Detecting JFIF Header using FORHEADER[1]

Kamaruddin Malik Mohamad, Tutut Herawan, Mustafa Mat Deris

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

Faculty of Computer System and Software Engineering,
Universiti Malaysia Pahang

{malik,mmustafa}@uthm.edu.my, tutut@ump.edu.my

Abstract

Header and footer are important in digital investigation for JPEG file detection as only 16% of files are fragmented. The use of efficient algorithm to detect them is vital to reduce time taken for analyzing ever increasing data in hard drive or physical memory. Even though there are few applications developed for file carving that rely on header and footer e.g. Foremost, Scalpel; however the algorithm used for header detection is not much discussed. In this paper, we introduce three novel algorithms; single-byte-marker, dual-byte-marker and 20-point-reference for JPEG File Interchange Format (JFIF) header detection using a newly introduced FORHEADER model. Three experiments have been carried out using an image from hard disk and physical memory; and raw data from Digital Workshop Forensics Research Workshop 2006 (DFRWS 2006) challenge. The results obtained showed that dual-byte-marker algorithm provides better performance in terms of processing time for JFIF header detection.

Keywords: *File Carving, Digital Forensics, JFIF Detection, JPEG*

1. Introduction

According to the Federal Bureau of Investigation (FBI), in the year 2000 there were 2032 cases opened involving cyber crime. Of those cases, only 921 were closed. Of those closed cases, only 54 convictions were handed down in court [1]. This indicates that, there are too many cases to be solved and there is a need for automatic analyzing tools to help shorten the processing time as opposed to analyzing using traditional manual approach. Furthermore, traditional approach could only be done by forensics experts. Moreover, the advent of the Internet in the 1980s has dramatically increases child pornography problem. People may produce, distribute or download images easily. Due to this, the police need the possibility to scan hard disks and networks for potential illegal material more efficiently [2]. There are many methods to ensure that computer evidence is correctly obtained [3]. These steps are outlined in Table 1.

[1] An early version of this paper appeared in the Proceeding of International Conference, ISA 2010, Miyazaki, Japan, June 23-25, 2010, CCIS 76 Springer-Verlag, pp. 17–26, 2010.

Table 1. General Methods Used in Computer Forensics

Method	Description
Protect	Protect subject computer system from alteration, data corruption, virus infection, and physical damage.
Discover	Uncover all files: normal, hidden, deleted, encrypted, password-protected
Recover	Recover as many of the deleted file as possible
Reveal	reveal the contents of hidden and temporary files
Access	Access the protected and encrypted files, if legal
Analyze	Analyze all relevant data, including data located in unallocated file space and file slack
Report	Print out a listing of all relevant files, and provide an overall opinion on the system examination
Testimony	Provide expert testimony or consultation, if required

Digital evidence need to be collected and analyzed to come up with a concrete hypothesis to convict a criminal. One of the method to analyze digital evidence is by file reconstruction or known as file carving [4, 5, 6, 7, 8, 9, 10, 11]. File carving for non-fragmented JPEG can be done much easier by using the file metadata. Nevertheless, there are also some file carving researches recently that look into the file content rather than using metadata to solve corrupted and fragmented file. File carving is useful for data recovery and computer forensics [4]. The most common form of file carving is that they analyze headers and footers of a file and try to merge all the blocks in between [5]. One of the most popular JPEG file carving tool is Scalpel [12], which is an enhanced version of Foremost. However, these file carving tools still fail to merge files that are fragmented. File carving involving fragmented files are discussed in [4, 5, 6, 8, 11].

In this paper, three novel algorithms are introduced; single-byte-marker, dual-byte-marker and 20-point-reference for JPEG File Interchange Format (JFIF) header detection using a newly introduced FORHEADER model.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes FORHEADER model which is used for header detection. Section 4 describes the JFIF files. Section 5 describes the JFIF header detection algorithms. Section 6 describes the experiments and then compares the results in terms of processing time. Section 7 describes the result and discussion. Finally the conclusion is described in Section 8.

2. Related Work

Scalpel [12] is one of the popular file carving tool is which is developed based on Foremost engine with improvement on its performance and memory usage. It is able to run on machines with modest resources and performs carving operations very rapidly. Scalpel reconstructs JPEG file based on headers and footers in an non-fragmented environment. This tool makes only two complete sequential passes (two passes) over a hard disk image. The offsets or locations of headers and footers are collected and indexed in the first pass, while the file carving process is done in the second pass. Our research work is focusing on detecting these headers through the use of few novel algorithms, which is more related to Scalpel's first pass process.

FORSIGS is a model used for searching malicious digital pictures in hard drive using fingerprinting [13]. FORWEB is another model adapted from FORSIGS, which is used for

searching malicious digital pictures resident in remote Web servers across the Internet [13]. Both of these models are using a 16-point reference (16 bytes of fingerprint signature) for matching signatures from the database. The 16-point reference has been the standard used in England and Wales [14].

3. FORHEADER Model

FORHEADER is adapted from FORSIGS and FORWEB models. FORSIGS and FORWEB are designed for searching malicious pictures, while FORHEADER is designed for header detections. In this research, tests are performed on JFIF images only. JFIF is used in our experiments because it is the de-facto JPEG format for sharing in many applications and widely used in the Internet [15]. Nevertheless, FORHEADER can also be extended to detect file headers of other JPEG format such as Exchangeable Image File format (Exif); by using appropriate header structure and markers criterion. The model is illustrated in Figure 1.

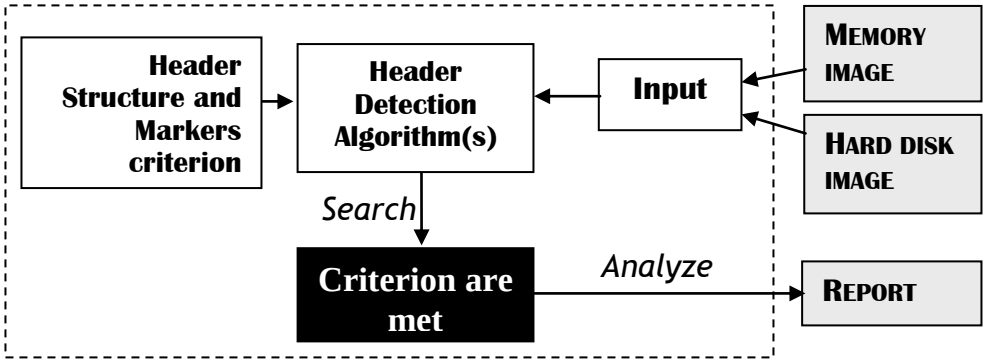

Figure 1. FORHEADER Model

FORHEADER is using input from two sources namely images from physical memory and hard disk. The header detection algorithms uses the header structure and markers criterion to process the input data. Once the criterion for the header structure and markers are met, the data from the input will then be analyzed for reporting.

There are few differences between FORHEADER, FORSIGS and FORWEB. These differences are listed below:

a. FORHEADER is used for file header detection while FORSIGS and FORWEB are used for malicious file detection.

b. FORHEADER uses markers and data structures for header detection while FORSIGS and FORWEB use signatures.

c. FORHEADER uses physical memory or hard disk images as input data, while FORWEB is using files from web server as input data; and FORSIGS is using files from hard disk as input data.

d. FORHEADER is using two variable length data structures; 18-byte (in single-byte-marker and dual-byte-marker algorithms) and 20-byte (in 20-point-reference algorithm) structures instead of a fixed 16-point reference fingerprinting signature as used in FORWEB.

e. FORSIGS and FORWEB use database of signatures while FORHEADER is using header structures and markers criterion which can be implemented simply by using a flat file.

Only one pass is used by FORHEADER for header detection. In this pass, information about the valid JFIF headers from the input data will be displayed and finally the total number of detected JFIF headers will be known. By focusing on a specific file format, this would help the forensics investigator to decrease the number of files for further analysis and directly cut down the processing time. In this paper, three header detection algorithms are discussed and compared according to their processing time.

4. JFIF

JFIF is the minimal JPEG format to enable file exchanges between a wide variety of platforms and applications [16]. Any effort done to reduce the duration for disk-imaging and data analysis should be encouraged because hard disk size is increasing exponentially. By focusing only on detecting and analyzing JFIF files, the time taken for analysis could be reduced and could finally give us some indication on the total number of JFIF files available in the data image. The JFIF segment format is illustrated in Table 2. The information in Table 2 is the basis used for reporting of detected JFIF headers. The application marker (or APP0) and identifier in the JFIF segment format is important in identifying JFIF headers.

Table 2. JFIF Segment Format

Field	Size (byte)	Description
APP0 marker	2	The application marker for JFIF is always equal to **0xFFE0**.
Length	2	Length of segment excluding APP0 marker
Identifier	5	Always equals to "**JFIF**" (0x4A 46 49 46 00) followed by **0x00;** or 'a zero terminated string: JFIF'.
Version	2	First byte is major version (currently 0x01); second byte is minor version (currently 0x02) or version 1.2.
Density Units	1	Units for pixel density fields • 0 – No units, aspect ratio only specified • 1 – Pixels per inch • 2 – Pixels per centimeter
X density	2	Integer horizontal pixel density
Y density	2	Integer vertical pixel density
Thumbnail width (tw)	1	Horizontal size of embedded JPEG thumbnail in pixels
Thumbnail height (th)	1	Vertical size of embedded JPEG thumbnail in pixels
Thumbnail data	3 x tw x th	Uncompressed 24 bit RGB raster thumbnail.

A valid JFIF file must starts with a two-byte start-of-image (SOI) marker which is 0xFFD8. This hex value will be used in algorithms developed for detecting headers. These algorithms will be discussed in Section 5. To validate a JFIF file, SOI and JFIF identifier must both be found. SOI must be the first two bytes located at the start of a JFIF file, and the identifier must be at the seventh byte from the start of the file. The identifier for JFIF is "JFIF\0" or a "JFIF" string terminated by a NULL or "\0". The first 11 bytes can be used to uniquely identify the JFIF file. Nevertheless, the whole JFIF header (18 bytes) will be displayed to the screen to aid forensics investigator to have more complete information. The sample data for the first 11 bytes of a JFIF is illustrated in Figure 2. The actual content of a JFIF file can be viewed in hex and ASCII formats by using a hex editor such as BinText (free software) [17] or HexAssistant (commercial software) [18].

	SOI	APP0	Length	Identifier
Hex codes	0xFF D8	0xFF E0	0x00 10	0x4A 46 49 46 00
ASCII				J F I F NULL
Size in bytes	2 bytes	2 bytes	2 bytes	5 bytes

Figure 2. First 11 bytes of a JFIF file header with a sample of valid hex codes

5. JFIF header detection algorithms

This section discusses about the three novel JFIF header detection algorithms namely single-byte-marker, dual-byte-marker and 20-point-reference algorithms. The performance of these algorithms are compared according to their processing time taken. All these algorithms will be discussed in details in the following sections.

5.1. Single-Byte-Marker Algorithm

The algorithm is named as single-byte-marker because one byte is read from the input data each time. The algorithm is illustrated in Figure 3. One byte is read from the input data (line 2). If the first byte read matches 0xFF (line 3); then read the second byte (line 4). If the second byte read matches 0xD8 (line 5), then obtain an 18-byte structure (line 6). The 18-byte structure is illustrated in Figure 4 where u8 represents unsigned 8-bit or unsigned byte. If JFIF identifier is successfully matched (line 7), then a valid JFIF header is successfully detected (line 8). If JFIF identifier is not matched, then return the structure and let the pointer points to the starting byte of the structure (line 9-11). If the second byte read is 0xFF (contiguous 0xFFFF), the second byte will be returned to the input stream. Then positioned the pointer to the returned byte (line 13-15).

The single-byte-marker algorithm can be generalized as the followings:

$$\text{Let } P(x) \text{ be 0xFF.}$$
$$\text{Let } Q(x) \text{ be 0xD8.}$$
$$\text{Let } R(x) \text{ be “JFIF\0”.}$$
$$\exists x \in Z^+, P(x) \wedge Q(x) \wedge R(x) \Leftrightarrow \text{JFIF Header is found.}$$

Best case scenario:

Best case scenario happens when the first read is $P(x)$, second read is $Q(x)$ and third read is $R(x)$, where *n* is the size of the input data in bytes. Thus, for every three reads, there is one successful detection of JFIF header. Thus, the big O value for best case scenario is shown below.

$$\frac{n}{20} \times 3 = \mathrm{O}(n).$$

Worst case scenario:

Worst case scenario happens when all the input data are made of 0xFFs. Therefore, when the first read and the second read are both 0xFFs, the second byte will be returned to the input stream. In the next round, the first read will obtain the byte that

has been returned to the input stream. Thus, the big O value for worst case scenario is shown below.

$$2(n-1) = \mathrm{O}(2n).$$

```
1.  Do
2.    Read 1 byte                    // FIRST read
3.    If (byte = 0xFF marker) then
4.      Read next 1 byte             // SECOND read
5.      If (byte =  SOI marker)      // SOI is 0xD8
6.        Read 18-byte structure     // THIRD read
7.        If (Match JFIF identifier)
8.          Valid JFIF Header is FOUND
9.        Else
10.         Return the structure (18 BYTES) to input stream
11.         Position the pointer to the STARTING byte of the returned structure
12.       End-if
13.     Else if (byte = 0xFF) marker   // found contiguous 0xFF
14.       Return the byte to input stream
15.       Position the pointer to the returned byte
16.     End-if
17. While not end-of-image
```

Figure 3. Single-byte-marker Algorithm

```
struct jpegHeader {
        u8 APP0 [2];            // e.g. 0xFF 0xE0
        u8 length [2];          // excluding APP0
        u8 identifier [5];      // "JFIF\0"
        u8 version [2];         // e.g. 0x01 0x02; v1.2
        u8 density_units;       // e.g. 0x01
        u8 x_density[2];        // e.g. 0x00 0x48
        u8 y_density[2];        // e.g. 0x00 0x48
        u8 thumbnail_width;     // e.g. 0x00
        u8 thumbnail_height;    // e.g. 0x00
};
```

Figure 4. The 18-byte structure used in single-byte-marker and dual-byte-marker algorithms

5.2. Dual-Byte-Marker Algorithm

The name dual-byte-marker is used because two bytes are read (line 1) from the input data at one time as shown in Figure 5. If these two bytes are 0xFFD8 (line 3), then read an 18-byte structure (line 4). If JFIF identifier is found in the 18-byte structure (line 5), then JFIF header is found (line 6). If JFIF identifier is not matched, then return the structure and let the pointer points to the starting byte of the structure (line 7-9). If the second byte read is 0xFF (contiguous 0xFFFF), the second byte will be returned to the input stream. Then positioned the pointer to the second returned byte (line 11-14) so that it can be read again in the next round.

```
1.  Do
2.    Read two-byte structure        // FIRST read
3.    If two-byte = '0xFFD8' marker then
4.      Read 18-byte structure       // SECOND read 18-byte structure
5.      If (Match JFIF identifier)
6.        Valid JFIF Header is FOUND
7.      Else
8.        Return the 18-byte structure to the input stream
9.        Position the pointer to the STARTING byte of the returned structure
10.     End-if
11.   Else if second byte of the two-byte structure = 0xFF
12.   // if contiguous 0xFFFF is found
13.     Return the 2 bytes to the input stream
14.     Position the pointer to the returned 2nd BYTE
15.   End-if
16. While not end-of-image
```

Figure 5. Dual-byte-marker Algorithm

The algorithm can be generalized as the followings:

$$\text{Let } P(x) \text{ be 0xFFD8 (SOI).}$$
$$\text{Let } Q(x) \text{ be "JFIF\0" string.}$$
$$\exists x \in Z^+, P(x) \wedge Q(x) \Leftrightarrow \text{JFIF Header is found}$$

Best case scenario:

This happens when the first read (two bytes) is $P(x)$, second read (18-byte structure) is $Q(x)$. Thus, for every two reads, there is one successful detection of JPEG header. Thus, the big O value for best case scenario is shown below.

$$\frac{n}{20} \times 2 = \mathrm{O}(n).$$

Worst case scenario:

This happens when no JFIF header is found but all are '0xFF's. Therefore, only two bytes are read every time, but every time the second byte is found to be '0xFF', the second byte will be returned. The byte will be read in the next round as the first byte of the two byte read. Thus, the big O value for worst case scenario is shown below.

$$n - 1 = \mathrm{O}(n).$$

5.3. 20-Point-Reference Algorithm

The name of the algorithm implies that a fixed 20-byte length structure is read from the input data every time. The structure is illustrated in Figure 6, while the algorithm is shown in Figure 7. Initially, the first read obtains 20-byte structure from the input data (line 2). If the first two bytes of the structure matches 0xFFD8 (SOI) and also match the string "JFIF\0" at the seventh to the eleventh byte (5 bytes), then JFIF header is found (line 3-5). Nevertheless, if mismatch is found in either SOI or JFIF identifier, then the whole structure will be returned

to the input stream, and then positioned the pointer to the second byte of the returned structure.

```
struct jpegHeader {
        u8 SOI[2];              // 0xFF 0xD8
        u8 APP0[2];             // 0xFF 0xE0
        u8 length[2];           // excludes APP0
        u8 identifier[5];       // "JFIF\0"
        u8 version[2];          // e.g. 0x01 0x02; v1.2
        u8 density_units;       // e.g. 0x01
        u8 x_density[2];        // e.g. 0x00 0x48
        u8 y_density[2];        // e.g. 0x00 0x48
        u8 thumbnail_width;     // e.g. 0x00
        u8 thumbnail_height;    // e.g. 0x00
};
```

Figure 6. Structure (20 bytes) used in 20-point-reference Algorithm

The algorithm can be generalized as

Let $P(x)$ be 0xFF 0xD8 (SOI) and identifier ("JFIF\0" string) is.

$$\exists x \in Z^{+}, P(x) \Leftrightarrow \text{JFIF Header is found}$$

```
1.  Do
2.    Read twenty-byte structure    // FIRST read
3.    If (first two byte = '0xFFD8' marker) and
4.      (the seventh-byte of the structure Match JFIF string)
5.        Valid JFIF Header is FOUND
6.    Else
7.        Return the whole read structure to input stream
8.        Position the pointer to the second-byte of the returned structure
9.    End-if
10. While not end-of-image
```

Figure 7. Algorithm for JFIF header detection using 20-point-reference Algorithm

Best case scenario:

Best case scenario occurs when the first read (20 bytes) is $P(x)$. Thus, every reads, jump twenty bytes forward when there is a successful detection of JPEG header. Thus, the big O value for best case scenario is shown below.

$$\frac{n}{20} = \mathrm{O}(n).$$

Worst case scenario:

Worst case scenario happens when no JFIF header is found. Therefore, the twenty bytes read will be returned to the input stream. The next read will start from the second

byte of the returned structure. Thus, the big O value for worst case scenario is shown below.

$$\frac{n}{2}+1=\mathrm{O}(n).$$

6. Experimentations

This section discusses on the experiment done using FORHEADER model. Three novel algorithms namely single-byte-marker, dual-byte-marker and 20-point-reference algorithms are tested using this model. These algorithms are developed using C language in Windows XP environment with Intel® dual core 1.8 MHz processor and 1 GB memory.

6.1. FORHEADER Model for JFIF Header Detection

Three algorithms are used with FORHEADER model to process the data from hard disk and memory images for detecting JFIF headers. All information about the JFIF headers found and the total time taken for the processing will be displayed on the screen for comparison. Additional tests are also carried out using data from DRFWS 2006 Challenge [16] as input to the FORHEADER using these algorithms.

6.2. Preparing Resources for Experiments using FORHEADER

Two input data are prepared for these tests. The first input is memory image taken from 1GB physical memory using Helix Live CD [19, 20, 21]. To ensure the presence of JFIF files in the image, few JFIF files are simultaneously opened prior to taking the image. Second input data is the hard disk image taken from 8MB hard disk partition using Helix Live CD. There are totally no other files in the partition except few JFIF files. 8MB is the minimum partition size that can be allocated by the partition manager. The third input data is taken from DRFWS 2006 Challenge which is downloaded from the Internet.

The input data to be used in the experiments need to be prepared and the information on the number of JFIF headers available in each of these input data should be clarified. HexAssistant is used to search for available JFIF headers in each of the input data. List of JFIF headers' offset addresses in the input data are shown in Table 3, 4 and 5. Finally, a list of resources to be used in the experiments using FORHEADER is shown below.

a. Input files
   1) *hdd.dd* - The total number of headers found are shown in Table 3 and Figures 8.
   2) *mem.dd* - The total number of headers and footers found are shown in Table 4 and Figures 9.
   3) *dfrws-2006-challenge.raw* - The total number of headers and footers found are shown in Table 5.

b. Algorithms developed using C language
   4) *single.c* – implement single-byte-marker algorithm
   5) *dual.c* – implement dual-byte-marker algorithm
   6) *point.c* – implement 20-point-reference algorithm

c. The experiment of running and recording the processing time of each algorithm against each input is done on Windows XP environment with an Intel® Dual Core 1.8 processor with 1 GB of memory. The summary of these experiments are summarized in Table 6.

Table 3. Number of Headers in *hdd.dd* of size 8MB

JPEG	Total number
header	6

There are 6 headers found in *hdd.dd* at these addresses.

Header #1 32768
Header #2 296960
Header #3 675840
Header #4 1161216
Header #5 1679360
Header #6 2344960

Figure 8. The Header Found

Table 4. Number of Headers in mem.dd

JPEG	Data read from the file						
	30MB	60MB	90MB	120MB	150MB	180MB	600MB
header	2	2	2	2	7	8	33

There are 9 headers found in *mem.dd* at these addresses:

Header #1	15,929,390
Header #2	15,929,722
Header #3	116,740,535
Header #4	119,779,120
Header #5	143,147,038
Header #6	143,147,362
Header #7	149,637,290
Header #8	156,525,259

Figure 9. The header found

Table 5. Number of headers from *DFRWS 2006 data set*

Algorithm	dfrws-2006-challenge.raw
header	19

Table 6. Summary of algorithms and input data used in experiments with FORHEADER

Experiment	Algorithm	Data Source
#1	single-byte-marker	Memory image (*mem.dd*)
	dual-byte-marker	Memory image (*mem.dd*)
	20-point-reference	Memory image (*mem.dd*)
#2	single-byte-marker	Hard Disk Image (*hdd.dd*)
	dual-byte-marker	Hard Disk Image (*hdd.dd*)
	20-point-reference	Hard Disk Image (*hdd.dd*)
#3	single-byte-marker	DFRWS 2006 Challenge data
	dual-byte-marker	DFRWS 2006 Challenge data
	20-point-reference	DFRWS 2006 Challenge data

Three experiments will be carried out according to the summary in Table 6, using three novel algorithms using FORHEADER on various input data for detecting JFIF headers.

6.3. First experiment : input data from memory image *mem.dd*

A 1GB memory image mem.dd is used in the first experiment. Each algorithm will execute this experiment on mem.dd for 6 times. Each experiment will stop the execution at the size limit of 30MB, 60MB, 90MB, 120MB, 180MB and finally at 600MB . The results of these tests are summarized in Table 7. Based on Table 7, the first experiment is done by using single-byte-marker to detect JFIF header from mem.dd and stops when it reaches the 30MB size limit. The processing time for this first test is recorded as 94.075 seconds. The final execution is done by using 20-point-reference on mem.dd and stops at the size limit of 600MB, with the processing time taken as 5256.027 seconds. From these results, dual-byte-marker clearly shows better performances for all size limits.

Table 7. Results of tests done on *mem.dd* of size 1GB

Algorithm	Data read from *mem.dd* of size					
	30MB	**60MB**	**90MB**	**120MB**	**180MB**	**600MB**
single	94.075s (1m 34s)	267.918s (4m 28s)	408.020s (6m 48s)	490.697s (8m 11s)	671.181s (11m 11s)	1959.070s (32m 39s)
dual	64.741s (1m 5s)	219.532s (3m 40s)	329.818s (5m 30s)	378.654s (6m 19s)	494.321s (8m 14s)	1365.277s (22m 45s)
20-point	262.484s (4m 22s)	590.376s (9m 50s)	858.203s (14m 18s)	1114.763s (18m 35s)	1629.573s (27m 10s)	5256.027s (87m 36s)

6.4. Second Experiment : input data from hard disk image *hdd.dd*

All three algorithms are executed using data from hard disk image *hdd.dd* according to FORHEADER model. The results are summarized in Table 8. Dual-byte-marker algorithm again showed a better performance in this experiment.

Table 8. Results of tests done on *hdd.dd* of size 8MB

Algorithm	Data read from the file
single	21.860s
dual	11.530s
20-point	69.480s (1m 9s)

6.5. Third Experiment : input data from *DFRWS 2006 Data Set*

All three algorithms are executed using *DFRWS 2006 data set*. The file *dfrws-2006-challenge.zip* of 40.6 MB is downloaded from the Internet and then extracted as *dfrws-2006-challenge.raw* of size 47.6 MB. The results are summarized in Table 9. Dual-byte-marker algorithm also showed a better performance for this experiment.

Table 9. Results of tests done on *DFRWS data set*

Algorithm	dfrws-2006-challenge.raw
single	131.330s (2m 11s)
dual	69.7s (1m 10s)
20-point	412.770s (6m 53s)

7. Result and Discussion

In the previous section, we demonstrate the comparisons of three novel algorithms namely single-byte-marker, dual-byte-marker, and 20-point-reference using FORHEADER for detecting JFIF headers according to their processing time. These algorithms are implemented using C programming language. From those results, it is clearly shown that dual-byte-marker gives better performance in all three experiments that have been carried out. All these algorithms display the same information format as shown in Figure 10 which includes SOI and JFIF header information. The screenshot of the algorithm output is shown in Figure 11. All of the algorithms have been designed to display similar output screen as shown in Figure 11.

*** ADDRESS OF HEADER 2 *** 15929716	
SOI (2 bytes)	0xFF 0xD8
APP0 (2 bytes)	0xFF 0xE0
Length (2 bytes)	0x0 0x10
Identifier (5 bytes)	0x4A 0x46 0x49 0x46 0x0
Version (2 bytes)	0x1 0x2
Density units (1 byte)	0x1
X density (2 bytes)	0x0 0x48
Y density (2 bytes)	0x0 0x48
Thumbnail width (1 byte)	0x0
Thumbnail height (1 byte)	0x0
The difference is :	15.320s

Figure 10. The general JFIF header output information for all three algorithms

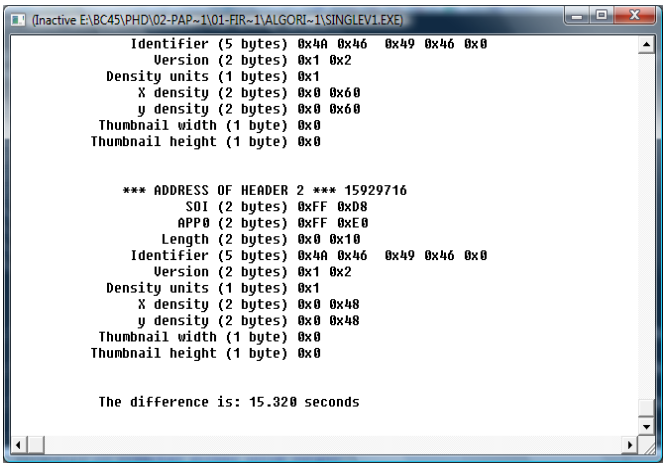

Figure 11. The screenshot of output from single-byte-marker algorithm

8. Conclusion

In this paper, we have proposed a new FORHEADER model for JFIF header detection. Three JFIF header detection algorithms namely single-byte-marker, 20-point-reference and dual-byte-marker have been compared using FORHEADER with input data from memory and hard disk images; and also from DFRWS 2006 data set. The results obtained from these

experiments clearly shown that dual-byte-marker algorithm gives better performance based on their processing time.

Acknowledgement

This paper is an extension to the papers that have been published in [22] and [23]. We also wish to note that this work was supported by Universiti Tun Hussein Onn Malaysia (UTHM).

References

[1] J.D. Isner, "Computer Forensics: An Emerging Practice in the Battle Against Cyber Crime", SANS Institute, 2003.

[2] R. Wortley, and S. Smallbone, "Child pornography on the Internet. In Problem-Oriented Guides for Police", number 41 in Problem-Specific Guides Series. US Department of Justice.

[3] R. Bassett, L. Bass, and P. O'Brien, "Computer Forensics: An Essential Ingredient for Cyber Security", Journal of Information Science and Technology, 2006, 3(1), pp. 22–32.

[4] S.L. Garfinkel, "Carving contiguous and fragmented files with fast object validation", Journal of Digital Investigation, 2007, 4, pp. 2–12.

[5] P. Anadabrata, T. Husrev, and N.M. Sencar, "Detecting file fragmentation point using sequential hypothesis testing", Journal of Digital Investigation, 2008, 5, pp. 2–13.

[6] M. Karresand, and N. Shahmeri, "Reassembly of Fragmented JPEG Images Containing Restart Markers", 2008, IEEE press.

[7] A. Pal, and N. Memon, "Evolution of file carving", IEEE Signal Processing Magazine, 2009, pp. 59–71.

[8] A. Pal, K. Shanmugasundaram, and N. Memon, "Automated Reassembly Of Fragmented Images", AFOSR Grant F49620-01-1-0243, 2003.

[9] R. Golden, G. Roussev, and L. Marzial, "In-Place File Carving", National Science Foundation under grant # CNS-0627226, 2007.

[10] M.I. Cohen, "Advanced JPEG Carving", e-Forensics, 2008.

[11] N. Memon, and A. Pal, "Automated reassembly of file fragmented images using greedy algorithms", 2006, IEEE press.

[12] R. Golden, G. Roussev, and V. Scalpel, "A Frugal, High Performance File Carver", In the Proceedings of the 2005 digital forensics research workshop, 2005.

[13] J. Haggerty, D. Liewellyn-Jones, and M. Taylor, "FORWEB: File Fingerprinting for Automated Network Forensics Investigations", 2007.

[14] I.W. Evett, and R.L. Williams, The Journal of Forensic Identification, 1996, 46 (1).

[15] L.H. Swee, "JPEG for Digital Panel", Texas Instrument, http://focus.tij.co.jp/jp/lit/an/spra664/spra664.pdf

[16] E. Hamilton, "JPEG File Interchange File Format – Version 1.02", http://www.w3.org/Graphics/JPEG/jfif3.pdf

[17] http://www.foundstone.com/us/resources/proddesc/bintext.htm

[18] http://www.verytools.com.

[19] D. Richard, Austin, "Digital Forensics on the Cheap: Teaching Forensics Using Open Source Tools", 2007, ACM press.

[20] http://www.e-fense.com/helix

[21] http://forensics.wikia.com/wiki/Helix_LiveCD

[22] K.M. Mohamad, and M.D. Mat Deris, "Single-Byte-Marker for Detecting JPEG JFIF Header Using FORIMAGE-JPEG", 5th IEEE International Joint Conference on INC, IMS and IDC (NCM 2009), Seoul, Korea, pp. 1693-1698.

[23] K.M. Mohamad, and M.D. Mustafa, "Dual-Byte-Marker for Detecting JFIF Header", 4th International Conference on Information Security and Assurance (ISA 2010). Miyazaki, Japan: CCIS # 76, Springer. pp. 27–36.

Authors

Kamaruddin Malik Mohamad received his BCS (Computer Science) degree from Acadia University, Canada in 1992. He has been working in private sectors for 10 years before joining UTM in 2002. He completed his MSc (Computer Science) from UTM in 2004 and later joined Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM) as a lecturer. Currently, he is pursuing his Ph.D. in Computer Science under the guidance of Prof. Dr. Mustafa Mat Deris. His research interests include Steganography and Digital Forensics. He has published more than 8 research publications in various International conferences, proceedings and Journal during his Ph.D study.

Tutut Herawan received his B.Ed and M.Sc degrees in Mathematics from Universitas Ahmad Dahlan and Universitas Gadjah Mada Yogyakarta, respectively. He obtained his Ph.D from Universiti Tun Hussein Onn Malaysia. Currently, he is a senior lecturer at Computer Science Program, Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang (UMP). He published more than 40 research papers in journals and conferences. His research area includes data mining and KDD, rough and soft set theories.

Mustafa Mat Deris received his B.Sc. from University Putra Malaysia, M.Sc. from University of Bradford, England and Ph.D from University Putra Malaysia. He is currently a professor in the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia (UTHM). His research interests include distributed databases, data grid, database performance issues and data mining. He has currently published more than 90 papers in journals and conference proceedings.

.