ON THE ROLE OF FILE SYSTEM METADATA IN DIGITAL FORENSICS

by Florian Buchholz and Eugene Spafford

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

On the role of file system metadata in digital forensics*

Florian Buchholz
florian@cerias.purdue.edu

Eugene Spafford
spaf@cerias.purdue.edu

CERIAS
Department of Computer Science
Purdue University
656 Oval Drive
West Lafayette
IN, 47901, USA

Abstract

Most of the effort in today's digital forensics community lies in the retrieval and analysis of existing information from computing systems. Little is being done to increase the quantity and quality of the forensic information on today's computing systems. In this paper we pose the question of what kind of information is desired on a system by a forensic investigator. We give an overview of the information that exists on current systems and discuss its shortcomings. We then examine the role that file system metadata plays in digital forensics and analyze what kind of information is desirable for different types of forensic investigations, how feasible it is to obtain it, and discuss issues about storing the information.

Keywords: Computer forensics; Digital forensics; Audit data; File systems; Intrusion analysis

1 Introduction

An important area of digital forensics is the retrieval of information from a computing system. This area is usually split into two separate tasks: disk imaging and disk analysis. Although numerous tools exist that assist an investigator with these tasks [15, 18, 16, 8], there is still plenty of research to be done in the area. This is documented by various recent needs assessment studies [35, 32, 22] that show that there are challenges left in data acquisition and forensic technology, analysis, and tool development.

A recent gap analysis study by the Institute for Security Technology Studies [23] shows that for Category 1 of the National Needs Assessment [22] - Preliminary Investigation and Data Collection - the section for operating systems is well-covered with existing tools that address the needs. This could lead to the conclusion that data collection for operating systems is a solved problem, which would be a poor conclusion. The reason for this is that all the above studies and the gap analysis are only addressing the retrieval of existing information. Apparently, no one has yet bothered to ask the question: "What information would you like to have available as a forensic investigator?" Some of the problem has been addressed by Kuperman [27], but the actual quality of the information retrieved on current systems and its usefulness for forensic purposes has not been addressed by any research, nor has the feasibility of obtaining and storing the desired information been examined.

We can roughly categorize desired information into the following groups:

*Published in Journal of Digital Investigation, Vol. 1(4), pp. 297-308, December 2004

1. Information that is available to the system and recorded on non-volatile media
2. Information that is available to the system but is not recorded
3. Information that is not currently available to the system but could be made available
4. Information that is impossible to be obtained by a computing system

Obviously, a forensic investigator will only have access to the first kind of information, or, if a live system analysis is performed, to the second kind. Most of the literature and development in the field is only concerned with the information that is actually present in the first category. We hold the opinion that it is the duty of future research to explore in what manner more forensically relevant information can be provided to an examiner in a feasible fashion. Considering the design of future systems it is useful to first evaluate what the desired information is, whether and how it can be obtained by a computing system, and if it can be stored in a reasonable fashion. This way, some or all information from the second and third categories could be moved to the first one, plus we will gain an understanding of what is possible.

The exact kind of information and the scope of its storage we are discussing in this paper will differ from system to system. One can hardly require all operating system vendors to start modifying their products to record more data for forensics, or force users to enable such logging. However, there are many cases where extra information is desirable, be it to be able to show due diligence, or to more quickly discover if a system was compromised or accessed in an unauthorized fashion.

In the following, we shall examine what kind of system information is currently available, which extra information is desirable from a forensics point of view, analyze how the currently existing information satisfies those wishes, and how feasible it is to obtain and store information that is currently not collected on a system.

2 Digital forensics

Casey and Palmer define *forensic* as "... a characteristic of evidence that satisfies its suitability for admission as fact and its ability to persuade based upon proof (or high statistical confidence)." [11]. When applying this definition to digital forensics, one can see that the area consists of members from a wide spectrum of disciplines and backgrounds. Apart from computer science, digital forensics is relevant to practitioners of disciplines such as law, law enforcement, politics, or standardization bodies. Literature in the field of digital forensics thus has much material to cover and needs to address a diverse audience including digital crime scene technicians, digital evidence examiners, digital investigators [11], lawyers, attorneys, judges, politicians, developers, and researchers.

Much of the current literature and guidelines for digital forensics focuses primarily on data retrieval and what information is present on existing systems. Given the diverse target audience and the different level of expertise in the area of computing in general, one of the main objectives is to teach practitioners the basic procedures of evidence retrieval and analysis on today's computing systems. Naturally, future research in the field of digital forensics cannot be addressed too thoroughly, although more recent publications also discuss research. Most of the current work explains how to recover data from a system in one form or the other. In respect to metadata, some of the work also discusses the forensic value and/or quality of the information that is found. By *forensic value* we mean the possibility to draw conclusions about events on the system from the data. For example, timestamps have a high value from an event reconstruction perspective because they allow an ordering of file operations into a timeline. This is provided that the timestamps have not been tampered with and that the system's clock is correct. Access control information on its own, however, holds less value from an event reconstruction point of view, because it generally only reflects static system policies, but does not provide information about individual events. The information that can be derived from access control information is a (group of) user(s) that may have had access to an object on the system. At this point further evidence (e.g. in the form of timestamps or login data) is needed to draw any conclusions. Under *quality* we understand how believable the information is. Is it easy to tamper with the information on the system? For example, on some operating systems a file's access and modification timestamps can be arbitrarily set by its owner.

In some cases, the discussion of evidence retrieval is limited to a description of where important system files are located and how to use tools that recover deleted files [12, 43]. Metadata is not discussed at all or only in the form of timestamps [43], but no critical discussion is given about the value of the forensic information. Other publications focus in great detail on the issue of information hiding and retrieval without mentioning file system metadata or issues such as how to obtain time, user, or location information [5].

Some of the current forensics literature actually addresses the value of file system metadata in the form of MAC times and user information [10, 26]. However, a critical discussion about the quality of the information is lacking. At some part of the discussion timestamps are presented as a powerful means to reconstruct events. However, either no critical discussion is given [10], or the whole value of timestamp information is undermined by statements such as: "Altering the modify and access times in an inode is simple, but not every suspect knows how to do it" [26]. The notion of an "owner" of a file is mentioned [26] but the term "owner" is not explained and may lead to incorrect assumptions about the relationship between a user and a file. A more thorough discussion about event reconstruction is given by Casey [11]. The actual techniques in terms of functional, relational, and temporal analysis are described on a higher level than what information a system may (reliably) provide. However, the author makes it very clear that timestamps may be altered and discusses techniques to detect the tampering or deduce the correct times of events. Carrier and Spafford [9] use the term *characteristics* of a digital object, the set of data and metadata associated with the object, in their event reconstruction model. This reflects the need for reliable metadata information for event reconstruction. They do not, however, discuss what the nature of these characteristics could or should be. Mohay et al. dedicate an entire chapter to research directions and future development [30]. Topics such as data mining, text categorization, authorship analysis, steganography, and cryptography are covered. In the part about evidence extraction they address the difficulty associating collected data from various sources with events on the system. They give a framework to correlate existing data on a system, whereas the purpose of this paper is to analyze what data can be added and how its forensic quality can be maintained.

All of the surveyed literature only describes the information that can be obtained from existing systems and this is their intended purpose. File recovery is the main focus, but a few documents elaborate on the value of timestamps or user information. In our survey of literature in the field we encountered no discussion about the requirements of future systems with respect to digital forensics and what type of meta information beyond MAC times and user information is desired. The discussion shows that the digital forensics community is aware of what tasks need to be performed and also aware of the fragility of digital evidence. What is lacking is an analysis of what information is necessary to perform those tasks or make them easier to perform, and further what kind of desired information can actually be obtained from a computing system.

3 A history of file metadata

Most computing systems have some type of long-lived data storage that may be examined for evidence. Even though it need not be the case for every system, the usual organization of this storage is comprised of files, directories, and metadata. For the remainder of this paper we will assume such an organization. As metadata we define all the data in the file system that describes the layout and attributes of the regular files and directories. This includes attributes such as timestamps, access control information, file size, but also information on how to locate and assemble a file or directory in the file system. This latter information contains pointers to data blocks, or even entire blocks used as internal nodes of lookup data structures such as B-trees.

File system metadata was not originally designed to be used for the purpose of reconstructing events that occurred on the system. Anderson [1] was the first to utilize such data for threat monitoring. He proposed to utilize System Management Facilities (SMF) records, which were kept by mainframe servers, such as those running IBM's OS/360.

In the 1960s most computing tasks were performed on mainframe computers, with OS/360 one of the dominating operating systems. Information stored on the servers' disks described the entire batch job of a user. The data for the jobs came from punch cards or tape media. The batch job information was kept in *records*, which described different aspects about the job, some describing the user data (which can be seen as a file). This included file type, minimum and maximum size, creation, access, and modification times, but also information about the job itself such as running times, duration and resources utilized. Compared to today's systems metadata there was more available information for forensic purposes. The information is still present on today's systems but usually not recorded or only in a temporary fashion such as the *proc* file system.

Multics was the first operating system that supplied a hierarchical file system, which is generally considered as the ancestor of today's most common file systems. Daley and Neumann describe in the Multics file system design paper [13] the need for users to store their data within the computing environment itself as opposed to storage media such as cards and tape. The user would have complete control and ownership of his data as well as the metadata. They formulated the following design objectives: "Little-used information must percolate to devices with longer access times, to allow ample space on faster devices for more frequently used files. Furthermore, information must be easy to access when required, it must be safe from accidents and maliciousness, and it should be accessible to other users on an easily controllable basis when desired." [13] To determine how frequently information was used they proposed an access timestamp. The need for modification and creation times came from the file system's backup system, which would commit newly created and modified files to tape backup. This is the original motivation for the use of today's MAC ("Modified, Accessed, Changed") times. To be able to allow other users to access files they proposed the inclusion of an access control list plus permissions (modes) for each file. All remaining metadata had to do with the actual on-disk layout of a file.

UNIX was introduced around 1970 [34] and its file system was strongly influenced by Multics. The metadata for a file was stored in an *inode* and it contained the file's location and size, its type (directory or file), the three timestamps, and the access control information. The latter was comprised of the user and group identifier and protection bits as all modern UNIX variants and derivatives use them today.

MS-DOS emerged in the early 1980s. Its file system, FAT [29], keeps track of the file type, size, location, and the timestamps. The space reserved for timestamps varies between 2 and 4 bytes, which results in differences in granularity. For example, the access time is only measured in days. Because DOS did not have any notion of a user, no user or permissions information is stored with FAT. The Windows operating system at first inherited the FAT file system, but when the limitations of FAT became too much of a problem, NTFS was introduced. NTFS carries detailed user and permission information as well as modified, accessed, created, and changed timestamps.

4 The relevance of metadata for forensics

If it were possible to record a system's state – register values, memory, timers, network events, interrupt information, etc. – for every single clock step, one could use that information to deterministically replay all events that took place on the system. The answers to most questions an investigator might have could be answered, albeit in a tedious and time-consuming fashion. Even if it were possible to record all that information it still would not be feasible as the amount of time necessary to record the information on a storage device would slow the system down several orders of magnitude. As this approach is not feasible, we have to utilize snapshots of the system's state instead. A snapshot reflects the system state at a given discrete point in time. In addition to knowing the actual state of the system for those points in time, one might be able to draw conclusions about the state changes that occurred between two given snapshots. Such an approach using external logging of processes, files, filenames, and system calls, was implemented by King and Chen [25].

Taking a snapshot of the entire system's state or large parts thereof on a frequent basis might be feasible for critical systems. In the general case, limited storage capacity and performance considerations prohibit this practice. For this reason we need to consider a further reduction of information quantity and frequency of recording, preferably through an already existing mechanism on the system. The hope is that through an audit trail of individual changes to parts of the system (small deltas in the system state) we obtain sufficient information to understand the changes in the system's state leading up to the current one.

Files play an important role in the operation of most computing systems. Usually the operating system itself as well as the boot mechanism are comprised of files. Program executables, configuration data and startup scripts, user information, as well as application data are stored in files. Therefore looking at files is a good indicator of what actions took place on a system: it gives a rough view of information flow, file accesses can show what programs were executed when, and file modifications show what was altered on a system. The operations on the system's files are only a subset of the system state. However, for the reasons discussed above, they can yield answers to many questions of interest to a forensic examiner. Furthermore, recording meta information about a file's operations as they occur is a mechanism that introduces little computational overhead. Thus a file's metadata seems a logical place to record our subset of the system's state. The metadata associated with a file can be seen as the characteristics of a digital object as discussed by Carrier and Spafford [9].

Information recorded by system or user programs may aid the forensics investigator during the analysis. System logging facilities such as *syslog* or shell history files and third party programs such as Tripwire [24] or tcpwrappers [42] provide valuable data for a forensic investigation. However, programs running outside of the system's kernel space may not have access to the necessary information. Plus, all the information is stored in files, which are subject to deletion or tampering. Other approaches that modify the kernel [3, 7] either do not store the added information on a permanent basis, or do so in log files, as well. Other approaches, such as Sun's Basic Security Module for Solaris [38] store extensive audit information in sequential log files using an "audit token" to relate records to each other. The audit records can become quite complicated and large in size and space management can be come very complex [17]. Furthermore, the information is not stored directly at the object of interest (i.e. the file) but rather operations on files have to be reconstructed from all of the audit records on the system. By storing the desired information directly as file system metadata we gain the following benefits:

- The information is automatically collected and stored by the system: all the information that is available to the system is available to be recorded.
- The information is collected automatically with no extra cost for setting up logging mechanisms.
- The information is directly stored with the object of interest. It is not necessary to correlate various system logs to obtain the desired information.
- Tampering with the information is not as simple as tampering with a file. If raw disk access is not allowed by the operating system, the recorded information is protected from all users. Even if raw disk access is allowed a malicious user still has to navigate the file system to get to the information. When modifying or deleting it he needs to be careful not to destroy any data that is crucial to the successful operation of the system.

5 Desired information

As can be seen from the overview given in Section 3, current operating systems and file systems were not designed with digital forensics in mind. On most Unix-like systems, the metadata associated with a file that holds forensically usable information is only 22 bytes, of which 16 are timestamps. In this section we will discuss the types of information that is desired from a forensics point of view.

When performing a forensic investigation on a computing system an investigator needs to reconstruct as many events and actions that took place on the system as are necessary to draw unambiguous conclusions. This may be as basic as locating contraband material on a system and determining when and how it got there. And even that is not a simple task. In its most complex form such an investigation will attempt to reconstruct all events that took place on a system. This could be to investigate a break-in or crimes committed by an insider. The main questions a forensic investigator has to ask are: *who, what, when, how, where* and *why.*

The *who* question is concerned with what user is (or which users are) responsible for certain actions on the system. *What* addresses what actions actually were performed on the system, *when* over which time interval they took place, and *how* in what manner those actions were executed. The *where* question is to determine both where the responsible users were located when they initiated the actions as well as where the data on the system, i. e. files, came from. Finally, the *why* question is concerned with the motives that lie behind the actions. As a computing system cannot know the intentions of its users the answer to this question is one that the investigator must infer from the answers to all the other ones.

5.1 Who did it?

The question of who is responsible for certain actions or the existence of data can be important in the course of an investigation. This holds especially true for systems with a large number of active users, such as a server in a thin-client environment or systems that offer network portal services. Most systems utilize some sort of authentication mechanism – usually a login procedure requiring a user name and a password – to bind a user identifier and often a group identifier to a process or a session.

The user and group identifiers for processes and files are also commonly referred to as the "owners" of those instances. It is thus tempting to associate everything that bears such an identifier as the result of that particular user's action. However, the original purpose of those identifiers in today's operating systems lies in access control, not true ownership. The only thing that may be deduced by looking at the identifiers is that a process bearing a particular id has been granted the permissions to an object that is associated with that identifier. Thus, in most of today's computing systems it is difficult or even impossible to determine the user id of the subject that is truly responsible for actions. The information might be gained by correlating other information, such as login times or typical user activities, with the times at which the file operations occurred, but this process is tedious and may not even lead to the correct conclusions as we discuss below. If the correct information is stored directly with the file, no correlation work will be necessary.

From a digital forensics perspective the question of who "owns" a file is irrelevant. We want to know who created, modified, accessed, and deleted it. So who performed those operations on a file? In many cases the user id of the file will be equivalent with the identity of the user responsible. There are exceptions, though. For example, a file may be created by User A who then changes the user id of the file to User B. In Unix, this may be done with the `chown` command. Such commands are used to transfer permission rights for an object from one user to the next, but once executed any notion of the creator of a file is lost to the system. For the remaining operations (modify, access, delete) it would make sense to look for the responsible users within the set of users that hold the proper permissions for that file. This set may be quite large (up to any user on the system), and also when the permissions or user and group identifiers change, the information deduced from that would be incorrect.

Simply introducing new fields that associate user and group identifiers with the various timestamps (MAC times) can solve the problems addressed above. However, this will not be enough to solve other ones where a user's actions affect a file. For example consider two processes, one controlled by User A and the other controlled by User B as illustrated in Figure 1. Process A read data from File 1. The two processes communicate using interprocess communication (IPC) and User A supplies the data it read from the file to B's process, which writes the data into File 2. The creator of the file clearly is User B. However, User A also played an important role in its creation and content. Current systems have no notion nor the ability to detect User A's effect on the file.

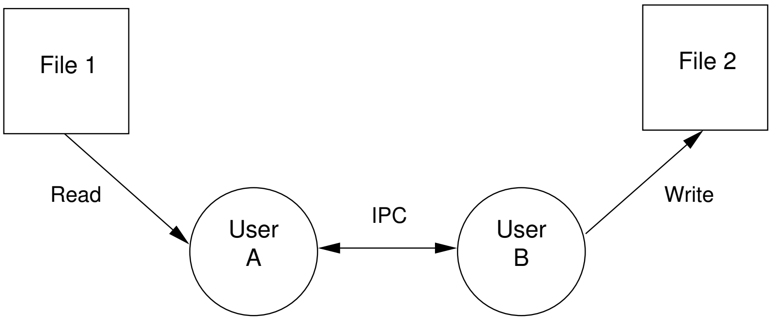

Figure 1: The contents of File 2 are influenced by User A

If we extend the example to more processes and users that play a role in the creation of the file we can observe that a fixed size field to hold user information for a file is not sufficient (unless it is large enough to hold information about all users). Also, if multiple users' processes communicate with each other, how can the system tell which ones played a role in a file operation and which ones did not? It turns out that this problem is actually undecidable for the general case. As the system is generally unaware of the information flow within a process, the only thing that can be done is observe its inputs (IPC in this case) and outputs (the creation of the file). Deciding which input caused an output to occur is equivalent to solving the Halting problem [39, 40]. The Halting problem states that given a Turing machine M and its input x, it is undecidable in the general case to determine whether M halts on x.

While this problem can not be solved, there are two ways to get around it. The first is to only allow programs to run on the system whose information flow has been determined through methods such as static analysis. In this case it is known what data from which input affects the output and the system has this information readily available. But while we obtain the correct desired information, we lose the ability to run any general program because of the requirement to perform information flow analysis for each program we allow to run. In addition to the fact that information flow analysis can be time-consuming, for some programs such an analysis might not even be possible because of randomness or race conditions. The second way to avoid the problem is using an approximation. Because we cannot tell which exact inputs affect the output, we can simply assume that all of them had an effect. This approach will result in some incorrect extra information being kept, but it also assures that no correct information is discarded. From a digital forensics perspective this is a good approach for two reasons: if information about a particular user is not associated with a file, we can be sure that that user did not have anything to do with the file's operation; also information that is present and might be false is better than no information being present at all, especially if the amount of false information is kept small. An approach using label sets bound to processes and system objects is currently part of our research [2]. We need to keep in mind that the information of what users played a role in the operations on a file cannot be determined in the general case, even though this information could be valuable for a forensic investigator.

By the discussion above we can see that the "who" information actually falls into Category 4 of our classification: it is impossible in the general case to obtain the correct information. We do believe, however, that the approximations we mention in this section are good enough in most cases and thus that recording of this information is justified.

5.2 Where did that come from?

There are many cases where it is desirable to know from where a particular file on a system originated. This origin information may be anything from an IP address [3, 4] to GPS coordinates [14]. It may even be as simple as a flag indicating "from within/from outside" the system. An Internet draft proposed by Leach and Salz [28] discusses Globally Unique Identifiers (GUIDs). A GUID has a fixed size of 128 bits and contains time and node (network location) information. The generation algorithms specified in the draft ensure that collisions among GUIDs occur only with a very low probability.

When locating contraband material, this kind of information may lead further down the chain of distribution, which may lead to follow-up investigations. When unknown files are found on a system, information about where the file came from may give clues as to what kind of file it could be, plus – if the file is malicious code – the information could be used trying to locate the author. Furthermore, some files may immediately be classified as benign on a system if their origin is from a trustworthy source. For example, this might be the operating system vendor's installation media for system binaries. With current systems, no such origin information is available. In some cases the origin of files may be deduced by correlating log information – such as mount times or web logs – with file timestamps. This information will only be available for origins that are logged on the system, plus some file systems do not have reliable creation timestamps and the information may be lost.

Where do files on a system come from? A user can create one by typing on the console, they can be read from storage media such as CD-ROMs or floppy disks, they can be downloaded from a network, they can be transfered from devices such as digital cameras or scanners, etc. Current computing systems have no notion of the origin of their files (or processes). The reason for that might lie in the fact that the systems from which today's modern ones descended were mostly self-contained and isolated units with their only inputs coming from a console or card readers. However, the problem of determining a file's origin is not as simple as the above examples might suggest. By definition a computer computes data from other data. The fact that a system produces new data makes what we mean by origin of a file (or data in general) more complicated.

Consider the following example: A user had typed and saved a C-code file on the console. He later logs in from a remote location and compiles it with a compiler that was installed from the operating system vendor's distribution CD. He also links in a library that was downloaded from an open source web site (see Figure 2). Taking into account the origins of the files and processes that played a role in the creation of the resulting executable, what should its origin be? Ideally we want to capture the origin of all data involved in the generation of the new data.

A specialized compiler that can take origin into account and knows which sources are used in the creation of a file could correctly keep track of all origin information. For general unknown programs that produce files, the same problem as with user influence applies: It is an undecidable problem. However, the same approximate solutions that can be used for the user influence problem can also be applied here.

As with the "who" information, the answer to the "where" question also falls into Category 4 of our classification. However, given that there is no location or origin information kept at all in existing systems, we believe that adding support for keeping and recording it will add substantial support for forensic investigation, especially in those cases where tracing back to an author of code or creator of contraband material is of importance.

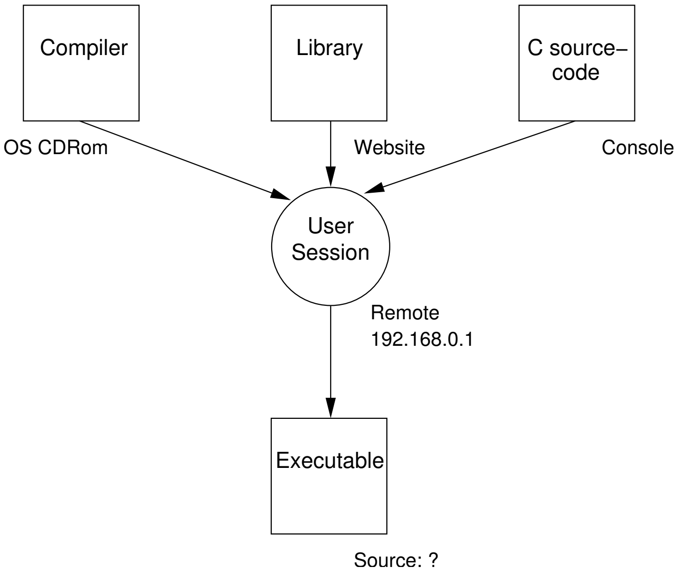

Figure 2: Given the origin of the involved entities, what is the origin of the new file?

5.3 When did what happen?

All commonly used file systems associate certain timestamps with their files and directories. The original purpose of timestamps is unclear, but many tools such as *find* [41] or *make* as well as back-up services utilize them.

The timestamps available for ext2 [6] are modification, access, change, and deletion times. The first three are commonly referred to as *MAC times*. NTFS has an altered (A), read (R), MFT changed (M), and creation time (C). As can be seen, different file systems have different kinds of timestamps available. Furthermore, confusion may arise from the different naming of the metadata fields. What is MAC in ext2 would be ARM in NTFS when using the Windows terminology. There is no deletion time in NTFS and there is no explicit creation time in ext2. This latter fact is a common misunderstanding about the file systems that emerged from UFS: the "C" in MAC time is often thought of as the creation time. Even the Linux source code refers to the *ctime* field in ext2 as "creation time." This is incorrect, as the original term is *change time*. This timestamp field is updated every time there is a change in the inode information (a change in the metadata itself). When a file is created, the ctime is set to the time of creation because in a sense the metadata has been changed. If the file's metadata changes after that the ctime is updated again. A simple change in permissions via *chmod*, or user or group via *chown* or *chgrp* is sufficient for updating the ctime field. In most cases the ctime will be equivalent to the creation time, but there is no guarantee for that and a forensics investigator needs to be aware of that fact [19].

For the Unix and Unix-like operating systems the POSIX standard [21] defines the timestamps, their meaning and when they are updated. The required timestamp fields under POSIX are modification, access, and change times. An overview of which Unix system calls change what timestamps can be found throughout the literature (e.g., see Stevens [37]). The specific time when an update has to occur is left vague, however. The standard specifies that the timestamp fields need to be marked for an update and that "[a]n implementation may update fields that are marked for update immediately, or it may update such fields periodically."

While some of the names for timestamp fields are unambiguous, others have room for interpretation. A creation and a deletion time are straightforward in their meaning, but what exactly is meant by access, read, modification, and alteration time? Simply because a metadata field is named a certain way does not mean that it really conveys the information suggested by that name. When is a file considered accessed or read?

As the semantics of the timestamps are not specified by any standard they are open to interpretation and different operating systems' implementations can show different behavior as to when the timestamps are updated. This may be sufficient for using timestamps with tools such as *make* and *find* or for backup purposes. From a forensics point of view a clear definition of what the timestamps mean is of much greater importance. Furthermore, different operating systems or file systems may keep different kinds of timestamps. As we have shown above, Unix-like operating systems usually do not keep a real creation time, whereas Windows using NTFS does.

The above examples of using *chmod* or *chown* to update a file's ctime show a side-effect of how timestamps may be tampered with using access control operations that affect the file's metadata. There are other commands, such as *touch* in the Unix world that allow a user to arbitrarily modify the timestamps. To obtain audit data of good quality in the sense we defined earlier creation, modification, and access timestamps should not be subject to alteration by any user. If modifiable timestamps are needed for certain tools or procedures such as backups, then they should be separate from the timestamps we use for forensics.

All current file systems have in common that only the latest respective time is kept. This is understandable for reasons of space constraints, but not satisfying from a forensics perspective. Ideally, all operations on a file should be recorded. A creation and a deletion time will only require one field, as the operation is only performed once per file. Operations that modify and access a file occur quite frequently, however, and there is no upper limit in the amount of space required to record these occurrences.

The GUIDs mentioned in the previous section also contain time information. Depending on what kind of location information one is interested in the GUID could replace time and location metadata fields. In this case the GUID is not bound to the file itself but rather to the individual actions to the file. However, as the above discussion shows, files may have many individual sources and events from the same sources may occur at different times. In these cases a coupling of location and time may not be desired and it might negate some of the space reductions discussed below. If the system does support GUIDs, though, the file metadata would be a good place to store it and use it for propagation purposes.

The "when" information falls into Categories 1 and 2 of our classification: some of it is recorded already, whereas the rest is available to the system but not recorded. For a forensic investigator it would be beneficial not having to worry about what timestamp means what on a particular system and which timestamps are available. For this standardized semantics of timestamps as well as a standard set of timestamps available are needed. The necessary information is present on all systems. While recording all of it might be infeasible, currently it is not possible to record the information at all, even if desired. A more fine-grained set of recording options as discussed below may help to adjust any particular system's requirements.

5.4 How did it happen?

The manner how an operation was performed on a file can be of importance to an investigator. By "how" we mean what controlling agent or executable program was used to perform the operations. It should make a difference for the course of an investigation whether a program named *explorer.exe* was used in the creation of contraband material on a system, or if the program was *backdoor.exe* instead. These programs can be seen as agents of the process's user to perform tasks on the system. The role of user agent may be further delegated to other programs, creating a chain of agents. For example, *command.exe* may invoke *explorer.exe*, which in turn may invoke *winamp.exe* to access an MP3 file on the system.

Having access to the entire chain of user agents for an operation on a file obviously can be valuable information in an investigation to reconstruct events on a system. Apart from scenarios described in the above example it also enables an investigator to identify automated system service routines that manipulate files such as automated system cleanup, log rotations, file indexing, etc.

It is necessary to record the chain of user agents because there are cases where ambiguities occur even with complete access and modification information available for a file and all executables on the system. This may happen, when several executables are accessed by the same process prior to performing the file operation.

In addition to simply referencing the name of (or a pointer to) an executable in the chain, it may also be desirable to know what the executable's version or patch-level is. For this purpose, a cryptographic checksum could be included in the chain information.

This kind of "how" information is currently not available on the common computing systems, but could easily made accessible by keeping a stack of agent information. Each time a user agent invokes another one, the information is pushed onto the stack and each time it finishes the information is removed. Thus, this information falls into Category 3 of our classification.

5.5 What was done to the file?

In addition to the metadata described in the previous sections the actual nature of the modification to a file is also important. Ideally, the entire chain of modifications from a file's creation through its current state should be available.

Alternatively to storing the actual modifications of a file, only the hash values of the different versions can be kept. This way it is at least possible to identify version changes for well known files, such as kernel versions and upgrades or patches to program binaries.

Considerable work has been done in the area of versioning file systems. However, the primary focus here lies in data recovery and undoing of write operations as well as versioning. Systems including AFS, Plan-9 [33], and WAFL allow for setting of checkpoints for files on a periodic basis. The Cedar file system [20] as well as the RSX, VMS operating systems, and the TOPS-20 file system [31] created new versions of a file but have limitations as to how many copies of a file may exist and use simple heuristics to decide what versions to delete after that. The Elephant file system [36] also provides the ability to keep a long-term or even complete history of a file. However, the long-term history is only achieved by retaining user-defined landmark versions and the authors do not address space considerations for the complete history option.

The "what" information is readily available on current systems but changes in files usually are not recorded. This falls into Category 2 of our classification. In general, for this type of information the same considerations are true as for the "when" data: it is probably infeasible to record everything, but even if desired the information can currently not be recorded at all.

6 Space considerations

In the previous sections we have suggested adding quite a considerable amount of new metadata to files. When recording all file operation times together with user information, origin, and user agent information there is theoretically no upper limit to the space requirements of that metadata. Practically, it might still be feasible to record at least part of the desired information.

6.1 Not all files are treated equally

Not all files on a system are used in the same manner or as frequently as other files. Current file systems do not differentiate between their files. In practice, though, we can roughly group files on a computing system into categories:

- executable files

- configuration or startup files
- data files

Furthermore, we can differentiate files based on whether they belong to the operating system itself (kernel, libraries, boot time executables, etc.) or whether they reside in user space (service programs, libraries, data files, etc.).

When considering file accesses, it seems reasonable to assume that many will be accessed on a frequent basis. This is especially true for certain executable files and data files that are part of the users' routine activities. Other files, such as startup files, will be accessed on a regular, yet much less frequent basis. Finally, there might be files on a system that rarely will or should be accessed. Such files could be backups, storage files, or highly sensitive information. In regard to the modification and creation of files it seems reasonable to assume that executable files as well as configuration and startup files do not (or at least should not) change much once they are in place.

The exact strategy for logging will be different for different systems and they depend upon and should reflect the policies in place for the system. There exists software such as Tripwire [24] that monitors such activity. However, it is not part of the file system, which means that it may not be present on all systems and that its logs may be compromised.

A file system that keeps track of file manipulation activity should be able to support varying degrees of forensic metadata generation as determined by the system's policy. This can significantly reduce the space requirements for the metadata.

6.2 Space reductions

Some of the desired information about a file seldom changes over the course of a user session. The user, origin, and agent information are unlikely to change with every single file access. It is therefore unnecessary to store the information with every operation. Instead it can be recorded as information describing the current context of a session and be referenced by the file metadata. If the context of the session changes, the new context is referenced from there on.

Another space reduction technique could be to group together similar operations that affect only themselves. For instance, successive modifications to a file could result in only one entry being recorded that reflects the change from the initial to the final state of the file. This can only be done, however, if no accesses to other files or changes in the session context occur in between the modifications. Successive read operations may be similarly grouped.

7 Conclusions

In this paper we have analyzed the role file system metadata plays or can play in forensic investigations. Of the questions who did what, where, when, how and why, only a few can be answered from the information collected by today's computing systems. This is in part because of space constraints, but also, as we have discussed, because some of the desired information is impossible to obtain on systems that run arbitrary programs.

Forensics and security were not design objectives for the most commonly used file systems. Some of our desired information could be easily obtained by, for example, recording more information on one-time events such as the creation of a file. The "create" timestamp, the user who created a file, and the user agent path could be recorded in a fixed amount of space. Other information such as detailed file modification or access information are unbounded in their space requirements and therefore recording them might not be suitable in every situation. Moreover, information such as user influence or location of file operations are generally undecidable, which means that the information is not recorded, heuristics must be used, or only verified programs are allowed to be executed on the system.

Not every system is suited to collect all of the desired information discussed in this paper. For a typical home computer none of the extra information may be worth the space requirements or restrictions that would result from recording it. However, when factors such as due diligence, protecting critical information, or being able to quickly determine what happened on a computing system are important, all of the extra information discussed in this paper may play an important role. Today's systems do not offer the ability to record much of the desired information even if one wanted to record it.

Some of the information that we classified outside Category 1 may immediately be recorded by existing systems. A file creation time that cannot be modified anymore should be present on any file system. Recording the user id of the process performing file creation, access, or modification is also a simple inexpensive addition of more valuable data. In general, however, it will depend on the kind of system as to which of the information we have discussed in this paper should actually be recorded and how much of it. Recording everything we have mentioned on every system is not realistic. However, policies in some organizations may require recording a large portion of it. These may range from high-security computing systems, where even the access of certain files should be documented in its most complete form (who, where, when, how?), to home computers where maybe only the question of where certain files came from matters.

In the future of file system design, forensics and security will play a more important role. In this paper we have laid out what kind of information is desirable but we do not offer explicit solutions on how to implement obtaining and storing it. This is part of future research in the field of digital forensics. Nor do we mandate what kind and how much information should be recorded. This will depend on individual systems and the requirements they have in regard to forensics. We do, however, hold the opinion that if desired, it should be able to record such information.

References

[1] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., April 1980.

[2] Florian Buchholz. *Pervasive Binding of Labels to System Processes (working title)*. PhD thesis, Purdue University, West Lafayette, IN, (in preparation).

[3] Florian Buchholz and Clay Shields. Providing process origin information to aid in network traceback. In *Proceedings of the 2002 USENIX Annual Technical Conference*, Monterey, CA, July 2002. CERIAS TR 2002-22.

[4] Florian Buchholz and Clay Shields. Providing process origin information to aid in computer forensic investigations. *Journal of Computer Security*, 12(5):753–776, September 2004.

[5] Michael A. Caloyannides. *Computer Forensics and Privacy*. Artech House, Norwood, MA, 2001.

[6] Rémy Card, Theodore Ts'o, and Stephen Tweedie. Design and implementation of the second extended filesystem. In *In Frank B. Brokken et al, editor, Proceedings of the First Dutch Inter national Symposium on Linux*, 1994.

[7] B. Carrier and C. Shields. A Recursive Session Token Protocol for use in Computer Forensics and TCP Traceback. In *Proceedings of the IEEE Infocomm 2002*, June 2002.

[8] Brian Carrier. Sleuthkit and autopsy forensic browser. `http://www.sleuthkit.org`.

[9] Brian D. Carrier and Eugene H. Spafford. Defining event reconstruction of digital crime scenes. *Journal of Forensic Sciences*, 49(6), 11 2004. CERIAS TR 2004-37.

[10] Eoghan Casey, editor. *Handbook of Computer Crime Investigation.* Academic Press, San Diego, CA, 2002.

[11] Eoghan Casey. *Digital Evidence and Computer Crime.* Academic Press, San Diego, CA, second edition, 2004.

[12] Franklin Clark and Ken Diliberto. *Investigating Computer Crime.* CRC Press, Boca Raton, FL, 1996.

[13] R.C. Daley and P. G. Neumann. A general-purpose file system for secondary storage. In *Fall Joint Computer Conference*, 1965.

[14] Dorothy E. Denning and Peter F. MacDoran. Location-based authentication: grounding cyberspace for better security. In *Internet besieged: countering cyberspace scofflaws*, pages 167–174. ACM Press/Addison-Wesley Publishing Co., 1998.

[15] Encase forensic software. `http://www.guidancesoftware.com`.

[16] Dan Farmer and Wietse Venema. The coroner's toolkit. `http://www.porcupine.org`.

[17] J. Chapman Flack and Mikhail Atallah. A toolkit for modeling and compressing audit data. Technical report, COAST, Purdue University, 1998. COAST TR 98-20.

[18] The forensic toolit. `http://www.accessdata.com/Product04_Overview.htm`.

[19] S. Garfinkel, G. Spafford, and A. Schwartz. *Practical Unix and Internet Security.* O'Reilly, third edition, 2003.

[20] R. Hagmann. Reimplementing the cedar file system using logging and group commit. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 155–162. ACM Press, 1987.

[21] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX). `http://standards.ieee.org/catalog/olis/arch_posix.html`.

[22] Institute for Security Technology Studies. Law Enforcement Tools and Technologies for Investigating Cyber Attacks: A National Needs Assessment. Technical report, Dartmouth College, 2002.

[23] Institute for Security Technology Studies. Law Enforcement Tools and Technologies for Investigating Cyber Attacks: Gap Analysis Report. Technical report, Dartmouth College, February 2004.

[24] Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[25] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 223–236. ACM Press, 2003.

[26] Warren G. Kruse II and Jay G. Heiser. *Computer Forensics: Incident Response Essentials.* Addsion-Wesley, Boston, MA, 2002.

[27] Benjamin A. Kuperman. *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources.* PhD thesis, Purdue University, West Lafayette, IN, 08 2004. CERIAS TR 2004-26.

[28] Paul J. Leach and Rich Salz. Uuids and guids. `http://www.webdav.org/specs/draft-leach-uuids-guids-01.txt`.

[29] Microsoft Corporation. FAT: General Overview of On-disk Format. `http://www.microsoft.com/hwdev/download/hardware/fatgen103.pdf`, 1999.

[30] G. Mohay, A. Anderson, B. Collie, O. De Vel, and R. McKemmish. *Computer and Intrusion Forensics*. Artech House, Norwood, MA, 2003.

[31] Daniel L. Murphy. A virtual memory distributed file system, 1989.

[32] The National Insistute of Justice. Electronic crime needs assessment for state and local law enforcement. `http://www.ojp.usdoj.gov/nij/pubs-sum/186276.htm`, April 2001.

[33] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[34] D.M. Ritchie and K. Thompson. The unix time-sharing system. In *Fourth ACM Symposium on Operating System Principles*, Yorktown Heights, New York, October 1973.

[35] M.K. Rogers and K. Seigfried. The future of computer forensics: a needs analysis survey. *Computers & Security*, 26, 2004.

[36] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir. Deciding when to forget in the elephant file system. In *Symposium on Operating Systems Principles*, pages 110–123, 1999.

[37] W.R. Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley, Reading, MA, 1993.

[38] Sun Microsystems. Sunshield basic security module guide. `http://docs.sun.com/db/doc/802-5757`.

[39] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc. Ser. 2*, 42:230–265, 1937. Reprinted in The Undecidable (Ed. M. David). Hewlett, NY: Raven Press, 1965.

[40] A. M. Turing. Correction to: On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc. Ser. 2*, 43:544–546, 1938.

[41] Unix System Manual Pages. Finding files: find(1).

[42] Wietse Venema. Tcp wrapper, a tool for network monitoring, access control, and for setting up booby traps. In *Proc. 1992 USENIX Security Symposium*, September 1992.

[43] Edward Wilding. *Computer Evidence: A Forensics Investigations Handbook*. Sweet & Maxwell, London, 1997.