Defining Digital Forensic Examination and Analysis Tools *

Brian Carrier
carrier@atstake.com

August 7, 2002

1 Introduction

The high-level process of digital forensics includes the acquisition of data from a source, analysis of the data and extraction of evidence, and preservation and presentation of the evidence. Previous work has been done on the theory and requirements of acquisition [5] and the preservation of evidence [2]. This paper addresses the theory and requirements of analysis and extraction of evidence from the acquired data.

This paper examines the nature of tools in digital forensics and proposes definitions and requirements. Current digital forensic tools produce results that have been successfully used in prosecutions, but lack designs that were created with forensic science needs. They provide the investigator with access to evidence, but typically do not provide access to methods for verifying that they are performing correctly. This is necessary when approaching digital forensics from a scientific point of view.

This paper addresses the layers of abstractions that exist in all digital data and the tools used to analyze them. The idea of using tools for layers of abstraction is not new, but a discussion of the definitions, properties, and error types of abstraction layers when used with digital forensics has not occurred. The concepts proposed here are applicable to any digital forensic analysis type, including those proposed by Baker in [6]:

- Media Analysis
- Code Analysis
- Network Analysis

This paper begins with definitions regarding digital forensic analysis tools, followed by discussions of abstraction layers. The abstraction layer properties are used to propose requirements for digital forensics analysis tools and finally an example of how the FAT file system uses abstraction layers is given.

*Digital Forensics Research Workshop 2002, Syracuse, NY

2 Definitions

The Digital Forensics Research Workshop in 2001 defined Digital Forensic Science as [6]:

> The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.

This definition covers the broad aspects of digital forensics from data acquisition to legal actions. This paper is limited in scope to the phases of identification and analysis. These phases come after the collection and validation phases where the digital data is acquired. The identification and analysis phases examine data acquired from the original system to identify evidence. Using this definition, one can make the following goal for identification and analysis of digital forensics:

> Identify digital evidence using scientifically derived and proven methods that can be used to facilitate or further the reconstruction of events in an investigation.

In general, the digital evidence needed by the investigation is found by analyzing all data that was acquired and evaluating it. As with any investigation, to find the truth one must identify data that:

- Verifies existing data and theories (*Inculpatory Evidence*)
- Contradicts existing data and theories (*Exculpatory Evidence*)
- Shows signs of tampering to hide data

To find these evidence types, all acquired data must be analyzed and identified. In other words, a digital forensic analysis is performed by analyzing all data to identify digital evidence that supports an existing theory, that which does not support an existing theory, and that which shows tampering.

Analyzing every bit of data is a daunting task when confronted with the increasing size of storage systems. Furthermore, the acquired data is typically only a series of byte values from the hard disk or network wire. Raw data like this is typically difficult to understand. In cases of multi-disk systems, such as RAID and Volume Management, acquired data from a single disk can not be analyzed unless it is merged with the data from other disks using complex algorithms.

The *Complexity Problem* in digital forensics is that acquired data is typically at the lowest and most raw format, which is often too difficult for humans to understand. It is not necessarily impossible, but often the skill required to do so is great, and it is not efficient to require every forensic analyst to be able to do so.

The Complexity Problem has thus far been solved by using tools to translate data through one or more layers of abstraction until it can be understood. For example, to view the contents of a directory from a file system image, the file system structures must be processed so that the

appropriate data structures are displayed. The data that represents the directory contents exist in the acquired file system image file, but in a format that is too low to identify. The directory is a layer of abstraction in the file system. Examples of non-file system layers of abstraction include:

- ASCII
- HTML Files
- Windows Registry
- Network Packets
- Intrusion Detection System (IDS) alerts
- Source Code

This paper is concerned with analysis tools that translate data from one layer of abstraction to another. It is proposed that the purpose of digital forensic analysis tools is to accurately present all data at a layer of abstraction and format that can be effectively used by an investigator to identify evidence. The needed layer of abstraction is dependent on the skill level of the investigator and the investigation requirements. For example, in some cases viewing the raw contents of a disk block is appropriate whereas other cases will require the disk block to be processed as a file system structure. Tools must exist to provide both options. The next section will cover abstraction layer properties with respect to digital forensics in more detail.

3 Layers of Abstraction

Layers of abstraction are used to analyze large amounts of data in a more manageable format. They are a necessary feature in the design of modern digital systems because all data, regardless of application, is represented on a disk or network in a generic format, bits that are set to one or zero. To use this generic storage format for custom applications, the bits are translated by the applications to a structure that meets its needs. The custom format is a layer of abstraction.

A basic abstraction example is ASCII. Every letter of the US English alphabet is assigned to a number between 32 and 127. When a text file is saved, the letters are translated to their numerical representation and the number value is saved on the media. Viewing the file raw shows a series of ones and zeros. By applying the ASCII layer of abstraction, the numerical values are mapped to their corresponding characters and the file is displayed as a series of letters, numbers, and symbols. A text editor is an example of a tool operating at this layer of abstraction.

Each abstraction layer can be described as a function of inputs and outputs. The layer inputs are a collection of data and a translation rule set. The rule set describes how the input data should be processed, and in many cases is a design specification of the object. The outputs of each layer are the data derived from the input data and a margin of error. In the ASCII example, the inputs are the binary data and the ASCII mapping ruleset. The output is the alphanumeric representation.

The output data of a layer can be fed as input to another layer, as either the actual data to be translated or as descriptive meta-data that is used to translate other input data. In the ASCII

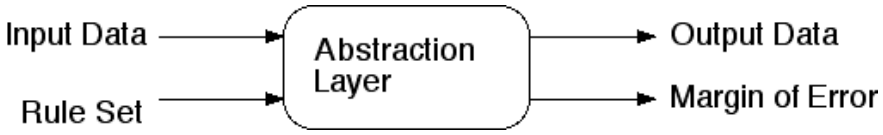

Figure 1: Inputs and Outputs of Abstraction Layer

example, if the text file was an HTML document then the output of the first layer, the characters, would be used as the input data to the HTML layer of abstraction. This layer takes the ASCII data and the HTML specification as input and outputs a formatted document. An HTML browser is an example of a tool that translates this, and typically the previous, layer.

An example of descriptive meta-data as input is the block pointer and type fields in a UNIX file system inode structure. The inode structure describes a file and includes a descriptor that indicates if the inode is for a file, directory, or some other special type. Another inode field is the direct block pointer that contains an address of where the file content is stored. Both values are used as descriptive data when processing the next layer of abstraction in the file system. The address is used to identify where to read data from in the file system and the type value is used to identify how to process it, since a directory is processed differently than a file. In this case, the output of the inode is not the only input to the next layer because the entire file system image is needed to locate the block address.

Abstraction layers occur in multiple levels. The file system itself is a layer of abstraction for the stream of bytes from the disk media. Within the file system are additional layers of abstraction and the end result is a smaller stream of bytes that represents a file. The file data is then applied to an application level of abstraction and it is processed further. This discussed further in Section 3.2.

3.1 Errors in Layers of Abstraction

Each layer of abstraction can introduce errors and therefore a margin of error is an output value of each. The errors introduced in this paper are not a comprehensive list of errors that exist during the investigation process. They are mearly the ones that exist because of analysis tools and the process of using layers of abstractions. Errors that are introduced from the attacker covering his or her tracks, from faulty imaging tools, or from an investigator misinterpreting the results of a a tool are not covered here, but are elsewhere [1].

Abstraction layers can introduce two forms of errors: Tool Implementation Error and Abstraction Error. *Tool Implementation Error* is introduced because of programming and tool design bugs. Examples of this include programming flaws, errors because the tool uses an incorrect specification, and errors because the tool uses the correct specification but the original application did not. This error is the most difficult to calculate because it requires extensive testing and code review. Efforts by the NIST Computer Forensics Tool Testing Group [4] can help reduce this error. One can assume that once a specific bug has been detected, it will be fixed and a new version of the tool will be released. Therefore, an investigator can keep this value minimal by keeping up to date on tool fixes.

To help identify the risk of unknown bugs in a given tool, each tool could have a margin of Tool Implementation Error calculated based on its bug history. The calculation would be based on the number of bugs found in recent years and the severity of each. This would be difficult with closed source applications because bugs that are not publicized could be quietly fixed and not added to the error value.

The second type of error is the *Abstraction Error*, which is introduced because of simplifications used to generate the layer of abstraction. This type of error occurs when a layer of abstraction is not part of the original design. For example, a file system image has several layers of abstraction in its design. Going from one layer to another will introduce no Abstraction Error. An Abstraction Error will exist in an IDS system that reduces multiple network packets into a specific attack. As the IDS does not know with 100% certainty that the packets were part of an attack, it introduces a margin of error. The error value introduced by the IDS can be different for different types of attacks that it is trying to detect. This error value can improve with research and better abstraction techniques.

In general, the *Abstraction Layer Error Problem* results from errors that are introduced by each layer of abstraction. The errors can be categorized into Abstraction Errors and Tool Implementation Errors. This problem is solved by calculating a margin of error for each layer and taking it into account while analyzing the resulting data. To help mitigate the risk associated with this problem, one needs access to the layer inputs, rule set, and outputs to verify the translation.

3.2 Layer Characteristics

Not all layers of abstraction or tools are the same. This section will provide four characteristics that can be used to describe a layer and the tools that process them.

Abstraction Error can be used to describe a layer by identifying it as a Lossy Layer or a Lossless Layer. A *Lossy Layer* is one that has a greater than zero margin of Abstraction Error associated with it. A *Lossless Layer* is one that has zero margin of Abstraction Error. Tool Implementation Error is not included in these definitions because it is a tool, not layer, specific value. File system abstraction layers and ASCII are examples of Lossless Layers, whereas IDS alerts are an example of a Lossy Layer.

A layer can also be described by its mapping attributes. A one-to-one layer has a unique mapping so that there is a one-to-one correlation between any input and output. The ASCII example and many layers of a file system fall into this category. The input of these layers can be determined given the output and rule set. A multi-to-one layer has a non-unique mapping where an output can be generated by multiple input values. The MD5 function is an example of this. Two inputs can generate the same MD5 checksum value, although it is difficult to find them. Another example of multi-to-one is with IDS alerts. One can generally not recreate the packets used to generate an alert using just the alert.

There can be layers of abstraction within a higher-level layer of abstraction. In the case of disk storage, there are at least three high-level layers of abstraction. The first is the media layer, which translates the unique on-disk format to the general format of sectors and LBA and CHS addressing that the hardware interface provides. The second layer is the file system layer that translates the sector contents to files. The third layer is the application layer that translates the file content to

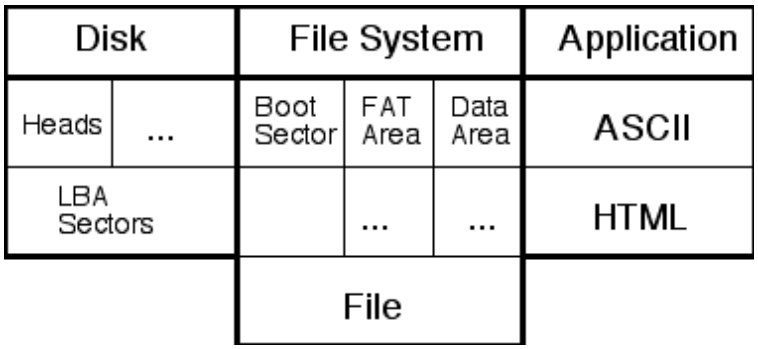

Figure 2: Levels and layers of abstraction of an HTML document

the needs of an application. The last layer in a level of abstraction is called the *Boundary Layer*. The output of this layer is not used as input to any other layers in that level. For example, the raw content of a file is a Boundary Layer in the file system. The translation to ASCII and HTML is done in the application layer level. This can be seen in Figure 2.

The tools for each layer can fall into different categories as well. A *Translation Tool* is one that uses a translation rule set and input data to generate output data. The purpose of this tool is to translate the data to the next layer of abstraction. A *Presentation Tool* is one that takes the data from the Translation Tool and displays it in a way that is useful to the investigator. From the investigators point of view, these tools need not be separate and in many current tools are not. Layers that produce a large amount of output data may separate them for efficiency. As an example, a Translation Tool would analyze a file system image and display the file and directory listings in the order that they exist in the image. One presentation tool would take that data and sort it by directory to display just the files within a given directory. A second presentation tool would sort the entries by the Modified, Access, and Changed (MAC) times of each file and display a time line of file activity. The same data exists in each result, but in a format that achieves different needs.

4 Analysis Tool Requirements

Using the above definitions and goals, a list of tool requirements can be generated.

Usability To solve the Complexity Problem (data at its lowest format is too difficult to analyze) tools must provide data at a layer of abstraction and format that helps the investigator. At a minimum, the investigator must have access to the layers of abstraction that are defined as Boundary Layers.

Comprehensive To identify both Inculpatory and Exculpatory Evidence, the investigator must have access to all output data at the given layer of abstraction.

Accuracy To solve the Error Problem (layers of abstraction introduce errors into the final product) tools must ensure that the output data is accurate and a margin of error is calculated so that

the results can be interpreted appropriately.

Deterministic To ensure the accuracy of a tool, it must always produce the same output when given a translation rule set and input.

Verifiable To ensure the accuracy of a tool, one needs to be able to verify the results. This can be done manually or by using a second and independent toolset. Therefore, one needs access to the inputs and outputs of each layer so that the output can be verified.

In addition to the required attributes, the following is proposed as a recommended feature.

Read-Only While not a necessity, this is obviously a highly recommended feature. As the nature of digital media allows one to easily make exact copies of data, copies can be made prior to using a tool that modifies the original. To verify the results, which is a requirement, a copy of the input is needed.

5 FAT File System Example

To illustrate the above, an example will be given using the FAT file system, one of the most basic file systems still used in many computers. This example will first give a brief overview of the file system layout, describe the proposed layers of abstraction, and provide an example of listing the root directory contents.

FAT32 is specifically used because it is more simple than FAT12 and FAT16 in the way that it stores the Root Directory.

5.1 FAT Design

This section provides a brief review of the FAT file system layout. For a more complete discussion refer to [3].

The FAT file system is broken up into main three areas. The first is the `Boot Sector` that contains the addresses and sizes of structures in this specific file system. The next two areas are the File Allocation Tables (FAT) and the Data Area. The locations of these areas are determined from the values in the Boot Sector. The `Data Area` is divided into consecutive sectors called `clusters`. Clusters store the contents of a file or directory. Each cluster has an entry in the `FAT` that specifies if the cluster is unallocated or which cluster is the next in the file that has allocated it.

Files are described by a `directory entry` structure. The directory entry structures are stored in the clusters allocated to the parent directory. The structure contains the file name, times, size, and starting cluster. The remaining clusters in the file, if any, are found by using the FAT.

5.2 FAT Abstraction Layers

All layers in this example will use the FAT32 specification as the input rule set. The FAT file system has seven layers of abstraction. The first layer uses just the partition image as input, assuming that

the acquisition was done of the raw partition using a tool such as the UNIX `dd` tool. This layer uses the defined Boot Sector structure and extracts out the size and location values. Examples of extracted values include:

- Starting location of FAT
- Size of each FAT
- Number of FATs
- Number of sectors per cluster
- Location of Root Directory

The second layer takes the image and information about the File Allocation Table (FAT) as input and gives the FAT and Data Area as output. The output of this layer is raw data from the image and is not structured.

The next two layers give structure to the FAT and Data Areas identified in the previous layer. One layer takes the FAT Area and FAT entry size as input and provides the table entries as output. The other layer takes the Data Area and cluster size as input and provides the clusters as output.

File and directory contents are stored in clusters in the Data Area. The fifth layer of abstraction in the File System Level takes a cluster and a type value as input. If the type is for a file then the raw cluster content is given as output. If the type is for a directory then a list of directory entries are given as output. If the raw content is given, then this is a Boundary Layer because there is nothing else that can be processed by the file system layers. The data would be passed to the application level.

If directory entries were given in the previous layer, then we have a partial description of a file or directory as we only have the first cluster in the file and not the rest. The sixth layer takes the starting cluster and the FAT as input and generates the full list of allocated clusters as output.

The seventh layer takes the clusters, the directory entry, and the full list of clusters as input and generates either the entire file contents or a directory listing. This layer uses the fifth layer previously described. Therefore, this layer is a Boundary Layer if the file contents are given. The layers are given in table form are given in Table 1.

A digital forensics analysis tool that was designed for the FAT file system with the requirements previously listed would provide the investigator with the inputs and outputs to each of the seven layers of abstraction. It would also present the output of each layer into one or more formats.

5.3 Directory Entry Listing Example

An analysis tool that allowed one to list the contents of the FAT32 root directory would do the following with an image:

1. Process layer 1 to identify Boot Sector values (including the location of the root directory).
2. Process layer 2 to identify the FAT and Data Areas.

	Input	Output
1	File System Image	Boot Sector Values
2	FAT information from SB	FAT and Data Area
3	FAT Area, FAT Entry Size	FAT Entries
4	Data Area, Cluster Size	Clusters
5	Cluster, Type	Directory Entries or Raw Content
6	Starting Cluster, FAT Entries	List of Allocated Clusters
7	All file meta-data, Clusters	All Directory Entries or Raw Content

Table 1: Abstraction Layers of FAT File System

3. Process layer 3 to get the FAT entries
4. Process layer 4 to get the clusters for the directory
5. Use the location of the root directory to process layer 6 to identify all allocated clusters.
6. Process layer 7 to get the listing of all names in the directory. The formatting tool in this case could display either all file details or just the names.

The tool would provide access to the outputs of each step above so that results can be easily verified.

6 Conclusion

This paper has documented the use of abstraction layers in digital forensic analysis. The use of abstraction layers is not a new idea, but little has been written about it. This paper proposes definitions and error types that can be found with abstraction layers so that they can be refined and expanded upon.

Abstraction layers are used in all modern digital systems. Therefore, digital forensics analysis tools are needed to translate them and provide an error value that will help determine how trustworthy the result is. No software is perfect and therefore each analysis tool will have an associated Tool Implementation Error based on its history. This value will help to establish trust when using an analysis tool.

The existence of lossy layers of abstraction is likely to increase as investigators are faced with an increasing number of logs, network packets, and file activity time lines to analyze. Abstraction layers will be used to reduce the number of log entries to examine, by grouping them together into higher-level events. This will introduce errors into the final result and therefore must be clearly understood and documented.

References

[1] Eoghan Casey. Error, Uncertainty, and Loss in Digital Evidence. *International Journal of Digital Evidence*, 1(2), Summer 2002.

[2] Chet Hosmer. Proving the Integrity of Digital Evidence with Time. *International Journal of Digital Evidence*, 1(1), Spring 2002.

[3] Microsoft Organization. *FAT: General Overview of On-Disk Format*, 1.03 edition, December 2002.

[4] NIST. Computer Forensic Tool Testing (CFTT). Available at: www.cftt.nist.gov.

[5] NIST CFTT. *Disk Imaging Tool Specification*, 3.1.6 edition, Oct 2001.

[6] Gary Palmer. A Road Map for Digital Forensic Research. Technical Report DTR-T001-01, DFRWS, November 2001. Report From the First Digital Forensic Research Workshop (DFRWS).