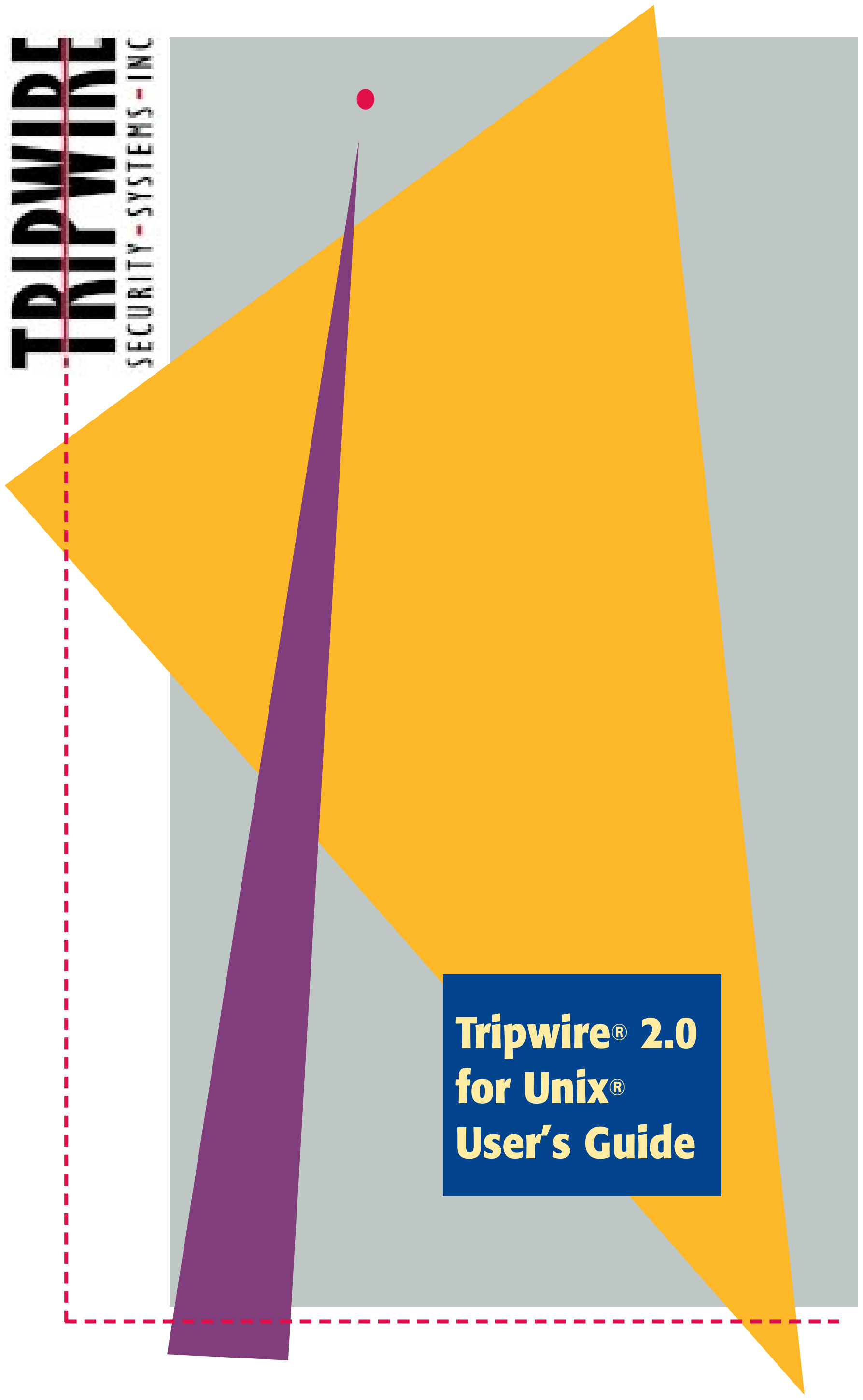

Tripwire™ Security Systems, Inc.
The Berg Building
615 SW Broadway, Second Floor
Portland, OR 97205, USA
tel: 503.223.0280
fax: 503.223.0182
(www.tripwiresecurity.com)
Email: tripwire@tripwiresecurity.com

We recommend that you read and follow the steps below regardless of your previous Tripwire experience. This document will help you to quickly and properly configure, install, and use Tripwire.

Installing Tripwire

1. Read the README and Release Notes for the latest Tripwire information.
2. Read the Installation chapter of the User's Manual.
3. Copy the Tripwire install configuration file (*install.cfg*) from the distribution CD to the hard disk of the machine on which you wish to install the product.
4. Modify *install.cfg* to specify appropriate installation paths for your system.
5. Run **./install.sh** [*path*]**/install.cfg**

Configuring Tripwire

6. Read the Running Tripwire chapter of the User's Manual for an overview of Tripwire fundamentals.
7. Read the Tripwire Policy File chapter of the User's Manual.
8. Modify the default policy file, or create your own.
9. Read the Tripwire Configuration File chapter of the User's Manual.
10. Modify the Tripwire configuration file.

Running Tripwire

(The next steps assume you are working from Tripwire's /*bin* directory. Change paths and filenames as appropriate)

11. Install your customized configuration file:
    ./twadmin --create-cfgfile --site-keyfile ../key/site.key twcfg.txt
12. Install your customized policy file:
    ./twadmin --create-polfile ../policy/twpol.txt
13. Initialize your Tripwire database: **./tripwire --init**

Congratulations! If you successfully completed these steps, then Tripwire is set and ready to go.

Dear Tripwire customer,

As you may know, Purdue University has elected to transfer management and product responsibility for the Tripwire security system to its co-developer Mr. Gene Kim and his firm Tripwire™ Security Systems, Inc. of Portland, Oregon, where he is now Chief Technology Officer.

This decision was made to ensure the continued development and support of Tripwire and to maintain the integrity of its design goals.

The results of the arrangement will benefit you, Purdue University, and Tripwire Security Systems by allowing the product to evolve and receive the necessary focus and resources needed to maintain its functionality in years to come.

Furthermore, by placing Tripwire with a focused commercial entity, we believe that there will be resources and motivation to enhance and port Tripwire to a wider range of uses and platforms. This can only help the user community concerned with security.

As a business, Tripwire Security Systems will be able to invest the time and resources for quality technical support. They will also be in a better position to respond to customer suggestions and integrate valuable new features into future Tripwire releases.

Thank you for making Tripwire one the most successful security programs ever written. I look forward to reviewing the progress of our decision to focus the resources needed to further advance the program´s functionality and value.

Gene Spafford, Ph.D.

The following typographic conventions are used in this manual:

Italic	is used for file and command names, and to denote terms the first time they are used.
Bold	is used for ftp and http URLs.
`Fixed Width`	is used in examples to show text that is entered literally, and in regular text to show variables, operators, and the output from commands or programs.
`Fixed Bold`	is used in examples to show actual user input on the command line.
`Fixed Italic`	is used in examples to show variables which should be replaced with a relevant value.

The `%` prompt is used for all command-line examples.

The `./` is specified for all Tripwire commands. This convention is recommended for all users, to ensure that Tripwire commands are executed from the appropriate directory, and to protect against Trojan Horse attacks.

1

INSTALLING TRIPWIRE

Introduction: Installing Tripwire is a simple process, whether you have a single machine or hundreds of networked machines. To enhance your understanding of the installation process, we have included both an overview, to demonstrate what you can expect to see, and installation instructions to take you step-by-step through the installation process.

Chapter contents:

Installation overview

The Tripwire installation CD contains two complete distributions of Tripwire 2.0 for Unix, one for Solaris SPARC™ and one for RedHat Intel Linux™. The root directory of the CD contains the README file and release notes. If you have not yet read the README, it is recommended that you do so now.

In addition to the documentation, each distribution of Tripwire will have its own directory. The directories are named *Solaris* and *RedHat*. Change to the directory appropriate for your operating system. You will see the following files and directories:

Tripwire installation files

install.cfg is the *installation configuration file*, a Bourne shell script used by the installer to set configuration variables. These settings specify the target directories where the installer will copy files and the desired behavior when existing Tripwire files would be overwritten.

install.sh is the *installation script*, which you run to begin installation. You can specify that it should read the default configuration file, or one that you customized for your site.

pkg is a directory containing files that the installation script needs to install Tripwire on your machine. These files are used exclusively by the installer and should not be modified.

README and *Release_Notes* are text files providing last-minute product information. These are identical to the files of the same names in the CD root directory.

Configuring your installation

All installation options are specified as command line arguments to the install script *install.sh*, or as settings in the configuration file.

An example of command line arguments for an unattended install is shown below.

```
% ./install.sh /tmp/install.cfg -s "Darth4Vader" -l "Sky8Walker" -f -n
```

This table summarizes installer command line options.

Argument	Meaning
[*configfile*]	Use the specified file for installation values. By default, the installer uses the values in *./install.cfg* for installation options.
[-n]	Suppress prompting. By default, the installer will display all the target directories that will be created and populated, and prompt the user for verification before proceeding. This mode requires the site and local passphrase arguments.
[-s *passphrase*]	Use the specified passphrase for site key.
[-l *passphrase*]	Use the specified passphrase for local key.
[-f]	Specifies that the installer should overwrite any existing files found in the target directories. This will override the CLOBBER setting in the *install.cfg* file.

An example of settings found in the configuration file is shown below:

```
# The root of the TSS directory tree.
TWROOT="/usr/TSS"

# Tripwire binaries are stored in TWBIN.
TWBIN="${TWROOT}/bin"
```

Target directories

The table below shows each installer target directory, as well as its default setting.

Key	Default Value	Description
TWROOT	`/usr/TSS`	The root directory for the Tripwire distribution. The default installation places all Tripwire files underneath this directory.
TWBIN	`${TWROOT}/bin`	Program executables. Contains all Tripwire program executables and the Tripwire configuration file.
TWPOLICY	`${TWROOT}/policy`	Policy files. Contains the Tripwire policy files.
TWMAN	`${TWROOT}/man`	Manual pages. Contains the Tripwire manual pages.
TWDB	`${TWROOT}/db`	Database files. Contains all the Tripwire database files, created from the system policy files.
TWSITEKEYDIR	`${TWROOT}/key`	Site key. Contains the site cryptographic key used by Tripwire to secure configuration and policy files.
TWLOCALKEYDIR	`${TWROOT}/key`	Local key. Contains the local cryptographic key used by Tripwire to secure database files and reports.
TWREPORT	`${TWROOT}/report`	Generated reports. Contains output of Tripwire integrity checks. Each integrity checking run will create a file in this directory for archival purposes.
CLOBBER	`false`	Specifies whether the installer will overwrite existing files. By default, the installer will not overwrite existing files. When files would be overwritten, a warning is printed and the file copy is skipped. An exception is the configuration and policy files. To insure a self-consistent installation, new copies of these files are always produced. Any existing configuration or policy files are saved with a *.bak* extension. Replace `false` with `true` if you wish to overwrite all files without warning.

With the default settings shown above, the installer will create the following directory tree on your system.

```
/usr
        /TSS                                                  <-TWROOT
                README
                Release_Notes

                /bin                                          <-TWBIN
                                siggen
                                tripwire
                                tw.cfg
                                twcfg.txt
                                twadmin
                                twprint

                /db                                           <-TWDB

                /key                                          <-TWSITEKEYDIR
                                site.key

                /key                                          <-TWLOCALKEYDIR
                                [machinename]-local.key

                /policy                                       <-TWPOLICY
                                tw.pol
                                twpol.txt
                                policyguide.txt

                /report                                       <-TWREPORT

                /man                                          <-TWMAN
                                /man4
                                        twconfig.4
                                        twpolicy.4

                                /man5
                                        twfiles.5

                                /man8
                                        siggen.8
                                        tripwire.8
                                        twadmin.8
                                        twintro.8
                                        twprint.8
```

Tripwire cryptographic keys

After the installation script has copied all files to your machine, Tripwire will sign sensitive files using public/private key cryptography. This prevents an intruder from modifying your policy, configuration, and database files. Tripwire uses two separate key files for this operation.

Keyfile types

Your *site key* is used to sign Tripwire files that can be shared among many machines. This key is used for the configuration and policy files.

Your *local key* is used to sign Tripwire files that are machine dependent. This key is used for the database files, and optionally for the report files.

Tripwire's site and local keys are encoded using passphrases chosen by the user. A *passphrase* is simply another name for "password," implying that passphrases should be longer than a typical English word. Your passphrases should be chosen carefully and should exhibit the following characteristics:

- Each passphrase should be at least 8 characters. (Note: no constraint checking is done.)
- Passphrases should include upper and lower case letters.
- For maximum security, we recommend that the passphrases not be words that you would find in a dictionary. Using a combination of letters and numbers is a good strategy.

It is recommended that the site key and the local key not use the same passphrase. With two passphrases, an intruder who compromises the local key on one machine does not necessarily have the ability to compromise all other machines.

Please refer to Chapter 6 for more information on Tripwire's cryptographic system.

Installing Tripwire

After customizing the installation configuration file, you are ready to run the install script. The installer creates and copies Tripwire files into the target directories. It also ensures that file and directory permissions are set correctly and creates the Tripwire site and local key files.

Starting the install

To start the installation, go to the directory containing the Tripwire installation files. Typically, this is either the subdirectory on the Tripwire CD appropriate for your OS, or a directory containing a copy of the Tripwire CD.

Run the installation script by typing:

```
 % ./install.sh
```

By default, the installation script reads the configuration file named *install.cfg* located in the same directory. However, you can specify an alternate configuration file on the command line. This argument is required if you customize the configuration file and are installing from read-only media.

```
% ./install.sh /tmp/install.cfg
```

Security issue: The *install.cfg* file is a Bourne shell script that is executed by the install script. Therefore, this file should be inspected before running to prevent a Trojan Horse attack. (One method of attacking a system is to replace the "harmless" contents of a file, such as an install script, with malicious instructions. Thus the user is tricked into running a compromised file. This is commonly known as a Trojan Horse attack.)

Creating and copying files

The install script will print an opening banner and list all directories that will be created:

```
Installer program for:
Tripwire(tm) 2.0 for Unix

Tripwire(tm) Copyright 1992-99 by the Purdue Research Foundation of Purdue
University, and distributed by Tripwire Security Systems, Inc. under
exclusive license arrangements.

Using configuration file install.cfg

This program will copy Tripwire files to the following directories:

         TWROOT:   /usr/TSS
          TWBIN:   /usr/TSS/bin
       TWPOLICY:   /usr/TSS/policy
          TWMAN:   /usr/TSS/man
       TWREPORT:   /usr/TSS/report
           TWDB:   /usr/TSS/db
   TWSITEKEYDIR:   /usr/TSS/key
  TWLOCALKEYDIR:   /usr/TSS/key

CLOBBER is false.

Continue with installation? [y/n]
```

If any of the install target directories already exists, files could potentially be overwritten. You should be certain before proceeding that overwriting Tripwire files in the directories listed is acceptable.

Inspect the list of directories that the installer will use. If you are satisfied that the list is correct, type “y” to proceed with the installation.

Generating cryptographic keys and signed files

When file copying is completed, the install script creates the site and local key files. This process may take several minutes. If you have not specified passphrases on the installer command line, you will be prompted for the site and local passphrases. At the end of this process, the installer will create signed policy and configuration files.

A clear text copy of the Tripwire configuration file is preserved in the $TWBIN directory as *twcfg.txt*. If you wish to change configuration settings, you can modify this file, and produce a new encoded/signed *tw.cfg file*.

By default, the installer creates a sample policy file called *tw.pol*. A clear text copy of this policy file is preserved in the policy directory as *twpol.txt*. This file is heavily commented and is specific to the operating system where Tripwire is being installed. An additional commented policy file, *policyguide.txt*, is installed in the policy directory. It illustrates all features of the policy language.

These initial policy files are intended only to confirm basic functionality of Tripwire. Once installation is complete, you should create an appropriate policy file to replace the sample file.

Security Issue: For added security, it is recommended that clear text copies of policy and configuration files not be stored on systems where Tripwire is deployed.

After installing

Congratulations!

You have successfully completed the Tripwire installation. In most cases, your next step is to modify your policy and configuration files, and then initialize the Tripwire database. Please proceed to the next chapter for more information on these tasks.

2 RUNNING TRIPWIRE

Introduction:

This chapter is a primer on fundamental Tripwire concepts. It provides a walkthrough of Tripwire operations, showing examples of policy file creation and updating, and report generation. Examples of other useful Tripwire features are also presented.

Chapter contents:

Tripwire files

Tripwire run-time operation is affected by the configuration, policy, and database files. Each of these files allows you to control a separate aspect of Tripwire behavior. Together, these files work together to ensure the security and integrity of your system.

The *configuration file* stores system-specific information, such as the location of the Tripwire data files. Some of this information is generated by the Tripwire installation script, but other parameters may need to be changed by the system administrator. See Chapter 3, Configuration Reference, for more details.

The *policy file* describes the system objects to be monitored, and the allowed changes. Should unexpected changes occur, the policy file can describe the person to be notified and the severity of the violation. See Chapter 4, Policy Reference, for more details.

The *database file* serves as the baseline for integrity checking. After the database is created, the Tripwire integrity checker compares each system object in the policy file against its actual entry in the database. If an object has changed outside of constraints defined in the policy file, a violation is reported.

Once the three files have been created, Tripwire can generate reports that describe the differences between the actual system and the data contained in the Tripwire database. This information is archived into *report files*, a collection of rule violations discovered during an integrity checking run.

It is critical that Tripwire files be protected—an attacker who is able to modify these files can subvert Tripwire operation. For this reason, all of the files above are signed using public key cryptography to prevent unauthorized modification.

Tripwire cryptographic keys

Two separate sets of keys generated during the installation process protect critical Tripwire data files. One or both of these key sets is necessary for performing almost every Tripwire task.

The *site key* is used to protect files that could be used across several systems. These files include the policy and configuration files.

The *local key* is used to protect files specific to the local machine, such as the Tripwire database. This key may also be used for signing integrity check reports.

Security Issue: You may also wish to back up Tripwire files on secondary media or to store them in a remote location for additional safety.

See Chapter 6, Appendices, for more information on Tripwire’s Cryptographic System.

Tripwire programs

The following programs are installed as part of the Tripwire package:

tripwire is used for creating the database, checking file system integrity, and updating the database or policy file.

twadmin is used for creating the site and local keys, configuration file, policy files, and signing files.

twprint verifies and prints Tripwire database and report files in a human-readable ASCII format.

siggen generates and prints cryptographic signatures for specified files. This can assist in verifying the integrity of system files without doing an entire Tripwire integrity check.

Walkthrough

This section describes the steps that a new Tripwire administrator might take after completing the installation process. These include generating a Tripwire database, creating Tripwire integrity checking reports, updating the database, and administering various configuration files.

Writing and installing the policy file

During installation, Tripwire creates a generic policy file *tw.pol*. For optimal performance, you will need to create a policy file tailored to your system. It may be helpful to refer to the *policyguide.txt* file as you do this.

A very simple policy file named *twpol.txt* might contain just one line:

```
/tmp    ->      $(ReadOnly) ;
```

In order for Tripwire to use this file as its policy, it must be encoded with the site key generated at install time. To do this, run the *twadmin* program in Create Policy mode.

```
% ./twadmin --create-polfile ../policy/twpol.txt
```

This will encode and sign the specified clear text file *twpol.txt* using the site key, and install it to the location specified by the POLFILE setting in the configuration file.

Initializing the database

After installing the policy file, the next step is to create the baseline Tripwire database. To do so, run the *tripwire* program in Database Initialization mode.

```
% ./tripwire --init
```

Checking integrity

After creating the baseline database, you can check the integrity of your file system by running the *tripwire* program in Integrity Checking mode.

```
% ./tripwire --check
```

This will print a Tripwire report to *stdout* and save a binary report file as specified by the REPORTFILE setting in the configuration file.

Updating the database

If acceptable changes are reported in the Tripwire report, you can update specific object information in the database. The easiest way to do this is to run *tripwire* in Integrity Checking mode using the interactive option.

```
% ./tripwire --check --interactive
```

An editor session will open, using the program specified by the EDITOR setting in the configuration file. Each change in the report will be displayed with a corresponding “ballot box.” Deselect all the unacceptable changes in the ballot boxes, write the file, and exit to update the database.

You can also update the database directly from any binary report file using the following command.

```
% ./tripwire --update --twrfile ../report/reportfile.twr
```

Security Issue: The $EDITOR environment variable is not used by Tripwire. This is to reduce the exposure to Trojan Horse attacks—for example, the $EDITOR environment variable could be changed to run an untrusted program with malicious side effects.

Updating the policy file

If you wish to update the Tripwire policy file, you will need to save it as a clear text file, edit it, and install it as the new policy file (similar to the first step in this walkthrough).

```
# print out the policy file
% ./twadmin --print-polfile > ../policy/twpol.txt

# edit the file
% vi ../policy/twpol.txt

# install the new policy file
% ./tripwire --update-polfile ../policy/twpol.txt
```

This process will sign the specified clear text file *twpol.txt* using the site key, and install it in the directory specified in the configuration file. For security reasons, you may want to delete the clear text version of the policy file at this time.

You can confirm that your policy changes have taken effect by running *tripwire* in Integrity Checking mode.

Security Issue: Once the initial policy file has been generated, any changes should be made with the *tripwire --update-policy* command, rather than by simply overwriting the policy file with the *twadmin --create-polfile* command. When a new policy file is created, the Tripwire database must be re-initialized. If an intruder has modified files since the last integrity check, these changes will not be detected, and will be included as part of the new baseline database.

Using email reporting

Email reporting adds considerable flexibility to Tripwire policies. Consider the following policy that describes two distinct areas of responsibility, distributed between two administrators.

```
/bin    ->      $(ReadOnly)
                        (rulename="OS executables", emailto=Susan) ;
/http   ->      $(ReadOnly)
                        (rulename="Web files",      emailto=Brian) ;
```

These rules specify that any violation to the OS executables rule are reported to Susan via email. Violations in the Web files rule will cause a report to be sent to Brian. To activate this feature, email reporting must be specified in the policy file, and the *--email-report* option must be used:

```
% ./tripwire --check --email-report
```

This will generate a Tripwire report to *stdout*, and also send email to the persons specified in the policy file using the program specified by the MAILPROGRAM setting in the configuration file.

3 CONFIGURATION REFERENCE

Introduction: The Tripwire configuration file contains information describing a variety of adjustable parameters that affect Tripwire operation. Although some parameters are generated by the installation script, others may need to be set by the Tripwire administrator. This chapter describes the format of the configuration file and consists of the following sections:

Chapter contents:

Overview

When Tripwire is first installed on a system, the installation script will read the installation configuration file *install.cfg*, unless an alternate configuration file is specified on the command line. By default, the installation script will create an encoded and signed Tripwire configuration file *tw.cfg* in the */usr/TSS/bin* directory, and a clear text copy of this configuration file *twcfg.txt* in the same directory.

Note that the *install.cfg* file is used only once, to designate target directories during the initial installation of the Tripwire software. See Chapter 1, Installing Tripwire, for more information.

The Tripwire configuration file is modified using the *twadmin --create-cfgfile* command. With this command, the user can designate an existing clear text file as the current configuration file. Using the current site key and passphrase, the new configuration file is encoded, signed, and saved. For example:

```
% ./twadmin --create-cfgfile --site-keyfile ../key/site.key configfile.txt
```

Components of the configuration file

The Tripwire configuration file is structured as a list of keyword-value pairs, and may also contain comments and variable definitions. The syntax for the configuration file is similar to, but not identical with, with the syntax used for the policy file (described in Chapter 4).

Comments

Any lines with # in the first column are treated as comments. For example:

```
# This is a comment in the TW config file.
```

Variables

The general syntax for variable definition is:

```
keyword = value
```

For example:

```
EDITOR = /usr/local/bin/jove
```

Variable substitution on the right hand side is permitted using the syntax:

```
$( varname )
```

For example:

```
DBFILE = $(ROOT)/db/$(HOSTNAME).db
```

Variable names are case sensitive, and may contain all alphanumeric characters, underscores, the characters `+-@:`, and period. It is an error to use a variable before it has been defined. Certain variables are read-only, and attempting to override such variables is an error.

Two variables are predefined by the Tripwire package and may not be changed:

Variable Name	Meaning
HOSTNAME	Unqualified hostname that Tripwire is running on. (e.g., leonardo)
DATE	String representation of date and time. (e.g., 19980930-180833)

The following variables must be set in order for Tripwire to operate. The initial values shown below are the values assigned during installation if the default *install.cfg* file is used. Relative pathnames are permitted, expressed in relation to the directory in which the Tripwire binaries reside.

Variable Name	Required	Set by Installer	Description
`POLFILE`	yes	yes	Default policy file. Initial value: `$(`*`ROOT`*`)/policy/tw.pol`
`DBFILE`	yes	yes	Default database file. Initial value: `$(`*`ROOT`*`)/db/$(`*`HOSTNAME`*`).db`
`REPORTFILE`	yes	yes	Specifies name of generated reports. Initial value: `$(`*`ROOT`*`)/report/$(`*`HOSTNAME`*`)-$(`*`DATE`*`).twr`
`SITEKEYFILE`	yes	yes	Specifies default site key file. Initial value: `$(`*`ROOT`*`)/key/site.key`
`LOCALKEYFILE`	yes	yes	Specifies default local key file. Initial value: `$(`*`ROOT`*`)/key/$(`*`HOSTNAME`*`)-local.key`

The following variables are not required to run Tripwire, but some of the program's functionality will be lost without them.

Variable Name	Required	Set by Installer	Description
`EDITOR`	no	yes	Specifies editor to be used in any interactive mode. If EDITOR is not defined, and no editor is specified on the command line, specifying interactive mode will cause an error. The editor specified must be able to take a filename as a command line parameter. If the process is killed, it must exit with a non-zero status. Initial value: `/bin/vi`
`MAILPROGRAM`	no	yes	MAILPROGRAM is used to specify the program that will be used for email reporting of rule violations detected by Tripwire. The program must take an RFC822 style mail header. Recipients will be listed in the “To:” field of the mail header Mail headers and the body of the report are then sent to *stdin* of MAILPROGRAM. The mail program must be able to ignore lines that consist of a single period (the `-oi` option to *sendmail* produces this behavior). If MAILPROGRAM is not defined in the configuration file, requests for email notification will cause an error. Initial value: `/usr/lib/sendmail -oi -t`
`LATEPROMPTING`	no	yes	Prompt for passphrases as late as possible to minimize the amount of time that the password is stored in memory. If the value is `true` (case sensitive), then late prompting is turned on. If it is set to any other value, or is removed from the configuration file, its value is interpreted as `false` and late prompting is turned off. Initial value: `false`

`LOOSEDIRECTORY CHECKING`	no	yes	When a file is added or removed from a directory, Tripwire reports both the changes to the file itself and the modification to the directory (size, number of links, etc.) This can creates redundant entries in Tripwire reports. With loose directory checking, Tripwire will not check directories for any properties that would change when a file is added or deleted. This includes: size, number of links, access time, change time, modification time, number of blocks, growing file, and all hashes.
			
			If the value for this variable is `true` (case sensitive), then loose directory checking is turned on, and these properties will be ignored for all directories. Turning loose directory checking on is equivalent to appending the following property mask to the rules for all directory inodes:
			`-snacmblCMSH`
			
			Initial value:
			`false`

(table continued next page)

(table continued)

Minimum configuration file

The following is an example of the minimum requirements for a Tripwire configuration file. Note that the values specified here are merely examples; the important point is that each of the variables below must have some valid assigned value.

```
POLFILE=/usr/local/tw/policy/tripwire.pol
DBFILE=/usr/local/tw/db/tripwire.db
REPORTFILE=/usr/local/tw/report/tripwire.twr
SITEKEYFILE=/usr/local/tw/key/site.key
LOCALKEYFILE=/usr/local/tw/key/local.key
```

If any of these variables is not defined, Tripwire will report an error and will exit. There is no limit to the number of additional variables that may be defined.

Note: Variables are case sensitive. “polfile” is not the same as “POLFILE”

Installing on multiple systems

To share a single site key file across multiple systems, place the site key file on a shared volume and specify the location of that key file in the configuration file. For example:

```
SITEKEYFILE=/mnt/server/keydir/site.key
LOCALKEYFILE=/usr/local/tw/key/local.key
```

4 POLICY REFERENCE

Introduction:

The Tripwire policy file describes which components of a system should be scanned by Tripwire, and specifies the types of data to be collected and stored in the database file. This chapter describes the format of the policy file and consists of the following sections:

Chapter contents:

Overview

The policy file describes which system objects Tripwire should monitor. In Tripwire 2.0, objects are defined as files and directories. A property mask is associated with each object in the policy file, describing what types of changes Tripwire should monitor and which ones can safely be ignored.

Comments, rules, directives, and variables are the standard components of the policy file. Each of these components is described in detail below.

Comments

In a policy file, any text following a #, up to the next line break, is considered a comment.

Example:

```
# This is a comment
/bin            ->  $(ReadOnly) ;        # A comment can go here, too.
```

Rules

Policy rules determine whether and to what extent Tripwire will check the integrity of particular files and directories.

There are two kinds of policy rules recognized by Tripwire: 1) normal rules define which properties of a particular file or directory tree Tripwire should scan; 2) stop points tell Tripwire not to scan a particular file or directory. Each of these policy rules is described in detail below. The meaning of a set of policy rules is unaffected by the order in which the rules appear.

Normal rules

A normal rule associates a system object with a property mask. The syntax for a normal rule is:

```
objectname   ->   property-mask   ;
```

An objectname is the fully qualified pathname for a directory or file, and the property mask specifies what properties of the object to examine or ignore. The `->` token separates the objectname and the property mask, and a semicolon must terminate the rule. If the pathname specified is a directory, the directory and all of its descendants will be scanned with the indicated property mask. If the pathname refers to an individual file, only that file will be scanned with the specified mask. Examples of normal rules are:

```
# Defines tripwire behavior for the entire /bin directory tree.
/bin                    ->      $(ReadOnly) ;

# Defines tripwire behavior for a single file.  In this case,
# Tripwire watches all properties of hostname.hme0.
/etc/hostname.hme0      ->      $(IgnoreNone) ;

# Scan the entire /etc directory tree using mask1, except
# the file /etc/passwd, which should be scanned using mask2.
/etc                    ->      $(mask1)  ;
/etc/passwd             ->      $(mask2)  ;
```

Only one rule may be associated with any given object. If an object has more than one rule in a policy file, *tripwire* will print an error message and exit without scanning any files.

```
# WARNING: this is an example of an illegal construct.
# It is an error to specify more than one rule for a given object.
/usr/bin                ->   $(mask3) ;
/usr/bin                ->   $(mask4) ;
```

Object names

In the policy file, objects may not be expressed using environment variables, for security reasons.

Examples:

```
/etc                    # valid
/etc/passwd             # valid
$HOME                   # not valid
```

The following regular expression defines the characters that are not allowed in object names:

```
([^!\{\}>\(\)\n\r\t \,;=$#\|\\\"]+)
```

In other words, any character is allowed except for an exclamation point, braces, greater-than sign, parentheses, newline, tabs, spaces, commas, semicolons, equal sign, dollar sign, pound, vertical bar, backslash, and quote.

Legal example:

```
/usr/local
```

Illegal example:

```
/usr/local/weird-filename-characters->;=-here
```

Because object names may contain characters that are not allowed on the left-hand side of rules, Tripwire supports quoted object names. Object name quoting might be needed when, for example, the object name contains spaces, exclamation marks, or equals signs. Object names with these characters must be double-quoted.

Examples:

```
"c:/program files"      ->      $(ReadOnly)      ;
"c:/bang!.doc"          ->      $(ReadOnly)      ;
```

Object names are concatenated; whitespace inserted between or within object names is ignored. Quotes are also ignored, unless inside a quoted string and preceded by a backslash. This allows more flexible handling of variable substitution and quoting.

Therefore, all of the following rules are equivalent:

```
/usr/local                 -> $(ReadOnly) ;
/usr /local                -> $(ReadOnly) ;
"/usr" "/local"            -> $(ReadOnly) ;
/usr / local               -> $(ReadOnly) ;
```

Filenames can contain escape sequences inside quoted strings to handle unprintable characters. The escaped sequences are interpreted in the same way as in the C++ language. The following examples define allowable sequences:

- octal numbers \412 (1, 2, or 3 octal digits)
- hex numbers \x2AFB1... (’x’ followed by one or more hex digits)
- characters: \t, \v, \b, \r, \f, \a, \\, \?, \', and \"
- all other escaped characters are treated as if not escaped.

Examples:

```
/test           # "/test"
"/te\x73t2"     # "/test2"
"/te\163t3"     # "/test3"
/tes\t          # ILLEGAL: escape sequences only valid in double quotes
```

Tripwire recurses into directories but only within the current filesystem. In other words, Tripwire does not cross mount points. More specifically, Tripwire will not recurse into any subdirectories which have a different device number as returned from *lstat*(2).

For example, if */usr/local* is a mount point, then

```
/usr                    ->                        +pinugsmc-a      ;
```

would cause all of */usr* to be scanned except the directory tree rooted at */usr/local*. If the goal is to scan */usr* in its entirety, including */usr/local*, the following rules should be specified:

```
/usr                    ->                        +pinugsmc-a      ;
/usr/local              ->                        +pinugsmc-a      ;
```

Property masks

Property masks describe object properties and whether Tripwire should examine each such property. The correct syntax is described by the following regular expression:

```
([+-]*[pinugtsldbamcCMSH])+
```

That is, the property mask must include one or more property symbols, each of which may be preceded by an optional plus or minus sign. Each one-character symbol is an abbreviation for a particular property that Tripwire is able to examine during integrity checking. If the character is preceded by a plus, checking is done for that property; if preceded by a minus, checking is not done for that property. For example:

```
+p              # compare permissions
-p              # ignore permissions
```

Properties not specified are ignored, so not specifying a property in the mask and explicitly turning it off with a minus sign are equivalent operations. The minus sign becomes most useful when variables (see below) are used to specify part of the mask. For example:

```
mask  =  +pinug ;                # define a variable called 'mask'
/file   ->   $(mask)-g ;         # use the mask defined by 'mask', but
                                 # turn off property 'g'
```

When property symbols appear in a selection mask without any preceding plus or minus sign, then plus is assumed.

```
+p+n            # compare permissions and number of links
pn              # same as above
```

The plus and minus signs are not unary operators; they toggle an internal state, so once a plus or minus appears in the selection mask, it applies to all successive properties until another plus or minus appears. All three of these selection masks are equivalent:

```
+p+n+s          # compare permissions, number of links, and file size
+pns            # same as above
pns             # same as above
```

In cases of duplicate or contradictory symbols, only the last symbol is acted upon. For example:

```
+p                      # compare permissions
+p-p                    # ignore permissions
+p-p+p                  # compare permissions
+p+-p                   # ignore permissions
```

It is an error to specify an empty selection mask. Specifying a selection mask that consists only of plus and minus characters is also an error. However, it is legal to specify a mask in which no properties are turned on. This is equivalent to specifying `$(IgnoreAll)` and is useful for monitoring only deletion or addition of files.

```
/temp -> +p-p ;      # This is legal but is equivalent to $(IgnoreAll).
/tmp  -> ;           # No property mask specified. This is not legal.
```

Characters which may be used to construct property masks:

Symbol	Description
`-`	Ignore the following properties
`+`	Record and check the following properties
`p`	Permission and file mode bits
`i`	Inode number
`n`	Number of links (i.e., inode reference count)
`u`	User id of owner
`g`	Group id of owner
`t`	File type
`s`	File size
`l`	Indicates that the file is expected to grow. If the file is smaller than the last recorded size, it is reported. Useful for log files, where Tripwire can check to make sure that files do not shrink. Note: If a file grows from size A to size B, where B>A, no violation is reported and the TW database is not updated. The most recent information in the TW database is that the file has size A. If the file then shrinks in size from B to C, where A<C<B, again, no violation is reported because C is still larger than A. Without explicitly updating the database, these violations cannot be reported, despite specifying this property.
`d`	Device number of the disk on which the inode associated with the file is stored
`b`	Number of blocks allocated
`a`	Access timestamp Note: The `+a` property is incompatible with the signature function properties (`+CMSH`). In order to calculate the hash, the file must be opened and read, which changes the access timestamp. Specifying any of `+CMSH` will always cause a violation of the `+a` property. Because enumerating a directory's contents changes that access timestamp, specifying `+a` in a directory rule will always cause a violation for the `+a` property during the next integrity check. To avoid this behavior, use `recurse = false` in the rule attribute, set `LOOSEDIRECTORYCHECKING = true` in the configuration file, or add `-a` to the rule.

(table continued next page)

(table continued)

Symbol	Description
`m`	Modification timestamp
`c`	Inode creation/modification timestamp
`C`	CRC-32, POSIX 1003.2 compliant 32-bit Cyclic Redundancy Check.
`M`	MD5, the RSA Data Security, Inc.® Message Algorithm
`S`	SHA, the NIST Secure Hash Algorithm (NIST FIPS 180)
`H`	Haval, a strong 128-bit signature algorithm

Stop points

To specify that certain directories or files not be scanned, stop points are used. The syntax is:

```
! objectname ;
```

As with normal rules, the objectname is the fully qualified pathname for a directory or file and a semicolon must terminate the stop point rule.

Consider the case where a policy rule has been specified for */etc*. The entire */etc* directory tree will be scanned recursively. Using stop points, it is possible to have Tripwire ignore particular files in the */etc* hierarchy.

```
# Scan all of /etc recursively, but do not scan two particular
# files present in the /etc hierarchy.
/etc            ->   $(ReadOnly) ;
!/etc/rc.d      ;       # ignore startup files
!/etc/mnttab    ;       # ignore dynamic listing of mounted filesystems
```

As described above, only one rule may be associated with any given object. This includes all types of rules, not just simple rules. The following illustrates this point.

```
# WARNING: this is an example of an illegal construct.
# It is an error to specify more than one rule for a given object.
/usr/bin                ->   $(mask3) ;
!/usr/bin ;
```

Rule attributes

Rule attributes provide additional information or modify Tripwire behavior and are associated with normal rules. For example, if a policy rule is broken, rule attributes can specify an email address to which notification should be sent. Rule attributes use the following format:

```
( attribute_name = attribute_value, attribute_name = attribute_value, … )
```

Rule attributes can take only one argument. To specify more than one address with the emailto rule attribute, the entire space-delimited list of addresses must be quoted:

```
(emailto="one@machine.com two@machine.com")       # Correct
(emailto=one@machine.com two@machine.com)         # Incorrect
(emailto="one@machine.com" "two@machine.com")     # Incorrect
```

Attribute names are case insensitive. It is an error to specify attributes for a stop point rule or to set an attribute that is not associated with any rule. Attributes are hard-coded in Tripwire; the following are currently supported:

Attribute	Description
`rulename`	Name associated with rule. Default is the object name to which the rule applies.
`severity`	Numeric severity level associated with rule. Default is 0. Range is 0 to `MAXINT` (operating system dependent).
`emailto`	Email address(es) to which notification of any violations is sent. Default is `none`.
`recurse`	Recursively scan the contents of a directory. Default is `true`.

Rule attributes are associated with individual normal rules according to the following syntax

```
object   ->  property-mask  ( attribute-list  ) ;
```

Example:

```
/usr/lib                  ->     $(ReadOnly) ( emailto = admin@foo.com ) ;
```

Rule attributes can also be specified for a group of rules:

```
( attribute-list )
{
       rule-list
}
```

Example:

```
( emailto = admin@foo.com )
{
       /usr/lib         ->      $(ReadOnly) ;
       /usr/sbin        ->      $(IgnoreNone) ;
}
```

The following two sets of rules (single and scoped respectively) are equivalent:

```
/usr/lib   ->   +pinug   (emailto = admin@foo.com) ;
/usr/bin   ->   +pinug   (emailto = admin@foo.com) ;

(emailto = admin@foo.com)
{
       /usr/lib         ->      +pinug ;
       /usr/bin         ->      +pinug ;
}
```

rulename

The rulename attribute is used to associate a symbolic name with one or more rules. This ability can be used to provide additional information in the report file. For example:

```
/home/.login   ->   $(ReadOnly) (rulename=rcfiles) ;
/home/.cshrc   ->   $(ReadOnly) (rulename=rcfiles) ;
/home/.logout  ->   $(ReadOnly) (rulename=rcfiles) ;
```

The effect of these three lines is to associate the symbolic name, `rcfiles`, with the three objects named in the rules. In a report file, the results for these three rules would be flagged as originating from rules named `rcfiles`. This feature is useful if you wish to track certain objects within a large Tripwire database. For example, important files in different directories can be tagged with a unique rulename (e.g. `rulename=watchme`). You can then run *tripwire* and interpret your data later using the rulename `watchme` as a sorting key.

severity

This attribute associates a severity level with a rule. When *tripwire* is run in Integrity Checking mode, it is possible to specify that only rules exceeding a certain severity level be used. For example:

```
# In the policy file:
/usr/lib  ->  $(ReadOnly) (severity=80) ;

# On the command line:
# Rule will be run in this case
% ./tripwire --check --severity 60

# Rule will not be run in this case
% ./tripwire --check --severity 90
```

The default severity value is 0, and the range of legal values is 0 to MAXINT, defined by the operating system.

emailto

The emailto rule attribute allows one or more email addresses to be associated with a rule. When the rule is violated, notification is sent to the specified email address(es). The emailto attribute takes only a single argument, so to specify multiple email addresses, include them as a quoted, space-delimited list. For example:

Correct examples:

```
/bin   ->   $(ReadOnly) (emailto=admin@foo.com ) ;                # ok
/etc   ->   $(ReadOnly) (emailto="admin@foo.com noc@foo.com" ) ;   # ok
```

Incorrect example:

```
/bin    ->   $(ReadOnly) (emailto = admin@foo.com noc@foo.com ) ; # WRONG!
```

Note that the emailto rule uses the mail program specified by the MAILPROGRAM variable in the configuration file. See Chapter 3, Configuration Reference, for more information. Also note that email is sent only if the *--email-report* argument of the *tripwire* command is specified.

recurse

When the recurse rule attribute is set to `true` and the rule refers to a directory object, *tripwire* will recursively scan the entire contents of the directory (both files and subdirectories). When the recurse rule attribute is set to `false` and the rule refers to a directory object, *tripwire* will scan the inode corresponding to the directory but none of the files or subdirectories contained therein. When a rule refers to a file (as opposed to a directory), specifying the recurse attribute has no effect—the file will be scanned no matter what value recurse is given. The default value for recurse is `true`.

For example, to monitor the attributes of the /tmp directory without monitoring any of its contents, the following would be used:

```
/tmp   ->   $(ReadOnly) (recurse=false) ;
```

Directives

Tripwire supports a small set of preprocessor-like directives that allow conditional interpretation of the policy file and perform certain diagnostic and debugging operations. The primary intent of this mechanism is to support sharing a policy file among multiple machines. Directives have the following syntax:

```
@@  directive_name   [arguments]
```

White space may precede or follow the @@ construct, but non-whitespace characters may not appear on the line before the @@ construct, nor may any characters intervene between the two @ characters. Directive names are case sensitive. The following directives are supported (each of these is described in detail below):

Directives	Description
`@@ifhost` `@@else` `@@endif`	Allow conditional interpretation of the policy file.
`@@print` `@@error`	Print a message to *stderr* and optionally exit.
`@@end`	Marks the logical end-of-file.

Note that the directives cannot result from variable expansion. The following illustrates correct and incorrect syntax:

```
machine = spock
@@ifhost  $(machine)             # This is correct syntax...

IFHOST=ifhost
@@ $(IFHOST)    spock            # ...but this will produce an error.
```

Conditional interpretation

The `@@ifhost`, `@@else`, and `@@endif` directives are used to allow conditional interpretation of a policy file. The syntax for each is:

```
@@ifhost host1 || host2 || …
@@else
@@endif
```

Where `host1`, `host2`, … are unqualified hostnames.

The following illustrates how one might employ directives to use one policy file with four different hosts:

```
@@ifhost spock || kirk
       /bin            ->      $(ReadOnly) ;
@@endif

@@ifhost chekov || uhura
       /usr/bin ->      +pinug ;
@@else
       /usr/bin ->      +pinugsmC ;
@@endif
```

If the unqualified hostname of the machine running Tripwire matches any of the hosts listed in the `@@ifhost` directive, all the lines between the `@@ifhost` and the matching `@@endif` are interpreted.

If there is no match, any lines between the `@@ifhost` and `@@endif` are skipped. However, if there is an `@@else` in those skipped lines, any lines between the `@@else` and `@@endif` are interpreted. There is no `@@elsif` directive.

Note that only the logical OR operation is supported.

The `@@ifhost` and `@@else` directives can be nested arbitrarily deeply. For example:

```
@@ifhost chekov || uhura
@@else
        @@ifhost bones
        @@endif
@@endif
```

Message reporting

The `@@print` and `@@error` directives are intended for debugging and remote diagnostics. The syntax is:

```
@@print STRING
@@error STRING
```

Note: Only one string is allowed as a parameter to these directives, so quoting may be necessary to achieve the desired result.

Examples:

```
@@print string                                # okay
@@print "Two strings"                         # okay
@@print two strings                           # ERROR
```

The `@@print` directive merely prints its arguments to *stderr*, while the `@@error` directive prints its arguments to *stderr* and then causes the calling program to exit with a status of 1.

Indicating end-of-file

The `@@end` directive marks the logical end of the policy file. Any text appearing after this directive will be ignored by Tripwire applications.

Variables

For user convenience and to allow reuse of properties, Tripwire supports variables for string substitution.

Variables can be defined anywhere between rules. The syntax for variable definition is:

```
varname  =  value  ;
```

Variable names are case sensitive, and may contain all alphanumeric characters, underscores, the characters `+-@:`, and period. The regular expression for variable names is:

```
varname: [A-Za-z0-9+\-@:]+
```

The scope of a variable begins at the point where it is defined to the end of the file. It is an error to use a variable before it has been defined. Examples of variable definition are:

```
path    =      /usr/local/lib/bigproject ;
mask1   =      +pinugC-a                 ;
```

Variable substitution is legal anyplace that a string could appear. The syntax for variable substitution is:

```
$( varname )
```

Variables may be used on the lefthand side of rules:

```
# Set the variable…
path   =   /usr/local/lib/bigproject   ;
# …and now use it.
$(path)/src   ->   +pug  ;
$(path)/exe   ->   +pugntmc  ;
```

Variables may also be used on the righthand side of rules:

```
# Set the variable…
mask1   =   +pinugC-a   ;
# …and now use it.
/home/projectA   ->   $(mask1)  ;
/home/projectB   ->   $(mask1)+MSH-db  ;
```

Variables may be used in directives:

```
# Define a machine.
server = jupiter        ;

@@ifhost $(server)
# etc.
@@endif
```

Note, however, that tokens cannot be included in variable substitutions, and therefore the following would not work (the "||" construct is a token).

```
# Define a variable listing all machines in sales.
sales_department  =  jupiter || mars || pluto || mercury  ;

@@ifhost $(sales_department)     # ERROR
# etc.
@@endif
```

Variables that are not predefined by Tripwire may be overridden by the user. However, it is not legal to override variables predefined by Tripwire (see below).

```
# This is fine…
mask   =   +p  ;
/fileA  ->  $(mask) ;
mask   =   -p+u  ;
/fileB  ->  $(mask) ;
```

The following are all examples of improper use:

```
# WARNING: these are examples of illegal constructs.
# It is an error to replace a literal token with a variable.
arrow  =  ->  ;
/file  $(arrow)  +pug  ;

# It is an error to replace a directive with a variable.
ifhostnameis  =  @@ifhost  ;
$(ifhostnameis) kirk

# It is an error to override a pre-defined variable.
ReadOnly  =  +pinugsmCMSH-ac  ;
```

Predefined variables

Symbol	Description
`ReadOnly`	File is read only: Expands to: `+pinugsmtdbCM-acSH`
`Dynamic`	File changes: Expands to: `+pinugtd-sacmbCMSH`
`Growing`	File is log file, which can grow (but not shrink): Expands to: `+pinugtdl-sacmbCMSH`
`IgnoreAll`	Ignore all attributes: Expands to: `-pinusgslamctdbCMSH`
`IgnoreNone`	Ignore no attributes: Expands to: `+pinusgsamctdbCMSH-l`
`Device`	Device file: Expands to : `+pugs-intldamcCMSH`

5 COMMAND REFERENCE

Introduction: This chapter provides an overview of the four programs used by Tripwire. The various modes for each program are described in detail, with a complete list of command line options.

Chapter contents:

Command introduction

All Tripwire applications (except *siggen*) observe the following convention for command lines.

```
% ./command  mode-selector [ options ] [ files ]
```

That is, the mode selector must always be the first argument on the command line, and any files not associated with the command line options must appear last on the command line. For example:

```
% ./twadmin --create-cfgfile --site-keyfile keyfile configfile.txt
```

Specifying command line arguments in any other order will generate a syntax error.

All Tripwire applications support the following argument for obtaining usage, version, and copyright information. If this argument is present on the command line, the help message will be displayed and all other command line arguments will be ignored.

Argument	Meaning
`-?` `--help`	Display version and usage information and exit.

TRIPWIRE

Section contents:

Introduction

The *tripwire* command runs in one of four modes: Database Initialization, Integrity Checking, Database Update, or Policy Update.

In Database Initialization mode, Tripwire builds a database of system objects as described in the policy file. This database will serve as the baseline for later integrity checks.

The Integrity Checking mode generates a report of additions, deletions, or changes for each object described in the policy file. By comparing the actual files residing on the system with information stored in the database, Tripwire maintains system security.

After running an integrity check, Database Update mode allows the Tripwire database to be updated with changes from an integrity check report. This process allows new information to be added to the database without having to regenerate the entire database.

Policy Update mode provides a way to synchronize the database and policy files after changes are made to the policy file. This command enables the user to change the way that Tripwire scans the system, without having to do a complete re-initialization of the database.

Valid command line arguments for each mode are shown in the sections below. The following conventions are used:

[-f *filename*]	Optional arguments are in square brackets. All other arguments are required.
-m {I, D, P, Z}	Arguments that must be chosen from a set are in curly braces.

Initializing the database

When run in Database Initialization mode, *tripwire* reads the policy file, generates a database based on its contents, and then cryptographically signs the resulting database. The database will be created in the location specified by the DBFILE variable in the Tripwire configuration file, unless another location is specified on the command line. Additional command line options can be entered to specify which policy, configuration, and key files are used to create the database. If no options are specified, the default values from the current configuration file are used.

Argument	Meaning
Mode of operation	

`-m i`
`--init`
	Selects database initialization mode.
	Create the baseline Tripwire database from the specified policy file.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration file overrides (input)	

`[-p polfile]`
`[--polfile polfile]`
	Use the specified policy file.

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

`[-S sitekey]`
`[--site-keyfile sitekey]`
	Use the specified site key file to read the configuration and policy files.

`[-L localkey]`
`[--local-keyfile localkey]`
	Use the specified local key file to write the database file.
	(Mutually exclusive with -e).

Configuration file overrides (output)	

`[-d database]`
`[--dbfile database]`
	Write to the specified database file.

(table continued next page)

(table continued)

Argument	Meaning
Unattended operation	
`[-P passphrase]` `[--local-passphrase passphrase]`	Use the specified passphrase with local key to sign database.(Mutually exclusive with -e).
Security settings	
`[-e]` `[--no-encryption]`	Do not sign the database. To sign the database after initialization requires the use of *twadmin*. (Mutually exclusive with -P and -L).

Checking integrity

After building the Tripwire database, the next step is typically to run *tripwire* in integrity checking mode. This mode scans the system for violations, as specified in the policy file. Using the policy file rules, Tripwire will compare the state of the current file system against the initial baseline database. An integrity checking report is printed to *stdout* and is saved in the location specified by the REPORTFILE setting in the Tripwire configuration file.

The generated report describes each policy file violation in detail, depending on whether the specified file system object was added, deleted, or changed. Each report item lists the properties of the object as it currently resides on the file system, and if appropriate, the old value stored in the database. If there are differences between the database and the current system, the administrator can either fix the problem by replacing the current file with the correct file (e.g., an intruder replaced */bin/login*), or update the database to reflect the new file (e.g., a fellow system administrator installed a new version of */usr/local/bin/emacs*).

Running an integrity check using the *tripwire --interactive* option is the same as running a non-interactive integrity check immediately followed by running *tripwire* in Database Update mode. In Interactive mode, the editor specified in the configuration file with the EDITOR variable is launched on a clear text copy of the Tripwire report. In that report, next to every rule violation, will be “ballot boxes” that look like this:

```
Modified:
[x] "/usr/local/tw"
      drwxr -xr -x root(0)      512 Tue Nov 17 13:36:50 1998
```

If you leave the “x” in the ballot box, the database will be updated. That is, you approve this change to the filesystem. If you remove the “x” from the ballot box, then the database will not be updated with the new value(s) for that object.

Argument	Meaning
Mode of operation	
`-m c` `--check`	Selects integrity checking mode.
`[-I]` `[--interactive]`	At the end of integrity checking, the resulting report is opened in an editor, where database updates can be easily specified using the ballot boxes included in the report. Updating the database in this fashion requires the local passphase.
Reporting	
`[-v]` `[--verbose]`	Verbose mode. (Mutually exclusive with -s).
`[-s]` `[--silent]` `[--quiet]`	Silent mode. (Mutually exclusive with -v).

(table continued next page)

(table continued)

Argument	Meaning
Configuration file overrides (input):	

`[-p polfile]`
`[--polfile polfile]`
	Use the specified policy file.

`[-d database]`
`[--dbfile database]`
	Use the specified database file.

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

`[-S sitekey]`
`[--site-keyfile sitekey]`
	Use the specified site key file to read the configuration and policy files.

`[-L localkey]`
`[--local-keyfile localkey]`
	Use the specified local key file to read the database and write the report file. Also used to write the database file when *--interactive* is used.

`[-V editor]`
`[--visual editor]`
	Use the specified editor to edit the report ballot box. Only applies in interactive mode.

Unattended operation:	

`[-P passphrase]`
`[--local-passphrase passphrase]`
	Specify passphrase for local key when signing the secure report. Also used to write the database file in interactive mode.
	(Only valid with -E or -I)

Output:	

`[-n]`
`[--no-tty-output]`
	Suppress the report from being printed at the console.

`[-r report]`
`[--twrfile report]`
	Specifies that the binary output *.twr* file be written to the specified file.

`[-M]`
`[--email-report]`
	Specifies that reports be emailed to the recipients designated in the policy file. The MAILPROGRAM variable must be set in the Tripwire configuration file.
	
	Email is sent to all persons specified by the `emailto` attribute in the policy file. One of two email reports is sent, according to the following rules: If the person is not in the set of rules that have violations, a “no violations found” email is sent. Otherwise, the entire violation report is sent.

`[-E]`
`[--signed-report]`
	Specifies that the Tripwire report will be signed. If no passphrase is specified on the command line, *tripwire* will prompt for the local passphrase.

(table continued next page)

(table continued)

Argument	Meaning
Scope of operation	

`[-l { level | name }]`
`[--severity { level | name }]`
	Check only policy rules equal to or greater than the given severity level. The level may be specified as a number or as a name. Severity names are defined as follows:
	Low 33
	Medium 66
	High 100
	Note: Rules which do not explicitly have a severity level set in the policy file, have an implicit severity level of zero.
	(Mutually exclusive with -R).

`[-R rule]`
`[--rule-name rule]`
	Run only the specified policy rule.
	(Mutually exclusive with -l).

`[-i]`
`[--ignore` *`list`*`]`
	Do not compute or compare the properties specified in *list*. Any of the letter codes (abcdgimnpstulCHMS) specified in property masks can be excluded. Use of this option overrides information from the policy file.

`[file1 [ file2 … ]]`	List of files and directories that should be integrity checked. Default is all files. (Overrides -l and -R).

Updating the database

The Integrity Checking mode looks for differences between the database and the system and creates a report of those changes. Database Update mode displays the report to the Tripwire administrator, who chooses which (if any) records in the database should be updated.

Rule violations from the report specified on the command line will be listed with a series of “ballot boxes.” If no report is specified, *tripwire* will read in the report file defined by the REPORTFILE variable in the Tripwire configuration file. In the report, every rule violation will have a “ballot box”:

```
Modified:
[x] "/usr/local/tw"
       drwxr -xr -x root(0)      512 Tue Nov 17 13:36:50 1998
```

The entries to be updated are specified by leaving the “x” next to each policy violation, indicating you approve this change to the file system. If you remove the “x” from the ballot box, the database will not be updated with the new value for that object.

After the user exits the editor and provides the correct local passphrase, *tripwire* will update and save the database.

Argument	Meaning
Mode of operation	

`-m u`
`--update`
	Selects database updating mode

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s).

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v).

Configuration file overrides (input):	

`[-p policyfile]`
`[--polfile policyfile]`
	Use the specified policy file.

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

`[-d dbfile]`
`[--dbfile dbfile]`
	Update the specified database file.

`[-S keyfile]`
`[--site-keyfile keyfile]`
	Use the specified site key file to read the configuration and policy files.

`[-L keyfile]`
`[--local-keyfile keyfile]`
	Use the specified local key file to read and write the database and to read the report file.

`[-V editor]`
`[--visual editor]`
	Use the specified editor to edit the update ballot box. (Mutually exclusive with -a).

`[-r report]`
`[--twrfile report]`
	Read the specified report file.

(table continued next page)

(table continued)

Argument	Meaning
Unattended operation:	

`[-P passphrase]`
`[--local-passphrase passphrase]`
	Use the specified passphrase with local key to sign the database file.

`[-a]`
`[--accept-all]`
	Specifies that all the entries in the *.twr* file are updated without any prompting.
	(Mutually exclusive with -V).

Security level:	

`[-Z {high, low}]`
`[--secure-mode {high | low}]`
	Specifies the security level, which affects how certain conditions are handled when inconsistent information is found between the *.twr* file and the current database:
	
	High: If properties of an object in the database do not match the expected (old) values in the *.twr* file, in high security mode, Tripwire reports the differences as warnings and exits without changing the database.
	
	Low (default): In the case of inconsistencies, the differences are reported as warnings, but the changes are made to the database.

Updating the policy file

Policy update mode is used by *tripwire* to change or update the policy file and to synchronize an earlier database with new policy file information. The filename of the new clear text version of the policy file is specified on the command line. The new policy file is compared to the existing version, and the database is updated according to the new policy rules. Any changes in the database since the last integrity check will be detected and reported.

How these violations are interpreted depends on the security mode specified with the *-Z* or *--secure-mode* option. In high security mode, Tripwire will print a list of violations and exit without making changes to the database. In low security mode (the default), the violations are still reported, but changes to the database are made automatically.

The Tripwire database is the baseline against which security violations are found. As a result, an assumption exists that the information in the database represents a file system that has not already been compromised.

Although *twadmin --create-polfile* can create a new policy file, it requires re-initializing the database. This is necessary for new Tripwire installations.

A potential problem exists with this procedure when you have been monitoring a system with Tripwire for any length of time. When you re-initialize the database, there exists a risk that files have been modified between your last integrity check and the database re-initialization.

The Policy Update mode addresses this security risk. In this mode, Tripwire checks the rules in the new policy file against the rules in the current database. Any substantively identical rules have their records copied directly into the new database. Changes in these rules will be detected in the next integrity check, as normal.

Conversely, any information on objects specified in the new policy file, but not in the old database, is gathered by Tripwire. Before writing this information to the database, Tripwire verifies that no information conflicts with information already in the database.

For example, suppose that after the last integrity check, a rule existed for */bin*, and resulted in information being stored about */bin/login*. Then, unknown to the administrator, an intruder deletes */bin/login*. Meanwhile, the Tripwire administrator decides to modify the rule for */bin*. When Tripwire is run in Update Policy mode, it gathers information from the file system for this “new” rule. Unfortunately, no record for */bin/login* will be created—it has been deleted by the intruder.

However, Tripwire will detect that the old rule used to cover */bin*, and that there used to be a record for */bin/login*. Therefore */bin/login* will be reported as a conflict.

The same concept applies to both added and modified files. If properties specified in the property mask differ, a warning is printed. These conflicts should be treated with the same seriousness as integrity checking violations.

For this reason, it is recommended that you always run Update Policy mode with *--secure-mode high*, so that these situations can be detected, and appropriate actions taken.

Note that this does not apply to file system objects and properties that the new policy file expressly excludes. Consequently, no warnings are generated for changes to files or properties you no longer want to monitor.

Argument	Meaning
Mode of operation	

`-m p`
`--update-policy`
	Selects policy updating mode

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s).

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v).

(table continued next page)

(table continued)

Argument	Meaning
Configuration file overrides (input):	
`[-p policyfile]` `[--polfile policyfile]`	Update the specified policy file.
`[-c cfgfile]` `[--cfgfile cfgfile]`	Use the specified configuration file.
`[-d dbfile]` `[--dbfile dbfile]`	Update the specified database file.
`[-S keyfile]` `[--site-keyfile keyfile]`	Use the specified site key file to read the configuration file and to read and write the policy file.
`[-L keyfile]` `[--local-keyfile keyfile]`	Use the specified local key file to read and write the database file.
Unattended operation:	
`[-P passphrase]` `[--local-passphrase passphrase]`	Use the specified passphrase with the local key to sign the database file.
`[-Q passphrase]` `[--site-passphrase passphrase]`	Use the specified passphrase with the site key to sign the policy file.
Security level:	
`[-Z {high, low}]` `[--secure-mode {high | low}]`	Specifies the security level, which affects whether the policy and database file are saved if a violation of the old policy exists. Violations are always printed to *stderr*, but if security is set to high, no files are changed if any errors occur. (Default is low.)
Input	
`policyfile.txt`	Specifies the clear text policy file that will become the new encoded and signed policy file.

TWPRINT

Section contents:

Running twprint

Tripwire database files are binary encoded and signed. Tripwire report files are encoded and may optionally be signed. The *twprint* application provides a way to view these files in clear text form.

Valid command line arguments for each mode are shown. The following conventions are used:

[-f *filename*]	Optional arguments are in square brackets. All other arguments are required.
-m {I, D, P, Z}	Arguments that must be chosen from a set are in curly braces.

Printing a database file

The *tripwire* application can create and update database files, but it cannot print them. This mode prints the contents of a Tripwire database.

Note: If no database is specified under the --*dbfile* command line option, the default database will be used. The default database is specified by the DBFILE variable in the configuration file (either *tw.cfg* or the configuration file specified under the --*cfgfile* command line option).

Argument	Meaning
Mode of operation	

`-m d`
`--print-dbfile`
	Database printing mode.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s).

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v).

Configuration file overrides (input):	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

`[-d database]`
`[--dbfile database]`
	Use the specified database file.

`[-L localkey]`
`[--local-keyfile localkey]`
	Use the specified local key file to verify the database file.

Printing a report file

The *tripwire* application can create report files, and also display them as part of an integrity check or update operation. However, only the --*print-report* mode prints the contents of a Tripwire report.

Note: If no report is specified under the --*twrfile* command line option, the default report will be used. The default report is specified by the REPORTFILE variable in the configuration file (either *tw.cfg* or the configuration file specified under the --*cfgfile* command line option).

The default filename format (as installed) for the report file name is $(*HOSTNAME*)-$(*DATE*).*twr*, where the $(*DATE*) variable includes the current time to the nearest second. **Unless *twprint --print-report* is run within one second of such a report's creation, *twprint* will be unable to find the report** because the $(*DATE*) variable will have changed to reflect the current time. **Therefore, when printing a report, you should always use the --*twrfile* command line option to explicitly specify the report.**

Argument	Meaning
Mode of operation	

`-m r`
`--print-report`
	Report printing mode.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s).

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v).

Configuration file overrides (input):	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

`[-r report]`
`[--twrfile report]`
	Print the specified report file.

`[-L localkey]`
`[--local-keyfile localkey]`
	Use the specified local key file to verify the signed report file.

TWADMIN

Section contents:

Running twadmin

The *twadmin* utility is used to perform certain administrative functions related to Tripwire files and configuration options. Specifically, *twadmin* allows binary-encoding/decoding and cryptographic signing/verifying of Tripwire files, and provides a means to generate local and site keys.

Valid command line arguments for each mode are shown. The following conventions are used:

[-f *filename*]	Optional arguments are in square brackets. All other arguments are required.
-m {I, D, P, Z}	Arguments that must be chosen from a set are in curly braces.

Creating a configuration file

This command mode designates an existing clear text file as the new configuration file for Tripwire. The clear text configuration file must be specified on the command line. Using the site key, the new configuration file is encoded, signed and saved.

Argument	Meaning
Mode of operation	

`-m F`
`--create-cfgfile`
	Take the specified clear text file (see Input below) and store it as a binary-encoded Tripwire configuration file.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration file overrides (input):	

`-S sitekey`
`--site-keyfile sitekey`
	Use the specified site key file to sign the new configuration file. (Mutually exclusive with -e.)
	
	Either -e or -S must be specified.

Configuration file overrides (output):	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Specify the destination configuration file.

Security settings:	

`[-e]`
`[--no-encryption]`
	Do not sign the configuration file. The configuration file will still be stored in a binary-encoded form and will not be human-readable. (Mutually exclusive with -Q and -S.)
	
	Either -e or -S must be specified.

Unattended operation:	

`[-Q passphrase]`
`[--site-passphrase passphrase]`
	Specifies passphrase to be used with site key for signing the configuration file. (Valid only in conjunction with -S.)

Input:	
`configfile.txt`	Specifies the clear text configuration file that will become the new binary-encoded and signed configuration file.

(table continued next page)

(table continued)

Printing a configuration file

After a configuration file has been created, it is stored in a binary-encoded form. This command provides a way of printing out the current contents of the configuration file in a readable text format.

Argument	Meaning
Mode of operation	

`-m f`
`--print-cfgfile`
	Prints the configuration file to *stdout*.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration file overrides (input):	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Print the specified configuration file.

Creating a policy file

This command mode designates an existing clear text file as the new policy file for Tripwire. The clear text policy file must be specified on the command line. Using the site key, the new policy file is encoded and saved.

Although you can modify a policy file and save the new version using this command mode, it is strongly recommended that you use *tripwire* in Update Policy mode instead. Using *twadmin --create-polfile* forces a database re-initialization, because the records in the old database will no longer match the rules specified in the policy file. This gives tacit (and possibly incorrect) approval that the current state of the filesystem is an appropriate baseline for future integrity checks. Unless you are certain that your system has not been compromised since your last integrity check, you should run *tripwire --update-policy* instead. This command mode updates the policy and the database simultaneously, checking for possible policy violations as it goes. See the *tripwire* section of the Tripwire Command Reference for more details.

Argument	Meaning
Mode of operation	

`-m P`
`--create-polfile`
	Takes the specified clear text file (see Input below) and stores it as a binary-encoded Tripwire policy file.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration file overrides (input):	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Specifies the destination binary-encoded configuration file.

`[-S sitekey]`
`[--site-keyfile sitekey]`
	Specifies site key file to be used.
	(Mutually exclusive with -e.)

Configuration file overrides (output):	

`[-p polfile]`
`[--polfile polfile]`
	Specifies policy file to be written.

Unattended operation:	

`[-Q passphrase]`
`[--site-passphrase passphrase]`
	Specifies passphrase to be used with the site key for signing the policy file. (Mutually exclusive with -e.)

Security settings:	

`[-e]`
`[--no-encryption]`
	Does not sign the policy file. The policy file will still be stored in a binary-encoded form and will not be human-readable.
	(Mutually exclusive with -Q and -S.)

Input:	
`policyfile.txt`	Specifies the clear text policy file that will become the new binary-encoded and signed policy file.

(table continued next page)

(table continued)

Printing a policy file

After a policy file has been created, it is stored in a binary-encoded form. This command provides a way of printing out the current contents of the policy file in a readable text format.

Argument	Meaning
Mode of operation	
`-m p` `--print-polfile`	Selects policy file printing mode. Output is sent to *stdout*.
Reporting	
`[-v]` `[--verbose]`	Verbose mode. (Mutually exclusive with -s.)
`[-s]` `[--silent]` `[--quiet]`	Silent mode. (Mutually exclusive with -v.)
Configuration file overrides (input):	
`[-c cfgfile]` `[--cfgfile cfgfile]`	Use the specified configuration file.
`[-p polfile]` `[--polfile polfile]`	Print the specified policy file.
`[-S sitekey]` `[--site-keyfile sitekey]`	Use the specified site key file.

Removing encryption from a file

This command mode allows the user to remove cryptographic signing from configuration, policy, database, or report files. Multiple files may be specified on the command line. The user will need to enter the appropriate local or site keyfile, or both if a combination of files is to be verified. With the *-K* or *--keyfile* flag, the user can force the use of a specific keyfile, regardless of the file types to be verified. Even with cryptographic signing removed, these files will be in a binary-encoded form which is unreadable to humans.

(see table next page)

Argument	Meaning
Mode of operation	
`-m R` `--remove-encryption`	Remove cryptographic signing for one or more files. The type of each file will be examined and the appropriate key (local key for databases and reports, site key for configuration and policy files) will be used to remove signing from it. The file will then be rewritten unsigned. Files will remain in binary-encoded format.
Reporting	
`[-v]` `[--verbose]`	Verbose mode. (Mutually exclusive with -s.)
`[-s]` `[--silent]` `[--quiet]`	Silent mode. (Mutually exclusive with -v.)
Configuration File	
`[-c cfgfile]` `[--cfgfile cfgfile]`	Use the specified configuration file.
Key files	
`[-K key]` `[--keyfile key]`	Specify the keyfile to use to remove cryptographic signing from the files. *twadmin* will attempt to use this key for all files without regard to the file type. The local passphrase will be used with this key.
`[-L localkey]` `[--local-keyfile localkey]`	Specify the local keyfile to use to remove cryptographic signing for database files and reports.
`[-S sitekey]` `[--site-keyfile sitekey]`	Specify the site keyfile to use to remove cryptographic signing for configuration and policy files.
Unattended operation	
`[-P passphrase]` `[--local-passphrase passphrase]`	Specify passphrase to use with the local keyfile when removing cryptographic signing from database files and reports.
`[-Q passphrase]` `[--site-passphrase passphrase]`	Specify passphrase to use with the site keyfile when removing cryptographic signing from the configuration and policy files.
File conversion:	
`file1 [file2…]`	List of files from which cryptographic signing is to be removed.

Encrypting a file

This command mode allows the user cryptographically to sign configuration, policy, database files, or reports. Multiple files may be specified on the command line. The files will be signed using either the site or local key as appropriate for the file type. To automate the process, the passphrase for the keyfiles can be included on the command line.

Argument	Meaning
Mode of operation	

`-m E`
`--encrypt`
	Sign one or more files.
	Target files must be currently unsigned.
	Each file will be singed using either the site or local key as appropriate for the type of file. The keyfile used for either site or local key may be overridden using the -S or -L option.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration File	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

Key files	

`[-L localkey]`
`[--local-keyfile localkey]`
	Specify the local keyfile to use to sign database files and reports.

`[-S sitekey]`
`[--site-keyfile sitekey]`
	Specify the site keyfile to use to sign configuration and policy files.

Unattended operation	

`[-P passphrase]`
`[--local-passphrase passphrase]`
	Specify passphrase to be used with the local keyfile.

`[-Q passphrase]`
`[--site-passphrase passphrase]`
	Specify passphrase to be used with the site keyfile.

File conversion:	
`file1 [file2…]`	List of files to sign using the site or local key (as appropriate, depending on the file type).

Examining the encryption status of a file

This command allows the user to examine the listed files and print a report of their encryption status. This report displays the filename, file type, whether or not a file is signed, and what key (if any) is used to sign it.

Argument	Meaning
Mode of operation	

`-m e`
`--examine`
	Examine the listed files. Report their type and whether or not binary-encoding is used.
	For each file specified a report will be given displaying whether or not it is signed and if it is signed with what key. Signing type for files will be determined by a trial and error method, first using the site key, then local key, then the keyfile provided with the -K option.

Reporting	

`[-v]`
`[--verbose]`
	Verbose mode.
	(Mutually exclusive with -s.)

`[-s]`
`[--silent]`
`[--quiet]`
	Silent mode.
	(Mutually exclusive with -v.)

Configuration File	

`[-c cfgfile]`
`[--cfgfile cfgfile]`
	Use the specified configuration file.

Key files	

`[-K key]`
`[--keyfile key]`
	Test to see if file is signed using the specified key.

`[-L localkey]`
`[--local-keyfile localkey]`
	Specify the key to use as a local key when examining database or report files.

`[-S sitekey]`
`[--site-keyfile sitekey]`
	Specify the key to use as a site key when examining policy or configuration files.

File examination:	
`file1 [file2…]`	List of files to examine.

Generating keys

This command provides the interface to create site or local keys for Tripwire. Although keys are generated by the install process, this command can be used at any time to regenerate the keys. The site and local keys may be generated simultaneously or one at a time with two separate invocations of *twadmin*.

Argument	Meaning
Mode of operation	
`[-m G]` `[--generate-keys]`	Selects key generation mode.
Reporting	
`[-v]` `[--verbose]`	Verbose mode. (Mutually exclusive with -s.)
`[-s]` `[--silent]` `[--quiet]`	Silent mode. (Mutually exclusive with -v.)
Output:	
`-L localkey` `--local-keyfile localkey` `-S sitekey` `--site-keyfile sitekey`	Generate keys into the specified files. At least one of these options must be specified.
Unattended operation:	
`[-P passphrase]` `[--local-passphrase passphrase]`	Specifies passphrase to be used when generating a local key.
`[-Q passphrase]` `[--site-passphrase passphrase]`	Specifies passphrase to be used when generating a site key.

SIGGEN

Section contents:

Running siggen

The *siggen* utility can be used to display cryptographic signature values for any specified file. See the signature function reference section of Chapter 6, Appendices, for more information about the signature functions supported by Tripwire.

Valid command line arguments for each mode are shown.

The *siggen* application displays some or all of the signature values for any specified file(s) in base64 notation.

Argument	Meaning
Reporting	
`[-t]` `[--terse]`	Terse mode. Prints requested signatures for a given file, delimited by spaces, all on one line with no extraneous information, one line per file.
Output options	
`[-h]` `[--hexadecimal]`	Display results in hexadecimal rather than base64 notation.
Signature selection	
`[-a]` `[--all]`	Display all signature function values (default).
`[-C]` `[--CRC]`	Display CRC-32, POSIX 1003.2 compliant 32-bit Cyclic Redundancy Check.
`[-M]` `[--MD5]`	Display MD5, the RSA Data Security, Inc.® Message Digest Algorithm.
`[-S]` `[--SHA]`	Display SHA, the NIST Secure Hash Algorithm (NIST FIPS 180).
`[-H]` `[--Haval]`	Display Haval value, a 128-bit signature code.
Input	
`file1 [ file2 … ]`	List of filesystem objects for which to display values.

6 APPENDICES

Introduction:

Tripwire's internal security relies on a private/public key system of cryptography. This chapter specifically describes Tripwire's cryptographic system. It also directs the reader to background information and references on cryptography.

Also included are descriptions of the cryptographic signature routines that Tripwire utilizes.

Chapter contents:

Tripwire's Cryptographic System

Tripwire uses public/private key cryptography to protect its critical data files. This chapter describes the Tripwire cryptography architecture, and the role of the site and local keys. It also directs the reader to background information and references on cryptography.

The Tripwire database, policy, and configuration files contain information critical to generating accurate reports. See Chapter 2, Running Tripwire, for more information on each of these files. Therefore, these files must be protected from unauthorized alteration. Allowing these files to be modified by an attacker compromises the security of the computer system that Tripwire has been installed to protect.

In the Tripwire ASR 1.3 (Academic Source Release) and earlier versions, storing these critical files on read-only media was required to ensure that they could not be modified by unauthorized individuals.

In Tripwire 2.0, these critical files are encoded and signed in a way that requires knowledge of a secret passphrase to modify them. This approach makes it computationally infeasible for an unauthorized individual to modify the files without detection. For this reason, it is no longer necessary to store the database, policy, and configuration files on read-only media.

Security Issue: If Tripwire files are stored on disk, an attacker who can not access the files can still delete them. It is unlikely that such an attack would go unnoticed and it is therefore very difficult for an attacker to silently subvert Tripwire operation. To address this issue, some sites may continue to store these files on read-only media, while other sites will rely on backup mechanisms to handle these contingencies.

Public and private keys

Tripwire files are protected with El Gamal asymmetric cryptography that uses public and private keys with a 1024 bit signature. Critical Tripwire files are stored on disk in an encoded and signed form. Reading these files requires only the public key, while writing them requires the private key and a secret passphrase.

Because one of the requirements for Tripwire is unattended operation, the public key is stored on disk along with the Tripwire executable. No one need type in the passphrase if Tripwire is set to run every morning at 3 AM, for instance.

When an encoded and signed file needs to be read, Tripwire uses the public key to remove encryption from the file. When one of these files needs to be modified, Tripwire uses the private key and prompts for the site or local passphrase. Only individuals authorized to modify Tripwire files should have access to these passphrases.

Public and private keys always work together in pairs. To simplify key management in Tripwire, both of these keys are stored together in a single *keyfile*.

Site and local keys

Tripwire protects the policy and configuration files with the same public/private key pair called the site key. The database file uses a separate public/private key pair called the local key. In the Tripwire configuration file, the LOCALKEYFILE and SITEKEYFILE variables specify where on disk the local and site key files, respectively, are stored.

Note: The configuration file embeds a copy of the public key. This is because key file locations are stored in the configuration file, which must be encoded and signed. Tripwire programs extract this public key and use it for verification when accessing configuration files.

Keyfiles are required for reading encoded Tripwire files. If a keyfile is lost, it is impossible to read the Tripwire files that have been encoded with that keyfile.

Standard security procedures should be in place to prevent discovery of passphrases by physical observation of keyboards or monitors (directly, via video camera, etc.) or by unauthorized electronic monitoring of your network. Note that the passphrases are not stored anywhere on disk. Therefore, they cannot be discovered by examining your system disks.

Security Issue: Although you can use the same private key for all machines in your network, it may not be advisable. Sharing a key amongst machines means that if the passphrase for that key is discovered, security of all machines in the domain could be compromised.

Of course, using different keys requires memorization or recording (in a secure place) of passphrases for all of the keys. For a large number of machines, this may be impractical.

A reasonable compromise for installations with numerous machines would be to divide the machines into a manageable number of groups, and assign a different private key and passphrase for each group. Tripwire can only use one site key and one local key per process. This means that the configuration and policy files for any given command must be protected by the same site key. Likewise, database and report files must be protected by the same local key. For example, if *host.db* was signed using *localkey1* and *host.twr* was signed with *localkey2*, the following command line will generate an error.

```
% ./tripwire --update --dbfile host.db --twrfile myhost.twr
```

Cryptography references

For background information and references on cryptography, please consult the Cryptography FAQ, available on the Internet via

anonymous FTP to:
rtfm.mit.edu
then proceed to:
/pub/usenet/news.answers/cryptography-faq/

The same information can be found on the Web at:
http://www.faqs.org/faqs/cryptography-faq/

Important:
In the aforementioned FAQ, an example is given in which public keys allow encryption only, and private keys allow decryption as well as encryption.

Tripwire's keys work in the opposite fashion. Public keys allow decryption only, while private keys allow encryption as well as decryption.

File signatures

When a Tripwire database is created, all pertinent attributes for objects to be monitored are stored in the database. However, to detect file changes, it is not sufficient merely to compare file attributes. For example, it is possible to change a file without changing its size or modification date (with the appropriate user privileges). This provides the motivation also to store *file signatures*, small fixed-size values that are generated from the object data (i.e. the file contents).

The Unix *sum*(8) is a trivial example of such a signature, called a checksum. However, because it merely adds all the values of all the characters in the file together, and truncates the sum to 16 bits, *sum*(8) is not a very suitable signature for Tripwire—it is easy to cover up a change and preserve the existing checksum.

Tripwire can utilize the MD5, SHA, and Haval message-digest algorithms, also known as one-way hash functions, finger-printing routines, or manipulation detection codes. Message-digests are usually large (at least 128 bits), and computationally infeasible to reverse. They employ cryptographic techniques to ensure that any small change in the input stream results in a large change in the output. Therefore, any unauthorized, malicious, or accidental change will be evident.

Because these algorithms use a 128-bit or larger signature, using a brute-force attack to introduce a deliberate change in the file while trying to keep the same signature becomes a computationally infeasible task. They provide a high degree of assurance that an intruder cannot change a file without being detected by Tripwire.

Message-digest algorithms have considerable computational cost when compared to simpler signature routines. In general, this higher security is obtained at the expense of speed—it will take longer to check your files with message digest algorithms. Tripwire provides several state-of-the-art message digest algorithms to allow the maximum amount of flexibility in deciding the security and performance balance.

Unlike message-digest algorithms, the CRC-32 algorithm uses simple polynomial division to generate the checksums. While this technique is very fast, the mathematics it employs are well-understood. Furthermore, since the 32-bit signature space is so small, a brute-force search for a CRC collision is well within the capabilities of most workstations. There are currently several programs in the public domain that can, for any given input file, provide a different output file with the same CRC signature.

Signature functions

Tripwire ships with four signature routines. This section briefly describes each signature routine. It is by no means an authoritative reference, but it provides some background on each of the signature routines provided.

All observed timing measures provided for the signature routines were performed using *siggen*(8) and the same input files under Solaris on a 167 MHz UltraSPARC 1 and under Linux on a Pentium 100 with 32 megabytes of RAM. The numbers provided are simply an informal gauge of throughput, rather than an authoritative metric.

Please note that the base64 encoding used by Tripwire for signature functions is the one described in RFC 1521.

MD5

MD5 is the RSA Data Security Inc.® Message-Digest Algorithm, a proposed data authentication standard. The Internet Draft submission can be found as Internet Working Draft RFC 1321, available from **http://www.merit.edu/internet/documents/**.

Alternatively, primary anonymous ftp repositories for RFCs include **nic.ddn.mil**, **wuarchive.wustl.edu**, and **nisc.jvnc.net**.

MD5 generates a 128-bit signature, and uses four rounds to ensure pseudo-random output.

Hardware	Throughput (Mbyte/sec)
167 MHz UltraSPARC 1	8.1
100 MHz Intel Pentium	3.1

SHA/SHS

SHS is the NIST Digital Signature Standard, called the Secure Hash Standard, and is described in NIST FIPS 180. Tripwire refers to it as the SHA, or Secure Hash Algorithm, because Tripwire uses a non-certified implementation and cannot claim standards conformance. SHS generates a 160-bit signature.

Hardware	Throughput (Mbyte/sec)
167 MHz UltraSPARC 1	1.5
100 MHz Intel Pentium	0.8

Haval

Haval was written by Yuliang Zheng at the University of Wollongong, and is described in

Zheng, Y., Pieprzyk, J., Seberry, J. (1993) “HAVAL: a one-way hashing algorithm with variable length of output” In Advances in Cryptology: AUSCRYPT’92, Lecture Notes in Computer Science, Springer-Verlag.

Haval is shipped with Tripwire configured similarly to the other signature algorithms: 128-bit signature using four passes.

Hardware	Throughput (Mbyte/sec)
167 MHz UltraSPARC 1	6.8
100 MHz Intel Pentium	3.6

CRC-32

Cyclic Redundancy Checks have long been the de facto error detection algorithm standard. These algorithms are fast, robust, and provide reliable detection of errors associated with data transmission. CRC-32 is provided as a fast but insecure alternative to the slower message-digest algorithms.

CRC-32 generates a 32-bit signature.

Hardware	Throughput (Mbyte/sec)
167 MHz UltraSPARC 1	15.2
100 MHz Intel Pentium	5.9

Tripwire quick reference

This section provides the Tripwire commands for the most common Tripwire tasks. The commands are presented in a practical, chronological order, from post-installation, to integrity checking, to file maintenance.

Note that the verbose GNU-style arguments are used in each example for clarity.

Desired action	Command
	Example
Getting started	
Generate site and local keys (usually done by installer).	Run *twadmin* in Key Generation mode.
	`./twadmin --generate-keys --local-keyfile ../key/local.key --site-keyfile ../key/site.key`
Install your customized configuration file.	Run *twadmin* in Create Configuration File mode.
	`./twadmin --create-cfgfile twcfg.txt`
Install your customized policy file.	Run *twadmin* in Create Policy File mode.
	`./twadmin --create-polfile ../policy/twpol.txt`
Initialize the Tripwire database.	Run *tripwire* in Database Initialization mode.
	`./tripwire --init --verbose`
Running and maintaining	
Compare system against baseline database, checking for policy violations.	Run *tripwire* in Integrity Check mode.
	`./tripwire --check --verbose`
Update Tripwire database for objects marked as policy violations.	Run *tripwire* in Interactive Update mode or in Database Update mode.

	`./tripwire --check --interactive`
	`Or:`
	`./tripwire --check --verbose`
	
	`./tripwire --update --verbose --twrfile /usr/TSS/report/`*`myhost-19981112-172634.twr`*

Update policy file to change files monitored by Tripwire.	Run *tripwire* in Policy Update mode.
	
	`./twadmin -m p >../policy/pol.txt`
	
	`vi ../policy/pol.txt`
	
	`./tripwire --update-policy ../policy/pol.txt`

Desired action	Command
	Example
Configuring	
Inspect policy file.	Run *twadmin* in Print Policy Mode.
	`./twadmin --print-polfile`
Inspect configuration file.	Run *twadmin* in Print Configuration mode.
	`./twadmin --print-cfgfile`
Inspect database file to ensure that files are being monitored.	Run *twprint* in Print Database mode.
	`./twprint --print-db --dbfile ../db/myhost.db`
Inspect an individual report file.	Run *twprint* in Print Report mode.
	`./twprint --print-report --twrfile ../report/`*`myhost-19981115-175828.twr`*
Encryption and key changes	

Change the local key.	Run *twadmin* in Remove Encryption mode, and then again in Encrypt File mode.
	
	Note: Make sure you make backup copies of these files before running these commands.
	
	`./twadmin --remove-encryption ../report/* ../db/*`
	
	`./twadmin --generate-keys --local-keyfile ../key/`*`myhost-`*`local.key`
	
	`./twadmin --encrypt ../report/* ../db/*`

Change the site key.	Run *twadmin* in Remove Encryption mode, and then again in Encrypt File mode. **Note:** Make sure you make backup copies of these files before running these commands.
	
	`./twadmin --remove-encryption ./tw.cfg ../policy/*`
	
	`./twadmin --generate-keys --site-keyfile ../key/site.key`
	
	`./twadmin --encrypt ./tw.cfg ../policy/*`

Usage	
Get *tripwire* usage message.	Print *tripwire* help.
	`./tripwire --help`
Get *twprint* usage message.	Print *twprint* help.
	`./twprint --help`
Get *twadmin* usage message.	Print *twadmin* help.
	`./twadmin --help`
Get *siggen* usage message.	Print *siggen* help.
	`./siggen --help`