Lab Section 3.2.1

Section Author: Jaci Rojas

Embedding Information into Files with Steganography

---

1. Overview

Section 3.2.1 uses steganography techniques to embed an image (cover image: nyc-skyline.jpg), an audio file (cover sound: crickets_1.wav), and text files (cover short text: short_text.txt & cover long text:long_text.txt). On my macOS environment, I used the tools **steghide** and **snow** to create16 stego files for these 4 cover files.

In the subsection ‘Embedding Process’, the lines in red demonstrate the challenges I faced while embedding. For example, when attempting to embed an image into the cover image, the initial attempt failed due to the cover file's insufficient capacity. This made it necessary to resize the embedded image (cat.jpg) so it fits within the cover file's limitations. Another challenge I faced was while embedding audio files into the cover audio file since it encountered a format compatibility issue. The original WAV file must be converted to a PCM format in order to successfully embed. This process involved using FFmpeg so the audio file meets the required specifications for steganographic embedding.

I had to use a different approach when embedding the text files with the **snow** tool since it allows you to hide messages with text files by encoding secret messages in the whitespace. Since snow only works with text, it was required to convert the data from non-text files into Base64 format before embedding to ensure that the text data can be correctly hidden within the cover files.

2. How would you embed information into a video (e.g. MPEG4) file? Give an example.

To embed information into a video file like an MPEG4 file using **steghide**, you would use the command-line tool to hide the information inside the video file. The following command is shown as an example.

steghide embed -cf video.mp4 -ef message.txt -sf output.mp4

3. Embedding Process

Embedding the Cover Image: nyc-skyline.jpg

steghide embed -cf nyc-skyline.jpg -ef cat.jpg -sf stego_img.jpg -v
Result: Failed because “the cover file is too short to embed the data”.
Need to check cover file capacity.
steghide info nyc-skyline.jpg
Result: Cover file had a capacity of 28.4K, which was less than the side of cat.jpg (36K).
magick cat.jpg -quality 65 compressed_cat.jpg
Result: cat.jpg was reduced to 28K after resizing.
steghide embed -cf nyc-skyline.jpg -ef compressed_cat.jpg -sf stego_skyline_img.jpg -v
Result: Successfully created stego_skyline_img.jpg.
steghide embed -cf nyc-skyline.jpg -ef elephant_1.wav -sf stego_skyline_sound.jpg -v
Result: Successfully created stego_skyline_sound.jpg.
steghide embed -cf nyc-skyline.jpg -ef short_text2.txt -sf stego_skyline_shorttxt.jpg -v
Result: Successfully created stego_skyline_shorttxt.jpg.
steghide embed -cf nyc-skyline.jpg -ef long_text2.txt -sf stego_skyline_longtxt.jpg -v
Result: Successfully created stego_skyline_longtxt.jpg.

Embedding the Cover Sound: crickets_1.wav

steghide embed -cf crickets_1.wav -ef cat.jpg -sf stego_crickets_img.wav -v
Result: Failed due to unsupported format.
Need to convert the WAV file to a PCM WAV file.
ffmpeg -i crickets_1.wav -acodec pcm_s16le -ar 44100 -ac 2 converted_crickets.wav
ffmpeg -i converted_crickets.wav -map_metadata -1 -c:a pcm_s16le -ar 44100 -ac 2 clean_crickets.wav
Result: To avoid segmentation fault, this step removes all metadata from converted_crickets.wav.
steghide embed -cf clean_crickets.wav -ef compressed_cat.jpg -sf stego_crickets_img.wav -v
Result: Successfully created stego_crickets_img.wav.

steghide embed -cf clean_crickets.wav -ef elephant_1.wav -sf stego_crickets_sound.wav -v
Result: Successfully created stego_crickets_sound.wav.
steghide embed -cf clean_crickets.wav -ef short_text2.txt -sf stego_crickets_shorttxt.wav -v
Result: Successfully created stego_crickets_shorttxt.wav.
steghide embed -cf clean_crickets.wav -ef long_text2.txt -sf stego_crickets_longtxt.wav -v
Result: Successfully created stego_crickets_longtxt.wav.

Embedding the Cover Short Text: short_text.txt

Need to convert image to Base64 before embedding.
base64 -i compressed_cat.jpg -o compressed_cat.txt
snow -C -f compressed_cat.txt -p 123 short_text.txt stego_shorttxt_img.txt
Result: Successfully created stego_shorttxt_img.txt.
Need to convert sound to Base64 before embedding.
base64 -i elephant_1.wav -o elephant_1.txt
snow -C -f elephant_1.txt -p 123 short_text.txt stego_shorttxt_sound.txt
Result: Successfully created stego_shorttxt_sound.txt.
snow -C -f short_text2.txt -p 123 short_text.txt stego_shorttxt_shorttxt.txt
Result: Successfully created stego_shorttxt_shorttxt.txt.
snow -C -f long_text2.txt -p 123 short_text.txt stego_shorttxt_longtext.txt
Result: Successfully created stego_shorttxt_longtext.txt.

Embedding the Cover Long Text: long_text.txt

snow -C -f compressed_cat.txt -p 123 long_text.txt stego_longtxt_img.txt
Result: Successfully created stego_longtxt_img.txt.
snow -C -f elephant_1.txt -p 123 long_text.txt stego_longtxt_sound.txt
Result: Successfully created stego_longtxt_sound.txt.
snow -C -f short_text2.txt -p 123 long_text.txt stego_longtxt_shorttxt.txt
Result: Successfully created stego_longtxt_shorttxt.txt.
snow -C -f long_text2.txt -p 123 long_text.txt stego_longtxt_longtext.txt
Result: Successfully created stego_longtxt_longtext.txt.

Completed Stego Files (16 in total):

stego_skyline_img.jpg
stego_skyline_sound.jpg
stego_skyline_shorttxt.jpg
stego_skyline_longtxt.jpg

stego_crickets_img.wav
stego_crickets_sound.wav
stego_crickets_shorttxt.wav
stego_crickets_longtxt.wav

stego_shorttxt_img.txt
stego_shorttxt_sound.txt
stego_shorttxt_shorttxt.txt
stego_shorttxt_longtext.txt

stego_longtxt_img.txt
stego_longtxt_sound.txt
stego_longtxt_shorttxt.txt
stego_longtxt_longtext.txt

Lab Section 3.2.2

Section Author: Jaci Rojas

Information Gathering and Extraction

---

1. Overview

This section builds off of 3.2.1 by extracting embedded data from our stego files. The following extraction process is broken up into two parts: information gathering & data extraction. The tools I used for this task include **steghide** for image and audio files, and **snow** for text files. **Snow** was used to extract the data and provide confirmation on whether or not extraction was successful by displaying the contents of the hidden files.

In the subsection ‘Information Gathering’, we retrieve the metadata for each of our 16 stego files. Additionally, this includes its format, capacity, and embedded files (if it has any). Each stego file also has a passphrase in order to access the embedded data, which is revealed in the output for each. After the correct passphrase is entered, the output reveals the hidden files, and its contents (their names, sizes, and if they are encrypted/compressed).

In the subsection ‘Data Extraction’, we perform the actual extraction of the embedded files. In order to do so, we must use the same passphrase that was used in ‘Information Gathering’ to allow the program to read the stego files and extract the hidden data into a given output file. For image and audio files, this process included verifying the integrity of the extracted data with CRC32 checksum as a means to ensure that extraction was successful.

2. Information Gathering

For stego_skyline_img.jpg:

steghide info stego_skyline_img.jpg

Output:
"stego_skyline_img.jpg":
  format: jpeg
  capacity: 28.4 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 219495116
  embedded file "compressed_cat.jpg":
    size: 27.6 KB
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_skyline_sound.jpg:

steghide info stego_skyline_sound.jpg

Output:
 "stego_skyline_sound.jpg":
  format: jpeg
  capacity: 28.4 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 219495116
  embedded file "elephant_1.wav":
    size: 7.7 KB
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_skyline_shorttxt.jpg:

steghide info stego_skyline_shorttxt.jpg

Output:
"stego_skyline_shorttxt.jpg":
  format: jpeg
  capacity: 28.4 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 219495116
  embedded file "short_text2.txt":
    size: 381.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_skyline_longtxt.jpg:

steghide info stego_skyline_longtxt.jpg

Output:
"stego_skyline_longtxt.jpg":
  format: jpeg
  capacity: 28.4 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 219495116
  embedded file "long_text2.txt":
    size: 811.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_crickets_img.wav:

steghide info stego_crickets_img.wav

Output:
"stego_crickets_img.wav":
  format: wave audio, PCM encoding
  capacity: 28.1 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 123
  embedded file "compressed_cat.jpg":
    size: 27.6 KB
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_crickets_sound.wav:

steghide info stego_crickets_sound.wav

Output:
"stego_crickets_sound.wav":
  format: wave audio, PCM encoding
  capacity: 28.1 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 123
  embedded file "elephant_1.wav":
    size: 7.7 KB
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_crickets_shorttxt.wav:

steghide info stego_crickets_shorttxt.wav

Output:
"stego_crickets_shorttxt.wav":
  format: wave audio, PCM encoding
  capacity: 28.1 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 123
  embedded file "short_text2.txt":
    size: 18.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_crickets_longtxt.wav:

steghide info stego_crickets_longtxt.wav

Output:
"stego_crickets_longtxt.wav":
  format: wave audio, PCM encoding
  capacity: 28.1 KB
Try to get information about embedded data? (y/n) y
Enter passphrase: 123
  embedded file "long_text2.txt":
    size: 434.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

For stego_shorttxt_img.txt:

snow -C -p 123 stego_shorttxt_img.txt

Output:
Shows content from hidden file compressed_cat.txt

For stego_shorttxt_sound.txt:

snow -C -p 123 stego_shorttxt_sound.txt

Output:
Shows content from hidden file elephant_1.txt

For stego_shorttxt_shorttxt.txt:

snow -C -p 123 stego_shorttxt_shorttxt.txt

Output:
Shows content from hidden file short_text2.txt

For stego_shorttxt_longtext.txt:

snow -C -p 123 stego_shorttxt_longtext.txt

Output:
Shows content from hidden file long_text2.txt

For stego_longtxt_img.txt:

snow -C -p 123 stego_longtxt_img.txt

Output:
Shows content from hidden file compressed_cat.txt

For stego_longtxt_sound.txt:

```
snow -C -p 123 stego_longtxt_sound.txt
```

Output:
Shows content from hidden file elephant_1.txt

For stego_longtxt_shorttxt.txt:

```
snow -C -p 123 stego_longtxt_shorttxt.txt
```

Output:
Shows content from hidden file short_text2.txt

For stego_longtxt_longtext.txt:

```
snow -C -p 123 stego_longtxt_longtext.txt
```
Output:
Shows content from hidden file long_text2.txt

3. Extracting Data

For stego_skyline_img.jpg:

```
steghide extract -sf stego_skyline_img.jpg -xf extracted_skyline_img.txt -v
```

Output:
Enter passphrase: 219495116
reading stego file "stego_skyline_img.jpg"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_skyline_img.txt"... done

For stego_skyline_sound.jpg:

```
steghide extract -sf stego_skyline_sound.jpg -xf extracted_skyline_sound.txt -v
```

Output:
Enter passphrase: 219495116
reading stego file "stego_skyline_sound.jpg"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_skyline_sound.txt"... done

For stego_skyline_shorttxt.jpg:

steghide extract -sf stego_skyline_shorttxt.jpg -xf extracted_skyline_shorttxt.txt -v

Output:
Enter passphrase: 219495116
reading stego file "stego_skyline_shorttxt.jpg"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_skyline_shorttxt.txt"... done

For stego_skyline_longtxt.jpg:

steghide extract -sf stego_skyline_longtxt.jpg -xf extracted_skyline_longtxt.txt -v

Output:
Enter passphrase: 219495116
reading stego file "stego_skyline_longtxt.jpg"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_skyline_longtxt.txt"... done

For stego_crickets_img.wav:

steghide extract -sf stego_crickets_img.wav -xf extracted_crickets_img.txt -v

Output:
Enter passphrase: 123
reading stego file "stego_crickets_img.wav"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_crickets_img.txt"... done

For stego_crickets_sound.wav:

steghide extract -sf stego_crickets_sound.wav -xf extracted_crickets_sound.txt -v

Output:
Enter passphrase: 123
reading stego file "stego_crickets_sound.wav"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_crickets_sound.txt"... done

For stego_crickets_shorttxt.wav:

steghide extract -sf stego_crickets_shorttxt.wav -xf extracted_crickets_shorttxt.txt -v

Output:
Enter passphrase: 123
reading stego file "stego_crickets_shorttxt.wav"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_crickets_shorttxt.txt"... done

For stego_crickets_longtxt.wav:

steghide extract -sf stego_crickets_longtxt.wav -xf extracted_crickets_longtxt.txt -v

Output:
Enter passphrase: 123
reading stego file "stego_crickets_longtxt.wav"... done
extracting data... done
checking crc32 checksum... ok
writing extracted data to "extracted_crickets_longtxt.txt"... done

For stego_shorttxt_img.txt:

snow -C -p 123 stego_shorttxt_img.txt extracted_shorttxt_img.txt
Use the following command to confirm successful extraction.
cat extracted_shorttxt_img.txt

Output:
The contents of the hidden file compressed_cat.txt

For stego_shorttxt_sound.txt:

snow -C -p 123 stego_shorttxt_sound.txt extracted_shorttxt_sound.txt
Use the following command to confirm successful extraction.
cat extracted_shorttxt_sound.txt

Output:
The contents of the hidden file elephant_1.txt

For stego_shorttxt_shorttxt.txt:

snow -C -p 123 stego_shorttxt_shorttxt.txt extracted_shorttxt_shorttxt.txt
Use the following command to confirm successful extraction.
cat extracted_shorttxt_shorttxt.txt

Output:
The contents of the hidden file short_text2.txt

For stego_shorttxt_longtext.txt:

snow -C -p 123 stego_shorttxt_longtext.txt extracted_shorttxt_longtext.txt
Use the following command to confirm successful extraction.
cat extracted_shorttxt_longtext.txt

Output:
The contents of the hidden file long_text2.txt

For stego_longtxt_img.txt:

snow -C -p 123 stego_longtxt_img.txt extracted_longtxt_img.txt
Use the following command to confirm successful extraction.
cat extracted_longtxt_img.txt

Output:
The contents of the hidden file compressed_cat.txt

For stego_longtxt_sound.txt:

snow -C -p 123 stego_longtxt_sound.txt extracted_longtxt_sound.txt
Use the following command to confirm successful extraction.
cat extracted_longtxt_sound.txt

Output:
The contents of the hidden file elephant_1.txt

For stego_longtxt_shorttxt.txt:

snow -C -p 123 stego_longtxt_shorttxt.txt extracted_longtxt_shorttxt.txt
Use the following command to confirm successful extraction.

cat extracted_longtxt_shorttxt.txt

Output:
The contents of the hidden file short_text2.txt

For stego_longtxt_longtext.txt:

snow -C -p 123 stego_longtxt_longtext.txt extracted_longtxt_longtext.txt
Use the following command to confirm successful extraction.
cat extracted_longtxt_longtext.txt

Output:
The contents of the hidden file long_text2.txt

Lab Section 3.2.3

Section Author: David Xiao

Covert TCP Communication using SPHERE XDC

---

1. Objective

This section focuses on the implementation and analysis of covert data transmission through TCP/IP headers using the **Covert_TCP** tool within the **SPHERE XDC** environment. The primary objective was to embed a secret message into TCP packets, capture the transmitted data, and reconstruct the hidden message through packet analysis.

---

2. Tools and Materials

- **SPHERE XDC Environment**: Virtual platform used for running the lab environment.
- **Covert_TCP**: Utility for embedding data within TCP/IP packet headers.
- **tcpdump**: Packet capture and analysis tool.
- **Files Utilized:**
    - `secret.txt`: File containing the original message.
    - `received-secret.txt`: Output file with the reconstructed message.
    - `covert-data.pcap`: Captured packet data.
    - Terminal logs: `terminal1.txt`, `terminal2.txt`, `terminal3.txt`.

---

3. Methodology

3.1 Receiver Setup

To begin the experiment, the receiver was initialized to listen for covert TCP messages:

- sudo ./covert_tcp -source 127.0.0.1 -server -file received-secret.txt

Result: The receiver captured incoming data and saved it to `received-secret.txt` without errors.

3.2 Packet Capture Initialization

Simultaneously, packet capturing was set up using tcpdump to monitor loopback interface traffic:

- sudo tcpdump -i lo -s 1500 -n -w covert-data.pcap

Result: A total of 118 packets were captured and stored in `covert-data.pcap`.

3.3 Data Transmission

The sender was activated to transmit the embedded message:

- sudo ./covert_tcp -source 127.0.0.1 -dest 127.0.0.1 -file secret.txt

Result: The message was sent successfully with no transmission errors.

---

4. Packet Analysis and Message Reconstruction

4.1 Extracting Packet Data

Using tcpdump, captured packets were analyzed to locate the covert data:

- tcpdump -n -r covert-data.pcap -X -v dst port 80

Observation: The IP Identification (ID) field was utilized for embedding the message.

4.2 Data Conversion Process

1. Extracted the IP ID field values from `terminal2.txt`.
2. Converted hexadecimal IP ID values to decimal equivalents.
3. Mapped the decimal values to their corresponding ASCII characters.

Conversion Sample:

IP ID (Hex)	Decimal	ASCII
0x5500	21760	U
0x7300	29440	s
0x6500	25856	e
0x2000	8192	(space)
0x7400	29696	t
0x6800	26624	h
0x6500	25856	e
0x2000	8192	(space)
0x6300	25344	c
0x6f00	28416	o
0x7600	30208	v
0x6500	25856	e
0x7200	29184	r
0x7400	29696	t
0x7800	30720	x
0x7400	29696	t
0x2000	8192	(space)
0x6600	26112	f
0x6900	26880	i
0x6c00	27648	l
0x6500	25856	e
0x7300	29440	s

4.3 Reconstructed Message

- Use the covertxt files as the embedfile and the coverfiles.

Result: The reconstructed message perfectly matched the original content of `secret.txt`.

4.4 Verification

A final verification was performed using the `diff` command:

diff -c secret.txt received-secret.txt

Result: No differences were found, confirming accurate reconstruction.

---

5. Challenges and Resolutions

- **Missing tcpdump Installation:** Resolved by running `sudo apt install tcpdump -y`.
- **Execution Sequence Errors:** Prevented data loss by starting the receiver and packet capture before initiating the sender.
- **Permission Denied Errors:** All commands requiring elevated privileges were executed with `sudo`.

---

6. Conclusion

This lab demonstrated the feasibility of using TCP/IP headers, specifically the IP ID field, to establish covert communication channels. The successful transmission and reconstruction of the hidden message emphasized the importance of monitoring seemingly innocent packet fields for potential data leaks in cybersecurity environments.

---

7. Recommendations

- Experiment with different TCP/IP header fields (e.g., sequence numbers, flags) for covert data embedding.
- Test the implementation over non-localhost networks to assess real-world implications.
- Explore encryption methods to increase the stealth of covert channels.
- Investigate mitigation techniques to detect and prevent similar covert communications.

---

8. Appendix

- `terminal1.txt`: Receiver terminal log
- `terminal2.txt`: Packet analysis log
- `terminal3.txt`: Sender terminal log
- `secret.txt`: Original message file

- `received-secret.txt`: Reconstructed message file
- `covert-data.pcap`: Captured packet data for analysis

Lab Section 4.1 Capacity of the Cover Files

Author: David Xiao

Objective

The objective of this section is to analyze the capacity of different cover files used in steganography using the `steghide` tool. The analysis includes determining how much information can be embedded into various files, expressed as a percentage of the file's capacity and overall size.

Procedure

We used the `steghide` tool to assess the embedding capacity of JPEG files and to measure the size of the embedded files. The process involved sequentially embedding different file types into the cover image and documenting the capacity usage at each stage.

Commands Used:

convert cats.jpg -quality 65 compressed_cats.jpg
convert jordans.jpg -quality 65 compressed-jordans.jpg
steghide info nyc-skyline.jpg
steghide info 4_1_1.jpg -p 123
steghide info 4_1_2.jpg -p 123
steghide info 4_1_3.jpg -p 123
steghide info 4_1_4.jpg -p 123
ls -lh nyc-skyline.jpg crickets_1.wav elephant_1.wav compressed_cats.jpg
compressed-jordans.jpg 4_1_1.jpg 4_1_2.jpg 4_1_3.jpg 4_1_4.jpg

---

Results

Capacity and File Size Data

File Name	Type	Capacity (KB)	Embedded File	Embedded Size (KB)	File Size (KB)	% of Capacity Used	% of Cover File Used
nyc-skyline.jpg	Image	28.4	None	N/A	467	N/A	N/A
4_1_1.jpg	Image	28.4	crickets_1.wav	14.9	488	52.46%	3.19%
4_1_2.jpg	Image	28.4	elephant_1.wav	7.7	489	27.11%	1.65%
4_1_3.jpg	Image	28.4	compressed_cats.jpg	27.6	492	97.18%	5.42%
4_1_4.jpg	Image	28.4	compressed-jordans.jpg	23.2	493	81.69%	4.71%

Observations

- **Consistent Capacity:** Each cover image maintained a capacity of **28.4 KB** regardless of file size differences.
- **Capacity Usage Impact:** Embedding larger files (e.g., `compressed_cats.jpg`) used up to **97.18%** of the available capacity, significantly raising the risk of detection.
- **Minimal File Size Growth:** Despite high embedding percentages, overall file size increased only marginally relative to capacity.
- **Detection Considerations:** High capacity utilization and increased file sizes make detection through steganalysis more probable.

Lab Section 4.2 Chaining the Techniques

4.2.1 Steghide Method

Author: David Xiao

Objective

To examine the effects of chaining steganographic embeddings using the `steghide` tool. The experiment analyzes how file size, embedding capacity, and detectability evolve with successive layers of hidden data.

Procedure

Four embedding steps were performed with the `steghide` tool, each using the output file from the previous step as the cover file:

1. Embed `crickets_1.wav` into `nyc-skyline.jpg` → `4_1_1.jpg`
2. Embed `elephant_1.wav` into `4_1_1.jpg` → `4_1_2.jpg`
3. Embed `compressed_cats.jpg` into `4_1_2.jpg` → `4_1_3.jpg`
4. Embed `compressed-jordans.jpg` into `4_1_3.jpg` → `4_1_4.jpg`

Results and Observations

Embedding Step	Embedded File	Before Size	After Size	Size Increase	Capacity Usage (%)
1st	crickets_1.wav (14.9 KB)	467 KB	488 KB	+21 KB	52.46%
2nd	elephant_1.wav (7.7 KB)	488 KB	489 KB	+1 KB	27.11%
3rd	compressed_cats.jpg (27.6 KB)	489 KB	492 KB	+3 KB	97.18%
4th	compressed-jordans.jpg (23.2 KB)	492 KB	493 KB	+1 KB	81.69%

Key Points:

- Capacity usage peaked at **97.18%** during the third embedding.

- File size growth was modest relative to capacity utilization.
- Higher capacity usage correlated with increased detectability.
- Subsequent embeddings after high capacity usage yielded diminishing returns.

Conclusion

The `steghide` method demonstrated that while multiple embeddings are possible, exceeding 90% capacity significantly raises detection risks. Practical usage suggests limiting embeddings to prevent noticeable anomalies.

4.2.2 CAT Command Method

Author: Shareena Wiggins

Objective

To investigate the effects of chaining steganographic embeddings using the `cat` command, focusing on file size growth and detection risks compared to specialized tools like `steghide`.

Procedure

The process was done in the MAC OS environment. Files were concatenated to embed data without specialized encryption or compression. The process involved:

cat cats.jpg secret.txt > stego1.jpg
cat stego1.jpg elephant_1.wav > stego2.jpg
cat stego2.jpg skyline2.jpg > stego3.jpg
cat stego3.jpg kung-fu-panda.bmp > stego4.jpg

Results and Observations

File Size Increase Through CAT Method

Embedding Step	Description	Before Size	After Size	Size Increase

1st	cats.jpg + secret.txt → stego1.jpg	36 KB	36.1 KB	+0.1 KB
2nd	stego1.jpg + elephant_1.wav	36.1 KB	43.8 KB	+7.7 KB
3rd	stego2.jpg + skyline2.jpg	43.8 KB	172 KB	+128.2 KB
4th	stego3.jpg + kung-fu-panda.bmp	172 KB	972 KB	+800 KB

Key Observations:

- The CAT method resulted in a much steeper increase in file sizes compared to `steghide`.
- By the fourth embedding, the file reached **972 KB**, making it highly suspicious.
- No encryption or compression made the hidden data easier to detect.
- Visual quality remained unaffected until large files were embedded.

Practicality of Chaining Comparison

Aspect	Steghide Method	CAT Command Method
Capacity Usage	Controlled (up to 97%)	Unlimited, no control
File Size Growth	Moderate	Exponential
Detectability Risk	Moderate	Very High
Data Security	Encrypted & Compressed	None
Practical Use	Covert Communications	Educational Demonstrations

---

Conclusion

While both methods demonstrate the ability to chain embeddings, the **CAT command** lacks encryption and compression, making it less practical for covert purposes. In contrast, the **`steghide`**** method** offers a balanced trade-off between capacity usage and detection risk.

Final Notes:

- **Maximum Capacity Reached:** 97.18% with `steghide`.
- **Highest File Size:** 972 KB using the `cat` method.
- **Best Practice:** Use controlled tools like `steghide` to maintain file integrity and minimize detection.

Lab Section 5: Word Problems

Author: Shareena Wiggins

1. Summarize the embedding techniques used by the tools.

- Steganography uses a variety of methods to conceal data within files or network traffic. For files, tools like Steghide rely on modifying the least significant bits of cover files, such as images or audio, to embed hidden data. This approach alters pixels or audio samples in a way that is undetectable while preserving the original file. For network traffic, tools like Covert TCP hide data within header fields such as the IP Identification or TCP sequence numbers. By encoding information in unused or less noticeable areas of packets, covert communication channels can be established. These techniques enable the transferring of hidden messages while maintaining the appearance of normal traffic or files.

2. How would you detect the presence of steganography?

- You can detect the presence of steganography through a combination of visual and analytical methods. For files, differences in size, metadata, or visual inspections can indicate the presence of hidden data. Bit-level analysis, particularly examining LSBs for irregularities, can also reveal concealed information. For network traffic, tools such as Wireshark or tcpdump allow for the inspection of packet headers and payloads. Statistical analysis of header fields, entropy calculations, or flow patterns can expose covert channels. Unusual repetition in IP ID values, high entropy in sequence numbers, or structured patterns in seemingly random fields often serve as red flags for steganographic activity.

3. To what extent are the embedding techniques composable?

- The composability of embedding techniques depends on the type of cover medium and the method used. For files, techniques can often be layered to increase the amount of hidden data. For example, a text file can be embedded into an image, and that stego-image can be used as a cover file for another embedding. However, successive embedding reduces the quality of the cover file and increases the risk of detection due to noticeability or reduced capacity. On the other hand, network-based techniques are less composable, as modifying multiple header fields or layering encoding methods increases traffic anomalies and the likelihood of packet corruption during transmission. While chaining techniques can be effective, it comes with diminishing returns and a higher risk of detection.

4. How would you thwart steganographic efforts if you could be in the middle of the transmission, i.e., you take the role of an active warden and modify traffic in transit?

- If we could be in the middle of the transmission, various methods can be employed to disrupt hidden data. Introducing random noise into the LSBs of images or audio can render embedded messages unrecoverable. Compressing files also helps to remove hidden data, as many steganographic techniques rely on specific file structures that are disrupted by such transformations. In the case of network traffic, altering or randomizing header fields such as IP ID or TCP sequence numbers can destroy covert channels. Traffic normalization tools can strip unexpected patterns from packet headers or payloads, while deep packet inspection (DPI) tools can analyze traffic for unusual entropy levels or payload contents. Dropping packets with anomalies or rate-limiting non-standard traffic further prevents the transmission of steganographic data. By actively monitoring and modifying traffic or files, steganographic efforts can be effectively thwarted.

Sources:

- https://steghide.sourceforge.net/index.php
- https://firstmonday.org/ojs/index.php/fm/article/download/528/449