Tripwire - an integrity checker, Part 1

SAFETY FIRST!

KLAUS BOSAU

Tripwire has developed into a highly-regarded instrument for administrative investigation. It's hardly surprising. The advanced concepts employed by this security tool provide opportunities that go far beyond those of its various competitors.

You don't have to be paranoid to be concerned about the subject of data security in these times of universal networking and multi-user operating systems. Woeful experiences with love-crazed e-mail viruses and Trojan horses that are all too eager for knowledge leave many a computer user calling for a powerful weapon. Unfortunately, in reality some popular aids aren't really worth very much. Which is why it's worth taking a look at a special kind of tool that, though primarily intended for administrators, can certainly be put to use by ambitious private users of UNIX-compatible operating systems.

Built on sand

Despite their popularity, conventional protective shields found in the PC domain, such as *virus scanners* or *signature scanners*, suffer from weaknesses. One example is the fact that the properties of new viruses – the so-called virus signatures – have to be determined by the manufacturer and incorporated into the customer's database before the virus scanner can detect the threat. In view of the number of attack points (which even operating systems like Linux offer) and the creativity of resourceful attackers this isn't an easy problem to resolve.

A different approach to the problem would be a tool that quickly detects the damage caused by a security attack. Changes to the file system as a consequence of viral or human activity may be too non-specific to be recognised on the basis of documented virus signatures. In many cases the only change consists of a hidden back door that a successful hacker has left behind in the system for further visits.

Security through inventory

Because these dangerous extras are easy to camouflage as regular system files, a search can only be successful given precise knowledge of the original state of the file system. This is something that has been known for years. It has led to the development of tools that take an approach known as *integrity checking*. One example that runs in the UNIX environment is *Tripwire*.

Tripwire was developed as part of the COAST Project (Computer Operations, Audit and Security Technology) of Purdue University, located in West Lafayette (Indiana). The launch at the start of 1994 (described in the online issue of the business magazine Forbes) was followed by a victory parade through the world of UNIX-type operating systems. This reached a peak of over 300,000 installations

and the winning of the "Editors' Choice Award '99" of the web magazine Linux-World.

The claim made by its developers, Gene Kim and Eugene Spafford was not exactly modest. The intention was to provide administrators with a tool that was as simple as possible to manage but nevertheless flexible, and which would make it possible to reliably and rapidly trace any kind of unauthorised or unintentional intrusion into the file system.

Conception

The difference in principle between an integrity checker and a signature scanner is evident even from the name. Whilst the latter aims to detect already-known programs, integrity checkers are designed to protect the integrity of data in the filesystem. They are able to detect any change, not just the documented changes that are produced when a known virus attacks a system.

In the simplest case, an integrity checker could make a backup of the entire filesystem and periodically compare it with the current content. However, this isn't really practical. Fortunately, there is a more elegant method. For effective identification of a change, a copy of the original isn't needed. Knowledge of a few special characteristics of the filesystem is perfectly adequate. The task when developing *Tripwire* consisted of extracting these pieces of data in a way that would facilitate subsequent monitoring.

Other design aims included a reduction in volume of the recorded data, and the use of a process that was not too computation-intensive. Kim found these properties combined in signature functions which were until then the domain of message encryption (so-called *One-Way Hash Functions* or *Message-Digest Functions*) like *MD5*

The Arsenal: Comparison of various hash algorithms for Tripwire

Algorithm	Data throughput (Pentum/200 MHz)	Security approximate checkpoint	Comments
MD5 (R)	7.2 MB/s	+++++	Message-Digest 5 Algorithm by Ronald Rivest (advanced development of MD4) -- a more powerful and currently the most-used hash algorithm. As with MD4, pseudocollisions (collisions for the *Compression Function*) are found, but to date its fundamental effectiveness has not yet been refuted.
Snefru (R)	1.4 MB/s	++++	This algorithm, conceived by Ralf Merkle at Xerox PARC, has rapidly become suspect in the four-stage variant used here despite its so-far undisputed effectiveness, since the two-stage version turned out to be unusable very early on. The large 128-bit signature should still, however, guarantee good security performance. (The latest variant to date with 8 rounds is deemed secure.)
CRC-32/CRC-16	9.3/16.2 MB/s	++/+	Both of these robust and fast algorithms were actually designed to detect transmission errors due to hardware. The size of the signature alone, with just 16 and 32 bit, prohibits any use with large or important files. But since a spurious file, apart from the appropriate signature, also has to bring with it the intended functionality to be of any use, it's certainly worth risking its use for less critical objects.
MD4	14.4 MB/s	+++	Introduced in 1990 and very fast on RISC processors. It wasn't until 1998 that disenchantment set in: an easily modified version proved to be reversible. MD4 is now seen as disproved and should therefore no longer be used to protect important objects.
MD2	300 KB/s	++++	Unusually slow, since it was the only one designed for antiquated 8-bit processors. Although MD2 is the oldest of the three message-digest algorithms from RSA, until now its effectiveness has been unquestioned. However, for a modified version the principal risk was revealed to be a constructive (and thus time-saving) determination of cases of collisions.
SHA	5.4 MB/s	+++++	The "Secure-Hash-Algorithm" from NIST [11] is, like most hash algorithms, structurally related to MD4. It was superseded in 1994 by SHA-1, which was supposed to correct an undocumented weak point. The large 160-bit signature nevertheless still makes this a good choice, even for security-critical objects. (The persistent rumours about a deliberately-implanted weak point, in view of the paltry state of knowledge in the theory of hash functions, are somewhat impudent. NASA, by the way, prefers this in its in-house Tripwire installation to the originally more popular Snefru. [12])
Haval	10.7 MB/s	++++	Was created in 1992 at the University of Wollongong by Yuliang Zheng [13] . It is the only one to hold both a variable signature size (128, 160, 192, 224 or 256 bit), as well as a variable number of work steps (3, 4 or 5). (The message is split into blocks of 1024 bits which are then processed in 3, 4 or 5 rounds by the *Compression Function*.) This means a total of 15 different variants of the algorithm are available for practical applications. In the Academic Source Release the 4-stage variant with 128-bit signature format is used.
RIPEMD-160 (*Tripwire-De-Luxe*)	4.9 MB/s	++++++	The "RACE Integrity Primitives Evaluation" algorithm [14] from the EU project of the same name is the offspring of a collaboration by well-known European cryptographers (Hans Dobbertin, Antoon Bosselaers and Bart Preneel) from 1996. The unconventional idea of two pipelines working in parallel after the MD4 model was first realised in RIPEMD (3 rounds, 128 bit). RIPEMD-160 represents a massive advance, which is also intended to take into account the most extreme demands made on collision freedom. In a total of five rounds each with 16 individual operations, a hash value of 160 bits is calculated. Together with SHA-1, RIPEMD-160 is currently viewed as the most powerful hash algorithm in existence.

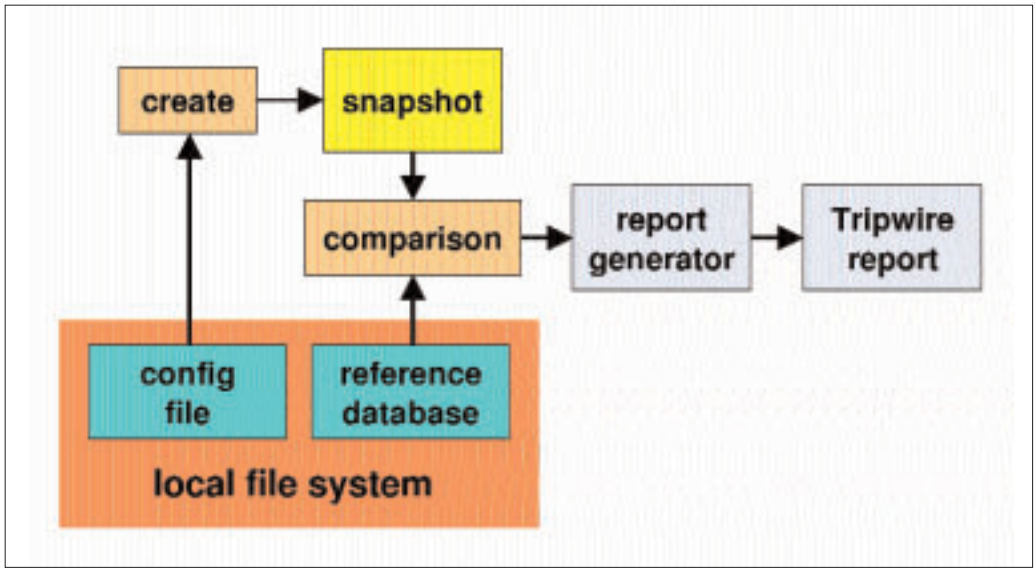

Figure 1: Block diagram of the basic functionality

from RSA Data Security, Inc. or *Snefru* from Xerox PARC. The departure from the intended use of such algorithms for effective protection of a filesystem is a central component of *Tripwire*.

Some advantages

An attractive quality of the integrity checker is the ability of the user to exert control over the process. It is easy to adjust the security performance according to your preference or need. There is no need to go to great lengths to maintain as complete as possible a library of virus signatures, as is necessary with a signature scanner. An integrity checker demands little attention. Once installed, maintenance is limited to a few actions that need only be performed when software is added or removed.

The performance of current signature algorithms turns the integrity checker into the ideal instrument for *perpetuation of evidence*. This is something that could open up entire new perspectives in the near future, especially in the corporate environment where the economic damage caused by a hacker attack could be considerable. A *One-Way Hash Function* has all the qualities of a piece of evidence that could be used in court. (Note: usually the Message-Digest of a message encoded using a private code is known as the "signature" . The term "signature" here has the same meaning.)

Step by step

The procedure for use can be broken into three parts: initialisation, integrity test and maintenance. For initialisation, a reference database *tw.db_hostname* is created in accordance with parameters in the configuration file *tw. config*. This database contains an exact description of all security-relevant objects in the local filesystem. Each object, together with attributes such as the modification timestamp, is assigned its own "fingerprint" in the form of a freely definable signature cocktail. The choice of information to be recorded for each object is made in *tw.config*. This is done by means of assigning so-called *select-masks*, which allows the extent of monitoring to be tailored to the respective importance and task of each object and the individual security requirements of the administrator.

It is of crucial importance at this stage that no viruses or other rogue software components are present in the filesystem. This is tricky, but not impossible to ensure, even for ageing systems. A preliminary manual check of critical files such as *passwd* or *inetd.conf* can assure their future integrity and erect a first barrier against incursion, which can be successively reinforced. The evaluation of the current security situation is ultimately the reserve of the administrator.

Integrity test

In accordance with the configuration laid down in *tw.config* a second databank is calculated. This, like a snapshot, describes the current status of the filesystems which can be referred to at any desired date. By comparing this with the reference database, information about added or deleted objects will come to light.

Listing 1: output example of an altered file

```
changed: -rw-r--r-- root         788 Aug  1
16:49:52 2000 /etc/hosts
### Attr       Observed (what it is)
Expected (what it should be)
### ========== ============================
============================
/etc/hosts
     st_mtime: Tue Aug  1 16:49:52 2000
Tue Aug  1 16:31:21 2000
     st_ctime: Tue Aug  1 16:49:52 2000
Tue Aug  1 16:31:21 2000
   md5 (sig1): 1gLd3EYIaQg04IweV.AMJS
3zbqWhNQTo.9Wytoqhgxik
 snefru (sig2): 0BfGPDfQVve2bm7VbHYD4S
0JCn1vTUQ8apn©eQ:Gc5x
```

Steps for use of Tripwire	
Initialization (single user mode!)	*tripwire --initialize* Creates the reference database. This is placed in the sub-directory *./databases* and must be moved by hand to *DATADIR* because Tripwire expects a read-only medium here.
Integrity test	*tripwire* Prints a report of all inconsistencies detected on the screen. If the output is large, or in case of use as a cron job it is advisable to divert the output to a file. The display of information can be regulated using *--quiet* (single-line outputs) and *--verbose* (isochronous output). All signatures are usually output in Base64 notation. *--print-hex* forces the output as a hexadecimal number, which becomes necessary if signature comparisons between incompatible systems are required.
Maintenance (single user mode!)	*tripwire --update [path]* Updates the reference database if the content of a specified directory has been changed as the result of administrative operations such as installation, uninstallation etc. Individual files or entire directories can be specified as arguments. Since this means the entire content of a directory can be declared as safe, great care is recommended when using this command! (In an emergency the old database, which *Tripwire* archives after each update as *tw.db_hostname.old* in *./databases* , can be restored.)
Interactive updating	*tripwire --interactive* Interactive updating of the reference database is the most common maintenance method. Any inconsistencies which have been detected result in a prompt for updating. This requires the presence of the administrator, of course, but in the end it is a better solution, since each case can be dealt with individually.

Changes to individual objects are detected by comparing the corresponding entries from both. Unlike added or deleted objects, which the integrity report clearly identifies, changes only appear in the report if specified in the corresponding select mask. This allows the administrator to distinguish between changes that are potentially dangerous and legitimate changes which are the result of normal use of the system. Items are logged in the integrity report using the comments *added*, *deleted* and *changed* (Fig. 1).

Listing 1 shows a case where both *modification timestamp* and *inode timestamp* (*st_mtime* and *st_ctime*), as well as both associated signatures (*md5* and *snefru*) differ from the original. This could occur if a text file was loaded into an editor, changed and saved using the same name with the same size.

The integrity test should be performed automatically, at regular intervals of hours or days, using a *cron job. By means of command line options the time-consuming calculation of signatures can be omitted. This makes it possible, in situations such as the deletion of an application, to obtain a rapid overview of any irregularities in the filesystem.*

Maintenance

Changing requirements can make restructuring of the filesystem necessary. In this event the reference database must also be updated. *Tripwire* incorporates the necessary functionality. The addition or removal of even complex packages is a simple exercise.

Installation

The source code of *Tripwire* version 1.2 from 1994, which is free, is still to be found on the website of the COAST Project as a compressed TAR archive. At the end of 1997 Gene Kim's company, Tripwire Security Systems, Inc. received an exclusive licence for advanced development and marketing from Purdue University. Apart from the product line intended for commercial marketing ("HQ Connector Bundle") TSS also offers on its website the so-called "Academic Source Release" (Version 1.3.1) as source code. This differs from the aforementioned COAST version mainly in a few detail improvements. This version, which is adequate for most uses, has been released for use on single-user systems. This is the version to which we will be referring in the following.

Anyone who would prefer to avoid handling the source code may download the free binaries for the prize-winning Linux version 2.2.1. What TSS claims

Table 1: Tripwire platforms

386BSD 0.1
Apple A/UX 3.x
AT\&T System V
BSDI BSD/386 beta
ConvexOS 9.1
DC/OSx (SVR4) 1.1 OSx 5.1
Domain/OS SR10.x AIX 3.x
dynix/PTX 2.0.x
Dynix 3.x
EP/IX 1.4.3
FPX 4.3.3
HP/UX 8.x, 9.x
IRIX 4.x
Linux 0.99.14 und newer
Mach (NeXtstep) 2.x, 3.x OSF/1 1.0.4
Mach 2.x
SunOS 4.0.3
SunOS 5.x (Solaris 2.x)
Ultrix 4.x
Umax V 2.4.1P3
Unicos 6.1.6 OSF/1 (alpha)
Xenix 2.2.4
Xenix System V 386

to be a "functionally compatible" variant of the source code has been freely available since October of last year. This is subject to the GPL and is available for downloading from VA Linux Systems. Any interested C expert is welcome to participate in further development.

With this departure from its previous corporate policy TSS is promising a considerable additional development push. The prestige of taking part in the development of a market-leading, even trailblazing security tool, ought to attract quite a few ambitious developers from all over the world. The binaries for this version will be available from the still dew-fresh website of the Tripwire Open Source Project, which is also to function as a central starting point for all collaborators. Users of other operating systems can only dream of this utopia. Buying the commercial full version represents an investment of around £400 per host.

A few Linux distributors such as SuSE have already taken into account the growing security awareness of their clientele and included versions of Tripwire on their CDs. A glance into the file jungle of your Linux distribution might therefore save you some download time!

Tripwire has been written in portable C. It can run on more than twenty UNIX derivatives. The procedure for installation of the package is standard and should pose no insoluble problems. For rapid installation on a Linux platform a bullet-point type brief introduction is given.

A bit of manual labour

The basic installation requirement is the presence of the well-known packages "Flex" (production of lexical scanner) and "Bison" (parser generator). All system-specific alterations are made in the two following and largely self-explanatory files:

Makefile: Here, the destination DESTDIR (*tw.config* and binaries), DATADIR (*tw.db_hostname*), MANDIR (Manpages) locations should be entered. For DESTDIR and DATADIR, for security reasons, only directories requiring root permission can be considered! (The access rights defined under *install* could be made a bit more restrictive: 400 for *tw.config*, 500 for DESTDIR, 600 for DATADIR.) Also, by uncommenting *LEX = flex* and *YACC = bison -y* both packages mentioned will be used.

include/config.h: Because, in the integrity test, there is a risk that spurious data could be used simply by changing two paths, the destinations of *tw.config* and *tw.db_hostname* are compiled into the executable file. The directories already selected, DESTDIR and DATADIR must therefore also be made known as CONFIG_PATH and DATABASE_PATH prior to conversion at this point. No further entries need be made here. The standard example *configs/conf-svr4.h*, which provides for further alterations required by some operating systems, should be suitable for most Linux distributions.

After changing to single user mode (which prevents any unauthorised accesses in this delicate phase) it should be sufficient to run *make && make test*. This should leave two executable files *tripwire* and *siggen* in the directory *src*, and then start a test of the binaries created. In the test phase, which takes a few minutes, the largely self-explanatory test report (Listing 2) should be watched closely. Compilation errors, which could occur as a result of incompatible libraries or compilers, could critically affect the security performance in later operation. *make test &>~/TestProt* would not be a bad idea!

Unfortunately the value of this test is limited by the fact that there is no adequate way to check the authenticity of the source text. The "original" COAST version shows how it's done: add the Message-Digest of the package, encrypted with the private cipher of a guarantor, as a separate file. The MAC (Message Authentication Code) must be put on at this point, and should for a company such as TSS, which wants to prove high standards in matters of data security, really be more than pure theory! The manipulation of the source code of a security tool by rogues could have unimaginable consequences. If there is doubt as to the authenticity of the code, the only way to lay your worries to rest is a time-consuming one. Sifting through ten thousand lines of source text may not be to everyone's taste.

Using *make install* eliminates the tedious installation of the two binaries, the Manpages, and an admittedly rudimentary model for *tw.config*. All remaining traces of the test run, which Tripwire tends to leave in the */tmp* directory, should then be removed. Any concerns may be mollified with the help of the attached FAQ (see also Points 2.0 and 2.1 in the *README*). NB: *contrib/README.linux* is obsolete and therefore misleading!

Installation should not be difficult even in other environments. *Ported* contains ready-made entries and recommendations concerning the most suitable compiler options for a whole range of known platforms. Additional information can be found in *configs/*, which contains a wide selection of preprepared INCLUDE-files and a few somewhat basic models for *tw.config* .

Rehearsing alert

One further comment on the test report. In the final test phase (TSS-Shellscript *test1.sh*) numerous inconsistencies are shown which give the impression that an undesirable change might have occurred. In fact, this is a true blue integrity test of the distribution against a reference database which is included in the package. This occurs not so much to check the integrity of the distribution, but rather

Listing 2: Self test

```
=== test0.sh: DESCRIPTION
    This shell script exercises all the signature routines included in
the Tripwire distribution.  This suite is run on a series of files
created by the authors of the signature routines.
=== test0.sh: BEGIN ===
=== test0.sh: PASS ===
=== test.twpre.sh: DESCRIPTION
    This script exercises the Tripwire preprocessor, testing correctness
variable expansion and include files.
=== test.twpre.sh: BEGIN ===
Tripwire(tm) ASR (Academic Source Release) 1.3.1
File Integrity Assessment Software
©1992, Purdue Research Foundation, ©1997, 1999 Tripwire
Security Systems, Inc. All Rights Reserved. Use Restricted to
Authorized Licensees.
=== test.twpre.sh: PASS ===
=== test.update.sh: DESCRIPTION
    This shell script exercises all the Tripwire integrity checking
and database update functionalities.
=== test.update.sh: Setting up auxiliary scripts ===
=== test.update.sh: BEGIN ===
../src/tripwire -loosedir -c /tmp/twtest/tw.config -d /tmp/twtest/tw.db -i all
=== test.update.sh: testing GROWING (safe) files ===
=== test.update.sh: testing GROWING (unsafe) files ===
=== test.update.sh: testing ADDED files ===
=== test.update.sh: testing DELETED files ===
=== test.update.sh: testing CHANGED files ===
=== test.update.sh: testing input schemes ===
=== test.update.sh:  tw.config from stdin
=== test.update.sh:  database from stdin
=== test.update.sh: testing complex UPDATE cases
=== test.update.sh:  changed ignore-mask (UPDATE file)
=== test.update.sh:  changed ignore-mask (UPDATE entry)
=== test.update.sh: testing UPDATED files (7 cases)
=== test.update.sh:  case 1: update: add new file ===
=== test.update.sh:  case 2: update: delete file ===
=== test.update.sh:  case 3: update: update file ===
=== test.update.sh:  case 4: nonsense case (skipping) ===
=== test.update.sh:  case 6: update: delete entry ===
=== test.update.sh:  case 5: update: add entry ===
=== test.update.sh:  case 7: update: update entry ===
=== test.update.sh: PASS ===
=== test.inter.sh: DESCRIPTION
    This shell script exercises all the interactive update of Tripwire
databases.
=== test.inter.sh: Setting up auxiliary scripts ===
=== test.inter.sh: BEGIN ===
../src/tripwire -loosedir -c /tmp/twtest/tw.config -d /tmp/twtest/tw.db -i all
=== test.inter.sh: testing interactive update ===
=== test.inter.sh: testing complex UPDATE cases
=== test.inter.sh:  changed ignore-mask (UPDATE file)
=== test.inter.sh:  changed ignore-mask (UPDATE entry)
=== test.inter.sh: testing UPDATED files (7 cases)
=== test.inter.sh:  case 1: update: add new file ===
=== test.inter.sh:  case 2: update: delete file ===
=== test.inter.sh:  case 3: update: update file ===
=== test.inter.sh:  case 4: nonsense case (skipping) ===
=== test.inter.sh:  case 6: update: delete entry ===
=== test.inter.sh:  case 5: update: add entry ===
=== test.inter.sh:  case 7: update: update entry ===
=== test.inter.sh: PASS ===
=== test.escape.sh: DESCRIPTION
    This is similar to the Tripwire update tests, but escaped
filenames are specifically exercised.
=== test.escape.sh: Setting up auxiliary scripts ===
=== test.escape.sh: BEGIN ===
../src/tripwire -loosedir -c /tmp/twtest/tw.config -d /tmp/twtest/tw.db -i all
=== test.escape.sh: testing complex UPDATE cases
=== test.escape.sh:  changed ignore-mask (UPDATE file)
=== test.escape.sh:  changed ignore-mask (UPDATE entry)
=== test.escape.sh: testing UPDATED files (7 cases)
```

```
=== test.escape.sh:  case 1: update: add new file ===
=== test.escape.sh:  case 2: update: delete file ===
=== test.escape.sh:  case 3: update: update file ===
...
=== test.escape.sh: PASS ===
=== test1.sh: DESCRIPTION
   This shell script tests all the Tripwire signature routines.
Consequently, this test may take a while to complete, because even the
slowest signature routines are exercised.  On a 200 MHz Intel Pentium
machine, this test takes 15 seconds to complete.
   This test suite will ascertain whether the byte-ordering and
machine-dependent routines are working correctly.
=== test1.sh: BEGIN ===
creating: ./tw.db_TEST.@
creating: ./@tw.config
Tripwire(tm) ASR (Academic Source Release) 1.3.1
File Integrity Assessment Software
© 1992, Purdue Research Foundation, © 1997, 1999 Tripwire
Security Systems, Inc. All Rights Reserved. Use Restricted to
Authorized Licensees.
### Phase 1:   Reading configuration file
### Phase 2:   Generating file list
### Phase 3:   Creating file information database
### Phase 4:   Searching for inconsistencies
###
###                      Total files scanned:               161
###                           Files added:          0
###                           Files deleted:                 0
###                           Files changed:                 161
###
###                      Total file violations:             161
###
changed: drwxr-xr-x root       1024 Sep 27 23:00:31 2000 /root/tw_ASR_1.3.1_src
changed: -rw-r----- root       2201 May  4 10:31:00 1999 /root/tw_ASR_1.3.1_src/COAST.info
changed: -rw-r----- root       5441 May  4 10:31:00 1999 /root/tw_ASR_1.3.1_src/FAQ
...
### Phase 5:   Generating observed/expected pairs for changed files
###
### Attr       Observed (what it is)        Expected (what it should be)
### =========== ============================ ============================
/root/tw_ASR_1.3.1_src
       st_ino: 128630                       605814
       st_uid: 0                           1016
      st_size: 1024                         0
     st_mtime: Wed Sep 27 23:00:31 2000     Fri Apr 30 23:03:53 1999
     st_ctime: Wed Sep 27 23:00:31 2000     Fri Apr 30 23:03:53 1999
/root/tw_ASR_1.3.1_src/COAST.info
       st_ino: 128632                       605816
     st_ctime: Wed Sep 27 22:51:45 2000     Tue May  4 15:20:44 1999

/root/tw_ASR_1.3.1_src/FAQ
       st_ino: 161308                       605817
     st_ctime: Wed Sep 27 22:51:45 2000     Tue May  4 15:20:44 1999
...
=== test1.sh: END ===
removing: ./tests/tw.db_TEST.@
removing: @tw.config
...
```

to ensure that the signature functions work correctly. Since the reference database supplied by TSS was not encrypted, it would be an easy task for a hacker to change the signatures concerned in the reference databank!

The report quite correctly shows all 161 files of the distribution as "changed" , because *inode number* (*st_ino*) and *inode timestamp* (*st_ctime*) in the filesystem of the destination computer don't match their original values, which were valid at the time of creation of the reference databank by TSS. Changed signatures should, on the other hand, certainly give you something to think about, since only arithmetic errors could have caused them. This can be clarified using *grep -n "md5" ~/TestProt* .

In the next article we will examine the potentially problematic configuration of *Tripwire* . Unlike a virus scanner, which makes few demands in this respect (but is also scarcely any use) an integrity checker demands a lot of effort on the part of the user. The use of this unusual set of tools will be explained later. ■

Info

[1] The "Tripwire-Story": http://www.forbes.com/tool/html/toolbox.htm

[2] Gene Spafford's Homepage: http://www.cerias.purdue.edu/homes/spaf

[3] "RSA Data Security, Inc.": http://www.rsasecurity.com

[4] "Xerox Palo Alto Research Center": http://www.parc.xerox.com/parc-go.html

[5] Snefru and accessories (Xerox): ftp://arisia.xerox.com/pub/hash

[6] "COAST Project": http://www.cerias.purdue.edu/coast

[7] "Tripwire Security Systems Inc.": http://www.tripwiresecurity.com

[8] Future "Tripwire-forge": http://sourceforge.net/projects/tripwire [9] "Tripwire Open Source Project": http://www.tripwire.org

■