twpolicy(4) - Linux man page

Name

twpolicy - *Tripwire* policy file reference

Description

The policy file describes system objects to be monitored by *Tripwire*, and specifies what properties for each object should be collected and stored in the database file. Each object in the policy file is associated with a property mask, which describes what changes to the file or directory *Tripwire* should monitor, and which ones can safely be ignored. By customizing the various aspects of the policy file, the system administrator can very closely control how *Tripwire* checks the integrity of any system.

During installation, an encoded and signed policy file (*tw.pol*) will be created in the */etc/tripwire* directory, and a plain text copy of this policy file (*twpol.txt*) will be generated in the same directory. An additional text file (*policyguide.txt*) illustrates all of the features of the policy language. Both of the text files are heavily commented and can be used as a reference during policy file creation.

A new policy file is first created using the **twadmin --create-polfile** command. With this command, the user can designate an existing plain text file as the current policy file. Using the current site key and passphrase, the new configuration file is encoded, signed and saved.

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. This is an important distinction--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.

Components Of The Policy File

The basic components of policy files are comments, rules, directives, and variables. Each of these components is described in greater 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 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* scans.

2) Stop points tell *Tripwire* not to scan a particular file or directory.

Normal Rules

The format for a normal rule is:

object_name -> property_mask;

where the *object_name* is the fully qualified pathname for a directory or file, and *property_mask* specifies what properties of an object to examine or ignore. The '->' token separates the pathname and the property mask. Whitespace must separate the object name and '->' token. 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 property mask. Examples:

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

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

# 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 any object has more than one rule in a policy file, *Tripwire* will print an error message and exit without scanning any files. For example:

```
# This is an example of an illegal construct.
/usr/bin   ->   $(mask3);
/usr/bin   ->   $(mask4);
```

Object Names

In this document, policy file objects are fully qualified pathnames of files and directories. Environment variables are not allowed for security reasons. Examples:

```
/etc         # valid object name.
/etc/passwd  # valid object name.
$HOME        # not valid.
```

Property Masks

Property masks designate which *Tripwire* properties of a given object should be examined. A property mask consists of a series of single-character symbols, each of which may be preceded by an optional plus or minus sign. Each character symbol stands for a particular *Tripwire* property to be examined 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.
```

Each rule in the policy file must have a property mask. Examples:

```
/etc        ->    $(IgnoreAll);     #valid property mask.
/etc        ->    +p-p;             #valid property mask.
/etc        ->    ;                 #invalid property mask.
```

Characters in a property mask without a preceding plus or minus sign are assumed to be plus. If a property is not specified in the property mask, it is ignored, which is equivalent to turning it off with the minus sign. Examples:

```
# Examine permissions and link count.
# All three of the following are equivalent.
+p+n
pn
pn-g
```

Characters used in property masks, with descriptions:

```
-      Ignore the following properties
+     Record and check the following properties
a     Access timestamp
b     Number of blocks allocated
c     Inode timestamp (create/modify)
d     ID of device on which inode resides
g     File owner's group ID
i     Inode number
l     File is increasing in size (a "growing file")
m     Modification timestamp
n     Number of links (inode reference count)
p     Permissions and file mode bits
r     ID of device pointed to by inode
      (valid only for device objects)
s     File size
t     File type
u     File owner's user ID
C     CRC-32 hash value
H     Haval hash value
M     MD5 hash value
S     SHA hash value
```

Stop Points

Stop points are used to specify specific files or directories that *Tripwire* should not scan. The syntax for stop points is:

```
!  object_name  ;
```

For example:

```
!/etc/init.d;
# The directory /etc/init.d will not be scanned.
```

```
/etc   ->   $(ReadOnly);
!/etc/rc.d;
!/etc/mnttab;
# Scan all of /etc, but do not scan two particular
# files in the /etc hierarchy.
```

Rule Attributes

Rule attributes work with normal rules to modify their behavior or provide additional information. Multiple attributes can be assigned to each rule. Rule attributes are not case-sensitive. Rule attributes may be applied to a single rule using the following syntax:

object_name -> property_mask (*rule attribute* = *value*);

For example:

```
/usr/lib -> $(ReadOnly) (emailto = admin@foo.com, severity = 80);
#This rule will notify the admin if any violations of the
#rule occur and designate the severity as 80.
```

Rule attributes can also be specified for a group of rules, using the format:

(*attribute list*)
{
rule list;
}

For example:

```
(emailto = admin@foo.com, severity = 80)
{
     /usr/lib  ->  $(ReadOnly);
}
```

is equivalent to the attribute example above.

The following four rule attributes are supported by *Tripwire*:

`rulename`

The `rulename` attribute is used to associate a rule or set of rules with a specific name. In a report file, this name will be associated with violations to the specified rule. This feature is useful if you want to track certain objects within a large *Tripwire* database. For instance, if you associate the rule name "watchme" with important files, you can sort through the *Tripwire* report using "watchme" as a sorting key.

Example:

```
/etc -> +ug (rulename=watchme);
```

emailto

The `emailto` attribute associates one or more email addresses with a rule or group of rules. When an integrity check is run with the **--email-report** option and a rule is violated, a report of that violation will be sent to the specified email address(es), using the report format specified by the

EMAILREPORTLEVEL variable in the configuration file.

Example:

```
/etc -> +ug (emailto=admin@domain.com);
```

To specify multiple email addresses, include them as a quoted, semicolon-delimited list.

```
/etc -> +ug (emailto="admin@foo.com;admin2@foo.com");
```

severity

The `severity` attribute associates a numeric 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 are used. The default severity level is 0, and values can range from 0 to 1,000,000.

Example:

```
/etc -> +ug (severity=50);
```

recurse

The `recurse` attribute specifies how a rule will scan directories. Valid values for `recurse` are *true*, *false*, or a number from -1 to 1,000,000. If `recurse` is set to *true* (or -1), tripwire will recursively scan the entire contents of the directory (both files and subdirectories). When `recurse` is set to *false* (or 0), and the rule refers to a directory, *Tripwire* will scan the inode of the directory but none of the files or subdirectories contained therein. For positive `recurse` value *n*, the rule will monitor all objects up to *n* levels below the start point. Stop points within the recursed directory still apply, and will prevent the specified file or directory from being scanned.

When a rule refers to a file, specifying the `recurse` attribute has no effect--files will be scanned no matter what value is given for `recurse`. The default value for `recurse` is *true*.

Example:

```
/etc -> +ug (recurse=2);
```

Directives

Tripwire supports a small number of directives that allow conditional interpretation of the policy file and certain diagnostic and debugging operations. The primary purpose of directives is to support sharing of a policy file among multiple machines. Directives use the following syntax:

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

Where the directive name is one of the directives listed below:

```
@@section  # Designates a section of the policy file.

@@ifhost   # Allow conditional interpretation
```

```
@@else     # of the policy file.
@@endif

@@print    # Print a message to standard output.
@@error    # Print a message to standard output and then exit.

@@end      # Marks the logical end-of-file.
```

The @@section directive is used to designate sections of the policy file that are OS-specific. With *Tripwire 2.4.1*, valid arguments for the @@section directive are FS and GLOBAL. If no @@section directive is specified, FS will be assumed. If the argument to the @@section directive is either NTFS or NTREG (which have meaning only on Windows NT systems), *Tripwire* will skip all policy file text down to the next valid @@section directive. Any other argument will cause an error.

The @@ifhost, @@else, and @@endif directives are used to allow conditional interpretation of the policy file. With the @@ifhost directive, multiple hostnames can be expressed as arguments, but they must be separated by an '||', interpreted as the logical 'OR'. This example illustrates how one might employ directives to use one policy file with multiple hosts.

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

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

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

```
@@print  "string"
@@error  "string"
```

The @@print directive prints *string* to *stdout*, while the @@error directive prints *string* to *stdout* and causes the calling program to exit with a non-zero status.

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

Variables

For user convenience, *Tripwire*'s policy file supports variables for string substitution. Variables can be defined anywhere between rules. The syntax for variable definition is:

```
variable  =  value;
```

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

```
$(  variable  )
```

Examples of variable definition and variable substitution on the left, right, and both sides of rules. Note that variable names are case sensitive.

```
param1 = +SMCH;          # Set variable param1.
dir1   = /etc/inet;      # Set variable dir1.
DIR1   = /etc/init.d;    # Variables are case sensitive.
$(dir1)   -> +tbamc;     # Left hand substitution.
/etc/inet -> $(param1);  # Right hand substitution.
$(DIR1)   -> $(param1);  # Double substitution.
```

A number of variables are predefined by *Tripwire* and may not be changed. These variables represent different ways that files can change, and can be used on the right side of rules to design a policy file quickly.
ReadOnly

ReadOnly is good for files that are widely available but are intended to be read-only.

Value: `+pinugtsdbmCM-rlacSH`

Dynamic

Dynamic is good for monitoring user directories and files that tend to be dynamic in behavior.

Value: `+pinugtd-srlbamcCMSH`

Growing

The Growing variable is intended for files that should only get larger.

Value: `+pinugtdl-srbamcCMSH`

Device

Device is good for devices or other files that *Tripwire* should not attempt to open.

Value: `+pugsdr-intlbamcCMSH`

IgnoreAll

IgnoreAll tracks a file's presence or absence, but doesn't check any other properties.

Value: `-pinugtsdrlbamcCMSH`

IgnoreNone

IgnoreNone turns on all properties and provides a convenient starting point for defining your own property masks. (For example, `mymask = $(IgnoreNone) -ar;`)

Value: `+pinugtsdrbamcCMSH-l`

Version Information

This man page describes *Tripwire 2.4.1*.

Authors

Tripwire, Inc.

Copying Permissions

Permission is granted to make and distribute verbatim copies of this man page provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this man page under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this man page into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by Tripwire, Inc.

Copyright 2000 Tripwire, Inc. Tripwire is a registered trademark of Tripwire, Inc. in the United States and other countries. All rights reserved.

See Also

twintro(8), **tripwire**(8), **twadmin**(8), **twprint**(8), **siggen**(8), **twconfig**(4), **twfiles**(5)