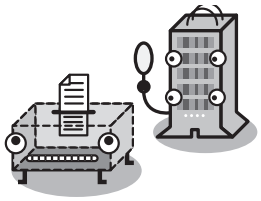
Chapter 36
Detecting Intrusions: File Integrity



File integrity tools can detect unauthorized modifications to critical system files and data.

Technology Overview

You suspect that a hacker got into your network. But how do you know if any data was damaged or changed? Even worse: you have no idea that a hacker got into your network, but accounting just called and found a really strange discrepancy in the books.

What people think: If a hacker gets into our network and does damage, it'll be obvious.

What we think: It's easy to tell if you've been hacked if the hacker changes your company Web page. But what if he breaks in and subtly change a few files? What if the programs you use to check up on your system have been replaced by hacked versions that hide the hacker's activities? How will you know?

File Integrity tools help determine if critical system and data files have been tampered with or altered. When something looks unusual the integrity checker will send out some type of alert (an email, a message to a pager, and so on). Some integrity systems will automatically replace the tampered file with a version that's known to be safe. This process can help detect and recover from intrusions. It can also help with general system problems such as corruption due to hard drive failure. Many situations that might result in the destruction of data can be identified, remedied, and possibly prevented with file integrity tools.

Most of the major file integrity tools will first make sure that critical system files haven't been altered. Under Windows, this means the registry, startup files (autoexec.bat), and many of the files that live in the Windows, or WinNT directory, especially the major system libraries (*Dynamic Link Libraries* [dlls]). It might also include major Microsoft programs such as Outlook, Word, and Excel. Under UNIX systems, the core files are the system configuration files, the boot files (kernel), the standard system programs (/bin, /sbin), standard library files (/lib), and some critical user applications (/usr/bin, /usr/lib, /usr/sbin).

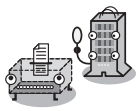
Past the basics, the rest is often up to the administrator to configure. Checking all files for changes is not an option as some files change too frequently while others are simply not important. The administrator will usually have to approve all changes to files that are being monitored. This can be a timesink, so finding the right balance between effective monitoring and critical monitoring is important.

How File Integrity Works

The fundamental concept behind file integrity is the ability to compare a file to a known good version of that file. Imagine you had a copy of every critical system file burned to a CD. You know that it's impossible to alter the CD (the CD drive on your server can only read), so you can safely assume that none of the files on the CD have been modified. The files on the CD are referred to as known good. There might be a serious problem if a file on the hard drive is different from the corresponding known good file on the CD.

If it were possible to maintain a CD drive with an updated copy of a systems' critical files, checking file integrity would be easy. The problem is that many critical files on a system occasionally change, but need to be monitored anyway. These files cannot be burned once, and considered "known good" forever, so a more flexible system is needed.

One of the better solutions is to take a snapshot of critical system files and place the snapshot in a protected location. While more flexible than the CD solution, this requires the use of hard disk space. To improve efficiency, most file integrity systems do not make a complete copy of each file. Instead, a mathematical process called a hash function is used to take a fingerprint/signature of each file. These fingerprints are securely stored in a protected location. If any portion of a critical file changes, the fingerprint of the altered file won't match the fingerprint of the original file.



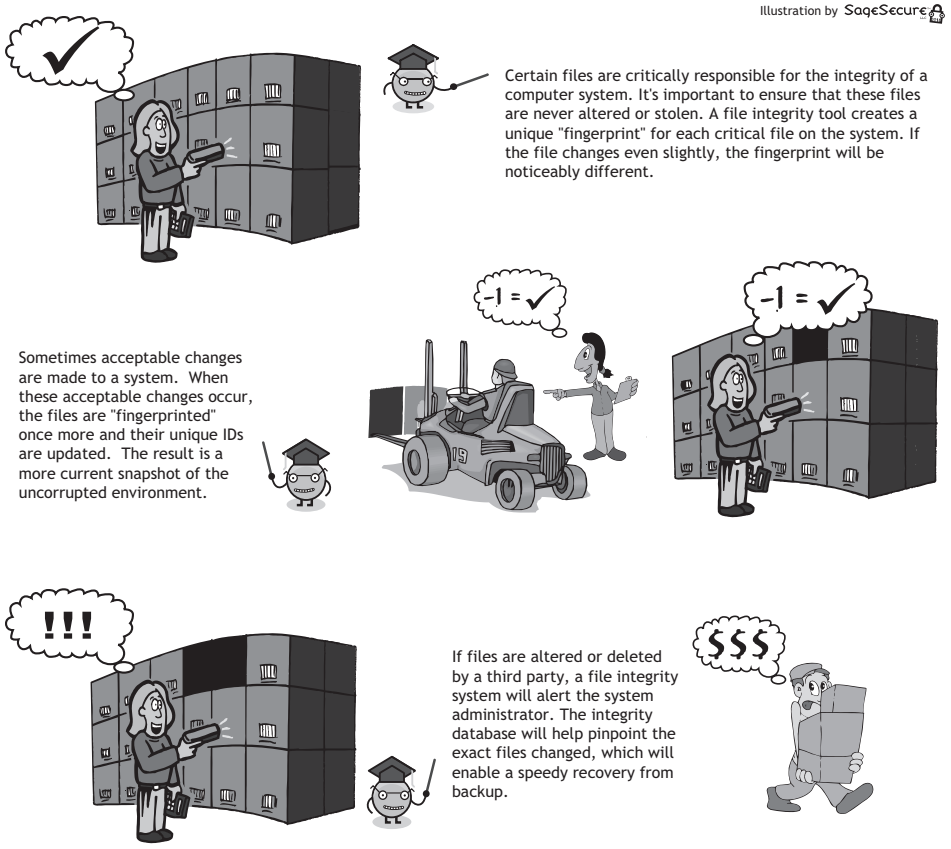


Figure 36-1

An automated process continuously takes fingerprints of all the critical files and compares them to the known good fingerprints in secure storage. An alarm is triggered whenever a mismatch occurs. This often results in an email or pager message being being sent out to the system administrator(s).

Security Considerations

Sneaking Past the Guard: Integrity checkers do not constantly check all of the files—it would hurt the performance of the system. Therefore, most checkers use a schedule. With a little bit of monitoring, a hacker can figure out the schedule and do

all the dirty work in between scans. With any luck, all the damage can be done and cleaned up before the checker makes its rounds a second time.

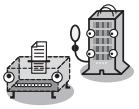
Blocking the Alarm: Most file integrity systems send out alerts via email. This is an obvious solution, but what if the email doesn't get out? A smart hacker may launch a *Denial of Service* (DoS) attack against the network's mail server. If the integrity checker can't reach the mail server, the message will stay on the hacked system. The resulting delay can buy the hacker enough time to delete the outgoing message and prevent the alert from ever being sent. The hacker could also use a number of tricks to cause the mail server to send all further alerts to an invalid system.

Compromising the Comparison Process: The program that actually does the file comparisons is vulnerable to being directly attacked. If a hacker can obtain access to the program files, he or she may be able to replace it with a non-functional version. The hacker can also alter the database of fingerprints, deleting or updating entries to reflect the new changes. Most current systems are vulnerable to this type of attack if the hacker gets full access to the system.

Altering Files without Changing the Fingerprint: Technically, it should be impossible to alter a file and still get the same fingerprint. In reality, it is possible, although very difficult, to accomplish. Of course, if somebody creates and distributes a point-and-click program that can automate this process, it's no longer all that difficult!

Memory Hacking: A number of critical system programs are loaded into memory when the computer first boots or when the program is first run. The core portion of the operating system (called the kernel) is one such system program. Other programs are placed into memory for efficiency purposes. Few, if any, file integrity systems can check the integrity of software that resides in memory. Hackers can either alter the version of these memory resident programs, or create a new version in memory that replaces the original. When the system or a user runs the program, the hacked version in memory gets executed instead of the clean version on the disk drive. This will continue as long as the system doesn't reboot and the memory doesn't get flushed. The result is a complete evasion of the integrity checking system. The most advanced hackers directly manipulate the kernel of the machine, giving themselves ultimate control over the system.

Database Data: Some information is highly valuable, yet changes often. Databases tend to hold information that is constantly being added, deleted, and edited by applications and users. How can you tell which modifications are legitimate and which are not? In most cases, the task is simply impossible. File integrity tools won't work. Instead, the database and related applications need to rely on their own security and integrity systems. Some interesting systems have been created to establish patterns of normal database use and observe anything out of the ordinary. Massive changes to table structures, large inserts/deletes, or modifications to password tables can trigger alarms in these systems.



Monitoring Too Much: Configuring the system to look at too many files will cause unnecessary alerts. For example, user data and temporary data should not be be monitored. It is important to flag only critical files for monitoring. While this may take a greater deal of time initially, it will save time in the long run.

Monitoring Too Little: If you don't monitor enough, you might miss something important. For example, some applications create their own user accounts and store configuration information in nonstandard locations. If you're just tracking system files, you'll miss any changes to application configuration files.

Ignoring Reports: On a busy network, things will change frequently. Rarely will these changes be the work of an intruder. After a while, the frequent alterations that result from routine changes can lull those watching into an apathetic state.

Poor Initial Design: If the systems being monitored are not designed well, or if the monitoring system is deployed inaccurately, there will be lots of spurious reports.

Making the Connection

Local Filesystems: File integrity systems check the local file system. If a security issue with the local file system exists, it creates a weakness that can be exploited to bypass the integrity checker.

Storage Media: As with local file systems, issues with the storage medium itself can be exploited by a hacker to work around an integrity checking system.

Internet Services: Alerts are often sent via email, which can be blocked or intercepted.

Intrusion Detection: Integrity checkers are often key components of any IDS.

Viruses and Trojans: Integrity checking systems and virus checkers are opposites of each other. The virus checker assumes that all files are good unless they match a "bad" virus signature. The integrity checker assumes files are bad unless they match a "good" signature.

Best Practices

Proper Configuration: It sounds so simple, but it's so rarely done. Spending the time planning out the files that need to be integrity checked and then activating the system *before* putting it online is critical. We've been guilty of putting off installing an integrity checker. By the time we got around to it, we couldn't guarantee that the system was in a known good configuration. So we had to reinstall the whole thing from scratch. Not fun. If high security is a must, think about this next suggestion.

Using Read-Only Media: A good strategy for systems that need to be very secure is to keep the integrity checking program, configuration files, and fingerprints on read-only media. In an ideal situation, you'd have a read-only CD drive in the machine and a CD-R in it with all of the critical integrity checking files and fingerprints. Whenever the fingerprints or configuration changes, an external CD burner is attached to the machine and the CD is updated. In such a situation, the hacker would have no opportunity to alter the the signature database as the CD is read-only.

This method doesn't completely stop the hacker—a number of ways to work around the integrity checking system are available, but it does mean that you can always trust the signatures on the CD. Therefore, in a clean controlled environment, files can always be compared from a clean source.

Period Reboot-Comparison: Every once in a while, you should shutdown critical machines and then load them using a custom boot disk. This boot disk should have a clean version of the integrity checking program and clean versions of all the critical signatures. The hacker can't control the environment on your boot disk—it stays locked away in a safe somewhere. Therefore, you're guaranteed to get a reliable status report on the cleanliness of your system.

Alternate Alerting System: It's a good idea to use an alternate system besides an email alert, since email service is easily disrupted. Hooking up a pager system is one option. Another is to invert the system: have an "OK" message sent out periodically. If too much time passes between "OK" messages an alert can be triggered.

Final Thoughts

File integrity is one of those ideas that sounds great, until it comes time for implementation. In practice, file integrity takes determination and patience to effectively implement. Furthermore, integrity is only good if the baseline file is clean. Can you trust most systems to have clean files? The best time to effectively implement file integrity tools is when a system is newly installed. Otherwise, relying on the integrity of the system files usually requires a file wipe and rebuilding the system from scratch.