Validation Algorithms Based on Content Characters and Internal Structure: The PDF File Carving Method

Mo Chen[1] Ning Zheng[2] Ming Xu[3] Yongjian Lou[4] Xia Wang[5]
College of Computer, Hangzhou Dianzi University
Hangzhou, 310018, China
mchen@stu.hdu.edu.cn, nzheng@hdu.edu.cn, mxu@hdu.edu.cn, louyjhz@hdu.edu.cn, annewang@stu.hdu.edu.cn

***Abstract*—This paper presents a new carving method for automatically and effectively carving PDF files from an unstructured digital forensic image. The carving method has five validation algorithms based on the content characters and the internal structure of PDF files. These validation algorithms include header/file length/maximal offset of objects/footer validation, internal structure validation, entropy difference validation, zlib/deflate decompression validation and character table validation. Effectively reassembling PDF file fragments including out-of-order fragments, exactly carving PDF files without any manual intervention, lower "false positives" are the advantage of this method. The PDF file carving method and other carving applications are illustrated over real world data using the DFRWS 2007 carving challenge dataset. The results show that this method is better than other's.**

Keywords- file carving; PDF validation; content characters; internal structure; DFRWS 2007 carving challenge dataset

I. INTRODUCTION

File carving is a powerful technique, often used to recover files from unstructured original disk images. It carves files based on their content, rather than using metadata that points to the content [1]. In forensics, file carving can identify and recover interesting files from raw, deleted or damaged file system, memory, and swap space data obtaining from live information. File carving has three steps: Firstly, obtaining an original disk image from an investigative target; secondly, identifying whether the files are intact or not, dealing with the fragmented files (all fragments present, but non-contiguous) and the broken files (missing pieces); finally, examining validities of the files and copying out the finished files from the image. Nowadays, most file carving techniques share three limitations: Firstly, they are difficult of reassembling the fragments of the fragmented files; secondly, their results have high "false positives" (files were presented as intact data, but which in fact contained invalid data and could not be displayed); thirdly, they can not carve the broken files.

The paper significantly presents understanding and improving PDF file carving in three ways: Firstly, it introduces a detailed analysis of the PDF file format. Secondly, it presents a new PDF file carving method. The method includes five PDF validation algorithms based on the content characters and the internal structure of PDF files. Thirdly, it discusses the results of applying the PDF file carving method and several existing file carving applications to the DFRWS 2007 carving challenge [2].

II. RELATED WORK

Header/footer carving was the classical file carving technique. It extracted all data between possible file header and file footer. In 2005, Richard designed a frugal, high performance file carver, Scalpel [3]; Mikus researched many general file formats, introduced example internal structure of files, and extended Foremost [4]. PhotoRec [5] is frequently used to recover destructed file systems or data, can carve more intact files than others. But above carvers can not exactly carve all files, especially the fragmented and broken files. And their carving results have high "false positives". In 2006, Metz designed a carving tool, Revit [6, 7]. Revit identified a block of raw data whether belonged to a file by "file definition states". It recognized the fragments of the file based on entropy, file characteristics, etc. Revit also could not exactly recovery the fragmented and broken files. In 2007, Cohen based his work on a theory of fragments and file mapping [8]. His PDF carving process consists of three steps: Firstly, indexing all objects and cross-reference tables from the image; secondly, merging together cross-reference tables belonging to the same file; thirdly, adding identified points to the object, then parsing the result and reparsing the ambiguous region test points. Cohen's carving result had low "false positives", but it needed overmuch manual intervention, generated much redundant data, stopped consecutively when error could not be solved.

III. PDF FORMAT

A PDF file is basically a binary file which also uses ASCII tags as delimiters to describe the header, trailer, data structures and special content characters [4]. For efficient random access, a canonical PDF file initially consists of four elements (Figure 1(a)). PDF specifies a particular file organization, "Linearized PDF", its structure shows in figure 1(b). And all tags in this paper are stored in ASCII.

"%PDF" is PDF file header, and "%%EOF" is PDF file footer (Figure 1(a)).

The body of a PDF file consists of a sequence of indirect objects representing the contents of a document [9]. A labeled object is called an indirect object, and has a unique "object identifier". The object identifier has two parts: a positive integer "object number" and a non-negative integer "generation

978-0-7695-3494-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ISISE.2008.209

IEEE computer society

number". The value of the object bracketed between the keywords "obj" and "endobj" followed the identifier (Figure 1 (a)). Following the header, the first object of the linearized PDF file must be an indirect dictionary object, linearization parameter dictionary. The value of the parameter "L" shows the actual length of the entire PDF file in bytes.

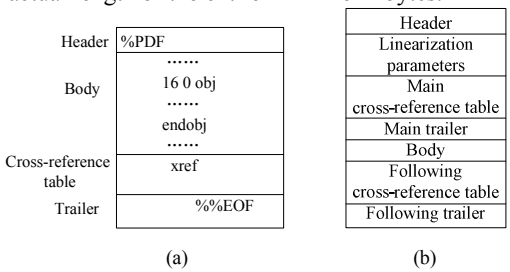

Figure 1. (a) is the initial structure of the PDF file, (b) is the structure of the linearized PDF file

The cross-reference table contains information that permits random access to indirect objects within the file so that the entire file need not be read to locate any particular object [9]. The table comprises one or more cross-reference sections. Each cross-reference section begins with a line containing the keyword "xref" (Figure 1(a)). Following this line are one or more cross-reference subsections. Each subsection begins with a line containing two numbers separated by a space: the object number of the first object in this subsection and the number of entries in the subsection [9]. Following this line are the cross-reference entries themselves, one per line. Each entry is exactly 20 bytes long, including the end-of-line marker (2-character). The eighteenth byte of each entry is the keyword "n" or "f", distinguishing an entry is in-use or free. If an entry is in-use, then the first 10 bytes denote the 10-digit byte offset of the object, otherwise the first 10 bytes denote the 10-digit object number of the next free object. And 5 bytes follow the first space denote 5-digit generation number.

The trailer of a PDF file has a "trailer dictionary", consisting of the keywords "Size" and "Prev". The value of the keyword "Size" in the trailer dictionary denotes the total number of entries in the PDF file's cross-reference table. The keyword "Prev"of the dictionary presents only if the file has more than one cross-reference section. Its value denotes the byte offset from the beginning of the file to the beginning of the previous cross-reference section.

IV. PDF FILE CARVING METHOD

Our PDF file carving method consists of five validation algorithms, based on the content characters and the internal structure of PDF files. The carving process shows in figure2:

- The disc image is first scanned by header/file length/maximal offset of objects/footer validation. The validation distinguishes whether the PDF file is linearized or not, then confirms the intact PDF file. The unconfirmed PDF files are tagged as the default fragmented PDF files.
- Internal structure validation searches the fragments belonging to the default fragmented PDF files, ascertains the files are broken or fragmented, identifies whether the object is intact or not, calculates the number of the fragments, finally confirms the exact or temporary boundaries of the fragments.
- The temporary boundaries of the fragments should be farther confirmed by surplus validation algorithms. If surplus validation can not confirm the exact boundaries, the temporary boundaries will be the final boundaries.
- The confirmed PDF files are copied out from the disc image.

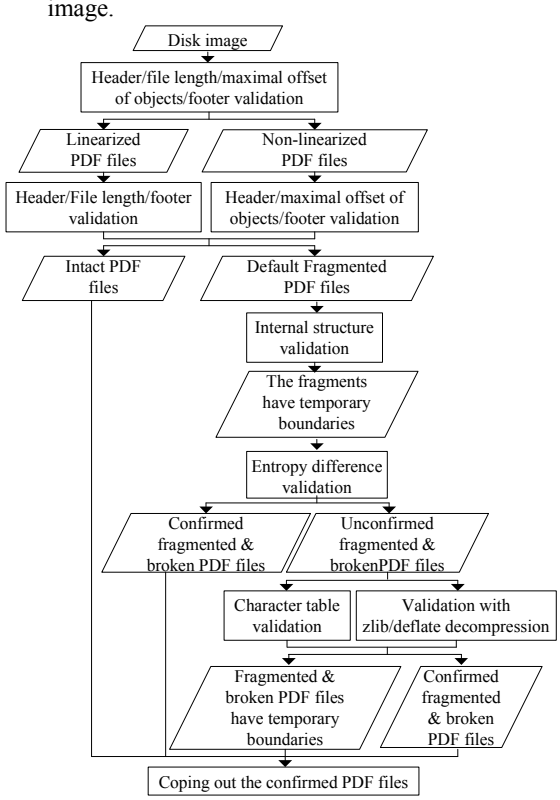

Figure 2. The new PDF file carving process

A. *Header/file length/maximal offset of objects/footer validation*

Header/file length/maximal offset of objects/footer validation first seeks the PDF file's header according to the keyword "%PDF". Then it confirms whether the PDF file is linearized or not, according to the linearization parameter dictionary. The validation uses different techniques to deal with the linearized PDF file and non-linearized PDF file:

- The linearized PDF file is dealt with header/file length/footer validation. This validation scans the linearization parameter dictionary of the linearized PDF file to seek the keyword "L". The following equation will hold when it is found.

$$f = h + l \tag{1.1}$$

where f is the end of the PDF file, h is the start of the file header, l is the value of the keyword "L". This

validation scans the end area according to *f*. If the keyword "%%EOF" is found, the PDF file will be tagged as intact file, otherwise it will be tagged as default fragmented file.

- The non-linearized PDF file is dealt with header/maximal offset of objects/footer validation. This validation firstly hypothesizes the tested PDF file is intact, so its cross-reference table is below the header. The validation scans the data following the header to seek its cross-reference table, calculates the maximal offset of objects in the cross-reference table. And the equation will hold.

$$objM = h + offM \tag{1.2}$$

where *objM* is the last object's location of the PDF file in the image, *h* is the start of the file header, *offM* is the maximal offset of objects in its cross-reference table. The validation checks the last object's location according to *objM*. If an object is found, the PDF file will be tagged as intact file, otherwise it will be tagged as default fragmented file.

Reducing "false positives" is the advantage of header/file length/maximal offset of objects/footer validation. Because it exactly finds intact PDF files, can identifies many deceptive instances. For example, the linearized PDF file will not be tagged as intact when its file length has been changed because of fragments; the non-linearized PDF file will not be tagged as intact when its sectors were inserted, deleted or modified.

B. Internal structure validation

Internal structure validation consists of two parts: intact object validation and the number of fragments validation. Intact object validation confirms whether the object is intact or not. The number of fragments validation seeks the fragments of the PDF file, ascertains the file is broken or fragmented, calculates the number of the fragments, and confirms the temporary boundaries of the fragments.

1) Intact object validation: The values of the offsets in the cross-reference table will not change when the file is fragmented (The paper hypothesizes the cross-reference table is not destroyed), so the margin of two contiguous offsets denotes the actual length of an object. Intact object validation firstly arranged the in-use entries in order, according to the offsets of the cross-reference table. Then the following equations confirm whether the tested object is intact or not.

$$\begin{cases} Len_1 = Off_2 - Off_1 & (1.3) \\ Len_2 = Toff_2 - Toff_1 & (1.4) \\ Len_1 = Len_2 & (1.5) \end{cases}$$

where Len_1 is the margin between two in-order contiguous offsets, Off_1 is the offset of the tested object, Off_2 is the offset of the next in-order contiguous object; Len_2 is the length of the tested object, $Toff_1$ is the location of the tested object in the PDF file, $Toff_2$ is the location of the next contiguous object in the PDF file. If (1.5) can not hold, the tested object is confirmed as fragmented, otherwise is tagged as intact.

2) The number of fragments validation: The validation based on intact object validation, the detailed process has three parts:

a) Pretreatment: The linearized PDF file or the updated non-linerarized PDF file has more than one cross-reference section, more than one keyword "Size"and keyword "Prev". The value of the keyword "Size" in the main cross-reference section indicates the total number of entries. The validation merges together all cross-reference sections belonging to the same PDF file in the pretreatment. For example, we hypothesize a fragmented PDF file has two cross-reference sections, then the equations (1.6) and (1.7) will hold.

$$\begin{cases} y_1 = x_2 & (1.6) \\ S = y_1 + y_2 & (1.7) \end{cases}$$

where y_1 is the number of entries in the first section, x_2 is the object number of the first object in the second section, y_2 is the number of entries in the second section, and S is the total number of entries in the cross-reference table. So if two cross-reference sections hold the two equations, then the validation will confirm the two cross-reference sections belonging to the fragmented PDF file.

b) Seeking and calculating fragments: The fragments of the PDF file (The paper hypothesizes the fragment is big enough) must have one or two fragmented object and some intact object. The validation firstly seeks the intact object below the PDF file header, if a tested object O_i is fragmented, the first fragment and the first fragmented object can be confirmed. The next objects O_{i+1} and O_{i+2} must be intact in the next fragment, so the validation uses the cross-reference table and intact object validation to find O_{i+1} and O_{i+2}. It continuously tests the integrality of the following object until next fragmented object is found or all following objects are intact. If all following objects are intact, the next fragment can be confirmed as the last fragment, otherwise the following fragment should be continuously searched by the above theory. If the following fragment can not be confirmed, then the PDF file can be tagged as broken file. The number of the fragments in the PDF file can be confirmed according to (1.8).

$$fragN = fobjN + 1 \tag{1.8}$$

where *fragN* is the number of the fragments, *fobjN* is the number of the fragmented objects.

c) Confirming temporary boundaries: The head and the foot of a fragment are calculated by (1.9) and (1.10).

$$\begin{cases} fragH = \lfloor objF / 512 \rfloor * 512 & (1.9) \\ fragF = (\lfloor objL / 512 \rfloor + 1) * 512 & (1.10) \end{cases}$$

where *fragH* is the head of the fragment, *fragF* is the foot of the fragment, *objF* is the first object in the fragment, *objL* is the last object in the fragment. *fragH* and *fragF* are the temporary boundaries of the fragment, need to be handled by surplus validations.

C. Entropy difference validation

Entropy is defined as a measure of randomness, or uncertainty, in a particular piece of information [10]. Claude E.

Shannon defined entropy as the sum of: the probabilities of each state multiplied by the log to the base 2 of those probabilities [11]. Written in formal notation we have:

$$H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i) \quad (1.11)$$

Some researchers used entropy values or entropy signature to identify different file types [6, 7, 12]. Luan [10] pointed out that the compressed or encrypted files had higher entropy values.

The PDF file has large amount of compressed data for reducing file size. These compressed data always are binary data, so their entropy values generally are higher. If a fragmented object is a compressed stream, its entropy value or entropy signature can be used to seek the boundary of the compressed data. But entropy can't exactly identify the fragments of the PDF file, for the entropy values of the different objects are different. So we design entropy difference validation to solve this problem. The equation is:

$$HD(x) = | (-\sum_{i=1}^{m} p(i) \log_2 p(i)) - (-\sum_{j=1}^{n} p(j) \log_2 p(j)) |$$

(j=i+1, n=m=256) (1.12)

where n is the number of the last 256 bytes in a sector, m is the number of the first 256 bytes in the next sector. The validation achieves good result when the entropy difference value equals to 0.8.

D. Validating with zlib/deflate decompression

"FlateDecode" is the most common filter in a PDF file decoding data that has been encoded "Flate" data compression method. The "Flate" method is based on the public-domain zlib/deflate compression method [9]. It is fully defined in Internet RFC 1950[13] and RFC 1951[14].

Validating with zlib/deflate decompression confirms whether the encoded "Flate" data are valid or not, by decoding the data of the fragmented object. The detailed process is:

1) Extracting the "Flate" data of the fragmented stream in the first fragment, $stream_1$;

2) Extracting the "Flate" data of the fragmented stream in the second fragment, $stream_2$;

3) Initializing circular number c and it equals to 0;

4) Calculating the number of the fragmented stream's sectors, $streamSN$, and the equation shows in the following:

$$streamSN = (objLen - streamL_1 - streamL_2) / 512 \quad (1.13)$$

where $objLen$ is the length of the fragmented object, $streamL_1$ is the length of the $stream_1$, $streamL_2$ is the length of the $stream_2$;

5) Connecting $stream_1$ and $stream_2$ to form the resultant data, $stream$, decompressing it; if success, the boundaries of two fragments are confirmed, going to 9), otherwise, going to 6);

6) $stream_1$ subtracts the last sector, the head of $stream_2$ augments one preceding contiguous sector, c adds 1;

7) Judging whether c surpasse $streamSN$, if surpassing, going to 8), otherwise going to 5);

8) Decompressing unsuccessful;

9) The boundaries are confirmed by the locations of $stream_1$ and $stream_2$

E. Character table validation

Character table validation initially establishes PDF characters table (It is small and needs to test more dataset for adding more characters) for matching characters in the boundaries of the fragmented PDF file's fragments. For example, if the letters "/BaseFont /JMODBM+H" appear as the last characters of a sector, the validation will know this belong to a font. For a font subset, the PostScript name of the font – value of the font's "BaseFont" entry and the font description's "FontName" entry – begins with a tag followed by a plus sign (+) [9]. The tag consists of exactly six uppercase letters, the choice of the letters is arbitrary, but different subsets in the same PDF file must have different tags [9]. The character table stores standard font names coming from Adobe font name reference table [15]. So if the letters "elveticaNeue-BoldCond" appear as the first characters of a sector, it is reasonable to assume two sectors are consecutive in the fragmented PDF file.

V. RESULTS AND DISCUSSION

This section discusses the experimental results of applying the new PDF file carving method and several existing file carving applications to the DFRWS 2007 carving challenge. We compare our method to Foremost [4], Revit [6, 7] and Cohen's PDF file carving method [8].

The DFRWS 2007 challenge creates 89 different scenarios to test specific situations that might occur in a real file system. And the scenarios have four levels, based on the difficulty of recovering the files. This dataset has 1 intact PDF file, 6 two-fragmented PDF file, 1 three-fragmented PDF file, 5 broken PDF file. It assumed that the discontinuities of files only occur on sector boundaries.

Our experiments under Linux operating system and the material configuration is 2.66 GHz Pentium 4 with 512MB of RAM, 7200 rpm 40GB+120GB drive, operating system is Fedora 5 with 2.6.20 kernel.

The comparison of four applications shows in table 1, where C denotes the successful rate of carving the intact PDF files, F denotes the successful rate of carving the fragmented PDF files according to the number of files. B denotes the successful rate of carving fragments of the broken PDF files according to the number of fragments.

1) Carving time. Foremost is the fastest, only runs 26.885s. Revit follows Foremost, runs 40.818s. The time of Cohen's PDF file carving method can't be exactly calculated, for it needs much manual interference and stops consecutively when error can not be solved. The paper only counts its right automatic carving time. The new PDF file carving method runs 2m6.202s.

2) Carving result. The new PDF file carving method carves 8 PDF file and 5 fragments, makes different file directories to store them. Cohen's PDF file carving method

carves 19 PDF files. Each PDF file generates two file types (map, map.final), some PDF files have two or three same PDF files. So it has many data redundancies. Revit carves 2 PDF files. Foremost can only carve intact PDF file, so its result has one PDF file.

3) Successful rate. The paper presents (14) is the formula of successful rate.

$$E(x) = Cn(x) / Tn * 100\% \quad (1.14)$$

where x indicates carving method, $E(x)$ indicates successful rate, $Cn(x)$ is the number of the accurate carved PDF files/fragments, Tn is the total number of the PDF files/fragments in the dataset. The new PDF file carving method is the highest, not only exactly carves the intact and fragmented PDF files (their carving successful rates achieve 100%), but also handles with the broken PDF files. Cohen's PDF file carving method is the second. It works well on the intact and fragmented PDF files, recovers 6 fragmented PDF files (one of the 6 fragmented PDF files is not exact, needs Adobe PDF reader to recover), but it can not handle with the broken PDF files. Foremost exactly carves the intact PDF file. Revit is the worst, for its carved PDF files are partial data of the PDF files.

VI. THE CARVING RESULTS OF FOUR APPLICATIONS

Carving Method	Time	Carved Result	Successful Rate
The new PDF file carving method	2m6.202s	8 PDF files 5 fragments	*C:*100% *F:* 100% *B*: 25%
Foremost	26.885s	1 PDF file	*C*: 100% *F/B*: 0%
Revit	40.818s	2 PDF files	*C:* 0% *F/B*: 0%
Cohen's PDF file carving method	>2m30.474s	19 PDF files	*C:* 100% *F:* 85.7% *B:* 0%

Taking one with another, we think that the new PDF carving method is more useful. It can identify whether PDF files are intact or not, successfully reassembles the fragments, accurately recovers PDF files without any manual intervention, has lower "false positives".

VI. CONCLUSION AND FUTURE WORK

In this paper, we introduce the information about file carving and PDF format, presents a new PDF file carving method including five validation algorithms. They based on the content characters and the internal structure of PDF files. Finally, we discuss the carving experiments of different carving applications. The results show our carving method is better than others.

Future works include three factors. The first factor is further development of the new PDF file carving method. It should be more tested, for improving its accuracy. The second factor is further research of other file types like ZIP, HTML and OLE etc. The third factor is that we will research more algorithms to help improving carving method.

ACKNOWLEDGMENT

Supported by the Natural Science Foundation of Zhejiang Province of China under Grant No.Y106176, and the science and technology search planned projects of Zhejiang Province of China under Grant No.2007C33058.

REFERENCES

[1] Simson L.Garfinkel, "Carving contiguous and fragmented files with fast object validation", Digital Investigation. Vol. 4, Supplement 1, pp. 2-12, 2007.

[2] DFRWS, "Foresics challenge" http://www.dfrws.org/, 2001.

[3] Golden G. Richard Ⅲ, Vassil Roussev, "Scalpel: A frugal, high performance file carver". Proceedings of the 2005 Digital Forensic Research Workshop. New Orleans, LA, 2005.

[4] Nicholas Mikus, "An analysis of disc carving techniques". Master's thesis. Monterey: Naval Postgraduate School, 2005.

[5] PhotoRec, http://www.cgsecurity.org/wiki/PhotoRec , 2006.

[6] Joachim Metz and Robert-Jan Mora, "Analysis of 2006 DFRWS forensic carving challenge". http://sandbox.dfrws. org/2006/mora/, 2007.

[7] Joachim Metz, Bas Kloet and Robert-Jan Mora. "Analysis of 2007 DFRWS forensic carving challenge". http://sandbox. dfrws.org/2007 /metz/, 2007.

[8] Michael Cohen, "Advanced carving techniques". Digital Investigation. Vol.4, Issues 3-4, pp.119-12, 2007.

[9] Adobe Systems Incorporated, "Portable document format version 1.7". http://www.adobe.com., 2007.

[10] Haiying Luan and Simon Mackey. "Entropy analysis". http://polya.computing.dcu.ie/wiki/index.php/Entropy_Analysis, 2006.

[11] Shannon, C. E, "A mathematical theory of communication". Bell System Technical Journal. Vol 27, pp. 379-423,623-656, 1948.

[12] Cor J. Veenman, "Statistical disk cluster classification for file carving". proceedings of the Third International Symposium on Information Assurance and Security. Manchester, UK, 2007.

[13] P. Deutsch and J-L. Gailly, "ZLIB compressed data format specification version 3.3". RFC 1950.

[14] P. Deutsch, "DEFLATE compressed data format specification version 1.3". RFC 1951.

[15] Adobe Systems Incorporated "Adobe font name reference table". http://www.adobe.com,1997.