Secure Network Filesystem (Secure NFS)

By Travis Zigler

Overview of Secure NFS

- Problems with NFS
- Security of Basic NFS Configurations
- Securing NFS with SSH Tutorial
- Securing NFS with SSL Overview
- Conclusions
- Resources

Brief Introduction of NFS

NFS is a widely used protocol that allows users to share files across a network. NFS has been around for a long time and is a mature and well understood protocol.

Since it has been around for awhile and is well understood, the short comings of NFS are also well understood and exploited. If security is necessary, BASIC NFS IS NOT THE SERVICE TO USE.

So Why is NFS so bad?

Problems of NFS

- NFS relies on the inherently insecure UDP protocol
- Transactions between host and client are NOT encrypted
- Hosts and users cannot be easily authenticated, IP Spoofing is possible
- Persmissions are granted by normal access rules
- Firewall configuration is difficult because of the way NFS daemons work

So when is a good time to use basic NFS?

The only time that basic NFS should ever be used is on a secured LAN that everyone on the network is trusted. This philosphy is great, but is very impratical and very rarely the case. Most networks don't provide the comfort of having users that are completely trusted. And most networks aren't completely secured LANs. If a hacker/cracker could get into your LAN and basic NFS configurations were in place, any file in your shared directories could be compromised. And if you allow a user to be able to get root UID in the shares, your entire systems could be easily compromised.

Security of Basic NFS Configurations

1) Firewall Setup

Networking NFS through firewalls is a nightmare for system administrators because port numbers for the service may change between reboots, network restarts, or may be different on other systems. The ports are randomly assigned. To make a firewall that can control NFS, you must first tie down the ports it uses.

Port Listings for NFS using rpcinfo -p

ghostface@ghost1: /home/ghostface - Shell - Konsole

Session Edit View Bookmarks Settings Help

```
[ghostface@ghost1 ghostface]$ su
Password:
[root@ghost1 ghostface]# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp    941  status
    100024    1   tcp    944  status
    391002    2   tcp  32768  sgi_fam
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100021    1   udp  32771  nlockmgr
    100021    3   udp  32771  nlockmgr
    100021    4   udp  32771  nlockmgr
    100021    1   tcp  32786  nlockmgr
    100021    3   tcp  32786  nlockmgr
    100021    4   tcp  32786  nlockmgr
    100005    1   udp    928  mountd
    100005    1   tcp    931  mountd
    100005    2   udp    928  mountd
    100005    2   tcp    931  mountd
    100005    3   udp    928  mountd
    100005    3   tcp    931  mountd
[root@ghost1 ghostface]#
```

Shell

KDE

Secure_NFS.sxi - OpenOffice.org

ghostface@ghost1: /home/gho

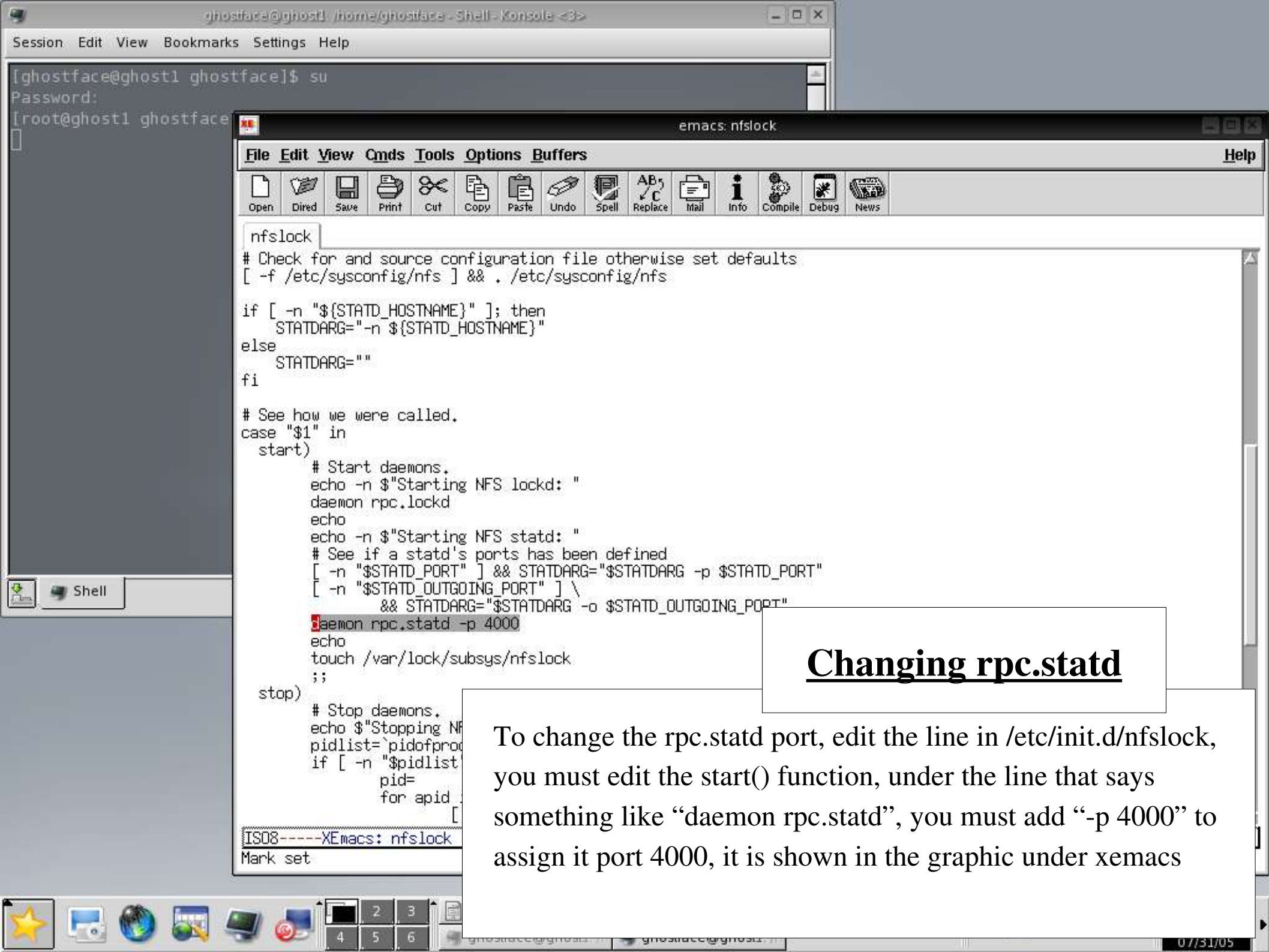
11:44 am
07/31/05

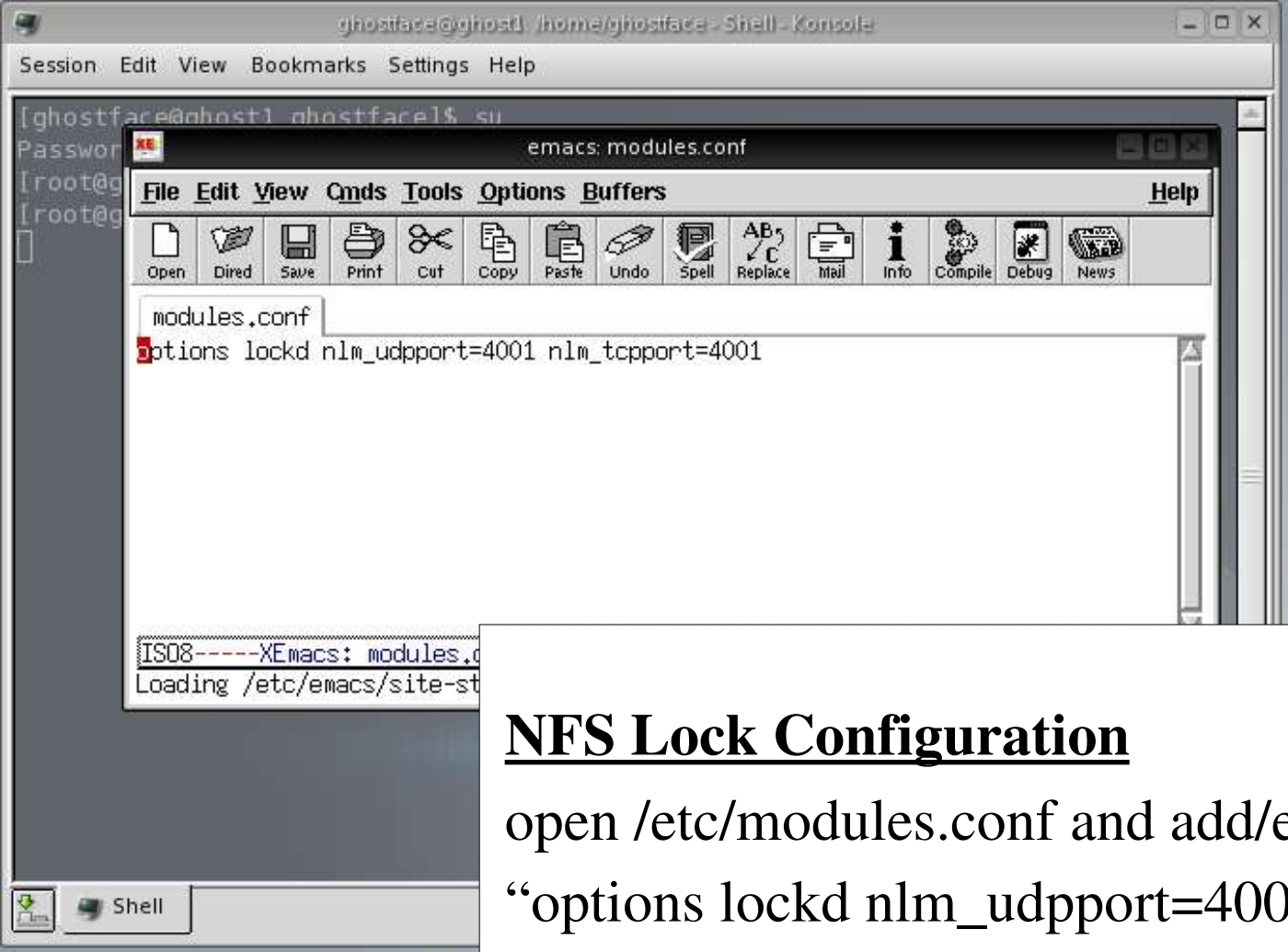

Home
Trash
Before
After
Session Edit View Bookmarks Settings Help
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
100024 1 udp 941 status
100024 1 tcp 944 status
391002 2 tcp 32768 sgi_fam
100003 2 udp 2049 nfs
100003 3 udp 2049 nfs
100003 4 udp 2049 nfs
100003 2 tcp 2049 nfs
100003 3 tcp 2049 nfs
100003 4 tcp 2049 nfs
100021 1 udp 32771 nlockmgr
100021 3 udp 32771 nlockmgr
100021 4 udp 32771 nlockmgr
100021 1 tcp 32786 nlockmgr
100021 3 tcp 32786 nlockmgr
100021 4 tcp 32786 nlockmgr
100005 1 udp 928 mountd
100005 1 tcp 931 mountd
100005 2 udp 928 mountd
100005 2 tcp 931 mountd
100005 3 udp 928 mountd
100005 3 tcp 931 mountd
[root@ghost1 ghostface]#
Shell
Session Edit View Bookmarks Settings Help
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
100024 1 udp 941 status
100024 1 tcp 944 status
391002 2 tcp 32768 sgi_fam
100003 2 udp 2049 nfs
100003 3 udp 2049 nfs
100003 4 udp 2049 nfs
100003 2 tcp 2049 nfs
100021 1 udp 32773 nlockmgr
100021 3 udp 32773 nlockmgr
100021 4 udp 32773 nlockmgr
100021 1 tcp 32804 nlockmgr
100021 3 tcp 32804 nlockmgr
100021 4 tcp 32804 nlockmgr
100005 1 udp 694 mountd
100005 1 tcp 697 mountd
100005 2 udp 694 mountd
100005 2 tcp 697 mountd
100005 3 udp 694 mountd
100005 3 tcp 697 mountd
[root@ghost1 ghostface]#
Shell
KDE
After restarting the service, the ports have changed.
11:48 am
07/31/05

If you wish to control NFS you must fix the port settings in their proper configuration files. But since I am going to talk about other protocols that route NFS through their ports, I will not lecture about that now, but have attached the instructions on fixing the ports if you wish to learn how do this.

Standard Port Settings for NFS

Deamon	RPM	Standard Port	Recommended Port	Needs Changed?
Portmap	Portmap	111	111	No
rpc.statd	Nfs-utils	Random	4000	Yes
rpc.nfsd	Nfs-utils	2049	2049	No
rpc.lockd	Nfs-utils and kernel	Random	4001	Yes
rpc.mountd	Nfs-utils	Random	4002	Yes
rpc.rquotad	Quota	Random	4003	Yes

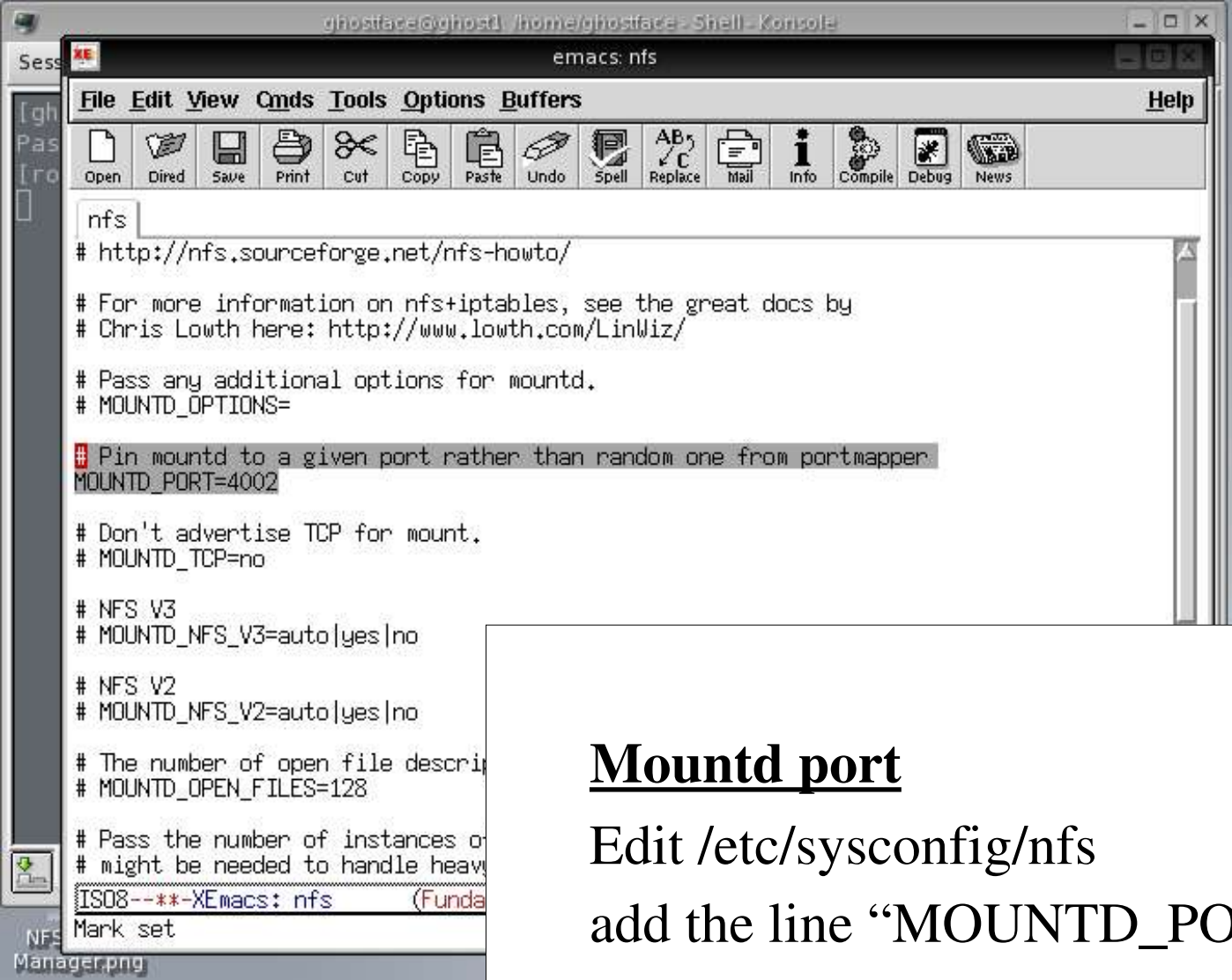

ghostface@ghost1 /home/ghostface - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
[ghostface@ghost1 ghostface]$ su
Password:
[root@ghost1 ghostface
Shell
emacs: nfslock
File Edit View Cmds Tools Options Buffers Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
nfslock
Check for and source configuration file otherwise set defaults
[-f /etc/sysconfig/nfs] && . /etc/sysconfig/nfs
if [-n "${STATD_HOSTNAME}"]; then
STATDARG="-n ${STATD_HOSTNAME}"
else
STATDARG=""
fi
See how we were called.
case "$1" in
start)
Start daemons.
echo -n $"Starting NFS lockd: "
daemon rpc.lockd
echo
echo -n $"Starting NFS statd: "
See if a statd's ports has been defined
[-n "$STATD_PORT"] && STATDARG="$STATDARG -p $STATD_PORT"
[-n "$STATD_OUTGOING_PORT"] \
&& STATDARG="$STATDARG -o $STATD_OUTGOING_PORT"
daemon rpc.statd -p 4000
echo
touch /var/lock/subsys/nfslock
;;
stop)
Stop daemons.
echo $"Stopping NF
pidlist=`pidofproc
if [-n "$pidlist
pid=
for apid
[
ISO8-----XEmacs: nfslock
Mark set
Changing rpc.statd
To change the rpc.statd port, edit the line in /etc/init.d/nfslock, you must edit the start() function, under the line that says something like “daemon rpc.statd”, you must add “-p 4000” to assign it port 4000, it is shown in the graphic under xemacs



NFS Lock Configuration

open /etc/modules.conf and add/edit the line "options lockd nlm_udpport=4001 nlm_tcpport=4001"

if the lockd is compiled directly into the kernel instead of a loadable module, you will need to add the parameters "lockd.udpport=4001 lockd.tcpport=4001" to the kernel command line in lilo or grub configuration instead



Mountd port

Edit /etc/sysconfig/nfs

add the line “MOUNTD_PORT=4002”

Rquotad port configuration

- Verify your system is running the quota version 3.08 or higher
- Make sure the /etc/rpc has this line present “rquotad 100011 rquotaprog quota rquota”, do not change any of these values
- Add or modify /etc/services

  “rquotad 4003/tcp” & “rquotad 4003/udp”

Basic Security for NFS - Issues

Access to NFS is provided by a mount to a particular directory listed in the /etc/exports file. Names or IP addresses are setup for machines that are allowed access. If a clients IP matches a share, it can mount, regardless if it is really the IP it claims to be. This makes it insecure. Someone spoofing IP addresses or a compromised machine can mount on your access points.

File access is done using normal file access controls because access control is not a function of NFS particularly. This means, that once a drive is mounted, user and group permissions take over the access control of the files.

Example: Let's say I map to UID 7777 on the server, and create a file that is only accessible to me. Then later, on another client, someone else maps to UID 7777 and has full access to files that I made only accessible to myself. Also, if someone has root on the client, they can do “su” and become ANY user.

Basic Security for NFS – Issues Continued

These steps will help you decrease some security risk, but not all, advanced tips will be given later.

- Edit **/etc/hosts.deny** with "**ALL:ALL**" to deny everyone
- Edit **/etc/hosts.allow** with "**portmap: your_subnet/subnet_mask**"
- Do the same for lockd, rquotad, mountd, and statd
  - this will allow everyone in your subnet to access portmap and other NFS daemons but no one else, these settings can be modified differently if you want more specific ranges

  Use IP numbers in these fields, not hostnames
- On the server machine, make sure you specifically add the line "**root_squash**" to your shares in the **/etc/exports** file
  - This tells the server that we don't want to trust root on the client machines

Basic Security for NFS – Issues Continued

- "root_squash" will look something like this in **/etc/exports**

  /home/share host_ip(rw, root_squash)
- These options say the host computer at "host_ip" is sharing "/home/share" with read/write access and no root UID
- Even with this option, someone could still "su" and change users. That cannot be helped, but you can make all files on the shared drive owned by root, and since root is the only account a client can't change too, the files cannot be changed.
- Another good option is to make everything read-only with "ro"
- Some good options for the client machine are "nosuid" and "noexec", nosuid limits root executables on the client, noexec limits all executables on the mount but this may be impractical
- I have appended at the end, links for Firewall ipchain configurations

Now that we have looked at basic security, how secure are we?

The answer is still, not secure at all. These options limit the ports used, and access controls on the servers and clients, but what about the transmissions? NFS is sent in Clear Text. This means it is not encrypted. Anyone running a sniffer program can take a look at your packets travelling across the network. This could mean bad news if you're sharing something like PGP keys or SSH keys over your NFS shares.

So what can we do to limit and fix our network transmissions?

We all know that portmap and other NFS daemons are problems for security, so one of the best options we have to fix these problems are tunneling through other services.

Tunneling Programs Overview

So now, let's take a step back from the basic configurations and start over with a more secure way of thinking. The things we need to fix about NFS are:

- What ports should we use for our transmissions
- Setting up firewall settings to handle a tunneled version of NFS
- Handling encryption and authentication over a transmission

Tunneling Options

When it comes to tunneling, the tunnel is only as secure as the protocol that creates it, with that said, the security of your "Secure NFS" is only as good as the services you use to create it. After doing a whole lot of googling around online, I find very few tutorials on how to setup a Secure NFS, but as it seems most tutorials I found were using SSH or SSL. The details of these tunneling programs will be given to you by other speakers, so I will try not to go into much detail on them.

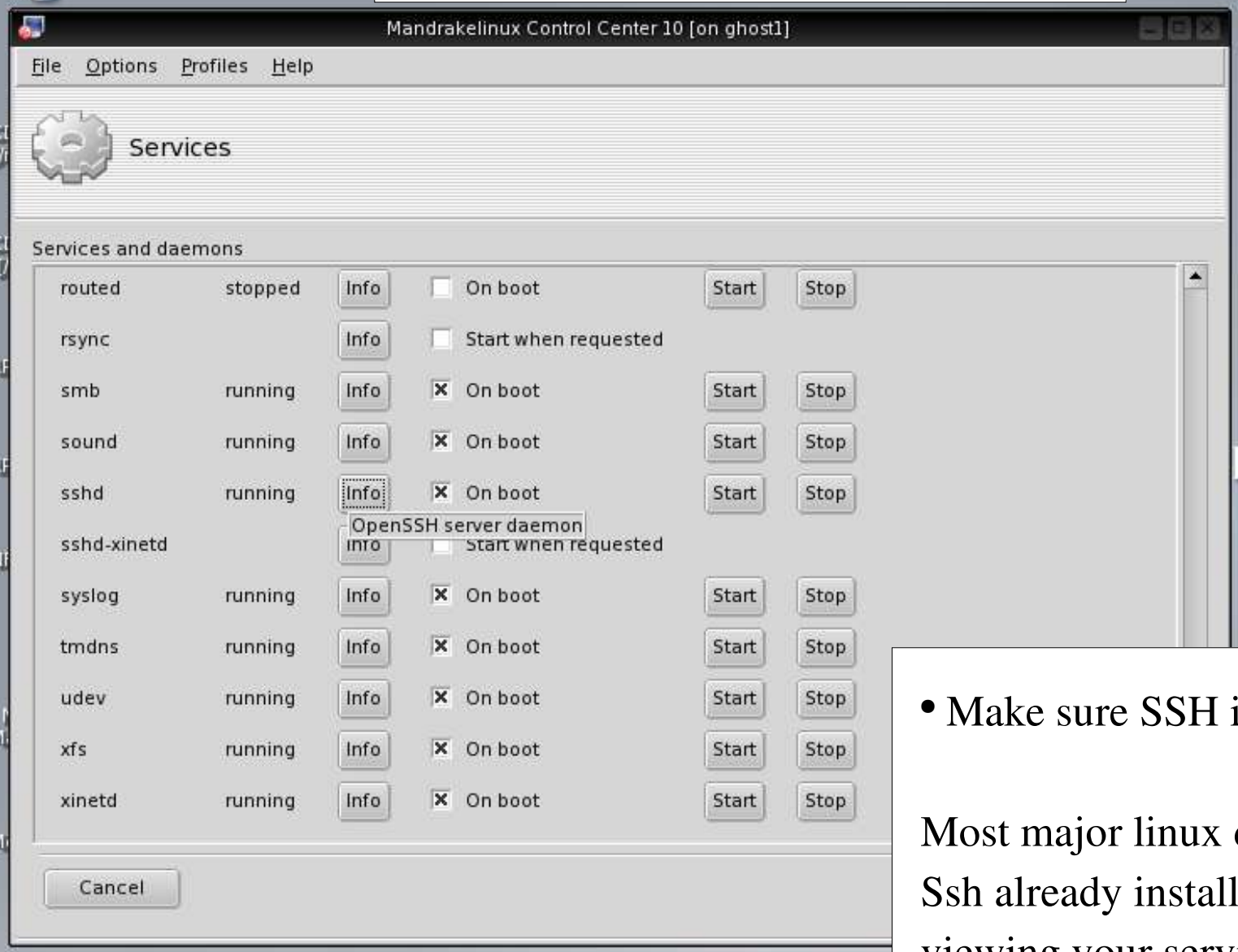
The tutorial I will give, shows secure NFS through SSH, but other ways are definitely possible. Since these configurations are fairly lengthy, I will try to limit them as much as possible, but provide the links to the sources I have used if further clarification is needed.

I will also give a brief overview of Secure NFS over SSL.

Secure NFS over OpenSSH Overview

- Prerequisites
  - Getting Started
- Server configuration
  - /etc/exports
  - Firewall design for forwarding / ipchains and iptables
- Client configuration
  - Find what ports mountd and nfsd are running
  - Set local forwarding rules
  - Mount the shares

Prerequisites - Getting Started



- Make sure SSH is installed.

Most major linux distrobutions come with Ssh already installed, you can check by viewing your services, in my case, I am running Mandrake 10.1 and from the picture you can see it is running

Server Configuration - /etc/exports

Setup a directory to share, this could be anything, but in my example I will use /home/username/ssh_share, this can be done by typing "**mkdir /home/username/ssh_share**" as root

Open and edit the /etc/exports file to contain the share to itself, this will be something like

/home/username/ssh_share 10.10.10.1(rw,insecure,root_squash)

This adds the share **/home/username/ssh_share** and **10.10.10.1** is the IP of my machine, the **insecure** option just means it can use ports above 1023

Server Configuration - Firewall

Note: when starting the services, NFS must be started before SSH, because SSH may bind the ports used by NFS, and NFS will not start

The following rules should be setup for the firewall to allow NFS over SSH to work:

IPTABLES

```
iptables -A INPUT -i eth0 -p tcp -s client --dport ssh -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport ssh -d client -j ACCEPT
iptables -A INPUT -i eth0 -p tcp -s client --dport 111 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 111 -d client -j ACCEPT
```

IPCHAINS

```
ipchains -A input -i eth0 -p tcp -s client --dport ssh -j ACCEPT
ipchains -A output -i eth0 -p tcp --sport ssh -d client -j ACCEPT
ipchains -A input -i eth0 -p tcp -s client --dport 111 -j ACCEPT
ipchains -A output -i eth0 -p tcp --sport 111 -d client -j ACCEPT
```

**client is an IP or network to be allowed, this is for a firewall with default policy of DENY

Client Configuration

Steps To Follow

- Find out the port numbers that mountd and nfsd are using
- Setup forwarding rules locally for SSH
- Mount the shares

Port Numbers

On the server, use the “rpcinfo -p” command to find out the server port numbers for mountd and nfsd

In my example they are: **nfs 2049** and **mountd 4002**

Client Configuration - Continued

Now that we have the port numbers they can be used to create the tunnels from the server to the client, I will use 2222 as the port nfsd will connect on the local machine, and 3333 as the port mountd will use. We will call our server "server", and out client "client" The user trying to connect will be called "user", so on the command line We would type something like this:

For connecting the nfsd:

ssh -f -L 2222:server:2049 -l user server sleep 600

For connecting mountd:

ssh -f -L 3333:server:4002 -l user server sleep 600

OR concatenate the two above commands with extra options:

ssh -f -c blowfish -L 3333:server:2049 -L 3333:server:4002 -l user server /bin/sleep 600

**sleep simply waits the number of seconds specified for connection, the extra option "-c blowfish" just changes the encryption type, which is faster than the default (3des)

After adding the tunnels we can now mount them. This is done using the command:

mount -t nfs -o tcp,port=2222,mountport=3333 localhost:/home/username/ssh_share /mnt/nfs/ssh_share

**where /mnt/nfs/ssh_share is the local directory for the mount

This can be made easier by automating the process with scripts and such but that is not the focus of this presentation.

Firewall Config for the Client

iptables -A INPUT -i eth0 -p tcp ! --syn -s server --sport ssh -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp -d server --dport ssh -j ACCEPT
iptables -A INPUT -i eth0 -p tcp ! --syn -s server --sport 111 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp -d server --dport 111 -j ACCEPT

ipchains -A input -i eth0 -p tcp ! -y -s server --sport ssh -j ACCEPT
ipchains -A output -i eth0 -p tcp -d server --dport ssh -j ACCEPT
ipchains -A input -i eth0 -p tcp ! -y -s server --sport 111 -j ACCEPT
ipchains -A output -i eth0 -p tcp -d server --dport 111 -j ACCEPT

Additions, Comments, and Warnings

Generating ssh keys will help with logging on to the mounts by eliminating the need to type in passwords

- Users are still not authenticated, but hosts can be authenticated by the use of the ssh keys

- Since all client connections using SSH are encrypted, there will be a Significant increase in CPU overhead, use of different encryptions for the tunnels may decrease this (our example we used blowfish instead of 3des)

NFS Over SSL Overview

- NFS can also be used in a similar way over an SSL tunnel

- Clients will tunnel to the server via the SSL-secured ports, therefore the connections from the nfsd, mountd, etc are coming from the localhost machine, and not directly from the client, therefore you must appropriately edit the /etc/exports file to except mounts from the localhost

- You may also close the ports used for portmapper, nfsd, mountd, etc because the services are using the SSL ports, and you may setup your firewall in a way that only allows specified IP addresses to contact you SSL ports

- SSL can provide certificate verification between clients and hosts

Conclusions

- Basic NFS setups are insecure over network transmissions
- The only use of basic NFS is over a secured LAN with trusted users
- The only way to increase security of NFS is to use an encrypted channel
- SSH and SSL can provide the means for an NFS channel from client to server
- The use of authentication keys will increase security of your connections
- Increased security also increases the CPU overhead
- Different kinds of encryption techniques can balance between CPU overhead & security
- The process of setting up your Secure NFS tunnels can be automated through scripts
- Secure NFS is only as good as the tunneling services used to implement it

Resources Page

- Secure NFS and NIS via SSH Tunneling

http://www.math.ualberta.ca/imaging/snfs/

- Encrypted NFS with OpenSSH and Linux

http://www.samag.com/documents/s=4072/sam0203d/sam0203d.htm

- NFS over SSL

http://www.edu.helsinki.fi/atk/ltsp_kiosk/nfs_over_ssl.html

- Implementing NFS

http://www.linuxplanet.com/linuxplanet/tutorials/2684/1/

- sNFS Secure NFS

http://www.crufty.net/ftp/pub/sjg/help/sNFS.html

- Setting Up A NFS Server

http://nfs.sourcefogre.net/nfs-howto/server.html

- Security and NFS

http://www.higs.net/85256C89006A03D2/web/PageLinuxNFSSecurity

End of Presentation