Forensics

Slides stolen from @balex but it's ok because she didn't copyright them 😇

What is Forensics?

- Any type of problem that has to do with finding flags inside of files, memory dumps, images, and more
    - Essentially, just a static file (not source code)
- Can also involve ZIP cracking, file carving, steganography and more

What is a file, really?

- Bits; 1s and 0s
- Your computer will clump these all together in different nice ways so humans can understand it, but at the lowest level the computer is just looking at bits.

How are files stored?

- All of these bits get added together into a giant binary string: thousands, millions, or trillions of 1s and 0s in a row which represent the file's data.
- A bit can only have 2 options: 1 or 0, on or off
- This is a lot to work with; let's break it down into something more manageable

Bytes

- Each byte can be represented by a decimal (base 10) number
- Starting at the right and moving left, each position represents a power of 2
- A 1 or 0 will tell us if we want to add that power of 2 to our number

0	0	1	0	1	1	0	0	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	+ 0	+ 32	+ 0	+ 8	+ 4	+ 0	+ 0	= 44

- If we do a little math, we can see that each byte has 2^8 - 1 possible values
- This is still kind of long... could we shorten it?

Hex

- Hexadecimal
  - Base 16: Numbers 0-9 and A-F represent 0-15, usually prefixed by 0x
  - Since 16 = 2^4, we can only represent 4 bits of data with one hex character
- Since a byte is 8 bits, we need 2 hex characters to represent one byte

0	0	1	0	|	1	1	0	0	= 44
2^3	2^2	2^1	2^0	|	2^3	2^2	2^1	2^0	
0	+ 0	+ 2	+ 0	|	8	+ 4	+ 0	+ 0	
2	dec	=>	0x2	|	12	dec	=>	0xC	= 0x2C in hex

- Most of the time you will see files represented in hexadecimal, since it's more compact than binary but also easier to think about than Base 32 or 64

File Extensions

- Like I said earlier, files aren't really a thing to computers, it's just a chunk of bytes. Files exist to help humans understand what's going on.
- File extensions (`.txt`, `.jpg`, etc.) are mainly there for humans. All of the information about what type of file it is is stored inside the data.
- This means that if you change the file extension of a file, you may confuse a human and it may open with the wrong program, but you haven't changed what type of file it is
  - E.g. If I change `foo.txt` to `foo.png`, the contents inside are still a text file. You can even still open the "image" `foo.png` in a text editor!

File Extensions

- Whenever you get a file in a CTF, run `file` on it!

  ```
  $ file foo
  ```

  ```
  foo: JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96,
  segment length 16, Exif Standard: [TIFF image data, big-endian, direntries=5],
  baseline, precision 8, 600x320, components 3
  ```

  The response from the `file` command will always tell you what the file **actually** is

- It will also tell you some handy information (e.g. resolution, metadata) for files like JPEGs, PNGs, etc

Magic Headers

- Then how does your computer know what format a file is? Magic headers! (or magic bytes, file signatures, file header, etc.)
- Each file format (PNG, JPEG, MP3) starts (or ends) with a specific sequence of bytes. When you computer starts to read the file, it can then recognize whether it's a picture, audio file, text file, etc.
    - E.g. `FF D8 FF DB` are the first 4 bytes of a JPEG file
    - Fun note: since some magic bytes are at the end, and some are at the beginning, you can mash 2 files together and REALLY confuse your file explorer

Magic Headers

- You’ll usually get problems that are either missing magic headers, or have incorrect magic headers
- You can use a hex editor to edit the file and fix the headers
  - like Bless (you know....if you want to use the devil’s hex editor)
  - GHex (linux only)
- Wikipedia has a handy list of file headers that you can consult!

Strings

- A common easy problem is to hide the flag in the metadata of a file, or somewhere in the data of the file itself
- You can use `strings` to look for any strings in the file, and then pass it to `grep` to search for the flag format

```
$ strings foo | grep utflag
utflag{you_found_me}
```

- This can work on non-forensics problems too!
    - E.g. reversing problems where the flag is harcoded somewhere in the program

File Carving

- A common type of forensics CTF problem you'll see is called **file carving**
- Problems writers will hide files within other files, or concatenate multiple files together into one big file
- In order to see what's inside of a file you're given, use `binwalk`!

```
$ binwalk foo

DECIMAL    HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0          0x0             JPEG image data, JFIF standard 1.01
30         0x1E            TIFF image data, big-endian
66995      0x105B3         Zip archive data, at least v1.0 to extract
69256      0x10E88         End of Zip archive, footer length: 22
```

File Carving

- To extract these files, you can use these tools
  - Foremost
  - Scalpel

ZIP Cracking

- Problems where you're given a ZIP archive that is password protected
- Most of the time, the password is one from a top X most common password lists, so you can use tools like `fcrackzip`, John the Ripper or `hashcat` to automate figuring out the password

```
$ fcrackzip -v -D -u -p passwords.txt secret.zip
'secret/' is not encrypted, skipping
found file 'secret/flag.png', (size cp/uc   2185/  2940, flags 9, chk 5816)

PASSWORD FOUND!!!!: pw == qwerty123
```

- You can get good password lists from GitHub, or use rockyou.txt

Steganography

- Hiding information AND hiding the fact that you hid information
- For CTFs, this usually means hiding information in image files, but you’ll also see info hidden in audio files too (technically, you can hide any file in any type of file, but these are the most common)

Steganography

- Least Significant Bits
- Embed the message in the least x significant bits of the image
    - The more bits you use, the more obvious it is that you are hiding a message
- Images are typically encoded with 1 byte per color channel per pixel
    - To represent 1 pixel, you'll need 3 bytes: one for red, one for green and one for blue
    - If you can convert your message/file to binary, then change the last bit of each of these bytes to the values of your secret

Steganography

	red	green	blue
🟩	10001001	11001111	10001001
🟩	10001000	11001110	10001000
🟩	10001000	11001111	10001010

Steganography

original

$\longrightarrow$

steg

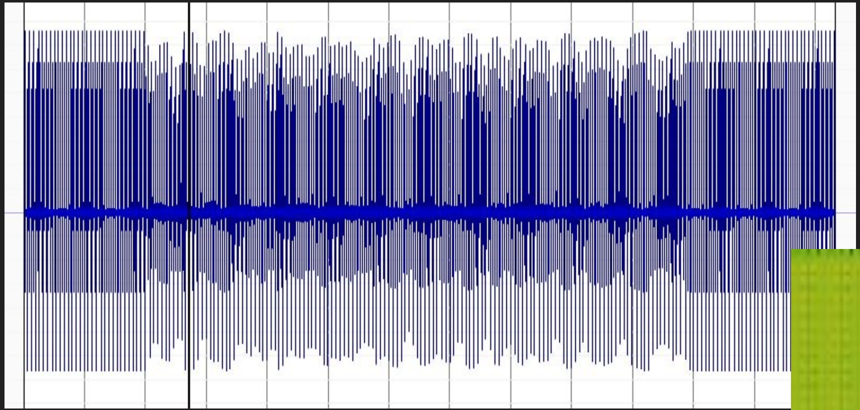
Steganography

- To recover the message, you can write your own program to recover the bits of the secret, or you can use some of these tools
  - Stegsolve
  - Steghide
  - zSteg
  - Online decoder

Steganography

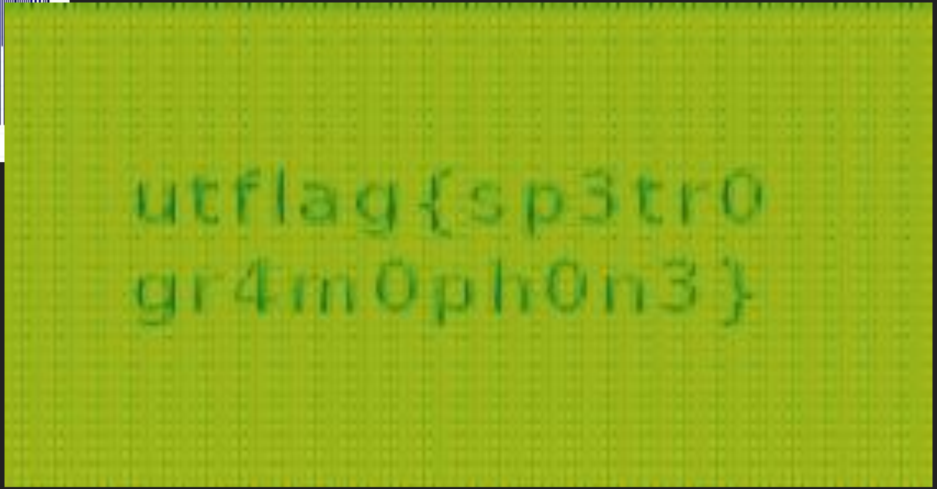
- Audio Steganography
- Hide text or images in audio files that can only be seen by looking at the spectrogram
  - A spectrogram is a visual representation of the spectrum of frequencies
- you can use a tool like Sonic Visualizer or Audacity (both work on win, *nix, mac) to view the spectrogram

Steganography



waveform (default)

pane > add spectrogram



spectrogram

Practice

- Connect to forever.isss.io to try your hand at some forensics problems!