Detecting Steganographic Content in Images Found on the Internet

Jeremy Callinan Donald Kemick
Faculty Sponsor: Don Lewicki

Department of Business Management
University of Pittsburgh at Bradford

ABSTRACT

Steganographic techniques are being applied across a broad set of different digital technologies. It has been rumored that terrorists have been utilizing steganographic techniques within digital images to communicate on the Internet. There have been no confirmed reports of successful detection of steganographic content within images found on the Internet. In this paper, the authors describe their efforts in performing their own analysis. Over 200,000 images were collected and over 20,000 files were analyzed in-depth.

INTRODUCTION

Steganography is the art and science of hiding information by embedding messages within other, seemingly harmless messages. However, steganography is not cryptography. Cryptography involves transforming an original message so that any individual that happens to find the transformed message will not be able to understand it without knowing the correct method that will reverse the transformation, usually through some contact/agreement with the original encryptor. [9] Steganography is used to hide the very existence of the message. [2] Unlike encryption, steganography cannot be detected. Therefore, it is used when encryption is not permitted. Or, more commonly, steganography is used to supplement encryption. An encrypted file may still hide information using steganography, so even if the encrypted file is deciphered, the hidden message is not seen.

Steganography (literally meaning covered writing) dates back to ancient Greece. Common ancient Steganographic practices consisted of etching messages in wooden tablets and covering them with wax. Another method involved tattooing a shaved messenger's head, letting his hair grow back, then shaving it again when he arrived at his contact point. [2]

Computer steganography works by replacing bits of unused data areas in regular computer files, such as graphics, sound, text, HTML, or even floppy disks, with bits of useful, invisible information. Examples of this hidden information can be plain text, cipher text, or images. [9] Many Steganographic methods hide content in the “least significant” areas or the “noise” of carrier messages. Software algorithms have been developed to defect and defeat these steganographic methods.. [7]

Why do we need to hide information? There are two major issues that drive the technology to hide information. In the first group are those who are trying to protect their intellectual property rights. With the high availability of information via the Internet it is becoming more difficult to protect intellectual property and enforce copyright laws. Digital watermarks provide a way to insert a copyright notice into a document or image. The watermark is often a small image or a block of text that is repeated frequently through out the document or image. A similar technique is to embed a digital fingerprint or serial number. The advantage of a fingerprint is that it can be used to trace the copy back to the original and is a powerful tool for prosecuting copyright violators. [5]

The second group of people who are interested in hiding information are those who wish to convey information in a covert manner and avoid observation by unintended recipients. In this case the hidden message is more significant than the "carrier" object that is used to transport it.

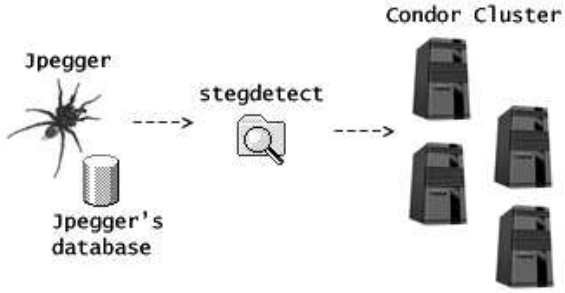
The Internet is said to be a storehouse of steganograpic material. Rumors persist that terrorists are exchanging messages over the Net by using steganography. In February 2001 Jack Kelly wrote two articles in USA TODAY, which indicated that Osama Bin Laden and his organization, Al-Qaeda, as well as other known terrorist groups were using steganography to plan and implement terrorist acts. It was suggested that stego-images were being placed on auction sites such as e-Bay and Amazon as well as sports chat rooms and pornographic sites. Several other news agencies ran similar articles later in the month. None of the articles offered definitive proof, other than anonymous quotes from federal law enforcement agencies, indicating that stego-images had been found. There were several references to encrypted e-mail and files that had been recovered. [11]

Based on the allegations that terrorist organizations were using steganography, Niels Provos and Peter Honeyman, researchers at the University of Michigan, launched a project to determine the truth of the matter. In their technical report published on August 31, 2001, Provos and Honeyman outline the tools they used (Stegdetect, Stegbreak, Crawl and Disconcert) to launch an automated, statistical analysis of over 2 million JPEG images found on the e-Bay web site, and 1 million images from the USENET archive. Our framework for detection is rooted in Provos' and Honeyman's work. [3]

METHODS

Whereas Provos and Honeyman used images from E-bay and USENET, the JPEG images samples for this work was derived from a random set of spidered websites. The web crawler was started the a set of large websites, including yahoo.com, news.google.com, msn.com, dmoz.org, and other aggregator and informational sites.

We use a tool developed by Provos, Stegbreak, to scan the image set to determine possible Steganographic suspect images.



DETECTION FRAMEWORK – Jpegger collects images, stores their information in a database, which is queried by stegdetect to find the potential images, which are sent to the cluster

This framework include both passive and active steganalysis. These are defined as:

- Passive steganalysis: Detect the presence or absence of a secret message in an observed message.
- Active steganalysis: Extract a (possibly approximate) version of the secret message from a stego message. [1]

Revisiting Provos and Honeyman

As we have previously mentioned, the framework of this research is similar to the University of Michigan's, however we wanted to complete this task with a cross platform approach. Our process involves:

1. Jpegger, a Java web crawler to scan and save JPEG images from the Internet. Jpegger is multi-threaded, and stores a list of URLs it has visited to avoid backtracking, and stores its current state during it's crawling in case of a failure. Jpegger uses HTMLParser, an Open Source html parser available at http://htmlparser.sourceforge.net.
2. Condor, a specialized workload management system for compute-intensive jobs developed by the University of Wisconsin. [11] Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress,

and ultimately informs the user upon completion.

3. Stegbreak, a tool for launching dictionary attacks against JSteg-Shell, JPHide, and OutGuess, developed by Provos. Stegbreak utilizes Jack the Ripper, a popular hacker's tool to formulate effective dictionary attacks.

How Jpegger works

Jpegger is a multi-threaded Java web crawler that performs a breadth-first crawl on the Internet, collecting JPEG images to download and test.

Jpegger uses a MySQL database to keep track of what URLs it has visited, and to keep a queue of potential URLs to visit. Jpegger can be configured as to how many concurrent threads it will run, most of our searching ran with 5 concurrent searching threads.

How Condor works

Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor's ease of setup and use were great advantages of us; it allowed us to easily create and distribute jobs to client machines, and monitors the execution of these jobs with great efficiency.

Our use of Condor collection includes 50 condor nodes (ranging in speed from 1 to 1.8 Ghz with an average of 256mb RAM), with a 4 processor condor master server.

Part of the decision to use Condor involves it's usability on a Windows platform; one of the main changes we wanted to make in our detection framework was to use a platform for development of the framework that could be deployed on a end-to-end Microsoft Windows network. All of these tools will work on Linux/UNIX platform, however we believe it is important to point out this fact. We believe there is an untapped potential in Windows based network machines used for University students labs and the like, which could become great resources for clustered scientific computing available to campuses without the need or resources for actual super computing facilities.

How Stegdetect works

Stegdetect detects images that have content hidden with JSteg, JPHide and OutGuess 0.13b.

One property of encrypted data is that the one and the zero bit are equally likely. When using the least-significant bit method to embed encrypted data into an image that contains color two more often than color three, color two is changed more often to color three than the other way around. As a result, the difference in color frequency between two and three is reduced by the embedding.

Instead of measuring the color frequencies, Provos / Honeyman analyzed the frequency of the DCT coefficients.

They used a x2-test to determine whether an image shows distortion from embedding hidden data. Because the test uses only the stego medium, the expected distribution for the x2-test has to be computed from the image. They assume that an image with hidden data embedded has similar frequency for adjacent DCT coefficients. As a result, they can take the arithmetic mean to determine the expected distribution.

$$y_i^* = \frac{n_{2i} + n_{2i+1}}{2}$$

The expected distribution is compared against the observed distribution.

$$y_i = n_{2i}.$$

The x2 value for the distribution can then be computed.

$$\chi^2 = \sum_{i=1}^{\nu+1} \frac{(y_i - y_i^*)^2}{y_i^*}$$

The probability of embedding is then given by the complement of a cumulative distribution function.

$$p = 1 - \int_0^{\chi^2} \frac{t^{(\nu-2)/2} e^{-t/2}}{2^{\nu/2}\Gamma(\nu/2)} dt$$

Stegdetect can compute the probability of embedding for different parts of an image. The selection depends on what steganographic system we try to detect. For an image that does not contain any hidden information, it is expected the probability of embedding to be zero everywhere.

Stegdetect cannot detect content embedded using Outguess 0.2, the latest version as of this writing. Outguess 0.2b uses an embedding method which does not modify any known statistical distribution. Since the development of Stegdetect, researchers have discovered a way to reliably detect Outguess 0.2, however this is not implemented in Stegdetect.[4] [6]

For each system that we want to detect, they select the DCT coefficients in the order that they are modified and apply a x squared test. The test compares the image against known statistical properties of images, determining the possibility of embedding. Note: This is a simplified explanation of the steganographic detection process. There is a much more exact process for detection based on encoding methods of each stegonographic medium. See the references for more information.

The output from Stegdetect lists the steganographic systems found in each image or "negative" if no steganographic content could be detected. Stegdetect expresses the level of confidence of the detection with one to three stars.

All of these tools are free and available for download on a Windows / Linux / UNIX platform. Stegdetect and Stegbreak are available at http://www.outguess.org, Condor at http://www.cs.wisc.edu/condor/, and Jpegger and any other tools and scripts we have developed in the process, are available at http://www.upb.pitt.edu/something/.

How Stegbreak works

To verify that the flagged images have hidden content, it is necessary to launch a dictionary attack against the JPEGs in order to decrypt them. Stegbreak does just that for content hidden with the methods used by Stegdetect

Our run of Stegbreak used a 1,323,398 word dictionary, which was a combination of several freely available normal English word dictionaries at http://wordlist.sourceforge.net/.

Stegbreak utilizes the password breaking facilities of Jack the Ripper, which provides us with some variations on the normal dictionary space – Jack the Ripper reorganizes and adds extra characters to the dictionary words during testing, allowing using to include some extra, more specialized passwords.

RESULTS

We ran Jpegger, and downloaded over 250,000 images. Out of those 250,000 images, Stegbreak found 20,009 images as suspect. The images were distributed to the Condor cluster.

The Condor cluster machines attempted a dictionary attack on Stegbreak using a dictionary of 1,323,398 words, which takes 3.8 minutes to run on a local machine, up to 15 minutes under Condor's distribution system, from job submission to the final return of the output. This time can vary, however – some jobs were completed in 5 minutes, many up to 12 or 13, depending on network traffic, server load, etc. The average job completed in 7.38 minutes. Most of this time was used on transferring of the Condor job, since Stegbreak completed with a median 2.88 minutes to run the dictionary attack, averaging 7649.7 words per second.

LIMITATIONS

It would help to have the time to collect a larger sample size of random images, however, we began to run into time restraints in our collection using Jpegger. Spanning the web collection across a group of clustered nodes would be a more efficient way to scan the Internet, however, we did not have the time to fully test Jpegger running on more than one machine asynchronously. Also, we would have liked to implement our detection and scanning system to run concurrently, allowing for a continuous scan of the Internet, unabated by the Stegbreak phase. Stegbreak itself could also be improved, by adding the detection method for Outguess 0.2.

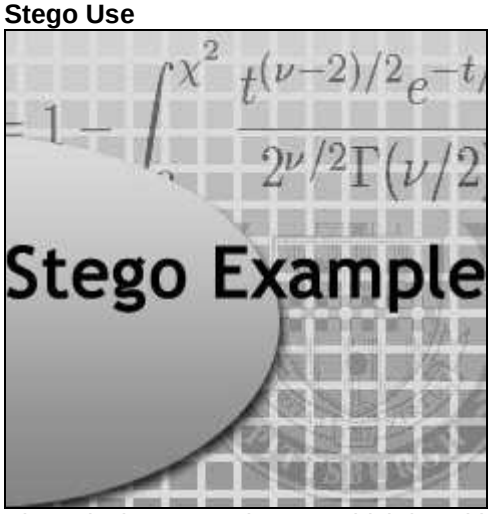
CONCLUSIONS

In our scanning of 200,000 images, we found a result set of 20,009 potential 'suspects' of encryption. This is a fairly high number, roughly 10 percent, exponentially higher than Provos's results. This could be due to a number of reasons:

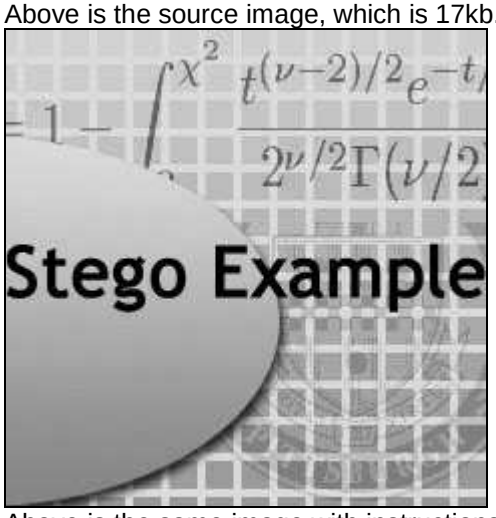
- There is a relatively high use of stegonagraphy on the Internet, and the creation of stegonagraphy monitoring and detection systems is important.
- Our set of testing images was somehow skewed; it would be advantageous to collect a larger number of images to confirm or deny this.
- The detection of suspects is flawed, and the number of suspects is considerably higher than it should be. This is unlikely, due to Provos' research and work, it is reasonable to believe that stegbreak is a reliable source for potentials.
- We also used images of any size – Provos and Honeyman used a limit 20 to 400k. However, we can reliabily embed high amounts (at least 50% of the image size) of content in images smaller than 20k, without extremely noticeable visual distortion. Combining this with the relative image qualities on the Internet, we believe it is more than possible images under 20k could be used as carriers. (refer to the sidebar title 'The 20k File Size Barrier' for an example).

If we re-evaluate our results without the images under 20k, we see 1303 suspects, which is ~.51 percent of the images we downloaded. This is still higher than Provo's numbers – they had 17,000 suspect images out of 2,000,000 images. This computes to .085 percent, which means we received 6 times the amount of suspect images that Provos's test did.

The 20kb File Size Barrier : An Example of Stego Use



Above is the source image, which is 17kb.



Above is the same image with instructions to create tear gas embedded within it using JPHide. (the password is 'example'). Both of these images are under 20kb.

We have tested all images using Stegbreak, however, we as of yet have not been able to break into an image found in 'the wild'. We are planning on running another attack with a larger dictionary.

Stegdetect Results	
Stego Type	**Number (Percent)**
JSteg	172 (.0085)
JPHide	19,806 (.98)
Outguess(old)	30 (.0014)
F5	1 (.0000049)

There are several potential reasons why we returned no positives, including:

- None of these images actually contain steganographic content. This is possible, but not likely; Stegdetect has a refined detection method.
- Our dictionary was not strenuous enough; a larger dictionary may allow us to gain success.

We believe the latter is true; we plan on returning to this data set with a larger dictionary attack.

ACKNOWLEDGEMENTS

We would like to thank the developers of the software we have used, most importantly Niels Provos and Professor Honeyman, for Stegbreak and the inspiration for this research, and the Condor development team at the University of Wisconsin.

REFERENCES

[1] R. Chandramouli, *"A Mathematical Approach to Steganalysis"*, Jan. 2002

[2] Paul Davern, "*Steganography: its history and its application to computer based data files*", Dublin City University School of Computer Applications, Working Paper: CA-0795

[3] Niels Provos and Peter Honeyman, "*Detecting Steganographic Content on the Internet*", Center for Information Technology Integration, University of Michigan.

[4] Jessica Fridrich, Miroslav Goljan, and Dorin Hogea, "*Attacking the Outguess*", Dept. of ECE at SUNY Binghamton

[5] Lisa M. Marvel, "*Image Steganography for Hidden Communication*", Dissertation – University of Delaware, Spring 1999.

[6] Niels Provos, "*Defending Against Statistical Steganalysis*", Center for Information Technology Integration – University of Michigan.

[7] Jessica Fridrich, and Miroslav Goljan, "*Practical Steganalysis of Digital Images – State of the Art* " , Dept ECE at SUNY Binghamton.

[9] Jonathan Watkins, "*Steganography – Messages Hidden in Bits*", Multimedia Systems Coursework, Dept of Electronics and CS, University of Southampton, SO17 1BJ, UK.

[10] Joshua Silman "Steganography and Steganalysis: An Overview**"** Sans.org http://www.sans.org/rr/steg/overview.php

[11] The Condor Project Homepage University of Wisconsin-Madison Computer Science Department website http://www.cs.wisc.edu/condor/