Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection*

Purdue Technical Report CSD-TR-93-071

Gene H. Kim and Eugene H. Spafford
COAST *Laboratory*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398

21 February 1994

Abstract

Tripwire is an integrity checking program written for the UNIX environment. It gives system administrators the ability to monitor file systems for added, deleted, and modified files. Intended to aid intrusion detection, Tripwire was officially released on November 2, 1992. It is being actively used at thousands of sites around the world. Published in volume 26 of `comp.sources.unix` on the USENET and archived at numerous FTP sites around the world, Tripwire is widely available and widely distributed. It is recommended by various computer security response teams, including the CERT and CIAC.

This paper begins by motivating the need for an integrity checker by presenting a hypothetical situation any system administrator could face. An overview of Tripwire is then described, emphasizing the salient aspects of Tripwire configuration that supports its use at sites employing modern variants of the UNIX operating system. Experiences with how Tripwire has been used in "in the field" are then presented, along with some conjectures on the prevalence and extent of system breakins. Novel uses of Tripwire and notable configurations of Tripwire are also presented.

*This paper appeared as [8]

1 Introduction

Tripwire is an integrity checking program written for the UNIX environment that gives system administrators the ability to monitor file systems for added, deleted, and modified files. Intended to aid intrusion detection, Tripwire was officially released on November 2, 1992,[1] and is being actively used at thousands of sites around the world. Published in volume 26 of `comp.sources.unix` and archived at numerous FTP sites around the world, Tripwire is widely available and widely distributed. It is now recommended by many computer security response teams, including the ARPA Computer Emergency Response Team (CERT).

Testing of Tripwire started in September 1992. Since then, its design and code have been available for scrutiny by the public at large. The design and implementation are described in detail in [6].

An intensive beta test period resulted in Tripwire being ported to over two dozen variants of UNIX, including several versions neither author had ever encountered. Currently entering its seventh (and possibly last) revision, Tripwire has met our design goals of being sufficiently portable, scalable, config-

[1] That release date was chosen for its historical significance as well as being convenient to our development schedule.

urable, flexible, extensible, secure, manageable, and malleable to enjoy widespread use.

This paper documents some of our experiences and discoveries based on our development and use of Tripwire. It begins by motivating the use of an integrity checking tool (such as Tripwire) through the presentation of a hypothetical scenerio that a UNIX system administrator could face. Next, we present an overview of Tripwire's design, emphasizing the salient configuration aspects that allow its use in modern UNIX variants. We then discuss experiences gathered from Tripwire users since its September 1992 release. These stories seem to confirm the practicality of this integrity checking scheme. There are at least seven documented cases of Tripwire notifying system administrators of intruders' system tampering. We present our conjectures on the prevalence and extent of system breakins based on our data. We also describe novel uses of Tripwire and surprising configurations that have been reported to us. Feedback that has shaped the direction of Tripwire development is also presented.

Tripwire stands as an example how a simple idea can be developed into a general and effective tool to enhance UNIX security while also posing almost no threat to the systems under guard. Unlike programs like password crackers or flaw probes, Tripwire cannot be turned against a system to identify or exploit weaknesses or flaws. It is also an example of how a program may have uses unanticipated by its authors.

2 Motivation

2.1 A cautionary tale[2]

Ellen runs a network of 50 networked UNIX computers representing nearly a dozen vendors — from PCs running Xenix to a Cray running Unicos. This morning, when she logged in to her workstation, Ellen was a bit surprised when the `lastlog` message indicated that root had logged into the system at 3 AM. Ellen thought she was the only one with the root password. Needless to say, this was not something Ellen was happy to see.

A bit more investigation revealed that someone — certainly not Ellen — had logged on as root, not only on her machine but also on several other machines in her company. Unfortunately, the intruder deleted all the accounting and audit files just before logging out of each machine. Ellen suspects that the intruder (or intruders) ran the compiler and editor on several of the machines. Being concerned about security, Ellen is worried that the intruder may have thus changed one or more system files, thus enabling future unauthorized access as well as compromising sensitive information. How can she tell which files have been altered without restoring each system from backups?

Poor Ellen is faced with one of the most tedious and frustrating jobs a system administrator can have — determining which, if any, files and programs have been altered without authorization. File modifications may occur in a number of ways: an intruder, an authorized user violating local policy or controls, or even the rare piece of malicious code altering system executables as others are run. It might even be the case that some system hardware or software is silently corrupting vital system data.

In each of these situations, the problem is not so much knowing that things might have been changed; rather, the problem is verifying exactly which files — out of tens of thousands of files in dozens of gigabytes of disk on dozens of different architectures — might have been changed. Not only is it necessary to examine every one of these files, but it is also necessary to examine directory information as well. Ellen will need to check for deleted or added files, too. With so many different systems and files, how is Ellen going to manage the situation?

Resolving such a situation would prove tedious and labor-intensive for even the most well-prepared system administrator. Consider the problems facing system administrators who use simple checklisting schemes:

[2] This is taken from [7].

2.2 The resulting challenges

Established techniques for monitoring file systems for potentially dangerous changes include maintaining checklists, comparison copies, checksum records, or a long history of backup tapes for this kind of contingency [4, 2]. However, these methods are costly to maintain, prone to error, and susceptible to easy spoofing by a malicious intruder.

For instance, the UNIX utility `find(1)` is often used to generate a checklist of system files, perhaps in conjunction with `ls(1)`. This list is then saved and compared using `diff(1)` to determine which files have been added or deleted, and to find which files have conflicting modification times, ownership, or sizes. An added level of security could be added by augmenting these lists with information from `sum(8)` or `cksum(8)`, as is done by the *crc_check* program included with COPS [3].

However, numerous shortcomings in these simple checklisting schemes prevent them from being completely trustworthy and useful. First, the list of files and associated checksums may be tedious to maintain because of its size. Second, using timestamps, checksums, and file sizes does not necessarily ensure the integrity of each file (e.g., once intruders gain root privileges, they may alter timestamps and even the checklists at will). Furthermore, changes to a file may be made without changing its length or checksum generated by the `sum(8)` program. And this entire approach presumes that `ls(1)`, `sum(8)`, and the other programs have not been compromised! In the case of a serious attack, a conscientious administrator needs stronger proof that important files have remained unchanged. But what proof can be offered that is sufficient for this situation?

2.3 The resulting wishlist

A successful integrity checking scheme requires a high level of automation — both in generating the output list and in generating the input list of files. If the scheme is difficult to use, it may not be used often enough — or worse, used improperly. The automation should include a simple way to describe portions of the filesystem to be traversed. Additionally, in cases where files are likely to be added, changed, or deleted, it must be easy to update the checklist database. For instance, files such as `/etc/motd` may change daily or weekly. It should not be necessary to regenerate the entire database every time this single file changes to maintain database accuracy.

Ideally, our integrity checking program could be run regularly from `cron(8)` to enable detection of file changes in a timely manner. It should also be possible to run the program manually to check a smaller set of files for changes. As the administrator is likely to compare the differences between the "base" checklist and the current file list frequently, it is important that the program be easy to invoke and use.

A useful integrity checker must generate output that is easy to scan. A checker generating three hundred lines of output from each machine for the system administrator to analyze daily would be self-defeating — this is far too much to ask of even the most amazingly dedicated system administrator! Thus, the program must allow the specification of filesystem "exceptions" that can change without being reported, and hence reduce "noise." For example, changes in system log file sizes are expected, but a change in inode number, ownership, or file modes is cause for alarm. However, a change in any value stored in the inodes (except for the access timestamp) for system binaries in `/bin` should be reported. Properly specified, the integrity checker should operate unobtrusively, notifying Ellen when a file changes outside the specified bounds, and otherwise running quietly.

Finally, assuming that Ellen wants to run the integrity checker on every machine in her network, the integrity checker should allow the reuse and sharing of configuration files wherever possible. For example, if Ellen has twenty identical workstations, they should be able to share a common configuration file, but allowing machine-specific oddities (i.e., some software package installed on only one machine). The configuration should thus support selective reuse to reduces the opportunity for operator error.

3 Tripwire design

The criteria we describe above represent the motivation for some of the key design issues behind Tripwire. Ultimately, the goal of Tripwire is to detect and notify system administrators of changed, added, and deleted files in some meaningful and useful manner. However, the success of such a tool depends on how well it works within the realities of the administration environment. This includes appropriate flexibility to fit a range of security policies, portability to different platforms in the same administrative realm, and ease of use.

3.1 Component overview

A high level model of Tripwire operation is shown in Figure 1. This shows how the Tripwire program uses two inputs: a *configuration* describing the file system objects to monitor, and a *database* of previously generated signatures putatively matching the configuration.

In its simplest form, the configuration file contains a list of files and directories to be monitored, along with their associated selection mask (i.e., the list of attributes that can be safely ignored if changed). The database file is generated by Tripwire, containing a list of entries with filenames, inode attribute values, signature information, selection masks, and the configuration file entry that generated it.

3.2 Modes of Operation

In the four (mutually exclusive) modes of Tripwire operation, program operation is driven by the contents of the configuration file, `tw.config`. Each mode is described in turn below.

In *database initialization* mode, Tripwire generates a baseline database containing entries for every file specified in the configuration file, `tw.config`. Each database entry contains the filename, inode attributes, signature information, its selection mask, and the configuration entry that generated it. As entries in `tw.config` can be directories, each entry can map to many entries in the database.

In *integrity checking* mode, Tripwire reads the `tw.config` file and generates a new database. Tripwire then compares this database with the baseline, producing a list of added and deleted files. For those files that have changed, the selection mask is applied to determine whether a report should be generated. Note that the selection mask stored in the baseline database is used, not the one in `tw.config`, based on the premise that the base database has been stored on some secure media (e.g., read-only floppy).

When files change for legitimate reasons and no longer match the baseline database description, updating the baseline database becomes necessary. Tripwire offers two modes to ensure database consistency. In *database update* mode, Tripwire is given a list of files or configuration entries on the command line. The database entries for these files are regenerated, and a new database written out. Tripwire then instructs the system administrator to move this database to secure media. In *interactive database update* mode, Tripwire first generates a list of all changes (ala integrity checking mode). For each of these changes, Tripwire then asks the system administrator whether the specified file or entry should be updated.

3.3 Scalability aids

Tripwire includes an M4-like preprocessing language [5] to help system administrators maximize reuse of configuration files. By including directives such as "`@@include`", "`@@ifdef`", "`@@ifhost`", and "`@@define`", system administrators can write a core configuration file describing portions of the file system shared by many machines. These core files can then be conditionally included in the configuration file for each machine.

To allow the possible use of Tripwire at sites consisting of thousands of machines, configuration and database files do not need to reside on the actual machine. Input can be read from file descriptors, open at

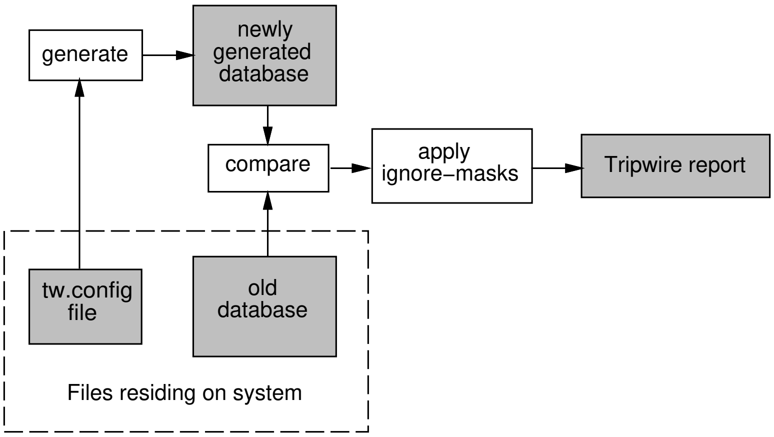

Figure 1: Diagram of high level operation model of Tripwire

the time of Tripwire invocation. These file descriptors can be connected to UNIX pipes or network connections. Thus, a remote server or a local program can supply the necessary file contents. Supporting UNIX style pipes also allows for outside programs to supply encryption and compression services — services that we do not anticipate including as a standard part of the core Tripwire package.

Tripwire does not encrypt the database file so as to ensure that runs can be completely automated (i.e., no one has to type in the encryption key every night at 3 AM). Because the database contains nothing that would aid an intruder in subverting Tripwire, this does not undermine the security of the system. However, if Tripwire is used in an environment where the database is encrypted as a matter of policy, the interface supports this, as described above.

3.4 Configurability aids

Tripwire makes a distinction between the configuration file and the database file. Each machine may share a configuration file, but each generates its own database file. Thus, identically configured machines can share their configuration database, but each has its integrity checked against a per-machine database.

Because of the preprocessor support, system administrators can write Tripwire configuration files that support numerous configurations of machines. Uniform and unique machines are similarly handled. This helps support reuse and minimize user overhead in installation.

The configuration file for Tripwire, `tw.config`, contains a list of entries, enumerating the set of directory (or files) to be monitored for changes, additions, or deletions. Associated with each entry is a selection-mask (described in the next section) that describes which file (inode) attributes can change without being reported as an exception. An excerpt from a set of `tw.config` entries is shown in Figure 2.

Prefixes to the `tw.config` entries allow for pruning (i.e., preventing Tripwire from recursing into the specified directory or recording a database entry for a file). Both inclusive and non-inclusive pruning are supported; that is, a directory's contents only may be excluded from monitoring, or the directory and its contents may both be excluded.

By default, all entries within a named directory are included when the database is generated. Each entry is recorded in the database with the same flags and signatures as the enclosing, specified directory. This allows the user to write more compact and inclusive configuration files. Some users have reported using

```
# file/dir      selection-mask
/etc            R       # all files under /etc
@@ifhost solaria.cs.purdue.edu
 !/etc/lp               #  except for SVR4 printer logs
@@endif
/etc/passwd     R+12    # you can't be too careful
/etc/mtab       L       # dynamic files
/etc/motd       L
/etc/utmp       L
=/var/tmp       R       # only the directory, not its contents
```

Figure 2: An excerpt from a `tw.config` file

configuration files of a simple `/`, naming all entries in the file system!

The `tw.config` file also contains the names of files and directories with their associated selection-mask. A selection-mask may look like: `+pinugsm12-a`. Flags are added ("+") or deleted ("-") from the set of items to be examined.

Tripwire reads this as, "Report changes in permission and modes, inode number, number of links, user id, group id, size of the file, modification timestamp, and signatures 1 and 2. Disregard changes to access timestamp."

A flag exists for every distinct field stored in an inode. Provided is a set of templates to allow system administrators to quickly classify files into categories that use common sets of flags:

- **read-only files** Only the access timestamp is ignored.
- **log files** Changes to the file size, access and modification timestamp, and signatures are ignored.
- **growing log files** Changes to the access and modification timestamp, and signatures are ignored. Increasing file sizes are ignored.
- **ignore nothing** self-explanatory
- **ignore everything** self-explanatory

Any files differing from their database entries are then interpreted according to their selection-masks. If any attributes are to be monitored, the filename is printed, as are the expected and actual values of the inode attributes. An example of Tripwire output for changed files is shown in Figure 3.

A "quiet option" is also available through a command-line option to force Tripwire to give terse output. The output when running in this mode is suitable for use by filter programs. This allows for an external script to execute automated actions if desired. One example would be to use the terse output of Tripwire after a breakin to quickly make a backup tape of only changed files, to be examined later.

By allowing reporting to be dictated by local policy, Tripwire can be used at sites with a very broad range of security policies.

3.5 Signature support

Tripwire has a generic interface to "signature[3]" routines and supports up to ten signatures to be used for each file. The following default methods are included in the latest Tripwire distribution: MD5[11] (the RSA Data Security, Inc. MD5 Message-Digest Algorithm), MD4[10] (the RSA Data Security, Inc. MD4 Message-Digest Algorithm), MD2 (the RSA Data

[3] We use the term *signature* to include checksums, message digests, secure hash functions, and/or cryptographic signatures.

```
changed: -rw-r--r-- root          20 Sep 17 13:46:43 1993 /.rhosts
### Attr        Observed (what it is)        Expected (what it should be)
### =========== ============================ =============================
/.rhosts
st_mtime:       Fri Sep 17 13:46:43 1993      Tue Sep 14 20:05:10 1993
st_ctime:       Fri Sep 17 13:46:43 1993      Tue Sep 14 20:05:10 1993
```

Figure 3: Sample Tripwire output for a changed file

Security, Inc. MD2 Message-Digest Algorithm),[4] Snefru[9] (the Xerox Secure Hash Function), and SHA (the NIST proposed Secure Hash Algorithm). Tripwire also includes POSIX 1003.2 compliant CRC-32 and CCITT compliant CRC-16 signatures.

Each signature may be included in the selection-mask by including its index. Because each signature routine presents a different balance in the equation between performance and security, the system administrator can tailor the use of signatures according to local policy. By default, MD5 and Snefru signatures are recorded and checked for each file. However, different signatures can be specified for each and every file. This allows the system administrator great flexibility in what to scan, and when.

Also included in the Tripwire distribution is `siggen`, a program that generates signatures for the files specified on the command line. This tool provides a convenient means of generating any of the included signatures for any file.

The code for the signature generation functions is written with a very simple interface. Thus, Tripwire can be customized to use additional signature routines, including cryptographic checksum methods and per-site hash-code methods. Tripwire has room for 10 functions, and only seven are preassigned, as above.

[4]The copyright on the available code for MD-2 strictly limits its use to privacy-enhanced mail functions. RSA Data Security, Inc. has kindly given us permission to include MD-2 in the Tripwire package without further restriction or royalty.

4 Experiences

Since the initial Tripwire release in November 1992, seven subsequent versions have been released to incorporate bug fixes, support additional platforms, and add new features. The authors estimate Tripwire is being actively used at several thousand sites around the world. Retrievals of the Tripwire distribution from our FTP server initially exceeded 300 per week. Currently, seven months after the last official patch release, we see an average of 25 fetches per week. This does not include the copies being obtained from the many FTP mirror sites around the net.

We have received considerable feedback on Tripwire design, implementation, and use. We believe that version 1.1 of Tripwire has succeeded in meeting most of the goals of system administrators needing an integrity checking tool.

In this section, we present feedback from system administrators that seem to validate the workability of integrity checkers and present conjectures on the prevalence and extent of system breakins. We also prevent novel uses of Tripwire and surprising configurations that have been reported to us. Feedback that has shaped the direction of Tripwire development is also presented.

4.1 First, the good news. . .

We have gathered reports of at least seven cases where Tripwire has alerted system administrators to intruders tampering with their systems. In at least two of these cases, the penetration was widespread, with numerous system programs and libraries replaced with trojan horses.

Potentially less exciting than these stories, but equally inspiring, are the dozens of stories we have received of sites using Tripwire as a system administration enforcement tool. System administrators report having found hundreds of program binaries changed, only to find that another person with system-level access had made the changes without following local notification policy.

There has also been one reported case of a system administrator detecting a failing disk with Tripwire. The normal system logging reporting the failure was not read very often by the system administrator, but the Tripwire output was surveyed daily.

All three classes of stories validate the theory behind integrity checking programs. Although the foundations of integrity checkers in UNIX security have been discussed in [1, 2, 4], when Tripwire design was started in May 1992, no usable, publically available integrity tools existed — providing one of the primary motivations for writing Tripwire.

4.1.1 Where are all the bad guys?

The dramatically increased number of network breakins throughout the Internet in early 1994 presented an opportunity to compare the prevalence of system breakins at sites running Tripwire with those sites that did not.[5]

One of us (Spafford) posted a query on USENET asking whether any sites running Tripwire were successfully subverted as described in the CERT advisory. Surprisingly, no system administrator at any site reported such a breakin. Why? Furthermore, out of the thousands of machines running Tripwire, why have we heard of only seven Tripwire-discovered breakins since 1992? We offer some possibilities:

- The intruders have given up: if the competence and ambition of intruders have dropped since 1992, the small number of reported incidents could be explained away. However, this is not consistent with the reports from response teams and the frequent advisories reporting newly discovered system vulnerabilites being exploited by intruders.
- The sites running Tripwire are not interesting: if sites running Tripwire offer nothing of interest to an intruder, then one would expect few breakin attempts. However, given that some of the highest-profile UNIX sites in the nation (e.g., public access UNIX sites, government information servers, military sites) are running Tripwire, this seems implausible.
- The site admins are not telling: it may be the case that system administrators at sites where breakins have occurred are not willing to tell us that they have been sucessfully attacked. This is possible considering the nature of many of the sites running Tripwire, but we would expect at least a few reports to be made in confidence.
- The sites are more security-conscious: if system administrators running Tripwire are also considerably more successful in securing their systems than other UNIX sites, intruders would find known vulnerabilities corrected, greater than usual protection measures employed, and more vigilance from system administrators. This would explain the low number of reported intrusion incidents from a sample made up exclusively of sites running Tripwire.
- The sites have already been attacked: the Tripwire baseline database should be generated from a clean distribution; one that is assured to be free of trojan horses, logic bombs, etc. This usually means reinstalling the operating system from vendor-supplied media. However, this is often prohibitively inconvenient. If databases are being generated on machines already compromised, then Tripwire will have been installed too late to have reported those critical file tamperings. If this is the case, then many sites have

[5] See, in particular, CERT advisory CA-94:01 dated February 3, 1994.

not reported breakins because they continue to be unaware of them.

- The intruders have completely subverted integrity checking schemes: changed files are usually detected through the use of file signatures. An intruder could be modifying files in such a way that the resulting files preserve their original signatures. However, Tripwire includes seven signature routines, and the choice of which signatures are used for any file is not fixed. That an intruder could be using such a technique is possible, but the possibility is so small as to be almost nonexistant.

Of the conjectures offered, the supposition most credible is that system administrators who run Tripwire represent a poor sample for determining system breakins. A substantial portion of the thousands of Tripwire e-mail messages we have received underscore the competence and paranoia of these system administrators.

However, despite these system administrators' best intentions, their lack of available, trusted signatures for critical operating system files is especially noteworthy. Instead of installing a system from distribution media, they typically choose a machine that they believe is "clean," using it to generate the baseline database. This assumption, however, may be completely ungrounded. We know of at least one case of a reported breakin at a site where system administrators discovered that their "safe baseline" was actually the first to be attacked.

We believe that operating system distributions, and perhaps other software (e.g., compilers, system administration tools), should be shipped with a complete signature database. This information could then be stored locally on some secure media or offline, and then used in the event of a suspected breakin.

Not coincidentally, Tripwire could be used effectively in such a role if software vendors supplied a Tripwire-conformant database with their distributions. The Tripwire database is suited for such a purpose: it is ASCII, mostly human readable, simple to parse and construct (21 fixed fields), compact (signatures are stored in base 64), self-contained (all the database information is encapsulated in a single file), and individual entries can be checked using the `siggen` program.

4.1.2 How about the good guys?

> *"The mark of a good tool is that it is used in ways that its author never thought of."* [6]

As noted previously, many system administrators are using Tripwire primarily as a tool to enforce local policy. When system administration duties are delegated among numerous people, changes made by one person often go unnoticed and unexplained to others. Running Tripwire allows these changes to be noticed in a timely manner — a goal very similar to intrusion detection.

Another application we note uses Tripwire to help salvage file systems not completely repaired by `fsck`, the program run at system boot that ensures consistency between file data and their inodes. In cases when file blocks cannot be bound to its file name, they are placed in the `lost+found` directory and renamed to some (less than useful) number. If a database of file signatures is available, this file could be rebound to its original name by searching the database for a matching signature.

Because providing a useful tool to system administrators was one of the goals of writing Tripwire, the variety of applications of Tripwire outside the domain of intrusion detection has been especially surprising and satisfying for us. We are still collecting other stories of novel use of the Tripwire package.

4.2 Stealth-Tripwire

Several site administrators have reported going to considerable lengths to conceal the operation of Trip-

[6] Brian Kernighan has said this, in one form or another, in several of his presentations and written works. This particular version was in private e-mail to one of us in response to a citation request.

wire. These system administrators feel strongly that they should not advertise their security measures or policies.

As a result, Tripwire is not being run through programs like `cron(8)`, the conventional means of executing programs on a regular schedule. Instead, a wide variety of local tools are used. For example, a special daemon might be loaded at system startup, waking only to run Tripwire at a scheduled time.

Where `cron` is used, deception through indirection is sometimes used to prevent an intruder from immediately detecting evidence of Tripwire operation. In one case, a system administrator uses three levels of indirection before finally executing Tripwire (e.g., `cron` runs a script that runs a script than runs a script that runs Tripwire).

We wonder whether these measures to conceal Tripwire are necessary, or even desirable. One of us (Spafford) has heard of an "underground" publication warning of the need for special vigilance when attempting to crack systems running Tripwire. If this warning is heeded, then the presence of Tripwire may have the ability to deter crackers. Advertising the use of Tripwire (even if not true) could thus help avert attacks.

4.3 Security is nice, if it's not too difficult

Because Tripwire reports are only as reliable as its inputs, the design document stresses the need to ensure the integrity of the baseline database. Thus, we suggest that the baseline database be moved to some secure read-only media immediately after it is is generated.

The most common Tripwire configuration reported to us to facilitate this is the use of a "secure server," a specialized server receiving extra scrutiny from administrators. A remote file system or server process is then used to export the baseline database to clients.

However, several sites have gone to much further lengths to maintain the integrity of Tripwire databases. At least two sites have considerably modified Tripwire to support alternate channels for receiving the database and transmitting the report, adding layers for networking support, encryption, and host authentication.

Since its original release, we have added full support for using open UNIX file descriptors to read the Tripwire configuration and database files. This allows system administrators to easily add support for encryption and compression without having to modify the Tripwire package so drastically. Instead, a wrapper program (even a shell script) can be used to supply these facilities. Used with named pipes, wrapper scripts in Perl or Tcl, or simple network clients this also allows centralized administration of Tripwire checks in large installations.

It is interesting to note that mistrust of networked file systems has motivated many of the enduser-modifications to Tripwire. However, some of the replacements we have have been told about sound as if they include other weaknesses. A sound, portable solution to the problem has yet to appear.

4.4 Paranoia is unbounded

The Tripwire design document recommends running Tripwire in integrity checking mode on a regular basis (e.g., daily) to ensure that file system tampering can be detected in a timely manner.

However, there have been two reported cases of large sites running Tripwire far more frequently. In fact, these experiences motivated the option of including a signature selection feature to allow skipping certain signatures by specifying choices on the command line. Because these site admins were running Tripwire on their machines *hourly* with all signature checking enabled, the Tripwire runs were not completed by the time the next Tripwire run started!

We were left wondering what these machines did besides spending all the CPU cycles computing file signatures. We also wonder why they placed so little faith in their other security measures, and what level of threat they were actually fearing.

In contrast is the lack of use of an ideal Tripwire-aided bit of paranoia. One of the ideas behind Tripwire's design (and the name itself) was for system managers to scatter "plant" files on their system, similar to what was done by Cliff Stoll[12]. These files would have interesting names (e.g., `master-passwords`), but useless contents. These files would not normally be accessed by users, but might be prime targets for intruders. By monitoring these files as "mini-tripwires," it would be possible to detect an otherwise stealthy intrusion. We have yet to hear of anyone using this scheme to good effect.

4.5 Scalability includes sites large and small

When designing Tripwire, we were more concerned about the problems facing system administrators at large sites. Although design considerations were made for these configurations, how Tripwire was used at small sites was more surprising.

4.5.1 Mega-Tripwire

Tripwire provides a configuration language intended to aid system administrators in managing larger sites. We were especially interested in how these tools would be used by system administrators – the Tripwire design document suggests that a core configuration file could be shared by numerous hosts by using the `@@include` directive.

From reports we have gathered, this appears to be a less than popular method. Instead, system administrators create one configuration file to be shared by all machines, using the `@@ifhost` directive to segregate non-common file groups.

We suspect that the overhead of tracking multiple configuration files outweighs the inconvenience caused by files obfuscated by many "`@@ifdef`" statements. These shared configuration files are apparently still manageable, as the number of entries in the file is usually not large. (We suspect that if files had to be individually enumerated, these configuration files would be far larger, and therefore unmanageable.)

Tripwire has proven scalable, with documented cases of sites of almost one thousand machines running Tripwire, as well as sites of only one machine. That system administrators have done so using a different mechanism than suggested in the design document is especially interesting. That system administrators are not slavishly following our design document is reassuring.

4.5.2 Micro-Tripwire

How Tripwire is used on workstations with minimal disk resources proved surprisingly elegant. Although the Tripwire configuration file allows considerable flexibility in specifying files and directories to monitor, configuration files concocted by system administrators for these workstations consist of only one character: "`/`"

Thusly, Tripwire scans all the local disk partitions under the root directory, collecting the default MD5 and Snefru signatures. For some sites, this has proved adequate for all their machines!

4.6 Running Tripwire on the Sasquatch Kumquat Mark VIIa/MP

Tripwire has proven to be highly portable, successfully running on over 28 UNIX platforms. Among them are Sun, SGI, HP, Sequents, Pyramids, Crays, Apollos, NeXTs, BSDI, Lynix, Apple Macintosh, and even Xenix. Configurations for new operating systems has proven to be sufficiently general to necessitate the inclusion of only eight example `tw.config` files.

However, potentially challenging situations result when we receive requests from system administrators asking for help compiling Tripwire on machines that neither of us have ever heard of. In one case, this was a machine only sold in Australia and shipped with incorrect system libraries. Other instances included an especially ignoble machine that has not been sold since 1986 (predating college for of us), and numerous machines with non-standard compilers, libraries, system calls, and shells.

In all but two cases (of the last variety), we have incorporated changes in Tripwire sources to accomodate these machines. In most cases, there has been a sufficiently large group of system administrators with similarly orphaned machines who put together a suitable patch to allow correct Tripwire compilation and operation.

It is interesting to analyze the time needed to fully support a configuration. Full support for Sun's new Solaris operating system was added two months after the initial Tripwire release. A workaround for the two afforementioned Australian machines was released six months after the problems were first reported. However, some Tripwire users running machines from a large workstation vendor continue to be unable to find a compiler that correctly generates a Tripwire that passes the entire test suite; investigation has determined that this is because of non-standard and broken compilers and libraries on those platforms.

4.7 You added WHAT to Tripwire?

We recently received a report from a user who is adding support for Intel machines running UNIX to allow Tripwire to check mounted MSDOS file systems. In such a manner, they are using Tripwire to check not only UNIX file systems, but also their DOS files (for viruses, etc.).

We also received, and are incorporating into a future Tripwire patch release, a set of changes to allow Tripwire to check the integrity of symbolic links — a weakness noted in [13]. One novel and elegant solution was implemented by storing the contents of the symbolic link as a signature.[7] Our actual solution will involve taking the signatures of the link field contents.

We are especially pleased that system administrators can so easily make feature additions that they perceive as necessary. We believe this reflects well on the design and coding of the Tripwire release, although we realize that the code is rather opaque in many spots.

[7] This solution was proposed and implemented by Paul Szabo of the University of Sydney.

4.8 Static file systems aren't

According to system administrators, the ability to update Tripwire databases is among its most important features. Files seem to change for many unforeseen reasons. Consequently, the database is updated regularly. The addition of the interactive update facility in Tripwire was among the most enthusiastically received features.

Allowing database updates was a feature that we resisted for several months during the beta test period in 1992. We believed (and still do, in part) that ease of update may lead some administrators to be careless in their storage of the database, thus weakening the assurance Tripwire is capable of providing. That users acquiesced and still used Tripwire despite its lack of ability to update the baseline database without regenerating the entire database astounds the authors — in hindsight, at least.

5 Conclusions

Tripwire has proven to be a highly portable tool that system administrators can build using commonly available tools. It is completely self-contained, and once built, requires no other tools for execution. Tripwire is publically available, is widely distributed, and widely used.

Tripwire has been used by system administrators in large and small sites: we have documented Tripwire's active use at single machine sites, as well as sites having several hundreds of machines. We have yet to hear a report of a site where Tripwire was installed and then removed because it did not function according to expectation, or because it was too difficult to build or maintain. Coupled with the many positive comments we have received, and the fact that Tripwire has already caught several intruders, leads us to conclude that our analysis and design are successful. We hope this effort serves as a model for others who consider building security tools with similar goals.

6 Availability

The beta version of Tripwire was made publically available and posted to `comp.sources.unix` on November 2, 1992 after three months of extensive testing. Over three hundred users around the world critiqued the four preliminary releases during Summer 1992, guiding the development towards a shippable, publically available tool. The formal release of Tripwire occurred in December of 1993.

Tripwire source is available at no cost.[8] It has appeared in `comp.sources.unix` (volume 26) on Usenet, and is available via anonymous FTP from many sites, including `ftp.cs.purdue.edu` in `pub/spaf/COAST/Tripwire`. Those without Internet access can obtain information on obtaining sources and patches via e-mail by mailing to `tripwire-request@cs.purdue.edu` with the single word "help" in the message body.

We regret that we do not have the resources available to make tapes or diskette versions of Tripwire for anyone other than COAST Project sponsors. Therefore, we ask that you not send us media for copies – it will not be returned.

References

[1] Vesselin Bontchev. Possible virus attacks against integrity programs and how to prevent them. Technical report, Virus Test Center, University of Hamburg, 1993.

[2] David A. Curry. *UNIX System Security: A Guide for Users and System Administrators*. Addison-Wesley, Reading, MA, 1992.

[8] It is not "free" software, however. Tripwire and some of the signature routines bear copyright notices allowing free use for non-commercial purposes.

[3] Daniel Farmer and Eugene H. Spafford. The COPS security checker system. In *Proceedings of the Summer Conference*, pages 165–190, Berkely, CA, 1990. Usenix Association.

[4] Simson Garfinkel and Gene Spafford. *Practical Unix Security*. O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The M4 Macro Processor*. AT&T Bell Laboratories, 1977.

[6] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: A file system integrity checker. Technical Report CSD–TR–93–071, Purdue University, nov 1993.

[7] Gene H. Kim and Eugene H. Spafford. Monitoring file system integrity on unix platforms. *InfoSecurity News*, 4(4):21–22, July 1993.

[8] Gene H. Kim and Eugene H. Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. In *Systems Administration, Networking and Security Conference III*. Usenix, 1994.

[9] Ralph C. Merkle. A fast software one-way hash function. *Journal of Cryptology*, 3(1):43–58, 1990.

[10] R. L. Rivest. The md4 message digest algorithm. *Advances in Cryptology — Crypto '90*, pages 303–311, 1991.

[11] R. L. Rivest. RFC 1321: The md5 message-digest algorithm. Technical report, Internet Activities Board, April 1992.

[12] Cliff Stoll. *The Cuckoo's Egg*. Doubleday, NY, NY, October 1989.

[13] David Vincenzetti and Massimo Cotrozzi. ATP anti tampering program. In Edward DeHart, editor, *Proceedings of the Security IV Conference*, pages 79–90, Berkeley, CA, 1993. USENIX Association.