Computer Networks

Semester Project Report

Muhammad Talha Paracha

Faizan Zafar

Amin ur Rashid

Project Status:

Following requirement(s) have been met:

- Server sends appropriate HTTP response messages as manually verified by a chrome extension:

#	Method	Status	Url
3	GET	200	http://localhost:55555/file1.txt
2	GET	404	http://localhost:55555/file100.txt
1	GET	400	http://localhost:55555/

- Server is able to encode any text file (from the 10 available) into an image (randomly chosen from the 5 24-bit PNGs available)
- Server is able to serve the client with the encoded image
- Client is able to open the image into any image editing software
- There are no visual differences between the encoded image and the original image
- An independent script is provided for anyone to decode the encoded image

Following requirement(s) have not been met:

- Client's ability to see the decoded text in the browser.

Actually this requirement cannot be met within the scope of our project. That's because for a client to see the decoded text in a browser, we need to provide him with a JS script or a browser extension. But if we did so, then anyone would be able to see our decoding logic and would thus figure out our encoding logic, which would diminish security. The only correct way would be to modify the browser itself, or to create a browser of our own with the decoding logic compiled into it.

Project Performance:

- In one benchmarking test, encoding a complete book of 492KB into an image of 3.2MB took around 7 seconds and the decoding took around 6 seconds.
- It has been ensured that the algorithm is able to encode all ASCII characters. And during testing, various text files of various sizes have been encoded into several distinct images to ensure a bug free program.
- **To improve the algorithm performance, bit hacks have been employed in all three cases of fetching/setting/clearing the LSB of a byte.**
- It has been ensured that every byte of an image is encoded with one bit of data. So, three pixels of an image are being used for encoding one byte of data.
- A regular expression is used for fetching the number of file requested by client

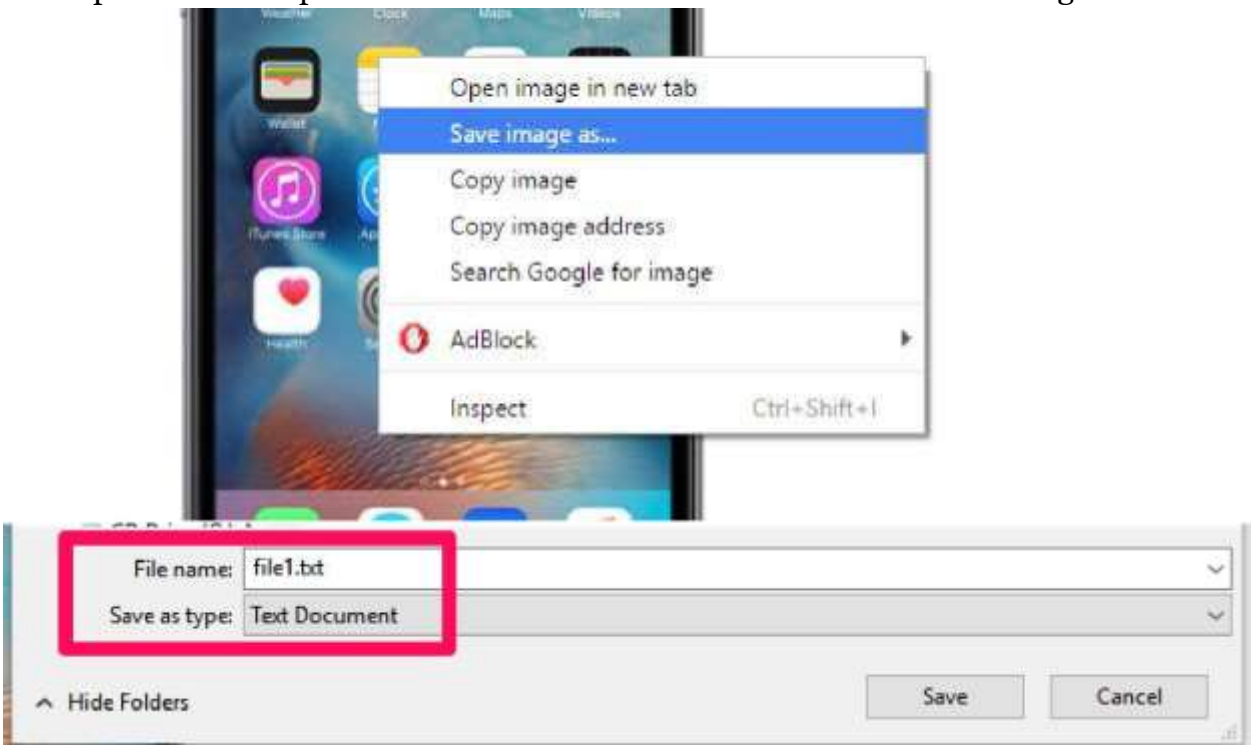
- The binary of text to be encoded is terminated with bits 10000000000... so the decoder knows when to stop. It has been ensured that this technique works well. In fact, a sequence like 00000000000... would've caused severe bugs in the program. But we've started our sequence with a 1 and have then proceeded it with several dozen zeros.

Project Dependencies:

The project has been built using Python 3.5. Other external libraries used in the project are: Socket, Sys, RE, OS, Random, Binascii & Pillow.

Project Limitations:

- Since the client requests a txt file, the browser assumes that the response will be a txt file. Now the response from our server does contain "Content-type: Image/PNG" to tell the browser of correct MIME type. But still the earlier assumption causes a problem when the client tries to save the encoded image:



As it can be seen, the client must manually specify the file type every time he tries to save the encoded image. This is bad from a usability point of view.

One work-around for the problem would be to redirect the client from localhost:55555/**file1.txt** to localhost:55555/**file1.png** via HTTP headers and then serve the client with the encoded image.

- The other project limitation is the inability of client to see decoded text on the browser, but it has already been documented above.