

# CMSC389R

## Forensics I



**COMPUTER SCIENCE**  
UNIVERSITY OF MARYLAND



## announcements

Midterm next week

14 multiple choice, 1 bonus

64 total points

---

Any questions?

## today's topics

- Digital forensics overview
- Data recovery and/or extraction
- Basic analysis techniques (tool-based)
  - Metadata scanning
  - String/pattern searching
  - Steganographic analysis

# what is (digital) forensics?

- The **recovery/extraction** and **analysis** of data from a storage medium
- Who does digital forensics?
  - Law enforcement: local and federal crimes
  - The courts: legal discovery
  - Malware analysts
  - Pentesters





# data recovery/extraction

- Various sources:
  - physical media (HDDs/SSDs/thumb drives/SIM)
    - Even partially destroyed media!
  - storage formats (disk images, archives)
  - “live” sources (RAM dumps, packet captures)

1 0.0000000...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
2 0.0002853...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
3 0.0003035...	192.168.1.113	52.54.193.39	36214 → 443 [ACK] Seq=1 Ack=4381 Win=124 Len=0	TCP
4 0.0006434...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
5 0.0006511...	192.168.1.113	52.54.193.39	36214 → 443 [ACK] Seq=1 Ack=7301 Win=130 Len=0	TCP
6 0.0009709...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
7 0.0009781...	192.168.1.113	52.54.193.39	36214 → 443 [ACK] Seq=1 Ack=10221 Win=136 Len=0	TCP
8 0.0013290...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
9 0.0013360...	192.168.1.113	52.54.193.39	36214 → 443 [ACK] Seq=1 Ack=13141 Win=141 Len=0	TCP
10 0.0015912...	52.54.193.39	192.168.1.113	Application Data	TLSv1.2
11 0.0091438...	192.168.1.113	52.54.193.39	Application Data	TLSv1.2
12 0.0203207...	52.54.193.39	192.168.1.113	443 → 36154 [ACK] Seq=1 Ack=728 Win=132 Len=0	TCP
13 0.0401375...	192.168.1.113	52.54.193.39	36214 → 443 [ACK] Seq=1 Ack=14350 Win=144 Len=0	TCP
14 0.0689534...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
15 0.0689798...	192.168.1.113	52.54.193.39	36154 → 443 [ACK] Seq=728 Ack=2921 Win=398 Len=0	TCP
16 0.0692278...	52.54.193.39	192.168.1.113	[TCP segment of a reassembled PDU]	TCP
17 0.0692425...	192.168.1.113	52.54.193.39	36154 → 443 [ACK] Seq=728 Ack=7301 Win=398 Len=0	TCP

## data recovery: physical media

- Copying files is not sufficient!
  - We want deleted files, bad sectors\*, ...
- We need to take an **image**:
  - Byte-by-byte copy of the media
  - Don't attempt to interpret, just save
  - Common extensions: `.iso`, `.img`
    - Your OS install image is one of these!

\* Copying bad sectors requires us to go beneath OS-level copies



## data recovery: physical media

```
$ sudo dd if=/dev/sdX of=sdX.img bs=4M
```

- Legend:
  - if= : input file (SATA device in ex.)
  - of= : output file (.img file in ex.)
  - bs= : block size (4MB here, for speed)
- See man dd for more information



## data recovery: formats

- Sometimes we just want a single file/archive
  - Which means we'll need to know how to parse different file formats (next lecture!)
- Copy the file, tear it apart
  - Record file metadata (when/who created, etc)
  - Automated analysis (in a bit)

## data recovery/extraction

- We have the data; now what?
- Golden rule(s) for digital forensics:
  - All good forensics begins with a **verified copy**
  - Never do forensics work on the “master”
  - If you can't verify your copy, it's worthless (legally speaking)
- So: how do we **copy** and **verify**?

## copy and verify

- First, take a **strong cryptographic hash** of the image file (more on this in crypto):

```
$ sha512sum sdX.img
```

- Give the hash to a **trusted/authoritative party**
  - e.g., legal counsel
- Copying: use `dd` again (`if=sdX.img`), or `cp`!
- Make sure your copy has **the same hash**!

## copy and verify

- What does this accomplish?
  - Makes **tampering** difficult: third party has own copy and hash, so modifications appear as hash changes
  - Not just about trust: keeps forensic samples from being mixed up, corrupted
    - Lots of legal cases are lost because of unverified digital evidence (analysis done on file w/ different hash)

# analysis techniques

- How should we analyze our copy?
  - Depends on what we're looking for!
- Content-agnostic analysis
  - Often just string/pattern searching
  - Generally less accurate/more verbose
- Content-aware analysis
- Metadata analysis
- Steganographic analysis

## analysis techniques: content-agnostic

- We often don't know what we're looking for...
  - So we turn to generic tools!
- General strategy:
  - Maintain a list of **interesting signatures**\*
  - Scan input for each signature
  - Give the user a list of offsets  
corresponding to signature matches

\* or just look for ranges: ASCII, decimals, UTF-8, ...

# content-agnostic analysis: strings

- `strings` does exactly what it says:
  - Finds strings of printable characters in binary files
  - Options for min length, encoding, whitespace
  - Some additional executable searching features
    - Why might this be a problem? (hint: `libbfd`)
  - Available on pretty much every Linux/\*nix

```
$ strings /bin/bash # what does this do?
```



```
$ strings -n 15 /bin/bash
```

```
show-all-if-unmodified  
skip-completed-text  
$else found without matching $if  
completion-prefix-display-length  
print-completions-horizontally  
/var/tmp/rltrace.%ld  
readline_callback_read_char() called with no handler!  
unknown expansion error  
event not found  
bad word specifier  
substitution failed  
unrecognized history modifier  
no previous substitution  
LS_COLORS: syntax error: %s  
unparsable value for LS_COLORS environment variable  
LS_COLORS: unrecognized prefix: %s  
p->minfo.mi_magic2 == 0x5555  
bcoalesce: CHAIN(mp2) != mp1  
malloc: %s:%d: assertion botched  
free: called with already freed block argument  
free: called with unallocated block argument  
free: underflow detected; mh_nbytes out of range  
free: start and end chunk sizes differ  
malloc: block on free list clobbered
```

## strings in practice

- `strings` isn't very useful on its own
  - No filtering, lots of garbage "text"
- So use `grep` (or another tool) to filter!

```
$ # search for "error" with 1 line of context
$ strings /bin/bash | grep -C1 "error"
$ # search for things that look like env vars
$ strings /bin/bash | grep -P "\S+=\S+"
```

## content-agnostic analysis: binwalk

- binwalk is a great tool for unpacking multiple files stuck together:
  - (Partial) disk image, embedded photos and videos in other formats, archives...

```
$ man binwalk
$ binwalk my_file.bin
$ binwalk -e my_file.bin
$ binwalk --dd="png:png"
```

## content-agnostic analysis: file

- Files don't necessarily need to have a filename extension
  - They don't even need to have a *correct* one
- `file` tells you what kind of file based on magic bytes
  - [https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html)

```
$ file flag.png
JPEG image data, JFIF standard 1.01 ...
```

## content-aware analysis

- If we know the format of the file, we can analyze it more precisely!
- Fingerprinting: figure out who generated it based on *how* they generated it
  - Binaries: every compiler has its own quirks
  - [Android Compiler Fingerprinting](#)
- PDFs/rich documents: embedded scripts, embedded changelogs, default field values, references to other files...

## analysis techniques: metadata

- Files have all kinds of **metadata** in them
  - Chain of authors, original computer, ...
  - Particularly: PDFs, DOCs, JPG/PNG/MP4, ...
- Images (JPG/PNG) can be especially interesting:
  - GPS tags, time taken tags
  - Device/vendor/software tags
    - Why might this be interesting?  
(think vuln scanning)

Learned the hard way



## metadata analysis

- `exiftool` is great for both viewing and modifying image/audio/video metadata:

```
$ sudo apt install exiftool
```

```
$ # dump all tags in the file
```

```
$ exiftool my_image.jpg
```

```
$ # wipe all tags from the file
```

```
$ exiftool -all= my_image.jpg
```



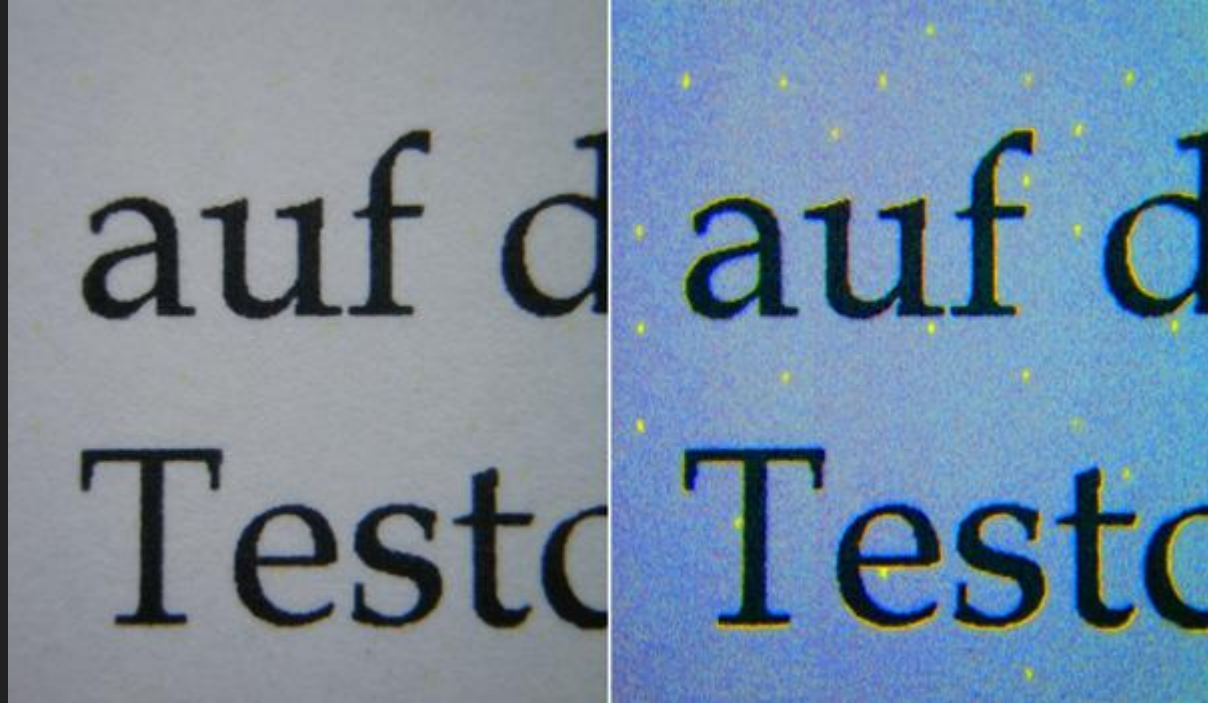
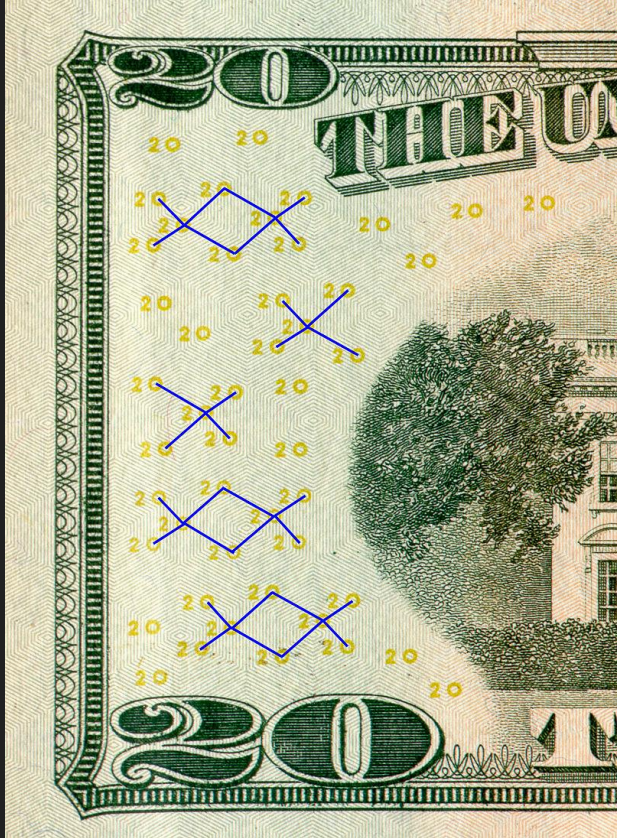
```
$ exiftool foo.jpg
```

```
File Type           : JPEG
File Type Extension : jpg
MIME Type           : image/jpeg
Exif Byte Order     : Big-endian (Motorola, MM)
Make                : OnePlus
Camera Model Name    : ONEPLUS A5000
Orientation         : Unknown (0)
X Resolution        : 72
Y Resolution        : 72
Resolution Unit     : inches
Software            : OnePlus5-user 7.1.1 NMF26X 327 release-keys
Modify Date         : 2017:11:28 13:34:38
Y Cb Cr Positioning : Centered
Exposure Time       : 1/33
F Number            : 1.7
Exposure Program    : Not Defined
ISO                 : 1600
Exif Version        : 0020
Date/Time Original  : 2017:11:28 13:34:38
Create Date         : 2017:11:28 13:34:38
Components Configuration : 1, Cb, Cr, -
Shutter Speed Value : 1/33
Aperture Value      : 1.7
Brightness Value    : -3.74
Metering Mode       : Center-weighted average
Flash               : Off, Did not fire
```

# analysis techniques: steganography

- Steganography is the practice of concealing information within other information.
  - Encoding text in the RGB values of an image, in the opcodes of an executable, ...
  - Often used for physical tracking/linking:
    - Printer dots (link paper documents to a printer)
    - Anti-counterfeiting (detect attempts to photocopy money)
    - Anti-piracy (detect attempts to duplicate a film)
    - User tracking (screenshot watermarking)
- *Not* encryption, but can be used to store encrypted information

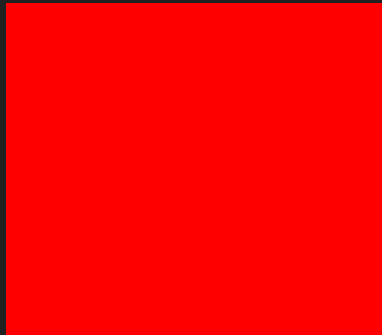
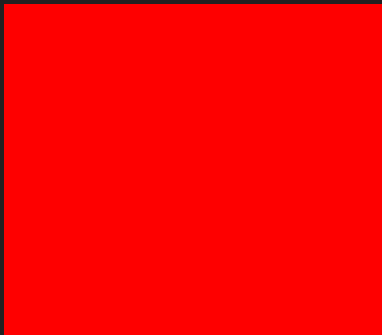
# analysis techniques: steganographic



Resources: [EFF Printer Tracking](#), [Coded Anti-Piracy](#), [EURion Constellation](#)

# analysis techniques: steganographic

- How does image steg work?
  - 24-bit RGB colorspace: 8 bits per red/green/blue
    - 1-bit changes to each color -> 3 bits/px ~> 3 px/byte
  - 32-bit RGBA colorspace: 8 bits per red/green/blue/alpha
    - 1-bit changes to each color -> 4 bits/px ~> 2 px/byte
  - Works because 1-bit changes to colors are (almost) imperceptible:



- Left: #FE0000
- Right: #FF0000

## steganographic techniques: steghide

- steghide can be used to hide data in the RGB/sound values of common image/audio formats
  - JPEG, BMP, WAV, AU
  - Data can be password-protected

```
$ sudo apt install steghide
$ # store data from stdin w/ pass 'hunter2'
$ steghide embed -cf foo.jpg -ef - -p hunter2
$ steghide extract -sf foo.jpg -xf out.txt
```

## homework #5

will be posted.

Let us know if you have any questions!

This assignment has 1 part.

It is due by 10/18 at 11:59PM.