

## 19:08 Steganography in .ICO Files

by Rodger Allen

These days, with a megapixel camera in all our phones, we are used to full colour, 24-bit images. The days of 256 colour images may seem to be something that only our older neighbours might remember. But these low-res images are still with us and so ubiquitous that they go unnoticed.

Minimize all the windows on your desktop and you'll likely see a dozen or more of them. Check the tabs in your browser and you'll see many more. Yep, a great deal of those icons and favicons are actually low resolution bitmaps.

And they're a great place to hide data!

### BMP Palettes

First, let's discuss how Palettized BMPs work. The basic structure of a bitmap file is a bit like so.

```
//14 Byte FileHeader.
2 typedef struct tagBITMAPFILEHEADER {
  WORD bfType;
4  DWORD bfSize;
  WORD bfReserved1;
6  WORD bfReserved2;
  DWORD bfOffBits;
8 } BITMAPFILEHEADER;

10 //5 different sizes, 20 to 124 bytes.
  struct DIBHeader;

12 //Optional, 8 to 1024 bytes.
14 struct Palette;

16 //Rows are null-padded, divisible by four.
  RGBQUAD pixels [];
```

Bitmap images that don't use a palette define the colour independently for each pixel. Each pixel uses three bytes (24 bits) to define the Red, Green and Blue (RGB) channels. The pixels in a palettized image reference the Palette to define the colour for each pixel. 256-colour bitmaps use 8-bit pixels, 16-colour bitmaps use 4-bit pixels, and 2-colour bitmaps use a single bit for each pixel.

The palette structure uses four bytes to define each RGB, with the fourth byte being reserved. The

<sup>52</sup>MSDN tagRGBQUAD Structure

For the delight and amusement of the Reverend Pastor Manul Laphraoig and his flock,

MSDN page on the RGBQUAD struct states that the fourth byte is "reserved and must be zero."<sup>52</sup>

The depth of colour in a palettized image is then still the same as a full 24-bit colour image - each pixel is still a full 24-bit colour. It's just that the palettized image is likely to contain fewer overall colours than the 24-bit-per-pixel image. Indeed, even the so-called monochrome 1-bit image isn't restricted to just black and white; the two colours can both be full 24-bit colours.

The choice as to whether to use a palettized image or just have 24-bit pixels mostly comes down to file size. For a small image, such as an icon (and we'll come back to these soon) you might find it better to use 24-bit pixels instead of allocating 1k for the palette. For example, a 16×16 image might use just 20-odd different colours. If it used a palette, then the file size would be (roughly) 1.25k (1024 bytes for the palette and then 256 bytes (16×16) for the pixels), with roughly 900 bytes of palette unreferenced and unused. Using 24-bit pixels would yield a file size of approx .75k (0 bytes for the palette and 768 bytes (16×16×3) for the pixels). The figures for a 32×32 pixel image would be 2,048 bytes for the palettized image and 3,072 bytes for the 24-bit version.

### Palette Histograms

The key element of this steganographic technique is to take a histogram of the palette colours that are used in the pixels. It is often the case that not every colour defined in the palette is actually used by the pixels. The histogram makes a count of the number of times each colour is used. We are interested in the colours that have a count of zero, since we can then overwrite those colours (bytes) in the palette array, and it won't affect the display of the image.

To extract the data utilises the same process - take a histogram of the pixels per palette colour, and read those bytes out.

This technique has three important advantages over the LSB (Least Significant Bit) method:

First, there is no need to have a reference image. The LSB method makes comparison between the original image and the injected image to determine which bits have been altered. With this technique, the original pixel array is the key to which bytes are to be read from the palette.

Second, and depending on the image size, there is the potential to store quite a bit more data into the image. The LSB method generally only uses one bit per colour channel, so even with 24-bit images it can only store three bits per pixel. This method though has an upper-limit on the amount of data that can be stored per image - an 8-bit palettized image that only uses two colours leaves 254 free colours, therefore leaving 762 bytes to inject into. The size of the image itself doesn't change this.

Finally, there is an element of deniability in the histogram method. Steganography is framed as a game between two prisoners, Alice and Bob, who wish to privately communicate in the presence of a warden, Mallory, who can read all of their messages. Even if Mallory does notice that the palette is weird, Alice or Bob could quite plausibly say, "Hey, that's just the palette that the image creation software made." Of course, Alice and Bob could only use their image once without drawing attention to them.

You might remember from earlier that each palette entry uses four bytes. I quite deliberately only use the three RGB bytes to inject and leave the reserved bytes alone, mostly on the grounds of detectability.



<sup>53</sup>man 1 convert

## Detectability

Despite the claim to deniability, there are some obvious markers of the injection. For starters, take a look at the examples of a palette from an image processed by MS Paint, which is for the most part the old web-safe palette, or the palette generated by Image Magick's `convert` utility,<sup>53</sup> which is front-loaded with the actual colours in the image, and then the rest is solid black (0x000000). Yet another palette that was converted from 24-bit to 256 colours by Image Magick does display quite a spread of colours:

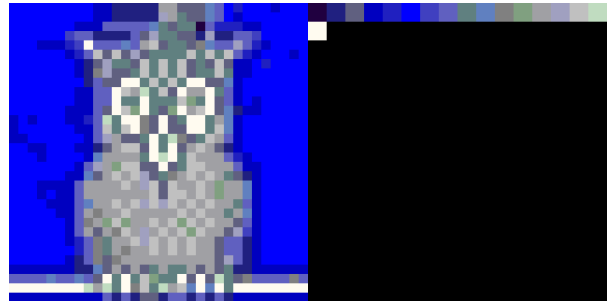
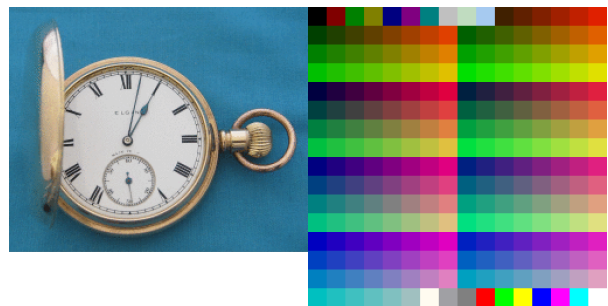


Image Magick Short Palette



Microsoft Web-Safe Palette

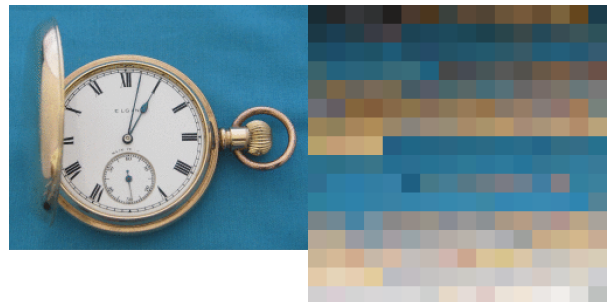


Image Magick Full Palette

Then compare these to the palette from an injected image. It is obvious that the colours have been all jumbled up.

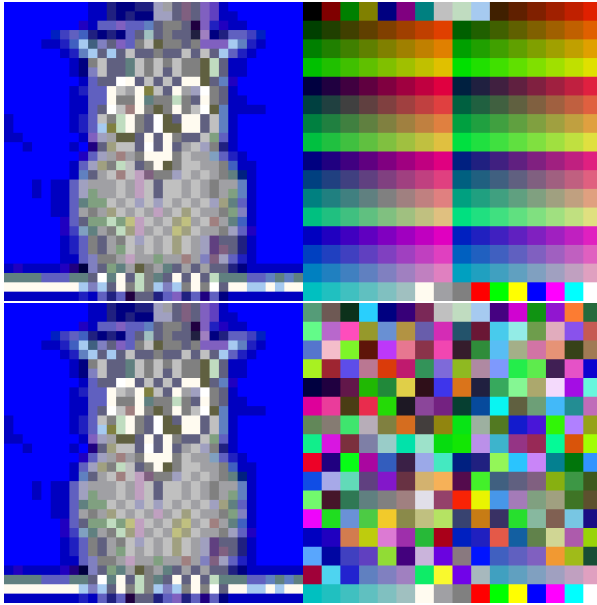


Image Before and After Injection

## Icons

But who uses those palettized bitmaps any more? The camera in your phone, heck, even the display on your phone, is capable of taking and displaying images with a bewildering depth of colour. And nowadays, bandwidth is cheap and fast, and image compression algorithms are good enough, that there is little reason to lower the quality of the images.

There are two places, however, where these images are, if not ubiquitous, at least quite widespread. Take a moment, and minimize all the windows on your desktop. Most of those icons will be using bitmaps. Now open a browser and navigate to some random page. That little icon in the browser location bar or in the tab is also most likely a bitmap, and is known as a favicon. Not every website has them, but almost every browser will request them.

The Icon file format is basically a little directory of multiple images. The format for an Icon header follows this general schema:

```

1 typedef struct {
2     WORD idReserved; //Always zero.
3     WORD idType;     //Often 0x0100.
4     WORD idCount;    //Count of dire entries.
5 } ICONHEADER;

```

It is followed by one or more 16-byte directory entries.


```

1 typedef struct {
2     BYTE bWidth;
3     BYTE bHeight;
4     BYTE bColorCount;
5     BYTE bReserved;
6     WORD wPlanes;
7     WORD wBitCount;
8     DWORD dwBytesInRes;
9     DWORD dwOffset;
10 } ICONDIRENTRY

```

The rest of the file is nominally contiguous blocks of images. The standards suggest that there are only two types of valid images: BMP and PNG. The BMP image blocks are basically the same as for BMP files, but don't use the first 14 bytes of the FileHeader. That is, they use the DIB Header, optionally the Palette, and of course the Pixels.

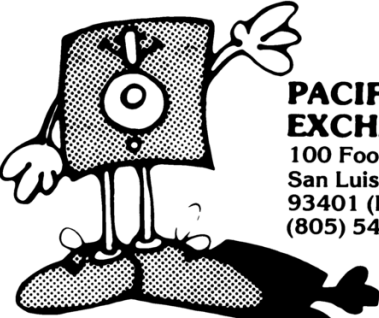
The DIB pixels in an icon have one other complication. The pixel array is in fact two separate arrays. The first is the actual coloured pixel



# Dysan

**CORPORATION**

Solve your disk problems, buy 100% surface tested Dysan diskettes. All orders shipped from stock, within 24 hours. Call toll FREE (800) 235-4137 for prices and information. Visa and Master Card accepted. All orders sent postage paid.



**PACIFIC EXCHANGES**  
 100 Foothill Blvd.  
 San Luis Obispo, CA  
 93401 (In Cal. call  
 (805) 543-1037)

✓ 274

array. The second is literally an array of bits that act as a mask that is used to determine the transparency of the icon.

One major difference between the Icon format and the DIB format (the actual image format contained in the BMP), is that the Icon header information is little-endian, and the DIB format is big-endian. So the resultant file is a mix of both big and little endians.

Consider that `idCount` field. An icon file can contain up to 65,536 image resources. That's up to 48Mb worth of injectable palette space!



Injected Icon and its Palettes

### Example of an Icon header

		— ico header
2	00 00	idReserved
	01 00	idType
4	02 00	idCount
6		— resource header 1
	10	bWidth
	10	bHeight
	00	bColorCount (0 if >=8bpp)
10	00	bReserved (must be 0)
	01 00	wPlanes
12	08 00	wBitCount
	68 05 00 00	dwBytesInRes
14	26 00 00 00	dwOffset
16		— resource header 2
	etc	
18		— resource data 1
20	etc	Starts at 0x00000026, containing 0x0568 bytes.
22		Consists of:
24		* DIBHeader
26		* Palette (maybe)
28		* Pixels
		* Transparency mask
		— resource data 2
30	etc	

## Uses in the Past and Future

Taking a look at the favicons used by the top thousand sites from the Alexa list. Just under seven hundred of the sites responded with an image file. Of these, 560 were icon resource files, that is, the type of icon files I've described above. The others were in general just PNGs or other image types simply renamed with the `.ico` extension.

Of these icon resources, at least 1-in-7 contained an 8-bit BMP image, suitable for palette injection. Around three quarters of these files contained only one or two images, but there were four favicons that contained ten or more bitmaps.

Given how widespread these favicons are and their variety, and the fact that they are effectively ignored by most web security monitoring systems, they would be an excellent mechanism for at least part of a C2 (Command and Control) channel for malware. Indeed, there is some history with the Vawtrak malware using LSB steganography to communicate updates from their C2 servers.<sup>54</sup> Other malware rootkits have just renamed their malware to `favicon.ico`, but are in reality just raw (or obfuscated) PHP code or the like.

As for prior art, I haven't been able to discover any other previous uses of this technique of repurposing the unused bytes in an image palette. If any brethren know of similar techniques, I'd love to hear about it.

Bitmaps aren't the only image type that use a palette. PNGs, for instance, have a PLTE chunk that describes the colours in the image. But the PNG format removes the dead colours and the PLTE chunk only contains a list of the actual used colours, thereby reducing the size. The PNG standard does however allow the PLTE chunk to contain more colours than are actually used. This histogram technique would then reduce to adding extra bytes to the image file, a method I was trying to avoid.

On the subject of adding extra bytes, notice that both BMPs and Icons are what I call indexed file formats; that is, the header contains information about the offset (where the image data starts) and size (how big the image data is). This makes it possible to introduce arbitrary data into the files and then manipulate the offsets to skip over the padded data.

You can also, of course, just tack on the extra data at the end of the file, and it should be ignored by the image viewer.

The default image viewers (e.g. `shotwell`) on the version of Linux I am currently using doesn't like the padding before the pixels, rendering the image with those padded bytes; maybe one of our memory-bug hunting friends could find some delight here. Gimp is okay though. Windows seems to behave correctly and ignores the extra bytes.

## Where's the code?

The POC code is a tool called `Stegpal`, written in Haskell. If the source is not yet available from Hackage, you'll find it attached to this PDF and as the Favicon for the most popular PoC||GTF0 mirror.<sup>55</sup>

## Creating icons

I used Image Magick to create sample icons. I wasn't too worried about the transparency bits, as they don't change anything about the palette.

Start with an image that is going to bear being reduced down to a small size. The number of colours doesn't matter too much as this process will reduce that anyway. It's best if the original image has equal dimensions for width and height.

Create a bunch of smaller scaled images from the original. Favicons are usually 16x16 (ish), but you can create them any size you want.

Then feed all of the smaller BMPs into one `ico`.

```
# Creating icons
2 convert source.bmp -scale 64x64 \
4   -type Palette -depth 8 -compress none \
   temp-64x64.bmp
6 convert source.bmp -scale 32x32 \
   -type Palette -depth 8 -compress none \
8   temp-32x32.bmp
convert source.bmp -scale 16x16 \
10  -type Palette -depth 8 -compress none \
   temp-16x16.bmp
12 convert temp-64x64.bmp temp-32x32.bmp \
   temp-16x16.bmp favicon.ico
```

<sup>54</sup>[unzip pocorgtfo19.pdf avgvawtrak.pdf](#)

<sup>55</sup>[unzip pocorgtfo19.pdf stegpal-0.2.8.0.tar.gz; wget https://www.alchemist.org/favicon.ico](#)