available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Using purpose-built functions and block hashes to enable small block and sub-file forensics

Simson Garfinkel[a,*], *Alex Nelson*[a], *Douglas White*[a], *Vassil Roussev*[b]

[a] *Naval Postgraduate School, Graduate School of Operational and Informational Science, Department of Computer Science, Monterey CA 93943, USA*

[b] *Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA*

A B S T R A C T

This paper explores the use of purpose-built functions and cryptographic hashes of small data blocks for identifying data in sectors, file fragments, and entire files. It introduces and defines the concept of a "distinct" disk sector—a sector that is unlikely to exist elsewhere except as a copy of the original. Techniques are presented for improved detection of JPEG, MPEG and compressed data; for rapidly classifying the forensic contents of a drive using random sampling; and for carving data based on sector hashes.

1. Introduction

Much of computer forensics practice has focused on the recovery of files from media and the establishment of timelines. This paper presents research in the area of *bulk data analysis*, and specifically in the forensics of small data blocks.

There is a growing need for automated techniques and tools that operate on bulk data, and specifically on bulk data at the block level:

- File systems and files may not be recoverable due to damage, media failure, partial overwriting, or the use of an unknown file system.
- There may be insufficient time to read the entire file system, or a need to process data in parallel.
- File contents may be encrypted.
- The tree structure of file systems makes it hard to parallelize many types of forensic operations.

All of these problems can be addressed through the use of small block forensics. When individual files cannot be recovered, small block techniques can be used to analyze file fragments. When there is insufficient time to analyze the entire disk, small block techniques can analyze a statistically significant sample. These techniques also allow a single disk image to be split into multiple pieces and processed in parallel. Finally, because each block of an encrypted file is distinct, block-level techniques can be used to track the movement of encrypted files within an organization, even when the files themselves cannot be decrypted, because every block of every well-encrypted file should be distinct.

In this paper we introduce an approach for performing small block forensics. Some of this work is based upon *block hash calculations*—that is, the calculation of cryptographic hashes on individual blocks of data, rather than on entire files. Other work is based on *bulk data analysis*—the examination of blocks of data for specific features or traits irrespective of file boundaries.

Although we discuss these techniques in the context of files, they can be applied with equal validity to data from memory images and from computer networks at either the level of IP packets or reassembled TCP streams.

* *Corresponding author.*

E-mail addresses: slgarfin@nps.edu (S. Garfinkel), ajnelson@cs.ucsc.edu (A. Nelson), douglas.white@nist.gov (D. White), vassil@cs.uno.edu (V. Roussev).

1742-2876/$ – see front matter © 2010 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2010.05.003

1.1. *Outline of this paper*

This paper starts with an extended review of the prior work (Section 2). Next we present theoretical arguments and experimental results using the approach of using globally distinct sectors for content identification (Section 3). We then present a series of *discriminators* that we have developed for identifying small blocks of JPEG, MPEG, and Huffman-encoded data (Section 4). We present lessons learned (Section 5) and conclude with opportunities for future work (Section 6).

2. Prior work

Libmagic is a widely used file identification library that is the basis of the Unix file command (Darwin, 2008). Libmagic is reasonably accurate at identifying complete files and does so by looking for characteristic headers or footers ("magic numbers"). As a result, libmagic can only classify fragments of files that contain these elements.

The classification of fragments taken from the middle of a file has been an area of research for the past decade. Much of this work has been performed with the goal of making file and memory carving more efficient. McDaniel introduced the problem with a technique that combined the recognition of type-specific file headers and footers with statistical classification based on the frequencies of unigrams (McDaniel, 2001). Others attempting file fragment classification based on unigram and bigram statistics include Calhoun and Coles (2008); Karresand and Shahmehri (2006); Li et al. (2005); Moody and Erbacher (2008) and Veenman (2007).

Roussev and Garfinkel analyzed the statistical approach and determined that many of the reported results were inaccurate as they did not take into account the fact that certain file types, such as Adobe Acrobat files, are actually container files (Roussev and Garfinkel, 2009). Attempts to distinguish PDF fragments from fragments of JPEG files are inherently flawed, they argued, because Acrobat files frequently contain embedded JPEGs; furthermore, naïve statistical classification approaches based on n-gram statistics are unlikely to be successful due to the statistical properties of compressed data. They advocated a more specialized approach based on a better understanding of the underlying file formats.

Speirs *et al.* filed a US patent application that presented a variety of approaches for distinguishing compressed and encrypted or random data (Speirs and Cole, 2007). We are not aware of any other work in this area.

As part of his solution to the DFRWS 2006 Carving Challenge, Garfinkel used fragments of text from the Challenge to create Google query terms, from which he was able to find the very source documents on the Internet that had been used to assemble the Challenge. Garfinkel then broke these documents into 512-byte blocks, computed the MD5 hash of each block, and searched the Challenge for 512-byte sectors with matching hash codes. Using this technique, dubbed "the MD5 trick," Garfinkel identified three of the documents in the challenge image, including a Microsoft Word file that was divided into three fragments (Garfinkel, 2006a).

As part of their solution to the DFRWS 2007 Carving Challenge, Al-Dahir *et al.* developed mp3scalpel, a program that recognizes adjacent sectors of an MP3 file by validating frame headers (Al-Dahir et al., 2007). We have used their idea and some of their code in the development of our MP3 discriminator (Section 4.3.3).

Believing that MD5 and SHA1 algorithms were too slow for hash-based carving, Dandass *et al.* performed an analysis of the SHA1, MD5, CRC64 and CRC32 hash codes from 528 million sectors taken from 433,000 JPEG and WAV files to determine the collision rate of these algorithms for data found in the wild (Dandass et al., 2008). Collange *et al.* introduced the term "Hash-based Data Carving" for Garfinkel's "trick" and proposed using GPUs to speed hash computations (Collange et al., 2009).

3. Distinct block recognition

Cryptographic hashes are a powerful tool for analyzing the flow of content within criminal networks. If a network is known to be distributing a file with a specific hash and if a file on the subject's hard drive has the same hash, then it is reasonable to infer that the subject has had contact with the criminal network. This inference depends upon the hash being collision resistant so that it is extremely unlikely for two files to have the same hash. But it also depends on the file itself being *rare*, and not commonly found on systems that are used by individuals outside the criminal network.

As discussed in Section 2, many have tried to extend the concept of file hashes to hashes of small blocks or even individual disk sectors. For example, Garfinkel suggested that finding a shared sector between two different hard drives may imply that a file containing that sector was copied from the first drive to the second (Garfinkel, 2006b). Using hashes for this purpose requires more than collision resistance on the part of the hash function: it requires that the sectors or blocks being hashed be *distinct*.

There has been surprisingly little formal discussion or analysis regarding the prevalence of distinct sectors. For example, it is believed that individual digital photographs are distinct because of the amount of randomness within the world around us. On the other hand, our analysis of JPEGs shows that many JPEGs contain common elements such as XML structures, EXIF information and color tables. So while many JPEGs as a whole may have a distinct cryptographic hash, the individual blocks within one JPEG may be repeated in others.

3.1. *Understanding "distinct"*

The Greek philosopher Heraclitus is credited with the expression "you cannot step twice into the same river." This comment on the nature of change is surprisingly relevant to the study of computer forensics—but with an important twist.

The flexibility that language gives us to assemble words into sentences means that many sentences we say or write in day-to-day discourse have never been used before and will never be used again. The widespread existence of such distinct sentences is made all the more clear today with search engines that allow searching for quoted phrases: take a sequence of seven or eight sentences from any newspaper story and search for them. In many cases search engines will report only a single match—the original story.

Consider the phrase "Greatness is never given. It must be earned" taken from President Obama's inauguration speech. Although the sentiment seems common enough, our searches with Google and Yahoo for this quoted phrase only found references to the President's inauguration speech, and no occurrence of this eight-word sequence beforehand. The great variation made possible by natural languages is one reason that plagiarism detection engines such as Turnitin can be so successful.

Other word sequences are not distinct. "To be or not to be" and "four score and seven years ago" were once original constructions, but their fame has made them commonplace. However, it is fair to suggest that any occurrence of these six-word sequences are either copied from the original, or else a copy of a copy.

The same measure of distinctness can be applied to individual disk sectors. There are $2^{512\times8} \approx 10^{1,233}$ different 512-byte sectors. This number is so impossibly large that it is safe to say that a randomly generated 512-byte pattern will never appear anywhere else in the universe unless it is intentionally copied. You cannot step twice into the same random number generator.

On the other hand, a sector that is filled with a constant value cannot be distinct: there are only 256 such sectors.

It is possible to determine that a sector is not distinct, but it is impossible to create a function that states with certainty that a sector is distinct. Certainly sectors that have high entropy are likely to be distinct, but some sectors that have low entropy are also likely to be distinct. A 512-byte sector with 500 NULLs and 12 ASCII spaces is likely to be distinct if the spaces are randomly distributed within the sector, since there are $512!/500! \approx 10^{33}$ possible arrangements of the spaces. Nevertheless, the arrangement of all of the spaces at the beginning of the sector is almost certainly not distinct. A randomly generated sector is certainly distinct at the moment it is created. But if that sector is widely distributed and incorporated into other files, then it is no longer distinct.

3.2. The distinct block definition and hypotheses

Given the preceding, we propose this definition:

Distinct Block (definition): A block of data that will not arise by chance more than once.

Distinct blocks need to be manufactured by some random process. Blocks taken from a JPEG created by a digital camera in total darkness are unlikely to be distinct, but the same camera taking pictures outside on a sunny day will surely generate distinct blocks: You cannot step twice into the same sunny day.

Distinct blocks can be a powerful forensic tool if we assume these two hypotheses:

Distinct Block Hypothesis #1: If a block of data from a file is distinct, then a copy of that block found on a data storage device is evidence that the file was once present.

Distinct Block Hypothesis #2: If a file is known to have been manufactured using some high-entropy process, and if the blocks of that file are shown to be distinct throughout a large and representative corpus, then those blocks can be treated as if they are distinct.

3.3. Block, sector, and file alignment

Many of the techniques that we have developed take advantage of the fact that most file systems align most files on sector boundaries. That is, a 32KiB file F can be thought to consist of 64 blocks of 512 bytes each, $B_0 \ldots B_{63}$. If this file is stored contiguously on a drive that has 512-byte sectors, then it will be the case that block B_0 will be stored at some sector S_n, B_1 will be stored at S_{n+1} and so on. This so-called *sector alignment* has a significant performance advantage, as it allows the operating system to schedule data transfers directly from the storage medium into user memory.

Sector alignment can be exploited in forensic analysis by choosing a block size that matches the the sector size: if F contains 32KiB of distinct data—that is, data which is not found elsewhere on the disk—then there will be an exact correspondence between block hashes of the file's 64 512-byte blocks and 64 sectors of the drive. Notice that this will be true even if the file is fragmented, provided that it is fragmented on a sector boundary.

Not all files are stored sector aligned. In particular NTFS stores small files in the Master File Table. Such files are not susceptible to some of the techniques presented here.

3.4. Choosing a standard block size

It is useful to employ a consistent block size when performing small block forensics. The block size must be smaller than the typical size of a file of interest to avoid padding issues. But there is no optimal size. Small block sizes increase the resolving power of our tools but also increases the amount of data that needs to be analyzed. Nevertheless, algorithms must be tuned to a specific block size, and databases of hash codes need to be computed with a specific block size for general distribution.

Today most hard drives use 512-byte sectors, so this is a logical size for small block forensics. Nevertheless, we standardize on 4096-byte blocks for most of our work for a variety of reasons, including:

- When working with block hashes, a block size of 4096 bytes generates $\frac{1}{8}$ the data as a block size of 512 bytes. This dramatically reduces data storage requirements and speeds processing times.
- The bulk data discriminators that we have developed (Section 4) are significantly more accurate with 4096-byte blocks than 512-byte blocks.
- Since most files of forensic interest are larger than 4096 bytes, the decreased resolution that results from working with 4096-byte blocks is less significant.
- Finally, the storage industry is moving to 4096-byte sectors, meaning that future drives will write 4096 bytes atomically the way that current drives write 512 bytes atomically (Fonseca, 2007).

3.5. Managing 4096-byte blocks with 512-byte sectors

When working with devices that have a 512-byte sector size it is a simple matter to combine eight sectors into a single block. However, because the alignment of sectors-to-blocks might change within a single disk, in practice it is useful to combine a run of 15 sectors into eight overlapping 4096-byte blocks:

$S_0S_1S_2S_3S_4S_5S_6S_7 => B_0$
$S_1S_2S_3S_4S_5S_6S_7S_8 => B_1$
…
$S_7S_8S_9S_{10}S_{11}S_{12}S_{13}S_{14} => B_7$

Thus, when performing a match against a database of 4KiB sector hashes with a 512-byte device, it is useful to search the database for $B_0 \ldots B_7$, rather than just B_0.

3.6. *Experimental results*

Using the **nps-2009-domexusers** (Garfinkel et al., 2009) disk image, we computed the number of distinct and duplicated blocks with block sizes of 2^9 (512 bytes) through 2^{14} (16,384 bytes) (Table 1).

The first processing step was to remove constant blocks. Overwhelmingly most of the blocks removed were filled with hex 00, although many sectors filled with FF and other values were also removed. As there are only 256 constant blocks we felt that these blocks would be of limited probative value.

We found that **nps-2009-domexusers** contains roughly 6.7 GB of data, a figure that includes allocated file content, residual data, file system metadata, directories, and other non-file content. The fact that roughly the same amount of data remained irrespective of the blocksize implies that the feature size of the file system's allocation strategy is larger than 16,384 bytes. This lends credence to our claim that small block forensics need not be performed using the native media's sector size (in this case, 512 bytes).

Next we computed the SHA-1 hash of each block and stored the results in a 2^{32}-bit Bloom filter with $k = 4$. Our implementation started with Farrell's (Farrell et al., 2008) and added a GNU C++ Standard Template Library map to count the number of times that each SHA-1 value is encountered (space in the map is conserved by only adding hash values the *second* time that a hash is encountered). This code allowed us to calculate the number of *distinct* blocks and the number of *duplicated* blocks—that is, blocks that appeared more than once on the disk.

By design, Bloom filters cannot have false negatives but can have false positives. A Bloom filter with $m = 2^{32}$ and $k = 4$ storing 80 million elements is predicted to have a false positive rate $p < 2.66 \times 10^{-5}$. Thus, even if every 512-byte sector in nps-2009-domexusers were distinct, there would be less than 2500 false positives, which is irrelevant for the purposes of the statistics shown in Table 1. The low false positive rates are reflected by the very low BF utilization shown in the last row of Table 1. In fact, this experiment could have been done with a significantly smaller BF; with $M = 30$ the BF would consume only 128 MiBytes (instead of 512 MiB) and would still have a worst-case false positive rate of $p \approx 0.0044$.

Sector duplication can result from duplication within files (repeated regions), or from multiple copies of file on a drive. As Table 1 shows, approximately half of the non-constant containing sectors on **nps-2009-domexusers** are distinct.

The fraction of distinct sectors increases with larger block sizes. One possible explanation is that the duplicate sectors are from multiple copies of the same file. Recall that most files are stored contiguously. With small sampling block sizes there is a good chance that individual files will align with the beginning of a block sample. But as the sampling size increases, there is an increased chance that the beginning of a file will not align with a sampling block. If two files align differently, then the block hashes for the two files will be different.

Moving from 512-byte blocks to 4096-byte blocks results in an 8-fold reduction in data processing requirements but produces only a 27% increase in the percentage of distinct sectors. As a result, we feel that the 4KiB block size is a good compromise between performance and accuracy when performing block hashing.

To test this hypothesis, we conducted a large-scale study of data blocks from a dataset of reference files and from a corpus of disk images.

We started with 7,761,607 unique files extracted from the National Software Reference Library (National Institute of Standards and Technology, 2005) as of September 30, 2009. For each file we calculated the SHA-1 hash for each 4096-byte block. Where a file did not have a content size that is a multiple of 4096, we have padded the file with NUL (00) bytes to a size that is a multiple of 4096. These NSRL files had a total of 651,213,582 4KiB blocks (1,436 GiB).

In our data set we identified 558,503,127 (87%) blocks that were distinct and 83,528,261 (13%) that appeared in multiple locations. By far the most common was the SHA-1 for the block of all NULLs, which appeared 239,374 times. However many of these duplicates are patterns that repeat within a single file but which are not present elsewhere within the NSRL, allowing the hashes to be used to recognize a file from a recognized fragment.

Table 1 – Self-similar measurements of nps-2009-domexusers with different sector sizes. Constant blocks are removed.

Block Size:	512	1024	2048	096	8192	16,384
Blocks	83,886,080	41,943,040	20,971,520	10,485,760	5,242,880	2621,440
Removed blocks	70,897,785	35,413,863	17,683,597	8,833,713	4414,612	2,206,318
Data (blocks)	12,988,295	6,529,177	3287,923	1,652,047	828,268	415,122
Data (MB)	6650	6685	6733	6766	6785	6801
Distinct Blocks	7,236,604	3741,737	1,929,396	989,963	609,760	364,472
Duplicated Blocks	5,751,691	2787,440	1,358,527	662,084	218,508	50,650
Percent Distinct	56%	57%	59%	60%	74%	88%
BF Size	2^{32}	2^{32}	2^{32}	2^{32}	2^{32}	2^{32}
BF Utilization	1%	0%	0%	0%	0%	0%

3.7. *Application*

This section discusses three applications that we have developed that use of the properties of distinct blocks.

3.7.1. *Hash-based carving with frag_find*

Carving is traditionally defined in computer forensics as the searching for data objects based on *content*, rather than following pointers in metadata (Garfinkel, 2007). Traditional carvers operate by searching for headers and footers; some carvers perform additional layers of object validation. Hash-based carving, in contrast, searches a disk for "master" files already in a corpus by performing sector-by-sector hash comparisons.

We have developed a tool called frag_find that achieves high-speed performance on standard hardware. Here we describe the algorithm using the terminology proposed by Dandass et al. (2008) and Collange et al. (2009), although our algorithm does not require the hardware-based acceleration techniques that are the basis of their research:

1. For each **master** file a filemap data structure is created that can map each master file sector to a set of sectors in the image file. A separate filemap is created for each master.
2. Every sector of each master file is scanned. For each sector the MD5 and SHA-1 hashes are computed. The MD5 code is used to set a corresponding bit in a 2^{24} bit Bloom filter. The SHA-1 codes are stored in a data structure called the shamap that maps SHA-1 codes to one or more sectors in which that hash code is found.
3. Each sector of the **image** file is scanned. For each sector the MD5 hash is computed and the corresponding bit checked in the Bloom filter. This operation can be done at nearly disk speed. Only when a sector's MD5 is found in the Bloom filter is the sector's SHA-1 calculated. The shamap structure is consulted; for each matching sector found in the shamap, the sector number of the IMAGE file is added to the corresponding sector in each master filemap.
4. Each filemap is scanned for the longest runs of consecutive image file sectors. This run is noted and then removed from the filemap. The process is repeated until the filemap contains no more image file sectors.

Our hash-based carving algorithm covers both fragmented master files and the situation where portions of a master are present in multiple locations in the image file.

Our original hash-based carving implementation used Adler-32 (Deutsch and Gailly, 1996) instead of MD5 as the fast hash, but subsequent testing found that most MD5 implementations are actually faster than most Adler-32 implementations due to extensive hand-optimization that the MD5 implementations have received. Although the new algorithm could dispense with SHA-1 altogether and solely use MD5, the MD5 algorithm is known to have deficiencies. If the results of hash-based carving are to be presented in a court of law, it is preferable to use SHA-1. To further speed calculation we found it useful to precompute the SHA-1 of the NULL-filled sector; whenever the system is asked to calculate the value of this sector, this value is used instead.

We prefer to use *sectors* for hash-based carving because they are the minimum allocation unit of the disk drive. Larger block sizes are more efficient, but larger blocks complicate the algorithm because of data alignment and partial write issues. As a result, hash-based carving may fail to identify valid data when used with block sizes larger than the sector size. Hence our decision to carve with a blocksize equal to the sector size.

3.7.2. *Preprocessing for carving with precarve*

Often when performing file carving it is useful to remove from the disk image all of the allocated sectors and to carve the unallocated space. Our experience with frag_find showed us that fragments of a master file are often present in multiple locations on a disk image. This led us to the conclusion that it might be useful to remove from the disk image not merely the allocated files, but all of the distinct sectors from the image's allocated files. The result, we hoped, would be a carving target that would be smaller and from which it might be possible to recover objects without the need to resort to fragment recovery carving.

After some experimentation we created a tool called precarve that performs a modified version of this removal. We found that removing distinct blocks was not sufficient, as there were blocks that were shared in multiple files which could not be safely removed. We re-designed the tool so that it would remove any sequence of sectors from the unallocated region that matched more than r allocated sectors. After trial-and-error we found that $r = 4$ provided the best performance when carving JPEGs.

We tested precarve using the **nps-2009-canon2-gen6** (Garfinkel et al., 2009) disk image. The disk image was created with a 32 MB SD card and a Canon digital camera in 2009. Photos were taken and then deleted in such a manner to create JPEGs that are fragmented in multiple places and other images that can only be recovered through file carving. Carving was performed with Scalpel version 1.60 (Richard and Roussev, 2005).

Scalpel recovered 76 JPEGs with some displayable content and 37 for which nothing could be displayed. After running precarve we were able to recover two displayable JPEG fragments and one full-size image (G2-3) that had not previously been recovered. Because the precarve process requires no human intervention, there is no reason not to include this algorithm in current forensic protocols that involve the carving of unallocated space.

3.8. *Statistical sector sampling to detect the presence of contraband data*

Sector-based hashing can be combined with statistical sampling to provide rapid identification of residual data from large files. This might be especially useful at a checkpoint, where the presence of a specific file might used as the basis to justify a more thorough search or even arrest.

Consider a 100 MB video for which the hash of each 512-byte block is distinct. A 1 TB drive contains approximately 2 billion 512-byte sectors. If one 512-byte sector is sampled at random, the chance that the data will be missed is overwhelming— 2,000,000,000 − 200,000/2,000,000,00 = 0.9999. If more than one sector is sampled the chances of missing the data can be calculated using the equation:

$$p = \prod_{i=1}^{n} \frac{((N-(i-1))-M)}{(N-(i-1))} \quad (1)$$

Where N is the total number of sectors on the media, M is the number of sectors in the target, and n is the number of sectors sampled. Readers versed in statistics will note that we have described the well-known "Urn Problem" of sampling without replacement (Devore, 2000).

If 50,000 sectors from the TB drive are randomly sampled, the chance of missing the data drops precipitously to $p \approx 0.0067$ ($N = 2,000,000,000$, $M = 200,000$ and $n = 50,000$.) The odds of missing the data are now roughly 0.67%—in other words, there is a greater than 99% chance that at least one sector of the 100 MB file will be found through the proposed random sampling.

4. Fragment type discrimination

Given a fragment of a file, the first thing that one might wish to do is to determine the kind of file from which the fragment was taken: did the fragment come from a JPEG image file, a PDF file, a Microsoft Word file, or some other source?

4.1. *"Discrimination," not "identification"*

To the trained forensic investigator that has seen the inside of many files, the fragment type identification problem doesn't seem so hard. After all, many file types have distinct characteristics. For example, the ASCII sequences shown in Fig. 2 are commonly seen in JPEG files, Fig. 3 is indicative of Microsoft Word files, and Fig. 4 is characteristic of a PDF file.

Although the investigator's intuition may be correct, it is of limited value for two reasons. First, although every file type does have distinct sequences, a block-by-block analysis of files indicates that most file blocks lack these distinctive features. A second important problem, noted in the literature review, is that much of the previous work has failed to take into account the fact that PDF, Microsoft Office, and ZIP files are *container files* in which JPEGs are frequently embedded without alteration. Thus, there is no discernible difference between a block taken from the middle of a JPEG file and one taken from the middle of a JPEG image embedded within a PDF file.

Because container files can combine files of different types on byte boundaries, we believe that the phrase *fragment type identification* is inherently misleading: a single file fragment can contain multiple types! Instead, we have adopted the phrase *fragment discrimination* for this work. Adapting the phrase from electronics, we are creating software devices that produce an output when their input exceeds a certain threshold. We argue that discrimination, rather than characterization, is the correct approach when working with small file fragments, since a fragment taken from a container file might actually contain traces from multiple document types—for example, a fragment taken from a PDF file might contain both PDF and JPEG metadata, if a JPEG was embedded within the PDF.

```
^V^W^X^Y^Z%&'()*456789:CDEFGHIJSTUVWXY
:exif='http://ns.adobe.com/exif/1.0/'>
```

Fig. 2 – ASCII sequences commonly seen in JPEG files. The first is the ASCII representation of the JPEG quantization table; the second is a fragment of XML that is embedded in many JPEGs by Adobe PhotoShop.

4.2. *Approaches*

We have identified several approaches for small fragment discrimination:

4.2.1. *Header recognition*

When a fragment is taken from the beginning of a file, traditional approaches based on the first bytes of a file can be used to identify the fragment type. This approach takes advantage of the fact that most file systems align the start of files on sector boundaries for files larger than 1500 bytes.

4.2.2. *Frame recognition*

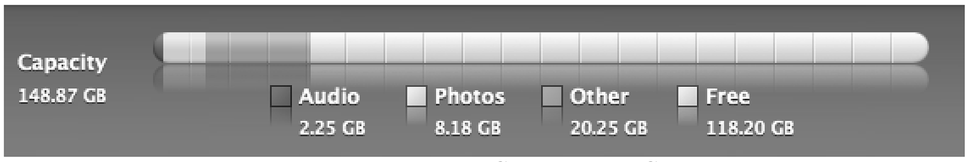
Many multimedia file formats employ repeating frames with either a fixed or variable-length offset. If a frame is recognized from byte patterns and the next frame is found at the specified offset, then there is a high probability that the fragment contains an excerpt of the media type in question.

4.2.3. *Field validation*

Once headers or frames are recognized, they can be validated by "sanity checking" the fields that they contain.

4.2.4. *n-Gram analysis*

As some n-grams are more common than others, discriminators can base their results upon a statistical analysis of n-grams in the fragment.



Audio Data reported by iTunes:	2.25 *GiB*	2.42 *GB*	
MP3 files reported by file system:		2.39 *GB*	
Estimated MP3 usage with random sampling :		2.49 *GB*	10,000 random samples
		2.71 *GB*	5,000 random samples

Fig. 1 – Usage of a 160 GB iPod reported by iTunes 8.2.1 (6) (top), as reported by the file system (bottom center), and as computing with random sampling (bottom right). Note that iTunes usage actually in GiB, even though the program displays the "GB" label.

```
New York, September 2008^M\223Security
 Metrics: What can you test?\224, Veri
fy 2007 International Software Testing
 Conference, Arlington, Virginia, Octo
ber 2007.^M\223Attacks and Countermeas
```

Fig. 3 – ASCII sequence taken from a Microsoft Word file showing the use of carriage returns between paragraphs and "smart quotes.".

4.2.5. *Other statistical tests*

Tests for entropy and other statistical properties can be employed.

4.2.6. *Context recognition*

Finally, if a fragment cannot be readily discriminated, it is reasonable to analyze the adjacent fragments. This approach works for fragments found on a hard drive, as most files are stored contiguously (Garfinkel, 2007). This approach does *not* work for identifying fragments in physical memory, however, as modern memory systems make no effort to co-locate adjacent fragments in the computer's physical memory map.

4.3. *Three discriminators*

In this subsection we present three discriminators that we have created. Each of these discriminators was developed in Java and tested on the NPS *govdocs1* file corpus (Garfinkel et al., 2009), supplemented with a collection of MP3 and other files that were developed for this project.

To develop each of these discriminators we started with a reading of the file format specification and a visual examination of file exemplars using a hex editor (the EMACS hexl mode), the Unix more command, and the Unix strings command. We used our knowledge of file types to try to identify aspects of the specific file format that would be indicative of the type and would be unlikely to be present in other file types. We then wrote short test programs to look for the features or compute the relevant statistics for what we knew to be true positives and true negatives. For true negatives we used files that we thought would cause significant confusion for our discriminators.

4.3.1. *Tuning the discriminators*

Many of our discriminators have tunable parameters. Our approach for tuning the discriminators was to use a *grid search*. That is, we simply tried many different possible values for these parameters within a reasonable range and selected

```
0000146009 00000 n^M
0000146048 00000 n^M
0000001356 00000 n^M
trailer^M
<</Size 236/Prev 313946/Root 184 0 R/I
```

Fig. 4 – ASCII sequence taken from within a PDF file, showing a portion of the xref table and the characteristic metadata encoding scheme.

the parameter value that worked the best. Because we knew the ground truth we were able to calculate the *true positive rate* (TPR) and the *false positive rate* (FPR) for each combination of parameter settings. The (FPR,TPR) for the particular set of values was then plotted as an (X,Y) point, producing a ROC curve (Zweig and Campbell, 1993).

4.3.2. *JPEG discriminator*

To develop our JPEG discriminator we started by reading the JPEG specification. We then examined a number of JPEGs, using as our source the JPEGs from the *govdocs1* corpus (Garfinkel et al., 2009).

JPEG is a segment-based container file in which each segment begins with a FF byte followed by segment identifier. Segments can contain metadata specifying the size of the JPEG, quantization tables, Huffman tables, Huffman-coded image blocks, comments, EXIF data, embedded comments, and other information. Because metadata and quantization tables are more-or-less constant and the number of blocks is proportional to the size of the JPEG, small JPEGs are dominated by metadata while large JPEGs are dominated by encoded blocks.

The JPEG format uses the hex character FF to indicate the start of segments. Because this character may occur naturally in Huffman-coded data, the JPEG standard specifies that naturally occurring FFs must be "stuffed" (quoted) by storing them as FF00.

Our JPEG discriminator uses these characteristics to identify Huffman-coded JPEG blocks. Our intuition was to look for blocks that had high entropy but which had more FF00 sequences than would be expected by chance. We developed a discriminator that would accept a block as JPEG data if the entropy was considered high—that is, if it has more than *HE* (*High Entropy*) distinct unigrams—and if it had at least *LN* (*Low N-gram count*) FF00 bigrams.

Our ground truth files were drawn from the *govdocs1* corpus. For true positives we used JPEGs. For true negatives we used a combination of CSV, GIF, HTML, PNG, MP3s (without cover art), PNGs, random data, ASCII text, WAV and XML files. Our ground truth files were treated as a set of 30 million 4KiB blocks (Table 2). These files were chosen specifically to avoid container files such as PDF and Microsoft Word.

We performed a grid search with values of HE from 0 to 250 stepping by 10 and values of LN from 0 to 10. Each experiment evaluated the JPEG discriminator with 30 million blocks of data. The resulting ROC curve is shown in Fig. 5.

To find the best tunable values, we first chose points that were closest to the upper-left corner—that is, with the highest TPR and lowest FPR. Because there were three clusters of values, we used accuracy as a tiebreaker. The cluster of values at $LN = 1$ was 98.91% accurate at best; $LN = 2$ was 99.28% accurate at best; and $LN = 3$ was 99.08% accurate at best. In each of these cases the best HE value was 220. We chose $LN = 2$. Table 3 shows the confusion matrix for this tuning point.

4.3.3. *The MP3 discriminator*

MP3 files have a frame-based structure that is easily identified in bulk data. Although frames may occur on any byte boundary, each frame consists of a 4-byte header containing

Table 2 – The file types used to tune the JPEG discriminator. Ground truth negative files were chosen such that they contain no JPEG data, so container formats such as PDF's were excluded.

Type	# 4KiB blocks	% of Ground Truth
Ground truth negative		
csv	826377	3.9
gif	728041	3.5
html	2789111	13.
log	1070871	5.1
mp3	107020	0.5
png	274013	1.3
rng	163840	0.8
text	12869592	61
wav	155561	0.74
xml	2129966	10.
Total	21114392	100.0
Ground truth positive		
jpg	9055284	100.0

metadata for only the current frame. Frame headers have two properties that makes them further amenable to fragment identification:

- Each frame header starts with a string of 11 or 12 *sync bits* (SB) that are set to 1.
- The length of the frame is exactly calculable from the header.

To identify a fragment as MP3, we scan the fragment until we find a byte pair that contains 11 or 12 set bits. We then extract the frame's bit rate, sample rate and padding flag from their binary representation in the header. These values are

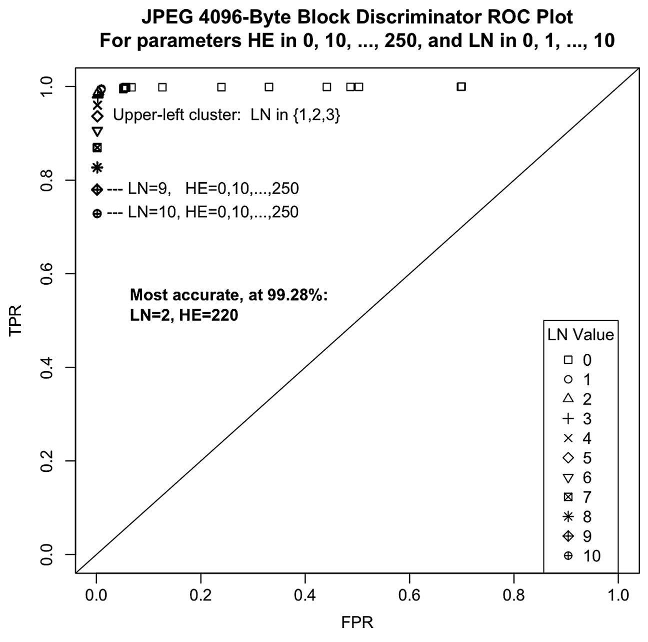


Fig. 5 – ROC plot for JPEG discrimination, varying High Entropy (HE) and Low FF00 N-gram Count (LN). Decreasing HE raises false positives. Increasing LN lowers true positives.

Table 3 – Confusion matrix for the tuned JPEG identifier. There were 30,169,676 total samples.

	True Positive	True Negative
Predicted Positive	8,950,321	113,757
Predicted Negative	104,963	21,000,635

sanity-checked, after which the frame's length is calculated according to the formula:

$$\text{FrameSize} = \frac{144 \times \text{BitRate}}{\text{SampleRate} + \text{Padding}} \tag{2}$$

The discriminator then skips forward FrameSize bytes and checks to see if a second MP3 frame header is present. If the bytes at the location do not contain sync bytes or sane values for BitRate, SampleRate and Padding, then the discriminator concludes that it there was a false positive and it starts searching for the next byte pair that might contain sync bytes. If the bytes at the location do contain sync bytes and sane values for the header parameters, the discriminator skips forward again, looking for the next frame. This is the same algorithm as developed by Al-Dahir *et al.* for mp3scalpel (Al-Dahir et al., 2007).

Our intuition was that the mp3scalpel algorithm could be turned into a discriminator by simply accepting blocks that had more than a certain number of frame headers. We call this parameter *CL* (*chain length*). (This approach was not applied to the JPEG discriminator because the JPEG format does not contain framing information from which the chains could be readily validated.)

We tuned the MP3 identifier in a similar manner to the JPEG discriminator. We defined ground truth positive as MP3s without embedded album art. For ground truth negative we used pseudorandom data, generated with/dev/urandom. We found that SB values of 11 and 12 produced identical results, indicating that our MP3s all had 12 sync bits. Chains of length 0 and 1 performed identically, but longer chains proved more accurate. Table 4 shows the confusion matrix for chains of length 4. The accuracy is 99.56%.

4.3.4. *The huffman-coded discriminator*

The DEFLATE compression algorithm is the heart of the ZIP, GZIP and PNG compression formats. Compression symbols are coded with Huffman coding. Thus, being able to detect fragments of Huffman-coded data allows distinguishing this data from other high entropy objects such as random or encrypted data. This can be very important for operational computer forensics. (Techniques that might allow one to distinguish between random data and encrypted data are beyond the scope of this paper.)

Table 4 – Confusion matrix for the tuned MP3 identifier. There were 801,076 total samples.

	True Positive	True Negative
Predicted Positive	635,738	1993
Predicted Negative	1498	161,847

We have developed an approach for distinguishing between Huffman-coded data and random or encrypted data using an autocorrelation test. Our theory is based on the idea that Huffman-coded data contains repeated sequences of variable-length bit strings. Some of these strings have 3 bits, some 4, and so on. Presumably some strings of length 4 will be more common than other strings of length 4. When a block of encoded data is shifted and subtracted from itself, sometimes the symbols of length 4 will line up. When they line up and the autocorrelation is performed, the resulting buffer will be more likely to have bits that are 0s than bits that are 1s. Although the effect will be slight, we suspected that it could be exploited.

As a result, we came up with this algorithm for a Huffman-coded discriminator:

1. As with the JPEG and MPEG discriminators, we evaluate the input buffer for high entropy. If it is not high entropy, it is not compressed.
2. We perform an autocorrelation by rotating the input buffer and performing a byte-by-byte subtraction on the original buffer and the rotated buffer, producing a resultant autocorrelation buffer.
3. We compute the vector cosine between the vector specified by the histogram of the original buffer and the histogram of each autocorrelation buffer. Vector cosines range between 0 and 1 and are a measure of similarity, with a value of 1.0 indicating perfect similarity. Our theory is that random data will be similar following the autocorrelation, since the autocorrelation of random data should be random, while Huffman-coded data will be less similar following autocorrelation.
4. We set a threshold value MCV (*minimum* cosine *value*); high-entropy data that produces a cosine similarity value between the original data and the autocorrelated data that is less than MCV is deemed to be non-random and therefore Huffman coded.

The number of histogram bins to compare is the second tunable parameter. We call this value *VL* (*vector length*). For each *VL*, we chose the minimum cosine of the encrypted data as the MCV, using a training set of 0.1% of our data. Ground truth positive was a set of large text and disk image files, compressed, and negative was the compressed files AES-encrypted. We then ran a grid search over the other 99.9% to determine which was the most accurate for a particular block size.

Unlike the JPEG discriminator, this discriminator does not need to be tuned for the block size. This discriminator also improves with accuracy as the block size increases, as shown in Table 5 and Table 6. This discriminator rarely mistakes encrypted data for compressed data, and correctly identifies approximately 49.5% and 66.6% of the compressed data with 4KiB and 16KiB block sizes, respectively. The low false positive rate lets us estimate the amount of Huffman-coded data seen in a random sample. We produce a rough estimate by taking a randomly chosen distribution of 4KiB blocks and multiplying the percentage of identified compressed data by TPR^{-1}.

Table 5 – Confusion matrix for the cosine-based Huffman-coded data discriminator on 4KiB, using $VL = 255$. There were 3,569,107 total samples for 4KiB blocks. Accuracy is 49.5%, TPR is 21.1%, and FPR is 0.0197%.

	True Positive	True Negative
Predicted Positive	482,015	95
Predicted Negative	1,802,869	1284,128

Table 6 – Confusion matrix for the cosine-based Huffman-coded data discriminator on 16KiB, using $VL = 250$. There were 594,851 total samples for 16KiB blocks. Accuracy is 66.6%, TPR is 48.0%, FPR is 0.450%.

	True Positive	True Negative
Predicted Positive	182,939	827
Predicted Negative	197,875	213,210

4.4. *Application to statistical sampling*

Although fragment type identification was created to assist in file carving and memory analysis, another use of this technology is to determine the content of a hard drive using statistical sampling.

For example, if 100,000 sectors of a 1 TB hard drive are randomly sampled and found to contain 10,000 sectors that are fragments of JPEG files, 20,0000 sectors that are fragments of MPEG files, and 70,000 sectors that are blank, then it can be shown that the hard drive contains approximately 100 GB of JPEG files, 200 GB of MPEG files, and the remaining 700 GB is unwritten.

Fig. 1 shows the results of statistical sampling applied to a 160 GB Apple iPod that we created using Apple's iTunes, iPhoto, and the popular TrueCrypt cryptographic file system. First the iPod was zeroed using dd and new firmware was loaded. Next the iPod was loaded with thousands of photographs and audio data. While it was loading the disk was mounted using iTunes' "Enable Disk Use" option and a 20 GB TrueCrypt volume was created. This interspersed encrypted data among the more commonly expected data.

The analysis of the Apple iPod is complicated by the fact that the Apple iPod stores thumbnail images that are displayed on the screen in a proprietary non-JPEG file format called a.ithmb file. We do not yet have a fragment recognizer for this format. Nevertheless, we were able to accurately determine the amount of JPEG and MPEG data, as well as the free space.

5. Lessons learned

Research and development in small block forensics is complicated by the large amount of data that must be processed: a single 1 TB hard drive has 2 billion sectors; storing the SHA1 codes for each of these sectors requires 40 GB—more storage than will fit in memory of all but today's largest computers. And since SHA1 codes are by design high entropy and unpredictable, they are computationally expensive to store and retrieve from a database. Given this, we wish to share the following lessons:

1. We implemented frag_find in both C++ and Java. The C++ implementation was approximately three times faster on the same hardware. We determined that this speed is due to the speed of the underlying cryptographic primitives.
2. Because it is rarely necessary to perform database JOINs across multiple hash codes, it is straightforward to improve database performance splitting any table that references SHA1s onto multiple servers. One approach is to use one server for SHA1 codes that begin with hex 0, and one for those beginning with hex 1, and so on, which provides for automatic load balancing since hashcodes are pseudo-random. Implementing this approach requires that each SELECT be executed on all 16 servers and then the results recombined (easily done with map/reduce). Storage researchers call this approach *prefix routing* (Bakker et al., 1993).
3. Bloom filters are a powerful tool to prefilter database queries.
4. In a research environment it is dramatically easier to store hash codes in a database coded as hexadecimal values. In a production environment it makes sense to store hash codes in binary since binary takes half the space. Base64 coding seems like a good compromise, but for some reason this hasn't caught on.
5. We have made significant use of the C++ STL map class. For programs like frag_find we generally find that it is more useful to have maps of vectors than to use the multimap class. We suspect that it is also more efficient, but we haven't tested this.

6. Conclusions

This paper explores forensic analysis at the block and sector level. Although this work is performed below the level of files, we take the blocks of data analyzed to be representative of files—either files that were once resident on the disk and have now been partially overwritten, or else files that are still resident on the disk but not necessarily in a sequence of contiguous sectors.

We showed that there exist *distinct* data blocks that, if found, indicate that the entire file from which the block was extracted was once resident on the media in question (Section 3). We showed how this notion of distinctiveness can be used for hash-based carving (Section 3.7.1), for preprocessing a disk image so that carving will be more efficient (Section 3.7.2), and for rapid drive analysis (Section 3.8).

We showed that it is possible to recognize a fragment of a JPEG or MPEG file on a disk with extraordinarily high accuracy using an algorithm that has been written with knowledge of the underlying file format. We showed how to tune these algorithms using the *grid search* technique. We then showed that fragment type recognition can be used to rapidly determine the contents of a storage device using statistical sampling.

6.1. Future work

The technique for discriminating encrypted data from compressed data is in its infancy and needs refinement. More generally, we are in need of more file fragment identifiers and a larger database of distinct sector hashes.

Hash-based carving could be performed using a sector-based similarity digest instead of a hash to search for similar files.

Our approach of using grid search for finding optimal tuning parameters is promising but needs refinement. Specifically, an alternative tuning approach would be to dispense with the ROC plots and simply use the combination of tunable parameters that produce the highest F-score. Also, it would be useful to re-run our grid search with tighter bounds and a smaller step value to find improved values for the tuning.

It may be possible to combine the recognition techniques from our JPEG, MPEG and Huffman discriminators for improved accuracy. For example, it may be possible to read frames in the JPEG files. On the other hand, the JPEG and MPEG discriminators are so accurate that there is little reason to improve them further.

In the future, we hope to augment our random sampling system with Andersen *et al.*'s FAWN-KV system (Andersen et al., 2009).

Finally, we will be producing a release of our discriminators and discriminator evaluation toolbench written in C. The software will be downloadable our website at https://domex.nps.edu/deep/.

Acknowledgements

We also wish to acknowledge Bruce Allen, Jesse Burns, George Dinolt, Ethan Miller, and Beth Rosenberg for their guidance and support of this project. We also wish to thank the anonymous reviewers, whose comments were very helpful in revising this paper.

REFERENCES

Al-Dahir Omar, Hua Joseph, Marziale Lodovico, Nino Jaime, Richard III Golden G, Roussev Vassil. MP3 scalpel, http://sandbox.dfrws.org/2007/UNO/executables_and_source/; 2007.

Andersen David G, Franklin Jason, Kaminsky Michael, Phanishayee Amar, Tan Lawrence, Vasudevan Vijay. FAWN: a fast array of wimpy nodes. In: SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 1–14. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-752-3.

Bakker EM, Leeuwen J, Tan RB. Prefix routing schemes in dynamic networks. Computer Networks and ISDN Systems 1993;26(4): 403–21.

Calhoun William C, Coles Drue. Predicting the types of file fragments. In: Digital Investigation: The Proceedings of the Eighth Annual DFRWS Conference, vol. 5, 2008.

Collange Sylvain, Daumas Marc, Dandass Yoginder S, Defour David. Using graphics processors for parallelizing hash-based data carving, http://hal.archives-ouvertes.fr/docs/00/35/09/62/PDF/ColDanDauDef09.pdf; 2009.

Dandass Yoginder Singh, Necaise Nathan Joseph, Thomas Sherry Reede. An empirical analysis of disk sector hashes for data carving. Journal of Digital Forensic Practice 2008;2:95–104.

Darwin Ian F. Libmagic, ftp://ftp.astron.com/pub/file/; August 2008.

Deutsch LP, Gailly J-L. RFC 1950: ZLIB compressed data format specification version 3.3, May 1996. Status: INFORMATIONAL.

Devore Jay L. Probability and Statistics for Engineering and the Sciences. 5th ed. Duxbury; 2000.

Farrell Paul, Garfinkel Simson, White Doug. Practical applications of bloom filters to the nist rds and hard drive triage. In: Annual Computer Security Applications Conference 2008, December 2008.

Fonseca Brian. Hard-drive changes: Long block data standard gets green light. Computerworld, http://www.computerworld.com/action/article.do?articleID=9018507; May 2007.

Garfinkel Simson L, Farrell Paul, Roussev Vassil, Dinolt George. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the 9th Annual Digital Forensic Research Workshop (DFRWS), August 2009.

Garfinkel Simson L. Dfrws 2006 challenge report, http://sandbox.dfrws.org/2006/garfinkel/part1.txt; 2006a.

Garfinkel Simson L. Forensic feature extraction and cross-drive analysis. In: Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS). Elsevier, Lafayette, Indiana, August 2006b. http://www.dfrws.org/2006/proceedings/10-Garfinkel.pdf.

Garfinkel Simson L. Carving contiguous and fragmented files with fast object validation. In: Proceedings of the 7th Annual Digital Forensic Research Workshop (DFRWS), August 2007.

Karresand Martin, Shahmehri Nahid. Oscar—file type identiï¬cation of binary data in disk clusters and ram pages. In: Annual Workshop on Digital Forensics and Incident Analysis, pp. 85–94. Springer-Verlag, 2006.

Li WJ, Wang K, Stolfo SJ, Herzog B. Fileprints: identifying file types by n-gram analysis. In: 6th IEEE Informaton Assurance Workshop, June 2005.

McDaniel Mason. Automatic file type detection algorithm. Master's thesis, James Madison University, 2001.

Moody Sarah J, Erbacher Robert F. Sadi: Statistical analysis for data type identification. In: May 2008. Proceedings of the 3rd IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, pp. 41–54, May 2008.

National Institute of Standards and Technology. National software reference library reference data set, http://www.nsrl.nist.gov/; 2005 [Online; Accessed 06.03.09].

Zweig MH, Campbell G. Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. Clin Chem 1993;39:561–77.

Roussev Vassil, Garfinkel Simson. File fragment classification—the case for specialized approaches. In: Systematic Approaches to Digital Forensics Engineering (IEEE/SADFE 2009). Springer, May 2009.

Richard III Golden G, Scalpel Roussev V. A frugal, high performance file carver. In: Proceedings of the 2005 Digital Forensics Research Workshop. DFRWS, August 2005. http://www.digitalforensicssolutions.com/Scalpel/.

Speirs II William R, Cole Eric B. US patent application 20070116267, methods for categorizing input data, May 2007.

Veenman CJ. Statistical disk cluster classification for file carving. In: Proceedings of the First International Workshop on Computational Forensics, August 31 2007.