Security and System Maintenance Automation

Introduction

Making and keeping Unix System secure with today's dynamic workforce is a problem for all companies big and small.

In this paper, I will outline a different approach to automating system and security changes using rdist as the core program. Using this method, the updates can be as simple as copying a file to the correct distribution directory. This level of simplicity allows even the most junior of Security Administrators the ability to add files and make changes across a large set of systems.

This automation system is easy to use for all levels of Security Administrators. It will do the initial security customization of a system and all future upgrades. It can be used as an Intrusion Detection Systems, similar to fcheck, to monitor your systems for changes. It can easily be learned and understood by your Junior Security Administrators.

Installing and Maintaining Systems

There is often too much emphasis placed on the initial installation and initial security configuration of a system and not enough on the future maintenance and monitoring of the system. If we look at the basic steps for installing and maintaining a system (and the time frames for these steps) we see:

- Initial Systems installation (<1 Day)
- Initial Security and System setup (<1 Day)
- Security and System Updates (months to years)
- Monitoring updates and key files for changes. (months to years)

I.T. Problems faced by today's companies, large and small:

If we also look at a couple key problems faced by companies today we see:

- Experienced Security Administrators leaving to join the latest .com company.
- New, perhaps junior, Security Administrators that need to be brought up to speed.

Stages of Security/System Administration

If we take a look at how Security or System Administrators experiences and capabilities advance over time, you see three distinct stages.

Stage One - do it by hand

Junior Security or System Administrators will typically install an OS on a system and then go back and customize them by hand. Initially the changes will often be memorized and eventually written down. This process is okay for single systems but is very time consuming for any number of systems. It also leads to inconsistencies and provides no inherit file or change monitoring.

Stage Two - do it with scripts

Security or System Administrators will quickly learn the shortfalls of doing things by hand and incorporate the changes into scripts (awk, bourn-shell, csh, perl, etc...). This makes the systems much easier to customize. All systems that are installed with the same parameters and customized with the same script should be rather identical. The problems occur as the security needs change or are extended. The master security script or scripts need to be changed and the changes need to be applied to all the previous systems. Many questions and problems come up. You will need to keep the new script separate until it is run on all the older systems but then do you roll all the changes into one big script or do you keep the scripts separate small pieces? What happens if a system is down when you are running the update? Do you keep lists of what is run where? Some of these problems can be avoided by re-writing all the scripts or parts of the script such that they can be run over and over. If the script is complex, this could be a difficult problem and will often result in a script that the Junior Security or System Administrators will have a problem with.

Stage Three - do it with rdist

After trying to juggle many large and small scripts and always having one or two systems that get overlooked with every update, you begin to think "there has got to be a better way".

Enter rdist!

The rdist program runs off of a simple script or distribution file (distfile). The format of the file in its simpler form is easy for even the most Junior Security or System Administrators to understand. All systems and "master" directories and files are listed in the distfile. Each time you add a directory or file to the master directory and you run rdist, it will check for the files on the remote host and distribute the "new" files out to the listed hosts. If you add a host to the host list within the distfile, rdist will be able to update the new host with all the files in the master directories. This will make the new host look just like all the others in the distfile's host list. All the hosts now have identical copies of all the files. As you can see, host additions are simple, file and directory additions are simple, all the system are kept in sync.

The rdist program

A Brief History

The rdist program has been around since about 1985. It first appeared in the 4.3 Berkeley Software Distribution. (1) It was developed by the University of California, Berkeley. It was used before the times of NIS, NFS and DNS primarily to propagate changes of system files (/etc/hosts, /etc/passwd, /etc/groups, etc...) between multiple systems. Today it is still very useful for keeping security related system files safe, consistent, and up to date.

Details

The rdist program runs off of a simple script or "distfile". The rdist program can be run in update mode where it fixes differences it finds or in "verify" mode (-v) where it will tell you what files are different. It also has a binary comparison mode to do a bit-for-bit comparison (for the truly paranoid). The rdist program can also, if necessary, run scripts when select files are updated.

Examples

The rdist program has a simple control or "distfile" to tell it what goes where.

The Simplest Form

Here is a simple form of a distfile:

```
ToHosts = ( fred wilma betty barney )
Files = ( FilesCommon )
    ${Files} -> ${ToHosts}
    install / ;
```

The above distfile will distribute the files in the directory "FilesCommon" to the hosts named: fred wilma betty and barney The FilesCommon directory contains the following files (edited):

```
FilesCommon/etc/
FilesCommon/var/

FilesCommon/etc:
LOCAL/
default/
inet/
defaultrouter.betty
defaultrouter.default
group
nsswitch.conf
passwd
resolv.conf

FilesCommon/etc/inet:
inetd.conf

FilesCommon/etc/LOCAL/Scripts:
add.ptys.to.etc_system.sh*
auto-link.sh*
cron-reread.pl*

FilesCommon/etc/default:
login
su

FilesCommon/var/spool/cron/crontabs:
root.barney
root.default
```

(This file layout will be explored more in examples below.) From the listing above, you can see that all systems would get a number of files from the FilesCommon directory. For example, all will get the files at FilesCommon/etc/nsswitch.conf and FilesCommon/etc/resolv.conf installed into their local /etc/ directory. These will replace the original files as required by this site.

Handling Differences

All systems will not be completely identical. Occasionaly you will have a system that just cannot use an generic or default file. You have three optional ways of handling these differences.

Different Directories

If you find you have a large number of files that differ, for example, by Operating System, you can name the directories and put the hosts in their respective lists. The distfile could look something like:

```
            ToHostsSun = ( fred wilma )
            ToHostsHP = ( betty barney )

            FilesSun = ( FilesCommonSun )
                ${FilesSun} -> ${ToHostsSun}
                install / ;

            FilesHP = ( FilesCommonHP )
                ${FilesHP} -> ${ToHostsHP}
                install / ;
```

Here hosts fred and wilma are Sun systems so they get the Sun files and betty and barney are HP systems so the get just the HP files.

Different Single Files on Few Systems

Another way to solve the differences is to link in the correct file at install time with a script. The files will have additional information, such as a hostname, attached to the files. The script will tell from this attached info, which file to link to. From the file list above, you will notice that there are two files located at

```
              FilesCommon/etc/defaultrouter.betty
              FilesCommon/etc/defaultrouter.default
```

and

```
              FilesCommon/var/spool/cron/crontabs/root.barney
              FilesCommon/var/spool/cron/crontabs/root.default
```

For the defaultrouter files, host betty is on a different network so it will get it's host specific link:

```
            defaultrouter -> defaultrouter.betty
```

and other hosts will get

```
            defaultrouter -> defaultrouter.default.
```

The crontab files are linked-to the same way. Host barney will get a link "root -> root.barney" and all others will get a link "root -> root.default".

This linking is done by a simple script, auto-link.sh, that gets called whenever any of the files with the same prefix (FilesCommon/etc/defaultrouter* or FilesCommon/var/spool/cron/crontabs/root*) change.

The Distfile for this would be:

```
          ToHosts = ( fred wilma betty barney )
          Files = ( FilesCommon )
              ${Files} -> ${ToHosts}
              install / ;
              special ( FilesCommon/etc/defaultrouter.* )
                  "/etc/LOCAL/Scripts/auto-link.sh /etc/defaultrouter" ;
              special ( FilesCommon/var/spool/cron/crontabs/root.* )
                  "/etc/LOCAL/Scripts/auto-link.sh /var/spool/cron/crontabs/root;\
                  /etc/LOCAL/Scripts/cron-reread.pl " ;
```

Note: the update of the root crontab file causes two scripts to run. The script /etc/LOCAL/Scripts/auto-link.sh links in the correct file and the script /etc/LOCAL/Scripts/cron-reread.pl sends signals to cron to cause it to re-read the crontab files.

Smarter Scripts

This method leaves the whole problem up to the script. The script can have a simple test to see which system it is running on. If it is not running on the correct system, it simply exits. This works great for scripts or programs that can be front-ended with a script. It will not work on data files like /etc/passwd or /etc/group. No changes to the distfile are needed.

Script Design Rules

- Scripts are simple. You need to keep the new and/or Junior members of the team in mind. The simpler the scripts the sooner they can come up to speed. Also, if they make mistakes within these simple scripts, the effect of the mistake will be limited. If they make mistake in a large "jumbo" script, they could bring down a large number of systems.
- Keep the complexity to a minimum number of scripts. As with all rules, their are exceptions. We want to keep the the complexity out but when it creeps in, try to keep it to as few scripts as possible. Make sure these scripts are well marked with warning messages that the un-informed should stay clear.
- No sed, ed, or massive awk scripts to update files. Keeping the Junior Security or System Administrators in mind, rather than trying to repair files, they should simply be replaced. All of the /etc/init.d startup files can simply be replaced. For files that cannot be simply replaced (like /etc/system on Solaris systems), use simpler grep to check for additions and append to the end of the file.
- Scripts can be run again and again without causing problems. Since the scripts are simple, it should be easier to make them such that they can be run over and over again. From the above file list we have the three following scripts:
    - add.ptys.to.etc_system.sh - This simply greps the /etc/system file looking to see if the "ptys" lines are in the file. If not it simply adds them to the end. Since the check is done first, this script can be run over and over again without causing problems.
    - auto-link.sh - This script is passed the full path name of a file. It takes that file name and looks at all of the files that start with that name to determine which file to link to for this host. If the correct file is linked in, nothing else is done.
    - cron-reread.pl - This script sends cron a message to tell it to re-read a particular crontab file. Sending this message many times to cron, does no harm, it simply re-reads the crontab each time.
- The scripts need to be copied and run locally. This is done for a few reasons. First, you can use updating of the file to cause it be run (controlled by the rdist file). Second, since all systems have the same crontab files the cron scripts will all be running at the same time. With larger installations, this could swamp an nfs server holding the script.

Getting Started

Picking your "master" host.

Setup a small, simple, reliable system to be your master host. A small desktop system that no-one will hunt down, no-one will miss, no-one will notice. You can close down all the ports (nfs, tftp, rshd, rlogin, cmsd, etc.) that are known to have problems. This will make your master host very secure. Never pick your biggest system that everybody logs into and has tons of daemons on. This type of system is easier to break into and has so much stuff going on that you will not be able to tell when something new (like a break in) has occurred. Since all the other systems are trusting the master host, you want the master to be as trustworthy as possible.

Security Concerns

- rdist transfers files in the clear. Systems inside your company could, perhaps, be able to view files (like /etc/passwd and /etc/shadow) as they are passed over the net. Note: ssh or Secure Shell should be installed to make rdist more secure.
- /.rhosts files are required to allow root in from the "master" host.
- Bugs. Beware of rdist security bugs. Check with your OS supplier for the latest patches and/or updates.

What rdist Doesn't Do Well

- Fixing Permission Problems - When rdist is run in update mode and permissions are incorrect, you will see warnings about the differences. Although rdist does not fix these itself, a simple script can process the output of rdist and run "rsh" or "remsh" commands to fix the problems.
- Patch Updates - If you have completely identical systems (including the master), you should be able to install patches on the master system and then rdist the whole system portion of the file system out from the master to all the slave systems. And, in fact, back in the good old SunOS 4.X days we did just that. However, Solaris is now too "smart" to install patches that you don't need. It won't install patches for hardware that does not exist. So, if your master is a little different, this method won't work. It is best to install patches automatically with a patch automation program. However, rdist is very useful for finding permission problems after patches have been applied. See "Fixing Permission Problems" just above.

Experiences

Using rdist in this manner has been setup in the following environments:

- Managing a set of about 60 SunOS systems at major Unix vendors Desktop Development group.
- Managing a set of as many as 1000 systems at major Unix vendors microprocessor development group.
- Managing a set of 25 Sun servers and desktop systems and 5 HP-UX systems at a Design Automation startup.

References

- rdist.1 - http://www.linuxcentral.com/linux/man-pages/rdist.1.html
- src/usr.bin/rdist/rdist.1 http://people.freebsd.org/~knu/cgi-bin/cvsweb.cgi/src/usr.bin/rdist/rdist.1?rev=1.14&content-type=text/x-cvsweb-markup
- System management issues: http://www.beowulf.org/pipermail/beowulf/1998-July/001103.html
- "rdist to the Rescue!" by Judith Ashworth, Sys Admin November/December 1992 issue, pg 18.
- "FCheck: A Solution to Host-Based Intrusion Detection" by Ron McCarthy, Sys Admin Magazine, December 2000
- "New Approaches to Making Solaris More Secure" by Rich Teer, Sys Admin Magazine-Solaris Admin supplement, November 2000