Who Dun It: A Study of MAC Time Update on NTFS

Dae Glendowne[1], Christopher Ivancic[2], and David Dampier[3]

Abstract—When gathering evidence from a system an investigator often relies on timestamps in order to construct a timeline of events that occurred on that system. These timestamps contain the modified, created, and accessed (MAC) times for a given file or directory. These MAC times, specifically the accessed time, can be changed by a variety of processes. This needs to be accounted for when analyzing digital evidence. In this paper we have performed an initial analysis of some processes that commonly take place on the Windows 7 operating systems using the NTFS file system and how they affect the MAC times of files. We discuss this analysis as well as some of the challenges faced with determining what exactly causes the updated MAC times to be stored on a physical disk.[1]

Keywords- File System, NTFS, Accessed Time, MAC

1. INTRODUCTION

Digital forensics deals with the recovery and interpretation of evidence from a digital crime scene. A key part of this process is event reconstruction. Event reconstruction examines the evidence to determine why is has certain characteristics. [2]

The Modified, Accessed, and Created (MAC) times are significant forensic artifacts that can be leveraged by the examiner to help perform event reconstruction of a system. These timestamps tell when the file was created, when the file was last modified, and when the file was last accessed. This information, when combined with other temporal cues, can tell a story about the system itself. While each of these times is important, we consider the last accessed time to be the most interesting due to the manner in which it is updated. We believe it is important that the investigator be aware of how timestamps are updated as well as which user actions and processes in Windows 7 can cause these updates in order to draw more accurate conclusions from the evidence presented.

The MAC times of a file can be updated in a variety of ways by numerous processes, user actions, or malicious tampering. Of these, the last accessed time is the most volatile. Many processes can affect it, sometimes without direct interaction or the realization of a user. As a system is being examined, actions and processes that may have affected timestamps needs to be taken into account.

This paper is a product of ongoing research with the National Forensics Training Center (NFTC). The goal of the NFTC is to provide instruction to law enforcement of modern and relevant forensics technique. To continue providing education and training to the law enforcement we conduct research in techniques and tool development in digital forensics. We believe this work will aid law enforcement in more accurate event reconstruction in the Windows 7 environment.

2. RELATED WORK

While timestamp analysis is not new in the area of digital forensics, there has not been significant research into just how MAC times are formed. What exactly is an access of a file and when does it occur? When are these accesses actually recorded in a file's attributes on the NTFS file system? These are the types of questions that began this research and the ones that we attempt to answer in this paper.

Chow et. al. did a study to determine a set of rules they could apply based on the observed MAC times for a file or group of files that would indicate a likely cause for those MAC times. The focus for their paper was on

[1] Mississippi State University, 665 George Perry Street, Butler Hall, Mississippi State, MS 39762, dglendowne@gmail.com
[2] Mississippi State University, 665 George Perry Street, Butler Hall, Mississippi State, MS 39762, cpi4@msstate.edu
[3] Mississippi State University, 665 George Perry Street, Butler Hall, Mississippi State, MS 39762, dampier@cse.msstate.edu

the behavioral characteristics of MAC times in order to aid in event reconstruction. [4] This paper will focus on the why and how of MAC time updates, particularly the last accessed time value.

3. NTFS

The New Technology File System (NTFS) is the default file system for all current versions of the Windows operating system (XP, Vista, Server, and 7). In NTFS, everything is a file including the file system administrative data. The primary data structure in NTFS is the Master File Table (MFT) and every file will have at least one entry in the MFT. [1]

The MFT holds information about the files and directories in MFT entries. These MFT entries store attributes and an attribute is a data structure containing a specific type of information such as a file's filename or security information for that file. In Windows interactions with the NTFS file system take the form of reading and writing attributes for a given file. An example of an attribute is the $DATA attribute which is common to every file in the file system. This attribute stores the actual content of a file such as the text for this paper. [1]

The $STANDARD_INFORMATION attribute contains the timestamp information for that file. It is from this attribute that Windows determines the MAC times for a file that are displayed to the user when they view the properties of a file. There is also a $FILE_NAME attribute that contains the MAC time information as it relates to the filename for a given file. [1] This attribute is used by directories to obtain information about the files that are contained within that directory.

4. WINDOWS I/O ARCHITECTURE

In our efforts to better understand what causes updates to MAC times that are written to disk, we peeked into the architecture of Windows I/O system. This section contains a brief overview of that system.

The I/O system in Windows consists of a group of components that manage the interactions between hardware devices and applications. At the center of the I/O system is the I/O manager. The I/O manager provides a framework with which I/O requests can be passed to device drivers from applications as well as other drivers. These requests take the form of I/O request packets (IRPs). An IRP is a data structure constructed by the I/O manager that contains all the relevant information for a given I/O operation such as the file to be accessed and the type of access desired. [5]

The components that make up the I/O manager that we are interested in are the NTFS driver, volume manager, and the disk driver. These components work closely with the log file service, cache manager, and memory manager to implement efficient I/O processing.

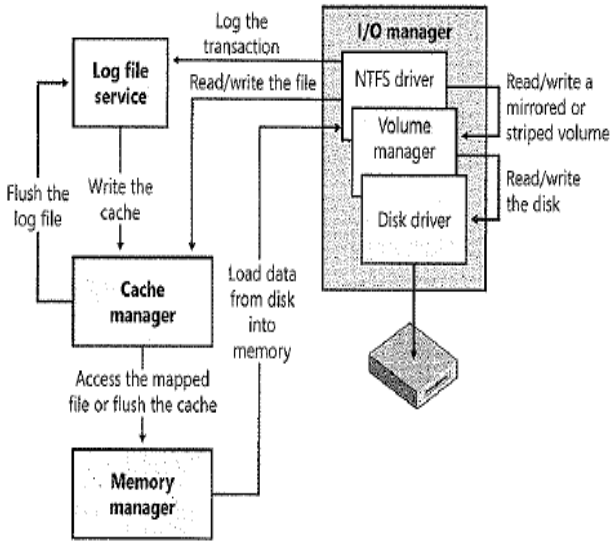


Figure 1: NTFS and Related Components

NTFS Driver, Volume Manager, Disk Driver - These three drivers work together to enable the reading and writing of files from the disk. When user applications and system processes need access to a file they will call the NTFS driver to locate it. The NTFS driver sends an IRP to a device object that represents a volume. The volume manager will then send an IRP (via the I/O manager) to the disk driver containing the volume reference. The disk manager will locate the file and the contents that are required will be loaded into memory.

Log File Service - The log file service (LFS) is composed of several functions that exist inside the NTFS driver. These functions are used to record all metadata changes that occur in the file system for the purpose of recovery. Once the LFS has written the log file to disk, the cache manager will flush the actual metadata changes to disk.[5]

Memory Manager - Part of the memory manager's responsibilities include mapping accessed files from disk into memory. The memory manager uses a structure called a section object, also called a file mapping object, to map virtual addresses to these opened files. The portion of the file that is mapped into memory (the view) may be utilized by an application to read or write the data for that file.

Once a file is mapped into a process's virtual address space in memory, the program can access the file view without performing extra disk I/O. If the process needs to access a portion of the file that is not mapped into memory then the memory manager uses it's paging mechanism to load that view into memory. This is the same way the memory manager writes changes back to disk. When a process writes to a file view, the memory manager just pages the written pages back to disk. [5]

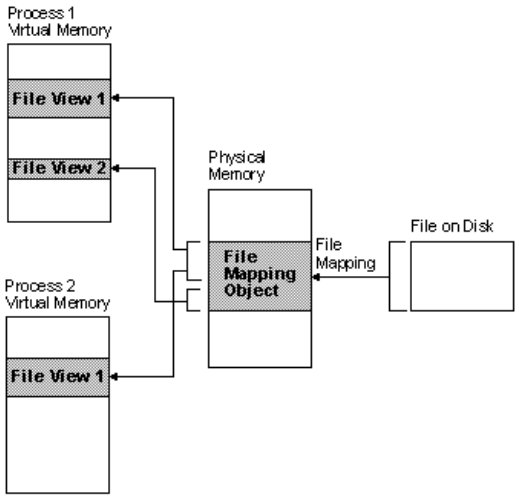


Figure 2: Relationship between file on disk, file mapping object, and file view. [8]

Cache Manager - The cache manager helps the memory manager and NTFS driver maintain efficiency for I/O operations.

5. TESTING METHODS

Windows, for the sake of efficiency, does not immediately write attribute modifications to disk. To test this we took the approach of not just looking at the timestamps through explorer in the properties. We wanted to see actual system calls and determine what is being updated. This section describes our process for exploring how processes can affect the MAC times of a file.

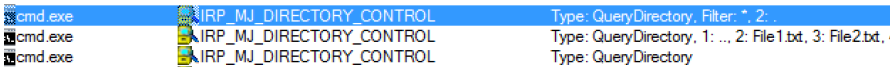
The first challenge in looking for updated MAC times was simply to view the MAC times for a file without altering them by that same action. We knew from previous experience that simply right clicking a file and checking the properties can cause a last accessed time update. Our primary method for checking this was to use the dir command from the command line with the *dir /t:a* parameter. This displays all the files in a directory along with their last accessed time.

A directory in NTFS contains an $INDEX_ROOT and $INDEX_ALLOCATION attribute. Each index entry will contain a $FILE_NAME attribute for a given file in that directory. Since the $FILE_NAME attribute stores temporal information about the file, it is possible to view a file's timestamps without actually touching the file itself. According to Carrier, "Windows updates the temporal and size information so that it is accurate." [1] This

method does update the accessed time for the directory entry itself, but this does not affect out results since we are only concerned with the contained files.

Another tool we made extensive use of was Process Monitor. Process Monitor is a system activity monitoring utility and an example of a passive filter driver. It monitors the flow of IRPs between various applications and the NTFS driver. IRPs that pass through Process Monitor have their information recorded (such as target file, read lengths, desired access, etc..) [5]

Using process monitor we were able to obtain a more detailed view of the file system stack in action. It allowed us to see the processes that were interacting with various files, the IRPs that are being sent to the NTFS driver, and stack traces for each event that occurred. These stack traces allow us to peer into the I/O subsystem to see which functions were being called in both user and kernel space.



Process Monitor output while running dir /t:a

The above image shows output from Process Monitor when we ran the *dir /t:a* from the command line on a folder. The IRP that was sent to the NTFS driver was the IRP_MJ_DIRECTORY_CONTROL which queries a directory about the files it contains.

Throughout our use of Process Monitor we noticed several operations that revealed useful information regarding file MAC times. Two of these were the IRP_MJ_QUERY_INFORMATION and IRP_MJ_SET_INFORMATION operations. These IRPs are sent when certain routines, such as *ZwQueryInformationFile* and *ZwSetInformationFile*, are called. These routines contain a FILE_BASIC_INFORMATION data structure which holds various time values. [6] The structure looks like this

```
typedef struct _FILE_BASIC_INFORMATION {
 LARGE_INTEGER CreationTime;
 LARGE_INTEGER LastAccessTime;
 LARGE_INTEGER LastWriteTime;
 LARGE_INTEGER ChangeTime;
 ULONG FileAttributes;
} FILE_BASIC_INFORMATION, *PFILE_BASIC_INFORMATION;
```

In the case of this structure being used as part of a set file information routine, the caller can specify a valid time, a 0, or -1. We did not observe a large number of the IRP_MJ_SET_INFORMATION in Process Monitor, but when we did the time values in the FILE_BASIC_INFORMATION structure contained either 0 or -1. "A value of zero for any of the XxxTime members of the FILE_BASIC_INFORMATION structure, the *ZwSetInformationFile* function keeps a file's current setting for that time." [6] A value of -1 indicates that the file system should not update one or more of these members for any I/O operations that are performed on the caller's file handle. [6]

6. EXPERIMENT

In this section we attempt to perform controlled events in which we would watch to see if any time updates are called by the system. Our main control scheme was to manipulate a file and watch Process Monitor for what was being passed in memory then confirm any updates by checking the directory attributes via the command line. While we do check the modification and we mostly check to see when an access time is updated and what actions happen that might cause the access update. The system clock was not physically manipulated and all time was allowed to run naturally between each check. We chose a few common applications to check when opening the files:

- Notepad
- Notepad++
- Microsoft Word
- WordPad
- Windows Media Player
- Windows Photo Viewer

- VLC
- WinRAR

These applications were run while viewing Process Monitor to see what packets are being sent by different applications. In addition to using applications we tested opens, moves, and copies from command line and hotkey shortcuts to see if any different calls are made.

This section also describes how we tested processes which could be running that does not involve the user of the system directly manipulating the files. We tested out two common processes running on computers, Anti-Virus software and file sharing software. Section 6.2 and 6.3 describes the process used and the results we obtained while running the tests.

For these experiments to work we need to set windows to accept changes to last access time. To clarify, in Vista/Server08 as well as Windows 7 there is a registry value that is enabled by default called NtfsDisableLastAccessUpdate that is stored in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem. We had to set this to 0 in order for any last accessed timestamps to be updated in Windows 7. Table 1 below shows if the applications changed the access time of the files. In the table we see that notepad does not affect it and is explained later in the section.

Win32 API Call	Access Time Update
Notepad	No
Notepad++	Yes
Microsoft Word	Yes
WordPad	Yes
Windows Media Player	Yes
Windows Photo Viewer	Yes
VLC	Yes
WinRAR	Yes

Table1: Applications and Access Times

6.1 Copying and Opening Files

Copying files and viewing the files are two common actions taken on a computer. This section describes how we looked to see the processes involved with opening and copying the files on the computer. Initially we manually performed these actions then we wrote scripts running that ran several days mimicking the process of copying and opening the folders to give us a large data set to look at.

6.1.1 Manual Manipulation

For this experiment we were physically opening and moving files while checking information passed through Process Monitor and checking the time of the files in the directory through the command prompt. This allowed us to see discrepancies between the last accessed time in memory and on disk.

6.1.1.1 Copying and Modifying Files

We noticed that when a file is copied that the modified time of the new file is before the creation time. To check this we looked at the time stamps of a file or group of files through the command line and then manually copied the files over. The new files' times were checked via command line. We verified that the new files had new creation times yet still had the same modified times as the original files. We discovered that the last accessed time of the new files and of the original were changed as well as the creation time of the new file. The last modified times remained the same on both and were not updated to the time of copy.

We also wanted to check to see if modifying the file immediately updated the last accessed time immediately or if it followed the 60 min rule [6,9]. We opened a few files and made small changes to them, saved the files then closed them. By doing this we discovered that the modified times and last accessed times of the files were

immediately updated. We made changes to files in under an hour and after an hour (compared to original time stamp) and saw that changes to the times were both made regardless of how much time had passed.

6.1.1.2 Opening Files

Modifying and copying files always resulted in time updates, specifically accessed times. This is consistent with the idea that anytime data is made or changed the system regards it as an important update and writes the new times out to disk. Opening a file may not always need to have a disk I/O operation occur. NTFS has policies in place when determining when to update a last accessed time stamp. If a file is opened and not modified then the OS will not update the accessed time unless 60 minutes have passed since the last access time [1, 6, 9].

For this experiment we would open a file in various ways while watching the packets being sent through Process Monitor then checking the timestamps from the command prompt. Opening actions included: double clicking a file, clicking just once in window (which caused, right clicking a file and opening with program, and right clicking a file to check the properties.

6.1.2 Automatic Manipulation

For this experiment our goal was to generate more data similar to the previous section. The goal was to check the time stamps of several files after an open or copy. This experiment was to gather more timestamp data to add to the previous section. For this experiment we only were after the last accessed time as this timestamp was the most inconsistent of all the others when running simple hand checks and manual operations.

We wanted to simulate accessing files over an extended period of time (8+ hours). A python script was written that would open a random file from a folder on the desktop. It would then open the file using a random appropriate program selected from a list to open the file. This script would perform this operation every 15 to 30 minutes (time was selected randomly between them each run). This would show operations over a day as if a user was accessing these files. This simulation would give us an idea of how actions can be performed on files over time rather then when we wanted to test it.

In addition to opening the files the scripts would also copy the files to simulate a copy/move of files. For the sake of making scripting easier, the files were separated into three different folders: Open Text Files, Open Media Files, and Copy Files.

The script started each iteration by writing the current system time to a log file then piping the results of the *dir /t:a* command of the directory being searched, to the log file. The script then writes the operation performed and target file to the log file followed by a second *dir /t:a* command.

6.1.3 Results

Once the script had run for the full day the resulting log files were parsed for files that had experienced an accessed time update. The parser then compared the old timestamps to the new ones. The time difference for an accessed time update was calculated in order to identify any file's being updated in a period less than 60 minutes from its previous accessed time update.

We found similar results for the manual manipulation of the files. All copy operations would update the last accessed time after 60 minutes the same as opening a file without modification.

When it came to opening the files, when the script opened media files the last accessed time would be updated if it had been at least 60 minutes since the last time update.

Things took an interesting turn when the script opened text files. It would not update any times before 60 minutes had happened, however inconsistent results were logged as to when a file's last accessed time was updated. For the most part if a file was opened and it had been at least 60 minutes it would be updated; however, there were some files that would not update even as far as 3 hours after the last time the file's last accessed time was updated.

While looking into why we were getting these inconsistent results, we discovered that this was only happening when notepad was used as the corresponding program to open up these files. Process Monitor logs showed that when a file was updated it had an IRP_READ associated with it. This accounted for why some applications would not change access time while others did.

Two other actions taken were creating new files and moving files. When a file is created the CMA times are created and so are updated from a null value to the current time. Moving a file did not register any IRP to read the file and did not result in CMA times being updated. Table 2 below describes our results of common activities performed on files and what times are updated.

Action	Time Altered
Copy(original)	A
Copy(new)	C, A
Modifying	C, M, A
Opening	A
Creating	C, M, A
Moving	none

Table 2: Times Affected by Common Actions

6.2 Bit Torrent

File sharing is another common action people will find running on computers. Even if a person is not opening the file him or herself, then can simply sharing the files cause an access time update? For this experiment we used BitTorrent as our file sharing method and attempted to see if sharing the files would update the accessed times of the files.

6.2.1 Setup

For this experiment we had a folder with several pdf files stored in it. Two computers A and B were used and installed a torrent client. In this case both used BitTorrent client. Computer A created a .torrent file out of the directory containing the pdfs. The torrent file was sent to computer B and was used to download the directory from A to B. During the course of the sending A had process monitor running as with all of our experiments monitoring the calls to the files.

6.2.2 Results

We discovered two things about using a torrent to share files. The first thing is that the creation of the torrent itself caused access time updates. When the file was created it had to read all the files in the directory and was not making the appropriate IRP calls to insure that the timestamps would not be modified.

The second thing was that seeding the files also caused an update. The torrent file itself was run to allow for the other clients to download the documents and so was run as a seed. When it ran it updated the accessed timestamp. That was to be expected since we were physically telling the torrent to run; however, while the download was being run, the client had to read the files to get the information causing the last accessed time to be updated.

6.3 Anti-Virus Software

Many machines run some form of anti-virus (AV) software solution. AV solutions will typically be set up to periodically scan folders and files. Here we tested to see if those scans would cause updates to any of the MAC times of a file.

6.3.1 Setup

For this experiment we used 3 anti-virus (AV) software packages:

- Sophos Endpoint Security and Control v9.5
- AVG-Free Edition v10.0.1375
- Spybot Search and Destroy v1.6.2
- Norton 360 v5.0.0.125
- Malwarebytes Anti-Malware v1.60.1.1000
- Avast Anti-Virus v6.0
- Kaspersky v12.0.0.449

We scanned monitored directories with each AV package. The folder was an amalgamation of files from the pc copied into this folder to give a variety of files to scan.

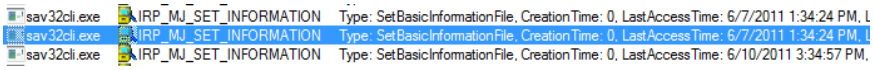


Figure 4: Process Monitor showing Sophos setting the Last Access Time

6.3.2 Results

We discovered that Kaspersky, Notron, AVG, and Spybot both changed the last accessed time of the files in the folder if it had been more than 60 minutes. Avast, Malwarebytes, and Sophos did not update the last accessed time. Sophos did reveal an interesting trait**.** The timestamps remained the same as the times before Sophos ran. Looking at process monitor (Figure 4) revealed IRP packets from Sophos that are used for changing times.

Sophos retrieves the MAC times of a file by creating an IRP of type IRP_MJ_QUERY_INFORMATION and storing the results. Once it had opened the file and scanned it, another IRP was created of type IRP_MJ_SET_INFORMATION and using the pre scan values. Sophos did actually cause an update, but it just overwrote it, masking the access.

Table 3 below shows the results we gathered on several anti-virus software.

AV Software	Accessed Time Changed
Sophos Endpoint Security and Control v9.5	No
AVG-Free Edition v10.0.1375	Yes
Spybot Search and Destroy v1.6.2	Yes
Norton 360 v5.0.0.125	Yes
Malwarebytes Anti-Malware v1.60.1.1000	No
Avast Anti-Virus v6.0	No
Kaspersky v12.0.0.449	Yes

Table 3: Anti-Virus Access Times

6.4 Win32 API Calls

Using Python 2.7, we made calls to some of the Win32 API functions to see how they affected files. This granted us a more concise picture of what exactly an access consists of. We consider file access to be the reading or writing of file attributes, specifically the data attribute.

We created a script that opened a file via *CreateFile* with *dwFlagsAndAttributes* set to FILE_ATTRIBUTE_NORMAL and then close that file. We ran this script over time and did not note any updated last accessed times for the corresponding files. However, adding a single *ReadFile* command instantly caused an access time update. *WriteFile* did cause modify and access time updates to the files. *CreateFile* will only directly modify the MAC times when it is actually used to create a file. Typically *CreateFile* is used to get a handle to a file that other functions or processes can use.

We ran a similar script for several hours on certain files to note the behavior. The script would open a file, read some of the data, close the file, then reopen it and repeat the process. We ran this for several hours with Process Monitor watching the specified directories. This showed us that the 60 minute rule does indeed hold as the updates always appeared within 60 minutes and 1 second of the last accessed time update. Times were checked with the *dir /t:a* command.

We also noticed a single System process that would appear with an IRP_MJ_WRITE to the parent directory of the files we were affecting. We know that at some point the temporal information in the $FILE_NAME attribute gets updated by the system. We believe this System process may be what is actually doing that. It only spawned once every 60 minutes and it would be shortly after the time an accessed time update should occur.

Win32 API Call	CMA Times Updated
CreateFile	none
ReadFile	A
WriteFile	M,A

Table 4: Win32 API Calls

7. ANALYSIS

Through each of the experiments we were able to generate a large number of operations of files. From the conglomeration of results we were able to draw numerous conclusions. In this section we look at the how the actions in the experiments related to the CMA times of files.

Create - Every time a new file is made then the create time is set. This time lets the user know when the file first was made. When copying a file you will have two files with the same information. The new file will have a new create time from the older file. Every action we performed on the file would not change the create time.

Modify - The modified time lets the user know when the information of the file was last changed. When a file is opened and closed with no changes made then the modify time is not updated. Opening a file then making some change to the data and saving will update the modify time. Unlike the access time, the modify time was being changed as soon as new data was saved. It did not rely on waiting at least 60 minutes to pass before updating. For efficiency the system chooses not to change access times before an hour has passed.[5] Too many access could slow performance with unnecessary I/O calls.[5] Since changing the contents of the file requires the new contents to be written to disk anyway, the modify time can be written at the same time.

The modified time does not change in the event of a copy. As stated above, when a file is copied, the new file will have a different create time then the original. The modify time is not changed. When a file is copied, the new file and the original will have the same modify time. Modify times are only changed when the data of a file is changed.

Accessed - As we have mentioned before, NTFS will not change an access time of the file unless an hour has passed since the last access time update.[1, 5, 6] Our observations verified this and so any access we saw only occurred after an hour had passed.

Copying a file would update the last access time of the original files and would assign the time to the new ones created. Opening a file with applications did change the last access time with the exception for Notepad. With this program we noticed that there were no IRP_READ calls and Notepad would send *SetInformation* calls with -1 and 0 as the values passed to last access time. This would ignore any update Notepad needed to make to this time stamp. This IRP_READ appeared in other applications wanting to read the file and would also cause time updates upon being called.

We tested anti-virus software against a folder to see if AV scans could affect the last access time. We noticed that AVG and Spybot did affect the access times and we saw it directly from the directory command via command line. Sophos at first appeared to have not affect on the last access time of the files. When we looked at the Process Monitor log we noticed that Sophos was sending a *SetInformation* with the last access time (as shown in section 6.3). Sophos was causing the IRP_READ packets to be sent but would immediately reset the access time to hide the fact it had read the files in the folder.

8. CONCLUSIONS

The MAC times of files on a computer are important forensic artifacts that can be used to aid an investigator's evaluation of a system. We believe it is important to understand not only what user applications or system processes can cause these updates to occur, but also what is going on in the lower level of the kernel when they do.

We have tested several processes, particularly in the context of the last accessed time, to determine if and how they affect a file's timestamps. We did this by using a combination of manual testing as a user would, scripting events, and win32 API calls in Python, and observing IRPs and fast I/O through Process Monitor. We believe that an access time update will typically occur when a file is read from or written to by a process. There are times however when this is not always the case. For instance, unless the modified time is being updated as well as the last accessed time, such as when you save a file, the last accessed time will not be updated unless the current last accessed time was at least 60 minutes prior.

The ways in which a process can interact with a file in the NTFS file system on the Windows OS are many and it depends on the application developer's methods as to how that interaction will occur and whether or not it will cause a change in the MAC times of a file. As we've noted, when the Sophos antivirus scan runs against a file, it actually causes an update to the last accessed time, but it actually writes the pre-scan accessed time back out for the file after the scan has completed, presenting the investigator with no obvious sign that an access has occurred. This type of behavior amongst applications and processes should be accounted for during the course of an investigation where MAC times play a role in the reconstruction of events.

REFERENCES

[1] B. Carrier, File System Forensic Analysis, Addison Wesley, Boston, MA, 2005.

[2] B. Carrier, E. Spafford, 'Defining event reconstruction of a digital crime scene', Journal of Forensic Sciences 2004

[3] Gartner Inc., "Gartner Says Windows 7 Will Be Running on 42 Percent of PCs in Use Worldwide By the End of 2011", Gartner Newsroom, http://www.gartner.com/it/page.jsp?id=1762614 (accessed 12 January 2012)

[4] K.P. Chow, Frank Y.W. Law, Michael Y.K. Kwan, and Pierre K.Y. Lai, "The Rules of Time on NTFS File System", Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07), pp71-85, 2007

[5] M. E. Russinovich and D. A. Solomon, Windows Internals Covering Windows Server 2008 and Windows Vista, Microsoft Press, One Microsoft Way, WA, 2009.

[6] Microsoft, "FILE_BASIC_INFORMATION Structure", MSDN Library, http://msdn.microsoft.com/en-us/library/ff545762(v=vs.85).aspx (accessed 10 June 2011)

[7] Microsoft, "File Caching", MSDN Library, http://msdn.microsoft.com/en-us/library/aa364218(v=vs.85).aspx (accessed 10 June 2011)

[8] Microsoft, "File Mapping", MSDN Library, http://msdn.microsoft.com/en-us/library/aa366556(v=vs.85).aspx (accessed 10 June 2011)

[9] Microsoft, " File System Behavior in the Microsoft Windows Environment", Microsoft.com, http://download.microsoft.com/download/4/3/8/43889780-8d45-4b2e-9d3a-c696a890309f/File%20System%20Behavior%20Overview.pdf (accessed 27 February 2011)

[10] Microsoft, "File Times", MSDN Library, http://msdn.microsoft.com/en-us/library/ms724290(v=vs.85).aspx (accessed 10 June 2011)

[11] Microsoft, "IRPs Are Different From Fast I/O", MSDN Library, http://msdn.microsoft.com/en-us/library/ff548576(v=vs.85).aspx (accessed 10 June 2011)

Dae Glendowne
Dae Glendowne is currently enrolled at Mississippi State University pursuing his PhD in Computer Science. He is a Senior Research Assistant at the National Forensics Training Center. His duties include performing research in the areas of digital forensics and computer security as well as conducting digital forensics training for law enforcement officers.

Christopher Ivancic
Christopher Ivancic is currently enrolled at Mississippi State University pursuing his PhD in Computer Science. He is a Senior Research Assistant at the National Forensics Training Center. His responsibilities are to perform research in digital forensics and security techniques as well as training law enforcement in digital forensics.

Dr. David Dampier
Dr. Dampier is currently a professor of Computer Science and Engineering at Mississippi State University. He is the head of the National Forensics Training Center and oversees all training provided by the center. He received his Ph.D in 1994 from the Naval Postgraduate School in Computer Science. He has been awarded the Bagley College of Engineering Academy of Distinguished honors in 2011.