/0 –d 0/0 –j DENY
```

Cookbook: allow DNS

DNS=“64.89.114.157”

/sbin/ipchains –A input –p TCP –s $DNS domain –j ACCEPT

/sbin/ipchains –A input –p UDP –s $DNS domain –j ACCEPT

Cookbook: allow user to browse Web and use FTP

```
# allow connections that are NOT connection
# initiation attempts
/sbin/ipchains -A INPUT -p TCP ! -y -j ACCEPT

/sbin/ipchains -A input -p TCP -s 0/0 ftp-data \
  -d $MYIP 1024:5999 -j ACCEPT
/sbin/ipchains -A input -p TCP -s 0/0 ftp-data \
  -d $MYIP 6255: -j ACCEPT
# now, block everything but the loopback
/sbin/ipchains -A input -i ! lo \
  -p ! ICMP -j DENY -l
```

Masquerading

- Kernel must be compiled to support it!
- Enable forwarding
  - Echo 1 > /proc/sys/net/ipv4/ip_forward
- Modify /etc/sysconfig/network
  - IP_FORWARD=yes
- Script:

```
#!/bin/sh
MYNET=192.168.1.0/24
/sbin/ipchains -A input -i eth1 -s $MYNET -j ACCEPT
/sbin/ipchains -A forward -s $MYNET -j MASQ
/sbin/ipchains -A forward -j DENY -l
```

Port forwarding

- It is fairly easy to forward a port BEHIND the firewall to the outside world. All you need is the *ipmasqadm* command
  - You can easily forward ssh, Web, telnet ports to a machine behind the firewall. Note: FTP requires insmod of a masquerade module

```
# /sbin/ipmasqadm portfw –a –p tcp \
#  –L $MYIP $MYWEBPORT \
#    –R $SERVERIP $SERVERWEBPORT
# example suppose your firewall internal IP is
# 192.168.1.1 and your private web server is
# 192.168.1.10
/sbin/ipmasqadm portfw –a –p tcp \
   -L 192.168.1.1 80 –R 192.168.1.10 80
```

ipchains: saving, restoring

- Typically, we will use ipchains rules issued from scripts. But, if we create an ad-hoc firewall by issuing commands interactively, ipchains suite has two commands of interest:
  - *ipchains-save* – outputs a stream to stdio of the current ruleset
  - *ipchains-restore* – reads stdin previous created by the ipchains-save command.

Using *ipchains-save* and *ipchains-restore*

- Example: saving current chains

```
/sbin/ipchains-save > /etc/ipchains.conf
```

- Example: restoring saved chains

```
/sbin/ipchains-restore < /etc/ipchains.conf
```

You might choose to use the restore in the /etc/rc.local initialization sequence.

Summary: ipchains

- Discussed the purpose of ipchains command
- Listed the kernel options required to make ipchains function
- Presented syntax of the command
- Wrote a simple firewall script
- Showed how masquerading can be established

iptables for Firewalling and Routing

- More advanced/flexible than *ipchains*
- Generally available on distributions which ship with the Linux 2.4 kernel
  - RedHat 7.1, 7.2; Mandrake 8.0, 8.1
  - SUSE 7.2, TurboLinux 7

The iptables utility

- Used to filter packets at the Kernel level
- Best firewall/routing utility for the 2.4 Kernel
- Works with TCP/IP packets only
  - Protocols: TCP, ICMP, UDP
- Must understand TCP/IP thoroughly to take full advantage to the utility.

Kernel/Network options

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_FTP
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_UNCLEAN
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_TARGET
- CONFIG_IP_NF_TARGET_REJECT
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_MASQUERADE
- CONFIG_IP_NF_TARGET REDIRECT
- CONFIG_IP_NF_TARGET_LOG

iptables

- Three major tables
  - Filter table, NAT table, Mangle table
- Each table has a series of *Rule chains*
- The Filter table is analgous to the *ipchains* command. Like *ipchains*, it has three chains:
  - INPUT, FORWARD, and OUTPUT
- *iptables* is better than *ipchains*:
  - it does not need auxillary commands to handle routing and port forwarding

iptables continued

- NAT table
  - Handles network address translation
  - Contains three chains:
    - PREROUTING, OUTPUT, and POSTROUTING
- Mangle table
  - Used for packet mangling
  - Has two chains:
    - PREROUTING and OUTPUT

Other similarities to ipchains

- All table chain rules are processed in a sequential manner. That means each chain is *order dependent*.
- User defined chains are allowed in each table. These user defined chains are analogous to subroutines in normal program code – they must be called, and possibly returned from.

Rules of the *iptables* game

- Each “rule” has a “target”, “commands”, and “options” like ipchains, but different syntactic structure

Common *iptables* targets

- ACCEPT – a match allows packet to pass
- DROP – silently drops packet (like DENY)
- REJECT – drops the packet, but sends an ICMP signal back to acknowledge rejection
- MASQUERADE – prepares a packet for masquerade, and can only be used in POSTROUTING chain of NAT table.
- SNAT – a packet's source address is prepared for use routing. Can only be used in the POSTROUTING chain of the NAT table

Common *iptables* targets cont'd

- DNAT – like SNAT, but used to prepare the destination address for routing. This is used only in the PREROUTING chain of the NAT table
- LOG – if a packet matches this rule, it triggers another packet which is sent to a logging target.
- User chain – a user defined sub chain
- RETURN – a target for the END of a user defined chain. Analogous to a RETURN statement in a program subroutine.

Syntax: *iptables*

iptables [-t table] CMD [chain] [rule-spec|num] [options]

Common iptables Commands

- -A (--append)
- -D (--delete)
- -R (--replace)
- -I (--insert)
- -L (--list)
- -F (--flush)

Common iptables commands cont'd

- -N (--new-chain)
- -X (--delete-chain)
- -E (--rename-chain)
- -P (--policy)
- -C (--check)

Common iptables options

-p [!] protocol

-s [!] addr[/mask] (source)

-d [!] addr[/mask] (destination)

--destination-port [!] port (destination port)

--icmp-type [!] type

-i [!] interface[+] (specify interface)

-o [!] interface[+] (specify outbound interface)

-j target (ACCEPT, REJECT, DROP, etc.)

[!] --syn (matches if TCP packets are initiating a connection)

Packet state match rules

- With iptables, you have the ability to establish rules regarding matches based on packet states

```
iptables –m state –state [!] [state1,state2…]
```

- State Module Matches
  - NEW – matches packets that initiate a new connection
  - ESTABLISHED – matches packets that belong to an established connection.
  - RELATED – matches packets related to another connection
  - INVALID – matches packets that are invalid or could not be fully resolved

ipchains converted to *iptables*

- If you already have an ipchains firewall, it is fairly easy to convert ipchains rules to iptables
- Example:

```
# allow all connections to my ssh server
/sbin/ipchains -A input -s 0/0 -d $MYIP -p tcp \
   --dport 22 -j ACCEPT
# CONVERTS to iptables
/sbin/iptables -t filter -A INPUT -s 0/0 -d $MYIP \
   -p tcp --destination-port 22 -j ACCEPT
```

Simple firewall

```
/sbin/iptables -t filter -A INPUT -m state \
--state ESTABLISHED, RELATED -j ACCEPT
/sbin/iptables -t filter -A INPUT -p udp \
   -s $DNS -source-port domain -j ACCEPT
```

Masquerading and NAT with iptables

- Enable forwarding

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Instruct kernel to drop packets that come in on mismatched interfaces

```
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter
```

- Rules

/sbin/iptables –t nat –A POSTROUTING –o $EXTIF –j MASQUERADE

/sbin/iptables –t nat –A POSTROUTING –o $EXTIF –j SNAT –to $MYIP

Port forwarding with iptables

- Syntax:

```
iptables –t nat –A PREROUTING [-p protocol] \
–d $MYIP –dport original_port \
–j DNAT –to destaddr:port
```

Example: enable port forwarding to our web server on the private network 192.168.1.10

```
/sbin/iptables –t nat –A PREROUTING \
  –p tcp –d $MYIP –dport http \
  –j DNAT –to 192.168.1.10:80
```

Summary: iptables

- iptables is a powerful packet filtering and routing tool
- Listed the kernel options required to make iptables function properly
- Studied the syntax and function of the iptables command
- Presented a how-to on simple port forwarding