Implementation of a File Integrity Check System

Don Mosley, East Carolina University, Department of Technology Systems

***Abstract*—The area of real-time Intrusion Detection and Prevention utilizing intelligent routers or various network attached appliances has received much press in the last few years. Should any of these defenses provide less than 100% effective coverage the user will be left unaware of any 'mischief' that might have gotten through. There is still a need for non real-time scanning of system files to determine any unauthorized modifications. This type of audit is often the only effective way to spot malicious activity originating from inside the enterprise network. This paper will outline mechanisms and practices for effective file integrity checking.**

I. Introduction

The current state of Intrusion Detection Systems(IDS) would have to be considered fairly mature. The market for IDS and Intrusion Prevention Systems (IPS) is a large percentage of the $14 billion security software[1] industry with dozens of vendors and service providers worldwide. The functionality provided by these systems can be broken down into three broad categories:

Perimeter control – monitoring and restricting access from the outside world to internal networks; detecting and reacting to anomalous traffic patterns.

Host access control – setting authentication and authorization parameters for users of systems and services.

Change control mechanisms - requiring that any modifications and updates to production systems be reviewed and approved and that implementation and recovery plans are in place.

II. Problems with IDS practices

A. Bypassing perimeter controls

The proliferation of wireless networking has increased access control problems by an order of magnitude. Malicious software can be introduced to the internal network by wireless guest access, returning company laptops infected during travels, or from surreptitious access to the wired LAN. Network Access Control systems can be put in place but these can be cumbersome to configure and intrusive to daily functions.

B. Host configuration mishaps

The typical Systems Administrator has seen a marked increase in the total number of systems under their responsibility. The potential for administrative accidents and misconfigurations is increasing. These can easily have the effect of decreasing the overall system security. Gartner estimates that "80 percent of unplanned downtime is caused by people and process issues, including poor change management practices, while the remainder is caused by technology failures and disasters."[2]

C. Malicious insiders

Institutions have faced this problem since the invention of commerce. An insider who knows the intimate workings of a security system certainly knows the weaknesses. They are already on the inside of most security layers. If you have the key, no lock is safe.

III. Mitigation options

If you've ever been on a ship or a large boat you might have seen a vital piece of equipment – the bilge pump. No ship is 100% watertight, there will always be some leakage. The bilge pump was designed to cope with the inevitable. You've no doubt noticed the electronic inventory systems of modern retail stores. Goods are tracked from receiving dock to cash register. You might have also noticed that retailers still conduct periodic physical inventories. There is always some degree of "leakage" - shoplifting, spoilage, employee theft. This must be quantified and accounted for. In some computing environments, there are regulatory requirements that mandate tracking all file modifications. "The FDA wants not only a life history of the server, but every file on the server."[3]

The same concepts apply to IDS and IPS. No system can guarantee to be 100% effective. There should be some mechanism in place to audit the effectiveness of these systems. There must be some method to detect or prevent unintended file modifications. There are several approaches to this:

Read Only Filesystem – the operating system, applications, and static data are placed on non-writable media, such as a CD-ROM. The server has no hard disk, floppy disk, or USB access. It gets booted off the CD-ROM with all filesystems mounted as read-only. This is not a perfect solution. A clever hacker who acquired administrative access could create a RAM disk, copy the CD contents, modify, and remount in place of the CD content.

Reload from image – this system contains writable media but the disk is periodically wiped clean and reloaded from a secure image of the filesystems. Any unwanted changes are thus replaced. This procedure can be labor and time intensive, has noticeable downtime, and still has the periods between reloads where unauthorized changes could be in play.

File Integrity Checker - this process will take a digital fingerprint of a file and compare it to previous fingerprints.

Don Mosley is a graduate student at East Carolina University in the Department of Technology Systems, concentrating on Information Security. He can be reached at dem0328@ecu.edu

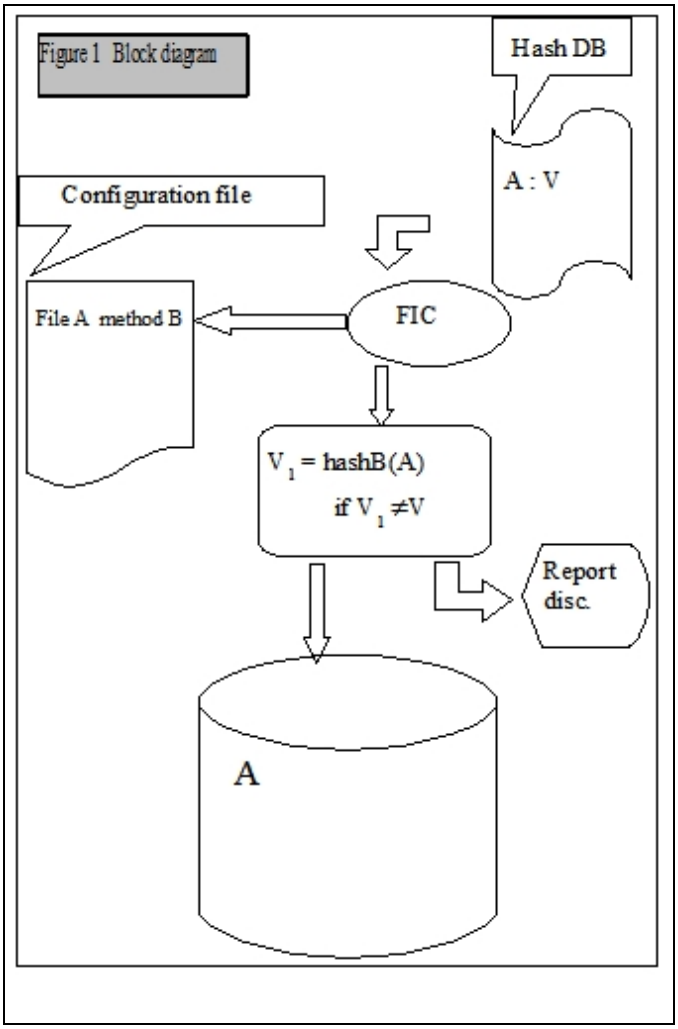
Any differences must be reconciled with change control requests that list files changed. Any discrepancies must be investigated. The drawbacks to this method are: not real time; initial setup can be cumbersome; only protects static files.

File Integrity Checking, also referred to as a Change Auditing System, is usually the best compromise between functionality and feasibility. This paper will examine operational aspects of implementing such a system.

IV. Basic functionality of a File Integrity Checker

Here is an outline of the basic principles of a File Integrity Checker (FIC):

1. FIC looks at configuration file to determine which files to check and what hashing algorithms to use.
2. Hash value is computed for file "A". $V_1 = \text{hash}(A)$
3. Computed hash value is compared to a previous stored value of the same file. $V_1 = V$
4. If $V_1 \neq V$ then report discrepancy.



There at least two systems that implement the Integrity Check in real time, every time a file is opened by the operating system.[4][5] Both of these research projects require kernel modifications to an open source Unix derivative. This limits their applicability in mainstream production computing environments.

A number of other systems provide Integrity Checking as a batch mode, non real time process. These do not require kernel modifications and can be run on a wide variety of commercial platforms. Their main drawback is that during the intervals between batch checks a compromised file could be used. This group is lead by the venerable Tripwire, the originator of the concept of hash based file checking.[6] Originating in 1992 at Purdue, the project was spun out in 1997 as a commercial entity. The source code for the original version was released as the Academic Source Release – free for use by universities and certain research institutions. Even though there is an open source implementation of Tripwire based on this release, there are licensing questions regarding it's use in non-academic settings. Aide is a project that re-implemented the functionality of Tripwire. It is actively maintained under the GPL license.[7] Another open source file checker is AFICK.[8] This is written in Perl and runs on any platform that the Perl interpreter has been ported to. It has a serious design flaw that renders it unacceptable in a production environment. This will be discussed in a later section describing problems with hashing algorithms.

We will look at the steps required to implement the Aide program since it is a much more 'basic' package than Tripwire but performs most of the same functions. Most deployment steps discussed could be applied to Tripwire as well.

V. Implementation steps

There are some basic design principles to follow in laying out a usable system for File Integrity Checking.

- The master system driving the FIC must be trusted
- The executable programs run by the FIC must be trusted
- There must be no "tracks" of the FIC process left on the client systems
- All data files produced by the checks must be written to read-only media
- All scripts and executable files should reside on and be run from read-only media
- The master system should have minimal exposure to production networks
- This file auditing system itself must be periodically audited

The build process must be run on a system that you can trust to be untainted. There is always a chance that an attacker has replaced files that are part of the building tools with trojaned versions. These could detect that they were compiling the Aide

package or libraries and insert backdoors. One suggestion is to download and verify all the source files, then do a fresh install of the operating system from vendor supplied CDs. Do the minimal install to create a functional system. Disable all unneeded services and daemons. If it's a Linux system, consider turning on the SE Linux extensions. Run any system hardening scripts you deem appropriate. Make sure this build platform is NOT connected to the network. Compile and install Aide on this machine and run a baseline integrity check. This system can then be used as the master system for driving checks on all your other hosts. It is possible to be too paranoid, but it doesn't hurt in this situation.

Acquire the source code for the Aide package and the prerequisite libmhash package from http://www.sourceforge.net . Be sure to verify the download integrity with the appropriate PGP key. If you are installing on a Unix or Linux platform, you will probably already have the required compiling tools:

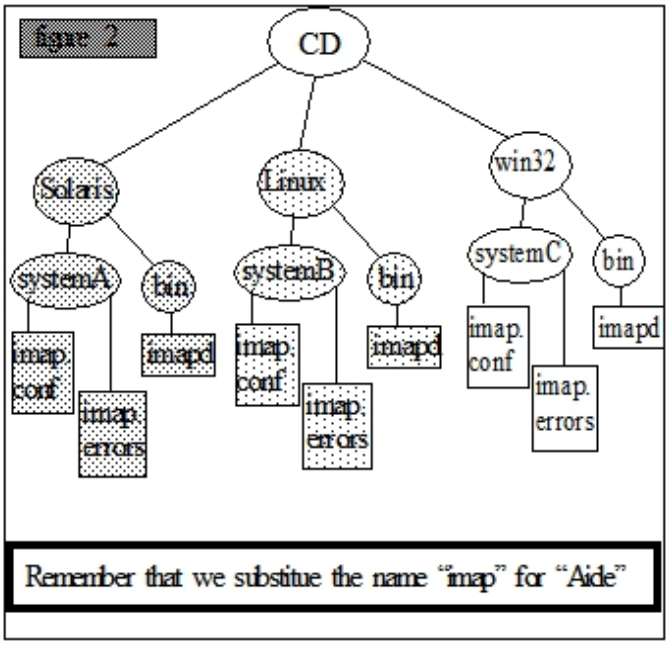
C compiler (such as Gcc)
GNU flex
GNU yacc (bison)
GNU make

If your target platform is Windows, download and install the 'cygwin' package first. This will provide a Unix environment on top of the Windows OS and includes the required tools.

Read through the instructions for building and installing both packages. Extract, compile, and install the libmhash library first. Run the 'configure' script for Aide. Remember to follow the steps to link the executable as 'static'. All library functions must be part of the executable, not provided by dynamically loaded library modules or dll's.

You will need to produce executables for each type of platform to be scanned. Either cross-compile or do a trusted build as described above. Their will be a unique configuration file for each system. Once the application has been built and tested, move all source files and documentation off to CD and put it away for safe keeping.

Rename the executable and configuration files. If an intruder should happen to gain access to the master system you don't want to advertise the fact that you're running Aide to check file integrity. Give it the name of some standard utility package that you do not have running on your servers. Perhaps "imapd" and "imap.conf" or "nessusd" and "nessus.conf". Alter the name of the past and current hash DB file as well; perhaps "imap.errors"? The Aide executables, all requisite scripts, and immediate past hash database files will be written to a CD, with multi-session disabled. See figure 2 for a suggested file layout .



The master system will need two CD-RW drives and at least two network connections. It should not be permanently connected to the production network (the network where the servers to be scanned reside). This interface should be disabled and only activated during the scanning process. Use DHCP if possible and make sure it is set to acquire a new IP address, not to renew a previous lease. Disable the interface after the last scan terminates. The directory structure outlined above will be on 'yesterdays' CD. This should only be mounted during the scanning interval. Create a volatile directory (RAM disk, tmpfs under Solaris) to store the output files of the current scans. Make sure this is cleaned out at the end of the process. Add a boot time startup script to scour this directory in case the master system crashes during the scanning. At the end of the process, the static binary files and configuration files along with the current hash DB generated by the scan will be written to the 'current' CD in drive two. Unmount and disable both drives at the end of processing.

VI. Details of the File Checking Process

A. Defining the List of Files

Aide can check a number of file attributes in addition to running the hash algorithms on the file contents. If the file modification time is 05:30 this morning, you don't need to compare the hash signatures to determine that the file has changed. Here is a list of the common attributes checked:

```
permissions
inode
number of links
user
group
size
```

```
block count
mtime
atime
ctime
check for growing size
```

The hash values of files can be determined by one or more of these algorithms:

```
md5
sha1
rmd160
tiger
```

The Aide configuration file is where you define which checks to run against which files and directories. This entry:

```
/etc p+i+u+g
```

Means to check permissions, inode, user, and group attributes for the `/etc` directory and all it's files and subdirectories. The configuration file allows for macro definitions so that combinations of checks can be easily defined:

```
SystemFile = p+i+n+u+g+s+b+m+c+md5+sha1
```

This would be used as:

```
/var  SystemFile
```

So the `/var` directory tree would be checked for permissions, inode, link count, user, group, size, block count, mtime, and ctime attributes. MD5 and SHA1 hashing would be run on each file and directory. To exclude a portion of the defined directory tree:

```
!/var/spool/.*
```

This will exclude the `/var/spool` subdirectory and all its contents, assuming that we are not interested in these transient files.

You must build your file checklist with care. Include all executables and libraries supplied with the OS installation. Include the install directories of third party application packages. Exclude those directories used for temporary storage or scratch space. On Unix systems, you'll find a number of "pseudo-filesystems" that should be excluded, such as `/proc` , `/var/run`, and others. Just make sure that any excluded real directories are cleaned out on a regular basis. Exclude users home directories. Running an Aide scan on a web browser cache directory is not pleasant. The user support forum for the Aide package can be the source for pre-built configuration files for specific platforms. Verify them against your installation before using. Test your configuration files in your development and test environments before installing in production. The fine tuning of the configuration file to minimize false positives is an iterative process.

B.Initiating the File Scan

All activity of the File Integrity Checker will be driven from the master system. Keep in mind that the execution of the hash functions can be quite CPU intensive. The file attribute scan is I/O intensive. Define a window of time when the target system is lightly loaded. Create a randomizing process that will select a random start time within the window, with allowance to complete the scan before the end of the window. Assuming that you will be scanning multiple targets, also randomize the sequence they get run in. This will break up the load patterns on the targets so that the fact that we're running a nightly FIC will not be readily obvious to attackers.

At the picked time, activate the network interface to the production network. Mount the 'yesterday' CD on the master. Export or share this to the target. Mount another scratch filesystem to the target. Initiate via SSH or some other secure means the FIC script for the target system. This will specify the appropriate executable and the system-unique configuration file. Output will be written to the remote scratch filesystem. On the completion of the scan, umount these two remote filesystems. Repeat this process in a random order and timing to all other target systems.

Upon completion of the entire scan sequence, run the comparison process between each pair of old and new signature files for each target system. If all goes well, there is 100% correspondence between the 'yesterday' and 'today' files and nothing else needs to be done.

VII.Reconciliation of File Changes

Following appropriate principles of separation of responsibilities, the FIC processing must be managed by the Information Security group, not the System Administration group. The output of the signature file comparison will come to the Security group. Part of the regular change management function should be a pre-notification of file modification. There should be a document that says:

```
Change control number 123456
April 27 2200 hrs.
Apply  Solaris10  patch  112233-44  to
systemA
files updated:

/usr/bin/ksh
/usr/bin/ps
```

If the file comparison for `systemA` on this date shows that the only files that are different are `/usr/bin/ksh` and `/usr/bin/ps` then these differences are already accounted for. If there are additional discrepancies, the system administrator for `systemA` must be notified and required to explain. Hopefully, the existing Change Control system will be

flexible enough to allow for all the normal daily functions of system administration. Any modifications, additions, or deletions of system files could be recorded by Change Control even if they should fall outside the mandates for a formal approval process. If not, then there should be some auxiliary mechanism for the Sys. Admin. to notify the security group of file changes. Either way, there must be some auditable record trail of every file modification that occurs on a production system.

The other possibility of the discrepancy reconciliation process is that a system file has been modified without the knowledge of the Sys. Admin. It could be something as innocuous as "Solaris patch 112233-44 also replaces the file `/usr/lib/libresolv.so.0` but did not list this in the README file.". Or something like "Update to backup script in Change Control ticket 556677 causes the dump process to update the file `/etc/dumpdates` .". It could also be an indicator of a system configuration problem: "Permissions were incorrectly set on directory `/usr/local/bin` and Oracle DBA installed a script file there. Problem corrected and the DBA has been signed up for Change Control training."

Of course, the worst case scenario is that we have uncovered a system file that has been replaced by an attacker for some malicious purpose. The most common case is the installation of a "RootKit" where some piece of malware is installed and running and various OS tools and utilities have been replaced by versions that will not report the presence of the malware process. There should be predefined policies in place for dealing with this scenario. These would typically include taking the server offline immediately, double checking any data updates or transactions, analysis of system and network log files. The affected file or files could be restored from backup or the entire system could be re-installed from a known clean image backup. A full File Integrity Check would need to be re-run before placing the system back into production. In all cases, all file actions would be logged with the Security group for audit purposes.

The overall outcome of the reconciliation process is that all reported file modifications are either accepted or rejected and repaired. These changes will need to be reflected in the 'today' version of the signature file. Once all signature files are accepted for all target systems, these files and the Aide executables and configuration files are burned to a CD. This disc will now become tomorrows 'yesterday' disc. The current 'yesterday' CD will be removed and stored for auditing purposes along with all relevant logs and documentation regarding reconciled file modifications.

VIII. Validity of the File Integrity Check

There are several issues regarding the validity and thoroughness of the FIC process as currently implemented by the Aide or Tripwire systems.

A. Not a Real-Time Check

As was pointed out previously, this is a batch mode process. Files are not checked in real time prior to opening. This is a compromise to be able to use existing operating systems and applications without requiring extensive modifications. There is flexibility in how often the scan is run. You could run it more frequently on highly critical directories and leave the full scan for a nightly run.

B. Only Useful for Checking Static Files

The FIC works by comparing a previously calculated, assumed good, file hash value with the currently calculated value. If the file has undergone legitimate modifications, the comparison will still report a discrepancy. For a legitimate file changes, a new hash value must be computed and used as the baseline figure. This will work for files that update infrequently, such as from a patch installation, but will not work for files that change many times a day, such as an email Inbox file.

Database systems that store data and indexes as regular files under the operating system cannot be checked with a FIC. The hash comparison can only detect that the file has changed, not that field 4 of record 192 in database *Customers* has been modified.

C. Is the Hash Comparison Really Foolproof?

The FIC can check most of the intrinsic file attributes, such as timestamps, owner, size, and i-node. Certainly, any unintentional file modifications would leave these markers in their altered state. A determined adversary with some level of skill could record these file attributes before making the malicious modifications and place them back on the altered file. It is highly likely that the code for producing RootKits will do this to avoid being discovered by a casual inspection of a directory listing.

The validity of several widely used hashing algorithms has recently been questioned in the cryptographic community. In 2004, a group of Chinese researchers published a paper describing their success in producing "collisions" in four widely used hashing functions.[9] In the context of hashing functions, a collision means there can be two non-identical messages (blocks of text) that produce the same hash value, thus appearing to be identical messages. The major implication of these findings is for those data communication systems that use these hashing algorithms to guarantee the validity of messages used in banking transactions, military communications, and other encrypted data streams. The original paper did not describe the algorithms used to produce the collisions. Subsequent researchers have succeeded in either discovering the "Chinese trick"[10] or in reimplementing their efforts.[11] It is probably only a matter of time before malicious entities disrupt encrypted communications by either inserting an arbitrary message with a valid hash value (denial of service) or by altering the message contents but leaving a valid hash.

In our discussion of File Integrity Checking, a compromised hashing algorithm has other implications. An attacker would

have a known, static file to work with, not a real-time data stream. For example, he could take the Windows system library file "authz.dll" and compute the MD5 hash value of the legitimate file. He could then prepare a bogus version of the same file from malicious code, pad the file to make it the same size, and manipulate the padding bytes to match the MD5 hash of the original file. This could then be installed by various means on a target system and would be undetectable by conventional means. If you are running a FIC and have it set to only do an MD5 hash check, the scan will not report a discrepancy. This is why it's vital to configure the FIC to calculate more than one type of hash value. While it is a foreseeable possibility that a malicious file could be crafted that would pass for the original by matching its size and a single hash value, it is inconceivable (at present) that such a trick could be used to match hash values of two or more different hashing algorithms. This is why I disqualified the AFICK system from consideration in section IV. It provides only an MD5 hash check.

Of course, it would still be wise to pick hashing algorithms for the FIC that are still considered resistant to compromise today. Of those currently supported by Aide (see section VI.A), the RMD160 and Tiger functions have no reported compromises.[12]

Hopefully, continued research into the integrity of hashing algorithms will produce new and more resistant functions or methods to strengthen existing implementations.[13]

IX. Conclusion

Implementing a File Integrity Check process for your organization can be a worthwhile tool in the security arsenal. It can add a "belt and suspenders" approach to existing Intrusion Detection/Prevention Systems. While there are definite drawbacks to the implementation of current FIC systems, you will likely find that the results are quite useful.

References

[1] The Software & Information Industry Association , Software Industry Statistics Page - Q1 2005,http://www.siia.net/software

[2] Donna Scott, VP and Research Director, "Best Practices for Operational Change Management", Gartner, Inc. 2003

[3] Greg Ashworth, former CTO of Rhône-Poulenc Pharma, conversation 06-2006

[4] V. Serafim, R. Weber, "Restraining and repairing file system damage through file integrity control",Computers & Security (2004) 23, 52-62

[5] M. Borchardt, C. Maziero, E. Jamhour, "An Architecture for On-the-fly File Integrity Checking", Lecture Notes in Computer Science, Volume 2847 / 2003, pp. 117 - 126

[6] *Tripwire (company)*, Wikipedia article, http://en.wikipedia.org/wiki/Tripwire%2C_Inc.

[7] *AIDE - Advanced Intrusion Detection Environment*, http://sourceforge.net/projects/aide

[8] *AFICK (Another File Integrity Checker)*, http://afick.sourceforge.net/

[9] X. Wang, F. Guo, X. Lai, H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD", eprint archive 2004/199, http://eprint.iacr.org/2004/199, presented at the Crypto 2004 rump session, August 17, 2004

[10] P. Hawkes, M. Paddon, G. Rose, "Musings on the Wang et al. MD5 Collision", *Cryptology ePrint Archive*, Report 2004/264, 13 October 2004

[11] Vlastimil Klima, "Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications", Cryptology ePrint Archive: Report 2005/102

[12] Paulo Barreto, University of Sao Paulo, "The Hashing Function Lounge", http://paginas.terra.com.br/informatica/paulobarreto/hflounge.html

[13] M. Szydlo, Y. Yin, "Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing". IACR Eprint archive 2005 #248