Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers

Brian Carrier
Research Scientist
@stake

Abstract
This paper uses the theory of abstraction layers to describe the purpose and goals of digital forensic analysis tools. Using abstraction layers, we identify where tools can introduce errors and provide requirements that the tools must follow. Categories of forensic analysis types are also defined based on the abstraction layers. Abstraction layers are not a new concept, but their usage in digital forensic analysis is not well documented.

1 Introduction

What does it mean to be a Digital Forensic Analysis Tool? How do we categorize the different types of analysis tools? For example, an investigator can view the files and directories of a suspect system by using either specialized forensic software or by using the operating system (OS) of an analysis system and viewing the files by mounting the drive. Both methods allow the investigator to view evidence in allocated files, but only the specialized forensic software allows him to easily view unallocated files. Additional tools are required if he is relying on the OS. Clearly both allow the investigator to find evidence and therefore should be considered forensic tools, but it is unclear how we should compare and categorize them.

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 data acquisition [7] and the preservation of evidence [4]. This paper addresses the tools that are used for the analysis of data and extraction of evidence.

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 the evidence is reliable. This is necessary when approaching digital forensics from a scientific point of view and could be a legal requirement in the future.

The core concept of this paper is the basic notion of abstraction layers. Abstraction layers exist in all forms of digital data and therefore in 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, which are defined later in the paper.

This paper begins with definitions regarding digital forensic analysis tools, followed by a discussion of abstraction layers. The abstraction layer properties are used to define analysis types and propose requirements for digital forensic analysis tools. Finally an example of how the FAT file system uses abstraction layers is given. This paper is an expanded version of the paper presented at the Digital Forensic Research Workshop II [1].

2 Definitions

The Digital Forensics Research Workshop I defined Digital Forensic Science as [8]:

> 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, which acquire data from the suspect system. The identification and analysis phases examine the acquired data to identify evidence. Using the broad definition from DFRWS, one can define the goal of the identification and analysis phases of digital forensics as:

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

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*)

To find both evidence types, all acquired data must be analyzed and identified. 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 are typically difficult to understand. In cases of multi-disk systems, such as RAID and Volume Management, acquired data from a single disk cannot be analyzed unless they are merged with the data from other disks using complex algorithms.

The *Complexity Problem* in digital forensics is that acquired data are 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.

To solve the Complexity Problem, tools are used 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, tools process the file system structures so that the appropriate values are displayed. The data that represents the files in a directory exist in formats that are too low-level to identify without the assistance of tools. 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
- Source Code

Similarly, the *Quantity Problem* in Digital Forensics is that the amount of data to analyze can be very large. It is inefficient to analyze every single piece of it. Data reduction techniques are used to solve this, by grouping data into one larger event or by removing known data. Data reduction techniques are examples of abstraction layers, for example:

- Identifying known network packets using Intrusion Detection System (IDS) signatures
- Identifying unknown entries during log processing
- Identifying known files using hash databases
- Sorting files by their type

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, are 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 value is saved on the media as bits. 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 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 rule set. The output is the alphanumeric representation.

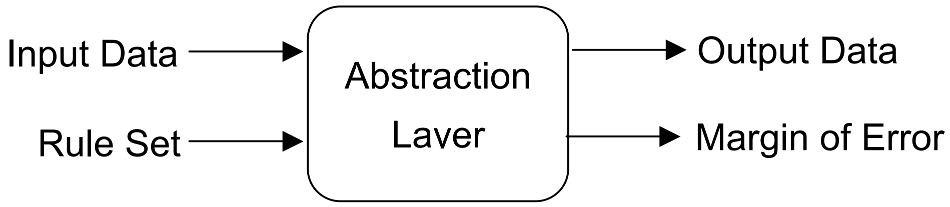

Figure 1: Abstraction Layer Inputs and Outputs

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 example, if the 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 output 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 layer 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, which is then applied to an application level of abstraction and it is processed further. Multiple levels of abstraction layers are discussed further in Section 3.2.

3.1 Abstraction Layer Errors

Each layer of abstraction can introduce errors and therefore a margin of error was identified as an output value. The errors discussed in this paper are not a comprehensive list of errors that exist during the investigation process. They are only the ones that exist because of analysis tools and the process of using layers of abstractions. Errors introduced from the attacker covering his tracks, from faulty imaging tools, or from an investigator misinterpreting the results of a tool are not covered here, but are elsewhere [2].

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 errors. 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 [6] can help reduce this type of error. Ideally, one can assume that if a fault (or 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 faults, a Tool Implementation Error could be calculated for each tool. The calculation would be based on the number of faults found in recent years and the severity of each. As it would be in a vendor’s best interest to have this value as small as possible, it could be difficult to calculate with closed source applications because faults that are not publicized could be quietly fixed and not added to the calculation.

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 would introduce no Abstraction Error. Alternatively, an Abstraction Error would exist in an IDS system that reduced multiple network packets into a specific attack. As the IDS did not know with 100% certainty that the packets were part of an attack, it introduced a margin of error. The error value for an IDS would be different for the different attacks that it was trying to detect. This error value could be improved with research and better abstraction techniques.

Using these, we can define the *Abstraction Layer Error Problem* as the errors that are introduced by the layers of abstraction. Calculating a margin of error for each layer and taking it into account while analyzing the resulting data could solve this problem. 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 Abstraction 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 hash 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 entire packet sequence that generated an alert.

There can be layers of abstraction within a higher-level layer of abstraction. In the case of disk storage, there are at least four high-level layers of abstraction. The first is the physical 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 media management layer that translates the entire disk to smaller partitions. The third layer is the file system layer that translates the partition contents to files. The fourth layer is the application layer that translates the file content to 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 level. The translation to ASCII and HTML is done in the application layer level. An example using an HTML file can be found for all four levels in Figure 2.

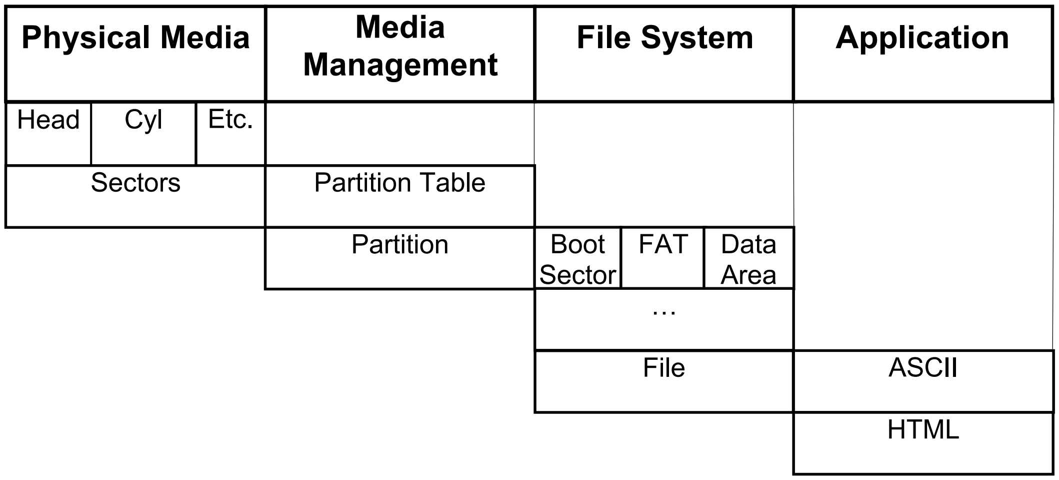

Figure 2: Abstraction Levels and Layers of an HTML File

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 investigator's point of view, these tools need not be separate and are not in many current tools. Layers that produce a large amount of output data may separate the tools for efficiency.

As an example, a Translation Tool could analyze a file system image and display the file and directory listings in the order that they existed in the image. One presentation tool could take that data and sort it by directory to display just the files within a given directory, similar to the output of 'ls' or 'dir' in UNIX or Windows. A second presentation tool could sort the entries by the Modified, Access, and Changed (MAC) times of each file and display a timeline of file activity. The same data exists in each result, but in a format that achieves different needs.

4 Analysis Categories

The major categories of digital forensics can be defined using the notion of abstraction layers. The ones given here differ from those previously defined, such as [8]. Previous attempts at defining categories appear to rely more on the needed skill sets of an investigator.

Physical Media Analysis: The analysis of the physical media layer of abstraction, which translates a custom storage layout and contents to a standard interface, IDE or SCSI for example. The boundary layer is the bytes of the media. Examples include a hard disk, compact flash, and memory chips. The analysis of this layer includes processing the custom layout and even recovering deleted data after it has been overwritten, [3] for example.

Media Management Analysis: The analysis of the media management layer of abstraction, which organizes storage media. The boundary layer is another collection of bytes from the media. Examples of this layer include dividing a hard disk into partitions, organizing multiple disks into a volume, and integrating multiple memory chips into memory space. This layer may not exist in all types of media, for example a database may access an entire hard disk and not create partitions.

File System Analysis: The analysis of the file system layer of abstraction, which translates the bytes and sectors of the partition to directories and files. The boundary layer is file content. The analysis in this layer includes viewing directory and file contents and recovering deleted files.

Application Analysis: The analysis of the application layer of abstraction, which translates data, typically returned from the file system, into the custom format needed by the application. Analysis in this layer includes viewing log files, configuration files, images, documents and reverse engineering executables. The input data will typically come from the file system, but applications such as databases may read directly from the disk.

Network Analysis: The analysis of the network layer of abstraction, which translates the lowest level data from a physical network or wireless network to the data that is used by an application. Analysis in this layer includes analyzing network packets and IDS alerts. Analysis of logs generated by network services, a firewall or web server for example, falls under Application Analysis.

Memory Analysis: The analysis of the memory layer of abstraction, which translates the bytes of the memory media to processes and system data. Analysis in this area includes identifying the code that a process was running and extracting sensitive data that was not stored elsewhere.

5 Analysis Tool Requirements

Using the previously stated 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. The tool should also present the data in a clear and accurate format so that the investigator does not interpret the data incorrectly.

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 tool set. 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 are proposed as recommended features.

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 result, which is a requirement, a copy of the input is needed.

Sanity Checks: All data values can be used as input to an abstraction layer, but only some outputs will be valid. Therefore, the investigator should be able to distinguish between valid and invalid outputs. To assist investigators, Presentation Tools should conduct sanity checks on the output and indicate if it is valid.

6 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 that is 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 simpler than FAT12 and FAT16 in the way that it stores the Root Directory.

6.1 FAT Design

This section provides a brief review of the FAT file system layout. For a more complete discussion refer to [5].

The FAT file system is broken up into main three areas. The first area 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 which are identified 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 identified using the FAT.

6.2 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, which was previously described. Therefore, this layer is a Boundary Layer if a file is being analyzed. The layers are shown in table form in Table 1.

Table 1: Abstraction Layers of the FAT File System

Layer	Input	Output
1	Raw file system image	Boot Sector values
2	File system image and values from Boot Sector (layer 1)	FAT and Data Area
3	FAT Area (layer 2), FAT Entry Size (layer 1)	FAT Entries
4	Data Area (layer 2), Cluster Size (layer 1)	Clusters
5	Raw Cluster Content (layer 4), Content Type	Formatted cluster content (Directory Entries if directory type and Raw Content if file type)
6	Starting Cluster, FAT Entries (layer 3)	List of Clusters in a run
7	List of Clusters (layer 6), Clusters (layer 4), Formatted Cluster Content (layer 5), Type	All Directory Entries in a directory or all raw content of a file

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 in one or more formats.

6.3 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 the Boot Sector values (including the location of the root directory).
2. Process Layer 2 to identify the FAT and Data Areas.
3. Process Layer 3 to extract the FAT entries from the FAT area.
4. Process Layer 4 to extract the clusters from the Data Area.

5. Use the location of the root directory to process Layer 6 to identify all allocated clusters in the root directory.
6. Process Layer 7 to get the listing of all names in the directory. The formatting tool in this case would display either all file details or just the names.

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

7 Conclusion

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

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 abstraction layers is likely to increase as investigators use data reduction techniques to manage the increasing number of logs, network packets, and files. These layers will introduce errors into the final result and therefore must be clearly understood and documented. As the field of digital forensics matures, a common language must be developed to discuss the tools and techniques that we use.

8 Acknowledgments

Prof. Gene Spafford provided helpful comments and feedback on this version of the paper.

References

[1] Brian Carrier, Defining Digital Forensic Examination and Analysis Tools. In *Digital Research Workshop II,* 2002. Available at: http://www.dfrws.org.

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

[3] Peter Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Proceedings of the 6th USENIX Security Symposium*, 1996.

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

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

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

[7] NIST CFTT. *Disk Imaging Tool Specification*, 3.16 edition, Oct 2001.

[8] Gary Palmer, A Road Map for Digital Forensic Research. Technical Report DTR-T0010-01, DFRWS, November 2001. Report from the First Digital Forensic Research Workshop (DFRWS).

About the Author

Brian Carrier (carrier@atstake.com) is a Research Scientist at @stake in Boston, MA, and the technical lead for the @stake Incident Management and Forensics Center of Excellence and Response Team. He has authored several open source forensic tools including The @stake Sleuth Kit (TASK), the Autopsy Forensic Browser, and TCTUTILs. Brian teaches forensics, incident response, and file systems at the @stake Academy and the SANS forensics track. Brian is a member of the Honeynet Project and the Forum of Incident Response and Security Teams (FIRST). Brian has a Masters in Computer Science from Purdue University where he was a Research Assistant at the Center for Education and Research in Information Assurance and Security (CERIAS). Additional papers and tools can be found at http://www.cerias.purdue.edu/homes/carrier/forensics.