



**Tripwire<sup>®</sup> Intrusion Detection System  
1.3 for UNIX<sup>®</sup>**

**User Manual**

July 27th, 1998

## COPYRIGHT NOTICE

All files in this distribution of Tripwire® are Copyright 1992-1998 by the Purdue Research Foundation of Purdue University and are distributed by Visual Computing Corporation™ under exclusive license arrangements. All rights reserved. Some individual files in this distribution may be covered by other copyrights, as noted in their embedded comments.

This release is for single CPU, single-site, end-use purposes. Duplication is only allowed for the purposed of backup. Any other use of this software requires the prior written consent of Visual Computing Corporation. If this software is to be used on a Web site, the "Tripwire Protected" logo can be used on the site home page along with appropriate copyright and trademark information. Neither the name of the University nor the names of the authors may be used to endorse or promote products derived from this material without specific prior written permission.

THIS SOFTWARE IS PROVIDED AS IS AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR ANY PARTICULAR PURPOSE.



Visual Computing Corporation™  
([www.visualcomputing.com](http://www.visualcomputing.com))  
615 SW Broadway, 2<sup>nd</sup> Floor  
Portland, OR 97205

Phone: (503) 223-0280  
FAX (503) 223-0182

Email: [tripwire@visualcomputing.com](mailto:tripwire@visualcomputing.com)

## QUICK START

If you have used an earlier version of Tripwire, or if you are a new user, we suggest that you follow the steps below to properly configure, install, and use Tripwire:

- 1) Read the “BUILDING TRIPWIRE IDS 1.3 for UNIX” section.
- 2) Read the “INSTALLING AND RUNNING” section.
- 3) Read the FAQ Section.
- 4) Edit the include/config.h file to set the appropriate values for your site.
- 5) Go to single user mode for the remaining steps.
- 6) Reinstall your operating system binaries to be certain they have not been compromised.
- 7) Run Tripwire in database initialization mode.
- 8) Set the destination directory's disk to *read-only* in hardware, and/or take standalone signatures of all the files using the *siggen* utility.
- 9) Resume normal operations.

Note that you can build and install Tripwire without reading the documentation, without setting a read-only disk, and without reinstalling your binaries. You can also leave your seat belt unbuckled when you drive, or leave your door unlocked when you leave.

# TABLE OF CONTENTS

---

## TABLE OF CONTENTS

<b>COPYRIGHT NOTICE .....</b>	<b>2</b>
<b>QUICK START .....</b>	<b>3</b>
<b>TABLE OF CONTENTS .....</b>	<b>4</b>
<b>OPEN LETTER TO TRIPWIRE CUSTOMERS .....</b>	<b>7</b>
OPEN LETTER TO TRIPWIRE CUSTOMERS: .....	7
OPEN LETTER FROM PROFESSOR EUGENE SPAFFORD, PH.D.: .....	7
OPEN LETTER FROM PROFESSOR EUGENE SPAFFORD, PH.D., DECEMBER, 1997 .....	8
<b>BUILD TRIPWIRE® IDS 1.3 FOR UNIX.....</b>	<b>9</b>
BACKGROUND .....	9
GOALS OF TRIPWIRE .....	9
HOW TO BUILD TRIPWIRE .....	9
COMMON TRIPWIRE COMPILATION PROBLEMS .....	11
TESTING TRIPWIRE.....	11
<b>HOW TO CONFIGURE TRIPWIRE® .....</b>	<b>12</b>
SOME TRIPWIRE SCALING HINTS FOR USING TRIPWIRE IN LARGE SITES .....	12
THE TW.CONFIG GRAMMAR .....	12
HOW YOU MIGHT USE THESE DIRECTIVES .....	13
<b>NOTES ON SIGNATURE ROUTINES .....</b>	<b>13</b>
PERFORMANCE VS. SECURITY .....	13
SIGNATURE ROUTINES .....	14
MD5 .....	15
SNEFRU .....	15
CRC-32 .....	15
CRC-16 .....	16
MD4 .....	16
MD2 .....	16
SHA/SHS.....	16
HAVAL .....	17
NULL SIGNATURE.....	17
FEEDBACK AND BUG-REPORTS.....	17
USER CONTRIBUTIONS AND EXPERIENCES .....	17

# TABLE OF CONTENTS

---

<b>HOW TO RUN TRIPWIRE® .....</b>	<b>18</b>
CREATING YOUR TW.CONFIG FILE .....	18
A CAVEAT ABOUT YOUR TRIPWIRE DATABASE .....	18
RUNNING TRIPWIRE AS AN INTEGRITY CHECKER.....	19
KEEPING YOUR DATABASE UP-TO-DATE .....	19
RUNNING TRIPWIRE IN INTERACTIVE MODE.....	19
RUNNING TRIPWIRE IN DATABASE UPDATE MODE.....	19
QUICK INTEGRITY CHECKING MODE .....	20
THE SIGGEN UTILITY .....	20
<b>FREQUENTLY ASKED QUESTIONS .....</b>	<b>21</b>
CONCEPTS: .....	21
TROUBLESHOOTING: .....	22
<b>MANUAL PAGE: TW.CONFIG(5) .....</b>	<b>25</b>
NAME.....	25
SYNOPSIS .....	25
DESCRIPTION .....	25
ENTRY FORMAT.....	25
ENTRY EXAMPLES .....	27
PREPROCESSOR.....	27
CAVEATS.....	29
DATABASE VERSIONS .....	29
SEE ALSO .....	29
<b>MANUAL PAGE: TRIPWIRE(8) .....</b>	<b>30</b>
NAME.....	30
SYNOPSIS .....	30
DESCRIPTION .....	30
OPTIONS.....	31
DATABASE GENERATION MODE .....	32
DATABASE UPDATE MODE.....	32
INTEGRITY CHECKING MODE.....	32
DIAGNOSTICS .....	34
ENVIRONMENT .....	34
BUGS.....	35
SEE ALSO .....	35
<b>MANUAL PAGE: SIGGEN(8).....</b>	<b>36</b>
NAME.....	36
SYNOPSIS .....	36
DESCRIPTION.....	36
OPTIONS.....	36

# TABLE OF CONTENTS

---

<b>PRODUCT RELEASE AND DEVELOPMENT HISTORY .....</b>	<b>37</b>
RELEASE 1.3 (RELEASE) .....	37
RELEASE VERSION 1.25 .....	37
RELEASE VERSION 1.2 .....	38
RELEASE VERSION 1.1 .....	39
NEW TRIPWIRE DATABASE FORMAT: .....	40
UPDATING THE TRIPWIRE DATABASE: .....	40
TRIPWIRE EXIT CODES: .....	41
TRIPWIRE QUIET OPTION: .....	41
MONOTONICALLY GROWING FILES: .....	41
LOOSE DIRECTORY CHECKING: .....	42
HOOKS FOR EXTERNAL SERVICES: .....	42
CHANGE IN TW.CONFIG PREPROCESSOR: .....	42
EXPANDED TEST SUITE: .....	43
CRC32 CHANGES: .....	43
"SIGGEN" REPLACES "SIGFETCH": .....	43
SOURCE CODE CLEANUP: .....	43
BUG FIXES: .....	43
LIST OF THANKS: .....	44

## OPEN LETTER TO TRIPWIRE CUSTOMERS

### *Open Letter to Tripwire Customers:*

January, 1998

Dear Customer,

As President of Visual Computing Corporation, I am pleased to announce our acquisition of exclusive worldwide rights of Tripwire, developed at Purdue University in conjunction with many individual, corporate and institutional testers and users. Please find an announcement and transition letter from Professor Spafford of Purdue below.

Since its original release in 1992, Tripwire has grown in popularity and is now one of the most widely deployed UNIX security tools. Several hundred thousand software downloads have been recorded, and we are aware of many active large government and commercial sites using the product.

Our goals for Tripwire are to continue with the excellent effort put forth by the original authors and many contributors since the code was originally released. We are planning several incremental releases in the immediate term adding value to the core engine and user interface. Later in 1998 we will release a major new commercial version.

We encourage existing and new users of Tripwire to stay tuned to our web site at [www.visualcomputing.com](http://www.visualcomputing.com) for more information about future releases of Tripwire and other marketing related activities. If you have any specific input or questions, please direct these inquiries to [tripwire@visualcomputing.com](mailto:tripwire@visualcomputing.com).

Sincerely,

W. Wyatt Starnes  
President  
Visual Computing Corporation

### *Open Letter from Professor Eugene Spafford, Ph.D.:*

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 Visual Computing Corporation of Portland, Oregon.

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 Visual Computing 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, Visual Computing Corporation 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.

A representative of Visual Computing Corporation will be available to discuss transferring any current use of Tripwire into a formal site license arrangement.

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.

## ***Open Letter from Professor Eugene Spafford, Ph.D., December, 1997***

In mid-December 1997, Visual Computing Corporation acquired the license for our Tripwire change/intrusion detection system. They will be marketing an enhanced, supported version of Tripwire for Unix-based machines, starting sometime in the early part of 1998. They are also planning a Windows NT version of Tripwire for release sometimes in mid-late 1998. I may have some technical advisory role in these developments, but all enquiries about Tripwire should be directed to:

Visual Computing Corporation  
615 SW Broadway  
Portland, Oregon 97205  
Phone: (503) 223-0280  
FAX: (503) 223-0182  
Email: [tripwire@visualcomputing.com](mailto:tripwire@visualcomputing.com)

Please also note that Tripwire is a registered trademark of the Purdue Research Foundation, and it is also licensed to VCC.

Gene Spafford, Ph.D.

## **BUILD TRIPWIRE<sup>®</sup> IDS 1.3 FOR UNIX**

This documentation serves as a quick-and-dirty primer on Tripwire. A PostScript formatted paper that fully describes the design and rationale is also included in the `./docs` directory. The design document is also available as a technical report (TR-CSD-93/71). This document will be referenced throughout the Tripwire distribution as the Tripwire design document or the comprehensive Tripwire paper.

This README file contains information needed to build, test, and run Tripwire.

### ***Background***

With the advent of increasingly sophisticated and subtle account break-ins on Unix systems, the need for tools to aid in the detection of unauthorized modification of files becomes clear. Tripwire is a tool that aids system administrators and users in monitoring a designated set of files for any changes. Used with system files on a regular (e.g., daily) basis, Tripwire can notify system administrators of corrupted or tampered files, so damage control measures can be taken in a timely manner.

### ***Goals of Tripwire***

Tripwire is a file and directory integrity checker, a utility that compares a designated set of files and directories against information stored in a previously generated database. Any differences are flagged and logged, including added or deleted entries. When run against system files on a regular basis, any changes in critical system files will be spotted—and appropriate damage control measures can be taken immediately. With Tripwire, system administrators can conclude with a high degree of certainty that a given set of files remain free of unauthorized modifications if Tripwire reports no changes.

### ***How to build Tripwire***

As of this writing, Tripwire has run successfully on (at least) BSD, OSF/1, Mach, Xenix, and late System V variants of Unix. Tripwire was built and tested on a wide variety of Unix variants.

- 1) Please examine the README file before you modify any files or try to build the program. This text describes various settings and strategies for configuration and operation. Then, after you've thought about the operation of the system, come back to this section and follow the instructions.

The file 'Ported' contains a list of platforms and operating systems where Tripwire has already been successfully ported. If you find your system in the list, note the system settings that were used to build Tripwire.

- 2) Look through the Makefile and make sure that the C compiler and all flag settings are reasonable for your configuration. Most of the potentially tricky system settings should be listed in the 'Ported' file.
- 3) Next, look in the './configs' directory to find a predefined 'conf-<os>.h' file that matches closest to your operating system. Note this file, because you will be inserting this filename in the './include/config.h' file. If no such file exists, pick one "near" your system type and modify appropriately (then mail it back to us for a future release).

*Don't do anything with this file yet! In particular, don't copy it over the config.h file! (Instead, keep reading...)*

- 4) Now that you have chosen your operating system header file, edit the './include/config.h' file to tailor Tripwire to your system.

Include the name of the predefined header file closest to your system at the appropriate line in config.h.

Paths and names of Tripwire configuration files are also set in the config.h file. Make sure you note the locations that Tripwire looks for its configuration and database files; change them for your system, as appropriate.

**NOTE:** We strongly urge you to locate the Tripwire configuration files on a disk that can be made read-only with a hardware setting. This will prevent the files from being altered by an attacker. The run-time version of Tripwire should be located in the same place. If you are unable to mark a disk (or diskette) as read-only, you might also consider putting it on a remote partition of a more secure machine, and import it read-only. See the design document for the rationale behind this note if the concept is not obvious.

- 5) Last, look in the './configs' directory again to find a tw.config file that matches your operating system. These files were custom-tailored to match the file layouts of various vendor-supplied operating systems. If no file in this subdirectory matches your system, choose the one that is closest in nature (e.g., BSD4.3 or SysV.4). Edit this file to include additional files and directories you want to monitor (e.g., local bins and critical databases), to correct paths if you have moved things or if they are mounted from a remote location (check them only on the server!), and to exclude locally-active files from the check. You should probably add the Tripwire binary itself to this file. See the next section for further details.

After you have customized your tw.config file, copy it to the location that you specified in your config.h file.

- 6) After you have configured the build environment, simply type 'make' at the top level. Note that all Makefiles in the subdirectories are driven by the top-level Makefile. (i.e., typing 'make' in the ./src directory will probably not work.)

Some common problems with building and using Tripwire are addressed in the FAQ file: you may want to read that if you have difficulties.

## *Common Tripwire compilation problems*

Tripwire was originally written using ANSI C. However, Tripwire now compiles with K&R, too. All of the prototypes remain embedded between “`#ifdef __STDC__`” directives. Sadly, compiling under ANSI is sometimes noisier than with K&R. Therefore, go ahead and compile with K&R unless religion dictates ANSI. (The code lints completely clean, excepting the `malloc()` and `exit()` return values.)

Common compilation trouble-spots are the `dirent(S5)/direct(BSD)` funkiness and `#defines` that changed for POSIX compliance.

If the Tripwire test suite fails, consider trying the following:

- double check that you’re including the correct `conf-*.h` file in your `./include/config.h` file.
- change the `CFLAGS` definition in the Makefile so no optimization is done (i.e., remove the “`-O`” option).
- do a “make clean”.
- try again.

If this fails, try a different C compiler (e.g., GCC).

It has been noted that newer versions of flex and bison (the GNU replacements for lex and yacc) do not generate code that passes all the test suites using the default Tripwire sources. Calvin Page helped contribute replacement `config.pre.l` and `config.pre.y` files that correct this problem. You’ll find that using lex usually solves the problems that make the Tripwire test suite fail.

If you use flex and bison, make sure you use the gcc compiler to avoid compile-time errors.

Please note that the SHA code in `./sigs/sha` seems to be poorly handled by many optimizing C compilers. For example, the stock C compiler included with SunOS 4.x takes almost two minutes to compile this file with the `-O` option on a Sparcstation10. Other compilers (such as GCC) do not have this problem.

## *Testing Tripwire*

Tripwire includes a script-driven test suite that checks the top-level build directory against the distribution package.

In the `./tests` directory, there is a Tripwire database of the entire Tripwire source distribution and a `tw.config` file. The test script automatically converts the pathnames in these Tripwire files to match those of your system. After converting the files, it then runs Tripwire in Integrity Checking mode.

To run the test, simply type ‘make test’ at the top level. This will invoke the script, and if all goes well, the output of Tripwire matches the expected values that the script provides.

In addition to checking all the files in the Tripwire distribution, a number of signature and functional tests are run to ensure the correct operation of the compiled program.

## HOW TO CONFIGURE TRIPWIRE®

### *Some Tripwire scaling hints for using Tripwire in large sites*

The `tw.config.5` manual page describes in detail the syntax supported by the `tw.config` file. Tripwire includes features that offer similar functionality to the C-preprocessor, and offer other directives that assist in the use of Tripwire at sites consisting of hundreds of workstations with local disk.

### *The `tw.config` grammar*

These commands are briefly described below:

```
@@define VAR VALUE
@@undef VAR

@@ifhost HOSTNAME
@@ifnhost HOSTNAME
@@ifdef VAR
@@ifndef VAR
@@else
@@endif

@@include FILENAME
```

Furthermore, the `tw.config` grammar also supports logical expressions. For example, you could have something like this in your `tw.config` file:

```
@@ifhost spam.cc.purdue.edu || weiner.cc.purdue.edu
...entries...
@@endif
```

Besides the cpp-like functionality, you can use `@@define` to create strings that are interpreted at run-time.

For example:

```
@@ifhost mentor.cc.purdue.edu
@@ define TEMPLATE_S +pinug-cas0123456789
@@else
@@ define TEMPLATE_S +pinug012-cas3456789
@@endif

/etc/tw.loginfo          @@TEMPLATE_S
```

## *How you might use these directives*

Because Tripwire allows run-time interpretation of the tw.config file, it becomes possible for many different hosts to share the same tw.config file. This allows the maintenance of Tripwire configuration files to still be manageable in a large, heterogeneous environment. Although each host must still have different database file, this has few consequences except for disk space.

## Notes on signature routines

The RSA Data Security, Inc. MD5, MD4, and MD2 Message Digesting Algorithm, Snefru (the Xerox Secure Hash Function), SHA (the Secure Hash Algorithm), and Haval code have been changed to eliminate big-endian and little-endian run-time specific routines. These changes have been sent back to the authors, but we are not aware of any buy-backs yet. Until then, there will remain some differences between the code in this package and their respective virgin distributions.

## *Performance vs. security*

Normally, only one checksum per file would be enough to detect changes. For purposes of speed, an easy to calculate checksum would be preferred. However, most easy-to-calculate signatures are also easy to defeat if a determined attacker wished to do so (see the chart in the design document to see how easy this is to do with random comparisons).

Tripwire includes six very difficult-to-forge signature algorithms, as well as two more conventional CRC routines. Using the default setup of recording two signatures (MD5 and Snefru) for each database entry gives very, very strong assurance that a file has not been tampered with. For tampering to have succeeded, the attacker would have had to have changed the file and added appropriate padding characters to recreate **both** checksums without also altering the size of the file. To do this at random might not even be possible with the MD5 and Snefru checksums used. Those two algorithms have not been exhaustively analyzed, but both are known to be strong message authentication codes.

This added assurance is at a heavy price, however. The two algorithms, and Snefru in particular, are expensive to calculate. To run the MD5 and Snefru algorithms against every file is likely to be overkill for almost all systems (unless you have cpu cycles to spare!). Both checksums should be run over only the most critical files...like the Tripwire database and program, and perhaps each setuid and setgid file on your system. All other files can be checked with MD5 (or Haval) alone for much faster operation and a high level of assurance. The task of altering a file and recreating the original MD5 checksum is also very difficult, and it is unlikely that any but the most determined, sophisticated, and well-equipped attacker would be able to do it in finite time.

To decrease the execution run-times of Tripwire, consider modifying your tripwire.config entries to ignore the Snefru (signature 2) attribute on files that do not need such stringent monitoring. This will skip the computationally-expensive Snefru signature collection entirely.

Balancing this equation of security vs. speed is a decision best made by the administrator, most closely tailored to his/her site needs.

For the extremely paranoid, Tripwire includes the MD2, MD4, SHA, and Haval signature algorithms, as well as the 16 and 32-bit CRC algorithms in its arsenal. Be forewarned, however, that MD2 is an order of magnitude slower than even Snefru, and probably guarantees no greater integrity checking. We include all these routines, however, so you can pick what you feel to be most appropriate for your site.

You may wish to add other routines as checksum/signature generators. For instance, if you have a fast DES implementation (including chip-based generation), you might wish to encrypt the file using CBC mode and some fixed key, saving the final 128 bits of output as the signature. The configuration file routines have several signature flags that are currently bound to a null function, so there is room for this expansion if you wish.

Clearly, with eight different signature algorithms at your disposal, Tripwire offers considerable flexibility in ensuring data security. Tripwire makes maintaining a trivial CRC database equally easy to administer and check as a full (but perhaps less practical) eight-signature database.

The following section describes each of the eight signature algorithms.

## *Signature routines*

Tripwire ships with eight signature routines. This section briefly describes each signature routine. This is by no means an authoritative list, but it does attempt to give some background on each of the signature routines provided:

MD5, Snefru, MD4, MD2, SHA, and Haval are all examples of message-digest algorithms (also known as one-way hash functions, fingerprinting routines, message authentication codes, or manipulation detection codes). They employ cryptographic techniques to ensure that any small change in the input stream results in immediate and widely diverging output. This way, even a small change in the input results in large change in the output. Therefore, any unauthorized, malicious, or accidental change will be evident. Furthermore, 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.

The CRC algorithms, on the other hand, use simple polynomial division to generate the checksums. While this technique is very fast, the mathematics of this technique is well-understood. Additionally, since the signature space is so small (usually 16 or 32 bits), 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 in 30 seconds or less.

All observed timing measures provided for the signature routines were performed on a Sequent Symmetry with ten 16 Mhz 80386 processors. The numbers provided are simply an informal gauge of throughput, rather than any authoritative metric.

## ***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 via anonymous FTP from NIC.DDN.MIL or from RSA.COM as ~/pub/md5.doc. MD5 attempts to address potential security risks found in the speedier, but less secure MD4, also by RSA Data Security Inc. MD5 was designed as a more conservative algorithm that backs “away from the edge” in terms of risks from successful cryptanalytic attack.

MD5 generates a 128-bit signature, and uses four rounds to ensure pseudo-random output. Observed throughput is about 70 Kbytes/second.

Currently, MD5 is considered by many to be a state-of-the-art signature algorithm.

## ***Snefru***

Snefru, the Xerox Secure Hash Function, was developed by Ralph Merkle at Xerox PARC. As an incentive to find a Snefru crack, there is a \$1000 cash prize promised to anyone who can find two sets of input that map to the same signature.

This reward has remained unclaimed since April 1990, when the 2-pass version of Snefru was broken by Eli Biham, a Ph.D. student of Adi Shamir. Currently, Ralph Merkle recommends using only the 4-pass version of Snefru, if not the 8-pass version. The Snefru README states, “Further study of the security of Snefru is required before production use is advisable.”

As shipped with Tripwire, Snefru is configured to run in 4-passes. Version 2.5 is the latest version available, and is the version included with Tripwire.

Snefru is slower than MD5, but is recommended as a backup for MD5 as a primary signature. As configured, Snefru runs at about 31 Kbytes/second.

Snefru can be obtained via anonymous FTP from arisia.xerox.com in directory /pub/hash.

## ***CRC-32***

Cyclic Redundancy Checks have long been the de facto error detection algorithm standard. These algorithms are fast, robust, and provides reliable detection of errors associated with data transmission. It has been shown that CRC-32 has a minimum distance of 5 for block lengths of less than 4K. However, this decreases as the size of the blocks increases. Therefore, using CRC-32 on long files is certainly a misapplication of this signature algorithm. However, CRC-32 is provided as a fast and speedy alternative to the slower message-digest algorithms. The version of CRC-32 included with Tripwire was written by Gary S. Brown.

This CRC-32 implementation runs at about 111 Kbytes/second.

## ***CRC-16***

CRC-16 is the predecessor to CRC-32, using only 16 bits to store the remainder of the data and the generator polynomial. CRC-16 is typically at the link level, usually done in hardware to detect transmission errors.

This CRC-16 implementation runs at about 131 Kbytes/second.

## ***MD4***

MD4, the RSA Data Security Inc. Message-Digest Algorithm, is the predecessor to MD5 described above. It was also submitted as a standard data authentication algorithm, and is described in the Internet Working Draft 1320.

The MD4 algorithm was designed to exploit 32-bit RISC architectures to maximize throughput. On a Sun SparcStation, throughput rates of over 1.4 Mbytes/second are achieved.

MD4 can be obtained via anonymous FTP from RSA.COM in ~/pub.

On a Sequent, MD4 throughput is about 332 Kbytes/second.

## ***MD2***

The RSA Data Security, Inc. MD2 Message-Digest Algorithm was created as part of the Privacy Enhanced Mail package—a package designed to authenticate and increase the security of electronic mail. Like the other algorithms by RSA Data Security, Inc presented here, MD2 generates a 128-bit signature.

The MD2 algorithm is quite slow. On a 16 Mhz 80386, expect only 3 Kbytes/second. It is not clear that using this slower algorithm instead of MD5 brings any comparative advantage.

The license for MD2 specifically states its use is exclusive to the Privacy Enhanced Mail package. Provisions have been made with RSA Data Security, Inc. for its inclusion and use in Tripwire in its present form. Note that MD2 is not in the public domain.

## ***SHA/SHS***

SHS is the NIST Digital Signature Standard, called the Secure Hash Standard. It is described in NIST FIPS 180. We refer to it as the SHA, or Secure Hash Algorithm, because we are using a non-certified implementation and we cannot claim standards conformance.

SHA is about one-half as fast as MD5. It has been noted that SHS appears to be largely based on MD4 with several key enhancements, not all implemented in MD5. The mid-1994 correction to the algorithm at the behest of NSA (reflected in the default version compiled into Tripwire; see sigs/sha/sha.h) raises some questions about the overall strength of both SHA and MD4 in the minds of some cryptographers.

## ***Haval***

Haval was written by Yuliang Zheng at the University of Wollongong, and is described in Y. Zheng, J. Pieprzyk and J. Seberry:

“HAVAL --- a one-way hashing algorithm with variable length of output”, Advances in Cryptology --- AUSCRYPT'92, Lecture Notes in Computer Science, Springer-Verlag, 1993.

Haval is shipped with Tripwire configured similarly to the other signature algorithms: 128 bit signature using four passes. Configured this way, Haval throughput is approximately 100K/sec. (30% faster than MD5.)

## ***Null signature***

Well, sig\_null\_get() is not really a signature algorithm. Instead, it is a place holder for unused slots in the signature array. It will always return a single character, “0”.

## ***Feedback and bug-reports***

Please send any bug-reports, questions, feedback, or any comments to  
[tripwire@visualcomputing.com](mailto:tripwire@visualcomputing.com).

## ***User contributions and experiences***

The ./contrib directory contains several programs contributed by users during the beta-test period. Each program is accompanied by a README file written by the program author.

## HOW TO RUN TRIPWIRE<sup>®</sup>

Tripwire runs in either of four modes: Database Generation, Integrity Checking, Database Update, and Interactive Update mode. In order to run Integrity Checking, Tripwire must have a database to compare against. To do that, you must first specify the set of files for Tripwire to monitor. This list is stored in `tw.config`.

### *Creating your `tw.config` file*

Edit your `'tw.config'` file, or whatever filename you defined for the Tripwire config file, and add all the directories that contain files that you want monitored. The format of the config file is described in its header and in the man page. Pay especially close attention to the `select-flags` and `omit-lists`, which can significantly reduce the amount of uninteresting output generated by Tripwire. For example, you will probably want to omit files like mount tables that are constantly changed by the operating system.

Next, run Tripwire with `'tripwire -initialize'`. This will create a file called `'tw.db_[hostname]'` in the directory you specified to hold your databases (where `[hostname]` will be replaced with your machine hostname).

### *A caveat about your Tripwire database*

NOTE: Tripwire will detect changes made to files from this point on. You must be certain that the system on which you generate the initial database is clean, however --- Tripwire cannot detect unauthorized modifications that have already been made. One way to do this would be to take the machine to single-user mode, reinstall all system binaries, and run Tripwire in initialization mode before returning to multi-user operation.

This database must be moved someplace where it cannot be modified. Because data from Tripwire is only as trustworthy as its database, choose this with care. We recommend placing all the system databases on a read-only disk (you need to be able to change the disk to writable during initialization and updates, however), or exporting it via read-only NFS from a "secure-server." (This pathname is hardcoded into Tripwire. Any time you change the pathname to the database repository, you must recompile Tripwire. This prevents a malicious intruder from spoofing Tripwire into giving a false "okay" message.)

We also recommend that you make a hardcopy printout of the database contents right away. In the event that you become suspicious of the integrity of the database, you will be able to manually compare information against this hardcopy. We have yet to hear of a way for "crackers" to alter an old piece of printout made before they penetrated the system!

You may also wish to generate a full set of signatures of the database, the configuration file, and the Tripwire executable using the "siggen" utility. (Be certain to generate siggen's signature too!) Store these on hardcopy for comparison if you need quick confirmation that the files involved have not changed. However, we advise that you do any comparison via a version of siggen stored on read-only media, or encrypted when not in use.

## ***Running Tripwire as an integrity checker***

Once you have your database set up, you can run Tripwire in Integrity Checking mode by 'tripwire'.

## ***Keeping your database up-to-date***

A common setup for running Tripwire would mail the system administrator any output that it generates. However, some files on your system may change during normal operation, and this necessitates update of the Tripwire database.

There are now two ways to update your Tripwire database. The first method is interactive, where Tripwire prompts the user whether each changed entry should be updated to reflect the current state of the file, while the second method is a command-line driven mode where specific files/entries are specified at run-time.

## ***Running Tripwire in Interactive mode***

Running Tripwire in Interactive mode is similar to the Integrity Checking mode. However, when a file or directory is encountered that has been added, deleted, or changed from what was recorded in the database, Tripwire asks the user whether the database entry should be updated.

For example, if Tripwire were run in Interactive mode and a file's timestamps changed, Tripwire would print out what it expected the file to look like, what it actually found, and then prompt the user whether the file should be updated:

```
/homes/genek/research/tw/src/preen.c
  st_mtime: Wed May  5 15:30:37 1993      Wed May  5 15:24:09 1993
  st_ctime: Wed May  5 15:30:37 1993      Wed May  5 15:24:09 1993
---> File: '/homes/genek/research/tw/src/preen.c
---> Update entry? [YN(y)nh?] y
```

You could answer yes or no, where a capital 'Y' or 'N' tells Tripwire use your answer for the rest of the files. (The 'h' and '?' choices give you help and descriptions of the various inode fields.)

While this mode may be the most convenient way of keeping your database up-to-date, it requires that the user be "at the keyboard." A more conventional command-line driven interface exists, and is described next.

## ***Running Tripwire in Database Update mode***

Tripwire supports incremental updates of its database on a per-file/directory or tw.config entry basis. Tripwire stores information in the database so it can associate any file in the database with the tw.config entry that generated it when the database was created.

Therefore, if a single file has changed, you can:

```
tripwire -update /etc/newly_installed.file
```

Or, if an entire set of files that made up an entry in the tw.config file changed, you can:

```
tripwire -update /usr/local/bin/Local_Package_Dir
```

In either case, Tripwire regenerates the database entries for every specified file. A backup of the old database is created in the ./databases directory.

Note that Tripwire can now handle arbitrary numbers of arguments in Database Update mode. This was added in version 1.0.1.

The script “twdb\_check.pl” was added in version 1.2 as an interim mechanism to ensure database consistency. Namely, when new entries are added to the tw.config file, database entries may no longer be associated with the proper entry number. The twdb\_check.pl script analyzes the database, and remaps each database entry with its proper tw.config entry.

The twdb\_check functionality will be put into the Tripwire program in a future release.

## ***Quick Integrity Checking mode***

Tripwire allows you to selectively skip certain signatures at run-time through a command-line option. For example, if you wish to run Tripwire on an hourly basis, even performing only MD5 checks might be computationally prohibitive. For this application, checking only the CRC32 signature might be desirable. To do this, assuming that only MD5, Snefru, and CRC32 were used when the database was initialized, you would type:

```
tripwire -i 1 -i 2
```

This tells tripwire to ignore signature 1 and signature 2. Furthermore, for daily Tripwire runs, you could specify using only MD5 and CRC32. Finally, for weekly runs, you could run Tripwire with all three signatures.

To find added or deleted files, with no signature checking, use:

```
tripwire -i all
```

## ***The siggen utility***

The siggen utility is provided so users can get signatures of files without having to run Tripwire. The syntax of siggen is simple.

```
siggen [-0123456789aqv] [ file ... ]
```

By default, siggen prints out all ten signatures. However, the signatures can be printed selectively by specifying the signature number on the command line. See the manual page for details.

## FREQUENTLY ASKED QUESTIONS

Listed below are the most frequently asked questions about Tripwire. The first section of the file covers Tripwire concepts and design, while the second section addresses troubleshooting.

### *Concepts:*

**Q: Does Tripwire suffer from any Year 2000 (Y2K) problems?**

**A:** There are currently no known Year 2000 problems. All versions of Tripwire use POSIX and ISO 9899 compliant library functions to store and represent dates. Dates are stored as the number of seconds since January 1, 1970, and are therefore not impacted by dates after Y2K. Similarly, to display dates and times.

Tripwire uses the POSIX `ctime()` function, used by virtually every other UNIX utility. Dates are represented by 26 characters (e.g., Fri Sep 13 00:00:00 2003), so dates after Y2K can be unambiguously displayed.

**Q: Why doesn't Tripwire ever traverse across mounted file systems?**

**A:** This is intentional. This behavior makes it possible to put a directory (e.g., '/') in your `tw.config` file, and you won't have to worry whether it will traverse all the locally-mounted file systems. If you want it include the whole file system, list each partition separately in the configuration file.

**Q: What is the difference between pruning an entry in your `tw.config` file (via "?") and ignoring everything (via the "E" template)?**

**A:** Ignoring everything in a directory still monitors for added and deleted files. Pruning a directory will prevent Tripwire from even looking in the specified directory.

**Q: Tripwire runs very slowly. What can I do to make it run faster?**

**A:** You can modify your `tw.config` entries to skip the Snefru signatures by appending a "-2" to the ignore flags. Or you can tell Tripwire at run-time to skip Snefru by:

```
tripwire -i 2
```

This computationally expensive operation may not be needed for many applications. (See README section on security vs. performance trade-offs for further details.)

## *Troubleshooting:*

**Q: I build Tripwire and the test suite fails. What do I do?**

**A:** Read the README section on “Common Compilation Problems.”

**Q: Tripwire reports that my database version is out of date. What should I do?**

**A:** The database format used by Tripwire v1.2 changed. You need to rebuild the database with Tripwire v1.2; see the README file for details.

**Q: Where do I find Larry Wall’s patch program?**

**A:** You can get it via anonymous FTP at <ftp.uu.net/pub/patch.tar.Z>.

**Q: When running Tripwire in Integrity Checking mode, Tripwire fails when it tries to find a file with a name consisting of dozens/hundreds/thousands of ‘/’s. What went wrong?**

**A:** Your setting for the #define DIRENT value in your conf-<os>.h file is probably incorrect. Try switching the setting and see if the problem goes away. (i.e., switch #define to #undef, or vice versa.)

**Q: I have /tmp in my tw.config file, but none of the files in the directory are being read by Tripwire. What’s going on?**

**A:** Check to see that your /tmp directory isn’t a symbolic link to another file system. When recursing down into directories, Tripwire never traverses symbolic links or enters another file system.

**Q: Is there any way I can get Tripwire to print out the names of the files as they are being scanned? I want to know which files Tripwire is spending all of its time crunching.**

**A:** Try using 'tripwire -v'. This wasn't documented in the first tripwire.8 manual page.

**Q: I try to initialize the database by typing 'tripwire -initialize' but I can't find the binary. Where is the tripwire executable?**

**A:** ./src/tripwire is where the binary is built. 'make install' will install in the \$(DESTDIR) of your choice, as defined in the top-level Makefile.

**Q: I have the following line in my tw.config file to do host specific actions. Why doesn't it work?**

```
@@ifhost chapel || chekov || chewie || data || guinan
    ....
@@endif
```

**A:** You must put the hostnames as returned by 'hostname' or 'uname' (depending on whether you're running a BSD or SYSV derived OS). So, the correct form would be:

```
@@ifhost chapel.enterprise.fed || chekov.enterprise.fed ...
```

The Tripwire preprocessor tries its best to figure out if you have used mis-formed hostnames.

**Q: As part of my operational security plan for my exported NFS partitions, I want to run "fsirand" regularly. Unfortunately, if I do this, Tripwire will complain that every file has changed (because the i-node numbers will change). I don't want to rebuild the entire system database each time. What can I do?**

**A.** We have included a Perl script in the distribution that will go through a Tripwire db file (the output database) and update the i-node fields while leaving everything else the same. To use it, you need to modify the first line to point to your Perl interpreter (if you don't have Perl, you'll need to write your own program in C or get Perl from an ftp site).

The Perl script is ./contrib/twdb\_newinode.pl.

After the next time you are in single-user mode running fsirand, run this script with the db as input. For example,

```
cd /usr/local/adm/tcheck/databases
./twdb_newinode.pl tw.db_myhost
```

It will automatically create a backup version of the file for you as a “just in case.”

\*Afterwards, be certain to set the disk with the database back to read-only!!\*  
Also, store the Perl script in the same secure place as the Tripwire program.

Last updated: July 27th, 1998. Mail comments to: [tripwire-support@visualcomputing.com](mailto:tripwire-support@visualcomputing.com)

## MANUAL PAGE: TW.CONFIG(5)

### *Name*

tw.config - configuration file for Tripwire

### *Synopsis*

tw.config

### *Description*

The *tw.config* file contains the list of files and directories to be scanned by Tripwire. Information on these files is collected and stored in the *tw.db* database file. Stored with each *tw.config* entry is a *selection-mask* that describes what changes Tripwire can safely ignore without reporting to the user (e.g., access times-tamp).

The first section in this manual page describes the entry format in *tw.config* for the files monitored by Tripwire. The second section describes the preprocessing directives that Tripwire provides. These directives, which provide functionality similar to the C preprocessor and M4 macro processor, allow Tripwire to makebindings at run-time. This allows system administrators to use common *tw.config* files across multiple machines - or even across an entire site.

### *Entry Format*

Each entry in *tw.config* is a single line in the following form:

**Format:** `[!|=] entry [select-flags | template] [# comment]`

*entry* An *entry* is the absolute pathname of a file or a directory. Without any prefixes, the *entry* is added to the list of files to be scanned.

Note that directories listed in the *tw.config* file are recursively descended. However, file systems are never crossed. (I.e., if */usr* and */usr/local* are separate file systems, a */usr* in *tw.config* entry will not scan files that reside in the */usr/local* filesystem.)

*!* Inclusive prune. Prunes *entry* from the list of files to be scanned. If *entry* is a file, the file is removed from the list of files. If *entry* is a directory, the directory and all of its children are removed from the list of files.

- = Exclusive prune. Does not prune *entry*, but does prune its children. This has no effect if *entry* is a file. This option is useful for monitoring directories with transient files (e.g., /tmp and /var/tmp).
- select-flags* *select-flags* describe inode and file attributes. *select-flags* either specifies Tripwire to ignore changes in a specific attribute, or to report them. *select-flags* are provided in the form: [ [+ / ] [pinugsam123456789] ... ]
- ignore the following attributes
  - + record and check the following attributes
  - 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
  - s* size of file
  - a* access timestamp
  - m* modification timestamp
  - c* inode creation/modification timestamp
  - 0* signature 0 - null signature
  - 1* signature 1 - MD5, the RSA Data Security, Inc. Message Digesting Algorithm.
  - 2* signature 2 - Snefru, the Xerox Secure Hash Function.
  - 3* signature 3 - CRC-32, POSIX 1003.2 compliant 32-bit Cyclic Redundancy Check.
  - 4* signature 4 - CRC-16, the standard (non-CCITT) 16-bit Cyclic Redundancy Check.
  - 5* signature 5 - MD4, the RSA Data Security, Inc. Message Digesting Algorithm.
  - 6* signature 6 - MD2, the RSA Data Security, Inc. Message Digesting Algorithm.
  - 7* signature 7 - SHA, the NIST Secure Hash Algorithm (NIST FIPS 180)
  - 8* signature 8 - Haval, a strong 128-bit signature algorithm
  - 9* signature 9 - null signature (reserved for future expansion)
- templates* *template* are predefined sets of *select-flags* that are commonly used by system administrators.

The following templates have been pre-defined to make these long *select-masks* descriptions unnecessary.

- R* [R]ead-only (+pinugsm12-ac3456789) (*default*)
- L* [L]og file (+pinug-sacm123456789)
- N* ignore [N]othing (+pinugsamc123456789)

*E* ignore [E]verything (-pinugsamc123456789)  
> monotonically growing file (+pinug>-samc1233456789) - the “>” indicates that file changes are ignored only when the file is smaller than the last recorded size. This is useful for log files that are expected to grow.

By default, Tripwire uses the R template. Because it applies the set of *select-flags* {+pinugsml2-a3456789}, Tripwire ignores those changed files where only the access timestamp changed.

You can combine the use of templates with *select-flag* modifiers. The following entry monitors only changes in user-id and group-id information.

```
/etc/lp E+ug
```

## Entry Examples

The following entry will scan all the files in /etc, and report any changes in mode bits, inode number, reference count, uid, gid, modification and creation timestamp, and the signatures. However, it will ignore any changes in the access timestamp.

```
/etc +pinugsml2-a
```

It is equivalent to:

```
/etc R
```

The following example shows a very simple *tw.config* file that monitors selected directories.

```
/etc R # all system files
!/etc/lp R # ...but not those logs
=/tmp N # just the directory, not its files
```

Note the difference between pruning (via “!”) and ignoring everything (via “N” template): ignoring everything in a directory still monitors for added and deleted files but pruning a directory will prevent Tripwire from even looking in the specified directory for any changes.

*Hint:* Is Tripwire running too slowly? Modify your *tw.config* entries to use only a few signatures (e.g., signatures1 and 5) when this computationally-exorbitant protection is not needed. (See README and design document for further details.)

## Preprocessor

Tripwire incorporates a general purpose preprocessor that parses the *tw.config* file in one-pass. Available preprocessing directives include file inclusion, macro defines, conditionals based upon hostname or macros, and on-the-fly macro substitution. These directives provide C-preprocessor and m4-like capabilities.

The Tripwire preprocessor was included to allow its scalable use at large sites, allowing system administrators to reuse *tw.config* files by either including component files or having multiple machines share a common *tw.config file*.

<i>@@ifhost HOSTNAME</i>	includes text until matching <i>@@endif</i> if the machine host-name matches the specified <i>HOSTNAME</i> . Remember that you must use the exact hostname that <i>uname(1)</i> or <i>host-name(1)</i> returns. This usually implies that you must use the fully qualified hostname (e.g., <i>mentor.cc.purdue.edu</i> ).
<i>@@ifnhost HOSTNAME</i>	includes text until matching <i>@@endif</i> if the machine host-name does not match the specified <i>HOSTNAME</i> .
<i>@@else</i>	provides if-else semantics to preprocessor.
<i>@@define VAR STRING</i>	defines variable <i>VAR</i> to <i>STRING</i> . If the second argument is not provided, then a null-string is assigned to <i>VAR</i> .
<i>@@undef VAR</i>	clears the definition associated with variable <i>VAR</i> .
<i>@@ifdef VAR</i>	includes text until the matching <i>@@endif</i> or <i>@@else</i> if the variable <i>VAR</i> has been defined.
<i>@@ifndef VAR</i>	includes text until matching <i>@@endif</i> or <i>@@else</i> if the variable <i>VAR</i> has not been defined.
<i>@@endif</i>	closes up <i>@@ifhost</i> , <i>@@ifnhost</i> , <i>@@ifdef</i> , and <i>@@ifndef</i> .
<i>@@include "PATHNAME"</i>	reads in the specified source file. The double-quotes are optional.
<i>@@VAR</i>	substitutes the definition of <i>VAR</i> with the <i>@@define</i> 'ed value.
<i>@@{VAR}</i>	substitutes the definition of <i>VAR</i> with the <i>@@define</i> 'ed value.

*Example:* A host-dependent inclusion can be specified many ways so *tw.config* files can be shared among multiple machines. So, if the machine "mentor.cc.purdue.edu" is the only machine that has a certain file, you could use:

```
@@ifhost mentor.cc.purdue.edu
/etc/tw.log.mentor      R
@@endif
@@define ARCHIVE      R
/etc/tw.log             @@ARCHIVE
```

## *Caveats*

Although Tripwire provides hooks for ten different signature routines, using all ten would certainly be overkill in almost any imaginable situation. However, having up to ten signature routines in your signature arsenal allows system administrators considerable flexibility in finding the balance between performance and security for their specific site. This is the reason for supplying CRC-16 and CRC-32, which are trivially simple to spoof. These routines are not secure, but they are faster than the message-digesting routines.

## *Database Versions*

Tripwire v1.0 used database version 1. Database version 2 changed the base-64 alphabet so that “0” retained its traditional value.

Database version 3 changed the base-64 encoding so that all the bits were packed, reducing the size of 160-bit signatures from 30 characters to 27 characters. Tripwire v1.1 used database version 3. The program *twconvert* is provided to convert from the older database formats to version 3.

Tripwire v1.2 uses database version 4, supporting signatures for symbolic links and more consistent handling of entry numbers. (Note that *twconvert* cannot convert older database versions to database version 4. These databases will have to be regenerated.)

## *See Also*

tripwire(8), twconvert(8)

## MANUAL PAGE: TRIPWIRE(8)

### *Name*

tripwire - a file integrity checker for UNIX systems

### *Synopsis*

**tripwire** [ *options ...* ]

### *Description*

Tripwire is a file integrity checker - a utility that compares a designated set of files and directories against information stored in a previously generated database. Added or deleted files are flagged and reported, as are any files that have changed from their previously recorded state in the database. When run against system files on a regular basis, any file changes would be spotted when Tripwire is next run, giving system administrators information to enact damage control measures immediately.

Using Tripwire, system administrators can conclude with an extremely high degree of certainty that a given set of files and directories remain untouched from unauthorized modifications, provided the program and database are appropriately protected (e.g., stored on read-only media). Note that reports of changed files indicate a change from the time of the last Tripwire database installation or update. For best effect, the files being monitored should be reinstalled from known good sources. (See the Tripwire design document for further details.)

Tripwire uses message-digest algorithms (one-way hash functions) to detect changes in a hard-to-spoof manner. This should be able to detect significant changes to critical files, including those caused by insertion of backdoors or viruses. Tripwire also monitors changes to file permissions, modification times, and other significant changes to inodes as selected by the system administrator on a per-file/directory basis.

Tripwire runs in one of four modes: Database Generation, Database Update, Integrity Checking, or Interactive Update mode. In Database Generation mode, Tripwire initializes the database based upon the entries enumerated in the *tw.config* file. Database Update mode provides incremental database update functionality on a per-file/directory basis. This obviates having to regenerate the entire database every time a file or set of files change. The Integrity Checking mode generates a report of added, deleted, or changed files, comparing all the files described by the *tw.config* file against the files residing on the filesystem. Lastly, the Interactive Update mode reports added, deleted, and changed files and prompts the user whether those database entries should be updated.

The Interactive Update mode provides a simple and thorough method for system administrators to keep Tripwire databases “in sync” with file systems that change.

## Options

When run without any arguments, *tripwire* runs in Integrity Checking mode.

<b>-initialize</b>	Database Generation mode. Creates the database which is used for all subsequent Integrity Checking runs.
<b>-update pathname/entry ...</b>	Database Update mode. This mode updates the specified <i>pathname</i> or <i>entry</i> in the database. If the argument provided is a file, only that file is updated. If the argument is a directory, that directory and all of its children are updated. If the argument is an <i>entry</i> in the <i>tw.config</i> file, the entire entry in the database is updated.
<b>-interactive</b>	Interactive Integrity Checking. Tripwire first reports all added, deleted, and changed files, then prompting the user whether the entry should be updated in the database. Note that Tripwire opens up <i>/dev/tty</i> instead of using <i>stdin</i> . This prevents automating interactive updates, reducing the chance of system administrators inadvertently updating entries. Updating the database should always be done with care and deliberation.
<b>-loosedir</b>	Loosens checking rules for directories in Integrity Checking modes so changes in size, nlink, modification and creation times no longer are reported. This significantly quiets Tripwire reports, at the possible risk of missing important changes.
<b>-d dbasefile</b>	Reads the database information from the specified file <i>dbasefile</i> . <i>stdin</i> can be specified by “-d -”.
<b>-c configfile</b>	Read the configuration information from the specified file <i>configfile</i> . <i>stdin</i> can be specified by “-c -”.
<b>-cfd openfd</b>	Read the configuration information from the open file descriptor <i>openfd</i> . This option allows programs outside of Tripwire to supply services such as networking, compression, and encryption.
<b>-dfd openfd</b>	Read the database file from the open file descriptor <i>openfd</i> . This option allows programs outside of Tripwire to supply services such as networking, compression, and encryption.
<b>-Dvar=value</b>	Defines the <i>tw.config</i> variable <i>var</i> to <i>value</i> . (As if @@ <i>define</i> were used.)
<b>-Uvar</b>	Undefine the <i>tw.config</i> variable <i>var</i> . (As if @@ <i>undef</i> were used.)
<b>-i [# all]</b>	Ignore the specified signature, and skip it when comparing against database entries. If <i>all</i> is specified, no signatures are collected or compared.

<b>-E</b>	Prints out preprocessed <i>tw.config</i> file to <i>stdout</i> .
<b>-preprocess</b>	Same as -E option.
<b>-q</b>	Quiet mode. In this mode, Tripwire prints only one line reports for each added, changed, or deleted file. Phase 5 is skipped, which prints all the pairs of expected and observed file attribute values.
<b>-v</b>	Verbose mode. Prints out filenames as they are being scanned during signature computation.
<b>-help</b>	Print out inode interpretation message (for parsing messages when files have changed).
<b>-version</b>	Prints out version information.

## ***Database Generation Mode***

In Database Generation mode, *tripwire* creates the database file based upon the entries in *tw.config*. The name of this database file is defined at compile-time in *config.h* - it defaults to *tw.db\_[hostname]*. The generated database is placed in the **./databases** directory, and must be moved to the target directory manually.

Note that you must manually move this file to your database directory. This is because the default database directory should be a read-only file system.

## ***Database Update Mode***

In Database Update mode, *tripwire* updates the specified files, directories, or entries in the database. The old database is saved in the **./databases** directory with the *.old* suffix. The new, updated database is also written to the **./databases** directory. As in the Database Generation mode, the new database must be manually moved to the Tripwire database directory. *tripwire* in Database Update mode requires at least one argument, which is used as an *entry*. The *entry* argument specifies which file or directory is to be updated, and is interpreted similar to *tw.config* entries. If the argument is a filename, only that file is updated in the database. Similarly, if the argument is a directory name, the directory and its children are updated. If the argument is also an entry in the *tw.config* file, the entire entry is updated.

Database updates yield a new database file with added, deleted, or changed entries. This functionality is provided to allow Tripwire databases to be updated in a controlled manner to reflect filesystem changes, obviating the need to regenerate the entire database again.

## ***Integrity Checking Mode***

In Integrity Checking mode, *tripwire* reads in the *tw.config* file, and rebuilds a new database to reflect the current files. Tripwire then compares the new database with the

existing Tripwire database stored on the filesystem, reporting added or deleted files, as well as those files that have changed.

The *tw.config* file, in addition to the list of files and directories, also lists which attributes can change and be safely ignored by Tripwire. Tripwire applies these *select-flags* to decide which changes can be safely unreported.

Each file that differs from the information stored in the database is considered “changed.” However, only the changes that remain after the *select-flags* are applied are displayed. For each change, the expected and actual information is printed. For instance:

```
2:30am (mentor) 985 % tripwire
### Phase 1:  Reading configuration file
### Phase 2:  Generating file list
### Phase 3:  Creating file information database
### Phase 4:  Searching for inconsistencies
###
###          Total files scanned:      82
###          Filesadded:              0
###          Filesdeleted:            0
###          Fileschanged:            80
###
###          After applying rules:
###          Changesdiscarded:       79
###          Changesremaining:       1
###
changed: -rw----- genek 4433 Oct 13 02:30:34 1992 /tmp/genek/tripwire-
0.92/config.h
### Phase 5:  Generating observed/expected pairs for changed files
###
### Attr      Observed (what it is) Expected (what it should be)
### =====
/tmp/genek/tripwire-0.92/config.h
st_size: 4441          4433
          md5 (sig1): 0aqL1006C3Fj1YBXz3.CPdcB 0cPX1H.DYS.s1vZdKD.ELMDR
snefru (sig2): 0PcgCk/MZvEm.8pIWe.Gbnn/ /8VoJv1JcoUA0NvoGN.k3P6E
crc32 (sig3): .EHA6x /OuGNV
crc16 (sig4): ...9/q ...6yu
md4 (sig5): /hQ0sU.UEbJo.UR4VZ/mNG/h .UR4VZ/mNG/h/VSG/W/Z643k
md2 (sig6): .hLwjB.VRA00.Z72y90xTYqA 1LR0Gg1l.vqB0.lg330Pi8/p
```

Tripwire in Interactive Update mode will look similar. However, for each added, deleted, or changed file, the user is prompted whether the entry corresponding to the file or directory should be updated. The user can answer with either “y”, “n”, “Y”, or “N”. The first two answers are simply “yes, update the specified file” and “no, don’t update the file” respectively.

Answering “Y” not only updates the specified file or directory, but all other files or directories that share the same *entry* in the *tw.config* file. For example, if “Y” were answered for /etc, then all the files generated by the /etc entry will also be updated. Answering “N” is similar, but skips all files and directories corresponding to the specified entry.

A possible Tripwire session running in Interactive Update mode may look like:

```
3:34pm (flounder) tw/src 5 %%% tripwire -interactive
### Phase 1: Reading configuration file
### Phase 2: Generating file list
### Phase 3: Creating file information database
### Phase 4: Searching for inconsistencies
###
### Total files scanned: 49
### Filesadded: 0
### Filesdeleted: 0
### Fileschanged: 49
###
### Afterapplying rules:
### Changesdiscarded: 48
### Changesremaining: 1
###
changed: -rw----- genek 7893 May 5 15:30:37 1993
/homes/genek/research/tw/src/databases/tw.db_flounder.Eng.Sun.COM.old
### Phase 5: Generating observed/expected pairs for changed files
###
### Attr Observed (what it is) Expected (what it should be)
### =====
=====
/homes/genek/research/tw/src/databases/tw.db_flounder.Eng.Sun.COM.old
st_mtime: Wed May 5 15:30:37 1993 Wed May 5 15:24:09 1993
st_ctime: Wed May 5 15:30:37 1993 Wed May 5 15:24:09 1993
---> File:
'/homes/genek/research/tw/src/databases/tw.db_flounder.Eng.Sun.COM.old'
---> Update entry? [YN(y)nh?] y
### Updating database...
###
### Phase 1: Reading configuration file
### Phase 2: Generating file list
### Phase 3: Updating file information database
### Phase 3: Updating file information database
###
### Old database file will be moved to 'tw.db_barnum.cs.purdue.edu.old'
### in ./databases.
###
### Updated database will be stored in
'./databases/tw.db_barnum.cs.purdue.edu'
### (Tripwire expects it to be moved to '/tmp/genek'.)
###
3:34pm (flounder) tw/src 6 %%%
```

## ***Diagnostics***

Tripwire exit status is 1 for any error condition. Otherwise, the exit status is the logical OR'ing of the following: 2 for files added, 4 for files deleted, and 8 for files changed. (e.g., if Tripwire exits with status code 10, then files were added and change.  $8 + 2 = 10$ .)

## ***Environment***

None.

## ***Bugs***

This manual page is not self-contained - users are referred to the Tripwire design document to better understand the issues of integrity checking.

## ***See Also***

tw.config(5)

*The Design and Implementation of Tripwire: A UNIX File Integrity Checker* by Gene Kim and Eugene Spafford. Purdue Technical Report CSD-TR-93-071.

## MANUAL PAGE: SIGGEN(8)

### *Name*

siggen - signature gathering routine for Tripwire

### *Synopsis*

**siggen** siggen [-0123456789aqv] [*file ...* ]

### *Description*

*siggen* is a utility that fetches signatures used by Tripwire.

### *Options*

When run without any arguments, *siggen* collects all the signatures on the specified files in a verbose manner.

When no arguments are given, *siggen* reads from standard input.

- h** prints the signature in hexadecimal instead of base-64.
- q** Quiet mode. Prints out all signatures in one line with no extraneous information.
- a** Generates all signatures. (default)
- v** Verbose mode. (default)
- 0** signature 0 - generates null signature
- 1** signature 1 - generates MD5, the RSA Data Security, Inc. Message Digesting Algorithm.
- 2** signature 2 - generates Snefru, the Xerox Secure Hash Function.
- 3** signature 3 - generates CRC-32, POSIX 1003.2 compliant 32-bit Cyclic Redundancy Check.
- 4** signature 4 - generates CRC-16, the standard (non-CCITT) 16-bit Cyclic Redundancy Check.
- 5** signature 5 - generates MD4, the RSA Data Security, Inc. Message Digesting Algorithm.
- 6** signature 6 - generates MD2, the RSA Data Security, Inc. Message Digesting Algorithm.
- 7** signature 7 - generates SHA, the NIST Secure Hash Algorithm (NIST FIPS 180)
- 8** signature 8 - generates Haval, a 128-bit signature code
- 9** signature 9 - generates null signature (reserved for future expansion)

## PRODUCT RELEASE AND DEVELOPMENT HISTORY

### ***Release 1.3 (release)***

Original Release: Fri Jul 10, 1998

Current Build: July 21, 1988

Version 1.3 integrates bug fixes for all known bugs, as well as solving the problem database consistency errors that could creep in when doing database updates. This fix obviates the need for "twdb\_check.pl" that used to be included with earlier releases.

In addition, several Tripwire reports have been changed slightly, reflecting the feedback that sent over the last year: integrity checking reports are more succinct, and much more quiet when there are no notable changes.

- Fixed database entry consistency bug.
- Fixed database filename construction routine.
- Made "loosedir" reporting the default. Makes superfluous directory changes go away.
- Made reports more succinct, and much more quiet when there's nothing worth reporting.
- Updated manual.
- Added Visual Computing Corporation banner to startup.
- Eliminated RCS banners for any changed files (RCS no longer being the source control system for our source archives).
- Pulled out user manual (.doc and .pdf files) out of Tripwire package. These items will be distributed separately.
- Removed twdb\_check.pl from Tripwire package.
- Updated README, README.FIRST, and COAST.info files.
- Aux directory is now util, to accommodate DOS FAT filename restrictions.

### ***Release Version 1.25***

Original Release: April 10, 1998

Current Build: April 18, 1998

Version 1.25 for Red Hat Linux adds RPM support, as well as several other changes. Among the changes are:

- Added RPM support for Red Hat Linux and automated build and installation for Red Hat Linux systems.
- Changed default select-masks to use only MD5, instead of MD5 and Snefru.
- Added tw\_isalnum() to workaround Linux libc bug.
- Updated tw.conf.linux file.
- Made filename\_escape() eight-bit clean.

- twdb\_check.pl now works with Perl 5.
- Changed test scripts to sleep 2 before doing Tripwire checks.
- Changed aux to util in src/Makefile.

## ***Release Version 1.2***

Original Release: July 15, 1994

Current Build: March 31, 1998

Version 1.2 adds several new features, as well as fixing reported bugs. Among the changes are:

- Signature checking for symbolic link contents has been added.
- Tripwire now correctly runs on Alpha AXP's, and other machines with "long" types that are not 32 bits wide.
- The Haval digital hash routine has been added as the eighth signature routine (faster than MD5, and purportedly more secure).
- The SHA signature routine has been changed to conform to the recent fix introduced in its FIPS definition by NIST/NSA to correct an unspecified weakness.
- The database format changes slightly to correct a boundary condition error. Because database entry numbers change, because the SHA signatures change, and because of Haval, old Tripwire databases must be reinitialized.
- Handling specified configuration and database files (and file descriptors) has been fixed to better accommodate pipes.
- Full support for flex added.
- Signature checking is now considerably faster through the use of the stdio library for file I/O.
- A Perl script has been added to update Tripwire databases where all inode numbers were changed by "fsirand" (NFS sites only);

See FAQ.

- Another fix to make database updates more predictable.
- All reported bugs have been fixed in this revision.
- A new README section describes some documented attacks on systems running Tripwire.
- Many small changes have been made to the documentation to correct and update information.

NOTE: The script 'twdb\_check.pl' (written in Perl) has been added to the distribution. It checks database consistency after updates of the tw.config file. This functionality will be put into the Tripwire program in the next release. Run this script after Tripwire database updates to ensure that database entry numbers are consistent with the tw.config file. See the README file for details (section 3.5.2).

Numerous other people played a crucial role in shaping this release:

Paul Szabo is responsible for a number of fixes and cleaning up the way filenames are handled internally, leading to a much simpler treatment to filenames with escaped characters in Tripwire.

Paul Hilchey is responsible for rewriting the list routines, fixing a number of bugs and replacing a hideous implementation with one that is elegant and succinct.

Asokan is responsible for going through all the Tripwire code to remove assumptions about the size of certain types (e.g., "long" is not 32 bits on the Alpha).

Casper H.S. Dik pointed out how some signature routines used very small reads, leading to suboptimal performance. He offered a simple fix through the use of the stdio library for file I/O.

Cal Page modified the lex and yacc files to accomodate newer versions of GNU flex and bison, which continue to diverge from the traditional tools.

Tom Orban spent many an afternoon on the phone with me, guiding me step-by-step to find elusive database update bugs. Among other things, Tom Orban and Terry Kennedy helped track down the problems that led to the addition of the `twdb_check.pl` script.

Keith Rickert and Eugene Zaustinsky painstakingly pointed out distribution errors. Keith Rickert and Greg Black helped us with the last batch of fixes that shortly preceded this release. We appreciate their help.

Thanks go to those people who helped test Tripwire:

Eric Berg, Eric M. Boehm, Lothar Butsch, John Crosswhite, Jason Downs, Peter Evans, Jon Freivald, Kevin Johnson, Lothar Kaul, Terry Kennedy, Chris Kern, Paul Madden, Fred Marchand, Mitchell Marks, Jim Moreno, Tom Orban, Lorraine Padour, Calvin Page, Tom Painter, Roger Peyton, Peter Phillips, Keith W. Rickert, Jim Roche, D Seddon, Paul Szabo, Gene C Van Nostern, John Wiegley, Robert Wilhite, Alain Williams, Eugene Zaustinsky.

## ***Release Version 1.1***

Version 1.1 considerably upgrades the functionality of Tripwire. All known bugs have been fixed, and many selected features have been added at the request of Tripwire users.

Among the major changes are:

- rewrite of the "-update" command.
- addition of an "-interactive" command that prompts the user whether a changed file's database entry should be updated.
- addition of a "-loosedir" command for quieter Tripwire runs.
- support for monotonically growing files in `tw.config`.
- addition of comprehensive test suite to test Tripwire functionalities.
- hooks for external services (i.e., compression, encryption, networking) through "-cfd" and "-dfd" options.

- addition of the new NIST SHA/SHS signature algorithm.
- corrections and changes in the MD2, MD4, MD5, CRC32, and Snefru signature routines.
- addition of a more rigorous signature test suite.
- more error checking in tw.config @@directives.
- siggen replaces sigfetch.
- addition of a tw.config file for Solaris v2.2 (SVR4).
- change of base-64 alphabet to conform to standards.
- preprocessor macro fixes.

## *New Tripwire database format:*

The Tripwire database format has changed since v1.0, using a different base-64 alphabet and encoding scheme. Use the twconvert program to convert v1.0 databases to v1.1 databases (located in the ./src directory).

If you have been using an older version of Tripwire, you will need to use twconvert convert your databases to the new format.

## *Updating the Tripwire database:*

There has been a major rewrite/rethink of the “tripwire -update” command, as well as the addition of a “tripwire -interactive” command which allows the user to interactively select which database entries should be updated. No vestiges of the “-add” or “-delete” command remain, since the “-update” command now automatically deletes and adds files.

However, the preferred way of keeping Tripwire databases in sync with the filesystems is using the “-interactive” command. A Tripwire session using Interactive mode might look like:

```
6:25am (flounder) tw/src 1006 %% tripwire -interactive
### Phase 1:  Reading configuration file
### Phase 2:  Generating file list
### Phase 3:  Creating file information database
### Phase 4:  Searching for inconsistencies
###
###                               Total files scanned:      49
###                               Files added:              0
###                               Files deleted:            0
###                               Files changed:           49
###
###                               After applying rules:
###                               Changes discarded:        47
###                               Changes remaining:       2
###
changed: drwx----- genek          1024 May  3 06:25:37 1993
/homes/genek/research/tw/src
changed: -rw----- genek          7978 May  3 06:24:19 1993
/homes/genek/research/tw/src/databases/tw.db_flounder.Eng.Sun.COM.old
### Phase 5:  Generating observed/expected pairs for changed files
###
### Attr          Observed (what it is)          Expected (what it should be)
### =====
/homes/genek/research/tw/src
```

```
st_mtime: Mon May 3 06:25:37 1993      Mon May 3 06:11:39 1993
st_ctime: Mon May 3 06:25:37 1993      Mon May 3 06:11:39 1993
---> File: '/homes/genek/research/tw/src'
---> Update entry? [YN(y)nh?] y

### Updating database...
###
### Phase 1:  Reading configuration file
### Phase 2:  Generating file list
### Phase 3:  Updating file information database
###
### Warning:  Old database file will be moved to
`tw.db_flounder.Eng.Sun.COM.old'
###          in ./databases.
###
6:25am (flounder) tw/src 1007 %%
```

Tripwire prompts the user whether the database entry of the current file should be updated to match the current file information. Pressing either 'y' or 'n' either updates the current file or skips to the next file. Pressing 'Y' or 'N' applies your answer to the entire entry. (I.e., if /etc is changed, typing 'Y' will not only update /etc, but it will also files update all the files in /etc.)

### ***Tripwire exit codes:***

Tripwire exit status can be interpreted by the following mask:

```
1:    run-time error. aborted.
2:    files added
4:    files deleted
8:    files changed
```

For example, if Tripwire exits with status code 10, then files were found added and changed. (i.e.,  $8 + 2 = 10$ .)

### ***Tripwire quiet option:***

When run with -q option, Tripwire really is quiet, printing only one-line reports for each added, deleted, or changed file. The output is more suitable for parsing with *awk* or *perl*.

### ***Monotonically growing files:***

The ">" template is now supported in the tw.config files. This template allows files to grow without being reported. However, if the file is deleted or is smaller than the size recorded in the database, it is reported as changed.

## ***Loose directory checking:***

This option was prompted by complaints that Tripwire in Integrity Checking and Interactive mode unnecessarily complains about directories whose nlink, ctime, mtime, or size have changed.

When Tripwire is run with the “-loosedir” option, directories automatically have these attributes included in their ignore-mask, thus quieting these complaints.

Note that this option is not enabled by default, making normal Tripwire behavior no different than previous releases. However, running with this option enabled considerably decreases “noise” in Tripwire reports.

(Ideally, this “loose directory checking” should be offered on a per-file basis in the tw.config file. However, adding another field to the tw.config file was too extensive a change to be considered for this release. A later release of Tripwire may rectify this.)

## ***Hooks for external services:***

Tripwire now supports the “-cfd” and “-dfd” option that allows the user to specify an open file descriptor for reading the configuration file and database file, respectively. Using these options, an external program can feed Tripwire both input files through open file descriptors. This external program could supply services not provided though Tripwire, such as encryption, data compression, or a centralized network server.

This program might do the following: Open the database and configuration files, process or decode (i.e., uncompress the file), and then write out the regularly formatted file to a temporary file. Open file descriptors to these files are then passed to Tripwire by command-line arguments though execl().

An example of using a shell script to compress and encrypt your files is given in ./contrib/zcatcrypt. It is a four line Bourne shell script that encrypts and compresses the database and configuration files.

## ***Change in tw.config preprocessor:***

The tw.config preprocessor has been changed to allow the proper expansion of @@variables in filenames. The following use of @@define now works as expected:

```
@@define DOMAIN_NAME      my_main_nis_domain
/var/yp/@@DOMAIN_NAME    L
@@DOMAIN_NAME/FOO L
```

(This is the third attempt at getting this working correctly. We finally fixed this by moving the macro expansion routines into the lexical analyzer.)

## ***Expanded test suite:***

The Tripwire test suite now includes runs a more standard signature test suite. This was prompted by discovery of several implementation errors in the MD2, MD4, and MD5 signature routines that was introduced right before the official release of Tripwire. (Thanks Eugene Zaustinsky.)

Two more test suites have been added. One iterates through all the Tripwire reporting functionalities, and exercises all the database update cases. The other test suite checks for proper Tripwire preprocessor macro expansions.

## ***CRC32 changes:***

Furthermore, the CRC32 signature routine is now POSIX 1003.2 compliant. (Thanks Dan Bernstein.)

## ***"siggen" replaces "sigfetch":***

As a tester noted, “sigfetch” was a misnomer since nothing was actually being fetched. Consequently, it was easy to (incorrectly) conclude that “sigfetch” retrieved signatures from the database.

The “siggen” command is the current incarnation of “sigfetch”. The manual pages reflect this change.

## ***Source code cleanup:***

The authors went through the sources, doing generic cleanups aid in code comprehension.

## ***Bug fixes:***

This release fixes all known bugs. The TODO list, however, gives a wish list of features that may be included in future releases.

*List of thanks:*

Special thanks go to the testers of disappearing v1.0.3. Reports of critical bug fixes go to (in no special order): E. Clinton Arbaugh, Pat Macdonald, Eric Demerling, John Rouillard, Bob Cunningham, and Neil Todd.

Sam Gassel, Edward DeHart, Drew Gonczi, Rik Farrow, Jim Napier, Drew Jolliffe, John Rouillard, Alain Brossard, Eric Bergren, Patrick Sullivan, Nora Hermida, Juergen Schmidt, Debbie Pomerance, Michael Hines, Tim Ramsey, Georges Tomazi, Mitchell Marks, Philip Cox, Kevin Dupre', Chris Kern, and Eugene Zaustinsky helped in getting the Tripwire v1.1 release in shape for our December 1993 release.