A Formal Logic for Digital Investigations: A Case Study Using BPB Modifications.

I. Mitchell

Middlesex University, UK

Abstract

A Formal Logic is developed and the following presented: i) Notation for Formal Logic; ii) developing the Formal Logic for Digital Investigations Experiments (FDIE); iii) Case study modifying BPB and demonstrating the Formal Logic in ii); and iv) extending the Formal logic to Digital Investigations (FDI). A case study using BIOS Partition Block (BPB) modifications to render the partitions unreadable is used to successfully demonstrate the FDIE. From this experiment the Formal logic (FDIE) is extended to Formal Logic for Digital Investigations (FDI).

Keywords

Formal Logic, Recovery, Anti-contamination, Evidential Integrity, BPB, FAT, NTFS.

1. Introduction

Recovery of data often leads to changes being made to digital evidence. During judicial proceedings such changes bring into question the integrity of the evidence, the reliability of the process or processes used to recover data, and the credibility, expertise and skills of the investigator (McKemmish, 1999). When a third party does not reproduce identical results it costs time to re-examine the original investigation. These events can become acrimonious with each party accusing the other of malpractice or incompetence. The formal logic implemented here indicates simple steps taken to certify digital evidence and is an attempt to satisfy cases where modifying evidence is necessary.

Hidden Data essentially relies on two forms: i) non-physical; and ii) physical (see Berghel et al, 2006). The former is associated with Cryptography and Steganography, whereas the latter is associated with finding undetected storage locations within the file system. Huebner (Huebner et al, 2006) defines effective hiding techniques as those meeting the following criteria:

- Standard file system check should not return any errors.
- Hidden data will not be overwritten, or the possibility of data being overwritten is low.
- Hidden data will not be revealed when using standard GUI file system interfaces.
- A reasonable amount of hidden data can be stored.

The first definition above does not include the modifications of VBR as hidden data, since a simple "chkdsk" would reveal there is something wrong with the volume; at this point most investigators would apply a partition recovery tool and retrieve the data. Boot sector modifications are quite common - most viruses developed during the 80's and 90's rely on boot sector modifications (Cohen, 1994). While most anti-virus software detects and eliminates such viruses, there exists software to repair damage boot partitions, alas, these were not designed for Forensic Computing and only work on live devices, not images, and therefore such use would risk contamination of digital evidence.

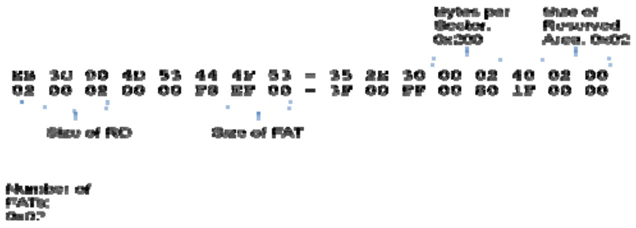
There are two problems: First, how to alter the boot sector for the recovery of the data; and the second how to recover the data without compromising the integrity of the image – *the formal logic aims to prove that no data tampering occurred during the modification of the boot sector.* The structure of this paper is as follows: Section 2 will give the background theory behind boot sectors; section 3 will provide guidelines on how to recover images; section 4 introduces the notation for a formal logic for Digital Forensic Experiments; section 5 argues how the authentication of the evidence remains intact and how the formal logic is extended to Digital Investigations; and, finally, section 6 makes conclusions.

2. File Systems and the BIOS Parameter Block, BPB.

This section gives a brief overview of the BPB under FAT16 and NTFS, respectively.

2.1. File Allocation Table, FAT16.

Figure 2.1 is a description of BPB and the associated values, taken and adapted from (Carrier, 2006; Microsoft, 2000).



```
EB 3C 90 4D 53 44 4F 53 - 35 2E 30 00 02 40 02 00
02 00 02 00 00 F8 EF 00 - 3F 00 FF 00 80 1F 00 00
```

Figure 2.1: BPB values showing location, associated values and representation.

From Figure 2.1 the FAT structure can be calculated and is represented in Figure 2.2. Cluster 2 is derived from Equation 2.1 and shows that an increase in the values in the Reserved Area will result in an increase in value for Cluster 2.

$$\textit{Cluster 2} = \textit{Reserved Area} + (\textit{Number of FATs} * \textit{FAT size}) + \textit{Size of Root Directory} \quad (2.1)$$

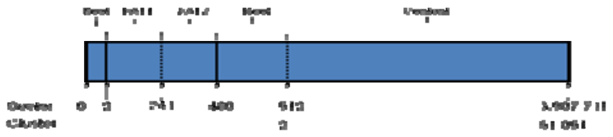
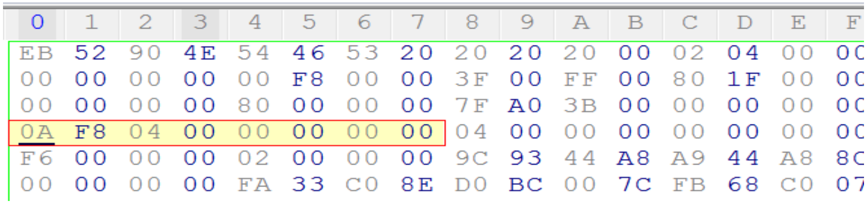


Figure 2.2: Schema of FAT structure derived from values in Figure 2.1, not to scale.

Modifying the BPB, e.g. the values of the bytes representing the size of the reserved area from 0x02 to 0xFF, results in the structure of the file system being shifted right without modifying any of the data stored in the content area. The file system does not adapt to the changes and becomes corrupted, leading to the file system being unrecognisable and unreadable and hence no access to the data. This process is reversible and resetting the parameters back to their original settings will result in the recovery of the drive.

2.2. New Technology File System, NTFS.

NTFS key feature is the Master File Table, MFT. Location of the MFT is stored in the boot sector as shown in Figure 2.3. Bytes 0x30-0x37 represent the Logical Cluster Number, LCN, of the MFT and the value of the offset is 0x4F80A, or 325,642 in decimal.



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
EB 52 90 4E 54 46 53 20 20 20 20 00 02 04 00 0C
00 00 00 00 00 F8 00 00 3F 00 FF 00 80 1F 00 0C
00 00 00 00 80 00 00 00 7F A0 3B 00 00 00 00 0C
0A F8 04 00 00 00 00 00 04 00 00 00 00 00 00 0C
F6 00 00 00 02 00 00 00 9C 93 44 A8 A9 44 A8 8C
00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07
```

Figure 2.3: Boot sector of NTFS partition. Highlighted bytes show the offset, in clusters, to the MFT.

Therefore the location of the $MFT is dependent on bytes 0x30-0x37 in the BPB and any changes to this value will result in the partition not being able to find the MFT; there is only one value it should not be, and that is the MFT Mirror, since if the boot sector pointed to the MFT Mirror the partition would still be readable.

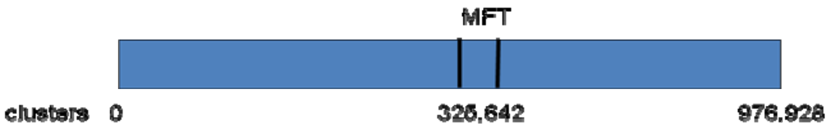


Figure 2.4: Schema of NTFS structure showing location of Master File Table, MFT, derived using values from Figure 2.3 (not to scale).

2.3. Summary

Experiment 1 will modify the values for the Reserved Area on a FAT16 device and render the partition unreadable. Experiment 2 will modify the values for Logical Cluster Number, LCN, for the MFT on a NTFS device and render the partition unreadable - an interview with Senior Forensic Computer Analysts of 7 years, revealed that such modifications would go undetected. The experiments have made two assumptions: i) changing these bytes has no effect elsewhere on the disk; and ii) reversing these changes recovers the data. By applying the formal logic to the experiment it can be proved that the recovery process above does not tamper the data. All methods that require altering evidence should be accompanied with explanatory/contemporaneous notes and there needs to be a method of identifying and correcting the problems detailed in sections 2.1 and 2.2. This will be investigated in the next section.

3. Analysis

The problem is given an image where it is suspected that there has been some boot sector tampering, then how is the process reversed if the default values are not known *a priori*? Some guidelines are given to reverse the process and return it to its original settings; this section will provide the theory behind steps taken to derive values independently of the boot sector, since it has become unreliable.

3.1. FAT Structure.

As mentioned in Section 2 the FAT structure is dependent on the Reserved Area. This analysis is restricted to restoring the value of the Reserved Area and independent of the values in the BPB. Table 3.1 outlines the parameters required to calculate the location of cluster 2.

Description	Acronym
Bytes Per Sector, located in boot sector; bytes 0x0B-0x0C.	**BPS**
Sectors Per Cluster, located in boot sector; byte 0x0D	**SPC**
Reserved Area, located in boot sector; bytes 0x0E-0x0F.	**RES**
Total Sectors in File System, located in boot sector bytes 0x20-0x23.	**Partition_Sectors**
Offset to partition, located in Partition table in MBR.	**Partition_Offset**
Size of Image, number of bytes allocated to image file. This can be derived from looking at the properties of the image.	**SoI**
Number of Sectors in Physical device.	**Physical_Sectors**

Table 3.1: Parameters used to calculate BPB values.

3.1.1. Data Acquisition.

Using a write-blocker image the MBR and the relevant logical partition; ensure that during data acquisition hashing algorithms have been employed and verified. Return the device. Use the MBR to determine the following:

1. **BPS**: Assume there are 512 Bytes Per Sector.

2. **Physical_Sectors:** Use `mmls` command from TSK (The Sleuth Kit, 2010) to calculate the size, in sectors, of the device.

3. **SoI**: Size of Image is equal to the product of **BPS** and **Physical_Sectors**.

4. If **SoI=BPS*Physical_Sectors** then move to next section; else then **BPS**=2***BPS** and goto 3. Repeat Until true.

3.1.2. **RES**:Size of Reserved Area on Logical Partition, bytes 14-15.

Bytes 14-15 represent the value of the number of reserved sectors that exist before FAT1. FAT1 will occur at the beginning of a new sector; therefore, using the value of BPS calculated in section 3.1.1 increment in steps of BPS through the document in a Hex Editor. Recognising FAT is non-deterministic, however, there are two markers that may help. First the end-of-file marker represented by the value, 0x0FFF FFFF and 0xFFFF in FAT32 and FAT16, respectively. Second are contiguous cluster chains that will have entries one value greater than the cluster they are representing, e.g. the 8th cluster would point to the 9th cluster and have a value of 0x09. The first two clusters have different values that cannot always be relied on between different devices, however, the two points of cluster chains and end-of-file markers are very good indicators of FAT1. The beginning of FAT1's offset address is a factor of **BPS** and **RES**, therefore once the offset for FAT1 is discovered then it is simply divided by **BPB** to give **RES**.

At this point, the recovery process is complete for our experiment and the reversal process of restoring the values will begin. When making changes to images, keeping contemporaneous notes updated is an important part of the process to ensure reproducibility. After the alterations the image should be run through a hashing algorithm. This process is explained in a later section.

3.2. NTFS Structure.

Section 2 showed that NTFS partitions are dependent on finding the location of the MFT. This analysis is restricted to restoring the value of the LCN and independently of the logical boot sector calculating the original value of the LCN. Repeat the stages in sub-section 3.1.1 to complete the Data Acquisition phase and the calculation of bytes per sector.

3.2.1. Logical Cluster Number, LCN.

For every file included on the partition there is a corresponding MFT Entry. The MFT itself is a file, $MFT, and is the first entry in the MFT; thus the MFT has a reference to itself. This self reference can be used to find the location of the MFT and ultimately the original values of the LCN.

The $MFT file can be found by a simple string search, e.g. search of "FILE0" and $MFT, should result in addresses in the same sector. Record this figure as **LCNOffset**. Open the image in a hex editor and go to the offset of the MFT. Figure 3.1 shows a hex-dump of the image. The **LCNOffset (0x27FF5000)** needs confirming as outline in Equation 3.1.

$$\mathbf{LCNOffset = ((LCN * SPC) + Partition\_offset) * BPS} \qquad (3.1)$$

Navigate to the Data Attribute in the MFT Entry, for more details on NTFS Data structures see Chapter 11 in (Carrier, 2006). Check that the non-resident flag is set true, see byte 264 in bold in Figure 3.1. The content of the Data Attribute contains the VCN run list for the file, i.e the location of the file. In this case it is **0x3280000A F804**. The VCN run list has two parts the length and the location in clusters. The low 4-bits of the first byte represent the number of bytes to represent the length, **0x8000**, and the high 4-bits of the first byte represent the number of bytes to represent the location, **0x0AF804**. Transform this value to big endian notation, 0x04F80A and enter the values in Equation 3.1

```
( (0x04F80A * 0x04) + 0x1F80) * 0x200 = 0x27FF5000
```

```
  0  46494C45 30000300 FE224000 00000000 FILE 0... ."@. ....
 16  01000100 38000100 A0010000 00040000 .... 8... .... ....
 32  00000000 00000000 06000000 00000000 .... .... .... ....
 48  02000000 00000000 10000000 60000000 .... .... .... `...
 :
144  00000000 00000000 30000000 68000000 .... .... 0... h...
 :
256  80000000 48000000 01004000 00000100 .... H... ..@. ....
272  00000000 00000000 7F000000 00000000 .... .... .... ....
288  40000000 00000000 00000400 00000000 @... .... .... ....
304  00000400 00000000 00000400 00000000 .... .... .... ....
320  3280000A F80400A6 B0000000 50000000 2... .... .... P...
 :
400  e8000100 0030a98a ffffffff 00000000 .... .0.. .... ....
```

Figure 3.1: Hex-dump of MFT entry of MFT. At offset 256 shows the beginning of the Data Attribute that contains the VCN of the MFT itself.

4. Formal logic for Digital Forensic Experiment.

Following the analysis in the previous sections the observation of two experiments on a NTFS and FAT partition are completed. These experiments cannot represent all the possible changes and different instances that can be made, but are merely a demonstration of applying the Formal logic to a Digital Forensic Experiment and primarily to see if the hypothesis that the reversal process recovers the data is correct.

4.1. Notation

Table 4.1 introduces the notation to represent physical devices and images as *I*, and hashing and copying as functions *H* and *D*, respectively.

I_n ,	where *I* represents images, *n=0,1,2,3,...,N* is the identification of the image e.g. I_2 represents Image *2* with no modifications and I'_2 represents Image *2* with a modification.
$H_k(I_n)$,	where *H* represents a hashing algorithm identified by *k*, e.g. $H_{md5}(I_2)$ employs the hashing algorithm *md5* to Image I_2 .
$D(I_n)$,	where *D* represents duplication $D(I_n) \rightarrow I_{n+1}$ and $H_{md5}(I_n) \equiv H_{md5}(I_{n+1})$. The duplication of I'_n will produce an image I_{n+1}, e.g $D(I'_n) \rightarrow I_{n+1}$ and $H_{md5}(I'_n) \equiv H_{md5}(I_{n+1})$. However, $H_{md5}(I'_n) \neq H_{md5}(I_n)$ since I'_n is a modification of I_n .

Table 4.1: Formal Logic Notation for Data Acquisition in Forensic Computing.

4.2. Method

The experiment is to image the physical device, record the MD5 and SHA hash values and ensure they are verified. Then make changes to the device, image the physical device, record the MD5 and SHA hash values and ensure they are verified. The aim of both experiments is to return the image of the device to its original state and see if the hash values match that of the original device. This will prove that the changes made to the evidence have not altered the data in any way and were necessary. Using the notation in Table 4.1 the experiments can be formalised in the enumerated list below:

1. Image I_0 . $D(I_0) = I_1$; (imaging of device).
2. Record and verify Hash the values, $H_k(I_0) \equiv H_k(I_1)$
3. Image I_1 . $D(I_1) = I_2$
4. Record and verify the hash values. $H_k(I_1) \equiv H_k(I_2)$.
5. Make relevant changes to the partition boot sector on the original device, I'_2, ie make it unreadable.
6. Image I'_2. $D(I'_2) = I_3$
7. Record and verify the hash values. $H_k(I'_2) \equiv H_k(I_3)$.
8. Reverse the changes, as noted in section 3, I'_3, ie make it readable.
9. Record the hash values of I_1 and I'_3 i.e. $H_k(I_1) \equiv H_k(I'_3)$.

List 4.1 Formal logic for Experiments – to be completed with contemporaneous notes – specific to Experiments 1 and 2.

In general there are two stages to the experiment: forward and backward chain of events. These are reflected in List 4.2 that should be applied to general experiments. By looking at the change between associated iterations can reveal where further research by the investigator is required.

Forward and Backward Chain of events

1. Image device, $D(I_0) = I_1$
2. Record and verify the hash values. $H_K(I_0) \equiv H_K(I_1)$.
3. Time, *t=1;*
4. Document and make any changes to I_t to yield I'_t
5. Duplicate I'_t. $D(I'_t) = I_{t+1}$;
6. Record and verify the hash values, $H_K(I'_t) \equiv H_K(I_{t+1})$.
7. Time, *t=t+1;*
8. Repeat stages 4 – 7 until no more modifications are required.

Check $H_K(I_0) \equiv H_K(I_t)$, if these are correct then the experiment is a success.

Listing 4.2 Formal logic for Digital Investigation Experiments, FDIE.

4.3. Results.

Two 2GByte USB Memory sticks, each formatted with a single logical partition as FAT16 and NTFS, where used in the experiment. After formatting, 35 digital images of various sizes were copied to the device. The hash values are documented for each experiment in Table 4.2. The experiments show that changes can be recorded at intermediate stages and that a third independent party when following these instructions will also yield the same result. Table 4.2 shows that the hash values in stage 9 are identical. Hence the image has returned to its former state, without changing the data in the content area. Also, Table 4.2 shows that the image is identical to the original state and therefore the 35 digital images can be retrieved. Stages 1-9 were developed to maintain the integrity of the experiment and ensure that no evidence spoliation occurred. In the next section these stages are adapted and proposed as a set of guidelines for Forensic Computing Practitioners when it is necessary to make changes to the image.

5. Evidential Integrity.

McKemmish (McKemmish, 1999) mentions "... the examiner has a responsibility to correctly identify and document any change...." and comments further "During the forensic examination this point may seem insignificant, but it becomes a critical issue when the examiner is presenting their findings during judicial proceedings. Whilst the evidence may be sound, questions regarding the examiner's skills and knowledge can affect their credibility as well as the reliability of the process employed".

	Experiment 1 – FAT16	Experiment 2 - NTFS
2	$H_k(I_0)$=d6655334452c5ce811867a0941c6eec0 $H_k(I_1)$=d6655334452c5ce811867a0941c6eec0	$H_k(I_0)$=fdeecf171e38c7928fdcbb6753f87ab7 $H_k(I_1)$=fdeecf171e38c7928fdcbb6753f87ab7
4	$H_k(I_1)$=d6655334452c5ce811867a0941c6eec0 $H_k(I_2)$=d6655334452c5ce811867a0941c6eec0	$H_k(I_1)$=fdeecf171e38c7928fdcbb6753f87ab7 $H_k(I_2)$=fdeecf171e38c7928fdcbb6753f87ab7
7	$H_k(I'_2)$=107ca1f93083cd434542ee8dff0d7e4c $H_k(I_3)$=107ca1f93083cd434542ee8dff0d7e4c	$H_k(I'_2)$=1eb2fe26df606a6bffc58de059639dc9 $H_k(I_3)$=1eb2fe26df606a6bffc58de059639dc9
9	$H_k(I'_3)$=d6655334452c5ce811867a0941c6eec0 $H_k(I_1)$=d6655334452c5ce811867a0941c6eec0	$H_k(I'_3)$=fdeecf171e38c7928fdcbb6753f87ab7 $H_k(I_1)$=fdeecf171e38c7928fdcbb6753f87ab7

Table 4.2. Table documenting results for experiment. Hashing algorithm, *k*, is MD5. Numbers in left hand-hand column relate to steps in section 4.2. The final stage in both experiments matched and all 35 images were recovered.

McKemmish makes two important points: "*document any change*"; and "*reliability of the process*". The 9 stages in section 4.2 combined with contemporaneous notes ensure these two important factors when handling digital evidence in the experiments in section 4.3. This allows independent third parties to follow steps above and verify that no other alterations have contaminated the data; only modifications documented in contemporaneous notes have been made. However the 9 stages in section 4.2 cover an experiment and have been refined for a digital investigation where it is necessary to modify the image:

1. Image device, $D(I_0) = I_1$
2. Record and verify the hash values. $H_k(I_0) = H_k(I_1)$.
3. Time, *t=1;*
4. Document and make any changes to I_t to yield I'_t
5. Duplicate I'_t. $D(I'_t) = I_{t+1}$;
6. Record and verify the hash values, $H_k(I'_t) = H_k(I_{t+1})$.
7. Time, *t=t+1;*
8. Repeat stages 4 – 7 until no more modifications are required.

List 5.1 Formal Logic for Digital Investigation (FDI) – to be completed with contemporaneous notes.

The proposed formal logic above not only documents the changes but also verifies the changes, by applying a hashing algorithm after each iteration, and thus assuring that independent third parties can reproduce identical results.

6. Conclusions.

The experiment shows that the proposed Formal Logic works, and the contributions are:

- Disabling partitions using BPB modifications; section 2 gives examples of how partitions can be modified and hence, all data cannot be read.
- Formal Logic Notation for Digital Investigations (section 4.1).
- Formal Logic for Digital Investigation Experiments involving Evidence Tampering, provided in section 4.2.
- Formal Logic for Digital Investigations involving Evidence Tampering, provided in section 5.

This experimental formal logic was extended to Digital Investigations. Documenting change and using hashing algorithms of the image to allow independent third parties to achieve identical results.

If assumptions are made about proprietary systems, then apply DFIE to see if these assumptions can be corroborated by comparing hash numbers in the forward and backward chain processes.

Finally, if during a digital investigation modifications are required in order to read the data then apply FDI. When third parties are required to reproduce results independently following FDI iterations and contemporaneous notes made will identify at which stage, by different hashes, and thus limit the scrutiny and cost of such an investigation.

7. References

Berghel, H., Hoelzer, D. And Sthultz, M. 2006. “Data Hiding Tactics for Windows and Unix File Systems”. http://www.berghel.net/publications/data_hiding /data _hiding.php [accessed February 2010].

Bowen, J, P., and Hinchey, M. J., 2006. “Ten Commandments of Formal Methods: Ten Years on....”. IEEE Computer Soceity, January issue.

Carrier, B. 2006. “File System Forensic Analysis”. Addison-Wesley.

Cohen, F. 1994. “A Short Course on Computer Viruses”. Wiley.

Huebner, E., Bem, D., and Cheong K-W. 2006. “Data Hiding in the NTFS file system.” Journal of Digital Investigation. Vol 3.

Microsoft. 2000. “FAT: General overview of On-Disk Format”. Hardware White Paper.

McKemmish, R,. 1999. “What is Forensic Computing?”. Australian Institute of Criminology. http://www.aic.gov.au

The Sleuth Kit. 2010. The Sleuth Kit, TSK downloaded from www.sleuthkit.org.