Troy Flagg
Project Milestone
4/14/19

StegHide On Encrypted Files.

Abstract

This paper describes the approach taken for encrypting specific text files with AES and then using an open source library called Steghide to hide the encrypted files within images. This will allow for a hidden transmission of files without any reduction in image quality. A second user who also possesses the AES key will be able to scrape the web page the initial user hosts and then extract the encrypted file which they would then be able to de-crypt the file.

Introduction

The motivation behind this is to create a proof of concept for hidden transmission of messages while also encrypting those messages. This allows me to explore three different fields through one project, web hosting and scraping, steganography, and AES. Also this allows me to explore different methods of communication over the internet.

Background

This project will take very strongly from open source projects such as Steghide. This is a package that facilitates steganography. Also I will be taking very heavily from AES documentation. The main reason for this project is to explore other forms of communication without others being aware of the two parties communicating. Even if their conversation is intercepted the contents of their chat will still be kept secret through AES encryption.

Methodology

It seemed ideal to split the work into four distinct parts. Each of those parts then had to be divided into two different parts, one for the server to encrypt, encode and host and one for the client to decrypt, decode and extract data.

The first of those parts is web hosting. On the server side I used pythons simple servers to host the images which I hid data away in. In tandem with this I used pythons built in HTTP methods to scrape the web page and grab those images.

The second part is the steganography piece. For this I created a wrapper class which will make all of the system calls to preform the steghide calls. On the server side I find out how large the image to hide was then determine how much data each picture that I plan to host can hold. If the image to hide can’t

fit within one picture I will break it apart into multiple files to hide in multiple images. On the server side, I scrape all of the images off the web page and iterate through them to grab all of the associated file parts. If there are more than one file part I will then re-join them into one file.

The third piece is the actual encryption part which happens just before the steganography piece. On the server side, before I check to split the file I run AES encryption algorithms on the entire file. Similarly on the client side, once the steganography piece is done and the files are re-joined together I will run the AES decryption algorithms to bring back the original file I hid.

Finally there is the main driver. This is where you will bring in the proper authentication keys and passwords, such as the AES key, the AES initial vector and the steghide password. From this point it will provide instructions to choose if you are the client or the server to host files or scrape the hosted server for pictures.

Experiments

The vast majority of experimentation that came with this project was using Steghide. I had to preform many different calls in the package to figure out which were needed. I then had to learn how to interact with steghide through the OS library in python.

I also had to research the cleanest web server to use to host the images. This involved me debating between creating a JS or flask front-end to interact with. I ultimately decided on functionality over visuals and just used a simple web server.

Finally I had no experience in breaking files apart and reassembling them. This involved a lot of trial and error and figuring out best practices for dealing with files when they are in byte forms.

Discussion/Analysis

As a consequence of dealing with files only in their byte forms I was able to encrypt and share files that went beyond the base of text files. This of course required multiple pictures to store them in. Similarly although untested I am sure we could store data into files other than pictures.

Conclusion

The end goal to discretely share secure files has been accomplished. The next step would be to add in a nice front-end where you can upload files rather than run everything through command line. With a front-end this opens up the ability to create a chat program between users where they use an image sharing site to upload their photos to communicate with one-another through files hidden in the images.

References

Steghide: http://steghide.sourceforge.net/

Python documentation: https://docs.python.org/3/tutorial/

Stack overflow: https://stackoverflow.com/