ExperienceReport

Setting up a High Availability Firewall on Linux for iSeries

Brett Leeser

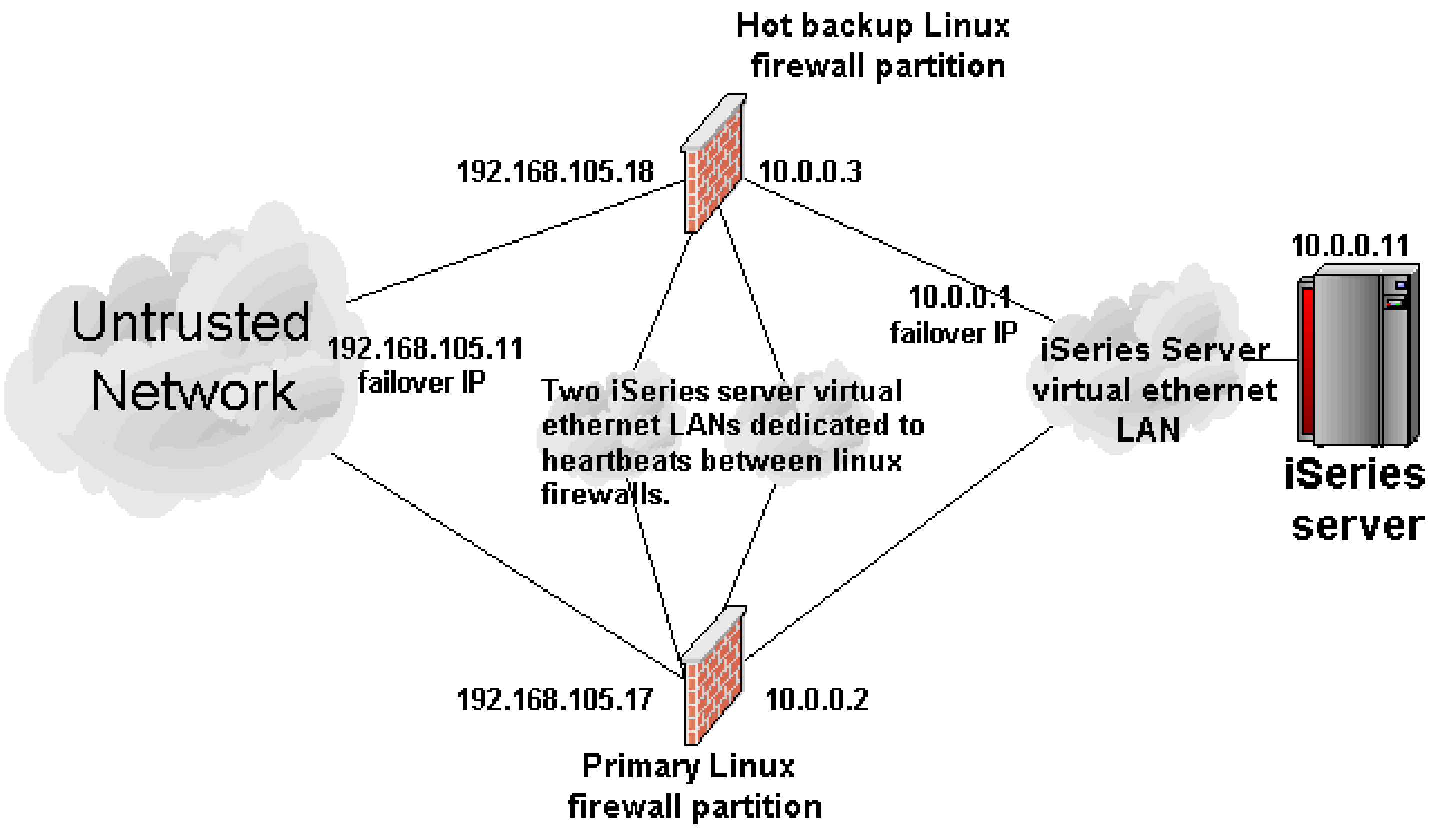
October 2002

Table of Contents

Setting up a high availability firewall on Linux for iSeries(R)

Introduction

Linux running on a PC has long been recognized as a cheap and secure firewall solution for home networks. Now that Linux can be run on an iSeries server, there is a great opportunity for iSeries customers to use a Linux firewall for their business network. But there are two very important considerations when considering adding a Linux firewall on an iSeries server for your business: reliability and cost. This scenario will show how to setup a Linux firewall on an iSeries server that is cheap, because all of the software is completely free, and reliable due to an automatic fail over within seconds. The following diagram shows the scenario setup.


Hot backup Linux firewall partition
192.168.105.18
10.0.0.3
10.0.0.11
10.0.0.1 failover IP
Untrusted Network
192.168.105.11 failover IP
iSeries Server virtual ethernet LAN
Two iSeries server virtual ethernet LANs dedicated to heartbeats between linux firewalls.
iSeries server
192.168.105.17
10.0.0.2
Primary Linux firewall partition

Heartbeat introduction

For businesses, an outage of their network can reflect badly on the company image and affect employee productivity. For example, customers could become frustrated while trying to access a web site and could potentially go to a competitor or employees could be prevented from receiving important emails. High availability is a must for businesses.

Linux provides firewall functionality natively in the kernel. But the Linux kernel does not provide high availability functionality for the firewall. That is where the open source community comes to the rescue. A quick search on the Internet for high availability functionality on Linux will yield the High-Availability Linux Project. The High-Availability Linux Project has developed the heartbeat application. This application is open source and free to use. Heartbeat provides high availability with the primary/backup model and handles the following functions: starting and stopping of needed services, monitoring of nodes in the cluster, and IP address take over. Heartbeat works with any Linux service, so to make the firewall highly available one only needs to write a service script for the Linux firewall. This step will be discussed under "Creating a firewall service" section.

Heartbeat installation and setup

The first step is to install and configure two Linux partitions on your iSeries server. If you do not already have two Linux partitions configured and installed on your iSeries server, consult the IBM$^{(R)}$ redbook "Linux on the IBM @server iSeries Server: An Implementation Guide" (SG24-6232) , for detailed instructions. You can view the redbook online at the IBM Redbooks$^{(TM)}$ web site.

Next, you need to compile and install the heartbeat application on each Linux partition. Heartbeat should compile and install on any Linux distribution, provided you have the needed compilers to compile the source code. This scenario used the SuSE Linux Enterprise Server(SLES) 7 for iSeries server distribution. First, download the heartbeat source code from the High-Availability Linux Project web site. The version downloaded for this scenario was named *heartbeat.4.9.1-tar.gz*. The *tar* and *gz* extensions mean the package was made with the Linux tar utility and compressed with the gzip utility. To extract the files from the package, issue the command `tar -zxvf heartbeat.4.9.1.tar.gz`. This will create a directory named *heartbeat.4.9.1*.

Change current directory to be the new heartbeat folder by using the command `cd heartbeat.4.9.1`. Then compile and install heartbeat with one easy command, `make install`. In the /etc/init.d directory there will now be a script called *heartbeat*. This script should be used to start, stop, restart, or check the status of heartbeat. The syntax of the script is as follows:

```
heartbeat {start|stop|status|restart}
```

But before heartbeat can be started, three configuration files must be created and placed in the /etc/ha.d directory. The three configuration files are *ha.cf*, *haresources*, and *authkeys*. Each of these configuration files are discussed in more detail below.

As a starting point, there are samples for each of these configuration files in the doc directory where you untarred the heartbeat source. You can copy each of these three files to the /etc/ha.d directory with these three commands:

```
cp /path-to-heartbeat.4.9.1-folder/doc/ha.cf /etc/ha.d/
cp /path-to-heartbeat.4.9.1-folder/doc/haresources /etc/ha.d/
cp /path-to-heartbeat.4.9.1-folder/doc/authkeys /etc/ha.d/
```

Once you have the configuration files copied, edit the configuration in each file to apply to your setup. Make sure the configuration files are the same for both Linux partitions. The configurations used in this scenario are discussed in the following sections.

ha.cf

The first configuration file is *ha.cf* and includes general configuration information for the heartbeat application. The following are the configuration lines in *ha.cf* used for this scenario:

`keepalive 1` => This parameter specifies how many seconds between heartbeats. In this case, there would be a heartbeat once every second.

`deadtime 10` => This parameter specifies how long to wait until declaring a host is dead. Specifying 10 indicates heartbeat will initiate a failover 10 seconds after a node stopped responding.

`initdead 60` => This parameter specifies how long deadtime should be the first time heartbeat starts up. This is important to allow time for the network to come up if heartbeat is started on boot up. The initdead time should be at least twice as long as the deadtime parameter.

`udpport 694` => This parameter specifies what port to use when sending out UDP heartbeat packets.

`udp eth2, eth3` => This parameter specifies what interface(s) to send the heartbeat out on. In this scenario, two iSeries server virtual ethernet LANs were totally dedicated to heartbeat. This helped keep the load off of the actual network, provided very high speed communication between nodes, and provided extremely high reliability as far as the network was concerned.

`node linux1`
`node linux2` => The node parameter tells heartbeat what systems are involved in the cluster. It is very important

that the node name is correct. To make sure the node name is correct, use the command *uname -n* on your Linux partition and use the EXACT name as shown from uname.

haresources
The second configuration file is *haresources*. This file specifies what resources are a part of the cluster and which node is the primary. The *haresources* file for this scenario contains just one line:

```
linux1 10.0.0.1 192.168.105.11 HA_firewall
```

The syntax for this line in the *haresources* file is *node-name resource1 resource2...resourceN*. In this scenario, the node name is *linux1* and this specifies that the linux1 node is the primary node. As long as linux1 is active it will have control over any of the resources specified in this configuration file. The parameters *10.0.0.1*, *192.168.105.11*, and *HA_firewall* indicate that the IP addresses 10.0.0.1 and 192.168.105.11 and the script HA_firewall are resources of the cluster.

When an IP address is specified as a cluster resource, heartbeat provides IP address takeover between nodes. Any IP address specified in haresources must not be configured as an IP address of any adapter on either node. When the heartbeat application is started, it will assign the failover IP address(es) to the appropriate node. This scenario uses two failover IP addresses so that the firewall always appears to have the same IP address to the iSeries server and to the untrusted network, regardless of which node is in control of the resources.

authkeys
The final heartbeat configuration file is *authkeys*. This configuration file is used to determine what type of authentication will be used between nodes and what key will be used for authentication. According to the High-Availability Linux Project website, there are three authentication algorithms available crc, md5, and sha1. The authentication algorithm you use depends on the amount of security you desire and the amount of processing power available. For the least security and processing power, use crc. For slightly more processing power and security use md5. Finally, for the most processing intensive and the most secure use sha1. The format for the authkeys file is:

auth <number>

<number> <authentication method> [<authkey>]

In this scenario, the heartbeat is broadcast over an iSeries server virtual ethernet LAN. The iSeries server virtual ethernet LAN is dedicated to the two Linux partitions and for this reason should be secure enough to require nothing better then crc for authentication. But in this scenario md5 is used to be on the safe side:

```
auth 1
1 md5 THIS_IS_MY_MD5_KEY
```

Obviously, you will want to specify something different for a key for security reasons. To use a different algorithm, just specify sha1 instead of md5. For the crc algorithm, no key is needed so just specify crc instead of md5 and remove the key portion.

One last important note about this configuration file is heartbeat requires the *authkeys* file to have strict permissions because it is important for security. To make sure *authkeys* has the right permissions, as the root user, issue the command `chmod 600 authkeys`. This sets the permissions on authkeys so that it is only readable by the root user.

Creating a firewall service

The *haresources* configuration file specifies a Linux service called *HA_firewall* as a cluster resource, so now this service must be created on both Linux partitions. Most Linux distributions put Linux service scripts in the /etc/init.d directory. However, the heartbeat application does not look in the /etc/init.d directory for services, so to make sure the *HA_firewall* script can be found by heartbeat it must be placed in the /etc/ha.d/resource.d directory. For this scenario, the *HA_firewall* service script is kept in the /etc/init.d directory to be consistent with all Linux services. Then a symbolic link to the *HA_firewall* script is placed in the /etc/ha.d/resource.d directory. Once the *HA_firewall* script is created, create a symbolic link for the service with the command, *ln -s /etc/init.d/HA_firewall /etc/ha.d/resource.d/HA_firewall*. Now there is a symbolic link to the *HA_firewall* script in the directory where heartbeat can find it. The symbolic link is really just a pointer to the actual script file.

If writing a Linux service script for the first time, look at other scripts in /etc/init.d directory for examples of how to do it. The firewall service script used for this scenario on both Linux partitions is as follows:

Code disclaimer information

```
# script to start the HA firewall
#

# include the /etc/rc.config file
. /etc/rc.config

#
# The echo return value for success (defined in /etc/rc.config).
#
 return=$rc_done

case "$1" in
  start)
      /root/start_firewall
      rc_status -v
      ;;
  stop)
      rc_status -v
      /root/end_firewall
      ;;
  restart|reload)
      $0 stop ${2+"$2"} && $0 start ${2+"$2"}
      rc_status
     ;;
  status)
      iptables -n -L | grep 'Chain INPUT' > /tmp/output
      read OUTPUT < /tmp/output
      rm -f /tmp/output
      case "$OUTPUT" in
```

```
30                "Chain INPUT (policy DROP)")
31                        echo "Firewall running."
32                        rc_status -v
33                  ;;
34              *)
35                      echo "Firewall not running."
36                      exit 0
37      esac
38      ;;
39  *)
40  echo "Usage: $0 {start|stop|status)}"
41  exit 1
42  esac
```

The purpose of this script is to provide heartbeat a way to start, stop, restart, and check the status of the firewall. The first 10 lines are comments and general setup for a Linux service. Line 12 is a case statement that checks the first parameter of the script (referred to as $1) for start, stop, restart or reload, status, or anything else. If the parameter is not one of the expected parameters, the case statement will print out the usage of the script. If the parameter to the script is start, the *HA_firewall* script starts the firewall by calling the *start_firewall* script. To end the firewall, it calls the *end_firewall* script. The *HA_firewall* script checks the status of the firewall by listing all firewall rules and checking for a policy of *DROP* on the input chain. If the default policy is *DROP*, the script assumes the firewall is running. The reason for this is the *end_firewall* script, discussed in the following section, clears all the rules of the firewall and sets the default policy for the input chain to *ACCEPT*. The change of the input policy to *ACCEPT* is needed to prevent all packets from being dropped once all the firewall rules are cleared.

Creating firewall start and stop scripts

The *HA_firewall* script calls two scripts, /root/start_firewall and /root/end_firewall, to start and stop the firewall. Creating these scripts is not difficult but does require some knowledge of iptables. Linux uses the command *iptables* to configure a Linux firewall. To get more information on iptables, check out the iptables web site or use the command `man iptables` in Linux.

The start_firewall script starts the firewall by loading needed modules into the kernel, enabling IP forwarding, and loading all the rules of the firewall into the kernel. The end_firewall script ends the firewall by removing all firewall rules except one from the kernel. The one remaining rule drops all traffic to and from the untrusted interface. All traffic must be dropped to the untrusted interface to ensure the Linux partition has no access to the untrusted network when no firewall is running. This step ensures the hot backup will not have any access to the untrusted network until it takes over for the primary node. At boot time, this same security hole exists for both the primary node and the backup node. For this reason, the end_firewall script should be run automatically at boot time. Consult documentation for your particular Linux distribution for configuring the end_firewall script to run at boot time.

Both the start_firewall and the end_firewall scripts should exist on each Linux partition and the start_firewall script should contain the exact same firewall rules on both Linux partitions. To help make sure this is the case, simply create the start_firewall script on one Linux partition, copy it to the other Linux partition, and then edit only the variables that are different on the second system, such as IP address.

The start and end scripts used in this scenario are included in the following sections for reference only. This experience report does not provide comprehensive information on network security or firewall configuration and does not recommend one security design alternative over another. The only intention of the scripts that follow is to show example firewall rules using iptables and to serve as a starting point only.

Code disclaimer information

end_firewall

```
#!/bin/sh

UNSECURE_IFC="eth0"

echo "Ending the firewall..."
#*******************************************************
# This script will clear out iptables and disable all
# traffic on the unsecure interface but still allow
# traffic on the secure interface for remote configuration
# when no firewall is running.
#*******************************************************

#*******************************************************
# Reset the default policies on the INPUT, FORWARD, and
# OUTPUT chains
#*******************************************************
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT

#*****************************************************
# Flush all the rules in the filter and nat tables
#*****************************************************
iptables -F
iptables -t nat -F

#*****************************************************
# Erase all chains that are not the default chains
# (INPUT, OUTPUT, and FORWARD)
#*****************************************************
iptables -X
iptables -t nat -X

#*****************************************************
# Drop all traffic coming in or going out the
# unsecure interface
#*****************************************************
iptables -A INPUT -i $UNSECURE_IFC -j DROP
iptables -A OUTPUT -o $UNSECURE_IFC -j DROP

echo "Done."
```

Code disclaimer information

start_firewall

```
#!/bin/sh

echo "Starting firewall..."
#*****************************************************
# Define variables, including the secure interface (inside interface)
# and the unsecure interface (outside interface).
#*****************************************************
INSIDE="eth1"
OUTSIDE="eth0"
HEARTBEAT_IFC1="eth2"
HEARTBEAT_IFC2="eth3"
LOOPBACK="lo"
#An IP address that will be seen by clients
# on the untrusted network and can be used to
# access to the iSeries server.
PUBLIC_IP="192.168.105.11"
#The iSeries server private IP address
PRIVATE_IP="10.0.0.11"
```

```
#*******************************************************
# Remove the ipchains module (if it is in the kernel) since it causes
# problems with the iptables module.
#*******************************************************
rmmod ipchains

#*******************************************************
# Insert needed modules for iptables and connection-tracking
# Note:  This is not needed if if the modules are built into
# the kernel.
# Note:  The modules are located in
# /lib/modules/2.4.1/kernel/net/ipv4/netfilter
#  replace the '2.4.1' with whatever kernel version you have
#*******************************************************
echo "Loading the needed modules..."
# for general iptables
insmod ip_tables
# for general iptables rules
insmod iptable_filter
# for connection tracking
insmod ip_conntrack
# for connection tracking of FTP
insmod ip_conntrack_ftp
# for doing SNAT and DNAT
insmod iptable_nat
# for logging
insmod ipt_LOG

#*******************************************************
# Enable IP forwarding
#*******************************************************
echo "Enabling IP forwarding..."
echo "1" > /proc/sys/net/ipv4/ip_forward

#*******************************************************
# Flush all rules from the kernel in case there are
# firewall rules already loaded.
#*******************************************************
iptables -F
iptables -t nat -F

#*******************************************************
# Set default policies for the INPUT, FORWARD, and OUTPUT chains
#*******************************************************
echo "Loading all the rules..."
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

#*******************************************************
# Allow all traffic to and from loopback device
#*******************************************************
iptables -A INPUT -i $LOOPBACK -j ACCEPT
iptables -A OUTPUT -o $LOOPBACK -j ACCEPT

#*******************************************************
# Allow all UDP traffic on the two VLANs
# because that is what the hearbeat packets use for
# the heartbeat.
#*******************************************************
```

```
iptables -A INPUT -i $HEARTBEAT_IFC1 -p UDP -j ACCEPT
iptables -A OUTPUT -o $HEARTBEAT_IFC1 -p UDP -j ACCEPT
iptables -A INPUT -i $HEARTBEAT_IFC2 -p UDP -j ACCEPT
iptables -A OUTPUT -o $HEARTBEAT_IFC2 -p UPD -j ACCEPT

#*******************************************************
# Perform NAT on all packets going to and from the iSeries
# server.  These two rules will change the source
# address(SNAT) on all traffic coming from the iSeries
# server and will change the destination address(DNAT)
# on all traffic going to the iSeries server.
#*******************************************************
iptables -t nat -A POSTROUTING -o $OUTSIDE -s $PRIVATE_IP -j SNAT
--to-source $PUBLIC_IP
iptables -t nat -A PREROUTING -i $OUTSIDE -d $PUBLIC_IP -j DNAT
--to-destination $PRIVATE_IP

#*******************************************************
# Allow ports 80 and 443, for HTTP traffic, going to
# the private IP address of the iSeries server.  We
# specify the private IP address because the packet
# going to the public IP address will have been NAT'd
# to the private IP address.
#*******************************************************
iptables -A FORWARD -d $PRIVATE_IP -p tcp --dport 80 --sport 1024:65535 -j
ACCEPT
iptables -A FORWARD -d $PRIVATE_IP -p tcp --dport 443 --sport 1024:65535 -j
ACCEPT

#*******************************************************
# Everything that hasn't matched any rules above will
# be logged here.  The log prefix is optional and allows
# for an "eye catcher" when viewing the logs.  Iptables
# logs to the file /var/log/messages.
#
# After logging, the packets will get dropped
# because we set the default policy to DROP above.
#*******************************************************
iptables -A INPUT -j LOG --log-prefix ' ## DROPPED (INPUT) ## '
iptables -A FORWARD -j LOG --log-prefix ' ## DROPPED (FORWARD) ## '
iptables -A OUTPUT -j LOG --log-prefix ' ## DROPPED (OUTPUT) ## '

echo "Done."
```