CSCI 400 Cryptography Lab 01 – Group 4

Topic: Steganography

Carl Laguerre: Image file embedding & section 4,5,6 for image files

Edward Cruz Peralta: Audio file embedding & section 4,5,6 for audio files

Jae Cho: Network embedding & section 4,5,6 for network

Akbor Uddin: Text file embedding & report writing

Tasks

1. Embedding information into files

Create a sample image file, which will be your cover file. Embed some information in the file, creating a separate stego file for each of the following:

1. another image

2. a sound file (e.g. a cricket chirp file)

3. a short text

4. a long text

Embed into each of the items (1. image file, 2. sound file, 3. short text file, 4. long text file) above another item (1-4). Having worked with the image file, move on to the sound file as a cover file, and embed another item into it, again creating 4 stego files. At the end, you should have 4 cover files (1-4), and 16 stego files.

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

Prepare a report on your findings.

2. Information gathering and extraction

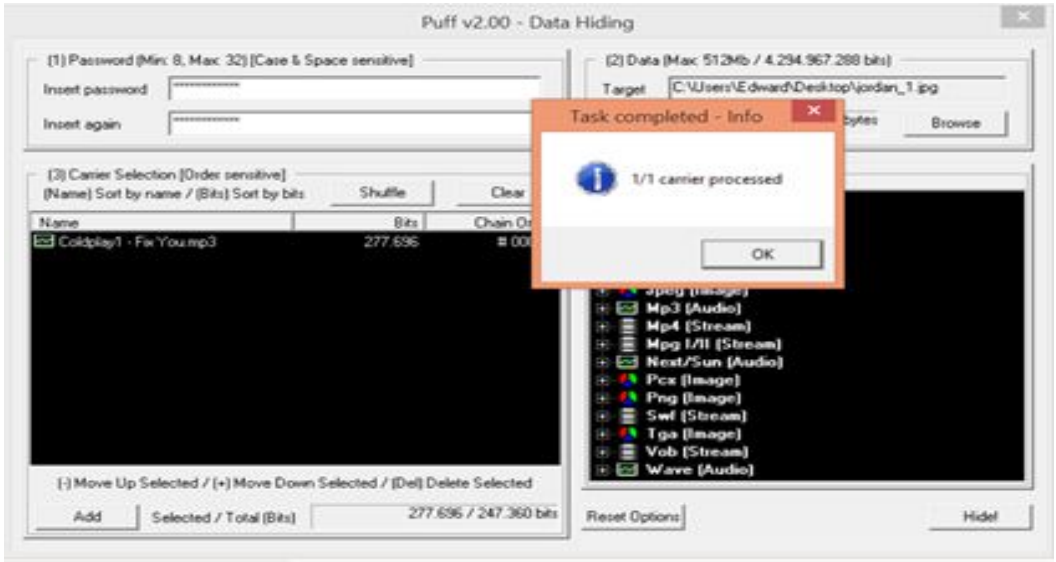
We can then check the file and extract information embedded in it, assuming we know what method was used to embed the information (e.g. steghide) and the passphrase is either known to us or empty.

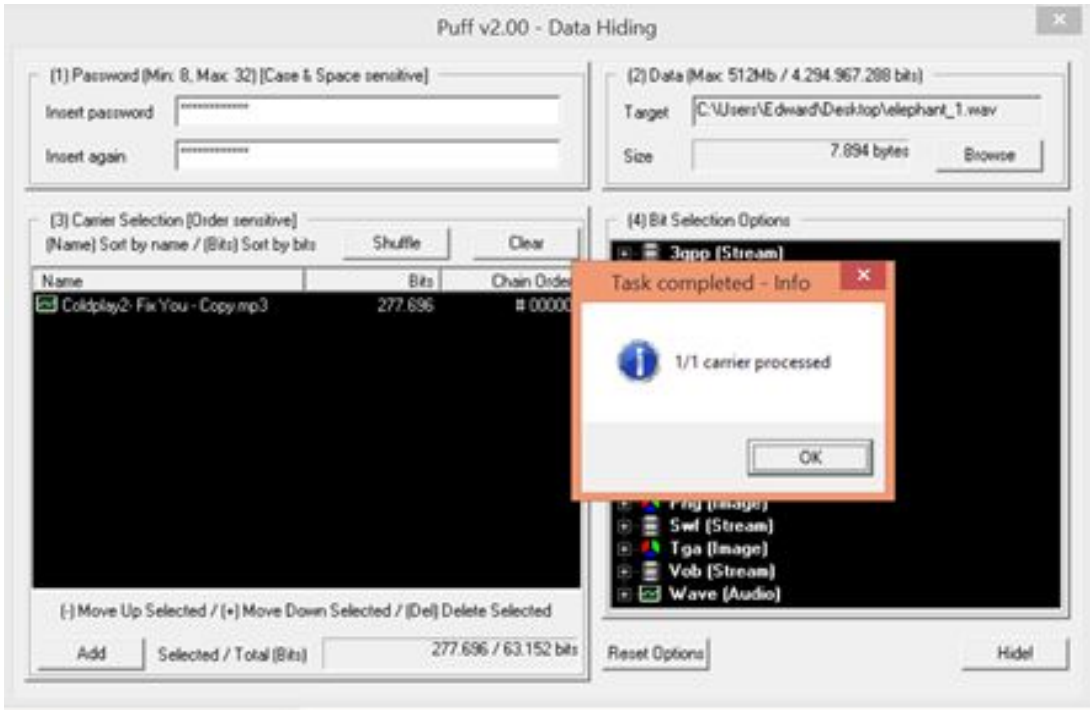
3.1 Information gathering and extraction - Sound file completed by Edward Cruz Peralta

We can then check the file and extract information embedded in it, assuming we know what method was used to embed the information (e.g. steghide) and the passphrase is either known to us or empty.

Embedding to a sound file

In this part of the project, the tool Puff v2.00 was used. While using the tool it was concluded that when mp3 file(SummerCricketsChirping) was used as a cover file, the bits where actually reduced from 76.2Kb to 900bits because the tool is using padding to hide the data shrinking the file size down. With this problem happening most of the files we were trying to embed wouldn't work. So the next idea was to obtain a music file in where the data was in MB instead of KB. While making the music file as a cover file, we notice that the same action happened, in where the bits size was reduced. The files that were being embedded would still work since the embedded files bits were much lower than the cover file. So, the embedding of a jpg, wav, short text, and long text was achieved. While performing the embedding, the password thekid149163 was used. While it was known that the files were embedded, the tool puff has an option to unhide cover files. So, the password that was used to embed the files was used to see if the embedding actually worked. Once the tool unhides the file and it was shown that the hidden file was discovered. It was known that this part of the project was complete.




Puff v2.00 - Data Hiding
(1) Password [Min: 8, Max: 32] [Case & Space sensitive]
Insert password
Insert again
(2) Data [Max: 512Mb / 4.294.967.288 bits]
Target
C:\Users\Edward\Desktop\elephant_1.wav
Size
7.894 bytes
Browse
(3) Carrier Selection [Order sensitive]
[Name] Sort by name / [Bits] Sort by bits
Shuffle
Clear
Name
Bits
Chain Order
Coldplay2- Fix You - Copy.mp3
277.696
00000
(4) Bit Selection Options
3gpp [Stream]
Swf [Stream]
Tga [Image]
Vob [Stream]
Wave [Audio]
Task completed - Info
1/1 carrier processed
OK
[-] Move Up Selected / [+] Move Down Selected / [Del] Delete Selected
Add
Selected / Total [Bits]
277.696 / 63.152 bits
Reset Options
Hide!


Puff v2.00 - Data Hiding
(1) Password [Min: 8, Max: 32] [Case & Space sensitive]
Insert password
Insert again
(2) Data [Max: 512Mb / 4.294.967.288 bits]
Target
C:\Users\Edward\Documents\longtext .txt
Size
1.104 bytes
Browse
(3) Carrier Selection [Order sensitive]
[Name] Sort by name / [Bits] Sort by bits
Shuffle
Clear
Name
Bits
Chain Order
Coldplay4 - Fix You.mp3
277.696
00000
(4) Bit Selection Options
3gpp [Stream]
Aiff [Audio]
Bitmap [Image]
Executable [File]
Wave [Audio]
Task completed - Info
1/1 carrier processed
OK
[-] Move Up Selected / [+] Move Down Selected / [Del] Delete Selected
Add
Selected / Total [Bits]
277.696 / 8.832 bits
Reset Options
Hide!

3.2 Embedding information into network traffic – completed by Jae Cho

The program covert_tcp is used to hide the data in TCP packets with 3 different encoding methods.

Lab Setup

Created 3 VirtualBox hosts with the network steganography tool, covert_tcp, and Wireshark installed.

Each VirtualBox host was configured with two network interfaces, enp0s3 and enp0s8 where enp0s3 is used for communication between host for network traffic sniffing test only and enp0s8 is used for internet connection. A destination port has been set as 80 on all tests.

Detailed network traffic capture file, pcap files, are included on network_capture.zip file

VirtualBox01: client/sender

```
jaecho@jae-VirtualBox01:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.11  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::a00:27ff:fe14:e6e8  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:14:e6:e8  txqueuelen 1000  (Ethernet)
        RX packets 3394  bytes 247950 (247.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3326  bytes 224255 (224.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

VirtualBox02: server/receiver

```
jaecho@jae-VirtualBox02:~/CSCI400/Lab01/Tools/covert_tcp/covert.tcp$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.12  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::a00:27ff:fe6f:dda3  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6f:dd:a3  txqueuelen 1000  (Ethernet)
        RX packets 2986  bytes 191150 (191.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3008  bytes 190790 (190.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

VirtualBox03: bounce IP

```
jaecho@jae-virtualbox03:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.13  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 fe80::def0:eff5:67b2:a06e  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:8c:6e:94  txqueuelen 1000  (Ethernet)
        RX packets 129  bytes 14897 (14.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 156  bytes 21134 (21.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Encoding\Transmission Method One: Manipulation of the IP Identification Field

Text file, secret1.txt has been sent via IP Identification field encoding from client_IP to server_IP:

This encoding method simply replaces the IP identification field with the numerical ASCII representation of the character to be encoded. This allows for easy transmission to a remote host which simply reads the IP identification field and translates the encoded ASCII value to its printable counterpart.

Destination Host: 192.168.43.12 Source Host : 192.168.43.11

Originating Port: random Destination Port: 80

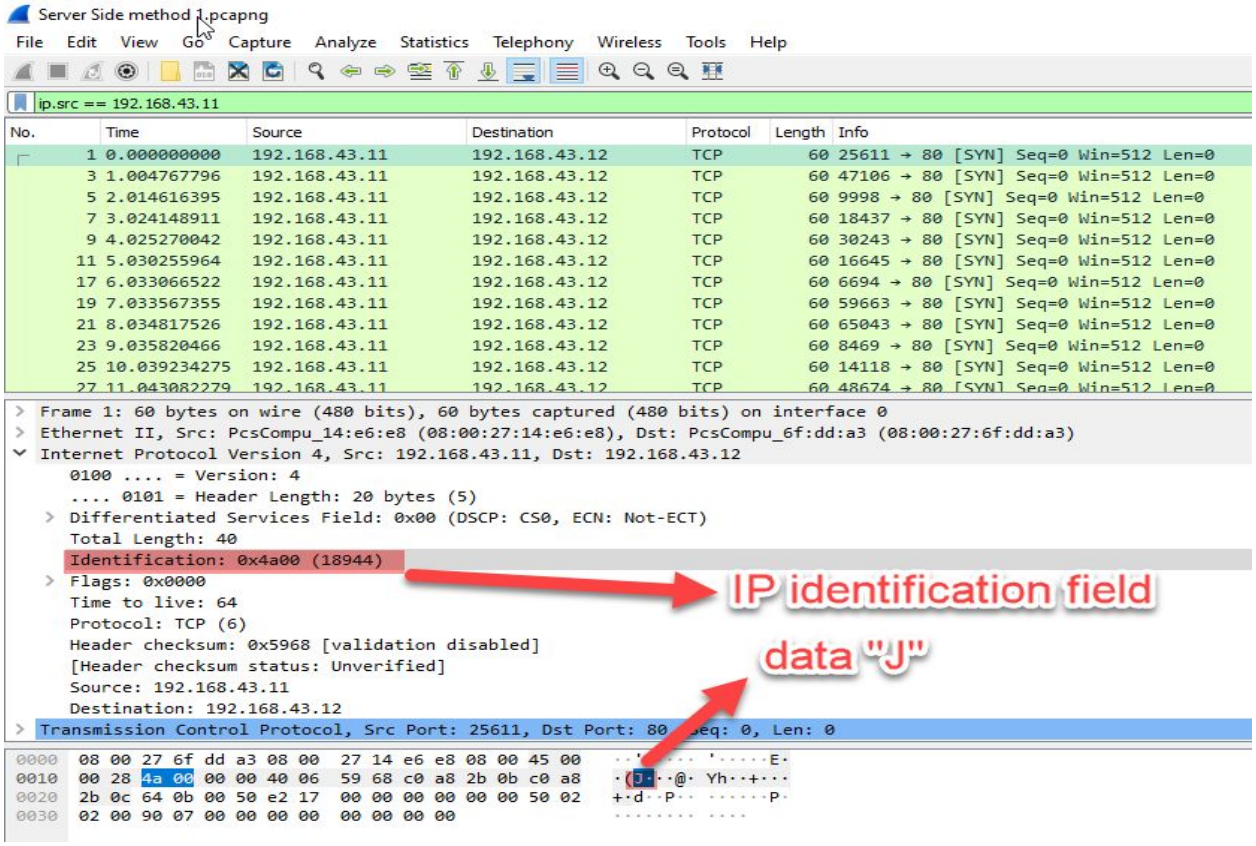
Encoded Filename: secret1.txt Encoding Type : IP ID

Client sender:

sudo ./covert_tcp -source 192.168.43.11 -dest 192.168.43.12 -dest_port 80 -file secret.txt

Server receiver:

sudo ./covert_tcp -source 192.168.43.11 -dest_port -server -file received-secret.txt



Encoding\Transmission Method Two: Initial Sequence Number Field

Text file, secret2.txt has been via TCP sequence number field encoding appearing to be from port 80 on client_IP destined for port 80 on server_IP. The sequence number field serves as a perfect medium for transmitting clandestine data because of its size (a 32-bit number). In this light, there are a number of possible methods to use.

Destination Host: 192.168.43.12 Source Host : 192.168.43.11

Originating Port: random Destination Port: 80

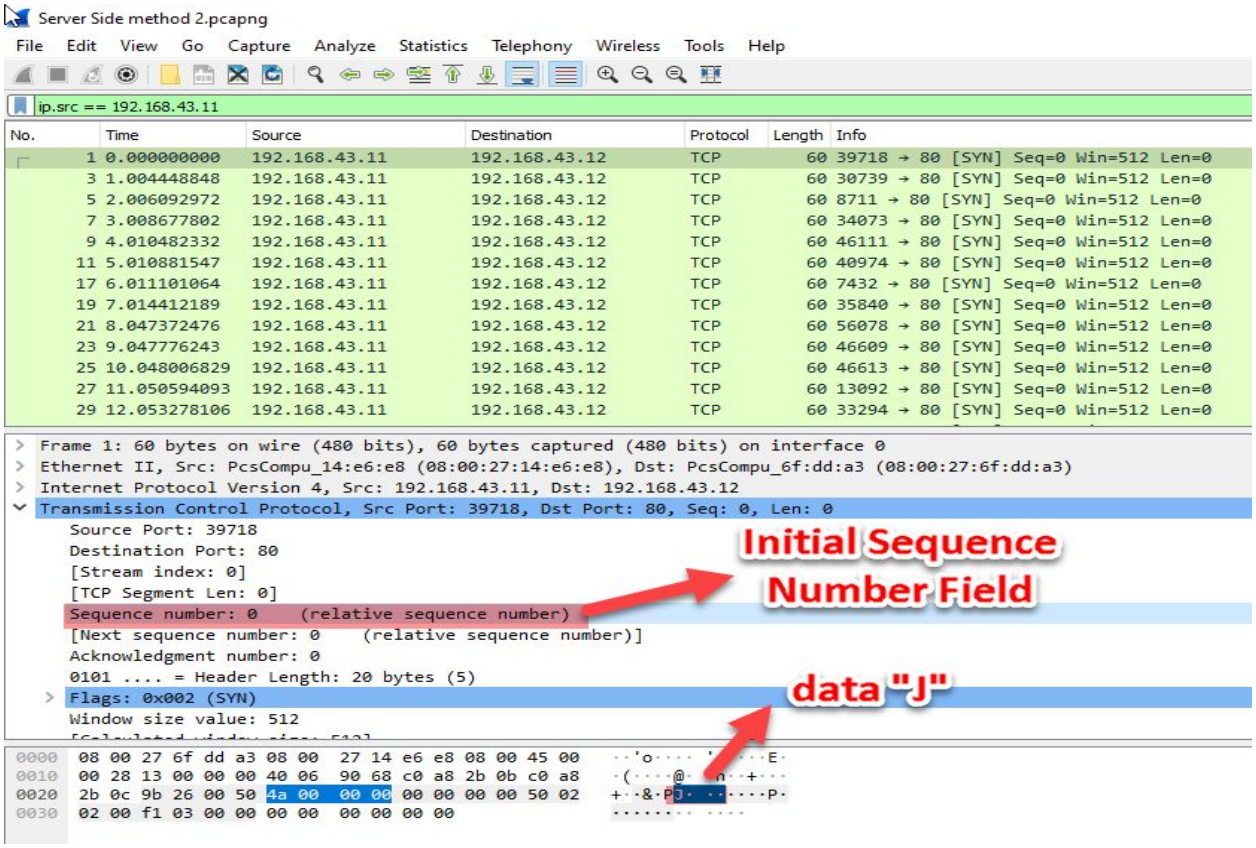
Encoded Filename: secret2.txt Encoding Type : IP Sequence Number

Client sender:

sudo ./covert_tcp -source 192.168.43.11 -dest 192.168.43.12 -dest_port 80 -seq -file secret2.txt

Server receiver:

sudo ./covert_tcp -source_port 80 -server -seq -file received-secret2.txt



Encoding\Transmission Method Three: The TCP Acknowledge Sequence Number Field "Bounce"

Text file, secret3.txt has been via TCP sequence number field encoding to be bounced of server bounce_IP and have the packet read by the destination server at server_IP. This method relies upon basic spoofing of IP addresses to enable a sending machine to "bounce" a packet of information off of a remote site and that site returns the packet to the real destination address. The source packet will appear to have come from server_IP and port 80. The return packet will go to server_IP port 80 and will be decoded by the passive server listening for any source IP talking to local port 80.

Bounce IP: 192.168.43.13 Destination Host: 192.168.43.12 Source Host: 192.168.43.11

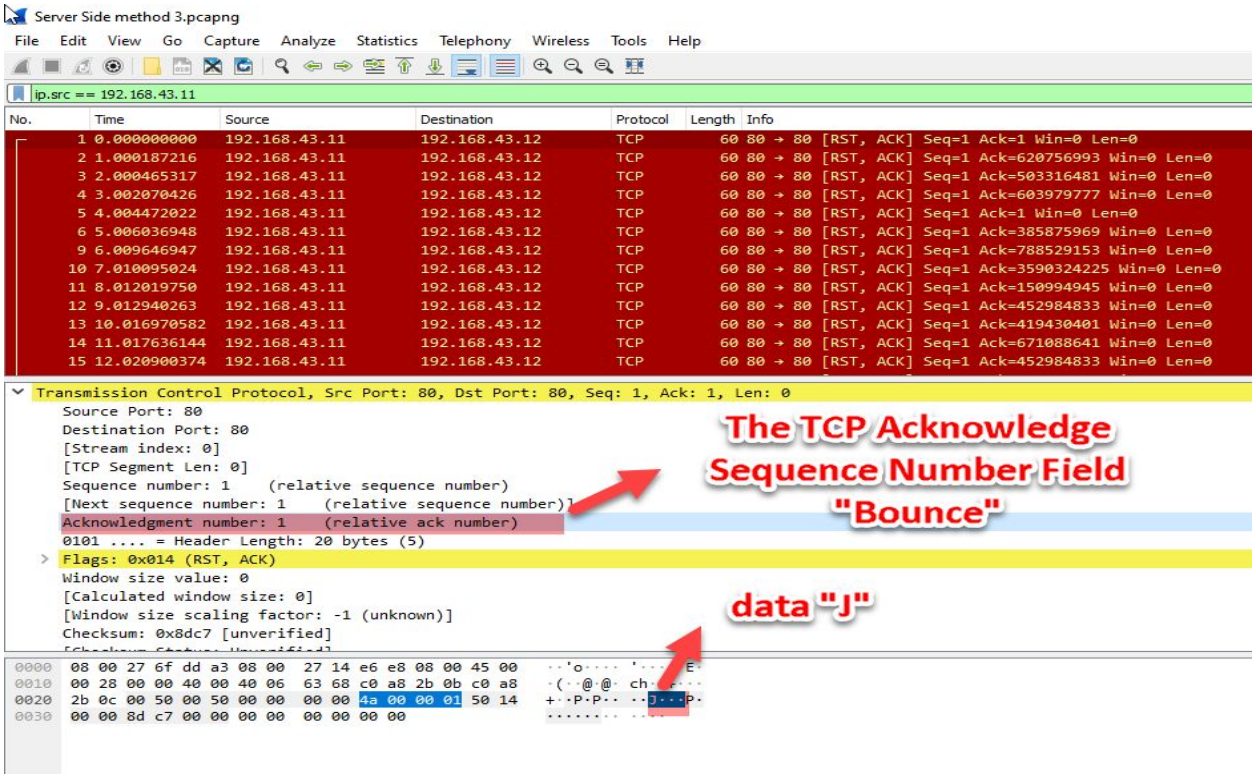
Originating Port: 80 Destination Port: 80

Encoded Filename: secret3.txt Encoding Type : IP Sequence Number

Client sender:

sudo ./covert_tcp -source 192.168.43.12 -source_port 80 -dest 192.168.43.11 -seq -file secret3.txt

Server receiver: sudo ./covert_tcp -source_port 80 -server -ack -file secret3.txt



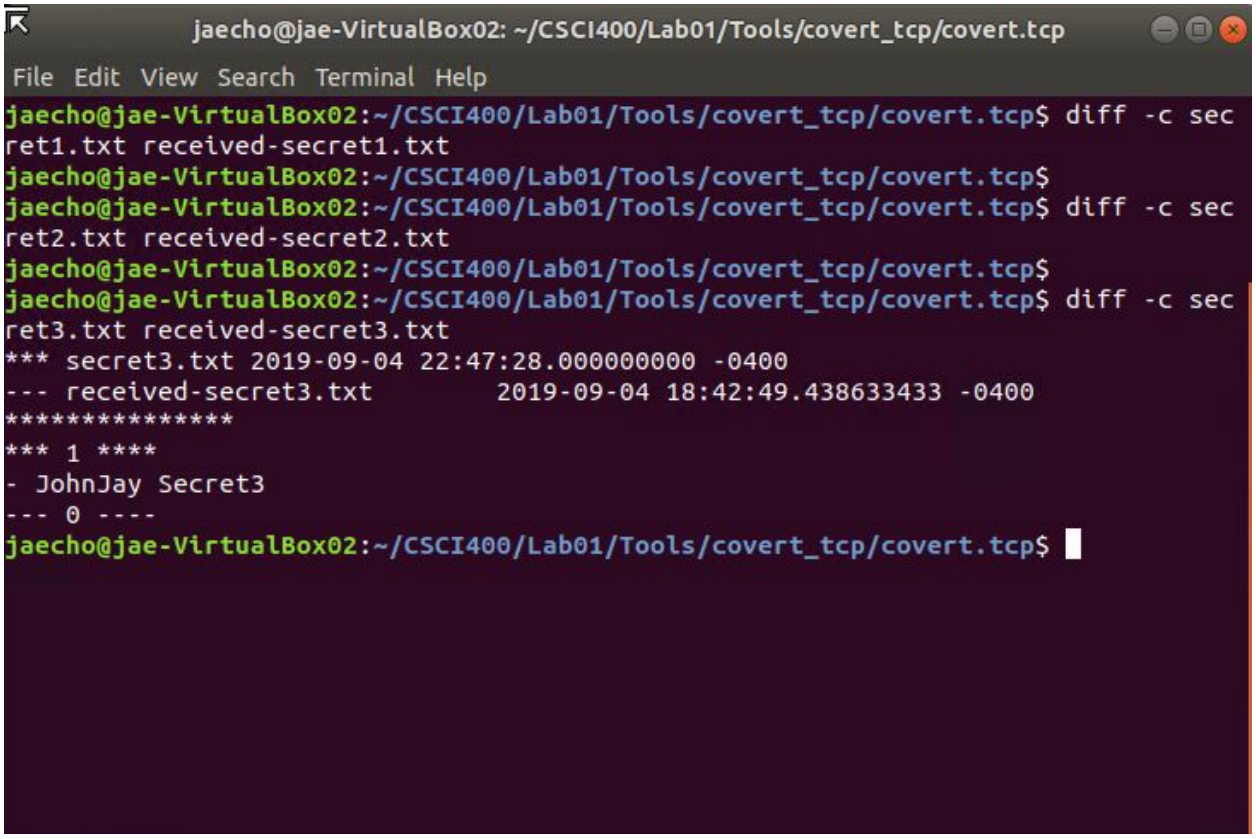
Verification of Transmitted file over network traffic with 3 different encoding method

All 3 methods were able to transmit the secret text file to server/receiver VirtualBox host as intended however method 3 did not create received-secret file as the other two did. Maybe there is some modification needed or lab setting was incorrect for transmission method 3.

As you can see from below, that the data sent and received are the same except the case of method 3.

$ diff -c secret.txt received-secret.txt

$



4. Digging deeper into the steganography tools

This final part of this lab consists of two sections: measuring the capacity of each file, that is how much can be embedded and finding possible embedding in existing files.

4.1 Capacity of the cover files

Depending on the tool in use, you will be given an estimate on how much information (size in bytes) can be embedded into the destination file. Express that in the percentage of the file in use, and specify the type of file used (image, audio, video, etc.)

(first jpg cover file):(6.6kb)

(second jpeg cover file):(2.2kb)

(third jpeg cover file):(741.1 byte)

(fourth jpg cover):(256.7 kB)

For the network traffic encoding method, there is no limit on how much data can be embedded into the network traffic as long as the sender can keep sending traffic to the receiver side.

4.2 Chaining the techniques

Try to embed more information in an already encoded file (effectively chaining the steganographic technique). Prepare a report of your observations in the format requested under Deliverables (section 6).

File capacity to small to embed another encoded file

Chaining techniques are not applicable for network traffic embedding

5. Word Problems

1. Summarize the embedding techniques used by the tools.

For image file

In terms of the embedding techniques, I used steghide. The method used to embed files such as .txt files into an image would be by using “steghide embed -ef short.txt -cf bloodhound.jpeg -sf innocent.jpeg -p candy” (“candy” being passphrase).

(FOR INNOCENT.JPEG COVER)

-Steghide used to embed short.txt file into an image.

-Order to extract short.txt hidden in image(use steghide steghide extract -sf innocent.jpeg -xf short.txt)

(passphrase: candy)

For audio file

In this part of the project, the tool Puff v2.00 was used. While using the tool it was concluded that when mp3 file(SummerCricketsChirping) was used as a cover file, the bits where actually reduced from 76.2Kb to 900bits because the tool is using padding to hide the data shrinking the file size down. With this problem happening most of the files we were trying to embed wouldn't work. So the next idea was to obtain a music file in where the data was in MB instead of KB. While making the music file as a cover file, we notice that the same action happened, in where the bits size was reduced. The files that were being embedded would still work since the embedded files bits were much lower than the cover file. So, the embedding of a jpg, wav, short text, and long text was achieved. While performing the embedding, the password thekid149163 was used. While it was known that the files were embedded, the tool puff has an option to unhide cover files. So, the password that was used to embed the files was used to see if the embedding actually worked. Once the tool unhides the file and it was shown that the hidden file was discovered. It was known that this part of the project was complete.

For text file

Text to text embedding would be not leaving any sign of changes on the stego cover. Hence when looking at the stego cover it should look exactly like the original cover to the third party. therefore in order to embed text to a text file, the stego file size should be smaller which can hide in the cover. so any text can be selected as long as they are two or more times the size of the embedded file.

For network

Encoding\Transmission Method One: Manipulation of the IP Identification Field

Text file, secret1.txt has been sent via IP Identification field encoding from client_IP to server_IP:

This encoding method simply replaces the IP identification field with the numerical ASCII representation of the character to be encoded. This allows for easy transmission to a remote host which simply reads the IP identification field and translates the encoded ASCII value to its printable counterpart.

Encoding Method Two: Initial Sequence Number Field

Text file, secret2.txt has been via TCP sequence number field encoding appearing to be from port 80 on client_IP destined for port 80 on server_IP:

The sequence number field serves as a perfect medium for transmitting clandestine data because of its size (a 32-bit number). In this light, there are a number of possible methods to use.

EncodingMethod Three: The TCP Acknowledge Sequence Number Field "Bounce"

Text file, secret3.txt has been via TCP sequence number field encoding to be bounced of server bounce_IP and have the packet read by the destination server at server_IP.

This method relies upon basic spoofing of IP addresses to enable a sending machine to "bounce" a packet of information off of a remote site and that site returns the packet to the real destination address. The source packet will appear to have come from server_IP and port 80. The return packet will go to server_IP port 80 and will be decoded by the passive server listening for any source IP talking to local port 80.

2. How would you detect the presence of steganography?

For image, audio and text file one way to detect the presence of steganography, in terms of an image would be to compare the actual image to the embedded one since the embedded one is a larger size. Otherwise, you can't really tell unless you're using a 3rd party tool.

For network

Detection of these techniques can be difficult, especially if the information being passed in the packet data is encrypted with a good software package (PGP and others). Particularly, hosts receiving a server bounced packet will have a difficult time determining where the packet originated unless they can put a sniffer on the inbound side of the bounced server, which will still only reveal that a forged packet originated from somewhere on the Internet.

3. To what extent are the embedding techniques composable?

Embedding techniques are composable to the extent to which the tool is compatible with whatever file type or method is being used.

Not applicable for network traffic embedding

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?

One way to thwart steganographic efforts is a firewall. According to Craig H. Rowland's covert.tcp document, protection from steganographic techniques include the use of an application proxy firewall system which does not allow packets from logically separate networks to pass directly to each other. Also, many traditional firewalls do have spoofing protection to thwart "The TCP Acknowledge Sequence Number Field Bounce" encoding.

6. Deliverables

A zip file containing:

The source and embedded files, including the particular technique and passphrases used if any.

(**passphrase**: **candy**) - for image embedding

(**passphrase**: **thekid149163**) - for audio embedding

Audio_cover_files.zip

Image_cover_files.zip

Network Screen Shots.zip

Network Secret files.zip

Network_WireShark_Captures.zip