ORPHANS IN THE NTFS WORLD

Dustin Hurlbut

Orphan Files – Definition

An orphan is defined for files just as it is for humans. Orphan files are files that no longer have a parent; the parent being the folder they were in. If a folder is deleted, the files within it are deleted as well but are not orphans. The folder and children files are potentially recoverable with the information intact in the Master File Table ($MFT). To orphan a file, the parent folder is overwritten. At this point, the records in the $MFT are still there as are the pointers from the children files to the parent folder, but they are referencing a parent record in the $MFT that no longer has the correct information. The child has been severed from the parent irrevocably.

$MFT Records 101

How do FTK and FTK Imager know this? How does it detect that the child file is pointing back to a parent folder that is no longer correct? The answers lies in the $MFT records for these files and folders.

Header (Windows 2000 - 48 bytes—Windows XP - 56 bytes)
Standard Information Attribute (SIA) 10 00 00 00 (contains dates/times)
File Name Attribute 30 00 00 00
Other Attributes

Figure 1 - $MFT Record and Attribute Information - 1024 Byte Records

The basic $MFT record for a file or a folder consists of 1024 bytes. Within those 1024 bytes exists a header (48 bytes for Windows 2000 and 56 bytes for Windows XP). There are various attribute areas associated with a particular file or folder. Attributes in the $MFT are not like the bit switched attributes we were used to in DOS such as the hidden, system, or archive bits. $MFT attributes contain information about the file in dedicated areas such as date and time stamps, file name, data runs, security, and other associated information. For the scope of this paper, we will limit our discussion to one attribute relative to orphan files, the File Name Attribute (FNA).

The File Name Attribute begins with a "header" of 30 00 00 00. You can locate it in the $MFT by parsing the previous header and attributes to get to it (See Figure 2). Once located, it, along with the $MFT record header, contain the data to solve this puzzle.

Finding the File Name Attribute

-Start at the beginning of the $MFT record and move in 56 bytes on an XP machine, 48 bytes on a Windows 2000 box.

-You will see the Standard Information Attribute (SIA) header of 10 00 00 00.

-Directly behind this header is a four byte value in little endian that will give you the size of the SIA.

-For Example, if it is 60h that equates to 90d.

-Set your cursor at the beginning of the SIA (10 00 00 00) and highlight 90 bytes over.

-You will then see the header 30 00 00 00 which is the beginning of the FNA.

Figure 2

In the header of any $MFT record for is an entry at offset 16-17d (d in this article represents a decimal number) called the stored sequence value. This number is a two-byte hex number in little endian format. It contains the number of times that

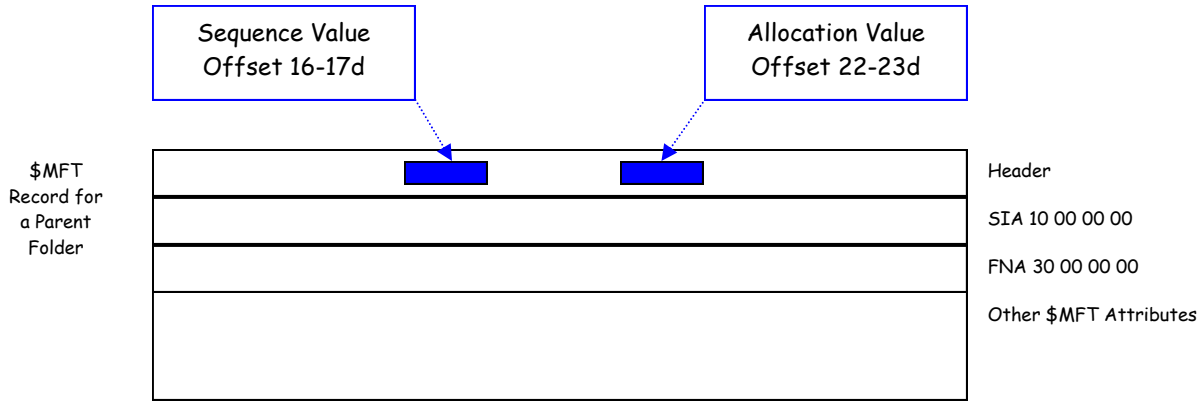

Figure 3 - Location of Sequence and Allocation Values

particular record in the $MFT has been used. When the $MFT record is created, it will carry the number 01 00h. Each time the object of that record is deleted, it will increment by one, but only when that file or folder is deleted. When the record is reallocated, the sequence number will not change. It will increment only when deletion occurs. See Figure 3 for a graphic depiction of where the sequence number is located.

Hex	Binary	File or Folder	Deleted or Allocated
00	00000000	File	Unallocated
01	00000001	File	Allocated
02	00000010	Folder	Unallocated
03	00000011	Folder	Allocated

Figure 4 - Allocation Value Translations

Also in the header of the $MFT record is a two byte value which indicates the status of the of the record. This is located at offset 22d (Figure 3). This little endian value indicates whether the record is occupied by an allocated file or folder, or an unallocated (deleted) file or folder (See Figure 4 for a chart of allocation number designations).

Evaluating Whether a Child is Orphaned

This information resides in the $MFT record for each folder or file. To answer the questions we asked at the beginning of the paper; how does FTK know whether the child belongs to the parent and how does the child keep track of the parent so NTFS knows what is going on? We need the sequence number and the status number.

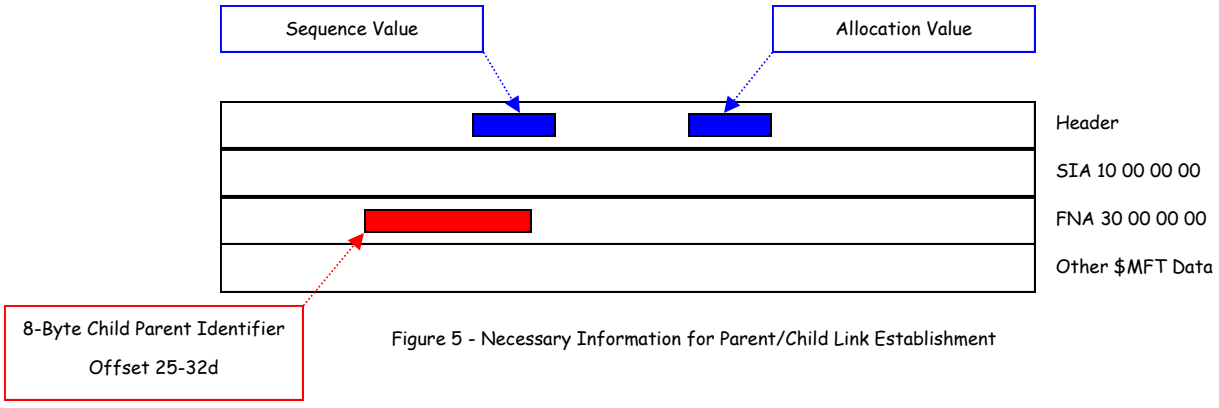

Figure 5 - Necessary Information for Parent/Child Link Establishment

To complete the process, we need one more piece of information. We need to know where the child thinks the parent is. In each $MFT record, there is an eight byte value located 24 bytes from the beginning of the File Name Attribute (See Figure 5). These eight bytes keep track of the parent in a unique way. The ***first six bytes*** are a pointer to the parent's record number in the $MFT. To locate the parent from this number, simply multiply the value found in those six bytes (little endian format) by 1024 (size of $MFT record) and it will give you the offset of the first byte of the parent's record location in the $MFT.

The remaining two bytes are the sequence number (in little endian) of the parent record. If the stored sequence value in the parent is the same as that stored in the child's File Name Attribute, then that child is still linked to the parent and all is well.

If the parent is deleted, the parent's sequence value increments by one. The child still points back to the parent, but now the sequence numbers don't match, the parent is one ahead of the child. This means that the file's parent folder has been deleted, and for that matter, so has the child. The parent's status will be 02h, folder unallocated. FTK will view this as a deleted folder and the children are considered associated files, not orphans. How then do we make them orphans?

When adding a file or folder to the $MFT, it does so by sorting down from the top and finding the next available record. If the parent folder of the children gets reused, the status number changes to either an active file or active folder. At this point, FTK will render the existing deleted children entries as orphans. Their record information is still in the $MFT, but their pointer back to the parent is no longer valid as the parent's record has been reused in favor of a new file or folder with no connection to those files.

Sequence numbers match	Parent Record Allocated	File = Child
Parent sequence number = +1	Parent record unallocated	File = Child (Deleted)
Parent sequence number = +1 or more	Parent record allocated	File = Orphan

Figure 6 - NTFS Orphan Decision Maker

How does FTK know this? It looks at the sequence value in the child's File Name Attribute for the parent. If the parent record sequence value is incremented by one, and the status value of the parent record indicates an allocated file or folder. FTK knows that there is no way the parent can be associated with that child. If the sequence number is one ahead, and the status of the record is unallocated, it knows that the child is still associated with the just deleted parent folder. The table in Figure 6 can be used to chart the status of the parent record.

FTK presents an accurate view of the status of folders and their associated files. In NTFS, orphans can occur when there is still valid data in the $MFT for each file that points back to a parent, even if that file's parent has been deleted and its record reused. FTK does not make an assumption that the data in the parent record is associated with the child, it checks using this procedure we have described. The Diagram in Figure 7 shows the process of comparing the eight bytes of data in the FNA of the child with the data in the parent folder to determine their status.

Practical Applications of NTFS Orphans

The Orphan folder in FTK or FTK Imager represents all the files FTK locates in an NTFS environment that have no legitimate parent folder. To envision how this works you can try this exercise using a thumb drive formatted in NTFS.

First take a newly formatted thumb drive in NTFS and add two folders to it called; Normal Files and Orphaned Files. Next, place two text files in the Normal Files folder (call them NormalFile1.txt and NormalFile2.txt) and three text files in the Orphaned Files folder (call these Orphan1.txt, Orphan2.txt, and Orphan3.txt). Image the thumb drive and call it Image1. Add this image to FTK Imager.

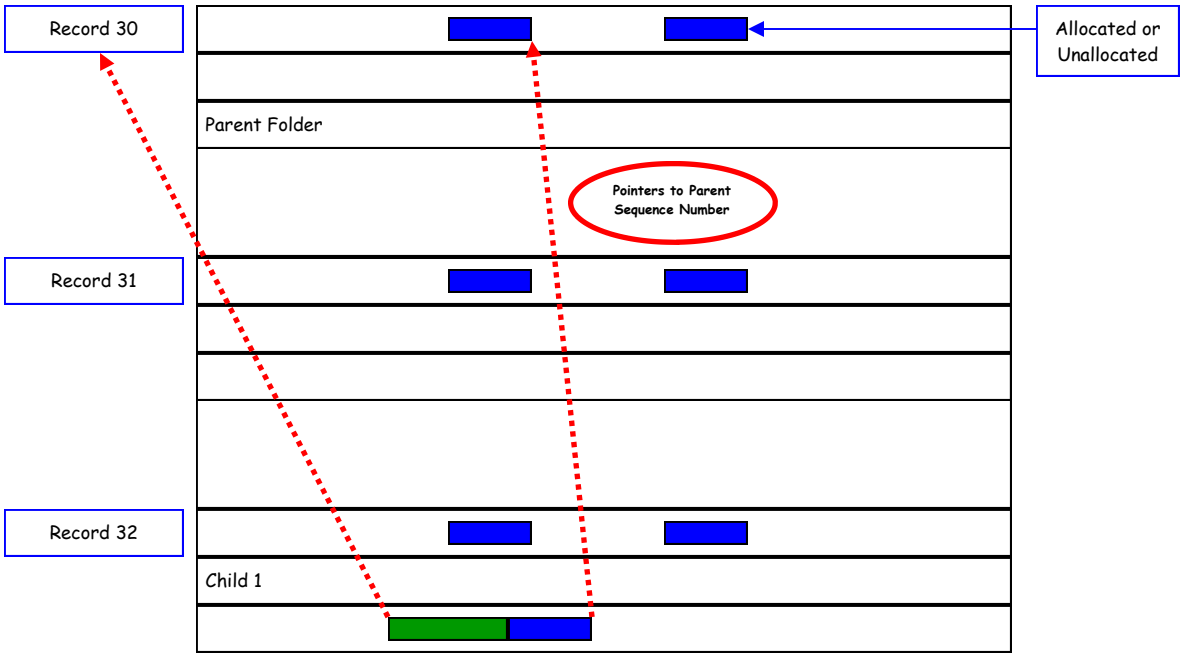

Figure 7 - Pointers in the Child File to the Parent Folder

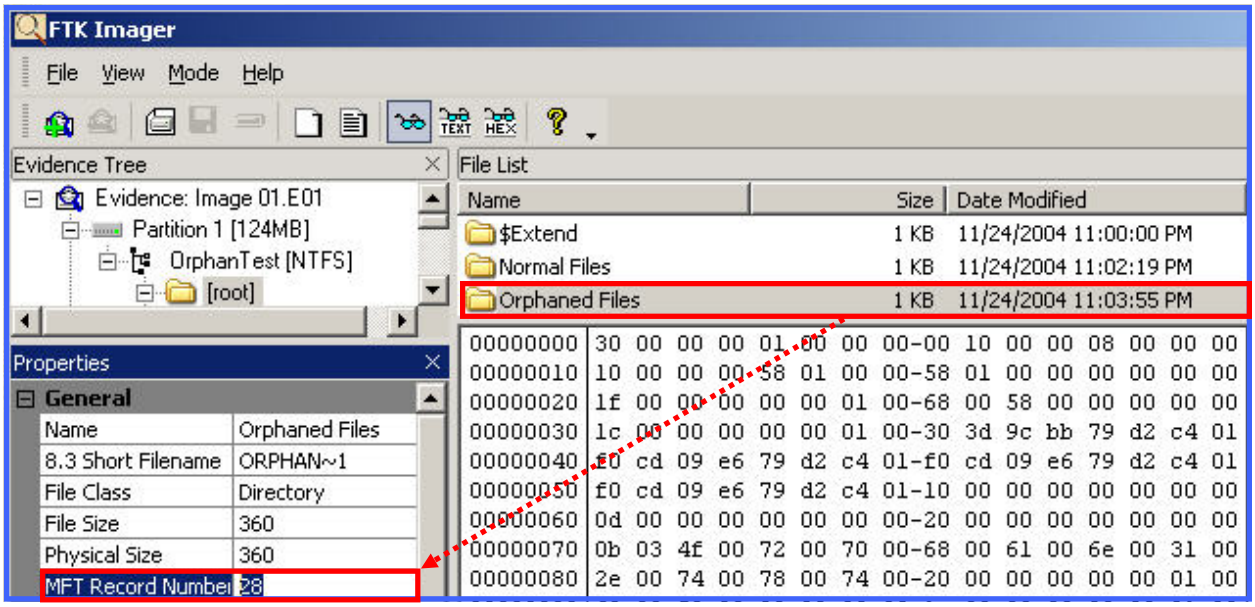

Figure 8 - $MFT Record Number for Orphan File Folder

Click on the drive root and select the Orphaned Files folder. Select the properties tab in the lower left corner and view the $MFT Record number (See Figure 8). In the example, it is #28. Multiply that record number by 1024 (the $MFT record size). In this example, it is 28,672.

Next, open the $MFT file in the Root Directory and right click on the hex viewing area in the lower right pane. Select Go To Offset and the resulting calculated offset of 28,672 from this example. This will place you in the first byte of the $MFT record for the Orphaned File folder.

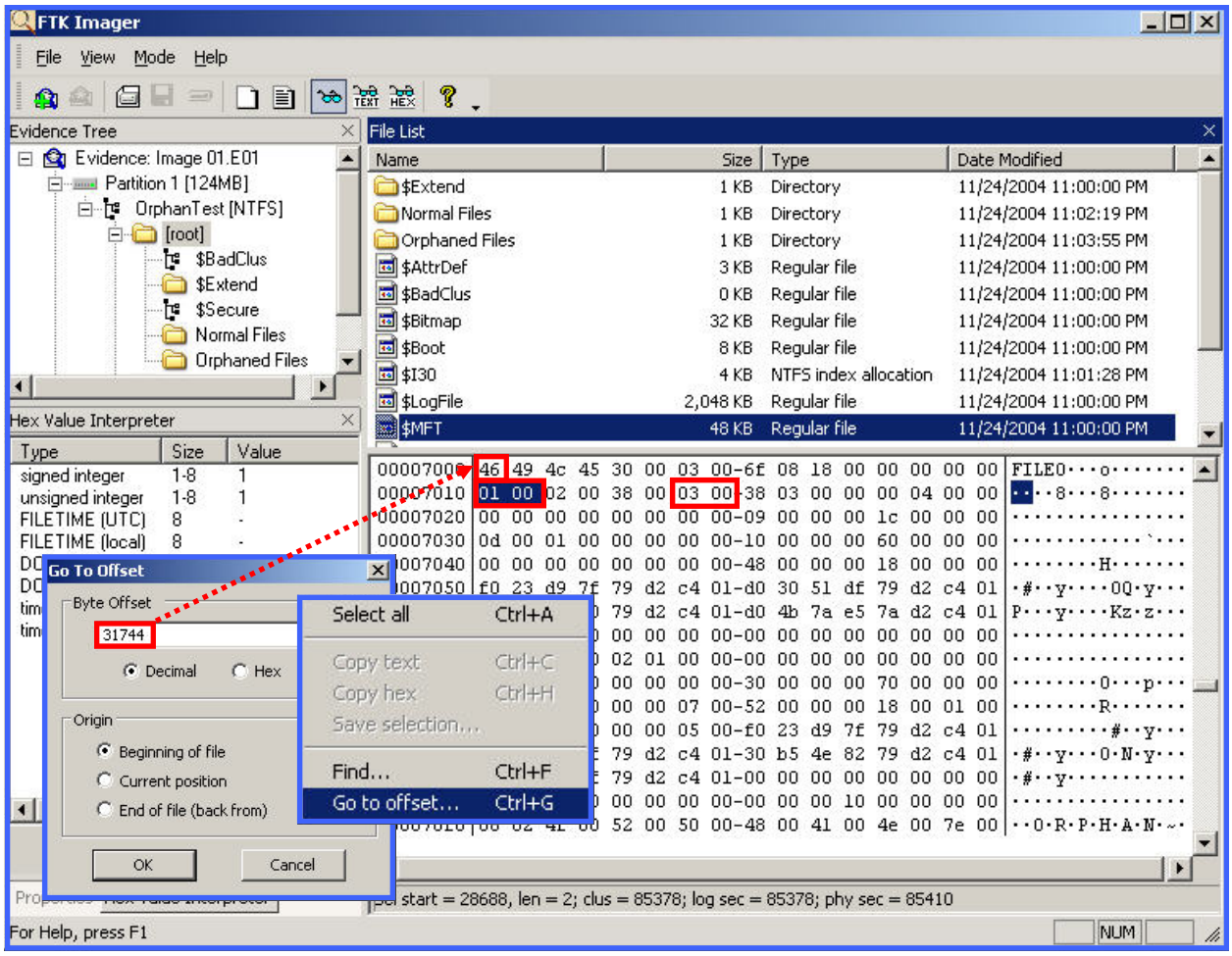

Figure 9 - Sequence and Allocation Values for the Orphaned Files Folder

Highlight the first row of hex numbers. At the beginning of the next row, highlight offsets 16-17d and note the decimal value in the Hex Value Interpreter tab to the lower left (See Figure 9). This is the sequence value and has a value of one in this example as this is the first time the record has been used.

Note offsets 22-23 also shown in Figure 9. This is the allocation value showing whether the folder is deleted or allocated. In this example, it is three, indicating an allocated folder (for allocation number interpretation, see Figure 4).

Next, open the Orphaned Files folder and highlight the first file, Orphan1.txt. Note the $MFT Record number (31 in this example). Multiply 31 x 1024 (31,744) to obtain the $MFT offset for this file. Navigate to the $MFT and right click in the hex area to enter this offset to view the record for this file. Note the sequence value for this file is one, indicating this record has been used only once, and the allocation number is one, indicating it is an allocated file.

We now must find the eight byte pointer to the parent folder's record. Figure 10 outlines the steps to locate the pointer. First, highlight the header of the record for Orphan1.txt. In this example it is a Windows XP system so the header is 56 bytes. On a Windows 2000 machine it will be 48 bytes.

The first four bytes after the header are the Standard Information Attribute (SIA) header. It consists of hex 10 00 00 00 followed immediately by a four byte little endian value that denotes the size of the SIA. We must sweep those four bytes to interpret them in the Hex Value Interpreter. In the example, 60h = 96d. Place your cursor at the beginning of the SIA and sweep 96 bytes in.

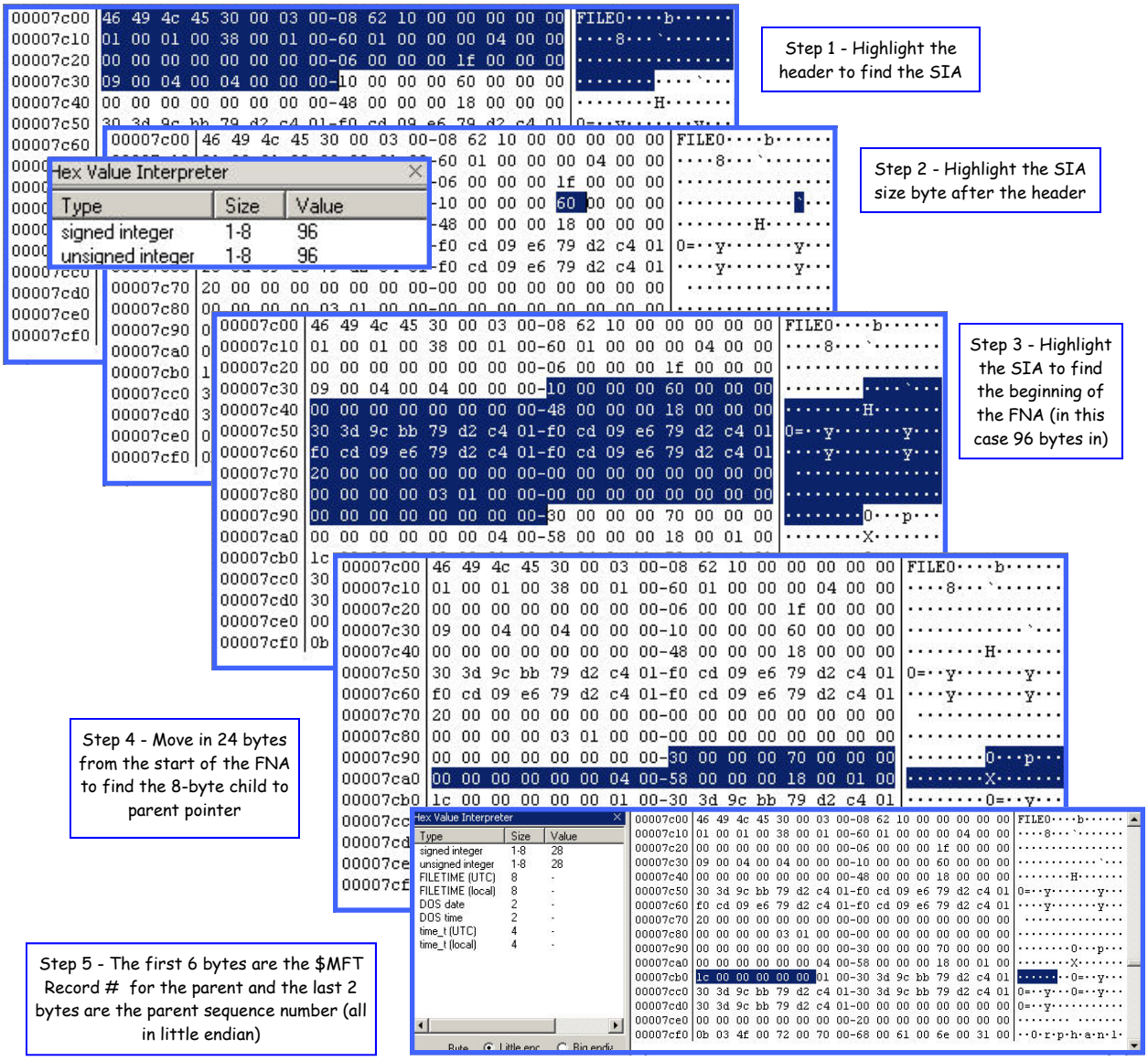

Figure 10 - Steps to find the Sequence and Allocation Pointer from Child to Parent

The next four bytes after the SIA is the File Name Attribute header which is 30 00 00 00h. Place the cursor at the beginning of this attribute, in front of the byte "30h", and sweep in 24 bytes. The next 8 bytes are the child to parent pointer for the file.

The first six bytes of this eight byte value is the $MFT record number for the parent. You can sweep them and read the parent's record number from the Hex Value Interpreter. In our example the six bytes in Little Endian are 1C 00 00 00 00 00 which equates to decimal 28. This is the $MFT record number for the Orphaned Files folder we created and examined earlier.

Also view the last two bytes of this eight byte sequence. Note that the value is 01 00 (also little endian) or decimal one. This is the sequence number for the parent we viewed earlier and at this point, agrees with the current value found in the parent at offsets 16-17d.

Next, go back to the thumb drive using Windows Explorer and delete the Orphaned Files folder. Make an image of the thumb drive and name it Image02. Open this image in FTK Imager and view the Root Directory. Note the Orphaned Files folder you created is now marked as deleted. If you click on it you will see that it still retains the $MFT Record number of 28. You can also see that the files contained within the Orphaned Files folder are also deleted (See Figure 11).

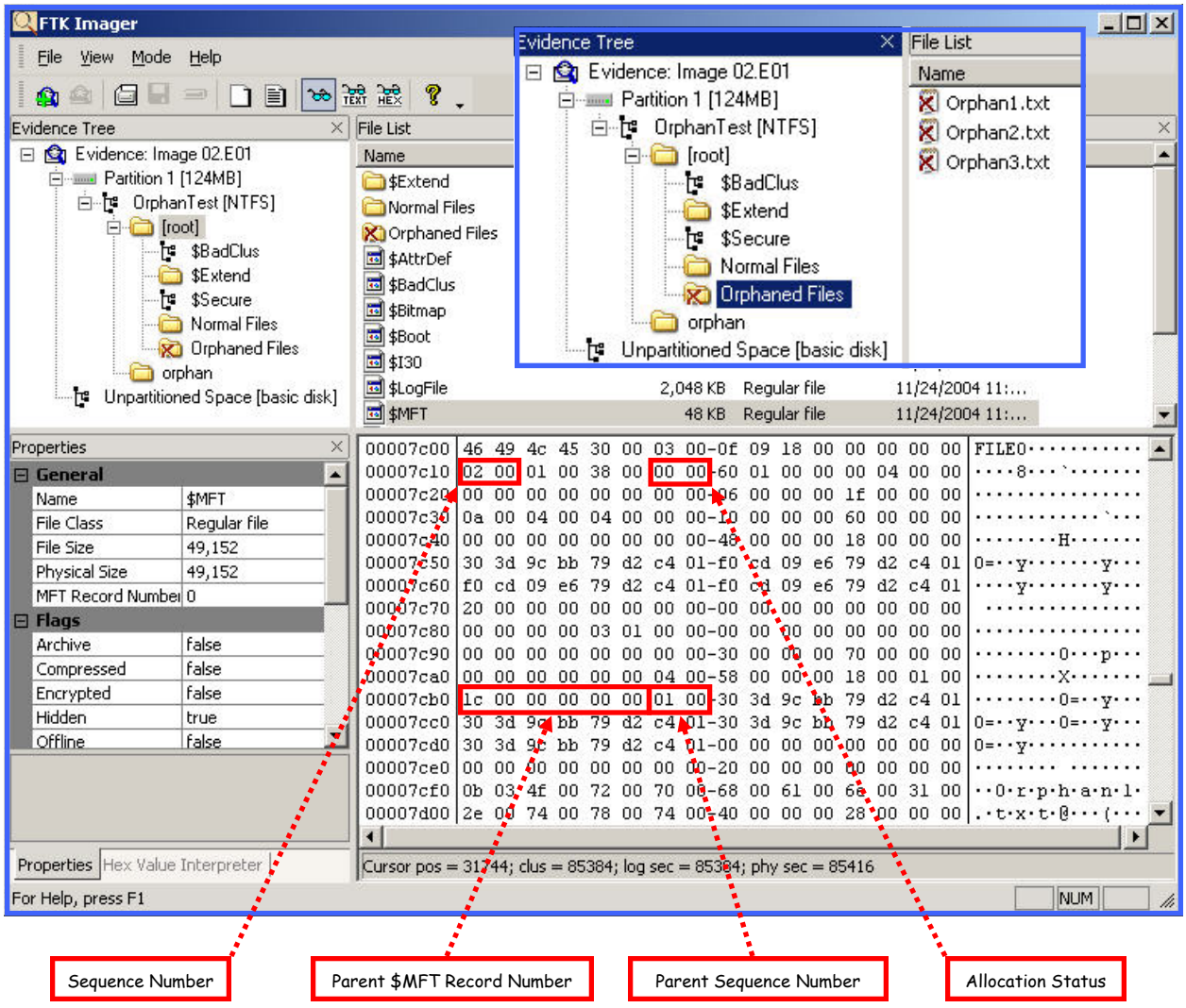

Figure 11 - Sequence, Allocation, and Pointers from Deleted Orphan1.txt to Parent

Check the Orphan1.txt deleted file by clicking on it and noting its $MFT Record number of 31. Navigate to that record in the $MFT (31 * 1024 = Offset 31,744) and note that offset 16-17d now reads 02 00 or 02d. This is the sequence value and you can now see that it has changed from 1 to 2 because the file has been deleted.

You can also see that offset 22-23d has also changed to a 00 value indicating, using Figure 4, that the status of the record is now a deleted file.

Navigate to offset 24 of the File Name Attribute and you will see that the eight byte sequence indicating the parent offset and sequence number has not changed. Figure 11 shows the Orphan1.txt file in the $MFT in its deleted state. FTK Imager considers this file as deleted but still belonging to the deleted parent folder Orphaned Files folder.

Now add a new file to the root of the thumb drive and name it ParentKiller.txt. This file will overwrite the original Orphaned Files folder record in the $MFT. The $MFT only adds new records if there are none available. It does so by searching from the top down to find an unallocated record. In this example, it will discover the unallocated folder record at number 28 and use it for ParentKiller.txt.

Once you have added this file, image the thumb drive and name the image, Image03. Open the image in FTK Imager and view what has changed. You will see in the Root Directory that the Orphaned

Files folder is gone, and in its place is the file called ParentKiller.txt. Select this file and note that its $MFT Record number is 28, the same as the original Orphaned Files folder.

Navigate to the $MFT offset for ParentKiller.txt (28 x 1024 = Offset at 28,672) and view the record data. Figure 12 shows the information in the $MFT for this file. You can see that the sequence number is two and hasn't been changed since the original folder was deleted, and the allocation number is ***now*** one for an allocated file. FTK Imager now considers the three deleted files to be orphans (See Figure 13).

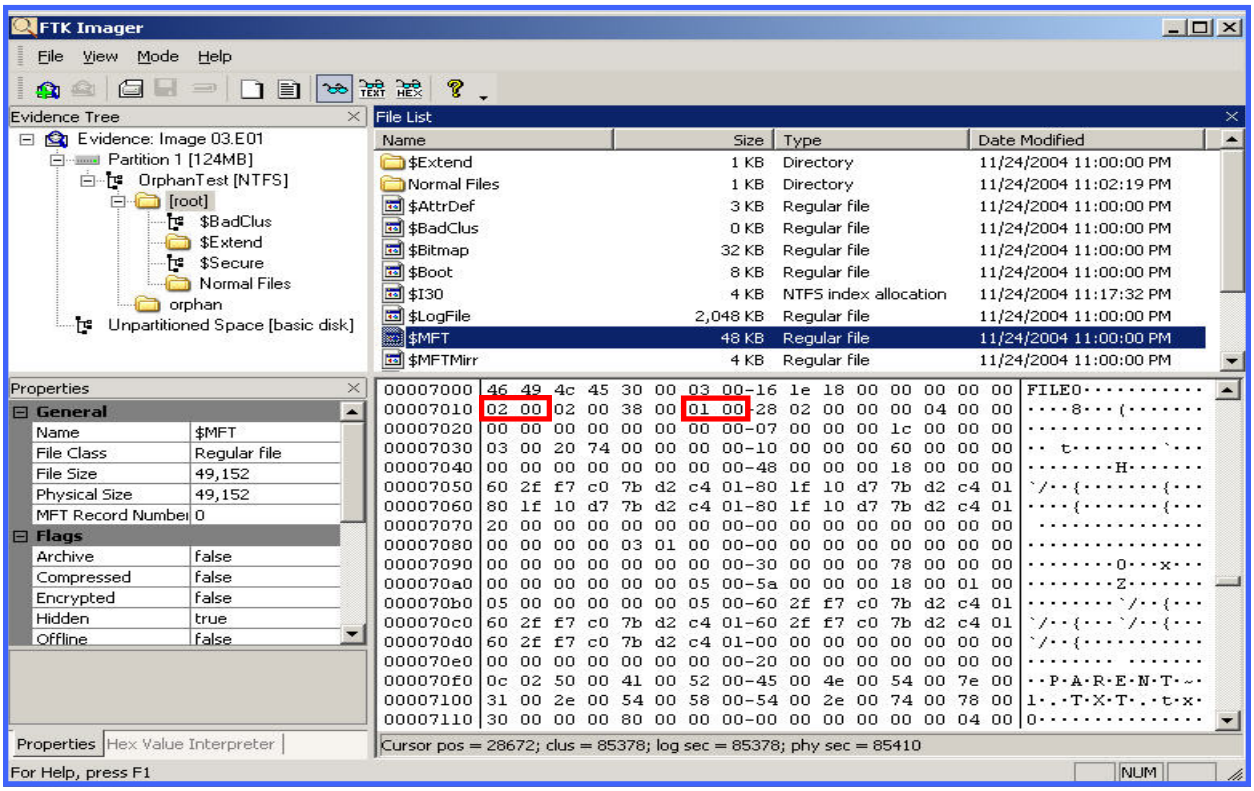

Figure 12 - Sequence and Allocation Numbers for ParentKiller.txt

Now navigate to the Orphan1.txt file and view the $MFT data for its record. The eight byte pointer in the File Name Attribute still points to record 28 and has the sequence number of the parent. FTK now knows this file is an orphan since its sequence number is one off and the allocation status of the record is 00, allocated.

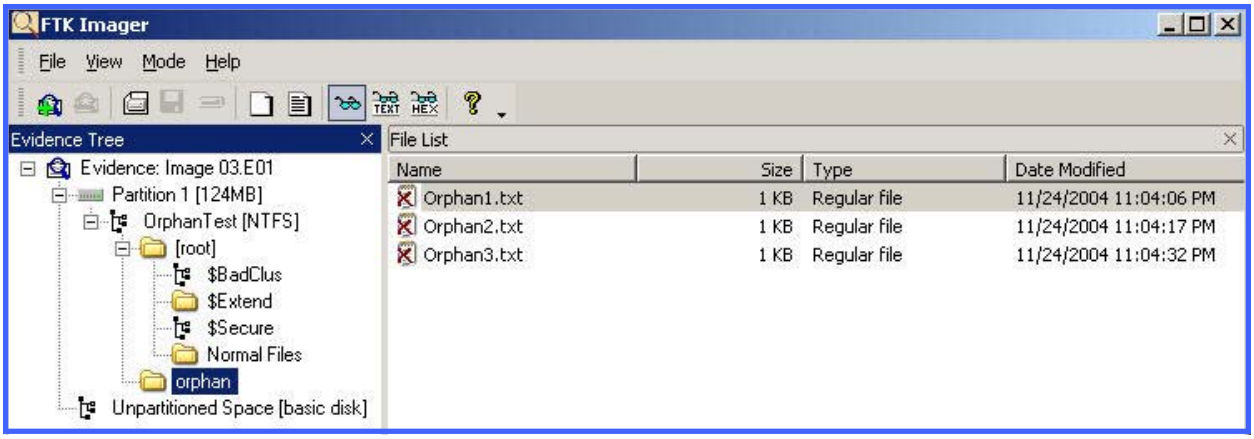

Figure 13 - FTK Imager Orphan Folder Now Occupied

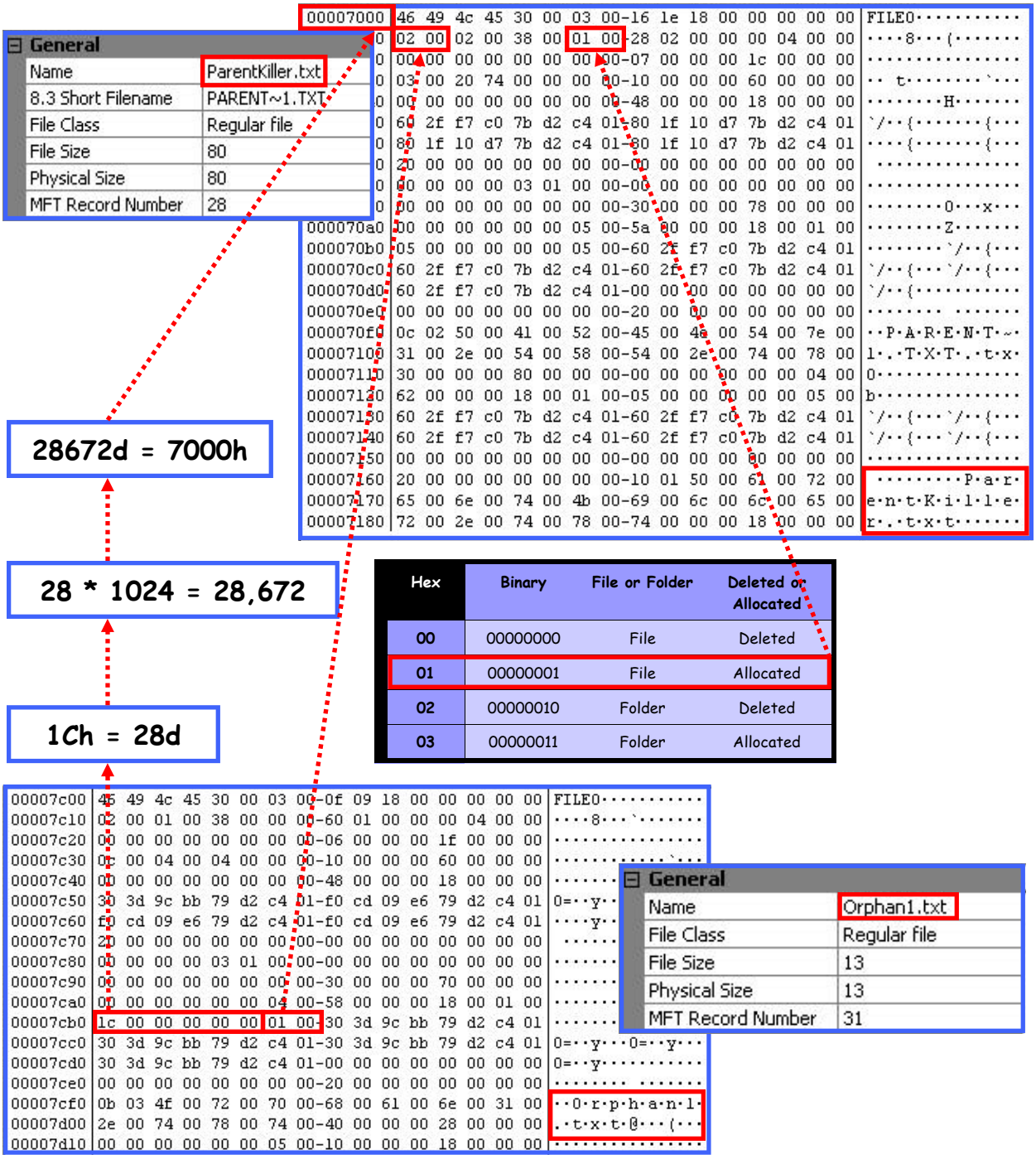

Figure 14 - NTFS, The Orphan Process

Figure 14 shows the process FTK or FTK Imager undertake to make the determination whether a file is orphaned or associated with a given folder. In our example, Orphan1.txt is an orphan of the $MFT entry now occupied by ParentKiller.txt. We can determine this by checking the eight byte pointer in Orphan1.txt.

This pointer indicates that the $MFT record for its parent is 1Ch or 28. Multiply the 28 times the record size, 1024 and you get an offset to navigate to of 28,672 or 70 00h. This pointer also gives you the sequence number of the parent record for comparative purposes.

Once you navigate back to the parent record, you can check the sequence number to see if they match. If they do, as in Figure 6, FTK knows the child belongs to the parent. If they do not, FTK checks the allocation status value. If the sequence number is one ahead, and the record is unallocated FTK knows the child is still associated with the parent record and is considered deleted. If the sequence is one or more ***and*** the allocation number indicates the record is allocated, FTK knows the child is not associated with that particular record and classifies it as an orphan and places it in the [orphan] folder.

FTK Imager and FTK both use this process to check the validity of child to parent. Thus in both utilities, you will see an [orphan] folder created off the root to store any file that meets the criteria of being orphaned from its associated folder.