

Linux Cluster Security

Neil Gorsuch

NCSA,

University of Illinois, Urbana, Illinois.

Abstract

Modern Linux clusters are under increasing security threats. This paper will discuss various aspects of cluster network security, including: firewalls, ssh, Kerberos, stateful and stateless packet filtering, and newer kernel security features. A program that generates ipchains/iptables packet filtering rule sets from a simple configuration file is discussed.

1 Introduction

Computer clusters are installed in a wide range of environments. At one extreme, clusters can be part of an organization's internal network that has a thorough set of security measures in place. At the other extreme, they can be part of an organization that has no border protection, no firewalls, and where every computer in the organization is exposed to the Internet at large. Various methods can be used to provide security for computer clusters. Effective computer security is usually accomplished through multiple layers of protection. Packet filtering provides one of the most effective security layers, considering the relative ease of setting up good packet filtering.

2 Security Models

Computer security can be divided into one of three models: loose, medium, and strict.

2.1 The loose security model

This type of security model has no firewalls, and every computer is exposed to the Internet at large. No packet filtering is done and any type of network traffic is allowed. This is the typical situation in university and research environments, and while it imposes no restrictions on network usage, it is dangerous for computer clusters to follow this model.

2.2 The medium security model

The middle of the road security model imposes reasonable security protections, but allows relatively free access to outside network resources. This is accomplished by shutting off outside access to all but a few network services, imposing a packet filtering firewall, and allowing all computers inside the firewall free access to outside resources, while hiding their existence from the outside. It is straightforward, using Linux, to set up this kind of situation. As applied to a cluster, this model means that the cluster will only be accessible through one or more head or login nodes.

2.3 The strict security model

The first security model is the strict model. This type of installation is usually found in large corporations and sensitive government installations. In this model, there is a very restrictive border firewall. This firewall does not forward any network packets through it. The only access from inside the firewall to the outside is through application proxies running on the firewall. This type of setup allows for a high level of security, but also restricts the usefulness of the network since new applications or protocols cannot be used until appropriate proxies are written and installed. For a computer cluster this model is inappropriate.

3 Cluster Network Topologies

To isolate the compute nodes of a cluster and to heighten cluster security, small to medium sized clusters should use a private subnet. One or more head or login nodes can have dual network interfaces to provide outside access to the cluster. This may not work in larger clusters for performance reasons. If the cluster nodes need high-speed access to file servers or other network resources that are outside the cluster private subnet, the head nodes may not be able to forward the traffic through themselves without unacceptable delays.

3.1 IP masquerading

When most of the nodes of a cluster are put on a private subnet, the hidden nodes should use private IP addresses that are not accessible outside the cluster. RFC 1918 reserves these ranges of private IP addresses: 10.x.x.x, 172.16-31.x.x, 192.168.x.x. If the hidden cluster nodes need to access network resources outside of the cluster, the head nodes that forward packets for them should also provide IP masquerading. With IP masquerading turned on, all packets being forwarded from a hidden cluster node to an address outside the cluster are altered so that they appear as if they came from the forwarding machine instead of from an un-routable private address. Packets coming back in as part of a masqueraded connection are properly translated and forwarded to the appropriate hidden cluster node.

4 Incoming Services

The most straightforward method of reducing network security breaches to any computer is to disable incoming network services. Any unused or unnecessary incoming network services should be disabled. Most of them can be disabled using the `linuxconf` program. A few of them need to have their entries commented out in the `/etc/inetd.conf` file.

4.1 Remote logins

One type of incoming network service that is usually needed is remote shell logins. For Unix computers there are three major network login services: `telnet`, `rlogin` and `ssh`.

4.1.1 telnet

Classic `telnet` should not be used because the username and passwords are transmitted in the clear with no encryption. Newer versions of `telnet` use Kerberos and if you can limit connections to Kerberos authenticated sessions, this will provide adequate security.

4.1.2 rlogin

The only way to have `rlogin` and the other “r-services” not send passwords in the clear over the network is to allow `.rhosts` files, which allow certain computers and usernames to login without them. This raises other security concerns and should be avoided. The short answer is, don’t use `rlogin`.

4.1.3 ssh

`ssh` provides completely encrypted connections, and is the remote login service of choice for security reasons. Its accompanying program, `scp`, allows for encrypted file transfers. The open source version of `ssh`, `openssh` has an excellent security track record. Another advantage of `ssh` is that it allows X window connections to be forwarded through `ssh` connections, avoiding many X window security concerns.

4.2 File Transfers

To transfer files in and out of a computer cluster, either `ftp` or `scp` is usually used.

4.2.1 FTP and TFTP

`Ftp` is one of the hardest programs to use through a firewall. Standard `FTP` sends login information including passwords in the clear over the network. Security holes are periodically found in `FTP` servers, so it is probably better to make people log into a cluster and then use an `FTP` client from within the cluster, rather than setting up an `FTP` server inside the cluster. `Ftp` operates in either active or passive mode.

In active mode, an `FTP` server receives commands on `TCP` port 21, but the client picks an unused `TCP` port from 1024 to 65535 on the client machine for the server to connect to. The server then connects back to the client machine on the

chosen port, originating from TCP port 20 on the server machine. Active mode FTP clients should only be allowed inside a firewall if the firewall can track active mode FTP transactions and selectively open incoming high ports based on the FTP conversations. This is a problem for most IP masquerading firewalls. For an FTP client to operate inside most firewalls, the firewall cannot block any incoming accesses to ports 1024 to 65535 from outside the firewall. For FTP servers inside a firewall, active mode works well, since the only connection coming from outside the firewall is on TCP port 21.

In passive mode, an FTP server receives commands on TCP port 21, and the server picks a port from 1024 to 65535 and tells the client the port number. The client then opens a connection to the server on the given port. This works well for FTP clients inside firewalls, since the connections are opened from inside the firewall to the outside. Passive mode FTP servers should only be allowed inside firewalls if the firewall can track passive mode FTP conversations and selectively open incoming high ports based on the FTP conversations. This is a problem for most firewalls. For an FTP server to operate inside most firewalls and be accessible from outside the firewall, the firewall cannot block any incoming accesses to ports 1024 to 65535 from outside the firewall.

Except in the case of some advanced stateful connection tracking firewalls, FTP clients inside the firewall should only operate in passive mode, and any FTP servers set up inside the firewall should only operate in active mode.

TFTP is simpler version of FTP that is a security nightmare. It should never be allowed to operate through a firewall.

4.2.2 scp file transfers

A better alternative to FTP for file transfers is the scp program, which is included with ssh. The entire conversation, including user names and passwords are sent encrypted over the network.

4.3 Remote windowing

Many of the applications that are run on a cluster require remote windowing access. In Unix, the X windows protocol is the standard. Direct connections to X servers should not be allowed through a firewall. Instead, X connections should be tunneled through existing ssh connections.

5 Security layers

As with any other system of protections, computer security works best when applied in overlapping layers.

5.1 Border router filtering

Border routers should be set to block all access to privileged ports from outside, except for specific ports on specific machines that are running servers.

5.2 Packet filtering

Packet filtering examines each network packet that travels through a computer, and based on various factors such as the source and destination addresses and port numbers, and a defined set of rules, either passes, blocks, or modifies each packet. Traditional packet filters only examine the headers of packets, and do not look at the packet data.

Linux kernels have had packet filtering since version 1.1 which used the `ipfw` command ported from BSD. Linux version 2.0 used the `ipfwadm` command to control packet filtering. Version 2.2 introduced many changes in the packet filtering code inside the kernel, and introduced the command `ipchains` to control packet filtering rule chains. Linux version 2.4 packet filtering is completely rewritten, uses the `iptables` command, and includes stateful connection tracking and filtering.

5.2.1 Stateless packet filtering

Stateless packet filtering examines each packet separately, and does not use any information about previous packets or connections when deciding the fate of a packet. Since many protocols such as FTP require a wide range of ports to be accessible, stateless packet filters cannot safely support many common protocols.

5.2.2 Stateful packet filtering

Stateful packet filtering keeps track of all open connections, and uses that information to help determine the validity of subsequent packets. This greatly increases security, especially in the case of protocols such as FTP. Unfortunately, only Linux kernel versions 2.4 and higher include connection tracking with stateful packet filtering support.

5.2.3 Source route verification

Source route verification should be turned on for every interface. This prevents packets from having their source address forged by automatically rejecting any packets that have a source address that would be impossible for the network interface that they came in on. This stops *people* from trying to bypass the packet filtering rules by forging packets that look like they came from the hidden network and injecting them into the outside interface. To turn on source route verification for a single interface, use the command:

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

where `INTERFACE` is the interface name such as `eth0`. To turn on source route verification for all present and future interfaces issue these commands:

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

Note these commands have to be issued after every powerup or reboot.

5.2.4 Martians

Packets that are dropped when source route verification is turned on are called “martians”. To automatically log all of these occurrences, issue the following commands:

```
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians
echo 1 > /proc/sys/net/ipv4/conf/default/log_martians
```

5.3 tcpwrappers filtering

Newer versions of Linux support tcpwrappers, which adds another layer of application level filtering. This is accomplished through inetd with the files /etc/hosts.deny and /etc/hosts.allow controlling access for each network service.

5.4 Application layer filtering

Most server applications can be set to block access to certain addresses, or to allow access only to certain addresses. This feature should be used wherever possible.

6 Linux packet filtering

Linux versions 2.2 and later use the concept of a “chain” for packet filtering. Each chain consists of one or more rules. Each rule has a set of matching conditions and an action to take if the packet is matched. Some chains are pre-defined and always exist; others can be defined or deleted by the user. Rules can be added to any chain, and any chain can have all its rules flushed from it. Packets can traverse more than one chain. Packets can be matched based on any or all of these attributes, some of which only apply to certain protocols:

- source IP address
- destination IP address
- protocol type such as TCP or UDP or ICMP
- network interface

TCP packets can also be matched on these attributes:

- source port
- destination port
- TCP flags

UDP packets can also be matched on these attributes:

- source port
- destination port

ICMP packets can also be matched on these attributes:

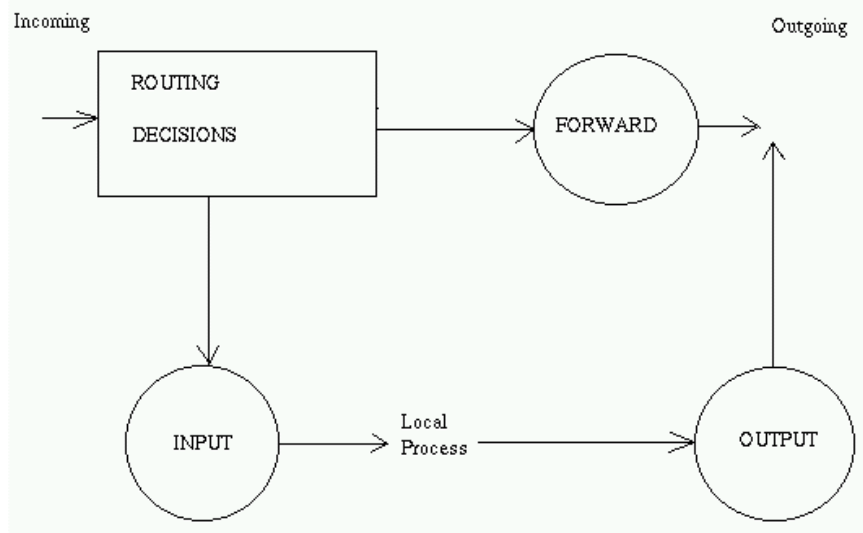
- ICMP type

6.1 Linux version 2.2 ipchains based packet filtering

The predefined chains are named “input”, “output”, and “forward”. All packets coming into the computer on any network interface traverse the “input” chain. All packets being forwarded from one network interface to another network interface traverse the “forward” chain. All packets going out to a network interface traverse the “output” chain.

6.2 Kernel version 2.4 packet filtering

Packet filtering in Linux kernel versions 2.4 and later is very different from previous versions of the kernel. Here is a diagram of the path packets travel:



When a packet comes in from a network interface, the kernel looks at the destination address and makes routing decisions. If it is destined to be forwarded out to another network interface, and the kernel has packet forwarding turned on, the packet traverses the “FORWARD” chain. If the packet passes through the “FORWARD” chain, it is sent out the appropriate network interface. If the packet is destined to be consumed within the host, it traverses the “INPUT” chain. If it passes through the “INPUT” chain, any processes waiting for that packet will receive it. Locally generated packets that are going out on a network interface traverse the “OUTPUT” chain. If it passes the “OUTPUT” chain it will be sent out the appropriate network interface.

Linux version 2.4 supports integrated connection tracking and stateful inspection. This adds more ways to match packets, based on whether they are new packets, packets that are part of an existing connection, or packets logically related to an existing connection. While greatly increasing security, this also vastly simplifies filtering rulesets.

Other added matching conditions are:

- MAC address
- limits to prevent log overflowing and some denial of service attacks
- user id for locally generated packets

7 pfilter a packet filtering ruleset generator

pfilter is a packet filtering set of software that reads an easy to understand input configuration file, and generates the appropriate ipchains or iptables commands for a complete packet filtering firewall. It can be used on both firewall machines that forward packets, and on machines that only have one network interface.

By default, any packets or connections not explicitly allowed by rules are blocked.

pfilter can be told to allow incoming connections to specified ports and/or services from specified hosts and/or address ranges

When pfilter is used on a packet forwarding firewall, it turns on IP masquerading by default. When masquerading is turned on, and pfilter is being used on a host with Linux version 2.4 or later, connections are tracked and any hosts hidden behind the firewall are allowed to open any connection to hosts outside the firewall. Hosts hidden behind the firewall can have certain ports or services on them be accessible from outside on separate IP addresses. Specified ports and/or services of hidden hosts can be redirected from ports in the firewall's IP address.

By default, incoming ICMP packets are blocked except for the following:

- echo replies so that the host or any hosts hidden behind it can use the ping command
- time exceeded packets so that the host or any hosts hidden behind can use the traceroute command
- destination unreachable and redirect packets so that normal error returns work

7.1 sample pfilter configuration file

The following file would be useful on the head or login nodes of a cluster. It assumes that there is a private subnet 192.168.1.* for the cluster nodes, and that a couple of the hidden cluster nodes also want to be accessible from outside the cluster for interactive logins. Explanations are interspersed as bullet items.

```
# sample configuration file for pfilter packet filtering system
#
# anything from the # character to end of a line is ignored
#
# separators can be any combination of whitespace (space or tab characters)
#
# version 1.00
```

```
# Service on this host that will be accessible from the outside.
# Each line consists of the following fields:
# the identifier word OPEN
# the protocol type (either tcp or udp)
# the port number of service name
# an optional source address
```

- These lines allow ssh and ftp access to the firewall machine. Ssh is allowed from anywhere, while ftp is only allowed from the same fictional domain IP range.

```
OPEN tcp ssh
OPEN tcp ftp 1.2.0.0/16
```

```
# Enable this host as an NFS server to our domain
# Each line consists of these fields:
# the identifier word OPENNFS
# a source address mask or list
```

```
#OPENNFS 1.2.0.0/16
```

```
# Protected hidden hosts that need to look like they are on
# the external network but only have a limited set of ports
# be accessible from the outside. Each accessible internal
# host will need an accessible external IP address, which this
# firewall machine will listen for. This will therefore not
# work on a PPP dialup connection. Each accessible host line
# has these fields:
```

```
# the identifier word ALIAS
# external IP address
# internal IP address
# services that the outside world will be able to access.
# The services can either be tcp port numbers or tcp service names.
```

- These next two lines cause pseudo IP addresses to be created on the outside of the firewall, and specific services (in this case ssh) to appear as if they are active on the pseudo IP addresses. Accessing ssh on the pseudo IP addresses actually tunnels through and accesses two of the hosts hidden behind the firewall. Specifically, when someone tries to access ssh on the address 1.2.20.160, they will actually be accessing ssh on the internal address 192.168.1.2, and so on.

```
ALIAS 1.2.20.160 192.168.1.2 tcp ssh
ALIAS 1.2.20.127 192.168.1.3 tcp ssh
```

```
# Protected internal NIC interface. This is normally not set since
# the rc.firewall script will figure out which one it is as long
# as RFC 1918 IP addresses are used for the internal network.
```

*# (RFC 1918 addresses are IP addresses reserved for local private
networks only they range from 192.168.1.0-192.168.1.255 and
from 10.0.0.0-10.0.0.255). If you have to specify the NIC interface
of the internal network, uncomment the following line and
set the name appropriately.*

#INTIF eth0

*# External NIC interface. This is normally not set since
the rc.firewall script will figure out which one it is as long
as RFC 1918 IP addresses are used for the internal network.
If you have to specify the NIC interface of the external network,
uncomment the following line and set the name appropriately.*

#EXTIF ppp0

*# Uncomment the next line for more verbose output.
(or use the -v or --version command line switches)*

#VERBOSE

*# Uncomment the next line for lots of script debug output
(or use the -d or --debug command line switches)*

#DEBUG

7.2 pfilter sample configuration script generated iptables commands

When pfilter is run with the above sample script on a Linux system with a version 2.4 kernel, the following commands are generated, which get executed whenever the /etc/rc.d/init.d/pfilter script is run with an argument of start or restart. Explanations are included as bullet items for each block of commands.

- First the modprobe lines make sure that all needed packet filtering modules are loaded in.

```
modprobe ip_conntrack
modprobe ip_conntrack_ftp
modprobe ip_conntrack_irc
modprobe ip_conntrack_rpc_tcp
modprobe ip_conntrack_rpc_udp
modprobe ip_nat_ftp
modprobe ip_nat_irc
modprobe ip_queue
modprobe ip_tables
modprobe ipt_LOG
modprobe ipt_MARK
modprobe ipt_MASQUERADE
modprobe ipt_MIRROR
modprobe ipt_REDIRECT
modprobe ipt_REJECT
```

```
modprobe ipt_TOS
modprobe ipt_limit
modprobe ipt_mac
modprobe ipt_mark
modprobe ipt_multiport
modprobe ipt_owner
modprobe ipt_record_rpc
modprobe ipt_state
modprobe ipt_tos
modprobe ipt_unclean
modprobe iptable_drop
modprobe iptable_filter
modprobe iptable_mangle
modprobe iptable_nat
```

- Then route source verification is turned on for all present and future interfaces.

```
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter
echo 1 > /proc/sys/net/ipv4/conf/default/rp_filter
echo 1 > /proc/sys/net/ipv4/conf/eth0/rp_filter
echo 1 > /proc/sys/net/ipv4/conf/eth1/rp_filter
echo 1 > /proc/sys/net/ipv4/conf/lo/rp_filter
echo 0 > /proc/sys/net/ipv4/conf/all/bootp_relay
```

- Bootp request packet forwarding is turned off to prevent DHCP servers on both networks from interfering with each other.

```
echo 0 > /proc/sys/net/ipv4/conf/default/bootp_relay
echo 0 > /proc/sys/net/ipv4/conf/eth0/bootp_relay
echo 0 > /proc/sys/net/ipv4/conf/eth1/bootp_relay
echo 0 > /proc/sys/net/ipv4/conf/lo/bootp_relay
```

- Martians packet logging is turn on.

```
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians
echo 1 > /proc/sys/net/ipv4/conf/default/log_martians
echo 1 > /proc/sys/net/ipv4/conf/eth0/log_martians
echo 1 > /proc/sys/net/ipv4/conf/eth1/log_martians
echo 1 > /proc/sys/net/ipv4/conf/lo/log_martians
```

- All packets from the INPUT and OUTPUT built-in chains are redirected to a new chain called “protect” later in the commands. This simplifies the packet matching rules and keeps them from being duplicated in three places. Before we have the packets jump to the new chain, we have to create it with the `iptables -v -N protect` command. The `iptables -v -F protect` command flushes the protect chain of any rules in case the chain was already created from some previous packet filtering.

```
/sbin/iptables -v -N protect 2>/dev/null
/sbin/iptables -v -F protect
```

- The next two commands cause packets in the built-in INPUT and OUTPUT chains to be redirected into the new “protect” chain. Note that packets in the pre-defined OUTPUT chain were generated on this host and do not need to be examined.

```
/sbin/iptables -v -A INPUT -j protect
/sbin/iptables -v -A FORWARD -j protect
```

- The command allows all packets coming in from the external interface that are part of current connections to be passed in, without having to allow every high port to be left open. This single command line is the most important one for adding to the effectiveness of the system’s network security, and is the major reason to upgrade to Linux kernel version 2.4 as far as security goes.

```
/sbin/iptables -v -A protect -m state --state ESTABLISHED,RELATED -j \
ACCEPT
```

- The next command allows any host inside the firewall to access any service on the outside of the firewall or on the firewall itself.

```
/sbin/iptables -v -A protect -i eth1 -m state --state NEW -j ACCEPT
```

- The next two commands allow ssh access to the firewall host from outside and from itself. Note that ssh access to the firewall host is guaranteed because of the line above that allowed hosts behind the firewall to access the outside and the firewall machine.

```
/sbin/iptables -v -A protect -m state --state NEW -d 1.2.20.209 -p tcp \
--destination-port ssh -j ACCEPT
```

```
/sbin/iptables -v -A protect -m state --state NEW -s 1.2.0.0/16 -d 1.2.20.209 \
-p tcp --destination-port ftp -j ACCEPT
```

- The next two commands allow ftp access to the firewall host from inside the host’s domain and from the firewall host itself on it’s loopback interface. Note that kernel version 2.4.4 and later handle both active and passive mode ftp connection tracking without having to leave high ports open when not in use.

```
/sbin/iptables -v -A protect -m state --state NEW -s 192.168.1.0/8 -d \
1.2.20.209 -p tcp --destination-port ftp -j ACCEPT
```

```
/sbin/iptables -v -A protect -m state --state NEW -s 127.0.0.1 -d 127.0.0.1 -p \
tcp --destination-port ftp -j ACCEPT
```

- The next two commands allow ssh access to a couple of machines behind the firewall through pseudo IP interfaces that will be created below.

```
/sbin/iptables -v -A protect -m state --state NEW -d 192.168.1.3 -p tcp \
--destination-port ssh -j ACCEPT
/sbin/iptables -v -A protect -m state --state NEW -d 192.168.1.2 -p tcp \
--destination-port ssh -j ACCEPT
```

- The next five commands block broadcast packets from being transferred through the firewall.

```
/sbin/iptables -v -A protect -i eth0 -d 255.255.255.255 -j DROP
/sbin/iptables -v -A protect -i eth0 -d 141.255.255.255 -j DROP
/sbin/iptables -v -A protect -i eth0 -d 1.2.255.255 -j DROP
/sbin/iptables -v -A protect -i eth0 -d 1.2.20.255 -j DROP
/sbin/iptables -v -A protect -i eth0 -s 255.255.255.255 -j DROP
```

- This block of commands logs packets that are about to be dropped.

```
/sbin/iptables -v -A protect -i eth0 -d 1.2.20.209 -j LOG --log-prefix "Bad \
packet from outside:"
/sbin/iptables -v -A protect ! -i eth0 -d 1.2.20.209 -j LOG --log-prefix "Bad \
packet:"
/sbin/iptables -v -A protect -i eth0 -d 1.2.20.127 -j LOG --log-prefix "Bad \
outside to protected:"
/sbin/iptables -v -A protect ! -i eth0 -d 1.2.20.127 -j LOG --log-prefix "Bad \
packet to protected:"
/sbin/iptables -v -A protect -i eth0 -d 1.2.20.160 -j LOG --log-prefix "Bad \
outside to protected:"
/sbin/iptables -v -A protect ! -i eth0 -d 1.2.20.160 -j LOG --log-prefix "Bad \
packet to protected:"
/sbin/iptables -v -A protect -i eth0 -j LOG --log-prefix "Bad packet from
outside:"
/sbin/iptables -v -A protect -i eth1 -j LOG --log-prefix "Bad packet from inside:"
/sbin/iptables -v -A protect -i lo -j LOG --log-prefix "Bad packet from
loopback:"
```

- The next line adds the last rule in the new chain, which causes packets that don't match any other rules in that chain to be dropped.

```
/sbin/iptables -v -A protect -j DROP
```

- This command takes care of all IP masquerading of hosts behind the firewall, allowing them to access the outside without revealing their IP addresses.

```
/sbin/iptables -v -t nat -A PREROUTING -d 1.2.20.127 -j DNAT --to 192.168.1.3
```

- These lines add two pseudo IP addresses on the outside of the firewall so that a couple of the hidden hosts can be accessed from the outside. Note that only the specific accesses defined above are allowed. The hidden hosts cannot be pinged, tracerouted, or seen in any other way, no matter how many ports they actually have open.

```
/sbin/ifconfig eth0:1 1.2.20.127
/sbin/route add -host 1.2.20.127 dev eth0:1
/sbin/iptables -v -t nat -A PREROUTING -d 1.2.20.160 -j DNAT --to 192.168.1.2
/sbin/ifconfig eth0:2 1.2.20.160
/sbin/route add -host 1.2.20.160 dev eth0:2
/sbin/iptables -v -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.20.209
```

- Finally we turn on packet forwarding.

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

With this setup, the firewall host and the hidden hosts behind it's firewall, can access the outside in any way, including pings and traceroutes. But port scans on the firewall host reveal only the specific services that were opened up.

7.3 pfilter invocation

The pfilter software package installs the following files:

- `/etc/pfilter.conf` – This is the default configuration file that determines what packet filtering rules are set up. Any modifications should be done to this file.
- `/etc/rc.d/initd/pfilter` – This is a standard `chkconfig` style daemon/service start/stop/restart shell script. The command `chkconfig pfilter on` is automatically run when pfilter is installed to enable pfilter. This script is also responsible for deleting any added rules and removing any added pseudo IP interfaces when pfilter is stopped.
- `/etc/rc.d/pfilter.pl` – This is a perl script that translates the `pfilter.conf` configuration file into a packet filtering shell script which is left in the location `/etc/rc.d/pfilter.cmds`. This happens every time the pfilter is started or restarted.

Conclusion

In conclusion, the first method of network security to be installed should be packet filtering. This will block access from outside the cluster to all but needed services while allowing the cluster to access the outside freely. With tools such as pfilter, it is straightforward to generate ipchains/iptables rule sets. If possible, all but a few of the cluster nodes should be on a private subnet with IP masquerading turned on

Bibliography

Sonnenreich, Wes & Yates, Tom. *Building Linux and OpenBSD Firewalls*,
Wiley Computer Publishing: New York, 2000.

Andreasson, Oskar, *Boingworld IPTables Tutorial*,
<http://www.boingworld.com/workshops/linux/iptables-tutorial>.

Russell, Rusty, *Linux 2.4 Packet Filtering HOWTO*,
<http://netfilter.samba.org/netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/packet-filtering-HOWTO>.

pfilter, <http://pfilter.sourceforge.net/> .