

Cheatsheet - Steganography 101

Nov 30, 2015 • By [phosphore](#)

Category: **cheatsheet**

Tags:

Steganography 101

First things first, always use [binwalk](#) or [foremost](#) to isolate files from any other embedded stuff.

```
$ binwalk -e flag.png
```

#Useful options

-e, --extract	Automatically extract known file types
-B, --signature	Scan target file(s) for common file signatures
-E, --entropy	Calculate file entropy, use with
-B (see the quickstart guide - https://goo.gl/JPKAIQ)	
-z, --carve	Carve data from files, but don't execute extraction utilities
-r, --rm	Cleanup extracted / zero-size files after extraction
-M, --matryoshka	Recursively scan extracted files
-R, --raw="\x00\x01"	Search for a custom string.

The search string can include escaped octal and/or hex values.

#Binary Diffing Options

-W, --hexdump	Perform a hexdump / diff of a file or files
-G, --green	Only show lines containing bytes that are the same among all files
-i, --red	Only show lines containing bytes that are different among all files
-U, --blue	Only show lines containing bytes

that are different among some files

And of course use `strings` (ASCII, UTF8, UTF16) or `hexdump -C` on the file, before anything advanced. Remember that, by default, strings decode ASCII characters, but you can set it to gather Unicode strings or to handle other types of encoding such as 32-bit big/little endian (e.g. the `-el` option will have the strings command handle 16-bit little endian encoding). Read “[Strings, Strings, Are Wonderful Things](#)” from the SANS blog.

- Check plaintext sections, comments (`cat`, `strings`)
- Hex Editors are your best friend now. We suggest [hexedit](#) for the console or [Bless Hex Editor](#) if you like it with a GUI. Check for suspicious magic bytes, correct file length, and use `dd if=inputfile.png of=anotherfile.zip bs=1 skip=12345 count=6789` to extract concatenated files (“skip” will be the starting position, “count” the number of bytes from the “skip” position to extract)
- Use [exiftool](#) to extract EXIF data
- Use [TinEye](#) to upload and search for the image. Select “best match” and hopefully you get the original image. [XORing](#) should do the rest of the job. Also use `compare a.png b.png result.png` from the ImageMagick suite, plenty of params available here (e.g. `-compose src`).
- Another steganographic approach is to hide the information in the first rows of pixel of the image. See [this chal](#) for more details.
- Use [pngcheck](#) for PNGs to check for any corruption or anomalous sections
`pngcheck -v` PNGs can contain a variety of data ‘chunks’ that are optional (non-critical) as far as rendering is concerned.
 - bKGD gives the default background color. It is intended for use when there is no better choice available, such as in standalone image viewers (but not web browsers; see below for more details)
 - cHRM gives the chromaticity coordinates of the display primaries and white point
 - gAMA specifies gamma
 - hIST can store the histogram, or total amount of each color in the image
 - iCCP is an ICC color profile
 - iTXt contains UTF-8 text, compressed or not, with an optional language tag. iTXt chunk with the keyword
 - pHYS holds the intended pixel size and/or aspect ratio of the image
 - sBIT (significant bits) indicates the color-accuracy of the source data

- sPLT suggests a palette to use if the full range of colors is unavailable
- sRGB indicates that the standard sRGB color space is used
- sTER stereo-image indicator chunk for stereoscopic images
- tEXt can store text that can be represented in ISO/IEC 8859-1, with one name=value pair for each chunk
- tIME stores the time that the image was last changed
- tRNS contains transparency information. For indexed images, it stores alpha channel values for one or more palette entries. For truecolor and grayscale images, it stores a single pixel value that is to be regarded as fully transparent
- zTXt contains compressed text with the same limits as tEXt
- If the image is relatively small check the palette (use `convert input.png output.xpm`). Be aware that sometimes colors are not preserved. In this case use the extra parameter.
- If there are large portions of the image that look the same colour check with a Bucket Fill (in gimp also remember to set the threshold to 0 when filling) for anything hidden, or play with the curves. Use [Grain extract](#) to check for watermarks.
- If you see Adobe Suite/CC metadata with `strings`, be sure to open the image with the corresponding program in order to not lose layers informations. If some layer are overlapped, gimp or other image viewers usually will merge all the visible layers in once.
- If you happen to extract a file with binwalk, but this is not the flag, you should check with an hex editor for other data before/after the file.
- Look for some gzipped data (`1F 8B 08`), or possible file signature/magic bytes (google it!), and extract 'em with `dd`. Remember that if decompressing with `tar xvf` doesn't work (e.g. incorrect header check), you may try to decompress it chunk by chunk with [this script](#).
- If you need to plot raw binary data to an image (bitmap/png) with given width and height, you can easily use `convert` from ImageMagick.

```
$ convert -depth 8 -size 1571x74+0 gray:pretty_raw_cutted
prett_raw_out.png
#Useful options
-depth 8: each color has 8 bits
-size 2x3+0: 2x3 image. +0 means starting at offset 0 in the
file. If there are metadata headers, you can skip them with
the offset.
gray:f: the input file is f, and the format is gray, as
```

defined at <http://www.imagemagick.org/script/formats.php> This weird notation is used because ImageMagick usually determines the format from the extension, but here there is no extension.

you may have problem viewing the output with a standard image viewer, so be sure to use gimp or convert it again `convert out.png -scale 300x200 out2.png`. An [example of challenge](#) where this technique is useful.

- Use the [steganabara](#) tool and amplify the LSB of the image sequentially to check for anything hidden. Remember to zoom in and also look at the borders of the image. If similar colours get amplified radically different data may be hidden there.
- [Stegsolve](#) (a simple jar `java -jar stegosolve.jar`) is also pretty useful to extract data (based on bitplanes) and analyze images, allowing you to go through dozens of color filters to try to uncover hidden text.
- Outguess

```
$ ./outguess [options] [<input file> [<output file>]]
#Useful options
-[kK] <key>   key
-[eE]         use error correcting encoding
-r           retrieve message from data
-m           mark pixels that have been modified
```

- [OpenStego](#) is another GUI tool used for Random LSB.
- [StegHide](#), to extract embedded data from stg.jpg: `steghide extract -sf stg.jpg`.
- [StegSpy](#) will detect steganography and the program used to hide the message, checking for classical steganographical schemes.

Scripts

- (Python) Pixel color inverting example:

```
import Image
if __name__ == '__main__':
```

```
img = Image.open('input.png')
in_pixels = list(img.getdata())
out_pixels = list()

for i in range(len(in_pixels)):
    r = in_pixels[i][0]
    g = in_pixels[i][1]
    b = in_pixels[i][2]
    out_pixels.append( (255-r, 255-g, 255-b) )

out_img = Image.new(img.mode, img.size)
out_img.putdata(out_pixels)
out_img.save("output_inverted.png", "PNG")
```

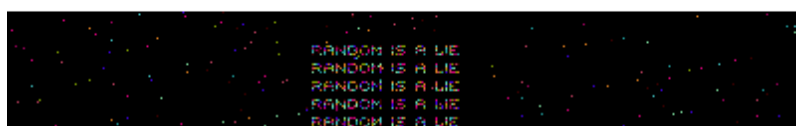
- (Python) Change the palette (or colormap) of a PNG: [link to the script - example of usage](#) // [Ruby version](#)
- (PHP) If the image looks like it's just a random noise we should make sure of it. We can, in fact, measure its randomness. Pixels of each color can appear in each place of the image with equal chance. If it's false for some colors, we certainly want to look at them. [Here](#) is a script for that, and the results appears below:

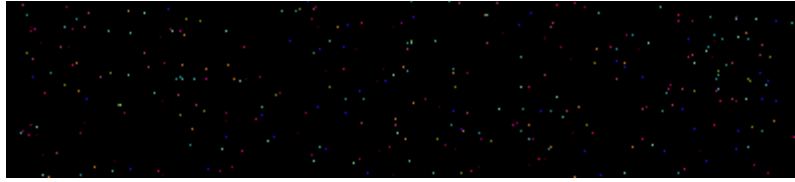
```
$ php solve.php image.png
MAX disp: 1492.41; AVG: 92.82
GAP: 351.61 ± 200
DONE.
```

Before



After





Audio Steganography

- Check the comments
- Load in any tool and check the frequency range and do a spectrum analysis.
- Use [sonic-visualiser](#) and look at the spectrogram for the entire file (both in log scale and linear scale) with a good color contrast scheme. See [this challenge](#) from the PoliCTF 2015 we solved with this method.
- A classic method for embedding data in an audio file is to hide it in the least significant bit of each sample. [See this article](#)

Video Steganography

- You can extract single raw frames with ffmpeg. See [here](#)
- Be sure to open the audio of the video with both Audacity and VLC. Also, for VLC there are multiple filters, [check em out](#).

0 Comments **P=NP CTF Team****1 Login** ▼ **Recommend** 4 **Tweet** **Share****Sort by Newest** ▼

LOG IN WITH


OR SIGN UP WITH DISQUS 

Be the first to comment.

ALSO ON P=NP CTF TEAM


Padding Oracle attack against Telegram Passport

3 comments • a year ago

 **Telegram Messenger** — This post describes an attack that won't work unless a developer ignores a security

Square CTF 2017 - Reading between the lines

4 comments • 2 years ago

 **TheZero** — You're welcome

PequalsNP -|- visit us on [github](#) - [twitter](#) - [ctftime.org](#) actually collaborating with [JBZ team](#) subscribe [via RSS](#)