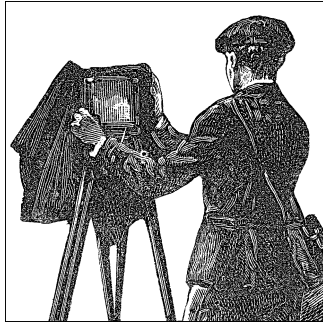


System Snapshots with Tripwire



1.0 Introduction

Suppose your system is infiltrated by the infamous Jack the Cracker. Being a conscientious evildoer, he quickly modifies some system files to create back doors and cover his tracks. For instance, he might substitute a hacked version of `/bin/login` to admit him without a password, and a bogus `/bin/ls` could skip over and hide traces of his evil deeds. If these changes go unnoticed, your system could remain secretly compromised for a long time. How can this situation be avoided?

Break-ins of this kind can be detected by an *integrity checker*: a program that periodically inspects important system files for unexpected changes. The *very first* security measure you should take when creating a new Linux machine, before you make it available to networks and other users, is to “snapshot” (record) the initial state of your system files with an integrity checker. If you don’t, you cannot reliably detect alterations to these files later. This is vitally important!

Tripwire is the best known open source integrity checker. It stores a snapshot of your files in a known state, so you can periodically compare the files against the snapshot to discover discrepancies. In our example, if `/bin/login` and `/bin/ls` were in Tripwire’s snapshot, then any changes in their size, inode number, permissions, or other attributes would catch Tripwire’s attention. Notably, Tripwire detects changes in a file’s *content*, even a single character, by verifying its checksum.



tripwire Version 1.2, supplied in SuSE 8.0, is positively ancient and supports an outdated syntax. Before attempting any recipes in this chapter, upgrade to the latest tripwire (2.3 or higher) at <http://sourceforge.org/projects/tripwire> or <http://www.tripwire.org>.

Tripwire is driven by two main components: a policy and a database. The *policy* lists all files and directories that Tripwire should snapshot, along with rules for identifying violations (unexpected changes). For example, a simple policy could treat any

changes in */root*, */bin*, and */lib* as violations. The Tripwire *database* contains the snapshot itself, created by evaluating the policy against your filesystems. Once setup is complete, you can compare filesystems against the snapshot at any time, and Tripwire will report any discrepancies. This is a Tripwire *integrity check*, and it generates an *integrity check report*, as in Figure 1-1.

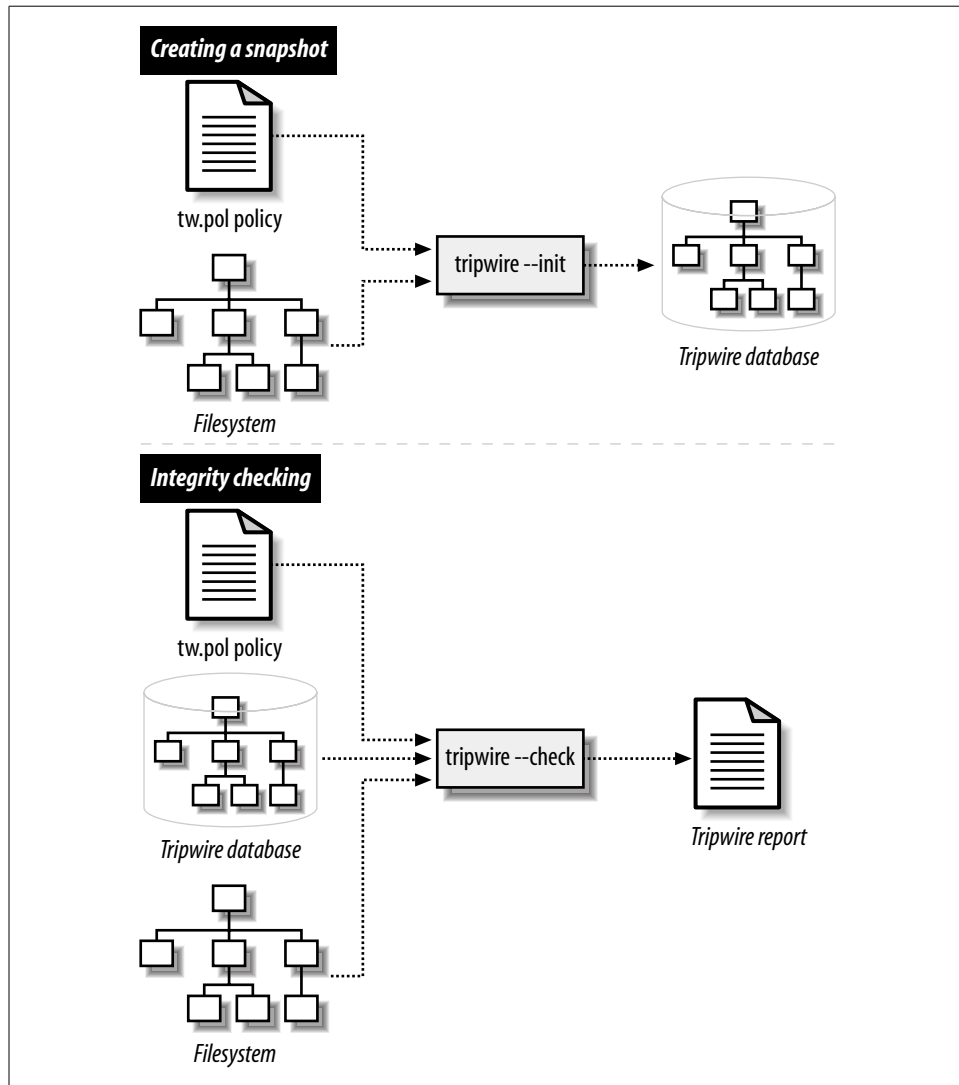


Figure 1-1. Creating a Tripwire snapshot, and performing an integrity check

Along with the policy and database, Tripwire also has a *configuration*, stored in a configuration file, that controls global aspects of its behavior. For example, the configuration specifies the locations of the database, policy file, and tripwire executable.

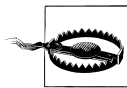
Important Tripwire-related files are encrypted and signed to prevent tampering. Two cryptographic keys are responsible for this protection. The *site key* protects the policy file and the configuration file, and the *local key* protects the database and generated reports. Multiple machines with the same policy and configuration may share a site key, whereas each machine must have its own local key for its database and reports.

Although Tripwire is a security tool, it can be compromised itself if you are not careful to protect its sensitive files. The most secret, quadruple-hyper-encrypted Tripwire database is useless if Jack the Cracker simply deletes it! Likewise, Jack could hack the `tripwire` executable (`/usr/sbin/tripwire`) or interfere with its notifications to the system administrator. Our recipes will describe several configurations—at increasing levels of paranoia and expense—to thwart such attacks.

Tripwire has several weaknesses:

- Its lengthy output can make your eyes glaze over, not the most helpful state for finding security violations.
- If you update your critical files frequently, then you must update the database frequently, which can be tiresome.
- Its batch-oriented approach (periodic checks, not real-time) leaves a window of opportunity. Suppose you modify a file, and then a cracker modifies it again before the next integrity check. Tripwire will rightfully flag the file, but you'll wrongly blame the discrepancy on your change instead of the cracker's. Your Tripwire database will be “poisoned” (contain invalid data) on the next update.
- It doesn't compile easily in some Linux and Unix environments.

Regardless, Tripwire can be a valuable security tool if used carefully and methodically.



Before connecting any Linux computer to a network, or making the machine available to other users in any way, TAKE A SNAPSHOT. We cannot stress this enough. A machine's first snapshot MUST capture a legitimate, uncompromised state or it is worthless. (That's why this topic is the *first* chapter in the book.)

In addition to Tripwire, we also present a few non-Tripwire techniques for integrity checking, involving `rpm` [1.15], `rsync` [1.16], and `find`. [1.17]

There are other integrity checkers around, such as Aide (<http://www.cs.tut.fi/~rammer/aide.html>) and Samhain (<http://la-samhna.de/samhain>), though we do not cover them. Finally, you might also check out runtime kernel integrity checkers, like `kstat` (<http://www.s0ftpj.org>) and `prosum` (<http://prosum.sourceforge.net>).

1.1 Setting Up Tripwire

Problem

You want to prepare a computer to use Tripwire for the first time.

Solution

After you have installed Tripwire, do the following:

```
# cd /etc/tripwire
# ./twinstall.sh
# tripwire --init
# rm twcfg.txt twpol.txt
```

Discussion

The script `twinstall.sh` performs the following tasks within the directory `/etc/tripwire`:

- Creates the site key and the local key, prompting you to enter their passphrases. (If the keys exist, this step is skipped.) The site key is stored in `site.key`, and the local key in `hostname-local.key`, where `hostname` is the hostname of the machine.
- Signs the default configuration file, `twcfg.txt`, with the site key, creating `tw.cfg`.
- Signs the default policy file, `twpol.txt`, with the site key, creating `tw.pol`.

If for some reason your system doesn't have `twinstall.sh`, equivalent manual steps are:

```
Helpful variables:
DIR=/etc/tripwire
SITE_KEY=$DIR/site.key
LOCAL_KEY=$DIR/`hostname`-local.key

Generate the site key:
# twadmin --generate-keys --site-keyfile $SITE_KEY

Generate the local key:
# twadmin --generate-keys --local-keyfile $LOCAL_KEY

Sign the configuration file:
# twadmin --create-cfgfile --cfgfile $DIR/tw.cfg \
  --site-keyfile $SITE_KEY $DIR/twcfg.txt

Sign the policy file:
# twadmin --create-polfile --cfgfile $DIR/tw.cfg \
  --site-keyfile $SITE_KEY $DIR/twpol.txt

Set appropriate permissions:
# cd $DIR
```

```
# chown root:root $SITE_KEY $LOCAL_KEY tw.cfg tw.pol
# chmod 600 $SITE_KEY $LOCAL_KEY tw.cfg tw.pol
```

(Or `chmod 640` to allow a root group to access the files.)

These steps assume that your default configuration and policy files exist: *twcfg.txt* and *twpol.txt*, respectively. They should have been supplied with the Tripwire distribution. Undoubtedly you'll need to edit them to match your system. [1.3] The names *twcfg.txt* and *twpol.txt* are mandatory if you run *twinstall.sh*, as they are hard-coded inside the script.*

Next, *tripwire* builds the Tripwire database and signs it with the local key:

```
# tripwire --init
```

Enter the local key passphrase to complete the operation. If *tripwire* produces an error message like “Warning: File System Error,” then your default policy probably refers to nonexistent files. These are not fatal errors: *tripwire* still ran successfully. At some point you should modify the policy to remove these references. [1.3]

The last step, which is optional but recommended, is to delete the plaintext (unencrypted) policy and configuration files:

```
# rm twcfg.txt twpol.txt
```

You are now ready to run integrity checks.

See Also

twadmin(8), *tripwire(8)*. If Tripwire isn't included in your Linux distribution, it can be downloaded from the Tripwire project page at <http://sourceforge.net/projects/tripwire> or <http://www.tripwire.org>. (Check both to make sure you're getting the latest version.) Basic documentation is installed in */usr/share/doc/tripwire** but does not include the full manual, so be sure to download it (in PDF or source formats) from the SourceForge project page. The commercial Tripwire is found at <http://www.tripwire.com>.

1.2 Displaying the Policy and Configuration

Problem

You want to view Tripwire's policy or configuration, but they are stored in non-human-readable, binary files, or they are missing.

* If they are different on your system, read *twinstall.sh* to learn the appropriate names.

Solution

Generate the active configuration file:

```
# cd /etc/tripwire
# twadmin --print-cfgfile > twcfg.txt
```

Generate the active policy file:

```
# cd /etc/tripwire
# twadmin --print-polfile > twpol.txt
```

Discussion

Tripwire's active configuration file *tw.cfg* and policy file *tw.pol* are encrypted and signed and therefore non-human-readable. To view them, you must first convert them to plaintext.

Tripwire's documentation advises you to delete the plaintext versions of the configuration and policy after re-signing them. If your plaintext files were missing to start with, this is probably why.

Although you can redirect the output of `twadmin` to any files you like, remember that `twinstall.sh` requires the plaintext policy and configuration files to have the names we used, *twcfg.txt* and *twpol.txt*. [1.1]

See Also

`twadmin(8)`.

1.3 Modifying the Policy and Configuration

Problem

You want to change the set of files and directories that tripwire examines, or change tripwire's default behavior.

Solution

Extract the policy and configuration to plaintext files: [1.2]

```
# cd /etc/tripwire
# twadmin --print-polfile > twpol.txt
# twadmin --print-cfgfile > twcfg.txt
```

Modify the policy file *twpol.txt* and/or the configuration file *twcfg.txt* with any text editor. Then re-sign the modified files: [1.1]

```
# twadmin --create-cfgfile --cfgfile /etc/tripwire/tw.cfg \
--site-keyfile site_key /etc/tripwire/twcfg.txt
```

```
# twadmin --create-polfile --cfgfile /etc/tripwire/tw.cfg \  
--site-keyfile site_key etc/tripwire/twpol.txt
```

and reinitialize the database: [1.1]

```
# tripwire --init  
# rm twcfg.txt twpol.txt
```

Discussion

This is much like setting up Tripwire from scratch [1.1], except our existing, cryptographically-signed policy and configuration files are first converted to plaintext. [1.2]

You'll want to modify the policy if tripwire complains that a file does not exist:

```
### Error: File could not be opened.
```

Edit the policy file and remove or comment out the reference to this file if it does not exist on your system. Then re-sign the policy file.

You don't need to follow this procedure if you're simply updating the database after an integrity check [1.11], only if you've modified the policy or configuration.

See Also

twadmin(8), tripwire(8).

1.4 Basic Integrity Checking

Problem

You want to check whether any files have been altered since the last Tripwire snapshot.

Solution

```
# tripwire --check
```

Discussion

This command is the lifeblood of Tripwire: has your system changed? It compares the current state of your filesystem against the Tripwire database, according to the rules in your active policy. The results of the comparison are written to standard output and also stored as a timestamped, signed Tripwire report.

You can also perform a limited integrity check against one or more files in the database. If your tripwire policy contains this rule:

```
(  
  rulename = "My funky files",  
  severity = 50
```

```

)
{
  /sbin/e2fsck          -> $(SEC_CRIT) ;
  /bin/cp              -> $(SEC_CRIT) ;
  /usr/tmp             -> $(SEC_INVARIANT) ;
  /etc/csh.cshrc      -> $(SEC_CONFIG) ;
}

```

you can check selected files and directories with:

```
# tripwire --check /bin/cp /usr/tmp
```

or all files in the given rule with:

```
# tripwire --check --rule-name "My funky files"
```

or all rules with severities greater than or equal to a given value:

```
# tripwire --check --severity 40
```

See Also

tripwire(8), and the Tripwire manual for policy syntax. You can produce a help message with:

```
$ tripwire --check --help
```

1.5 Read-Only Integrity Checking

Problem

You want to store Tripwire's most vital files on read-only media, such as a CD-ROM or write-protected disk, to guard against compromise, and then run integrity checks.

Solution

1. Copy the site key, local key, and tripwire binary onto the desired disk, write-protect it, and mount it. Suppose it is mounted at */mnt/cdrom*.

```

# mount /mnt/cdrom
# ls -l /mnt/cdrom
total 2564
-r--r----- 1 root  root      931 Feb 21 12:20 site.key
-r--r----- 1 root  root      931 Feb 21 12:20 myhost-local.key
-r-xr-xr-x  1 root  root    2612200 Feb 21 12:19 tripwire

```

2. Generate the Tripwire configuration file in plaintext: [1.2]

```

# DIR=/etc/tripwire
# cd $DIR
# twadmin --print-cfgfile > twcfg.txt

```

3. Edit the configuration file to point to these copies: [1.3]

```
/etc/tripwire/twcfg.txt:
ROOT=/mnt/cdrom
SITEKEYFILE=/mnt/cdrom/site.key
LOCALKEYFILE=/mnt/cdrom/myhost-local.key
```

4. Sign your modified Tripwire configuration file: [1.3]

```
# SITE_KEY=/mnt/cdrom/site.key
# twadmin --create-cfgfile --cfgfile $DIR/tw.cfg \
--site-keyfile $SITE_KEY $DIR/twcfg.txt
```

5. Regenerate the tripwire database [1.3] and unmount the CD-ROM:

```
# /mnt/cdrom/tripwire --init
# umount /mnt/cdrom
```

Now, whenever you want to perform an integrity check [1.4], insert the read-only disk and run:

```
# mount /mnt/cdrom
# /mnt/cdrom/tripwire --check
# umount /mnt/cdrom
```

Discussion

The site key, local key, and tripwire binary (*/usr/sbin/tripwire*) are the only files you need to protect from compromise. Other Tripwire-related files, such as the database, policy, and configuration, are signed by the keys, so alterations would be detected. (Back them up frequently, however, in case an attacker deletes them!)

Before copying */usr/sbin/tripwire* to CD-ROM, make sure it is statically linked (which is the default configuration) so it does not depend on any shared runtime libraries that could be compromised:

```
$ ldd /usr/sbin/tripwire
not a dynamic executable
```

See Also

twadmin(8), tripwire(8), ldd(1), mount(8).

1.6 Remote Integrity Checking

Problem

You want to perform an integrity check, but to increase security, you store vital Tripwire files off-host.



In this recipe and others, we use two machines: your original machine to be checked, which we'll call *trippy*, and a second, trusted machine we'll call *trusty*. *trippy* is the untrusted machine whose integrity you want to check with Tripwire. *trusty* is a secure machine, typically with no incoming network access.

Solution

Store copies of the site key, local key, and tripwire binary on a trusted remote machine that has no incoming network access. Use `rsync`, securely tunneled through `ssh`, to verify that the originals and copies are identical, and to trigger an integrity check.

The initial setup on remote machine *trusty* is:

```
#!/bin/sh
REMOTE_MACHINE=trippy
RSYNC='/usr/bin/rsync -a --progress --rsh=/usr/bin/ssh'
SAFE_DIR=/usr/local/tripwire/${REMOTE_MACHINE}
VITAL_FILES="/usr/sbin/tripwire
             /etc/tripwire/site.key
             /etc/tripwire/${REMOTE_MACHINE}-local.key"

mkdir $SAFE_DIR
for file in $VITAL_FILES
do
    $RSYNC ${REMOTE_MACHINE}:$file $SAFE_DIR/
done
```

Prior to running every integrity check on the local machine, verify these three files by comparing them to the remote copies. The following code should be run on *trusty*, assuming the same variables as in the preceding script (`REMOTE_MACHINE`, etc.):

```
#!/bin/sh
cd $SAFE_DIR
rm -f log
for file in $VITAL_FILES
do
    base=`basename $file`
    $RSYNC -n ${REMOTE_MACHINE}:$file . | fgrep -x "$base" >> log
done
if [ -s log ] ; then
    echo 'Security alert!'
else
    ssh ${REMOTE_MACHINE} -l root /usr/sbin/tripwire --check
fi
```

Discussion

`rsync` is a handy utility for synchronizing files on two machines. In this recipe we tunnel `rsync` through `ssh`, the Secure Shell, to provide secure authentication and to

encrypt communication between *trusty* and *trippy*. (This assumes you have an appropriate SSH infrastructure set up between *trusty* and *trippy*, e.g., [6.4]. If not, *rsync* can be used insecurely without SSH, but we don't recommend it.)

The `--progress` option of *rsync* produces output only if the local and remote files differ, and the `-n` option causes *rsync* not to copy files, merely reporting what it would do. The `fgrep` command removes all output but the filenames in question. (We use `fgrep` because it matches fixed strings, not regular expressions, since filenames commonly contain special characters like “.” found in regular expressions.) The `fgrep -x` option matches whole lines, or in this case, filenames. Thus, the file *log* is empty if and only if the local and remote files are identical, triggering the integrity check.

You might be tempted to store the Tripwire database remotely as well, but it's not necessary. Since the database is signed with the local key, which is kept off-host, *tripwire* would alert you if the database changed unexpectedly.

Instead of merely checking the important Tripwire files, *trusty* could copy them to *trippy* before each integrity check:

```
# scp -p tripwire trippy:/usr/sbin/tripwire
# scp -p site.key trippy-local.key trippy:/etc/tripwire/
# ssh trippy -l root /usr/sbin/tripwire --check
```

Another tempting alternative is to mount *trippy*'s disks remotely on *trusty*, preferably read-only, using a network filesystem such as NFS or AFS, and then run the Tripwire check on *trusty*. This method, however, is only as secure as your network filesystem software.

See Also

`rsync(1)`, `ssh(1)`.

1.7 Ultra-Paranoid Integrity Checking

Problem

You want highly secure integrity checks, at the expense of speed and convenience.

Solution

Securely create a bootable CD-ROM containing a minimal Linux system, the *tripwire* binary, and your local and site keys. Disconnect your computer from all networks, boot on the CD-ROM, and perform an integrity check of your computer's disks, using executable programs on the CD-ROM only.

Back up your Tripwire database, configuration, and policy frequently, in case an attacker deletes them from your system.

Discussion

This cumbersome but more secure method requires at least two computers, one of them carefully trusted. As before, we'll call the trusted system *trusty* and the Tripwire machine *trippy*. Our goal is to run secure Tripwire checks on *trippy*.

The first important step is to create a bootable CD-ROM securely. This means:

- Create the CD-ROM on *trusty*, a virgin Linux machine built directly from trusted source or binary packages, that has never been on a network or otherwise accessible to third parties. Apply all necessary security patches to bring *trusty* up to date.
- Configure the CD-ROM's startup scripts to disable all networking.
- Populate the CD-ROM directly from trusted source or binary packages.
- Create your Tripwire site key and local key on *trusty*.

Second, boot *trippy* on the CD-ROM, mount the local disks, and create *trippy*'s Tripwire database, using the *tripwire* binary and keys on the CD-ROM. Since the Tripwire database, policy, and configuration files are signed with keys on the CD-ROM, these files may safely reside on *trippy*, rather than the CD-ROM.

Third, you must boot *trippy* on the CD-ROM before running an integrity check. Otherwise, if you simply mount the CD-ROM on *trippy* and run the *tripwire* binary from the CD-ROM, you are not protected against:

- Compromised shared libraries on *trippy*, if your *tripwire* binary is dynamically linked.
- A compromised Linux kernel on *trippy*.
- A compromised mount point for the CD-ROM on *trippy*.

See, we told you this recipe was for the paranoid. But if you want higher security with Tripwire, you might need this level of caution.

For more convenience, you could schedule a cron job to reboot *trippy* nightly from the CD-ROM, which runs the Tripwire check and then reboots *trippy* normally. Do not, however, schedule this cron job on *trippy* itself, since cron could be compromised. Instead, schedule it on *trusty*, perhaps triggering the reboot via an SSH batch job. [6.10]

See Also

A good starting point for making a self-contained bootable CD-ROM or floppy is *tomsrtbt* at <http://www.toms.net/rb>.

Consider including post-mortem security tools on the CD-ROM, such as the Coroner's Toolkit. [9.41]

1.8 Expensive, Ultra-Paranoid Security Checking

Problem

You want highly secure integrity checks and are willing to shell out additional money for them.

Solution

Store your files on a dual-ported disk array. Mount the disk array read-only on a second, trusted machine that has no network connection. Run your Tripwire scans on the second machine.

Discussion

A dual-ported disk array permits two machines to access the same physical disk. If you've got money to spare for increased security, this might be a reasonable approach to securing Tripwire.

Once again, let *trippy* be your machine in need of Tripwire scans. *trusty* is a highly secure second machine, built directly from trusted source or binary packages with all necessary security patches applied, that has no network connection and has never been accessible to third parties.

trippy's primary storage is kept on a dual-ported disk array. Mount this array on *trusty* read-only. Perform all Tripwire-related operations on *trusty*: initializing the database, running integrity checks, and so forth. The Tripwire database, binaries, keys, policy, and configuration are likewise kept on *trusty*. Since *trusty* is inaccessible via any network, your Tripwire checks will be as reliable as the physical security of *trusty*.

1.9 Automated Integrity Checking

Problem

You want to schedule integrity checks at specific times or intervals.

Solution

Use cron. For example, to perform an integrity check every day at 3:00 a.m.:

```
root's crontab file:  
0 3 * * * /usr/sbin/tripwire --check
```

Discussion

This is not a production-quality recipe. An intruder could compromise cron, substituting another job or simply preventing yours from running. For more reliability, run the cron job on a trusted remote machine:

Remote crontab entry on trusty:

```
0 3 * * * ssh -n -l root trippy /usr/sbin/tripwire --check
```

but if an intruder compromises sshd on *trippy*, you're again out of luck. Likewise, some rootkits [9.12] can subvert the exec call to tripwire even if invoked remotely. For maximum security, run not only the cron job but also the integrity check on a trusted remote machine. [1.8]

Red Hat Linux comes preconfigured to run tripwire every night via the cron job */etc/cron.daily/tripwire-check*. However, a Tripwire database is not supplied with the operating system: you must initialize one yourself. [1.1] If you don't, cron will send daily email to root about a failed tripwire invocation.

See Also

tripwire(8), crontab(1), crontab(5), cron(8).

1.10 Printing the Latest Tripwire Report

Problem

You want to display the results of the most recent integrity check.

Solution

```
#!/bin/sh
DIR=/var/lib/tripwire/report
HOST=`hostname -s`
LAST_REPORT=`ls -1t $DIR/$HOST-*.twr | head -1`
twprint --print-report --twrfile "$LAST_REPORT"
```

Discussion

Tripwire reports are stored in the location indicated by the REPORTFILE variable in the Tripwire configuration file. A common value is:

```
REPORTFILE = /var/lib/tripwire/report/${HOSTNAME}-${DATE}.twr
```

The HOSTNAME variable contains the hostname of your machine (big surprise), and DATE is a numeric timestamp such as 20020409-040521 (April 9, 2002, at 4:05:21). So for host *trippy*, this report filename would be:

```
/var/lib/tripwire/report/trippy-20020409-040521.twr
```

When tripwire runs, it can optionally send reports by email. This notification should not be considered reliable since email can be suppressed, spoofed, or otherwise compromised. Instead, get into the habit of examining the reports yourself.

The `twprint` program can print reports not only for integrity checks but also for the Tripwire database. To do the latter:

```
# twprint --print-dbfile --dbfile /var/lib/tripwire/`hostname -s`.twd
Tripwire(R) 2.3.0 Database
Database generated by:      root
Database generated on:     Mon Apr  1 22:33:52 2002
Database last updated on:  Never
... contents follow ...
```

See Also

`twprint(8)`.

1.11 Updating the Database

Problem

Your latest Tripwire report contains discrepancies that tripwire should ignore in the future.

Solution

Update the Tripwire database relative to the most recent integrity check report:

```
#!/bin/sh
DIR=/var/lib/tripwire/report
HOST=`hostname -s`
LAST_REPORT=`ls -1t $DIR/$HOST-*.twr | head -1`
tripwire --update --twrfile "$LAST_REPORT"
```

Discussion

Updates are performed with respect to an integrity check report, not with respect to the current filesystem state. Therefore, if you've modified some files since the last check, you cannot simply run an update: you must run an integrity check first. Otherwise the update won't take the changes into account, and the next integrity check will still flag them.

Updating is significantly faster than reinitializing the database. [1.3]

See Also

`tripwire(8)`.

1.12 Adding Files to the Database

Problem

Tell tripwire to include a file or directory in its database.

Solution

Generate the active policy file in human-readable format. [1.2] Add the given file or directory to the active policy file.

To mark the file */bin/ls* for inclusion:

```
/bin/ls --> $(SEC_BIN) ;
```

To mark the entire directory tree */etc* for inclusion:

```
/etc --> $(SEC_BIN) ;
```

To mark */etc* and its files, but not recurse into subdirectories:

```
/etc --> $(SEC_BIN) (recurse=1) ;
```

To mark only the */etc* directory but none of its files or subdirectories:

```
/etc --> $(SEC_BIN) (recurse=0);
```

Then reinitialize the database. [1.3]

Discussion

The policy is a list of rules stored in a policy file. A rule looks like:

```
filename -> rule ;
```

which means that the given file (or directory) should be considered compromised if the given rule is broken. For instance,

```
/bin/login -> +pisug ;
```

means that */bin/login* is suspect if its file permissions (p), inode number (i), size (s), user (u), or group (g) have changed since the last snapshot. We won't document the full policy syntax because Tripwire's manual is quite thorough. Our recipe uses a predefined rule in a global variable, SEC_BIN, designating a binary file that should not change.

The *recurse=n* attribute for a directory tells tripwire to recurse *n* levels deep into the filesystem. Zero means to consider only the directory file itself.

It's actually quite likely that you'll need to modify the policy. The default policy supplied with Tripwire is tailored to a specific type of system or Linux distribution, and contains a number of files not necessarily present on yours.

See Also

The Tripwire manual has detailed documentation on the policy file format.

1.13 Excluding Files from the Database

Problem

You want to add some, but not all, files in a given directory to the Tripwire database.

Solution

Mark a directory hierarchy for inclusion:

```
/etc -> rule
```

Immediately after, mark some files to be excluded:

```
!/etc/not.me  
!/etc/not.me.either
```

You can exclude a subdirectory too:

```
!/etc/dirname
```

Discussion

The exclamation mark (!) prevents the given file or subdirectory from being added to Tripwire's database.

See Also

The Tripwire manual has detailed documentation on the policy file format.

1.14 Checking Windows VFAT Filesystems

Problem

When checking the integrity of a VFAT filesystem, tripwire always complains that files have changed when they haven't.

Solution

Tell tripwire not to compare inode numbers.

```
filename -> rule -i ;
```

For example:

```
/mnt/windows/system -> $(SEC_BIN) -i ;
```

Discussion

Modern Linux kernels do not assign constant inode numbers in VFAT filesystems.

See Also

The Tripwire manual has detailed documentation on the policy file format.

1.15 Verifying RPM-Installed Files

Problem

You have installed some RPM packages, perhaps long ago, and want to check whether any files have changed since the installation.

Solution

```
# rpm -Va [packages]
```

Debian Linux has a similar tool called `debsums`.

Discussion

If your system uses RPM packages for installing software, this command conveniently compares the installed files against the RPM database. It notices changes in file size, ownership, timestamp, MD5 checksum, and other attributes.

The output is a list of (possibly) problematic files, one per line, each preceded by a string of characters with special meaning. For example:

```
$ rpm -Va
SM5....T c /etc/syslog.conf
.M..... /var/lib/games/trojka.scores
missing  /usr/lib/perl5/5.6.0/Net/Ping.pm
..?..... /usr/X11R6/bin/XFree86
.....U.. /dev/audio
S.5....T /bin/ls
```

The first line indicates that *syslog.conf* has an unexpected size (S), permissions (M), checksum (5), and timestamp (T). This is perhaps not surprising, since *syslog.conf* is a configuration file you'd be likely to change after installation. In fact, that is exactly what the "c" means: a configuration file. Similarly, *trojka.scores* is a game score file likely to change. The file *Ping.pm* has apparently been removed, and *XFree86* could not be checked (?) because we didn't run `rpm` as root. The last two files definitely deserve investigation: */dev/audio* has a new owner (U), and */bin/ls* has been modified.

This technique is valid only if your RPM database and `rpm` command have not been compromised by an attacker. Also, it checks only those files installed from RPMs.

See Also

`rpm(8)` lists the full set of file attributes checked.

1.16 Integrity Checking with rsync

Problem

You want to snapshot and check your files but you can't use Tripwire. You have lots of disk space on a remote machine.

Solution

Use `rsync` to copy your important files to the remote machine. Use `rsync` again to compare the copies on the two machines.

Discussion

Let *trippy* and *trusty* be your two machines as before. You want to ensure the integrity of the files on *trippy*.

1. On *trippy*, store the `rsync` binary on a CD-ROM mounted at `/mnt/cdrom`.
2. On *trusty*, copy the files from *trippy*:

```
trusty# rsync -a -v --rsync-path=/mnt/cdrom/rsync --rsh=/usr/bin/ssh \  
trippy:/ /data/trippy-backup
```

3. Check integrity from *trusty*:

```
trusty# rsync -a -v -n --rsync-path=/mnt/cdrom/rsync --rsh=/usr/bin/ssh \  
trippy:/ /data/trippy-backup
```

The first `rsync` actually performs copying, while the second merely reports differences, thanks to the `-n` option. If there are no differences, the output will look something like this:

```
receiving file list ... done  
wrote 16 bytes  read 7478 bytes  4996.00 bytes/sec  
total size is 3469510  speedup is 462.97
```

but if any files differ, their names will appear after the “receiving file list” message:

```
receiving file list ... done  
/bin/ls  
/usr/sbin/sshd  
wrote 24 bytes  read 7486 bytes  5006.67 bytes/sec  
total size is 3469510  speedup is 461.99
```

Any listed files—in this case `/bin/ls` and `/usr/sbin/sshd`—should be treated as suspicious. This method has important limitations, most notably that it does not check inode numbers or device numbers. A real integrity checker is better.

See Also

`rsync(1)`.

1.17 Integrity Checking Manually

Problem

You can't use Tripwire for administrative or political reasons, but you want to snapshot your files for later comparison. You don't have enough disk space to mirror your files.

Solution

Run a script like the following that stores pertinent information about each file of interest, such as checksum, inode number, and timestamp:

```
#!/bin/sh
for file
do
    date=`/usr/bin/stat "$file" | /bin/grep '^Modify:' | /usr/bin/cut -f2- -d' '`
    sum=`/usr/bin/md5sum "$file" | /usr/bin/awk '{print $1}'`
    inode=`/bin/ls -id "$file" | /usr/bin/awk '{print $1}'`
    /bin/echo -e "$file\t$inode\t$sum\t$date"
done
```

Store this script as `/usr/local/bin/idfile` (for example). Use `find` to run this script on your important files, creating a snapshot. Store it on read-only media. Periodically create a new snapshot and compare the two with `diff`.

This is not a production-quality integrity checker. It doesn't track file ownership or permissions. It checks only ordinary files, not directories, device special files, or symbolic links. Its tools (`md5sum`, `stat`, etc.) are not protected against tampering.

Discussion

1. Run the `idfile` script to create a snapshot file:

```
# find /dir -xdev -type f -print0 | \
xargs -0 -r /usr/local/bin/idfile > /tmp/my_snapshot
```

This creates a snapshot file, basically a poor man's Tripwire database.

```
/bin/arch222      7ba4330c353be9dd527e7eb46d27f923Wed Aug 30 17:54:25 2000
/bin/ash 2194     cef0493419ea32a7e26eceff8e5dfa90Wed Aug 30 17:40:11 2000
```

```
/bin/awk 2171      b5915e362f1a33b7ede6d7965a4611e4Sat Feb 23 23:37:18 2002
...
```

Note that `idfile` will process `/tmp/my_snapshot` itself, which will almost certainly differ next time you snapshot. You can use `grep -v` to eliminate the `/tmp/my_snapshot` line from the output.

Be aware of the important options and limitations of `find`. [9.8]

2. In preparation for running the `idfile` script later from CD-ROM, modify `idfile` so all commands are relative to `/mnt/cdrom/bin`:

```
#!/mnt/cdrom/bin/sh
BIN=/mnt/cdrom/bin
for file
do
    date=`$BIN/stat "$file" | $BIN/grep '^Modify:' | $BIN/cut -f2- -d' '`
    md5sum=`$BIN/sum "$file" | $BIN/awk '{print $1}'`
    inode=`$BIN/ls -id "$file" | $BIN/awk '{print $1}'`
    $BIN/echo -e "$file\t$inode\t$sum\t$date"
done
```

3. Burn a CD-ROM with the following contents:

Directory	Files
/	my_snapshot
/bin	awk, cut, echo, diff, find, grep, ls, mdsum, sh, stat, xargs, idfile

4. Mount the CD-ROM at `/mnt/cdrom`.
5. As needed, rerun the `find` and do a `diff`, using the binaries on the CD-ROM:

```
#!/bin/sh
BIN=/mnt/cdrom/bin
$BIN/find /dir -xdev -type f -print0 | \
    xargs -0 -r $BIN/idfile > /tmp/my_snapshot2
$BIN/diff /tmp/my_snapshot2 /mnt/cdrom/my_snapshot
```

This approach is not production-quality; it has some major weaknesses:

- Creating the snapshot can be very slow, and creating new snapshots frequently may be cumbersome.
- It doesn't check some important attributes of a file, such as ownership and permissions. Tailor the `idfile` script to your needs.
- It checks only ordinary files, not directories, device special files, or symbolic links.
- By running `ls`, `md5sum`, and the other programs in sequence, you leave room for race conditions during the generation of the snapshot. A file could change between the invocations of two of these tools.

- If any of the executables are dynamically linked against libraries on the system, and these libraries are compromised, the binaries on the CD-ROM can theoretically be made to operate incorrectly.
- If the mount point */mnt/cdrom* is compromised, your CD-ROM can be spoofed.

See Also

find(1), diff(1). Use a real integrity checker if possible. If you can't use Tripwire, consider Aide (<http://www.cs.tut.fi/~rammer/aide.html>) or Samhain (<http://la-samhna.de/samhain>).