CS 798: Digital Forensics and Incident Response

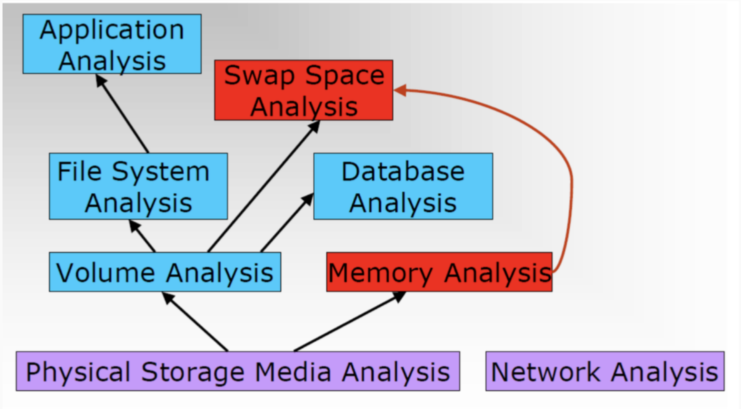
Lecture 9 - Deleted File Recovery and File Carving

Diogo Barradas

Winter 2025

University of Waterloo

We talked about file systems...



Source: „File System Forensic Analysis", Brian Carrier

- Is it possible to recover formerly deleted files?
- How to recover deleted files when no metadata is available?

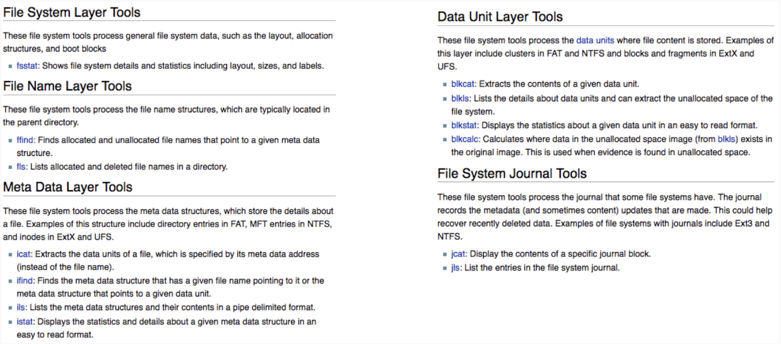
Outline

1. Recovery of deleted files
2. File carving
3. Advanced file carving techniques

Recovery of deleted files

TSK provides analysis tools in each category

- Data evidence categories of the ExtX file system family



File System Layer Tools

These file system tools process general file system data, such as the layout, allocation structures, and boot blocks

- fsstat: Shows file system details and statistics including layout, sizes, and labels.

File Name Layer Tools

These file system tools process the file name structures, which are typically located in the parent directory.

- ffind: Finds allocated and unallocated file names that point to a given meta data structure.
- fls: Lists allocated and deleted file names in a directory.

Meta Data Layer Tools

These file system tools process the meta data structures, which store the details about a file. Examples of this structure include directory entries in FAT, MFT entries in NTFS, and inodes in ExtX and UFS.

- icat: Extracts the data units of a file, which is specified by its meta data address (instead of the file name).
- ifind: Finds the meta data structure that has a given file name pointing to it or the meta data structure that points to a given data unit.
- ils: Lists the meta data structures and their contents in a pipe delimited format.
- istat: Displays the statistics and details about a given meta data structure in an easy to read format.

Data Unit Layer Tools

These file system tools process the data units where file content is stored. Examples of this layer include clusters in FAT and NTFS and blocks and fragments in ExtX and UFS.

- blkcat: Extracts the contents of a given data unit.
- blkls: Lists the details about data units and can extract the unallocated space of the file system.
- blkstat: Displays the statistics about a given data unit in an easy to read format.
- blkcalc: Calculates where data in the unallocated space image (from blkls) exists in the original image. This is used when evidence is found in unallocated space.

File System Journal Tools

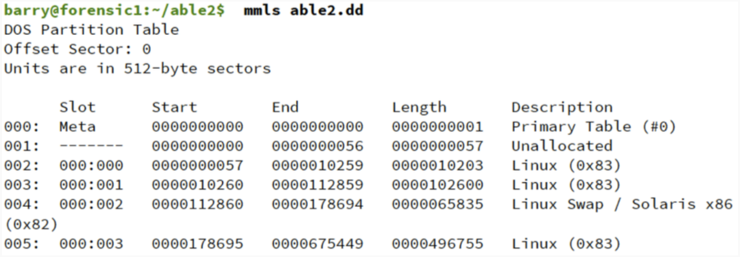
These file system tools process the journal that some file systems have. The journal records the metadata (and sometimes content) updates that are made. This could help recover recently deleted data. Examples of file systems with journals include Ext3 and NTFS.

- jcat: Display the contents of a specific journal block.
- jls: List the entries in the file system journal.

`https://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview`

Example: Deleted file identification and recovery

- The goal will be to identify and recover a deleted file from an Ext2 FS image `able2.dd` using the file's unallocated inode
- List the current partition images using TSK's `mmls` tool

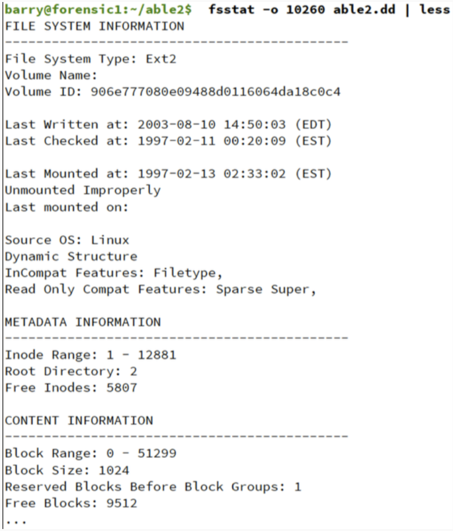


```
barry@forensic1:~/able2$  mmls able2.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000   0000000000   0000000001   Primary Table (#0)
001:  -------   0000000000   0000000056   0000000057   Unallocated
002:  000:000   0000000057   0000010259   0000010203   Linux (0x83)
003:  000:001   0000010260   0000112859   0000102600   Linux (0x83)
004:  000:002   0000112860   0000178694   0000065835   Linux Swap / Solaris x86
(0x82)
005:  000:003   0000178695   0000675449   0000496755   Linux (0x83)
```

- We are looking for information on the root partition (/)
  - Starts at sector 10260, numbered 03 in the `mmls` output

Gather information about the root partition

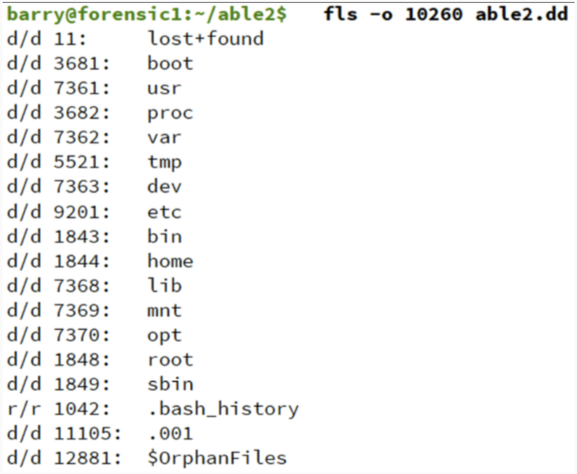
- Run `fsstat` with `-o` 10260 to gather file system information at that offset
- This offset is where the root partition is located



```
barry@forensic1:~/able2$  fsstat -o 10260 able2.dd | less
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext2
Volume Name:
Volume ID: 906e777080e09488d0116064da18c0c4

Last Written at: 2003-08-10 14:50:03 (EDT)
Last Checked at: 1997-02-11 00:20:09 (EST)

Last Mounted at: 1997-02-13 02:33:02 (EST)
Unmounted Improperly
Last mounted on:

Source OS: Linux
Dynamic Structure
InCompat Features: Filetype,
Read Only Compat Features: Sparse Super,

METADATA INFORMATION
--------------------------------------------
Inode Range: 1 - 12881
Root Directory: 2
Free Inodes: 5807

CONTENT INFORMATION
--------------------------------------------
Block Range: 0 - 51299
Block Size: 1024
Reserved Blocks Before Block Groups: 1
Free Blocks: 9512
...
```

List the contents of the root directory

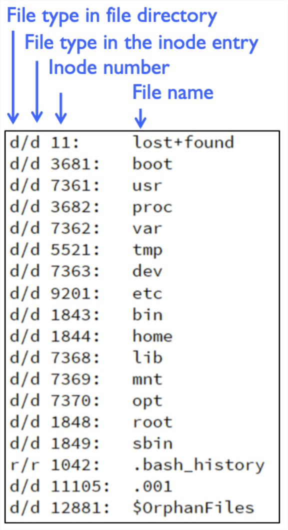
- Run the `fls` command with only the `-o` option and, by default, it will run on the FS's root directory (inode 2)
  - Would also work running: `fls -o 10260 able2.dd 2`



```
barry@forensic1:~/able2$   fls -o 10260 able2.dd
d/d 11:     lost+found
d/d 3681:   boot
d/d 7361:   usr
d/d 3682:   proc
d/d 7362:   var
d/d 5521:   tmp
d/d 7363:   dev
d/d 9201:   etc
d/d 1843:   bin
d/d 1844:   home
d/d 7368:   lib
d/d 7369:   mnt
d/d 7370:   opt
d/d 1848:   root
d/d 1849:   sbin
r/r 1042:   .bash_history
d/d 11105:  .001
d/d 12881:  $OrphanFiles
```

A closer look at the root directory

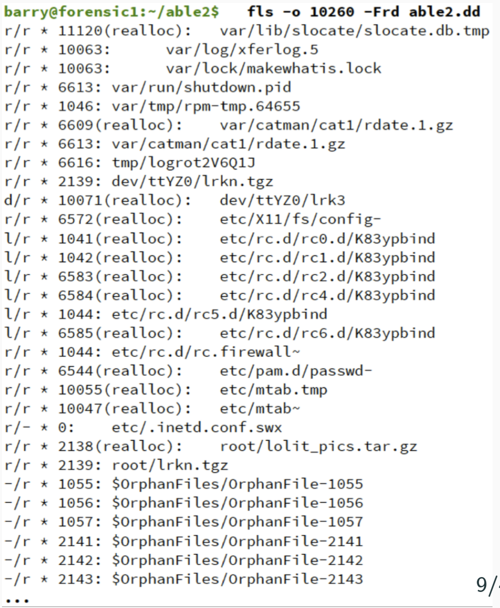
- `.bash_history` is a regular file in both the file's directory and inode entry
  - Its inode is listed as 1042
  - All others are directories
- $OrphanFiles is a virtual folder created by TSK and assigned a virtual inode
  - Contains virtual file entries that represent unallocated metadata entries where there are no corresponding file names
  - Orphan files can be accessed by specifying the metadata address but not through any file name



```
d/d 11:     lost+found
d/d 3681:   boot
d/d 7361:   usr
d/d 3682:   proc
d/d 7362:   var
d/d 5521:   tmp
d/d 7363:   dev
d/d 9201:   etc
d/d 1843:   bin
d/d 1844:   home
d/d 7368:   lib
d/d 7369:   mnt
d/d 7370:   opt
d/d 1848:   root
d/d 1849:   sbin
r/r 1042:   .bash_history
d/d 11105:  .001
d/d 12881:  $OrphanFiles
```

List deleted files

- By default `fls` shows both allocated and unallocated files
- Use `fls (-d)` to see inodes and file names of deleted files only
    - *: file was deleted
    - (realloc): inode reallocated
    - orphan: unallocated inodes



```
barry@forensic1:~/able2$   fls -o 10260 -Frd able2.dd
r/r * 11120(realloc):   var/lib/slocate/slocate.db.tmp
r/r * 10063:      var/log/xferlog.5
r/r * 10063:      var/lock/makewhatis.lock
r/r * 6613: var/run/shutdown.pid
r/r * 1046: var/tmp/rpm-tmp.64655
r/r * 6609(realloc):    var/catman/cat1/rdate.1.gz
r/r * 6613: var/catman/cat1/rdate.1.gz
r/r * 6616: tmp/logrot2V6Q1J
r/r * 2139: dev/ttYZ0/lrkn.tgz
d/r * 10071(realloc):   dev/ttYZ0/lrk3
r/r * 6572(realloc):    etc/X11/fs/config-
l/r * 1041(realloc):    etc/rc.d/rc0.d/K83ypbind
l/r * 1042(realloc):    etc/rc.d/rc1.d/K83ypbind
l/r * 6583(realloc):    etc/rc.d/rc2.d/K83ypbind
l/r * 6584(realloc):    etc/rc.d/rc4.d/K83ypbind
l/r * 1044: etc/rc.d/rc5.d/K83ypbind
l/r * 6585(realloc):    etc/rc.d/rc6.d/K83ypbind
r/r * 1044: etc/rc.d/rc.firewall~
r/r * 6544(realloc):    etc/pam.d/passwd-
r/r * 10055(realloc):   etc/mtab.tmp
r/r * 10047(realloc):   etc/mtab~
r/- * 0:    etc/.inetd.conf.swx
r/r * 2138(realloc):    root/lolit_pics.tar.gz
r/r * 2139: root/lrkn.tgz
-/r * 1055: $OrphanFiles/OrphanFile-1055
-/r * 1056: $OrphanFiles/OrphanFile-1056
-/r * 1057: $OrphanFiles/OrphanFile-1057
-/r * 2141: $OrphanFiles/OrphanFile-2141
-/r * 2142: $OrphanFiles/OrphanFile-2142
-/r * 2143: $OrphanFiles/OrphanFile-2143
...
```

A deeper look at the deleted file entries

- All of the files listed have an * before the inode
    - This indicates the file is deleted
- Some files are annotated with “realloc”
    - The file is marked as deleted, but the inode is in use
    - This means the inode may have been reallocated to a new file

```
r/r * 2138(realloc):  root/lolit_pics.tar.gz
```

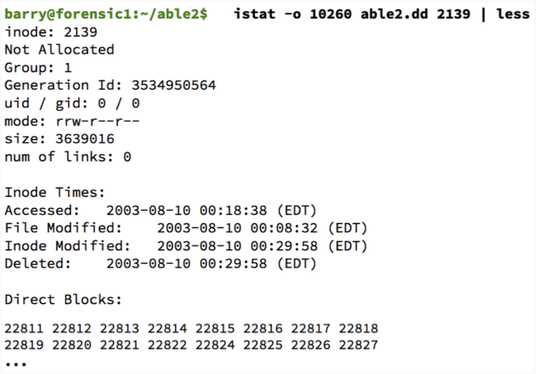
- If “realloc” is not present, both the directory entry and the inode allocated to the deleted file have been unallocated
- Orphan files point to former inode; directory entry reference is not available

Gather information about a file based on inode

- Using metadata (inode) tools, we can learn more information about a deleted file, e.g., `root/lrkn.tgz`

```
r/r * 2139:  root/lrkn.tgz
```

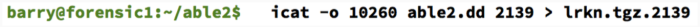
- Use `istat` to gather information about inode 2139



```
barry@forensic1:~/able2$   istat -o 10260 able2.dd 2139 | less
inode: 2139
Not Allocated
Group: 1
Generation Id: 3534950564
uid / gid: 0 / 0
mode: rrw-r--r--
size: 3639016
num of links: 0

Inode Times:
Accessed:   2003-08-10 00:18:38 (EDT)
File Modified:    2003-08-10 00:08:32 (EDT)
Inode Modified:   2003-08-10 00:29:58 (EDT)
Deleted:    2003-08-10 00:29:58 (EDT)

Direct Blocks:

22811 22812 22813 22814 22815 22816 22817 22818
22819 22820 22821 22822 22824 22825 22826 22827
...
```

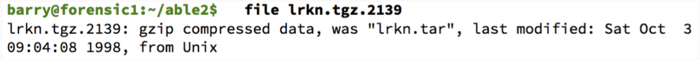
Extract and examine the deleted file

- Use `icat` to send the contents of the data blocks assigned to inode 2139 to a file



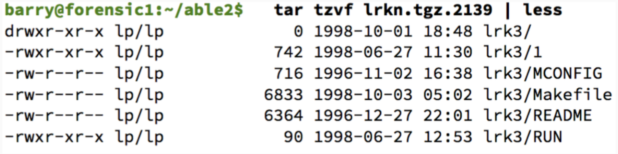
```
barry@forensic1:~/able2$   icat -o 10260 able2.dd 2139 > lrkn.tgz.2139
```

- Check if it really is a compressed archive tgz file



```
barry@forensic1:~/able2$   file lrkn.tgz.2139
lrkn.tgz.2139: gzip compressed data, was "lrkn.tar", last modified: Sat Oct  3
09:04:08 1998, from Unix
```

- List the contents of the archive



```
barry@forensic1:~/able2$   tar tzvf lrkn.tgz.2139 | less
drwxr-xr-x lp/lp             0 1998-10-01 18:48 lrk3/
-rwxr-xr-x lp/lp           742 1998-06-27 11:30 lrk3/1
-rw-r--r-- lp/lp           716 1996-11-02 16:38 lrk3/MCONFIG
-rw-r--r-- lp/lp          6833 1998-10-03 05:02 lrk3/Makefile
-rw-r--r-- lp/lp          6364 1996-12-27 22:01 lrk3/README
-rwxr-xr-x lp/lp            90 1998-06-27 12:53 lrk3/RUN
```

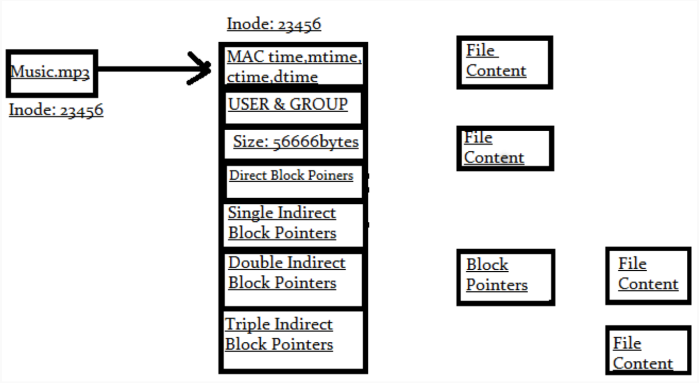
Summary: The general approach

- Start from higher to lower levels of abstraction:
    - Obtain info about the file system (file system category)
    - Obtain info about root folder & file names (file name category)
    - Obtain info about file's inodes (meta data category)
    - Obtain info about file's blocks (content category)

File carving

Deleting a file on the Ext2 file system

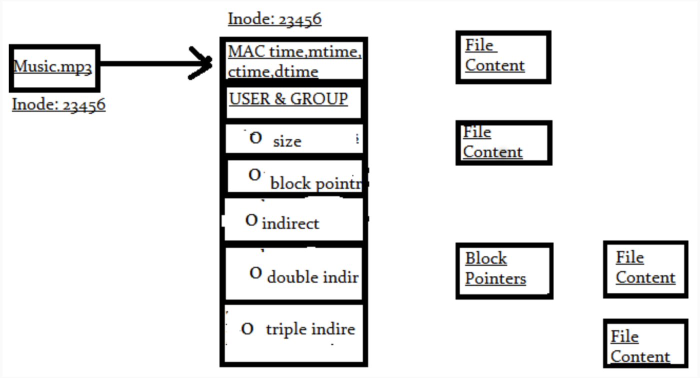
- Inode bitmap is cleared, but block pointers on inode remain unmodified



- As a result, we can fully recover the file from its inode
  - Unless it has been reallocated to another file...

However, on Ext3 file deletion is different

- The block pointers are also zeroed!

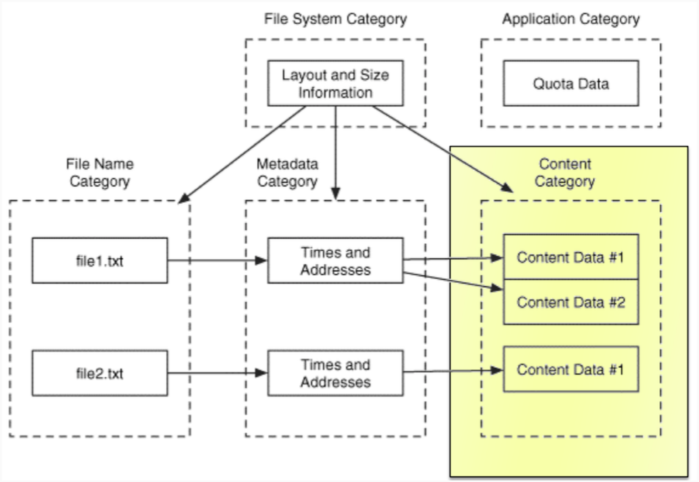


- As a result we can no longer recover the file by reading the block pointers from the inode

Given a raw byte stream, how can we extract the data of a particular file?

There is no metadata present in file system structures...

We can only operate in the content category



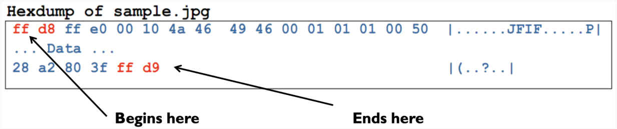
File carving

- File carving is a powerful technique because it can:
    - Identify and recover files of interest from raw, deleted or damaged file systems, memory, or swap space data
    - Assist in recovering files and data that may not be accounted for by the operating system and file system (e.g., when metadata is no longer available, after volume reformatting)

- Carving is a general term for extracting structured data out of raw data, based on format specific characteristics present in the structured data

Key insight: Leverage files' internal structure

- Some file formats have predefined header and footer
    - Include signatures aka "magic numbers" (i.e. byte sequences in known positions)
- File formed by clusters between header and footer (e.g., GIF)
    - Header: `0xFF 0xD8`
    - Footer: `0xFF 0xD9`

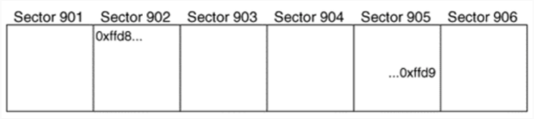


Hexdump of sample.jpg

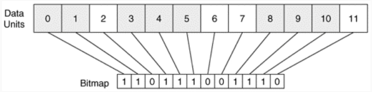
```
ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 50  |......JFIF.....P|
... Data ...
28 a2 80 3f ff d9                                 |(..?..|
```

Let’s search for signatures in unallocated space

- Locate signatures matching the start and end of known file types
- Commonly performed on unallocated space of a FS and allows for recovering files w/o metadata structures pointing to them



- First, isolate the unallocated blocks from the volume (as seen in the last class)
  - `dls`: displays the contents of all unallocated units of an FS



Structure-based carving

- Recover files based on the internal layout of a file
    - E.g., identifier strings, header, footer, and size information
- Known header and footers or maximum file size
    - JPEG: `0xFF 0xD8` header and `0xFF 0xD9` footer
    - BMP: “BM” header but no footer
- If the file format has no footer, a maximum file size is used
- Popular carvers:
    - Scalpel, Foremost and File finder (EnCase)

Examples of popular tools

- Scalpel

```
scalpel -c scalpel.conf -o lost_texfiles stick.dd.img
```

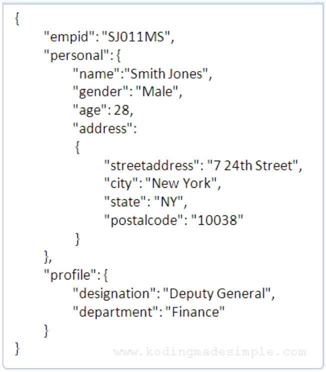
- Foremost

```
foremost -t jpeg,png,zip,pdf,avi -i disk.img -o recov -v
```

Content-based carving

- Identify file content based on internal file contents

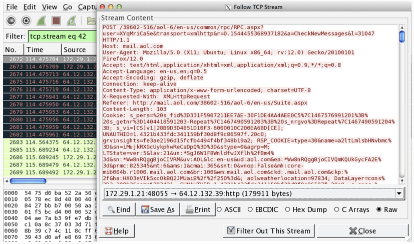
- Content structure
  - Loose structure (HTML, XML)
- Content characteristics
  - Text/Language recognition
  - Statistical attributes
  - Information entropy



```
{
    "empid": "SJ011MS",
    "personal": {
        "name":"Smith Jones",
        "gender": "Male",
        "age": 28,
        "address":
        {
            "streetaddress": "7 24th Street",
            "city": "New York",
            "state": "NY",
            "postalcode": "10038"
        }
    },
    "profile": {
        "designation": "Deputy General",
        "department": "Finance"
    }
}
```

Data carving is applicable beyond file systems

- Can carve any piece of data from raw data blob
- Examples:
  - Files from network streams
  - Individual packets from network traces
  - Malware code from compromised application



Filter: tcp.stream eq 42

```
No.  Time        Source
2672 114.475704  172.29.1.2
2673 114.475709  172.29.1.2
2674 114.475713  64.12.132.
2675 114.475716  64.12.132.
2676 114.475719  64.12.132.
2677 114.475941  64.12.132.
2678 114.475950  64.12.132.
2679 114.475953  64.12.132.
2680 114.475956  64.12.132.
2681 114.475959  64.12.132.
2682 114.475961  64.12.132.
2683 114.564375  64.12.132.
2685 115.689234  64.12.132.
2686 115.689245  172.29.1.2
2687 115.689479  64.12.132.
2689 115.689492  172.29.1.2

0000  54 75 d0 ba 52 2a 50
0010  05 78 ec 8d 40 00 40
0020  84 27 bb b7 00 50 aa
0030  01 f5 bc d4 00 00 52
0040  04 ae 7a b3 9f e7 db
0050  c1 0a 8c 37 03 3d 71
0060  8b 39 c7 4c 11 8c ff
0070  39 43 d0 af e8 69 73
```

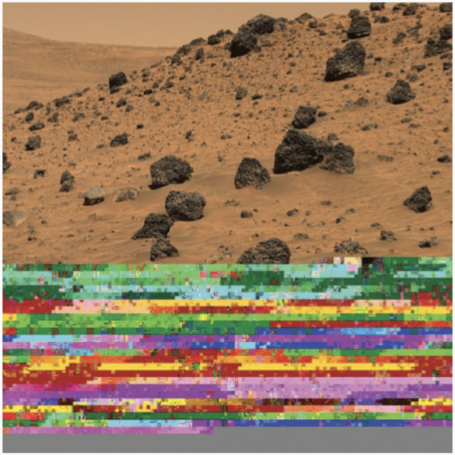
```
POST /38602-516/aol-6/en-us/common/rpc/RPC.aspx?
user=XYqMriCaSe&transport=xmlhttp&r=0.1544455368937182&a=CheckNewMessages&l=31047
HTTP/1.1
Host: mail.aol.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:12.0) Gecko/20100101
Firefox/12.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://mail.aol.com/38602-516/aol-6/en-us/Suite.aspx
Content-Length: 103
Cookie: s_pers=%20s_fid%3D331F5907211EE7AE-30F1DE4AA4AEE0C5%7C1467576991201%3B%
20s_getnr%3D1404418591203-Repeat%7C1467490591203%3B%20s_nrgvo%3DRepeat%7C1467490591204%
3B; s_vi=[CS]v1|28B9D3D4851D10F3-6000010C200EA68D[CE];
UNAUTHID=1.4321b433fdc341159bf30d8f9c86597f.20c0;
grvinsights=fe3ae2196d15fcfb4494f4bf348b19a2; RSP_COOKIE=type=30&name=a2ltLmlsbHNvbmc%
3D&sn=iMujkRXGcUykphwReCaDpQ%3D%3D&stype=0&agrp=M;
LTState=ver:5&lav:21&un:*SgJ6W1F8WnldfwJXflh%2fBm4%
3d&sn:*Ww8nRQggBjoCIVRM&uv:AOL&lc:en-us&ud:aol.com&ea:*Ww8nRQggBjoCIVQmKQUkGycFA2E%
3d&prmc:825345&mt:6&ams:1&cmai:365&snt:0&vnop:False&mh:core-
mib004b.r1000.mail.aol.com&br:100&wm:mail.aol.com&ckd:.mail.aol.com&ckp:%
2f&ha:HXO3eVIk5xcOkBQ2JMUaiB%2f%2f250%3d&; aolweatherlocation=97834; DataLayer=cons%
```

172.29.1.21:48055 → 64.12.132.39:http (179911 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays [✓ Raw]

Advanced file carving techniques

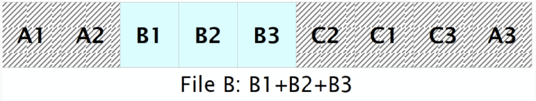
Here's an example of a poorly carved image file



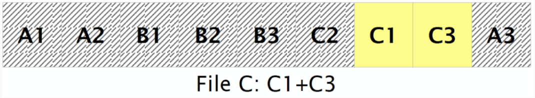
What happened?

Carving was supposed to be easy, right?

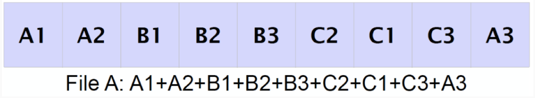
- Issue: Fragmentation
- Normally, files are broken up and stored into clusters
    - For file B, carving clusters sequentially yields correct results



- But data clusters may be out of order



- Or be interleaved with clusters of other files

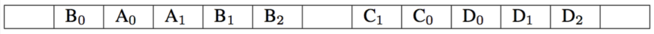


Assuming cluster continuity is not sufficient

- File are generally not fragmented, but those that are most likely to be are those that are forensically important:
    - According to some studies, 16% of JPEGS, 17% of Word Docs, 22% of AVI, 58% of MS Outlook files
- Fragmentation becomes more of a problem when:
    - The system is low on disk space
    - Files are appended to
- Signature false positives
    - Some files may have header signatures or the footer signatures occurring perhaps several times within the file!

Exercise: Which files can be entirely recovered?

- Consider the following unallocated disk space containing clusters of four deleted files.
  - HTML files – FileA: A_0, A_1
  - JPEG files – File B: B_0, B_1, File C: C_0, C_1, File D: D_0, D_1, D_2

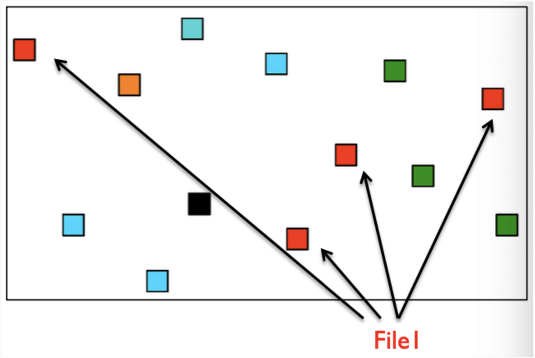


	B_0	A_0	A_1	B_1	B_2		C_1	C_0	D_0	D_1	D_2	

- The following list provides relevant details about their file formats:
  - HTML: no header and no footer, content follows HTML syntax
  - JPEG: header and footer, content must be decoded

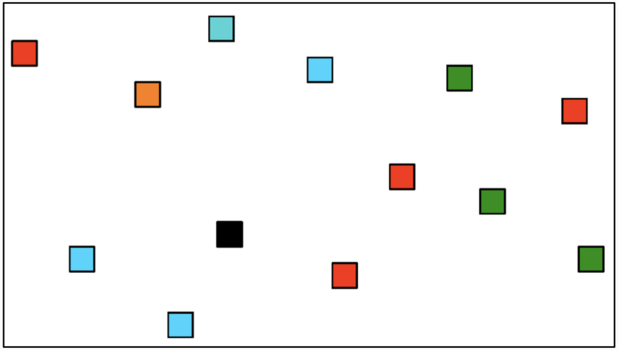
The nature of the problem

- Assume randomized clusters containing file fragments
- How to extract the files?
    - One way to solve it - try every piece with every other piece
- Not a very good (or tractable) idea
    - O(n!)



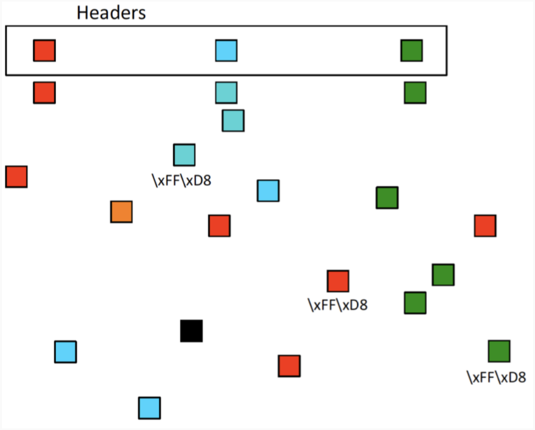
Parallel Unique Path (PUP)

- Key insight behind the PUP algorithm
  - Grow all files simultaneously, append best match at each step
- Initial state: assume all file clusters are randomized



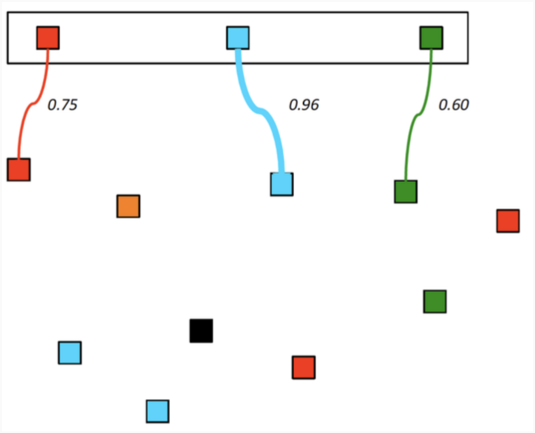
PUP first step: Locate file headers

- Identify headers using keywords / signatures
  - Consider 3 JPEG files
  - e.g., JPEG header is `0xFF 0xD8`



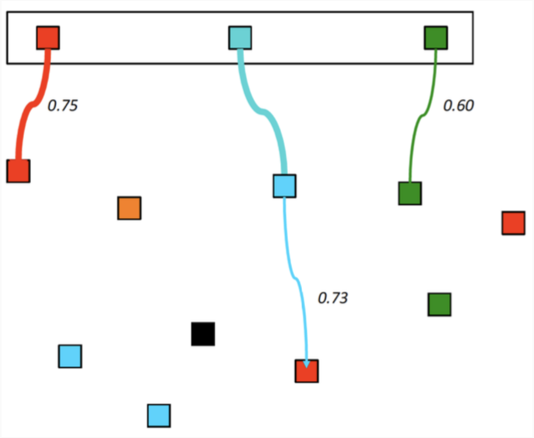
PUP steps: Assign weights

- For each header find best match (using matching metric)
- Choose the best overall match



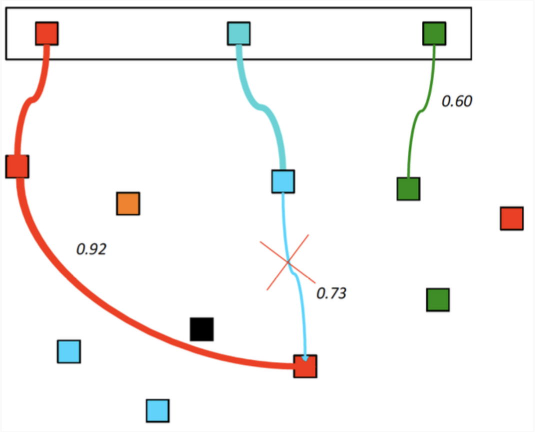
PUP steps: Continue match finding

- Find best match for recently added node
- Choose the best overall match again



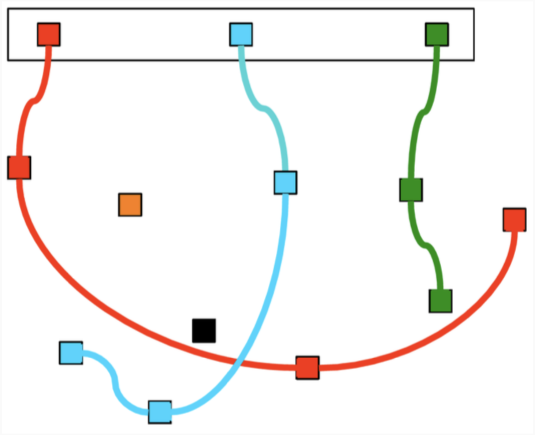
PUP steps: Paths may cross each other

- Repeat process
- Now a block is the best match for two files
- Choose the better of the two and continue



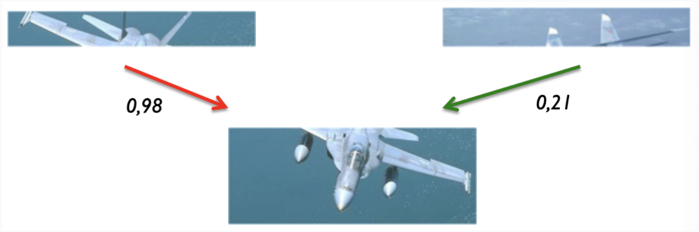
PUP steps: Finish the graph traversal

- Repeat until all files are built or no more nodes can be chosen



Example of a matching metric

- For images: look at the boundary formed by the addition of a new block
- Example:



PUP criticism

The good

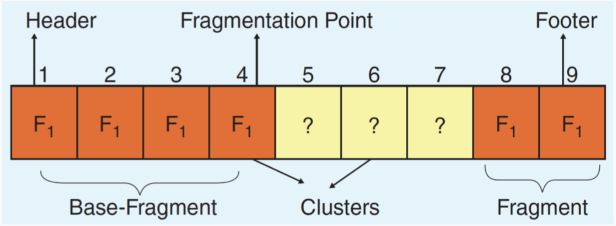
- Realistic
  - Each cluster usually belongs to a unique file
- Effective
  - 85% of files reconstructed

The bad

- Errors propagate in cascade
  - An incorrect cluster leads to the wrong reconstruction of two files
- Still slow in practice
  - Weight computation complexity: $O(n^2 log(n))$
  - Millions of clusters

Bifragment gap carving (BGC)

- One of the first carving techniques to efficiently recover data from real-world data sets
- Leverage an observation that bifragmentation (two fragments only) is the most common fragmentation type
    - Although files fragmented into +three pieces are not uncommon



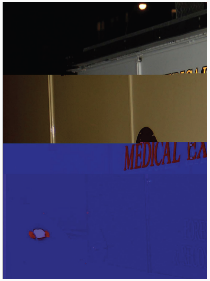
- BGC's goal: try to match both fragments of each file

BGC's key insights

- How to locate the header and the footer?
  - Use magic numbers for well know file formats
- How to ensure that header and footer fragments are properly sequenced?
  - Using fast object validation technique: verify if a file obeys the structured rules of its file type
  - Use consistency checks: error correction, size mismatch, etc.
    - e.g., PNG format has CRC at the file ending
  - Can use file-type specific decoders
    - e.g., JPEG, MPEG, ZIP, etc.

Disadvantages of BGC

- BGC performs satisfactorily when the two fragments are close to each other
- However, it has limitations in general case:
    - It only works for files of two fragments
    - It only works for files that can be validated
        - E.g., plain texts and BMPs cannot be recovered this way
    - Correct validation does not mean coherence/correctness
        - e.g., images that use same codec parameters



Concerns when designing a carving tool

- Carving quality
- Performance
- Memory and space efficiency

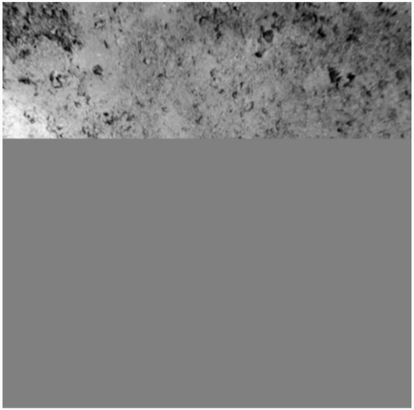
Tool quality: DFRWS 2006 dataset

- Quality metrics:
  - **Recall**: What proportion of the available files is recovered?
  - **Precision**: What proportion of the recovered files is correct?

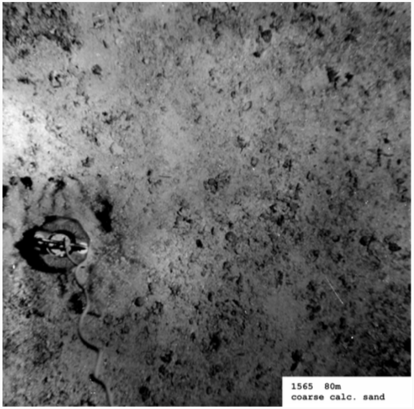
Tool	Carving Recall	Carving Precision
FTK 3.0	0	0.001
Scalpel	0.219	0.28
Encase 6.7	0.219	0.28
FTK 1.81	0.25	0.258
Foremost	0.281	0.36
Photorec	0.563	0.643
Revit	0.625	0.69

What does this mean in practice?

Encase on DFRWS 2006



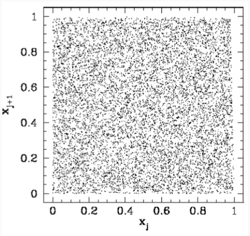
Photorec on DFRWS 2006



Carving encrypted volumes should be impossible!

- Tools such as TrueCrypt aim to make a volume look random
- Carvers can leverage this if the files are TOO random

- TrueCrypt volume analysis will reveal a near perfect randomness
  - Such randomness does not occur naturally!
- Can classify truly random clusters as "encrypted"!



Takeaways

- File carving is a file system analysis technique that faces many challenges in order to identify and retrieve file content, mostly due to data fragmentation issues
- Despite the considerable advances in data carving, there is still a lot of room for improvement, being data carving amongst the hottest topics in forensics research

Pointers

- **Textbook:**
  - Carrier – Chapter 8.7, Casey – Chapter 15.3.1
- **Other resources**:
  - Anandabrata Pal and Nasir Memon. "The evolution of file carving - the benefits and problems of forensics recovery". IEEE Signal Processing Magazine, 26(2):59-71, March 2009
- **Acknowledgements:**
  - Slides adapted from Nuno Santos's Forensics Cyber-Security course at Técnico Lisbon