CS 6v81-05

Digital Forensics III - File Carving

Kevin Hulin

Department of Computer Science
The University of Texas at Dallas

September 23rd, 2011

Outline

Outline

What is File Carving

File Carving is the recovery of files from a digital storage device especially files that are unrecoverable by conventional means.

Historical Context

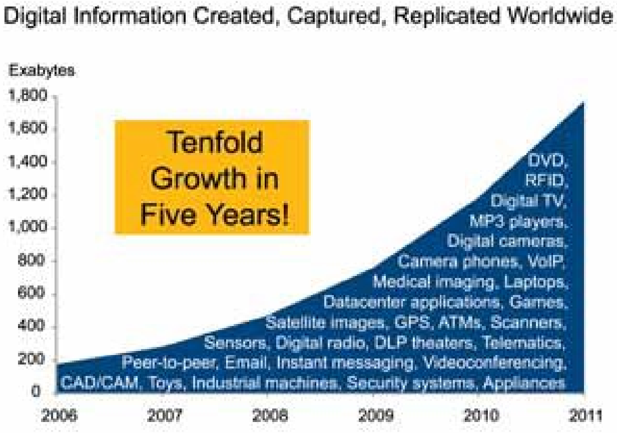
Problems of a physical world

- Important documents are stored in triplicate in multiple locations
- Criminals leave behind incriminating evidence that leads to conviction

Problems of a digital world

- How can we recover important documents that exist only in digital form?
- Where do we find evidence for purely digital crime

Where are we today?



Where are we today?

We're being overwhelmed by data!

- Exponential growth in the amount of data in existence.
- Increasing number of cases that use digital evidence – Evidence has to be admissible!
- Need a way to automate - remove burden from forensic investigator!

What makes File Carving possible?

File deletion and allocation

- When I delete a file, isn't it gone?
- Have to look at how the file system works.

Outline

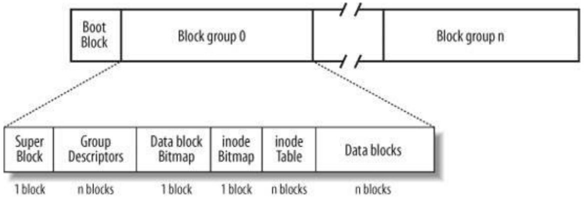
File System overview

File systems

- NTFS, UFS, FAT, EXT2, EXT3, ...
- Jobs: Allocate files, Read files, Maintain meta data about files
- One thing in common – Optimized for performance (not security)
- File deletion isn't that important to the average user

Outline

Unix-based FS Structure



Metadata stored in data structures

- Super Block
  - File system metadata
  - Free inodes
  - Free block-List
- Inode
  - file level metadata
  - pointers to data blocks
- Directory blocks
  - Filename and corresponding inode number

Unix-based FS - File creation and deletion

File Creation

- Inode retrieved from SuperBlock inode list and updated with file meta data
- Free Blocks are allocated and written to (indirect blocks if needed)
- Directory entry added that points to corresponding inode

File Deletion

- Directory entry is marked as deleted (but not removed)
- INode is returned to the inode-list
- Data blocks are added back to the free-block list

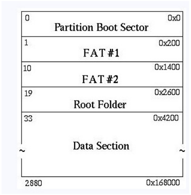
Unix-based FS - Pros and Cons

UFS Evaluation

- Pros
  - Superblock redudency makes file system failure less likely
  - Free-list ensures that file generally occupies contiguous data blocks
- Cons
  - Not supported by competitors (Microsoft)
  - File allocation on a near full disk is likely to be fragmented

Outline

FAT File System Structure



FAT Data structures we're interested in:

- Directory Entry - <FileName, StartingClusterNumber>
- File Allocation Table FAT[i] = nextClusterNumber.
  - FAT[i] = 0x00 // Unallocated
  - FAT[i] = 0xFF // (EOF) End of cluster chain
  - FAT[i] = N // Next cluster in chain is at N
- Each cluster has an entry in the File Allocation Table
- Disk Data Array - Where actual file contents are stored

FAT - File Creation and Deletion

Creating new files

- File system calculates the number of clusters needed for the file
- Stores file in the first N free clusters available
- Results in **Fragmentation**

When a file is deleted:

- Its directory entry is marked as deleted (but not actually removed)
    - The starting cluster data is still there
- The FAT[i] is set to 0x00 for all cluster entries allocated to the file
    - This makes it very hard to recover deleted files (since they could be fragmented)

FAT - Limitations

Out dated

- Microsoft 1980
  - FAT-12, FAT-16, FAT-32
  - Now used primarily in removable memory devices
    - Requires less space for meta data than NTFS
- Max file size = 4GB
- Very slow to determine free space
- No security enhancements

Outline

New Type File System (NTFS) Structure

File structures we're interested in

- Directory information stored in B-Tree
- Bitmap file ($bitmap) stores file allocation information
    - BMP[i] = 0b0 // cluster i is allocated
    - BMP[i] = 0b1 // otherwise
- File Cluster links (like in FAT)

NTFS - File Creation and Deletion

Creating new files

- File data is added to directory in B-tree
- BMP used to find the best place to store the file to avoid fragmentation
- Cluster links are updated to reflect the file's logical ordering and locations

Deleting files

- Associated clusters in BMP are set to 0b0
- Cluster links and file data are left un-touched

NTFS - Evaluation

Performance enhancements over FAT

Created by Microsoft in 1993 to replace FAT

- BMP provides an efficient methods for finding contiguous blocks to store files without fragmentation
- No file size limitation
- File recovery is straight-forward IF
  - The file entry is still present
  - The file cluster numbers are still present
  - File clusters have not been overwritten / allocated to another file

So what does this all mean?

The data is still there!

- Leverage the inherent insecurity of the file system to recover lost files
- Only problem is the file pieces may not be contiguous
- **Problem:** Solve this super massive jig-saw puzzle

Outline

Readings

Structure based approaches to file carving

- Mikus, Nicholas A. "An analysis of disc carving techniques," MS Thesis. Naval Postgraduate School, 2006
- Golden G. Richard and Vassil Roussev. "Scalpel: A Frugal, High Performance File Carver," In DFRWS 2005

Advanced approaches to file carving

- Anandabrata Pal and Nasir Memon. "The Evolution of File Carving," IEEE Signal Processing Magazine, Vol26(2) March 2009

Naive file carving

Consider a giant jigsaw puzzle

- One way to solve it - try every piece with every other piece
- Not a very good (or tractable) idea
  - $O(n!)$
- Consider now that there are looking at a 1 TB drive that has a cluster size of 4 KB...
  - $1TB/4KB \approx 2.6 * 10^8$
  - $(10^8)! > 10^{100000}$

Better ideas

Luckily there are better approaches

- Use known file type structures to classify and intelligently piece together files
- Devise "likeness" weights between chunks to estimate the likelihood that two chucks represent adjacent data
- Classify chunks based on the type of data they contain

File Structure-based Carving

A straight-forward approach

- Compile library of known file headers and footers
- Comb through clusters searching for header blocks
- Match header block with next corresponding footer block
- Recover everything between as our target file

Foremost 0.69

The open-source file carver

- Written by Jessie Kornblum for the US Government Center for Information Systems Security Studies and Research
- Modeled after DOS CarvThis program
- Required user to specify file header and footer data in a config file
- Open source

Foremost 0.69

How it works

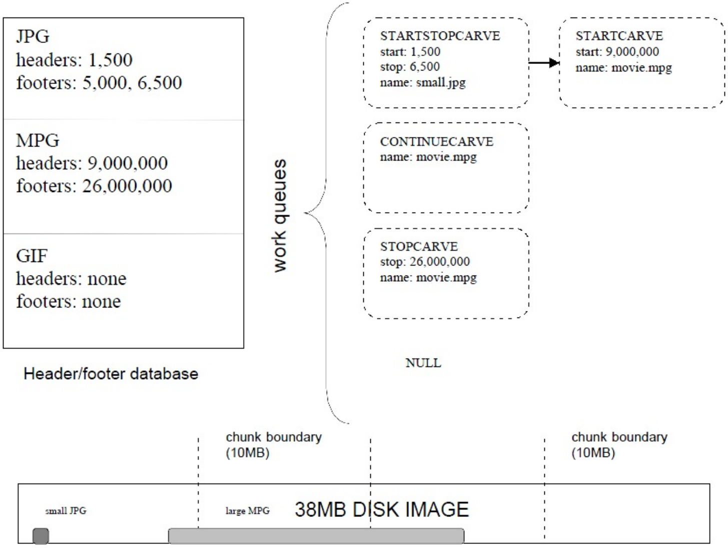
- Comb through disk looking for file headers
- Once header is found, begin building buffer (up to max file size)
- If the footer is found, carve file from header to footer
- Else, cave maximum file size
- Continue search from just past where header was found

Scalpel 1.5

A Frugal File Carver

- Developed in 2005 by Richard and Roussev at UNO (Geaux Louisiana!)
- Complete rewrite of Foremost 0.69
- Goals:
  - Maximize efficiency by minimizing disk reads and writes
  - Minimize disk reads and writes
  - High performance on modest hardware
  - Do not compromise performance
- Developed about the same time as Foremost 1.0
- Current version is 2.0 - Added support for regex, min carve size, multi-threading, and GPU-acceleration

Scalpel 1.5


JPG
headers: 1,500
footers: 5,000, 6,500
MPG
headers: 9,000,000
footers: 26,000,000
GIF
headers: none
footers: none
Header/footer database
work queues
STARTSTOPCARVE
start: 1,500
stop: 6,500
name: small.jpg
STARTCARVE
start: 9,000,000
name: movie.mpg
CONTINUECARVE
name: movie.mpg
STOPCARVE
stop: 26,000,000
name: movie.mpg
NULL
chunk boundary (10MB)
chunk boundary (10MB)
small JPG
large MPG
38MB DISK IMAGE

Scalpel 1.5

How it works

- First, a configuration file stating specs for files to be carved is loaded
- One full pass is made, recording locations for file headers and applicable footers in a database
- For each chunk, a work queue is built to ensure that the same data is not read twice
- Files are written as they are read, resulting in a small memory foot print
- Chunks that are not scheduled for any work are skipped over
- This results in at most two full passes over the disk

Foremost 1.0

Foremost - New and improved!

- Developed in 2005 as part of Nicholas Mikus' Master Thesis at the Naval Postgraduate School
- Goals:
  - Build a robust open source file carving tool for UNIX systems
  - Achieve sophistication beyond simple file header matching
  - Speed
  - Remove unnecessary burdens from the forensics investigator
  - File-system independent
- Mirror `file` command's file type identifying ability
- Current version = 1.5.7

Foremost 1.0

How it works

- Much like Foremost 0.69
- Improvements:
  - `file` utility is used for identifying files
  - File identification heuristics are included for common and complex file types
    - In depth Microsoft OLE document specifications (thanks to Chicago Project)
    - Reverse search mechanism for PDFs with multiple footers
    - Simultaneous checking for multiple GIF versions
    - Sanity checking for BMP size (it's header is only 2 bytes = lots of false positives)
  - Search is done efficiently using Boyer Moore algorithm (Search at string-length offsets)
  - Unix FS indirect blocks are identified and removed from carved file

The `file` utility uses `magic`!

`file <filename>`

- Utilizes `\usr\share\misc\magic`
- `magic` contains syntactic information organized hierarchically for determining a file's type
- Why Unix based file systems are able to operate on files that do not have proper file extensions

ZIP file structure

grep ZIP magic

```
ZIP archives (Greg Roelofs, c/o zip-bugs@wkuvx1.wku.edu)
0 string PK\003\004
>4 byte 0x00 ...  !:mime application/zip
>4 byte 0x09 ...  > v0.9 to extract ...
>4 byte 0x0a ...  > v1.0 to extract ...
>4 byte 0x0b ...  > v1.1 to extract ...
>0x161 string WINZIP ...  WinZIP self-extracting...
>4 byte 0x14
»30 ubelong !0x6d696d65 > ...  v2.0 to extract...
```

PNG file structure

grep PNG magic

```
0 string \x89PNG\x0d\x0a\x1a\x0a PNG image
!:mime image/png
>16 belong x \b, %ld x
>20 belong x %ld,
>24 byte x %d-bit
>25 byte 0 grayscale,
>25 byte 2 \b/color RGB, >25 byte 3 colormap,
>25 byte 4 gray+alpha,
>25 byte 6 \b/color RGBA,
>28 byte 0 non-interlaced
>28 byte 1 interlaced
```

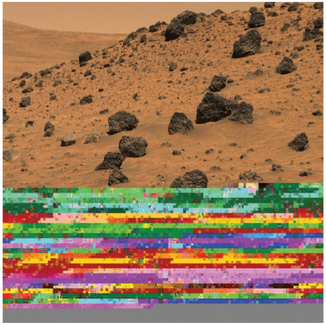
Comparison

Case		Foremost 0.69	Foremost 1.0	Scalpel 1.5
# of passes		$O(N^2)$	$O(N^2)$	$O(MN), M \leq 2$
Time carve 40 GB		9h50m	N/A	2h40m
EXT2 Acc.	perfect	1 of 10	5 of 10	N/A
	partial+	5 of 10	10 of 10	

Results

- In all tests, Scalpel extracted the exact same files as Foremost 0.69 in about half the time
- Foremost 1.0 out performed version 0.69 on all test cases across multiple file systems
- Unfortunately, no data exists comparing Scalpel 1.5 and Foremost 1.0 performance at time of publishing (though Scalpel came in 2nd at DFRWS 2007 Forensics Challenge)

Evaluation



Fast, but not robust

- File Structure based tools are fast
- Able to extract most data files
- Depend on data not being fragmented
- Need something more robust to tackle fragmented files

Outline

A more sophisticated approach

Assuming contiguity is not sufficient

Garfinkel's 2007 fragmentation statistics show that while it is true that files are generally not fragmented, the files that are most likely to be fragmented are those that are forensically important:

- 16% of JPEGS
- 17% of Word Docs
- 22% of AVI
- 58% of PST MS Outlook files

Fragmentation becomes more of a problem when the system is low on disk space, files are appended to, wear-level algorithms are used (e.g. SSDs)

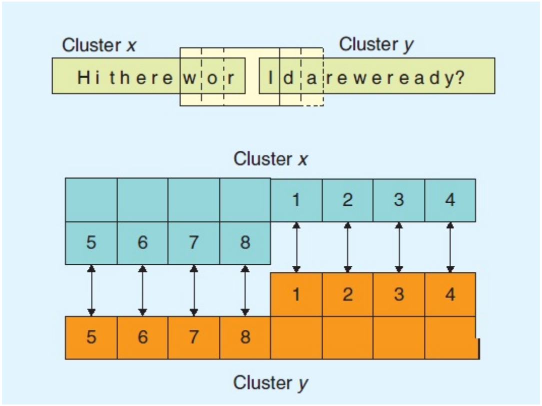
Outline

Graph Theoretic Approach

File Carving = Graph problem

- Approach the problem of reconstructing files as a shortest/longest path problem
- Represent clusters as verticies in the graph
- Weights represent how near/far two clusters are from each other
- Advantages:
  - Graph algorithms are very well known and have been thoroughly researched
  - Fragmentation is irrelevant

Assigning Edge Weights



How do we define edge weights between clusters?

- **Text files:** Sliding window sequence likelihood
- **Image files:** Pixel neighbor difference

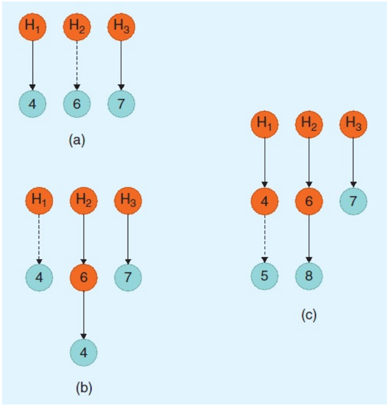
Graph Algorithms

Define a weight function between file clusters

$w_{ij} = SemanticLikeness(cluster_i, cluster_j)$

- Hamiltonian Path - Maximize total cost for trip that touches every node exactly once (NP Hard)
- K Vertex-Disjoint-Path problem - Maximize K disjoint paths (where K = number of file headers) (NP Hard)
  - Shortest Path First (SPF) - Construct and remove the minimal path until all clusters are used
  - Parallel Unique Path (PUP) - Grow all files simultaneously, append best match at each step

Parallel Unique Path



Algorithm:

- Initialize K paths to contain only clusters containing headers
- Until no more clusters remain:
  - Find the best match to append to each path
  - Append the highest scoring match to its corresponding path
  - Remove the matched pair from the pool of clusters

Performance

Good but not efficient

- Shortest Path First - 88% of files reconstructed
- PUP - 85% of files reconstructed
- Too slow in practice
  - Precomputation of adjacency matrix = $O(n^2 log(n))$
  - Consider that there are millions of clusters!

Outline

Bifragment Gap Carving

Another approach to handing fragmentation

- Most files are only split into two fragments
- Uses structure validation to identify bifragmented clusters
- How does it work?
  - Locate corresponding file header and footer no more than M clusters apart
  - Attempt to validate the file over all gap sizes between header and footer
  - Successful validation is a strong indicator of a correct carve
- Note: Validation requires that files be decoded so as to take advantage of file specific error checking

Bifragment Gap Carving

Efficient but limited

- Does not handle distant clusters (requires $O(M^2)$ validations for a gap size of M)
- Cannot be scaled to larger numbers of fragments (approaches $O(2^M)$)
- Validation does not guarantee a successful carve
- Cannot be applied to plain text or BMPs since they cannot be validated
- Lost or distant clusters will result in the worst case time often

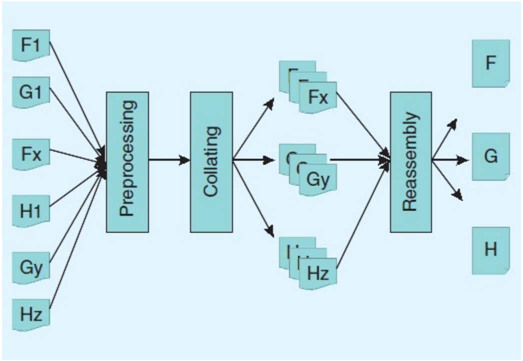
Outline

SmartCarver

Best of both worlds

- Combine graph theoretic approaches with knowledge of how the file system performs in practice
- Focus on the problem of detecting **Fragmentation Points**
- Reduce problem size by eliminating unnecessary comparisons between clusters of different types (e.g. text, image, video, etc)

SmartCarver



How does it work?

- Preprocessing - Remove all encryption and compression
- Collation - Group files based on their classification (text/binary/etc.)
- Reassembly - Discover fragmentation points and re-construct

SmartCarver - Collation

Classification problem

- Pattern Matching - e.g. <html> probably belongs to a HTML doc
- ASCII character frequency - Unlikely that high ASCII freq. belongs to binary data
- Entropy - Abstraction of file contents (not very good, many false positives)

SmartCarver - Collation

Classification problem

- File Fingerprints - Byte Frequency Analysis
  - Byte Frequency Distribution (30%)
  - + Byte Frequency Cross-Correlation (40%)
  - + Header/Footer Matching (95%) (limited to files with headers/footers)
- "Fileprints" - Multiple BFA models per file type
  - Shakespeare and French Novels probably do not share the same byte distribution
  - Achieve 77% - 100% accuracy
  - (Obviously) not as good on shorter texts (< 4KB)
- "Oscar method"
  - Similar to "Fileprint" (byte freq. based models)
  - + Keyword matching (97% accuracy)
  - + Rate of Change (order-based byte analysis) (99% accuracy)

SmartCarver - Reassembly

Finding the fragmentation point

- Neighboring clusters have a higher chance of being contiguous in a file
- Find the point where two clusters are significantly different
    - Keyword/Dictionary approach - Merge until boundry words are not in dictionary
        - Only applicable to certain text files
    - File Structure Merging - Merge files when structure is correct; Find fragmentaiton point when it is not.
        - Not applicable to file types that do not have known file structures
    - Sequential Hypothesis Testing - Parallel Unique Path (SHT-PUP)

SmartCarver - Reassembly

Sequential Hypothesis Testing - Parallel Unique Path

- Extend PUP algorithm to consider sequential clusters before others
- Sequential clusters are grouped through hypothesis testing:
  - Let $\mathbf{W} = W_1, W_2, \ldots$ be the sequence of observed weights between cluster sequence $b = b_1, b_2, b_3 \ldots$
  - Define
    $$\Lambda(\mathbf{W}) = \frac{Pr(\mathbf{W}|H_1\text{:the clusters do not belong together})}{Pr(\mathbf{W}|H_0\text{:the clusters belong together})}$$
  - While $\Lambda(b) \in [t^-, t^+] \&\&!(footer \in b)$
    - append the next cluster to b
  - if $w(b) \geq t^+$ then group b together as a fragment and the test is continued
  - if $w(b) \leq t^-$ or $footer \in b$ then stop test
  - if footer found, then reconstruct file from fragments

SmartCarver - Evaluation

Good but there's room for improvement

- Success! - Was able to carve images from DFRWS test sets that no technique had succeeded on prior
- BUT... dependant on good weight function and accuracy of conditional probabilities in computing $\Lambda(\mathbf{W})$
  - For images, Pixel boundary value difference
  - For text files, Prediction by partial matching (PPM)
  - Other file types were not tested (as good weight functions do not exist)

Outline

Encrypted volumes should be impossible to carve!

Not the case!

- Encryption products such as TrueCrypt aim to make a volume look random
- Carvers can take advantage of this - Sometimes the files are TOO random
- Upon analyzing a TrueCrypt volume, it will be found to have near perfect randomness
- Such randomness does not occur naturally!
- Classify truly random clusters as "encrypted!"

Outline

Where are we headed?

Still becoming a science

- ALOT of room for improvement (reiterated by Pal in *The Evolution of File Carving*)
- Foremost and Scalpel are good enough for most things
- As SSDs become more prevalent, disk fragmentation will worsen and Data Structure based Carving will be phased out

Questions

?

References

- Mikus, Nicholas A. "An analysis of disc carving techniques," MS Thesis. Naval Postgraduate School, 2006
- Golden G. Richard and Vassil Roussev. "Scalpel: A Frugal, High Performance File Carver," In DFRWS 2005
- Anandabrata Pal and Nasir Memon. "The Evolution of File Carving," IEEE Signal Processing Magazine, Vol26(2) March 2009
- S. Garfinkel. "Carving contiguous and fragmented files with fast object validation," in Proc. DFRWS 2007
- Carrier, Brian. *File System Forensic Analysis*. Addison Wesley. 2005.