

Advanced Data Carving

A paper by S/A Daniel Dickerman, SCERS, GSEC
daniel.dickerman@ci.irs.gov

of the IRS Criminal Investigation, Electronic Crimes Program

Submission for the DFRWS 2006 Data Carving Challenge

July 2006

Introduction.....	1
Process of Elimination	2
Zip Files	2
MS Compound Document Files.....	3
JPEG Files.....	5
HTML files and Text Files	6

Introduction

Data Carving is a technique used in the field of Computer Forensics when data can not be identified or extracted from media by “normal” means due to the fact that the desired data no longer has file system allocation information available to identify the sectors or clusters that belong to the file or data.

Currently the most popular method of Data Carving involves the search through raw data for the file signature(s) of the file types you wish to find and carve out. Since the file system has no information on the size of the file being carved, the current methods involve specifying a block size of data to “carve” upon finding the desired signature.

This current method relies on some assumptions: 1) that the beginning of the file, which is where the signature resides, is still present; 2) the signature you are searching for is not so common that you would find the string of characters in many other files, thereby creating many “false hits”; and 3) that the files identified through the signature search are contiguous and not fragmented.

In addition to the issue listed in the previous paragraph, the current Data Carving methods also rely on the user making adjustments to the “block size” they are carving out for a specific file signature. As files are identified through a search, the files are typically manually reviewed by opening in a program capable of viewing the specified file type. This manual review gives the examiner an idea if they need to “carve” a larger or smaller block of data for a given file in order to carve the file in its entirety.

This current process is not optimal, as it relies on guess work and a lot of trial and error on the part of the forensic examiner.

In this paper, submitted for the 2006 DFRWS Data Carving Challenge, I will look at the process of Advanced (Smart) Data Carving, which removes the “guess work” when carving certain compound file formats that contain information about the size and layout of the file in question, regardless of the existence of file system allocation information for the file.

The below documents, detailing the various file format specifications, were used to manually carve all files, listed on pages 1-2 of this submission, from the file "dfrws-2006-challenge.raw."

X-Ways Forensics was the tool I used to manually carve and hash all files.

<http://www.x-ways.net/forensics/index-m.html>

Office Document File Format Specification

<http://sc.openoffice.org/compdocfileformat.pdf>

Exif/Jpg File Format Specification

<http://www.media.mit.edu/pia/Research/deepview/exif.html>

Zip File Format Specification

http://www.pkware.com/business_and_developers/developer/popups/appnote.txt

Process of Elimination

In order to minimize the possible sectors that could make up your carved files, a process of elimination is used to identify which sectors DO NOT belong to the file(s) in question. To accomplish this “process of elimination”, as we go through the individual processes described below for carving each compound file format, you bookmark or otherwise reserve the sectors (or bytes) that you have determined to definitively belong to a specific files you have carved. Once sectors (or individual bytes) are bookmarked or reserved, they are no longer utilized and ignored by subsequent carving processes.

Zip Files

The first compound file format that we will look are Zip files, as specified in the document “APPNOTE.TXT - .ZIP File Format Specification”, revision date January 6, 2006 from PKWARE, Inc. For complete details of the file format specification, please refer to the hyperlink to the document, listed on page 1. The information described below applies to most common Zip files created with current versions of Zip archive utilities, such as WinZip.

A Zip file is broken into specific parts that can be searched for and identified based on separate signatures. The basic layout of a Zip file is first the individual compressed files within the archive. These individual files are known as “local files” and start with a local file’s decryption header of “50 4B 03 04”, followed by the file data for the compressed local file and then followed by a data descriptor, which can be identified by the signature “50 4B 07 08”. This sequence of decryption header followed by file data, followed by data descriptor continues for each local file within the archive. “The decryption header will contain the value of the local file’s compressed file size, which includes the bytes of the decryption header, unless bit 3 of a 2-byte general purpose flag located at offset 0x06 in the decryption header is set. If this bit is set, then the compressed size is stored in the “data descriptor” that immediately follows the local file’s data, and is also stored in a central directory record for the local file, as part of the central directory located that is after all individual local files in the archive.

The central directory at the end of each Zip archive can be identified by searching for the signature “50 4B 01 02”, which identifies the beginning of each central directory record contained within the central directory. And lastly, the signature “50 4B 05 06” identifies the “End of the Central Directory Record”, which identifies the size in bytes of the central directory and it’s starting offset location in relation to the beginning of the first local file decryption header in the archive.

Upon identifying the signature “50 4B 05 06”, and using the size and starting offset information in the “End of Central Directory Record”, you search backwards from the beginning of the “50 4B 05 06” the correct number of bytes (directory size + starting offset) and determine if that leaves you at the signature “50 4B 03 04”, which is the beginning of the first local file and the start of the archive.

The same search can also be performed in a forward manner, starting at the first “50 4B 03 04” you find and searching forward to the first “50 4B 05 06” you find and comparing the distance between the two with the result of the directory size + starting offset, located at offset 0x0C of the “End of Central Directory Record”.

If the location of the “End of Central Directory Record” is at a further offset than your calculation, then you have a fragmented archive file. The difference between the actual location and your

calculation is the size of the fragmented block of data that doesn't belong to the archive file. The next step is determining where the fragment occurs and distinguishing between the archive data and the fragment(s) that don't belong to the file.

To do this we next look at the data descriptor, if present, at the end of each local file in the archive, or the individual central directory records for each local file in the central directory. The compressed size of the local file, which includes the size of the decryption header for the local file, is located at offset 0x14 of each individual central directory record, which starts with the signature "50 4B 01 02."

Once you have determined the starting point of each local file in the archive, from its signature "50 4B 03 04" and you have determined the length of the local file from either the data descriptor at the end of the local file or from the length stored in its central directory record at the end of the archive, you can now determine which individual local file(s) contain the portion of the overall archive that is fragmented.

Starting from the first local file decryption header and going forward by the "size of compressed file" found in either of the two above locations, we should find the start of the next local file decryption header. If this brings you to the start of the next decryption header then this first local file is not fragmented. Continue with this method until there is a difference between the expected start of the next local file decryption header and the ACTUAL start of the specific local file decryption header. The size of the difference is the amount of fragmentation that has occurred. This difference is compared with the overall difference noted earlier between the overall size of the archive and the location of the "End of Central Directory Record" to determine if this is the entire amount of fragmentation within the archive or if more instances of fragmentation exist in another of the local files in the archive.

Once all individual local files in the archive, that contain fragmentation, are identified, and the size of the fragmentation is noted, you now review sectors of the fragmented local files for a block of data the size of the identified fragment that doesn't belong. This can sometimes be more difficult to determine than other times, depending on the type of the fragmented data.

MS Compound Document Files

(includes documents, spreadsheets, templates and other MS office files)

Next we will look at carving MS Compound Document (and spreadsheet) files, as specified in the document "OpenOffice.org's Documentation of the Microsoft Compound Document File Format." For complete details of the file format specification, please refer to the hyperlink to the document, listed on page 1 of this paper.

As quoted from the above referenced document, "Compound document files are used to structure the contents of a document in the file. It is possible to divide the data into several *streams*, and to store these streams in different *storages* in the file. This way compound document files support a complete file system inside the file, the streams are like files in a real file system, and the storages are like sub-directories."

All streams of a compound document file are divided into sectors. Sectors may contain internal control data of the compound document or parts of the user data. The entire file consists of a compound document header and a list of all sectors following the header.. The size of the sectors can be set in the header and is fixed for all sectors then.

Example:

```
HEADER
SECTOR 0
```

SECTOR 1
SECTOR 2
SECTOR 3
SECTOR 4
SECTOR 5
SECTOR 6
...and so on...

As we discussed in the section on Zip files, if you know what you are looking for, and where you expect to find it within the file, you can determine exactly what data belongs to the file in question and whether or not there is fragmented data within the file.

We start by searching for the Compound Document Header, "D0 CF 11 E0 A1 B1 1A E1," to identify the beginning of each of the MS compound documents. Next, at offset 0x1E from the beginning of the header we find a 2-byte value that identifies the sector size used in the document, which is usually 512-bytes/sector. Now, knowing the size of each sector that makes up the file, we can start looking for document structures and where within the file they should be located. As noted in the Zip file process mentioned earlier in this paper, the difference between the EXPECTED location of a structure and its ACTUAL location is the size of the fragmented data that doesn't belong to the file.

At file offset 0x2C, we find the # of sectors used by the Sector Allocation Table (SAT). Next, at file offset 0x30 we find the starting sector number (within the file) of the file's Directory. Another important file structure is the Short-Sector Allocation Table (SSAT), whose starting sector # is located at file offset 0x3C, followed by the number of sectors making up the SSAT, located at file offset 0x40. Not all compound documents utilize a SSAT, in which case you can ignore these 8 bytes. And lastly, we look at the Master Sector Allocation Table (MSAT), whose starting sector # is located at file offset 0x44, followed by the number of sectors making up the MSAT, located at file offset 0x48. The following 436 bytes of data, which make up the rest of the first 512 bytes of the compound document file, contain the first 109 sector IDs (SID) of the MSAT and starts at file offset 04C.

So, now that you know where certain items should be located, the next step is to locate them on the disk and find out if they are located at the expected sector number in relation to the start of the document.

First, using the first sector of the MSAT from the 4-byte value at offset 0x4C, search for "01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00" to find the beginning of the MSAT and compare the sector number you find the MSAT located at with the results of the sector # of the start of the document plus the 4-byte value at offset 0x4C. If there is a difference, then a fragmentation occurs before the start of the MSAT.

Secondly, search forward for the beginning of the Directory, starting from the document's header. The signature for the start of the Directory is "52 00 6F 00 6F 00 74 00 20 00 45 00 6E 00 74 00 72 00 79 00" (or "Root Entry" in case sensitive Unicode). There may be left over instances of previous Directory Entries from previous file edits, so look for more than one instance of the "Root Entry". Once you find the sector # of the start of the Directory, subtract the sector # of the start of the document, and compare the result against the 4-byte value at file offset 0x30. If the result matches your 4-byte value then no fragmentation exists between the start of the file and the Directory. If there is a difference, the difference is the amount of fragmented data that doesn't belong to the document.

And lastly, review of the individual Directory Entries for the starting sector numbers and stream size of the objects will assist in determining where, before or after each object, any file fragmentation occurs.

The largest object within the compound document is most likely the "WordDocument" object, or "Workbook" object for spreadsheets. Which means that if fragmentation exists within a large compound document, it is likely that the fragmentation occurs within those streams. As was mentioned earlier, through a process of elimination and/or manual review of the carved block for a block of data the size of your determined fragment for data that doesn't belong to the document.

The directory is an array of directory entries. Each directory entry is a 128-byte entry and is listed in order of their appearance in the document. It identifies the starting sector # of that file object, at directory entry offset 0x74 and the size of that object (in bytes) at offset 0x78.

JPEG Files

Next we will look at carving JPEG graphic files, as specified in the document "Description of Exif file format." For complete details of the file format specification, please refer to the hyperlink to the document, listed on page 1 of this paper.

The JPEG graphic file starts with a Start of Image (SOI) signature of "FF D8". Following the SOI are a series of "Marker" blocks of data used for file information. Each of these "Markers" begin with a signature "FF XX", where "XX" identifies the type of marker. The 2 bytes following each marker header is the size of the marker data. The marker data immediately follows the size and then the next marker header "FF XX" immediately follows the previous marker data. There is no standard as to how many markers will exist, but following the markers, the signature "FF DA" indicates the "Start of Stream" marker. The SOS marker is followed by a 2-byte value of the size of the SOS data and is immediately followed by the Image stream that makes up the graphic. The end of the image stream is marked by the signature "FF D9".

In the event that a thumbnail graphic exists within the file, the thumbnail graphic will have the exact same components as the full-size graphic, with "FF D8" indicating the start of the thumbnail and "FF D9", indicating the end of the thumbnail. Since thumbnails are significantly smaller and less likely to experience fragmentation than their larger parent full-size graphic, they can be used as a comparison tool for evaluating what the entire jpeg graphic is supposed to look like, in the event you must do a manual visual review of the carved graphic.

By searching first for all locations of the "FF D8 FF" signature, you identify the beginning of each jpeg graphic. The reason for searching for "FF D8 FF" is that there are different versions of jpeg files, some that start with "FF D8 FF E0" and some with "FF D8 FF E1", and leaving off the 4th byte in your signature will catch all instances, but may result in some false hits.

Rather than carve a specific length of data, in this case we will start at the beginning signature and carve until we find "FF D9". In the event of a non-fragmented jpeg graphic, without a thumbnail, this will carve the whole file. If we slightly modify our logic, by including a "if "FF D8" occurs again before "FF D9", then carve to the 2nd instance of "FF D9"" statement in our search for jpegs, then we will carve entire files including their thumbnail as long as they are not fragmented. Without this "if" logic, the first search would stop carving at the end of the thumbnail and result in an invalid jpeg. In the event of a fragmented jpeg file, the above carving method results in either a partial jpeg file or a complete jpeg file that contains extraneous data in the middle of it.

After carving all jpeg files based on these rules, we next quickly review which carved jpeg files are complete, versus which ones are fragmented and need further analysis. By carving all jpeg files to a folder, you next add that folder to your forensic tool that has partial graphic file viewing capabilities, such as the "Outside In" viewer that is built into many existing forensic tools. Using a gallery view, you can quickly identify which files are not displaying properly, only showing a partial file, and require further analysis.

Once all fragmented or partial jpegs are identified, manual visual inspection of each of these files was used to determine at what point the fragmentation occurred. This was done by approximating the percentage of the file that displayed correctly in the viewer before displaying corruptly. The raw data of the carved file was then reviewed at the data at that percentage of the file to attempt to identify where the valid graphic data ended. For this process it was assumed that the extraneous data started at an offset that was a multiple of 512-bytes from the beginning of the file. Once the extraneous data was identified, it was then removed from the partial jpeg and re-evaluated as possible sector data for other fragmented files that had previously been identified.

HTML files and Text Files

After all known compound file formats have been carved, their sectors are bookmarked and removed from consideration as possibly belonging to text, HTML or any other files. Using the "gather text" feature of X-Ways Forensics (or similar feature from a variety of existing forensic tools), text was extracted from the remaining sectors not bookmarked.

All .html and .txt files were manually carved and evaluated since no compound file format exists, identifying start, end, or location of structures within the file(s). Any fragmented text or .html files were manually put back together based on manual review of the content of the files.

If specific details are required, on any of the carved files in this data carving challenge, they will be provided upon request.

Appendix A (2006 DFRWS Data Carving Challenge Results)

Date Submitted: 10-Jul-06
 Submitted by: S/A Daniel Dickerman, SCERS, GSEC
 Agency: IRS Criminal Investigation, Electronic Crimes Program
 Phone: (401) 826-4731
 Email: daniel.dickerman@ci.irs.gov
 Submitted to: dfrws2006-challenge@dfrws.org

* - fragmented file
 ** - uncompressed (note: MD5s of files contained in an archive are the MD5 of the uncompressed file)
 *** - incomplete file

File Type	File Name (if known)	File Size (in bytes)	MD5	Sectors/offset(in HEX)
.zip		147150	EBABDE39BA44D38888DD82606980498A	28439/0 - 28726/CD
.jpg (compressed in a .zip archive)	4n6rodeo3-fix copy.jpg	62535	**63205) A39BBF1156E0BC8EDE80A46A56E6629*	28439/0 - 28561/9A 28561/9B - 28561/D1
	__MACOSX/			
	__MACOSX/._4n6rodeo3-fix copy.jpg	35	**82) A324C1F4959949CC9191E5B19B61BF1E	28561/D2 - 28561/153
.jpg (compressed in a .zip archive)	4n6rodeo4-fix copy.jpg	83701	**83818) 7DA4A0278B401372BE8641574D294C0/	28561/154 - 28725/9C
	__MACOSX/._4n6rodeo4-fix copy.jpg	35	**82) A324C1F4959949CC9191E5B19B61BF1E	28725/9D - 28725/11E
.zip		1163745	9A4C2D3A9BD203EB39C9F954A3C997E4	*28729/0 - 29528/200 & 29896/0 - 31368/1E0
.jpg (compressed in a .zip archive)	file1.jpg	488529	**513027) 5F12CEDA780FB1CAD7B6282D080542D5E	*28729/0 - 29528/200 & 29896/0 - 30050/98
.jpg (compressed in a .zip archive)	file2.jpg	674918	**725993) 924A56907A2A5E7F8E0D50542EFFB46C	30050/99 - 31386/144
.zip		270181	F940FCC37C82E8FF1431E5C3C061611E	*45015/0 - 45386/200 & 45390/0 - 45545/164
.jpg (compressed in a .zip archive)	1993-01-a-large_web.jpg	62104	**62179) B7BE0DCD06CC072041E50E0DDB5217F1	45015/0 - 45136/EC
.jpg (compressed in a .zip archive)	1995-45-a-large_web.jpg	37045	**37157) 1029EE7BE2E0F5853DDAF2DD48989345	45136/ED - 45208/1F6
.jpg (compressed in a .zip archive)	2000-06-a-large_web.jpg	44761	**44808) 90E82ABB4668FF61C4B4A0FDD53CD9C7	45208/1F7 - 45296/124
.jpg (compressed in a .zip archive)	2000-07-a-large_web.jpg	27158	**28740) C07759FAB56D4932AD26B5B46C6E688C	45296/125 - 45349/18F
.jpg (compressed in a .zip archive)	2002-29-a-large_web.jpg	42578	**42629) 614D985DE5803509E76A1D649E41B91E	*45349/190 - 45386/200 & 45390/0 - 45436/3E
.jpg (compressed in a .zip archive)	2005-02-e-large_web.jpg	55517	**55610) 87842289A512182A0FEBD8643F839991	45436/37 - 45544/168
.html		4608	EC89111E45DA8265B641655D0F68725E	0/0 - 8/200
.html		18148	C6723E21287B076EFA5B994D9DB1A2A	9/0 - 44/E3
.html		18426	8A1AC28FE7BD144B8230B38B284A3827	*4436/0 - 4455/200 & 4486/0 - 4501/1FA
.html		43236	80E6B9221EF308FF1639FD19DF036E34	*4456/0 - 4485/200 & 4502/0 - 4556/E3
.html		168527	A80EE062AED8279304FAAE8F20F6D48E	*27496/0 - 27606/200 & 27978/0 - 28196/4E
.html		187795	643D159339BD2C9E7604E6FAC8E7FACC	29529/0 - 29895/192
.html		20020	3976515D6CD4015826D9503F00C96725	*28244/0 - 28245/200 & 28307/0 - 28344/33
.xls		869888	03D1DEFF4774C932358D3580A3BBAE66	*2051/0 - 3050/200 & 3072/0 - 3770/200
.doc		450048	8D2A9A284E078805ADA47DB191F35244	*7964/0 - 8284/200 & 9474/0 - 10031/200
.doc		287232	0E52E75029E99CD2E9DCD0AF271CF4A2	32837/0 - 33397/200
.doc		943616	D7FF92B8CC1C89C46A78288B9C673152	*34288/0 - 34306/200 & 34413/0 - 36236/200
.doc		1667584	4A22F04B097920D11FFF4E192E0667A4	*36998/0 - 37649/200 & 37727/0 - 39427/200 & 39477/0 - 40380/200
.doc		71680	109284CC5ABDDC83879A29785795FD75	45964/0 - 46103/200
.jpg		287186	DAF4205574ABD6919B10CA8BE92D17A3	3868/0 - 4428/1D1
.jpg	thumbnail	5551	C1D7C67B0E81F902A2C7ED56651FEE2B	3868/14C - 3879/FA
.jpg	thumbnail	5551	C1D7C67B0E81F902A2C7ED56651FEE2B	3897/B3 - 3908/61
.jpg		608703	4EFC6C572683878EFD8F3404DDADED7B	8285/0 - 9473/1BE
.jpg		190720	7B07320709E0CAA947663F5DF3A0A390	*11619/0 - 11822/200 & 11849/0 - 12017/FF
.jpg	thumbnail	1789	B3F4CCFB61790AB745B39D3945C5FE3F	11619/1AE - 11623/AA
.jpg		7113968	B070BEAE1606F67A342BC5F78C29C743	12222/0 - 26116/EF
.jpg		189534	FE7E7AC67709F2D9C2483AA98C681B99	27607/0 - 27977/5D
.jpg		98354	DB89684C177168036E274140ECF766A1	*31475/0 - 31532/200 & 31753/0 - 31887/31

Appendix A (2006 DFRWS Data Carving Challenge Results)

File Type	File Name (if known)	File Size (in bytes)	MD5	Sectors/offset(in HEX)
.jpg		188693	0915313E99AF0F6BF13BC06BCD003113	*31533/0 - 31752/200 & 31888/0 - 32036/114
.jpg		178659	2FAE8770CC013D22E9EA1C070F2F509B	36292/0 - 36640/1E2
.jpg		297984	2A79E07559EB0FBD4E4F8A95EEAFA2A6	40638/0 - 41219/200
.jpg		1021085	7CCE072E518FD72484C97ADB1B4BE08E	*41611/0 - 43433/200 & 44029/0 - 44200/9C
.jpg		304413	C0DA37B3F1A07AF790E6E9171CEDC4D2	43434/0 - 44028/11C
.jpg		573499	2320FE9C41EADDB864A56C2DDC4DD186	*45566/0 - 45963/200 & 46104/0 - 46826/3A
.jpg	thumbnail	6600	14F703660BDD2798CB08C79EE8597212	45566/D0 - 45579/97
.jpg		24538540	DB32B271506B2F4974791957627C61CC	46910/0 - 94836/1AB
.jpg		924877	1A5A843000EF617AF93A9CAD645E3CDF	*94846/0 - 95628/200 & 95630/0 - 96653/CC
.jpg	thumbnail	2633	11DD149A694DF0F1AD47BFFED35FB038	94847/132 - 94852/17A
.png	embedded in Doc starting @ sector 7964	6428	13BF31F3F270E9755BA27C96A07FE615	7982/51 - 7994/16C
.txt		12826	F800A46E18FAFD309825C5EE84A654A2	11823/0 - 11848/19
.txt		53871	81A394A3B8D3CD03A4F2069D5084866C	34307/0 - 34412/6E
.txt (french)		30816	616A6BBE915C3DBF51014FD76F55B0E3	28246/0 - 28306/5F