

Carrier Groups for Steganography

The term 'Carrier Group' was created, by myself, to refer to the following concept of Steganography. This is not intended to be an overly technical paper, but should convey the concepts correctly. Questions about this paper can be directed to vertigosecuritytribecom.com.

There are concepts within the realm of Covert Channels, specifically in the area of Steganography which have become outdated. Although the basic premise of hiding data has remained the same for millennia, the evolution of these processes is likely to take place very soon. Because the detection of steganography is still trailing behind the actual creation of stego, this evolution will likely widen the gap between the two making it more difficult to detect stego in carrier files.

Let's take the stego process called Least Significant Bit (LSB) modification and use it as an example. The process I'll define within this paper will work most efficiently within the context of LSB stego.

When LSB steganography is created by an application, data is hidden within the value of the color values defined by bits. If we're dealing with an 8 bit image, there are 8 bits for each pixel that define the color value. If we're dealing with a 24 bit image, there are 3 sets of 8 bits that define the color for each pixel.

Let's focus on 8 bit images to simplify the concept. Let's also number the bits from 1-8, from left to right. Bit 1 is the most significant bit in our color code. Changing this bit will result in the most dramatic change in color for the pixel. The bit on the farthest right is the least significant bit and will result in the least color change if we alter its value. As we walk from bit 1 to bit 8, the bits become increasingly less significant and make less of a color shade change, with the last bit (bit 8) being the LSB.

If we intend on hiding data within an image, we want to have the least noticeable effect on the image, since in most cases the individuals that will likely see this image will not have the original file to compare with anyway. Our 8 bits each store a value of a 0 or a 1. When hiding binary data, we simply ensure that data is first stored in the LSB. The application will move through the file being hidden and change the LSB in our carrier file to reflect the placement of 0 (zeros) or 1 (ones). When all of the LSB bits have been used, if we still have data left to store, we move on to the next least significant bit; which is one step to the left.

Using this technique, you can see how it might be very difficult to detect small files within large carrier files. But if the file we want to hide is large enough, it starts to distort the image colors because we're altering the more significant bits in each pixel. This is where the evolution steps up to the challenge.

Detection algorithms cannot reliably detect stego within carrier files when the amount of data is very small. So let's assume we now have a file that we want to hide that might make detection easier, due to its size. What can we do to make it more difficult?

We'll use the term "target file" (also commonly referred to as a payload) to denote the data that we want to hide. We'll use the term "carrier file(s)" to denote the files within which we hide our data. These terms will be important to understanding the following concepts.

As an example of normal steganographic technique, let's take a carrier file of 400k. This is an image file. The basic rule of thumb is that our target file should be less than 30% of the carrier file size, but if we want detection to be more difficult so we should probably stay around the 20% - 25% range instead. Using this amended rule, we're looking at a target file of about 100k.

Now, let's evolve our thinking. We already know that the smaller the target file is, in comparison to the carrier file, the more difficult detection becomes. So let's take our target file and break it into 5 distinct pieces, each one of which is now 20k in size. We also want to use 5 carrier files now, instead of one. Our new application will take the 100k target file and begin with the first carrier image. 20k worth of data will be hidden in the first image and then the application will move on to the next carrier image. 20k will also be hidden in this carrier and the process will continue on until all 100k has been hidden using all 5 carrier files.

Let's now look at our target:carrier ratio. We're now looking at a 5% ratio of data held in each carrier file. Detection of this much data hidden within a potential carrier will be extremely difficult. Let's examine why this is. We already know that we're limiting our data to hiding within ONLY the LSB. The LSB within the carrier files will already be a 1 or a 0. The chances of the bit already being what we need it to be is 50%, on average. So over the life of steganography, we will likely only need to change 50% of the LSB values to reflect the data we're hiding. Using VERY rough math here we assume that, on average, only 5% of the actual file size of our carrier files will need to be changed using the example above.

This is rough math and the concept probably needs some work. But this is only one example of where data hiding is likely to expand past our ability to detect. Now, let's consider a simple carrier file naming scheme that can be utilized by an entity to identify those files that belong within our "Carrier Group".

I'm sure that many of you have seen web pages that are automatically created and contain hundreds of images taken by individuals. They can be images of a night out on the town, a birthday party, a concert, etc. It doesn't matter. The applications all work the same. They create a thumbnail of each image and place that thumbnail within the context of an html page as a link to the real image.

So let's create an image web page like this, but we want our 5 carrier files hidden in there as well. This requires that we have an identifiable means for someone looking at our images to pick out those particular 5 files as the correct ones. This is only an example off the top of my head. I'm sure there are some really smart folks that can come up with

something better in a short period of time. I'm just recording these ideas for posterity.

Digital cameras name images sequentially. For instance, you may find that your camera stores images using a file name format of DSC00000.jpg or P0000000.jpg. The zeros in these names are incremented sequentially as pictures are taken with the camera. So when we look at our web site, the file names will likely be in the same number range. How do we differentiate our carrier files from the others?

Let's assume that the images we're posting are DSC00001 - DSC00099. A sample naming scheme might continue on with that numbering but alter the name slightly to reflect the images in our carrier group. So we add 5 new images to the pool before creating the web page. The carrier images are named:

DSC01100.jpg, DSC02101.jpg, DSC03102, DSC04103.jpg, DSC05104.jpg

From the names, you can see that we've continued on with the numbering but added an extra number to denote which images belong to the carrier group. Since most of these web pages can contain hundreds of images at a time, these 5 images will not stand out.

So we take our 100k of data and hide it within a carrier group of five images. Then we post these images along with many other images of a social event on a web page, where they will remain hidden. Only those "in the know" will understand what they're looking for.

This same concept should work with audio files. The coolest thing about using audio files is that now we can hide much larger files within our carrier group. Large files mean more information. The more information we can hide without detection, the more significant and useful the stego has been. This is what makes the concept of a Carrier Group so interesting.

This paper was written primarily for myself as part of my own research. It is not intended to be a comprehensive document at this time.

Russ Rogers and SecurityTribe.com