System Baselining – A Forensic Perspective

Klayton Monroe and Dave Bailey

Version 1.3[*]

Abstract

In the analysis of a compromised system, it is important to identify what has been compromised, recover as much useful state information as possible, and restore the system to a usable, but less vulnerable state. The purpose of this paper is to demonstrate the utility of system baselining as a technique that supports these goals.

From a forensic point of view, the ability to detect change correctly and consistently is a top priority. When computers are accessed in an unauthorized manner, the state of various file objects will change. This change can manifest itself in various ways – slow system response, additional daemons in the process table, lost or damaged data, et cetera. Unfortunately, without the tools and techniques to detect and evaluate such change, systems could operate for quite some time, perhaps indefinitely, without exceeding operational limits generally considered to be abnormal.

Using tools and techniques that are "court worthy" will get the job done without ruling out the possibility of pursuing legal remedies. FTimes – a system baselining and evidence collection tool – addresses these specific problems and goals and is well-suited to handle the types of applications and environments incident handler's are likely to encounter.

1 Introduction

Incident response handlers are faced with responding whenever and wherever incidents occur. In enterprise environments, decision makers, administrators, and response handlers may all be in different locations and time-zones, and they may all be remote from the equipment. This problem is often complicated by an unwillingness, on the part of the victim, to vacate the premises while it is, in effect, burning down. Forensic examination of systems usually requires that they be shutdown so that their disks can be extracted and imaged. Victims are often unwilling to shutdown production systems for any reason, and they will seldom be amenable to having disks removed especially if it requires hardware disassembly.

Even worse, the client is frequently indecisive about the course of action to take: pursue the perpetrators and prosecute, or clean up, patch, and move on. In many cases, the client does not have adequately trained staff on-site who could assist with the initial response effort.

To combat this problem, practitioners need high quality, lightweight, easy to use tools and techniques. It is important to quickly identify what has been compromised, to recover as much useful state information as possible, and to restore the system to a usable, but less vulnerable state. One extremely useful technique in this situation is to create a "system baseline." The baseline provides a snapshot of the state of the system at the time the incident response begins. In some cases, the baseline may be sufficient to determine what occurred and to direct the response effort.

Since the victim may not initially be able to decide whether to pursue attackers through prosecution, it is incumbent on responders to begin as though prosecution was desired. Later, legal remedies may be

[*]Version 1.0 April 2000; Version 1.3 September 2006

abandoned, and some unnecessary work will have been done. However, if the response is not initiated in a way that supports prosecution, it is unlikely that such a decision could be taken at a later time – generally, poor handling of evidence would no longer support prosecution.

This paper defines baselining terminology, explains the mechanics of baselining, compares and contrasts different baselining techniques, and describes FTimes – a system baselining and evidence collection tool. The paper also explores some of the criteria that evidence collection tools and techniques must satisfy if they are going to support prosecutions. In closing, it presents a pair of war stories that are typical of the times.

2 Background

This section introduces and defines baselining terminology, describes and discusses critical baselining factors, explains the mechanics of baselining, and states baselining objectives from administrative and forensic perspectives.

2.1 Terminology

A *baseline* is a set of critical observations or data used for comparison [Mer94]. A *snapshot* is an impression or view of something transitory [Mer94]. In the context of file integrity, a snapshot refers to a set of critical observations (i.e., file attributes) taken at a particular point in time from one or more file objects stored on a given system. When a particular snapshot is used as the reference in change analysis it is called a baseline. In other words, a baseline is a particular, designated snapshot. The baseline is usually, but not necessarily, the first snapshot taken. This paper defines *change analysis* as the process of comparing a snapshot to a baseline. *Baselining*, in this context, is the process of taking a snapshot and subsequently designating it as the baseline. A *message digest* is a cryptographic hash computed over a given block of data such as a file's content. The term *subject* will be used, throughout this paper, to describe some aspect of the system being baselined or examined.

2.2 Critical Baselining Factors

There are four critical factors involved in baselining: provenance, perspective, acuity, and integrity.

2.2.1 Provenance

Provenance is the origin or source of an object [Mer94]. Provenance plays an important role in change analysis. One of the major advantages of baselining is that a file can be summarized by its attributes. If these attributes provide enough detail about a given file, then changes to that file can usually be detected by comparing snapshot data to corresponding baseline data. In particular, if a file's content changes, one may conclude with near certainty that the file's hash has changed. However, one can conclude nothing about the nature of this change unless the new hash represents a well-known value. In other words, a hash is useless for characterizing file content unless its provenance is known. This is because it is computationally infeasible to reconstruct a file's content from its hash. In the absence of such a binding, the practitioner must inspect a file's content to determine its type, purpose, and origin (if possible).

One way to determine the provenance of a given file object is to compare its hash to a previously recorded hash for that object. If there is no previously recorded hash, one may[1] be able to use hashes computed from the original distribution media to determine provenance. However, it would not be prudent to assume that

[1]This, of course, depends on how distributions are packaged – individual files may need to be uncompressed, decrypted, or dynamically created during an installation.

distributions are sacred; they can be compromised too, and this should be kept in mind when conducting post mortem analysis.

If provenance can't be confirmed through hash comparisons, it can, in some cases, be asserted. This is done by manually inspecting a given file. Once the file has been fully characterized, its hash may be computed, and the two (file and hash) may be bound together for future reference.

Sometimes exact provenance isn't needed to identify a file that is part of an intrusion. For example, in a directory such as `/usr/bin` consisting of "old" files, a newly created or modified file is sufficiently out of place to warrant further attention. Hidden directories, especially where no subdirectories would be expected, are always out of place. Files that grow between snapshots that are not known to be log files are also suspicious. Indicators such as these may be all that is needed to set the response handler on the right path.

2.2.2 Perspective

Perspective is the capacity to view things in their true relations or relative importance [Mer94]. Baselining tools must run in or be able to create an environment that maintains perspective. Without this, a tool can only see what it is allowed to see. In other words, it can only see part of the necessary context. An attacker with sufficient privilege may decide to construct an environment that artificially alters perspective. Baselining tools that run in such an environment produce results that can not, in general, be trusted. Perspective can be artificially altered by modifying application code, libraries, or system calls. In some cases, the same result can be achieved by manipulating relevant application or system data.

A good example of an environment that artificially alters perspective would be a compromised system where a rootkit has been installed. "A rootkit is a collection of tools an intruder brings along to a victim computer after gaining initial access. A rootkit generally contains network sniffers, log-cleaning scripts, and trojaned replacements of core system utilities such as `ps`, `netstat`, `ifconfig`, and `killall`. Although the intruders still need to break into a victim system before they can install their rootkits, the ease-of-use and the amount of destruction they cause make rootkits a big threat for system administrators. ... Arguably the most severe threat to system security that can be caused by a rootkit comes from those that deploy LKM (Loadable Kernel Module) trojans. Loadable Kernel Modules are a mechanism for adding functionality to an operating-system kernel on the fly – without requiring a kernel recompilation [Alt01]."

2.2.3 Acuity

Acuity is keenness of perception [Mer94]. In other words, it is the relative ability of a tool to resolve detail. Without good acuity, baselining tools effectively have impaired vision – they can only see what they were built to see. If the condition is bad enough, be assured that attackers will take advantage of the situation.

For example, in Microsoft's NTFS (New Technology File System) a file object is implemented as a series of streams. By default, file content is stored in the object's unnamed stream. However, a user can choose to create and store data in an alternate or named stream. One problem with this is that programs like `dir` and `explorer` have no mechanism to enumerate alternate streams. Consequently, such streams remain hidden to the observer. Even worse, the WIN32 API, itself, does not provide an adequate mechanism to enumerate alternate streams. Tool designers who want this level of acuity must either use the Backup{Read|Seek|Write} routines or drop down a level, to the Native API, and use the undocumented NtQueryInformationFile routine.

Perfect acuity is realized only when the designers of tools fully understand and can obtain necessary and sufficient access to the underlying mechanics of the system technology they are trying to interrogate.

2.2.4 Integrity

Integrity is an unimpaired condition or the quality or state of being complete or undivided [Mer94]. In other words, it's the state of being uncorrupted, complete and sound. The integrity of the baselining tool, its input, and its output must be protected throughout the baselining process, and, if possible, indefinitely. In any case, the baseline's integrity must remain steadfast throughout its lifespan if one expects to assert that future change analysis will yield accurate, reliable, and consistent results.

Two general types of attacks that can be used to compromise the integrity of a baselining tool and/or its data are on-disk and memory resident modification. On-disk modification of the tool/data could be accomplished by replacing it or applying a patch (i.e., through modification to a portion of the tool/data). Memory resident modification could be accomplished by manipulating memory buffers either while they are in memory or swap space.

2.3 Baselining Objectives

In our forensic practice, we approach baselining from two perspectives: forensic and administrative. The ultimate goal of a forensic process is to reconstruct, as accurately as possible, those events that led to change. The administrative process, on the other hand, is more concerned with the problem of automatically detecting, reporting, and possibly correcting change for a large number of host systems.

2.3.1 Forensic Objective

The forensic objective of baselining is to record an accurate snapshot while minimizing or eliminating perturbation to the subject system. Primary concerns are preservation of evidence, integrity, correctness, completeness, error reporting, consistency, and ease-of-use. Time and automation considerations are important, but not priorities. One of the forensic practitioner's priorities is to build a case based on evidence that stands up to cross examination in court. It may be the case that such evidence never makes its way to court, but adopting evidence collection and analysis processes that are "court worthy" is a good practice because it creates a firm foundation that is defensible.

2.3.2 Administrative Objective

The administrative objective of baselining is to record an accurate snapshot while minimizing the administrative effort involved. Primary concerns are time involved, automation, scalability, built-in analysis, correctness, and ease-of-use. The administrator wants problems to be detected and corrected automatically. Administrators also want intelligent alerting and solid logging facilities. In other words, "don't bother me unless there is a serious problem, but when that happens, give me access to a log trail that contains as much as possible."

2.4 Baselining Mechanics

The processes for creating a baseline and subsequently detecting change can be summarized as follows:

1. Define the set of attributes that are relevant to the type of change you wish to detect. Next, develop or obtain a minimally invasive tool with good acuity that systematically traverses specified directories and files and collects or derives the chosen attributes.

2. Choose a perspective that supports your baselining objective(s), and baseline the subject using the tool developed or obtained in step one. Archive the results in a manner that maintains provenance and preserves integrity.

3. At some point in the future, create a snapshot by repeating step two. Determine change by comparing the snapshot to the baseline.

4. Periodically repeat step three reverting, instead, to step two whenever a new baseline is desired.

3 Baselining Techniques

This section describes four different techniques for baselining a system: Alternate Platform, Alternate Operating System, Single-User-Mode, and Multi-User-Mode. These techniques are ordered from a forensic point of view according to their ability to maintain perspective and integrity. For each technique, we point out the advantages and disadvantages of its approach. In the end, the practitioner must determine, on a case-by-case basis, which technique to use based on available resources, time constraints, and operational limitations.

3.1 Alternate Platform

The safest way to create a baseline, from a forensic point of view, is to utilize a dedicated, stand-alone computer that has been configured specifically to mount and scan disks extracted from subject systems (i.e., a baselining system). However, the process of extracting subject disks may not be easy or practical, and it can be fraught with technical problems that, if not handled correctly, could result in irreparable damage.

Advantages

1. This technique is completely independent of the subject system. This provides the necessary out-of-band perspective to analyze the extracted disks.
2. This technique can effectively access all files contained within subject file systems because the baselining tool runs outside the context of the subject operating system. A baselining tool that operates within the context of a running operating system can't always access all files maintained by that system due to kernel or other locking mechanisms. This can be true even if the baselining tool runs with full system privileges.
3. If properly protected, the integrity of the baselining tool and the data it collects will be guaranteed.
4. The baselining system can mount subject file systems with software read-only access. Older SCSI disks may have a physical jumper that can be set to enforce hardware read-only access. Also, write blockers can be employed to ensure read-only access.

Disadvantages

1. The practitioner must supply, maintain, and control access to the dedicated hardware/software resources that constitute the baselining system.
2. This technique is time consuming, and it requires human interaction and physical access to all systems involved.
3. The subject operating system must be taken off-line. Down time could last minutes to hours.
4. The subject system may need to be disassembled to extract its disks. This increases the risk of irreparable damage. Also, the practitioner must have a working knowledge of the various hardware components that may be encountered.

3.2 Alternate Operating System

Another relatively safe way to create a baseline is to boot the subject system into an alternate operating system that can access and scan subject file systems. To minimize perturbation, the practitioner should not write output to subject media. If possible, subject file systems should be mounted with software read-only access.

Advantages

1. This technique is completely independent of the subject operating system. This provides an out-of-band perspective to analyze the system's disks. However, if the system's hardware/firmware has been compromised, perspective could be at risk.

2. This technique can effectively access all files contained within subject file systems because the baselining tool runs outside the context of the subject operating system.

3. If properly protected, the integrity of the baselining tool and the data it collects will be guaranteed provided that the system's hardware/firmware has not been compromised.

4. The baselining operating system can mount subject file systems with software read-only access.

5. Baseline output may be directed to one of several different types of media such as parallel/serial/usb port, network, attached SCSI, PCMCIA, et cetera. This gives the practitioner more ways to overcome operational limitations (e.g., no external SCSI port).

Disadvantages

1. The practitioner must supply, maintain, and control access to the dedicated hardware/software resources that perform this function. This is complicated by the fact that each subject system encountered may have unique characteristics that require different hardware/software resources and support.

2. This technique is time consuming, and it generally requires human interaction and physical access to the subject system.

3. The subject operating system must be taken off-line. Down time could last minutes to hours.

4. The practitioner must have detailed knowledge of the boot process for each subject system that is to be baselined – today many systems support multiple boot devices and various boot options.

5. The alternate operating system must provide driver support for the various subject file systems that may be encountered.

6. If the alternate and subject operating systems coexist, then it may be possible, from the context of the subject operating system, to corrupt the baselining tool or its data either prior to or after taking a snapshot.

3.3 Single-User-Mode

The third technique for creating a baseline is to bring the subject system into single-user-mode. In this state there should be a minimal number of system processes running, networked shares should not be mounted, normal users should not be able to access the system, and it should be possible to re-mount some or all subject file systems with read-only access. Unfortunately, not all systems support single-user-mode (e.g., Windows NT/2K/XP), so this technique does not apply universally.

One major issue that must be considered with this technique is where output will be written. If possible, output should be written to external media that has been attached specifically for this purpose. In some cases, it may turn out that the only practical place to store output is on a subject file system. In forensic scenarios, this is less desirable because it implies that potential evidence will be overwritten in the process. Baselines taken under these conditions still have value, but the practitioner must weigh the anticipated benefits and risks prior to taking action.

Advantages

1. This technique provides a restricted access environment for creating a baseline.
2. This technique can effectively access most to all files contained within subject file systems. Because the baselining tool runs within the context of the subject operating system, it may lack the necessary privileges to access all file objects.
3. In many cases, subject file systems may be re-mounted with read-only access. This provides additional protection against system perturbation.

Disadvantages

1. This technique is completely dependent on the subject system. Therefore, the baselining tool is forced to operate with an in-band perspective. This implies that output may not be complete or trustworthy.
2. This technique is time consuming, and it generally requires human interaction and physical access to the subject system.
3. The practitioner may be required to store output on a subject file system. This increases the likelihood that other potential evidence is lost, and it exposes baseline data to attack.
4. The subject system must drop into single-user-mode which effectively means the system will be unavailable. Down time could last minutes to hours.
5. The executable used to generate the baseline is at risk to subversion in two forms: in memory during execution and on disk before execution. On disk modification can be mitigated by running the executable from read-only media (e.g., CDROM). The likelihood of in memory modification is reduced, but not eliminated, by the fact that the system is in Single-User-Mode.

3.4 Multi-User-Mode

The final method of creating a baseline is to do it under normal system operation. In this state it may or may not be possible to re-mount subject file systems with read-only access. Issues regarding where output will be written must be considered here just as they were in the Single-User-Mode case. From the forensic perspective, this technique is the least sound of all the techniques, yet because of time or operational constraints, it may be the only viable technique.

Advantages

1. This technique could be effected without physical access to the subject system provided that remote access already exists.
2. This technique can effectively access most to all files contained within subject file systems. Because the baselining tool runs within the context of the subject operating system, it may lack the necessary privileges to access all file objects.

Disadvantages

1. This technique is completely dependent on the subject system. Therefore, the baselining tool is forced to operate with an in-band perspective. This implies that output may not be complete or trustworthy.
2. Resulting output must either be stored on the system being baselined or to a network resource. In both cases the data's integrity is at risk. Use of encryption can reduce this risk, but may not be able to eliminate it.
3. It is unlikely that any or all of the subject file systems can be re-mounted with software read-only access. Even if a file system could be re-mounted this way, it may not be practical to do so for operational reasons.
4. The executable used to generate the baseline is at risk to subversion in two forms: in memory during execution and on disk before execution. On disk modification can be mitigated by running the executable from read-only media (e.g., CDROM).
5. The system is available to users and normal processing continues. This may result in incomplete, inconsistent, or incorrect results. Evidence may be altered or destroyed before it can be collected.

4 FTimes

This section discusses FTimes (File Topography and Integrity Monitoring on an Enterprise Scale), a system baselining and evidence collection tool written to improve the practitioner's ability to conduct incident response and post mortem analysis. In particular, this section will address the motivating factors that led to the tool's development, provide a basic overview of the tool and its capabilities, describe how FTimes supports file topography, and explain how it relates to each of the critical baselining factors.

4.1 Motivation

When conducting incident response, one quickly realizes that time and spatial constraints are two major factors that degrade the overall effectiveness of the response. To combat these factors, response handlers need tools that are simple to use and can be deployed safely and securely in remote environments. These tools need to be so simple that any green administrator could correctly initiate diagnostic or evidence collection procedures with minimal coaching. They also need to provide (or be used in conjunction with) a mechanism that safely and securely transmits the output to a location of the handler's choosing.

One important objective in all of this is to build tools that can be used to support prosecutions. In order to sustain a procedure in a court, it must be possible to demonstrate the tool's scientific basis. This is necessary to satisfy the court's responsibility as the "gatekeeper" for admissible evidence established by the US Supreme Court in the *Daubert* decision.[2] *Daubert* was a case involving scientific investigation, and it was later asserted that the decision did not apply in situations involving technology. However, the Supreme Court affirmed in *Kumho Tire*[3] that the same scientific standards apply to technology as well as to pure science. Basically, this means that the scientific or technological practitioner must follow the precepts of "good science." The techniques should satisfy several questions (see Table 1) including "whether the theory or technique in question can be (and has been) tested, whether it has been subjected to peer review and publication, its known or potential error rate, and the existence and maintenance of standards controlling its operation, and whether it has attracted widespread acceptance within a relevant scientific community."[4]

[2]Daubert v. Merrell Dow Pharmaceuticals, Inc., 509 U.S. 579, 589, 1993.

[3]Kumho Tire Co. v. Carmichael, 526 US 137, 1999.

[4]Summary of *Daubert* decision. Cornell University Legal Information Institute. `http://supct.law.cornell.edu:8080/supct/html/92-102.ZS.html`

Table 1: **Daubert Tests**

Can the procedure be and has it been tested?
Has it been published and subjected to peer review?
Does it have a known or knowable error rate?
Does it have standards controlling its operation?
Has it been accepted by practitioners in the field?

Consequently, we assert that a system baselining and evidence collection tool should have the following characteristics:

- easy to use
- does one thing extremely well (i.e., collect file attributes)
- utilizes a simple, effective, and well understood algorithm that can be applied equally well to different operating systems
- generates output that is easily assimilated by a wide variety of existing tools
- has built-in logging that is complete, precise, and useful for analysis purposes
- is accurate, efficient, and minimally invasive
- doesn't need to be installed on the subject system
- is small enough to run from floppy even if statically compiled
- provides only a command line interface

It was precisely these factors and characteristics that led to the creation and ongoing development of FTimes.

4.2 Overview

The primary purpose of FTimes is to gather and/or develop topographical information and attributes about specified directories and files in a manner conducive to intrusion and forensic analysis. FTimes was designed to support the following initiatives: content integrity monitoring, incident response, intrusion analysis, and computer forensics.

FTimes is a lightweight tool in the sense that it generally doesn't need to be installed on the subject system, is small enough to fit on a single floppy, and provides only a command line interface.

Preserving records of all activity that occurs during a snapshot is important for intrusion analysis and evidence admissibility. For this reason, FTimes was designed to log four types of information: configuration settings, progress indicators, metrics, and errors. Output produced by FTimes is delimited text, and therefore, it is easily assimilated by a wide variety of existing tools.

FTimes implements two basic capabilities: file topography and string search. File topography is the process of mapping key attributes of directories and files on a given file system. String search is the process of digging through directories and files on a given file system while looking for specific sequences of bytes. Respectively, these capabilities are referred to as mapping and digging.

FTimes supports two operating environments: workbench and client-server. In the workbench environment, the operator uses FTimes to do things such as examine evidence (e.g., a disk image or files from a compromised system), analyze snapshots for change, search for files that have specific attributes, verify file integrity, and so on. In the client-server environment, the focus shifts from what the operator can do locally to how the operator can efficiently monitor, manage, and aggregate snapshot data for many hosts. In the client-server environment, the primary goal is to move collected data from the subject host to a centralized system, known as an Integrity Server, in a secure and authenticated fashion. An Integrity Server is a hardened system that has been configured to handle FTimes GET, PING, and PUT HTTP/S requests.

4.3 File Topography

This section focuses exclusively on how FTimes maps file objects – a process known as file topography. FTimes supports several other modes of operation (e.g., compare, dig, get, etc.). These modes are peripheral to the subject of system baselining. Therefore, they won't be discussed in this paper. Detailed information regarding all modes of operation may be found in the FTimes man page [Mon06].

To create a snapshot, FTimes essentially takes a list of directories and files as input, iteratively and/or recursively gathers attributes, and returns a set of text records containing one or more delimited attributes for each object encountered. In this context, an object is a directory, file, or stream. Some attributes (e.g., Size, ATime, etc.) are simply collected from existing data structures while others such as MD5, SHA1, or Magic are calculated or derived based on predetermined algorithms or signatures.

The FTimes mapping facility provides three modes of operation: mapauto, maplean, and mapfull. The first mode is designed to support ad-hoc but fairly narrow applications. Consequently, it only allows the operator to specify a FieldMask (i.e., what attributes to collect) and a list of directories and/or files to map. Its primary benefit is that output is written directly to stdout and stderr. Thus, it may be used in scenarios where writing to the subject's local disk is out of the question (e.g., during an investigation). The latter two modes were designed to provide a rich set of controls that may be used to customize runtime behavior. These controls are managed through the use of a configuration file.

Table 2 summarizes those attributes that are collected or derived from common file systems that FTimes supports. An 'X' indicates that a given attribute applies to the corresponding file system. An 'S' indicates that a given attribute may be simulated by the host operating system. In other words, it doesn't really exist in the underlying file system but the operating system provides a value for it anyway. The actual name that FTimes uses to refer to a given attribute is displayed in parentheses. File system types and attributes are described in appendices A and B respectively.

FTimes collects critical timestamp values before they are altered as a side-effect of the baselining process. This is accomplished, in part, by utilizing a depth-first search algorithm on directories. Initially, it was thought that FTimes should actively restore timestamps as part of the baselining process. This idea, while it seemed like a reasonable approach, was abandoned due to our belief that evidence collection tools should not attempt to artificially alter system state – such a capability could cast a shadow of doubt as to whether or not the tool is actually collecting or creating evidence.

A critical component of FTimes is the ability to accurately calculate message digests (e.g., MD5, SHA1, etc.). Typically, a digest algorithm takes an input message of arbitrary length and produces a fixed-length, message digest – also known as a fingerprint, hash, or checksum. "It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest [Riv92]." In other words, a message digest provides a mechanism to verify data integrity. A corollary to this proposition is that when the content of a particular message (e.g., data within a file or stream) is changed, its hash will also change.

4.4 Relationship to Critical Baselining Factors

FTimes was designed with provenance, perspective, acuity, and integrity in mind. However, the ability of FTimes to support each of these factors depends on the environment in which it runs. To a large extent, that

Table 2: **Attributes Collected by FTimes for Common File Systems**

	EXT{2|3}	FAT	FFS	JFS	NTFS	UFS
ATime (*atime*)	X	S	X	X	X	X
Attributes (*attributes*)	.	X	.	.	X	.
CTime (*ctime*)	X	X	X	X	X	X
ChTime (*chtime*)	.	S	.	.	X	.
Device (*dev*)	X	.	X	X	.	X
File Index (*findex*)	.	X	.	.	X	.
GID (*gid*)	X	.	X	X	.	X
Inode (*inode*)	X	.	X	X	.	X
MD5 (*md5*)	X	X	X	X	X	X
MTime (*mtime*)	X	X	X	X	X	X
Magic (*magic*)	X	X	X	X	X	X
Mode (*mode*)	X	.	X	X	.	X
NLinks (*nlinks*)	X	.	X	X	.	X
Name (*name*)	X	X	X	X	X	X
RDevice (*rdev*)	X	.	X	X	.	X
SHA1 (*sha1*)	X	X	X	X	X	X
Size (*size*)	X	X	X	X	X	X
Stream Count (*altstreams*)	.	.	.	.	X	.
Stream MD5 (*md5*)	.	.	.	.	X	.
Stream SHA1 (*sha1*)	.	.	.	.	X	.
Stream Size (*size*)	.	.	.	.	X	.
UID (*uid*)	X	.	X	X	.	X
Volume (*volume*)	.	X	.	.	X	.

environment is defined by the baselining technique employed.

FTimes supports provenance through its ability to calculate and analyze hashes. This task becomes more difficult without a known good baseline. With original distribution media and a spare system in hand, constructing and baselining a second system that approximates the subject is relatively straight forward. The resultant baseline may then be used to distinguish known good files from those that need review.

FTimes, itself, can't create or enforce an environment that maintains perspective. However, perspective can certainly be maintained if FTimes is used in conjunction with either the Alternate Platform or Alternate Operating System technique. Thus, the relationship between FTimes and perspective is the fact that it draws attention to the need for perspective.

For the attributes that it collects, FTimes has good acuity. Great care was taken to ensure that the data collected by FTimes is complete and accurate. FTimes does not, however, have perfect acuity. This is primarily due to the fact that not all possible attributes for each supported file system are collected. Originally, this came down to a design trade-off between special case complexity and maximum portability. Another reason that FTimes does not and may never have perfect acuity is the fact that it was designed and implemented using an imperfect process. Yes, tool builders can get it wrong despite their best intentions and efforts. This can happen for any number of reasons, but some of the more common reasons include: lack of subject matter knowledge, invalid assumptions, poor programming practices, and lack of testing. Fortunately, FTimes is open source software, so anyone can review the source code and test the tool's acuity under varying circumstances.

FTimes contains several mechanisms that work to ensure integrity: data uploads over HTTPS to include mutual authentication through digital certificates, dynamic creation of an output hash (i.e., a hash of the harvested output), and separation of log and data output streams. Additionally, the Alternate Platform and Alternate Operating System techniques may be used to provide even greater integrity assurance.

5 War Stories

FTimes has been used effectively and with great success in the areas of Incident Response (IR) and Integrity Monitoring (IM). The stories presented in this section are representative of the types of scenarios that are fairly common today. The first emphasizes how FTimes helped an IR team collect and analyze evidence in a situation where there were many constraints, including no prior system baselines, working against them. The second emphasizes how FTimes, being used in an IM scenario, provided the necessary information to quickly recover from a break-in.

5.1 Uptime at All Costs

Between January 31 and February 1, 2002, several identical systems were compromised. Administrators initially realized that something was wrong due to a failure in a critical application. Upon investigating this failure, administrators determined that several system files had changed. Analysis of the Network Intrusion Detection Systems (NIDS) logs revealed evidence of probing activity and successful attacks. Later, it was determined that root access was obtained and that rootkits were installed on each system.

These systems were all identical in configuration and mission – Solaris 2.6 out-of-the-box installations with third-party software that implemented a mission critical function. Due to deployment and support agreement constraints, the IR team was not allowed to take systems off-line to image their disks. The fallback plan was to take a snapshot of one affected system, copy off any interesting files, and perform live `dd`s of as many file systems as possible – all of this activity had to take place within the allotted one hour maintenance window. Unfortunately, it was also the case that these systems did not provide `ssh/scp` or floppy access. Consequently, data and diagnostic tools were moved around with the least common denominator (i.e., `tftp` and `ftp`). A statically compiled version of FTimes was brought in to take the snapshot.

To make things worse, the client had no prior system baselines. Fortunately, the IR team was able to baseline a system that had been freshly restored from vender media. The team felt that the chance of this system being tainted was small due to the fact that its snapshot was taken shortly after the system was restored.

Change analysis revealed classic evidence of a rootkit (i.e., a hidden directory and trojaned system binaries). Later, the actual rootkit was recovered, and its contents matched those found in the hidden directory. Because the IR team was able to baseline a freshly restored system, change analysis yielded a good reduction in the amount of by-hand analysis needed to reconstruct the events that took place. The baseline and snapshot contained 20,104 and 21,251 records respectively. Of these records, there were 379 changed, 92 missing, and 1,239 new files. Further by-hand analysis quickly narrowed this list down to 113 entries that were blatantly suspicious or unknown. When it was all said and done, it took approximately 10 minutes to map both the victim and restored machines. It took about 3 seconds for FTimes to perform change analysis, and the by-hand effort took less than one hour.

5.2 Just Another Day at the Office

On January 2, 2001 at 09:31:47 a Linux system was attacked through `rpc.statd` – a daemon that implements the NSM (Network Status Monitor) RPC (Remote Procedure Call) protocol. The attacker gained root access with this attack, and by 09:34:50 two new system accounts had been created: cgi (uid=0) and killer (uid=504). At 09:35:00 it just so happened that an FTimes' `cron` job began scanning various system critical files. This particular job ran every hour and was configured to upload its data to an Integrity Server where it could be protected, archived, and analyzed.

Within 10 minutes an alert email was generated and sent to the administrator of the system. When the administrator saw the message in his mailbox, he immediately knew something was up because, under normal circumstances, none of the monitored files were supposed to change.

The administrator looked at various system log files and consequently found evidence of the attack and the two new accounts. Instead of over-reacting, he calmly halted the network subsystem, permanently disabled the vulnerable daemon, and removed the new system accounts. Then, he rebooted the system and ran a full snapshot. This data was also shipped to the Integrity Server. After reviewing the change analysis for the entire system, he was satisfied that no additional damage had been done, so he went to breakfast.

6 Conclusion

This paper has defined baselining terminology, explained the mechanics of baselining, compared and contrasted different baselining techniques, and described FTimes – a system baselining and evidence collection tool. It also explored some of the criteria that evidence collection tools and techniques must satisfy if they are going to support prosecutions. In closing, it presented a pair of war stories that are typical of the times.

When it comes to incident response and forensic analysis, one snapshot is better than none. Two snapshots are better than one, and a complete history of snapshots is nearly ideal. However, even if no baseline exists, one should not rule out the usefulness of baselining tools. A significant amount of information can be collected or derived from data collected by such tools even though a history of prior snapshots does not exist.

Incident response is fraught with constraints. Therefore, tool designers need to take into account these issues and compensate, where possible, in their designs. Further, tool builders need to design their tools with Daubert principles in mind. Specifically, such tools need to have open architectures and utilize open data formats so that other practitioners and tool builders may thoroughly understand and appreciate their operation.

The design and construction of FTimes was influenced by all the considerations presented in this paper. Thus, it has a strong foundation from forensic and incident response perspectives. The fact that it is also ideal for integrity monitoring applications is simply a pleasant artifact of its design.

From a forensic perspective, the safest way to baseline systems is the Alternate Platform technique. In reality, however, the practitioner is more likely to be faced with the constraint of operating in a multi-user-mode environment. FTimes attempts to compensate for this deficiency through its lightweight design. Additionally, its ability to use strong authentication and encryption for snapshot uploads mitigates the inherent risk involved with transmitting sensitive data over untrusted networks.

FTimes can provide useful time and labor saving information during an incident response even where no prior information has been collected. When used as a regular part of system monitoring, FTimes can also provide warning of unauthorized system access. By comparing the baseline data to known data, the recovery process can also be dramatically shortened. In some cases, a situation that might ordinarily require rebuilding the system can be shortened to restoration of a few critical files.

A File System Descriptions

EXT{2|3}	Second or Third Extended File System (Linux)
FAT	File Allocation Table (NT/2K/XP)
FFS	Fast File System (FreeBSD)
JFS	Journaled File System (AIX)
NTFS	New Technology File System (NT/2K/XP)
UFS	UNIX File System (Solaris)

B Attribute Descriptions

ATime	ATime represents the time of last file access.
Attributes	File attributes is a combination of one or more of the following values: archive, compressed, directory, hidden, normal, off-line, read-only, system, and temporary.
CTime	For Microsoft operating systems, CTime represents the time that a given file object was created. For UNIX operating systems, CTime represents the time of last status change. In other words, it's the last time that a file's inode or meta data changed.
ChTime	The ChTime attribute is specific to NT-based operating systems, but remains an undocumented attribute. In practice, however, it appears to be analogous to the UNIX CTime attribute.
Device	The Device attribute represents the serial number or ID of a given file system. When used in conjunction with the Inode attribute, file objects within that file system are uniquely specified.
File Index	The File Index attribute represents a unique ID for each file within a given file system. When used in conjunction with the Volume attribute, files within a given file system are uniquely specified.
GID	The GID attribute represents the group ID that is bound to a given file object.
Inode	The Inode attribute represents a unique ID for each file within a given file system. When used in conjunction with the Device attribute, files within that file system are uniquely specified.
MD5	A 128-bit cryptographic hash of file's data.
MTime	MTime represents the time a file's content was last modified.

Magic	The Magic attribute provides an indication as to what type of data is stored in a particular file object (e.g., executable, text, jpeg image, etc.).
Mode	The Mode attribute is a combination of a file's protection (i.e., suid, sgid, sticky, and rwx,rwx,rwx) and device type bits (i.e., directory, regular file, socket, symbolic link, etc.).
NLinks	The NLinks attribute represents the number of hard links for a given file object.
Name	A fully qualified path. This attribute provides a descriptive way to identify unique files within a system while providing location information and hints as to what the file's purpose may be.
RDevice	The RDevice attribute contains the major/minor numbers for special device files. Otherwise, it is not defined.
SHA1	A 160-bit cryptographic hash of file's data.
Size	The Size attribute represents a file's size in bytes.
Stream Count	The Stream Count attribute represents the number of alternate or named streams bound to a given file object.
Stream MD5	A 128-bit cryptographic hash of stream's data.
Stream SHA1	A 160-bit cryptographic hash of stream's data.
Stream Size	The Stream Size attribute represents a stream's size in bytes.
UID	The UID attribute represents the user ID that is bound to a given file object.
Volume	The Volume attribute represents the serial number or ID of a given disk volume. When used in conjunction with the File Index attribute, files within that Volume are uniquely specified.

C FTimes Availability

FTimes is known to run on the following operating systems: AIX, BSDi, FreeBSD, HP-UX, Linux, OpenBSD, NetBSD, Solaris, and Windows NT/2K/XP. Source code and other project resources are available at:

`http://ftimes.sourceforge.net/FTimes/`

References

[Alt01] O. Altunergil. Understanding Rootkits, 2001. `http://linux.oreillynet.com/pub/a/linux/2001/12/14/rootkit.html`.

[Mer94] Merriam Webster's Collegiate Dictionary, 1994.

[Mon06] K. Monroe. FTimes Man Page, 2006. `http://ftimes.sourceforge.net/FTimes/Man+Pages/ftimes.shtml`.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, 1992.