

# CMSC389R

## Forensics II



**COMPUTER SCIENCE**  
UNIVERSITY OF MARYLAND



## midterm

- Midterm grades released
- We cannot give them back yet as we have makeups

# forensics I refresher

- Two stages: **recovery/extraction** and **analysis**
- Recovery tools:
  - Physical media: dd, EnCase, FTK
  - “Live” sources:
    - Network: Wireshark, tcpdump
    - Memory: /dev/{k,}mem, /proc/\$pid/maps
- Analysis tools:
  - Content-agnostic: strings, binwalk, file
  - Content-aware: Wireshark (again)
  - Metadata: exiftool
  - Steganography: steghide, stegdetect

## today's topics

- Network Analysis techniques
  - Capturing network traffic
  - Using wireshark
- In-depth analysis techniques
  - Carving techniques/algorithms
- Custom binary parsing
  - Common binary layouts
  - Aside: Secure parsing techniques

## data recovery: “live” sources

- Wiretaps, data hoses, live RAM on a seized PC
  - ...or just a file containing one of these
- Various tools/interfaces for generating these:
  - Net: Wireshark, tcpdump, a physical tap
    - Often saved in pcap format.
  - RAM: /dev/{k,}mem, /proc/\$pid/maps/
    - root can read the whole address space!

## data recovery: “live” sources

- We can collect all traffic on an **interface** with a single tcpdump command:

```
$ sudo tcpdump -w mydump.pcap -i eth0
```

### Legend:

- `-w <file>` : the file to write the pcap to
- `-i <ifce>` : the interface to listen on
  - Check `ifconfig` to see your interfaces!

## content-aware analysis: packet captures

- Packet captures often contain *all* traffic on a given interface over a period of time
  - A *lot* of data!
- To make sense of captures, we can use the filtering features of wireshark or tcpdump.
- Things we might want to filter for:
  - http / https - HTTP(S) connections
  - ip.src == 1337 / ip.dst == 1337 - port traffic

# packet capture analysis - live demo

Let's tear a packet capture apart and see what  
kind of data we can find

\*HW hint: Follow → TCP Stream



## analysis: physical media

- The goal: extract as many files as possible from the image for later analysis
- The problem: lots of data, limited time and RAM
- The solution: **carve** the disk image
- Carving?
  - Extracting **interesting signatures!**  
(recall forensics 1)
  - Involves file format **parsing**

# file carving algorithms

- (Naive) serial algorithm:
  - Inputs:  $K$  GBs of input data, 1 worker
  - Give the worker all  $K$  GBs, either as a big array of bytes or as a (seekable) input stream
  - Worker scans the entire input for **signatures**
  - Worker attempts to **extract** (or at least report) the correctly sized chunk of data for each signature
  - Pros: Simple to implement, fast for small  $K$
  - Cons: CPU bound, slow for large  $K$

# file carving algorithms

- (Naive) parallel algorithm:
  - Inputs:  $K$  GBs of input data,  $N$  workers
  - Give each worker  $K/N$  GBs of the input
  - Each worker scans their section of the input for **signatures**
  - Each worker attempts to **extract** (or at least report) the correctly sized chunk of data for each signature
  - Not I/O bound, so we can use threads!
  - **There's a major problem with this algorithm!**
    - What happens when a file of interest gets split between two workers?

# file carving algorithms

- (Better) parallel algorithm:
  - Inputs:  $K$  GBs of input data,  $N$  workers
  - Give each worker  $K/N$  GBs of the input
  - Assign each worker a specific magic byte to look for
  - Each worker scans their section of the input for *their signature*
  - Each worker attempts to **extract** (or at least report) the correctly sized chunk of data for each signature
  - Possibly use GPUs to speed up
  - **There are drawbacks to this approach!**
    - [https://www.dfrws.org/sites/default/files/session-files/pres-massive\\_threading\\_-\\_using\\_gpus\\_to\\_increase\\_the\\_performance\\_of\\_digital\\_forensics\\_tools.pdf](https://www.dfrws.org/sites/default/files/session-files/pres-massive_threading_-_using_gpus_to_increase_the_performance_of_digital_forensics_tools.pdf)

## interesting signatures

- Signatures come in all forms, but we're specifically interested in **file signatures**
  - But remember the others: emails, domains, &c
- **Magic bytes** are the most common file signature
  - A (mostly) fixed sequence of bytes that identify a particular type of file (PDF, JPEG, EXE)

## magic bytes

- Magic bytes are at the beginning of a file\*
- Examples of file magics:
  - ZIP: `\x50\x4B\x03\x04` (PK..)
  - ELF (Linux exec): `\x7F\x45\x4C\x46` (.ELF)
  - MZ (Windows exec): `\x4D\x5A` (MZ)
  - Mach-O (macOS exec): `\xCA\xFE\xBA\xBE`
  - PDF: `\x25\x50\x44\x46` (%PDF)

\* usually (some formats use footers)

# magic bytes

0000000	25	50	44	46	2D	31	2E	34	0A	25	C7	EC	8F	A2	0A	35	20	30	20	6F	62	6A	0A	3C	3C	2F	4C	65	6E	67	74	68	20	36	20	30	20	52	3E	3E	0A	73	74	72	65	%PDF-1.4.%...5 0 obj.<</Length 6 0 R>>.stre
00000001	61	6D	34	30	2E	30	39	33	37	35	20	77	0A	30	2E	31	31	20	30	2E	32	31	36	20	30	2E	35	34	31	20	52	47	0A	30	2E	31	31	20	30	2E	32	31	36	20	30	am.0.09375 w.0.1 0.216 0.541 Rg.0.11 0.216 0
00000005	2E	3D	34	31	20	72	6A	0A	2E	30	2E	30	20	33	30	35	2E	30	30	20	6D	0A	32	32	38	2E	30	30	33	30	35	2E	30	30	20	6C	0A	32	32	38	2E	30	30	541 rg.0.00 30.00 m.228.00 305.00 l.1.228.00		
00000008	2E	30	30	20	6A	0A	30	2E	30	2E	30	20	30	2E	30	30	20	6C	0A	30	2E	30	30	33	30	35	2E	30	20	6C	0A	66	0A	30	2E	38	37	35	20	30	2E	77	34	0.00 1.0.00 0.00 1.0.00 305.00 l.f.0.875 0.74		
000000b7	39	26	34	37	34	31	20	52	47	0A	30	2E	38	37	35	20	30	2E	37	34	39	20	30	2E	37	34	31	20	72	67	0A	37	30	2E	35	33	20	32	36	20	30	30	0.9 0.741 Rg.0.875 0.749 0.741 rg.70.53 260.03			
000000e1	6D	0A	37	32	2E	36	31	20	32	36	36	2E	37	30	20	37	31	2E	35	39	20	32	37	34	2E	37	34	20	37	36	2E	37	39	20	32	38	30	2E	31	36	20	63	0A	38	32	
0000010e	2E	30	37	20	32	37	35	2E	39	32	20	38	35	2E	35	36	20	32	36	39	2E	37	34	20	39	31	2E	36	35	20	32	36	2E	34	38	20	63	0A	39	35	2E	38	32	20	0.7 275.92 85.56 269.74 91.65 266.48 c.95.87	
0000013b	32	36	37	2E	36	39	20	31	30	2E	31	37	20	32	36	37	2E	31	38	20	31	30	34	2E	33	33	2E	32	36	36	2E	32	37	20	63	0A	31	30	34	2E	36	30	20	32	267.69 100.17 267.18 104.33 266.27 c.104.60 2	
00000168	36	34	2E	36	38	20	31	30	34	2E	37	35	20	32	36	33	2E	30	36	20	31	30	34	2E	36	33	20	32	36	31	2E	34	35	20	63	0A	31	30	33	2E	32	37	20	32	36	64.68 104.75 263.06 104.63 261.45 c.103.27 26
00000195	30	2E	30	38	20	31	30	33	2E	39	32	20	32	35	38	2E	36	35	20	31	30	2E	31	31	20	32	35	37	2E	38	39	20	63	0A	31	30	31	2E	36	34	20	32	35	39	0.08 101.92 258.65 100.11 257.89 c.101.64 259	
000001c2	2E	37	35	20	31	30	33	2E	35	30	20	32	36	31	2E	37	32	20	31	30	32	2E	39	37	20	32	36	34	2E	33	35	20	63	0A	39	38	2E	37	30	20	32	36	34	2E	36	75 103.50 261.72 102.97 264.35 c.98.70 264.6
000001ef	33	20	39	34	2E	34</																																								



## magic bytes

- Why are file magics used?
  - File extensions are easy for users to modify, so trusting them can result in attempting to parse the wrong format
  - Provides us a way to version file formats: users don't care about GIF87a vs. GIF89a, they just want their .gifs to open
  - **Not** a panacea: formats can have conflicting magics (see: Java classfiles and Mach-O)
    - Heuristics required!



# file extraction

- We need to know how large a file is/where it ends in order to extract it from the image!
- Techniques vary by file format:
  - Look for “footer” bytes (`\xFF\xD9` for JPEG)
  - Attempt to calculate the file’s size
    - Involves **parsing** a subset of the file
  - Heuristics:
    - Trim files after  $K$  MBs, note offset for later retrieval if the file isn’t correct

# binary parsing

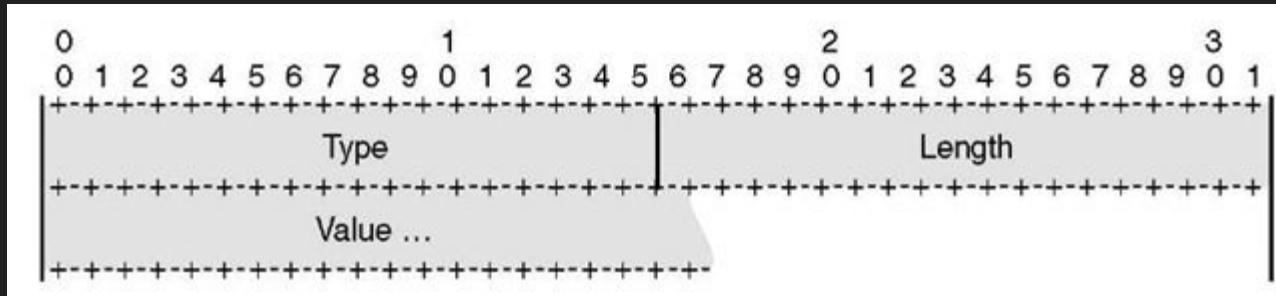
- Two views: carving view and “correctness” view
  - Carving: Just enough to extract correctly
  - Correctness: Understand the whole file
  - We usually end up with a mix of the two
- Unusual formats commonly pop up during forensics:
  - Camera-specific “RAW” photos (often TIFF-based)
  - Custom application formats (intermediate files)
  - Ancient/insane formats like pre-03 .docs
    - OLE Compound Files

# binary parsing: techniques

- Most binary files are divided into (at least) two components: a **header** and a **body**
- Header contains metadata (file magic, version, ...)
  - Usually **fixed-length**, e.g.: 4 bytes for magic, 4 for version, 8 for UUID, etc.
  - Can contain parsing directives (endianness, how much data to expect)
- Body contains file content (photo data, ...)
  - Often **variable-length**, with discrete sections tightly packed together

# binary parsing: techniques

- Many formats use a type-length-value (TLV) layout
- Type and length fixed at  $K$  bytes, value variable
  - Type indicated by a number, which gets looked up in a table (1 for text, 2 for array, etc...)
- Does the length field give you the length of value, or value +  $2K$ ?
  - Depends on the format!



# binary parsing: techniques

- How do we actually write a binary parser?
- Many scripting languages provide `pack/unpack`:
  - `array.pack(format) -> binary string`
  - `binstr.unpack(format) -> array`
  - Options for endianness, padding, ...
- C makes structure packing/unpacking “easy”:
  - Packing: write `sizeof(type)` bytes to the stream
  - Unpacking: cast the incoming bytes + offset to your type
  - Don't use C for this week's assignment!

## aside: secure binary parsing

- Parsers need to be *resilient*:
  - Shouldn't crash on weird inputs (exit gracefully)
  - Shouldn't hang (detect and avoid cycles)
    - ZIP bomb\*
  - Shouldn't leak memory (buffer overruns, overflows)
- Many security problems stem from *trusting input*:
  - Trusting specified length instead of the spec
    - Specs often limit values to a maximum length!
  - Not checking whether a specified length is beyond the input buffer

“... 1.3GB file full of zeroes, compress that into a ZIP file, make 10 copies, pack those into a ZIP file, and repeat this process 9 times ... when uncompressed completely, produces an absurd amount of data without requiring you to start out with that amount.”

## homework #6

has been posted.

Let us know if you have any questions!

This assignment has 1 part (project).

It is due by 11/1 at 11:59PM.