SUN BLUEPRINTS [illegible] APPROVED

Rapid Recovery Techniques: Auditing Custom Software Configuration

By Richard Elling - Enterprise Engineering

Sun BluePrints™ OnLine - February 2000

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-4610-10
Revision 01, February 2000

Rapid Recovery Techniques: Auditing Custom Software Configurations

System recovery time, also described as mean time to recover (MTTR), is an important part of availability calculations. Reducing recovery time directly and positively impacts the overall availability of systems.

This is the fourth article in a series that covers rapid recovery techniques for the Solaris™ Operating Environment. The focus of this series is to describe ways in which recovery time for repairing or restoring systems can be decreased, thus increasing overall availability. Computer industry analysts observe failures of three main types: product, people, and process. This series will also present recovery techniques for problems specifically unrelated to failed hardware—in particular, processes to recover from errors caused by people.

Software configuration management is critical for determining what has changed on a system. If you know what has changed, you can recover only those files which have been altered—thus reducing the recovery time. The Solaris Operating Environment software registry contains records of each file installed on the system. You can use the `pkgchk` program to query the software registry and determine if a file has changed since installation. Although this works fine for installed software, a more comprehensive configuration audit should include all files on a system. To do this, use a Bourne shell script to generate a `pkgmap` format file describing the file configurations. This `pkgmap` format file can be used by the `pkgchk` program to query against the files at a later date.

Software Packages

Software is typically managed in the Solaris Operating Environment via the package facility. Installing a software package via the `pkgadd` command updates the software registry and the database of installed software. You can use the `pkgchk` command to check a file against its installation time configuration. However, some files such as `/etc/hosts` and `/etc/passwd` often change after installation. Also, many files such as user files do not have entries in the software registry. A method of creating a database of installed files outside of the software registry will be useful for auditing software installations.

By default, the `pkgchk` command uses the software registry as the database which contains file mode, ownership, size, and checksum. The `-m` option to `pkgchk` allows files to be checked against a `pkgmap` file. The `pkgmap` file describes the files in the package which are to be installed and is a required component of the package. This `pkgmap` file is typically generated by the `pkgmk` program.

See the `pkgmap(4)` man page for a description of the `pkgmap` format. There are many options in the `pkgmap` format for handling special file types such as editable or volatile files.

The `pkgproto` command accepts a list of files and generates most of the fields required for `pkgmap` files. However, `pkgproto` does not generate the file size, checksum, or the modification timestamp fields which are crucial for performing comprehensive audits. These fields are generated and placed in a package's `pkgmap` file by the `pkgmk` command. However, the `pkgmk` command also copies the file to a spool area. This makes the `pkgmk` command unusable for creating `pkgmap` files for configuration auditing.

A command is needed which can accept the `pkgproto` output and generate a `pkgmap` file. This `pkgmap` file can then be used by `pkgchk` for auditing configuration.

The pkgproto Command

The pkgproto command accepts a list of pathnames and generates one line per pathname. For normal files and directories, you can use shell commands such as test to determine their type. However, for other files such as hard or symbolic links, pkgproto builds the proper link target expected in the pkgmap file. It is best to use pkgproto to build the bulk of the pkgmap file information. For example:

```
# find /etc -print | pkgproto
d none /etc 0755 root sys
d none /etc/default 0755 root sys
f none /etc/default/sys-suspend 0444 root sys
f none /etc/default/cron 0555 root bin
...
l none /etc/rc0.d/K00ANNOUNCE=/etc/init.d/ANNOUNCE
...
```

Missing from the pkgproto output is the size in bytes, checksum, and modification timestamp of the files. The information on file mode and ownership is in the pkgproto output and can be used to check these attributes. The size and checksum information is needed to test whether the file has been modified. The modification timestamp can also be used to help determine if a file may require incremental backup or has been changed on a particular date. These attributes exist in pkgmap files generated by the pkgmk command.

The Trouble With pkgmk

The trouble with pkgmk for simply generating pkgmap files is that the size, checksum, and modification timestamp is determined as part of the procedure of copying the files to the package spool. For software audits, the copy of the file is not needed and is a waste of resources.

The pkgtrans command will extract a pkgmap file from a spooled package. Such a pkgmap file would be acceptable for use in software audits, but the use of pkgtrans requires that a package be already built and spooled. A procedure could be used to build a package using pkgproto and pkgmk.Then extract the pkgmap file using pkgtrans, and remove the package to recover the disk space consumed by the package. Such a procedure is cumbersome, CPU resource intensive, and requires significant disk space.

The Trouble With pkgproto

The trouble with pkgproto is that it makes rather simplistic decisions about the file type. The pkgmap format describes the following file types:

- Block special device - the major and minor numbers are described
- Character special device - the major and minor numbers are described
- Directory
- Regular file - the size, checksum, and modification timestamp are described
- Hard link - the link target is described
- Symbolic link - the link target is described
- Named pipes
- Editable file - a regular file which is expected to be edited after installation
- Volatile file - a regular file which contains changing information such as a log file
- Exclusive directory - rarely used, indicates a directory which is under the exclusive use of the software packaged

pkgproto will not automatically determine if a file is editable or volatile, nor will pkgproto make an exclusive directory entry. For software packages, the developer may utilize many of these file types in the package prototype file. These attributes will be propagated to the package's pkgmap file and into the software registry upon installation. However, for the purposes of a software audit, regular files and directories will suffice.

Making pkgmap Files With mkpkgmap.sh

A program is needed which will take the pkgproto output and create a pkgmap format file that can be used with pkgchk. "Code for mkpkgmap.sh" on page 4 shows a Bourne shell script which will accept pkgproto output and generate a pkgmap format file that is suitable for pkgchk.

CODE EXAMPLE 1 Code for mkpkgmap.sh

```
#!/bin/sh
#
# mkpkgmap.sh
#
# Script to generate a simplified pkgmap file which can be
```

CODE EXAMPLE 1 Code for `mkpkgmap.sh`

```
# used with pkgchk. Expected input is the output of pkgproto.
#
#ident "@(#)mkpkgmap.sh  1.4   00/01/02 SMI"

PATH=/usr/bin

while read line
do

  # For entries which are not files, prepend the part number
  #
  echo $line | egrep -s -e "^[bcdilpsx] "
  if [ $? = 0 ]; then
    echo "1 $line"
  fi

  # For entries which are files, get length, checksum,
  # and modification timestamp
  #
  echo $line | egrep -s -e "^[efv] "
  if [ $? = 0 ]; then

    # The filename is in the input line
    #
   filename=‘echo $line | awk ’{print $3}’‘

    # pkgchk cannot handle files whose filename contains an
    # equals sign, "=".  If found, print warning and ignore.
    #
    echo $filename | egrep -s -e ’=’
    if [ $? = 0 ]; then
     echo "WARNING: filename $filename contains ’=’, ignored" >&2
    else

      # Get the length from ls -l
      #
      length=‘ls -l $filename | awk ’{print $5}’‘

      # Get the crc checksum from sum
      #
      checksum=‘sum $filename | awk ’{print $1}’‘

      # checksum will be a null string if sum failed for some
      # reason, eg. file mode doesn’t allow reading. But
      # we still need to have a checksum entry, so set to 0
      #
```

CODE EXAMPLE 1 Code for `mkpkgmap.sh`

```
      [ -z "$checksum" ] && checksum=0

      # There is no simple way to retrieve the file modification
      # timestamp, but we can extract it from truss of the lstat
      # system call.
      #
      timestamp=‘truss -o /dev/stdout -tlstat -vlstat \
        ls -l $filename | awk ’/^.mt = / {print $9}’‘

      echo "1 $line $length $checksum $timestamp"
    fi
  fi
done
```

Software Audit Example

In this example, the root file system, `/`, is processed to generate a file named `root.pkgmap`. The `root.pkgmap` file is used as the reference to check the configuration at a later date. For illustration, a change is made in the root file system by running the `passwd` command and a check is performed which should identify the change. The exact password change is replaced by asterisks, ’*’, to protect the innocent. This example is run as the `root` user since the root file system is being checked and the `root` password is being changed.

```
# cd /tmp
# find / -mount -print | pkgproto | mkpkgmap.sh > root.pkgmap
# passwd root
new password: ********
Re-enter new password: ********
passwd (SYSTEM): passwd successfully changed for root
# pkgchk -m root.pkgmap
ERROR: /etc/shadow
    file cksum <19705> expected <12972> actual
```

In many Solaris Operating Environments, the /, /usr, /var, and /opt directories are located in different file systems. The -mount option to the find command will restrict find to only the initial file system checked. The following example performs a system software installation audit:

```
# df
/      (/dev/dsk/c0t0d0s0):  481710 blocks   145079 files
/usr   (/dev/dsk/c0t0d0s5):  684910 blocks   249982 files
/var   (/dev/dsk/c0t0d0s4):   15104 blocks    46950 files
/opt   (/dev/dsk/c0t1d0s0):  360164 blocks   462345 files
...
# find / -mount -print | pkgproto | mkpkgmap.sh > root.pkgmap
...
# find /usr -mount -print | pkgproto | mkpkgmap.sh >usr.pkgmap
...
# find /var -mount -print | pkgproto | mkpkgmap.sh >var.pkgmap
...
# find /opt -mount -print | pkgproto | mkpkgmap.sh >opt.pkgmap
...
# pkgchk -m root.pkgmap
...
# pkgchk -m usr.pkgmap
...
# pkgchk -m var.pkgmap
...
# pkgchk -m opt.pkgmap
...
```

The pkgmap files can be stored in common directory and the pkgchk audits can be run via the cron facility at regular intervals.

Note that the mkpkgmap.sh script may require significant amounts of CPU resources to generate the file checksum. A Solaris Operating Environment server configuration may contain tens of thousands of files in the / and /usr directories for which the checksum file must be generated.

Caveat Emptor

Unfortunately, pkgchk is not quite the perfect solution for software auditing. It has a few minor problems and annoying quirks.

Read permission is required for calculating checksum. The sum command used in mkpkgmap.sh must be able to read the file in order to calculate the checksum. Similarly, pkgchk must be able to read the file to calculate the checksum to be compared with the pkgmap file. Though normally you may not have permission to read all files on the system, as root, you can read all local files and perform a complete audit, including checksum, on locally mounted systems. mkpkgmap.sh and pkgchk will print errors when files are readable. These errors are harmless but distracting and indicate incomplete auditing. For remotely mounted file systems, the local root user may not be able to read all files, depending on the file sharing security policy of the server. Normal users can use mkpkmap.sh and pkgchk to perform audits on their home directories.

pkgchk *does not check modification timestamps.* The modification timestamps are readily available to pkgchk. However, pkgchk does not check them against the software registry or the pkgmap file.

Legal directory names may confuse pkgchk. pkgchk uses simple string comparisons to test path names against the software registry or pkgmap file. Path names which are semantically correct may not have unique names. For example, a legal directory path name may contain a trailing slash, '/'. A symbolic link to /mydirectory/ will be checked against /mydirectory and fail. pkgchk does not understand the semantics of the UNIX® file system and may not be able to properly check directories so named.

Legal filenames may confuse pkgchk. pkgchk expects that a linked file entry in the pkgmap file will have an equal sign, '=', followed by the target. pkgchk will get confused if a valid, regular file name includes an equal sign. Unfortunately, pkgchk will stop checking when such a filename is found in the pkgmap file. The mkpkgmap.sh script deals with this by printing a warning message and ignoring such files.

Conclusion

This article describes a Bourne shell script, mkpkgmap.sh, which can be used with pkgchk to perform software configuration audits. The user with root privileges can perform comprehensive audits on the system files and users may perform comprehensive audits on their files. This information is important for maintaining the software configuration of a Solaris Operating Environment system.

References

Solaris Operating Environment manual pages for: `find(1)`, `pkgchk(1)`, `pkgproto(1)`, `pkgmap(4)`, `sum(1)`, `truss(1)`

Solaris Operating Environment *Application Packaging Developer's Guide*, available in the AnswerBook and at `http://docs.sun.com`

Author's Bio: Richard Elling

Richard Elling is a Senior Engineer in Enterprise Engineering for the Computer Systems at Sun Microsystems in San Diego, California. Richard had been a field systems engineer at Sun for five years. He was the Sun Worldwide Field Systems Engineer of the Year in 1996. Prior to Sun, he was the Manager of Network Support for the College of Engineering at Auburn University, a design engineer for a startup microelectronics company, and worked for NASA doing electronic design and experiments integration for space shuttle missions.