A shared write protected root filesystem for a cluster of network clients

Ignatios Souvatzis*

Abstract

A method to boot a cluster of diskless network clients from a single write-protected NFS root file system is shown. The problems encountered when first implementing the setup and their solution are discussed. Finally, the setup is briefly compared to using a kernel-embedded root file system.

1 Introduction

- Managing three diskless network clients can be done manually.
- Manually managing ten is still possible, but tedious.
- Manually managing hundreds is close to impossible.

When we got ten disk- (and head-) less network computers of a new type that we wanted to use as computing nodes for a parallel virtual machine for a practical course[1], we decided to set them up with a shared root file system.

However, a BSD root file system has to be unique and writable for every client machine for a couple of reasons, so that a single writable shared root file system does not work.

As an alternate solution, we placed most writable directories onto (virtual) memory file systems. The program area and configuration files need only to be exported by the server for read only access. This way, the system programs, system libraries, and the configuration are protected against malicious users, even if they should gain root privileges on the client machine.

This presentation elaborates on the problems we encountered and the solutions we implemented, using stock NetBSD 1.5/1.6 as the client operating system, with just a small script and a few configuration lines added.

*Universität Bonn, Institut für Informatik, Römerstr. 164, D-53117 Bonn, Germany; e-mail `ignatios@cs.uni-bonn.de`

2 System Environment

2.1 File- and Bootserver

SUN UltraSparc-10, originally running Solaris 2.6 (now Solaris 8), located in a secure room. However, the problem and its solution do not depend on the machine size and operating system, as long as the clients' root file systems are mounted via something similar to NFS.

2.2 Clients

Originally 10 Digital Network Appliance Reference Design (DNARD) machines, with 64 MB of RAM, no disk, no keyboard/monitor attached, running NetBSD 1.4, later 1.5.3 (now 1.6), located in a secure room.

We believe that the solution could be applied, with some modification, to other Unix-like operating systems, although certain features of NetBSD did help a lot.

2.3 Users

The machines are to be used by 10 to 20 students for the duration of a half-year course. They should be able to login from home. Even if we successfully limit logins to them, we can't really trust them not to break into the system when they can, or even more likely, by running some dangerous script given to them by evil people.

3 Why a shared, read-only root?

3.1 Easier administration

We want to install and configure on the server (or just one machine, under special circumstances) and at most have to reboot the (other) clients.

3.2 Saving disk space

NetBSD-1.5.3/arm32 or 1.6 needs about 20 megabytes of disk space in the traditional root file system (see figure 1).

```
server 196 # du -ks bin sbin etc
5795    bin
12760   sbin
606     etc
```

Figure 1: NetBSD-1.5.3/arm32 root filesystem usage

While the total size (200 megabytes for 10 machines) does not look like much today, even assuming 100 machines, disk space was more expensive when we started, and *reliable* disk space, especially *backed up* disk space, has not become cheaper as fast as unreliable ("desktop") IDE disks.

3.3 Server security

The root filesystem *has to* be exported with (client-side) root access rights. Leaving it writable allows a malicious client to permanently change (at least) its own configuration.

(Securing user data against read-out or modification from a manipulated client is beyond the scope of this work.)

4 Problems found

4.1 Booting

- The client has to learn its name and network address.
- The client has to learn the name of its NFS swap file.
- The client has to learn the name of the per-client filesystems on the server.

4.2 During operation

Files on the root file system that are written on traditional installations, or that are tied to a single machine, include:

- System log files, which are written to all the time.
- `*.pid` - files, written (at least) during startup of server processes.
- Sockets (`/dev/log, /dev/printer,` ...), which are created by the server processes that use them to communicate with their clients.
- Device nodes (of terminal-like devices, including pty, mice, keyboards) change their owner at each login and logout.
- The ssh host key is stored in the root file system.
- The package system database, normally `/var/db/pkg`. If we made `/var` per-machine, we'd need a per-machine `/usr/pkg`, too. Otherwise we would need to move the database to a shared location.

4.3 During shutdown

`/etc/nologin` is created by the first machine shutting down. It will prevent login on the other clients.

This is fine for a coordinated shutdown of the whole cluster, but not when booting a single client machine to test, say, a new kernel.

5 Methods

5.1 Some problems aren't

- As all our clients are equal in configuration and don't carry any permanent state, there is no point to make them distinguishable in a cryptographically secure way. So we just use the same set of host keys for sshd on all the machines.
- Network configuration for IPv4 is done by DHCP anyway (when booting the machines), so we can also use it to learn the client name (and a few other parameters). For IPv6, we're using stateless IPv6 autoconfiguration.
- We configured syslog to send all logged events to the server.

5.2 DHCP server setup

Here, nothing special is needed. The clients are set up with fixed addresses and names assigned to them and learn the name of the kernel to boot from and the location of the root file system (figure 5).

```
server:/BSD/root2/1.6     /           nfs ro               0 0
server:/BSD/root/vargames /var/games  nfs rw               0 0
swap                      /tmp        mfs rw,-s=32768      0 0
#1.5: swap                /var/run    mfs rw,-i=512,-s=256 0 0
```

Figure 2: /etc/fstab

```
rc_configured=YES
shroot_pfx_var=server:/BSD/root/var-
shroot_pfx_swap=server:/BSD/swap/
#1.5: critical_filesystems_beforenet="/var /var/run"
critical_filesystems_local="/var /var/run"
update_motd=NO
rpcbind=YES # nfs
domainname=nis.doma.in # NIS
ypbind=YES
amd=YES amd_dir=/var/amdroot
nfs_client=YES
ip6mode=autohost
defaultroute=10.10.10.10
sshd=YES postfix=YES ntpd=YES
lpd=YES lpd_flags=-s
inetd=YES       # ntalk, (c)fingerd
```

Figure 3: /etc/rc.conf

```
> mount
server:/BSD/root2/1.5 on / type nfs (read-only)
server:/BSD/root/var-client on /var type nfs
server:/BSD/swap/client on /swap type nfs
mfs:34 on /dev type mfs (asynchronous, local)
mfs:37 on /etc type mfs (asynchronous, local, union)
mfs:1000 on /var/run type mfs (asynchronous, local)
server:/BSD/root/vargames on /var/games type nfs
mfs:1085 on /tmp type mfs (asynchronous, local)
pid1121@client:/home on /home type nfs
studsrv:/opt/export/home/stud/user on \
        /var/amdroot/studsrv/opt/export/home/stud/user \
        type nfs (nodev, nosuid)
server:/export/home/2 on /var/amdroot/server/export/home/2 \
        type nfs (nodev, nosuid)
```

Figure 4: The filesystems at run-time

```
option domain-name "my.doma.in";
option domain-name-servers 10.10.10.2, 10.10.10.1;
deny unknown-clients;
use-host-decl-names on;
subnet 10.10.10.0 netmask 255.255.255.0 {
        option routers 10.10.10.3;
        group {
                option lpr-servers server.my.doma.in;
                server-name "server";
                next-server server;
                filename "netbsd-SHARK-1.6";
                option root-path "/BSD/root2/1.6";
                host client {
                        hardware ethernet 10:20:30:40:50:60;
                        fixed-address client.my.doma.in;
                }
                ...
        }
}
```

Figure 5: DHCP server configuration file excerpt

5.3 Sockets and `*.pid` - files

During boot time, a small memory file system is created and mounted on `/var/run`. (BSD mfs stores its data in the address space of the `newfs_mfs` process, so it's pageable[2]). We added this to `/etc/fstab` (figure 2) and marked `/var/run` as a filesystem to be mounted very early in `/etc/rc.conf` (figure 3). This is used for the following files:

- `*.pid` files - often in `/etc` - are created in `/var/run` by all daemons integrated into NetBSD or installed from the package system.
- `/dev/log` was moved to `/var/run/log` by changing the syslogd code; this is the standard location in NetBSD nowadays.
- `/dev/printer` was moved to `/var/run/printer`; this is the standard location in NetBSD.

5.4 Device nodes

We create a memory file system for `/dev`. `/dev` on the server only needs `/dev/console` (for the benefit of `/sbin/init`).

At boot time, we populate `/dev` by running the MAKEDEV script, which is installed in `/sbin` in our setup. This takes about 20 seconds. Should we use slower machines, we could tune the amount of devices created - currently, we run `sh MAKEDEV all`.

The code to do this–as all special code needed–lives inside a small script called `shroot` (figure 6).

NetBSD-1.6 /sbin/init does all of this automatically, when no /dev/console is found. (This was implemented to make CD-ROM and MS-DOS filesystem demonstration installations possible.) After upgrading, we were able to remove the `/dev/console` *on the exported root file system and remove the lines in* `shroot` *that handle* `/dev`*. They are in comment lines in figure 6.*

5.5 Swap, `/var`

During boot time we run `/bin/hostname` to find out how we're called - the kernel has learned it using DHCP. Using this name, we synthesize the server-side names of the swap file and `/var` filesystem to mount.

`/var` is per-machine to allow per-machine spool files for outgoing e-mail, printing and similar services.

5.6 Remaining files written to `/etc`

Some of the files on `/etc` can be configured not to be changed (e.g., `/etc/motd`). However, there are a few that can't be easily handled without code and functionality change, like `/etc/nologin`.

As a simple catch-all to this problem, we create a small memory file system and union-mount it over the NFS `/etc`. This is handled in `/etc/rc.d/shroot`, too (fig. 6).

```
#!/bin/sh

# PROVIDE: shroot
# REQUIRE: root
# BEFORE: mountcritlocal

# shared root setup.

. /etc/rc.subr

name="shroot"
start_cmd="shroot_start"
stop_cmd=":"
#required_files="/sbin/MAKEDEV /sbin/MAKEDEV.local"

shroot_start () {

    hostname=‘/bin/hostname‘

    case "$shroot_pfx_var" in
        "") ;;
        *) /sbin/mount -t nfs ${shroot_pfx_var}${hostname} /var ;;
    esac
    case "$shroot_pfx_swap" in
        "") ;;
        *) /sbin/mount -t nfs ${shroot_pfx_swap}${hostname} /swap\
            && /sbin/swapon /swap ;;
    esac

#   /sbin/mount -t mfs -o -i=256 -o -s=512 swap /dev
    /sbin/mount -t mfs -o -i=256 -o -s=512 -o union swap /etc
    /bin/chmod 755 /etc
#   /bin/chmod 755 /dev
#   echo -n "creating device nodes...";
#   /bin/cp /sbin/MAKEDEV /sbin/MAKEDEV.local /dev
#   (cd /dev; sh MAKEDEV all)
#   echo done.
}

load_rc_config $name
run_rc_command "$1"
```

Figure 6: /etc/rc.d/shroot

5.7 Where to place the code?

As mentioned already, we concentrated all special startup script code needed in the `shroot` script.

Obviously, this script has to run early in the boot process (before device nodes, `/var/run` etc. are accessed). Traditional `/etc/rc.local` is much too late. For NetBSD-1.4, we had hooked the equivalent script up in `/etc/netstart.local`; without that, we would have had to find a suitable place in `/etc/rc`, or among the zillions of SVR4 or Linux startup scripts.

The explicit startup script dependencies first implemented in NetBSD-1.5 [3] made the task easy: the `shroot` script is placed into the directory /etc/rc.d/ and states explicitly that it wants to be run after the root file system is there, but before critical local file systems are mounted (see figure 6).

This is important, because `/var/run` has to be mounted after shroot mounts `/var`!

Depending on applications, some subdirectories of `/var` have to be mounted from a shared NFS volume - e.g. `/var/mail` or `/var/games`. This is handled normally in `/etc/fstab` or using the automounter.

Figure 4 shows the run-time file system table. Note that `/usr` is embedded in the root file system! As it is never written by the clients, there is no point to seperate it from root.

5.8 Software Installation with the `pkg`-system

We are using a shared directory tree for the third-party packages installed through the NetBSD package system. The package database is moved inside `/usr/pkg`, so that it is shared, too.

During software installation, we give a single chosen client (which temporarily gets a keyboard and monitor) write access to the server, and revoke it afterwards.

The environment variable `PKG_DBDIR` has to be set to `/usr/pkg/libdata/pkgdb` for installation as well as any other use of the `pkg_xxx` tools. Fortunately, this is all that is needed to make the package system tools happy. `pkg_info` should be usable by the students to find out what software is installed.

However, from a security and performance viewpoint, it would be better to have cross-install tools and to run them on the server.

6 Why no embedded root file system?

An alternate method we've considered is to embed a root file system in the client kernel. This leads to the same security benefits outlined in section 3.3. However, there are two drawbacks:

- An embedded filesystem is completely RAM based, non-pageable, during execution. MFS, on the other side, is virtual memory based.

  To make this work at all, the part of the root file system actually inside the kernel has to be carefully tuned. The chosen method allows to use a stock NetBSD installation, with only the `shroot` script added and some configuration files changed.

- Both changing the kernel and changing some configuration file require to embed the kernel file system anew into the kernel file and reboot all clients.

  This is inconvenient, especially when only the configuration of some short-running component was to be changed.

  The chosen setup allows to change all of `/etc` on the server and have it immediately available, if so desired.

7 Summary

A method to make the root file system for network clients read-only and shared has been presented. Administration can mostly happen on the server. Client break-ins would not affect the system files on the server. The installation uses a mostly normal NetBSD-1.5.3 or -1.6 installation, with a single script added and some configuration files in `/etc` changed.

References

[1] Bonn University CS Dept., parallel systems student lab home page:

http://theory.cs.uni-bonn.de/info5/system/parlab/

[2] NetBSD `mount_mfs`(8) and `newfs_mfs` manual pages

[3] NetBSD `rc`(8), `rcorder`(8) and `rc.conf`(8) manual pages