

Hiding Secrets with Steganography

Dru Lavigne

I've always been fascinated by algorithms and the whole concept that a bit of mathematics can be used to compress an image or a sound or a video. Or keep track of where it put the files on my hard drive. Or perhaps scramble the contents of one of those files so that only my intended recipient is capable of descrambling it. Granted, I don't even pretend to understand the mathematics behind algorithms. But I'm somewhat comforted that there are people in this world who do, and that their efforts help to keep the computers of this world computing.

In this article, I'll introduce the science of steganography by demonstrating two applications from the ports collection. Along the way, we'll also discover some interesting features of compression algorithms.

What is Steganography, Anyway?

The term steganography comes from the Greek words for covered writing. If, as a child, you ever wrote an invisible message in lemon juice and had your friend hold it next to a light bulb in order to watch the message magically appear, you've used steganography.

When using steganography on a computer, you actually hide a message within another file. That resulting file is called a "stego file." The trick to computer steganography is to choose a file capable of hiding a message. A picture, audio, or video file is ideal for several reasons:

- These types of files are already compressed by an algorithm. For example, .jpeg, .mp3, .mp4, and .wav formats are all examples of compression algorithms.
- These files tend to be large, making it easier to find spots capable of hiding some text.
- These files make excellent distractors. That is, few people expect a text message to be hidden within a picture or an audio clip. If the steganographic utility does its job well, a user shouldn't notice a difference in the quality of the image or sound, even though some of the bits have been changed in order to make room for the hidden message.

If you're new to steganography, you're in for some interesting reading at the Steganography and Digital Watermarking web site.

Before we build the ports, you should be aware that steganography is also capable of encrypting a message before it is hidden in a file. Depending upon your geographic location, you may be limited by legal restrictions regarding the strength of encryption protocols, or even if you're allowed to use encryption in the first place. You'll see an example of this if you go to the outguess web site.

outguess

The outguess port builds several utilities that I'll demonstrate. Not all of them deal directly with steganography, as some are used to manipulate .jpeg images. You can find more information about these utilities at JPEGclub.org.

Let's start by building the port:

```
% cd /usr/ports/security/outguess
% make install clean
```

There's a fair bit of documentation on outguess and its related utilities. The port will install /usr/local/share/doc/README, as well as manpages for the following utilities: outguess, cjpeg, djpeg, jpegtran, rdjpgcom, and wrjpgcom. Finally, there is the outguess project home page.

Hiding Secrets with Steganography

Dru Lavigne

Since the outguess suite of applications deals primarily with .jpeg images, you may find the JPEG FAQ and the Compression FAQ helpful in getting up to speed.

I'll start with the rdjpgcom and wrjpgcom utilities. Did you know that the JPEG standard allows COM, or comment, blocks to be inserted into a .jpeg image? Being the curious type, when I first learned this, I was dying to know if the .jpegs on my hard drive had any interesting hidden comments. Fortunately, I had the rdjpgcom tool, so I could ReaD my JPG COMments:

```
$ cd ~/images
$ rdjpgcom pic1.jpg
$ rdjpgcom pic2.jpg
```

I was sorta disappointed to learn that most of my .jpegs had no messages at all. One indicated that it had been created using "VT-Compress (tm) Xing Technology Corp." and another indicated it had been "Created with The GIMP."

Fortunately, I could change this situation by using the wrjpgcom utility to WRite in my own JPG COMments:

```
$ wrjpgcom -comment "This picture was taken on my June 2003 canoeing trip"
pic1.jpg > pic1a.jpg
```

Make sure that you give the newly commented file a different name, or you'll end up with an empty original file.

Now, if I check out the results:

```
$ rdjpgcom pic1a.jpg
Created with The GIMP
```

This picture was taken on my June 2003 canoeing trip.

It's interesting to note that if I use the file command, it will pick up the original comment inserted by the GIMP, but not the comment I added myself.

If I had instead wanted to delete the previous GIMP comment, I would have used the -replace switch instead of the -comment switch.

If I visually view both files, say with gimp, I won't recognize any differences between the two. Let's see if there are any differences:

```
$ ls -l pic1*
-rw-r--r-- 1 dlavigne6 wheel 6817 Nov 15 14:36 pic1.jpg
-rw-r--r-- 1 dlavigne6 wheel 6873 Nov 15 14:36 pic1a.jpg
```

Okay, the file with the comments is a little bit bigger than the original file. However, the file utility doesn't indicate any difference:

```
$ file pic1*
pic1.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI),
"Created with The GIMP", 72 x 72
```

Hiding Secrets with Steganography

Dru Lavigne

```
pic1a.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI),  
"Created with The GIMP", 72 x 72
```

Hiding More Data

Let's carry this idea a bit further and hide a complete text file within a .jpeg file. For example, I may want to protect my great grandmother's chocolate chip cookie recipe. Right now, it's stored in cookie.txt:

```
$ ls -l cookie.txt  
-rw-r--r--  1 dlavigne6  wheel   296 Nov 15 14:56 cookie.txt
```

I also have a picture of my grandmother, who entrusted that recipe to me when I was much younger:

```
$ ls -l gramma.jpg  
-rw-r--r--  1 dlavigne6  wheel  50873 Sep  5 09:13 gramma.jpg
```

Let's see what happens if I hide the recipe in that picture:

```
$ outguess -k "don't worry, the recipe is safe" -d cookie.txt  
  gramma.jpg grandma.jpg
```

Let's take a look at that syntax. The -k or key switch is followed by a passphrase enclosed within double quotes. I need to remember that passphrase, in case I ever want to extract that secret recipe. I then used the -d switch to specify the name of the file to hide (cookie.txt), followed by the name of the file to hide it in (gramma.jpg) and the name of the new stego file (grandma.jpg). Once I had entered that command, I saw the following output:

```
Reading gramma.jpg....  
JPEG compression quality set to 75  
Extracting usable bits:  55365 bits  
Correctable message size: 25855 bits, 46.70%  
Encoded 'cookie.txt': 2368 bits, 296 bytes  
Finding best embedding...  
  0:  1219(50.8%)[51.5%], bias  1301(1.07), saved:   -4, total:  2.20%  
  1:  1215(50.6%)[51.3%], bias  1235(1.02), saved:   -3, total:  2.19%  
  5:  1192(49.7%)[50.3%], bias  1241(1.04), saved:   -1, total:  2.15%  
  7:  1164(48.5%)[49.2%], bias  1217(1.05), saved:    2, total:  2.10%  
 13:  1155(48.1%)[48.8%], bias  1176(1.02), saved:    3, total:  2.09%  
 25:  1163(48.5%)[49.1%], bias  1156(0.99), saved:    2, total:  2.10%  
 28:  1141(47.5%)[48.2%], bias  1145(1.00), saved:    5, total:  2.06%  
28, 2286: Embedding data: 2368 in 55365  
Bits embedded: 2400, changed: 1141(47.5%)[48.2%], bias: 1145, tot: 55200, skip:  
52800  
Foiling statistics: corrections: 499, failed: 0, offset: 46.129114 +- 142.525859  
Total bits changed: 2286 (change 1141 + bias 1145)  
Storing bitmap into data...  
Writing grandma.jpg....
```

If I now open both the original and new .jpeg files and examine them side by side, I'm hard pressed to see any differences between the two. This is to be expected, as the file to hide was very small (296 bytes) compared to the image file (50873 bytes). Interestingly, the new image file is slightly smaller than the original:

```
$ ls -l grandma.jpg
```

Hiding Secrets with Steganography

Dru Lavigne

```
-rw-r--r--  1 dlavigne6  wheel   50415 Nov 15 15:04 grandma.jpg  
Retrieving The Hidden File
```

To retrieve the hidden file, I need use the -r switch:

```
$ outguess -k "don't worry, the recipe is safe" -r grandma.jpg test.txt  
Reading grandma.jpg....  
Extracting usable bits:  55365 bits  
Steg retrieve: seed: 28, len: 296
```

I had to use the same key or passphrase I used to hide the message. If I read the resulting test.txt file, I'll see that the cookie recipe is still intact.

The outguess utility is capable of hiding messages in .jpeg, .ppm, and .pnm files. If you currently have a .bmp file that you'd like to hide a file in, use the cjpeg, or convert jpeg, utility:

```
cjpeg santa.bmp > test.jpeg
```

To my untrained eye, both files look the same in gimp. I can now use that new .jpeg file with the outguess utility.

Not surprisingly, djpeg converts the other way around; that is, from a .jpeg to the specified format:

```
$ djpeg -bmp frosty.jpeg > icicle.bmp  
$ djpeg -gif frosty.jpeg > icicle.gif
```

Both of these utilities have several switches to control the quality of the images. See their respective manpages for details.

The final utility in the outguess suite is jpegtran which can transform a .jpeg from, say, landscape to portrait. For example, the -flip horizontal switch will create a mirror image. That is, whatever is on the left will now be on the right:

```
$ jpegtran -flip horizontal family.jpeg > reverse.jpeg
```

The manpage contains other switches to flip and rotate .jpeg images.

steghide

Let's move on to the second port, the steghide utility:

```
% cd /usr/ports/security/steghide  
% make install clean
```

This utility will install a man steghide as well as some informative information to /usr/local/share/doc/steghide/README.

I liked outguess because of the extra .jpeg manipulation utilities that came with it. I liked steghide as its syntax is a bit more sensible, it supports more file formats (.jpeg, .bmp, .wav, and .au), and it allows you to specify an encryption algorithm.

Hiding The Cookie Recipe Again

Let's see what happens if I imbed that cookie recipe into a .wav file:

Hiding Secrets with Steganography

Dru Lavigne

```
$ steghide embed -cf hohoho.wav -ef cookie.txt -sf new.wav
Enter passphrase:
Re-Enter passphrase:
embedding "cookie.txt" in "hohoho.wav"... done%
writing stego file "new.wav"... done
Those switches make a lot of sense if you remember the three types of files you're
using:
-cf coverfile, or the file you want to cover/hide
-ef embedded file
-sf stegofile
```

If I listen to both the embedded file and the stegofile in xmms, I can't tell a difference in the audio quality, which, granted, I've never found that great for .wav files anyway.

Extracting the Recipe Again

When I wish to extract my cookie recipe, I'll extract from the stego file like so:

```
$ steghide extract -sf new.wav
Enter passphrase:
wrote extracted data to "cookie.txt".
```

Or like this:

```
$ steghide extract -sf new.wav -xf secret.txt
Enter passphrase:
wrote extracted data to "secret.txt".
```

The first invocation will extract the recipe into the same file name as the original cover file. The second invocation allows me to specify the name of the newly extracted file.

Miscellaneous steghide Extras

The steghide info command is quite useful. It will tell me if a file contains hidden data (however, only from steghide-created files, as far as I know):

```
$ steghide info new.wav
"new.wav":
  format: wave audio, PCM encoding
  capacity: 1.9 KB
```

```
Try to get information about embedded data ? (y/n) y
Enter passphrase:
  embedded file "cookie.txt":
    size: 296.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes
```

Notice that the default encryption algorithm is Rijndael, also called AES, at 128 bits. To see what other encryption algorithms are available:

```
$ steghide encinfo
encryption algorithms:
<algorithm>: <supported modes>...
cast-128: cbc cfb ctr ecb ncfb nofb ofb
```

Hiding Secrets with Steganography

Dru Lavigne

```
gost: cbc cfb ctr ecb ncfb nofb ofb
rijndael-128: cbc cfb ctr ecb ncfb nofb ofb
twofish: cbc cfb ctr ecb ncfb nofb ofb
arcfour: stream
cast-256: cbc cfb ctr ecb ncfb nofb ofb
loki97: cbc cfb ctr ecb ncfb nofb ofb
rijndael-192: cbc cfb ctr ecb ncfb nofb ofb
saferplus: cbc cfb ctr ecb ncfb nofb ofb
wake: stream
des: cbc cfb ctr ecb ncfb nofb ofb
rijndael-256: cbc cfb ctr ecb ncfb nofb ofb
serpent: cbc cfb ctr ecb ncfb nofb ofb
xtea: cbc cfb ctr ecb ncfb nofb ofb
blowfish: cbc cfb ctr ecb ncfb nofb ofb
enigma: stream
rc2: cbc cfb ctr ecb ncfb nofb ofb
tripleDES: cbc cfb ctr ecb ncfb nofb ofb
```

Wow, that's a lot of supported algorithms. To use a different algorithm, simply include the `-e` or encryption switch at the end of your embed command. In this example, I'll choose "blowfish":

```
$ steghide embed -cf hohoho.wav -ef cookie.txt -sf new.wav -e blowfish
```

Once the stego file is created, I'll double-check that it worked:

```
$ steghide info new.wav
"new.wav":
  format: wave audio, PCM encoding
  capacity: 1.9 KB
Try to get information about embedded data ? (y/n) y
Enter passphrase:
  embedded file "cookie.txt":
    size: 296.0 Byte
    encrypted: blowfish, cbc
    compressed: yes
```

Conclusion

This should get you started on using steganography utilities. The only question you may be asking yourself is "why use such a utility?" Probably the most common use is to safeguard passwords. We all know that we should use different passwords for various tasks. For example, you should use a different password to log into your computer, another to retrieve email, another for online banking, and yet another for when you create an account on a web server. It can be very handy to make a text file of each password and its usage, and to safeguard that file by hiding it in a place no one would suspect to look.

Until now, had you ever thought of looking in a picture or a sound file?