Automated event log file recovery based on content characters and internal structure

Yongjian Lou, Peng Wang
Computer and Software Institute
Hangzhou Dianzi University
Hangzhou Zhejiang, China
louyjhz@hdu.edu.cn hd_peter@163.com

Ming Xu, Ning Zheng
Computer and software Institute
Hangzhou Dianzi University
Hangzhou Zhejiang, China
mxu@hdu.edu.cn nzheng@hdu.edu.cn

***Abstract*—Rapidly retrieving valuable information is vital in computer forensic, especially information with respect to the computer system itself. Attentions on the system information such as registry and event log have increasingly promoted the forensic researches. Event log is a very import file in computer, which contains a large amount of available information about what happened on the observed system, but current forensic tool can only repair corrupted log files and has no effect on the situation that event log file has been fragmented. To address this problem, this paper presents an algorithm which allows search for windows event log file data fragments based solely on their data contents, without the need of any meta-data. The algorithm is based on searching the signature in log file combining with computing the entropy difference between neighboring clusters. A tool was developed to automatically recover and parse the Windows NT5 (XP and 2003) event logs without any user intervention. The evaluation of our method shows an average accuracy of 82.5%, with lower false positive.**

Keywords- file fragment; signature; entropy;entropy difference; reassembly

I. INTRODUCTION

In recent years, digital forensic has been studied from many perspectives, and researchers have obtained many promising results. However, there are still some problems need to be resolved. In digital forensic, researchers always encounter difficulties while collecting the available evidences from raw disk images .A typical example may be as follows: after obtaining an original disk image from an investigative target, researcher may find that the master file table (MFT) has been overwritten due to reinstalling system deliberately or not, thus retrieving the latent evidence without supports of file system's metadata is a pivotal step among the total procedures, and additionally a principal factor for the upcoming analysis phase. File carving technology, a powerful technique often used to recover files from unstructured original disk images, is then proposed. It carves files based on their content, rather than using file system's metadata which are pointers to the contents [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. In addition, Computer forensic analysts must often reconstruct a file from the fragments left behind after the system has been reinstalled or file has been deleted manually.

Presently, most researchers focus their program on the extraction and analysis with respect to intact event log files, few are concentrating on the situations such as log files are fragmented, broken, and partially be modified. In this paper, we are interested in the situation where parts or all of the file system metadata have been lost or corrupted. Even if there are fragments, we still hope to find and identify the fragments belonged to event log file, for many special signatures which can be used are embedded throughout these fragments

The rest of this paper is organized as follows: Section 2 describes previous work on disk forensic and file carving. Section 3 discusses our methodology, and in section 4, presents results on the efficacy of our approach. In section 5, conclusion and future work are outlined.

II. RELATED WORK

In order to data carve event logs from windows platform, some tools have been developed. In 2005, Richard and Roussev designed a frugal, high performance file carver, scalpel [2], which is designed for lower memory usage and does higher performance. The Scalpel 1.60 distribution includes a Win32 binary that can be downloaded and immediately run. Kendall and Kornblum have been developing a utility named Foremost [3] since 2007, Foremost has somewhat the same features with Scalpel, they are both open source data carving utilities. Mikus did researches on many general file formats, detailedly introduced the internal structures of each file and extended Foremost [4]. PhotoRec[5]is used to recover destructed file system or data. All above carvers can carve more intact files than others, but they did no effect on the case of fragmented and broken files, Additionally, their carving results have high "false positives". William C.Calhoun and Drue Coles proposed two algorithms for predicting the type of a fragment, one is based on Fisher's liner discriminant and the other based on longest common subsequences of the fragment with various sets of test files, their results are promising[6].In 2007, Rich Murphey proposed a event log carving solution based on former researchers' work. His method is consisted of four steps: Firstly, data carve interested artifacts from the raw disk image use Scalpel; secondly, repair the carved items as needed; thirdly, validate the resulting log records by parsing all of the events record entries in the log file with the excellent free tool from

Microsoft, called Log Viewer. Lastly, for analyzing Windows log files, collate each type of log record item, filter out the duplicates which might complicate the subsequent analysis and finally provide a single chronological table. Although this method can carve log files with low "false positives", it still need much manual intervention during the repair procedure, in addition, the tool "Scalpel" Rich exploited take little account of fragmentation which commonly occurs in disk image.

III. METHODOLOGY

Before proposing our recovery approach, features of event log format are necessary to be discussed.

A. Event log file format

1) Basic storage structure

Each event log file contains a header that has a fixed size of 48 bytes, followed by a variable number of event records, and an end-of-file record which also has a fixed size of 40 bytes. The basic storage structure in the disk is illustrated in Fig. 1(a). The event log file's header is represented by the ELF_LOGFILE_HEADER structure while the trailer is represented by the ELF_EOF_RECORD structure [7]. Structure of header is composed of header-size, Signature in ASCII "LfLe", Version information, Offsets, begin-end Record numbers, Max-size etc. Structure of trailer is similar to the header', definitely differentiated by the signature in hex-code 0x11111111222222223333333344444444, and has a duplicate set of the offsets similar to the header. Both the header and trailer structures are written in the event log when the event log file is created, they are updated each time when a new event is added to the log. Besides the header and trailer, there are a large amount of log records amply describing each event happened during the computer was running, such as Event ID, record length, time stamps, event source, description, hostname etc. Members of storage structure for header, trailer, and record can be found on Microsoft MSDN[8] in detail.

There is a 4-byte data item that immediately follows Max-size, it is used to indicate the status of the event log, we call it FLAG. When the FLAG equals to 0x0001(denoted as ELF_LOGFILE_HEADER_DIRTY), it indicates that records have been written into the log file, but the event log file has not been properly closed.

There is a critical data item in the event log record, i.e. record number, which denotes the location of a record in the logical file. In this paper, it is a major factor for us to accurately reconstruct the file from the fragments distributed in the disk.

2) Ways of organization

The event records may be organized in two ways: On one side, the oldest record immediately follows the event log header and new records are added just behind the last added record, ultimately end with the trailer. The situation can be defined as non-wrapping, it occurs after the event log is created or its data is cleared by logging service. An event log continues to be non-wrapping until its size reaches the Max value limited in the header's configuration or the amount of the system resource. On the other side, when the event log size reaches the limitation, it might start to wrap. Wrapping is controlled by the Retention configuration value. When system configuration permits logging service to rewrite the log file, new event will be appended just behind the header, followed directly by the trailer, overwriting one or more former oldest event records. So the offsets of both oldest and newest event records changed dynamically while new events are appended. Detailed illustrations for non-wrapping and wrapping are showed in Figure 1. In Figure 1(b), the former (m-1) log records has been overwritten by the newly added events.

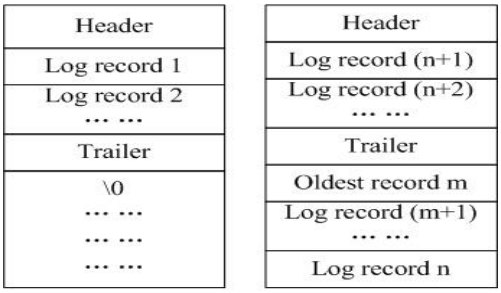

Figure1. (a) non-wrapping (b) wrapping (m<n)

B. Recovery algorithm

In the NTFS file system, a cluster is the smallest logical unit that can be allocated in file system. Theoretically, cluster size ranges from 1k to 128k taking 2 as common ratio, which is multiple of traditional sector size (512 byte). In our research, disk cluster is 4k-size, a cluster consists of 8 sectors. By observation, a file's content is allocated from the beginning of a cluster, i.e. from the beginning of a sector. A file can contain multiple clusters, and a cluster can only belong to one file. The clusters besides cluster boundaries should either belong to one identical file or two different file.

a) Identify the fragment point

Via observation we found that windows event log file is stored with many fragments, in addition, some of the fragments are out of order. This is greatly increased the difficulty of recovery.

In this paper, the block boundary is defined as the point between the neighboring blocks, a block default to contain 512 bytes. Additionally, we presume the cluster size as 4k (8 sectors). The approach presented in this paper is contributed to find the fragments of event log file and reassemble them together without the support of file system's MFT. Fragment point validation algorithm is described as follows:

Algorithm 1: Identify fragments point

1) Set file-read pointer to the beginning of the disk-image.
2) Scan the image sequentially and search for the *.evt file's special magic signature "LfLe".
3) When the signature "LfLe" is first found, validate whether the characters "LfLe" belong to a log record, or a log header, or a log trailer exploiting the acknowledged

internal structure. If one of the above log components is identified, this process is considered success, go to step4. Conversely, go to step2 and continue.

4) Move file-read pointer backward and calculate the start offset of the sector to which the signature belongs. We define the resulted sector as S. According to log file's format and inherent relationship between sector and cluster, we can predicatively deem that S is the start sector of a cluster.

5) Start with sector S, calculate entropy value of every latter clusters. Because different types of files' block content are not the same, the entropy value of block can be used to differentiate log files with other candidates. For the reason that the character's value in each block varies from 0 to 255, entropy of block's content is a discrete random variable. We compute the entropy by the equation below. P_i represents the frequency of each kind of character range from 0 to 255 in block's content.

$$\text{Entropy} = -\sum_{i=0}^{255} p_i \times \log_2 p_i \qquad (1.1)$$

According to the storage feature of log file, fragments of log file are stored in units of cluster, entropy values of sector blocks are close to each other, only computing entropy can't exactly identify the fragment boundaries. So we can calculate entropy difference of neighboring clusters by twos, entropy difference is designed as nether equation: E_i represents the entropy value of cluster j, E_i represents the entropy value of the cluster j-1.

$$ED = \left| E_j - E_i \right| \quad (j = i+1) \qquad (1.2)$$

Experimentally, log file blocks typically have entropy difference values in a stable scope. Thus we define a threshold T, once entropy difference is greater than T, the cluster boundary (i.e. block boundary) can be temporally considered as the file fragment boundary.

6) Write the location scope of the fragmented data block (from sector S to the fragment boundary) into a two-dimensional array "FragBound", storing the start-sector number and end-sector number respectively for each fragment.

7) Repeat step2 unless the file-read pointer reaches the end of the image.

Figure 2 illustrates the flow of algorithm1 vividly.

b) Reassembly

After all fragments in the disk have been identified, following work is that we need to check every fragment and try to make sense of the original order of the fragments. In the following, the brief algorithm of determining fragments order based on log file internal format is given.

Algorithm 2: Determine fragments order based on log file internal format

Step 1: Circularly read the each fragments' boundary information from the array "FragBound".

Step 2: For each fragment, check the existence of EvtHeader or EvtTrailer.

If (EvtHeader is found**) {**Compute the first record ID and last record ID which are followed the EvtHeader ;

} else if (EvtTrailer is found**) {**Compute first found record ID in the front of the fragment, and the last event record ID in the end of the fragment.

} else{Compute the first record ID and last record ID in the fragment. **}**

If (EvtHeader is found **&&** EvtTrailer is found**) {**

An intact log file is confirmed;

Continue to check the latter fragments;**}**

Store the first-last record ID into the fragment's corresponding array "RecIDs", which is also a two-dimensional array.

Step 3: Compare the record IDs, and determine the actual order of the fragments. Meanwhile, ordering of logical sector Number in array "FragBound" can be confirmed.

e.g. If the event record IDs in frag1 is from 0x07EE to 0x090A, the ones in frag2 is from 0x090B to 0x0A2C, then the two fragments can be confirmed as in the same log file and the collocate order is also confirmed.

Step 4: Reconstruction

Extract the fragments belongs to one identical log file according to the fragments' logical ordering, and store them into a new file, i.e. the recovered file.

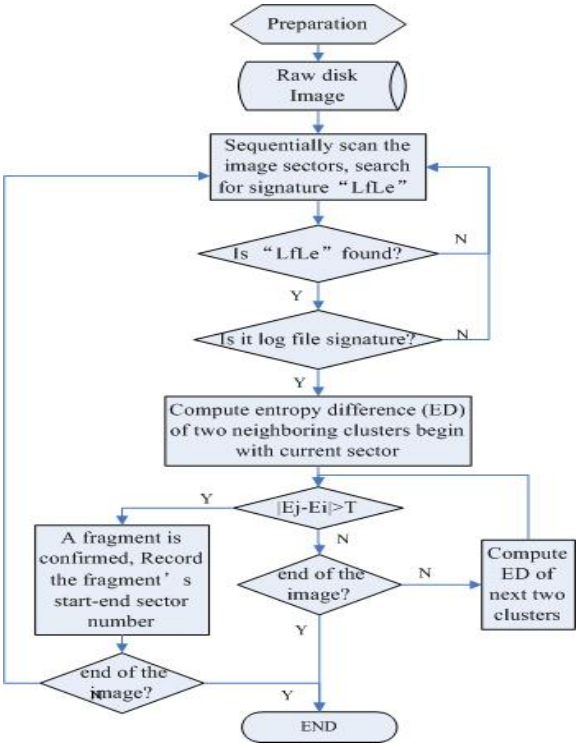

Figure2. Fragment identification algorithm

c) Validation and parse the recovered files

According to file format discussions in the section 3.1, before parsing the recovered file, we must assure that it has a valid offset to locate the first record in the log. Thus additional process should be done to make sure that the event log header and trailer are identical, and the flag byte is "clean", then the recovered image file is parsed by a tool developed by ourselves called Evt-Parser, which functioned identical to Log Viewer.

IV. RESULT AND DISCUSSION

a) Preparation

In order to test our method's availability, disk image should be acquired for examination. Our experiment is done under Linux operating system. In one scenario(scenario1), we mount a 8G-size disk image to the Linux file system, this image file is created manually by the disk tool "WinHex", so image's start cluster and end cluster can be predefined. In order to retrieve the most log files, we set the size of the image almost equals to the size of the system partition. This image contains AppEvent.evt with 8 disorder fragments, SecEvent.evt with 155 disorder fragments, and SysEvent.evt with 9 disorder fragments. In another scenario(scenario2), we mount another disk image, which is a copy of the system partition of a newly installed windows xp operation system. In this paper, we assume that each image size is multiple of cluster's size and clusters in each image are intact.

b) Experimental results

We have developed a tool called EvtCarver, which is a command line application that perform the carve method we proposed in a single pass with no user intervention. To evaluate our approach's availability, two criteria are exploited: carved result, accuracy. Table 1 describes the result in detail.

TABLE 1 CARVING RESULT OF OUR METHOD

Scenario	Raw Image fragments	Carved result	Accu-racy
Scenario1	172 fragments of 3 kind of evt files	3 files(161 fragments) 2 unmatched Fragments	93%
Scenario2	22 fragments	1 intact file(1 fragment) 1 file (15 fragments)	72%

In scenario1, we can see that the result is promising, 3 kinds of event log files contain 161 fragments: a 8-fragmented file, a 9-fragmented file and a file contains 144 fragments, 11 fragments can not be correctly reassembled. 2 additional unmatched fragments are found, we think the reason is that some other file's internal structure is similar to event log file, so our tool falsely deem them as the required fragment, but they can never be matched with any fragment during the reassembly process.

In scenario2, one intact file is found, this file is contiguously stored in a single fragment with about 128KB size. Only 1 fragmented file was recovered, it might due to that the threshold T in algothrim1 can not exactly differentiate the cluster boundary between two different files, as a matter of fact, the last cluster's entropy in the former file is quite similar to the first cluster's entropy in the latter file. In addition, in algorithm1, we exploited the method of searching the signature "LfLe" to locate the start sector of a cluster. If "LfLe" is found just in the start sector of a cluster, algorithm1 will perform better on finding correct start-end sector of a fragment. Contrarily, it will not show an ideal performance, so farther validation needs to be done for more accurate start-end sector.

Additionally, we exploit professional tool EasyRecovery[9] on the test volume, but only few intact files in consecutive clusters can be recovered, there are also a lot of false positives, reassembling fragmented file still need humanly intervention.

V. CONCLUSION

In this paper, we introduce the related work on file carving and the event log file format, and present an event log file carving method containing 2 algorithms which are based on the content characters and internal structure of event log files. Experiments are done to test our method, the result is promising. But this algorithm still has some weak points, such as the limitation that the disk image should be created with intact 4K-size cluster; the fragment boundary should occur at the cluster boundary; a more proper entropy difference threshold T need to be defined, etc.

In future, a more effective algorithm should be presented to accurately identify the log file's fragment boundary. The re-assembly algorithm should be improved not only according to the start-end event number of a fragment but also according to the time stamps, e.g. time stamps can also be used to confirm the ordering of two candidate fragments.

Our approach is contributive in improving the efficiency of disk forensic, which is not based on file system's information. We hope this method can be applied by forensic investigator to mine more valuable information from the log file remained in the corrupted disk and further analysis can be performed.

ACKNOWLEDGMENT: Supported by the Natural Science Foundation of Zhejiang Province of China under Grant No.Y106176, Y106427, and the science and technology search planned projects under Grant No.2007C33058, 2008C21075.

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] 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.

[3] http://jessekornblum.com/kornblum-cv.pdf

[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] William C. Calhoun, Drue coles, "Predicting the types of file fragments", Digital Investigation. S14-S20

[7] Event log file format. http://msdn.microsoft.com/en-us/library/bb309026(VS.85).aspx

[8] ELF_LOGFILE_HEADER structure. http://msdn.microsoft.com/en-us/library/bb309024(VS.85).aspx.

[9] http://download.cnet.com/EasyRecovery-Professional/3000-2242437386.html