CovertFS Documentation

Release .7

Kyle Gorak, Adam Sjoholm, David Hart, Ryne Flores

January 25, 2016

Contents

About the Project

Covert File System is a web-based application that allows users to covertly share files through social media sites while maintaining plausible deniability for both the user(s) and the social media site.

We created this project for a Capstone class for Computer Science at the United States Military Academy, West Point. This project was broken up into the following sprints over the course of one year.

1.1 Sprint 1: Knowledge acquisition

Sprint broken into three sub-goals:

1. Implement basic steg module to encode and decode an image in Python
   - `$ python3 Image_Manipulation/lsbsteg.py [encode/decode] -i [image_path] -m '[message]'`
   - encode saves a copy of the image_path with a '_1' appended encoded with the message
   - decode prints the encoded message in the image_path
2. Determine a suitable social media site that meets our requirements (anonymous user upload, no or lossless compression)
   - **social media sites investigated:**
     - SendSpace
     - Whisper
     - Flickr
     - Yogile
3. Design and implement basic upload application in Python for the selected social media site
   - `$ python3 Web_Connection/api_cons.py`
   - returns the download url for the uploaded random cat image (stores the delete URL as well)

1.2 Sprint 2: Covert Mapping Structure

Sprint broken into four sub-goals

1. Design the map structure for the covert file system to allow maximum flexibility and usability.

2. Break a large message file into parts to encode across multiple images.
   - Analysis of how much data can be encoded using LSB
   - Determine file system overhead in each image
3. Begin to add API connection and Encode/Decoder into Application.
   - `$ python3 main.py [url]` [1] // url can be the full url path or the file id (6 character ending, i.e xvdmcn)
   - `covertFS$ [command]`
4. Functional Design Documents
5. From previous Sprint:
   - Keep all images in memory
   - Error handling in API connection
   - Enforce restrictions on arguments in encode/decode

1.3 Sprint 3: Beta release

Basic stand-alone application to encode/decode a local covert file-system that is able to store, open, and delete files from the covert file-system. Command line program will work similar to a unix based directory system. Using these commands will require breaking the file structure across multiple encoded images. Everything is seamless to the user who only needs to keep track of the /root image URL and then navigate the file system with ease.

1.4 Sprint 4: Publication start and alpha release

Sprint broken down into 5 sub-goals:

1. Basic draft of paper for publication using LaTEX.
2. Create a backlog of things required to implement covertFS into a live operating system such as Tails.
3. Publish documentation using apidocs.
4. Create a FUSE module for covertFS.
5. Change steg technique to allow storage of larger files with dynamic sizes.

[1] Optional parameter

CHAPTER 2

Setup

- Clone the project from GitHub `git clone https://github.com/gorhack/covertFS.git`

Covert File System is written exclusively in Python 3 due to the vast modules and libraries that support our goals. Currently *covertFS* only supports using *sendspace* for upload on the web.

- **Dependencies:**
  - python3, pip3 `$ pip3 install -r utls/requirements.txt`
  - Get a sendspace API key here.
  - Copy your sendspace API key and create a file in /Web_Connection/API_Keys/ containing `sendSpaceKey='API KEY GOES HERE'`

Usage

- `$ python3 main.py [url of folder/root]` [1]
- `covertFS$ [command]` basic application usage.

Documented Commands:

- `newfs` uploads the old fs and returns the url. loads a new covert file system.
- `loadfs [url]` load a covert file system
- `ls [path]` [1] list directory contents
- `cd [path]` change directory in the covert file system to the path
- `cat [file]` concatenate and print files
- `upload [local path] [covert path]` upload a file to the covert file system
- `rm [path]` remove a file from the covert file system
- `mkfile [name] [text] [path]` create a text file in the covert file system at the path
- `mkdir [path]` make directories in the covert file system at the given path
- `rmdir [path]` remove directories in the covert file system at the given path
- `download [covert path] [local path]` download a file on the covert file system to disk
- `uploadfs [url]` save the covert file system, returns URL to the root image. To load the same file system this URL must be retained.
- `encodeimage [msg]` encode an image with a message, returns the URL to the image
- `decodeimage [msg]` decode an image, returns the message
- `hist` show the history of previous commands
- `shell [cmd]` run shell commands
- `help [cmd]` [1] show list of commands or documentation for a specific command
- `exit` exit the covert file system
- `proxy / noproxy` turns the built in proxy on/off respectively

[1] Optional parameter
[1] Optional parameter
[1] Optional parameter

CHAPTER 4

Source Documentation

4.1 Command line main module

covertFS is a command line based program created using the *cmd* module. The *main.py* file contains all commands available and additional helper functions.

class `main.`**`Console`**

Bases: `cmd.Cmd`, `object`

`addfiletofs`(*path*, *contents*)

Helper function to add a file to the fs.

`completedefault`(*text*, *line*, *begidx*, *endidx*)

Allow Tab autocompletion of file names.

`default`(*line*)

Called on an input line when the command prefix is not recognized. In that case we execute the line as Python code.

`do_EOF`(*args*)

Exit on system end of file character

`do_cat`(*args*)

View the contents of a file in the file system.

Use: cat [covert path]

`do_cd`(*args*)

Change directory to specified [path]

Use: cd [path]*

`do_createdownloadlink`(*url*)

Create a direct download link from a url.

`do_decodeimage`(*url*)

Decode the message in an image.

Returns the message in plain text.

decodeimage [download url]

`do_download`(*args*)

Download a covert file to the local file system.

Use: download [covert path] [local path]

do_encodeimage(*msg*)
Encode a message to an image and upload to social media.

Returns the url.

Use: encodeimage [message]

do_exit(*args*)
Exits from the console

do_help(*args*)

Get help on commands 'help' or '?' with no arguments prints the list of commands 'help <command>' or '? <command>' gives help on <command>

do_hist(*args*)
Print a list of commands that have been entered

do_loadfs(*url*)
Load a covert file system.

Use: loadfs [url]

do_ls(*args*)
List items in directory

Use: ls [path]*

do_mkdir(*args*)
Make a folder at the given path.

Use: mkdir [path]

do_mkfile(*args*)
Create a text file with a message to the file system.

Use: mkfile [path] [message]

do_newfs(*args*)
Create a covert file system, return the URL of the old fs.

Use: newfs

do_noproxy(*args*)
Turns off the built-in proxy.

Use: noproxy

do_proxy(*args*)
Turns on the built-in proxy.

Use: proxy

do_rm(*args*)
Remove a file from the covert file system.

Use: rm [path]*

do_rmdir(*args*)
Remove a folder in the current directory.

Use: rmdir [path]

do_shell(*args*)
Pass command to a system shell when line begins with '!'

do_upload(*args*)
Upload a local file to the covert file system.

Use: upload [local path] [covert path]

do_uploadfs(*args*)
Upload covert fileSystem to the web

down_and_set_file(*filename*)
Download a file. Put it in the filesystem.

emptyline()
Do nothing on empty input line

loadfs(*url=None*)
Load the filesystem from a URL: Download pic, decode it, then send the string to the load function in fsClass.

postcmd(*stop*, *line*)
If you want to stop the console, return something that evaluates to true. If you want to do some post command processing, do it here.

postloop()
Take care of any unfinished business. Despite the claims in the Cmd documentaion, Cmd.postloop() is not a stub.

precmd(*line*)
This method is called after the line has been input but before it has been interpreted. If you want to modifdy the input line before execution (for example, variable substitution) do it here.

preloop()
Initialization before prompting user for commands. Despite the claims in the Cmd documentaion, Cmd.preloop() is not a stub.

san_file(*file_contents*)
Sanitize file before 1)viewing contents or 2)putting on host OS

uploadfile(*contents*)
Helper function to upload file, return the download url.

4.2 File_System package

4.2.1 Submodules

File_System.fsClass module

The *fsClass* module extends the *pyfilesystem* package, using the *MemoryFS* file system. The MemoryFS file system stores all directory and file info in main memory, to allow for instantaneous file access as well as to avoid writing any FS information to disk. This allows for plausible deniability.

class `File_System.fsClass.`**CovertFile**(*path*, *memory_fs*, *mem_file*, *mode*, *lock*)
Bases: `fs.memoryfs.MemoryFile`

class `File_System.fsClass.`**CovertFilesystem**(*url=None*)
Bases: `fs.memoryfs.MemoryFS`

The CovertFilesystem class is the FS object used in main.py. It is a subclass of MemoryFS, and it has methods within it that utilize (but do not extend) methods from the superclass.

addfile (*path*, *contents*)
Add a file, with given contents, to given path. Error if path is a file, directory, or if parent directory is not present.

cd (*path='/'*)
Changes current directory. Superclass has no concept of current directory (all calls are made from root dir), so this method is purely local. Error if given path does not exist, or is a file.

check_parent_dir (*path*)
Checks to ensure parent directory is present before attempting to add a file to it.

loadfs (*fsstring*)
Iterates through a string that represents the filesystem (either pulled from online, or given as a test string), makes necessary directories, and creates necessary files (empty for now) that are then loaded by main.py.

ls (*path=None*)
Returns a list of the files and directories in the given path, or the current directory if no path is given. Error if given path does not exist, or is a file.

mkdir (*path*)
Makes a new directory at given path. Error if path is a directory already, or a file.

rm (*path*)
Removes file at given path. Error if no such file.

rmdir (*path=None*, *force=False*)
Removes empty directory at given path, or non-empty directory if force option is given. Error if path is not a directory.

sanitize_path (*path=None*)
This method takes a user-input path and makes it method-readable, by adding the current path to the front (if the desired path doesn't start with /) or returning the root path if no path is given.

save ()
Turns the entire filesystem into a string to be uploaded. Returns that string.

4.3 Image_Manipulation package

4.3.1 Submodules

stegByteStream module

Least-significant bit stegenographic technique based on Adrian-George Bostan's implementation on GitHub. This technique embeds text messages into images using the Least Significant Bit steganographic algorithm.

The basic idea of the algorithm is to take each individual bit of the message and set it as the least significant bit of each component of each pixel of the image. Usually, a pixel has Red, Green, Blue components and sometimes an Alpha component. Because the values of these components change very little if the least significant bit is changed, the color difference is not particularly noticeable, if at all.

class `Image_Manipulation.stegByteStream.`**Steg** (*proxy*)
Bases: `object`

checkImageIntegrity (*msg*, *img*)
The checkImageIntegrity method checks to see if a message has been properly encoded into an image. This method takes a message as string, and an image as a BytesIO object. This method returns a boolean.

decode (*img*)

decodeImageFromURL(*url*)
The decodeImageFromURL method retrieves an image from a url, and extracts a message from the image. The image needs to have been encoded using the stegByteStream.encode(msg) method. This method takes a url as a string. This method returns the decoded message as a string.

encode(*msg*)
The encode method use the least significant bit stegonography technique to encode a message into an image. This method takes a message as a string. This method returns an image as a BytesIO object.

`Image_Manipulation.stegByteStream.`**testSteg**(*testNum*, *url*, *newImageName*, *message*, *predicted*)
The testSteg function tests the least significant bit stegonography technique.

genImage module

The *genImage* module returns an image on request as a *BytesIO* object.

`Image_Manipulation.genImage.`**genCatImage**()
The genCatImage function returns an image from The Cat API. This function does not take any parameters. This function returns a BytesIO object.

4.4 Web_Connection package

4.4.1 Submodules

Web_Connection.api_cons module

The *api_cons* module creates an anonymous connection to a given social media file hosting website and provides connection, upload image, and download image parameters.

class `Web_Connection.api_cons.`**SendSpace**(*key*, *proxy*)
Bases: `object`

The SendSpace class is creates a connection to the SendSpace file sharing website.

connect()
The connect method creates an anonymous connection to SendSpace. The connection uses the sendspace API (1.1) and does not require authentication or login. This method requires no parameters. This method returns the URL and the extra info needed for uploading an image to SendSpace.

downloadImage(*file_id*)
The downloadImage method retrieves the direct download URL from the download url returned by SendSpace. This method take the download url returned by uploadImage as a parameter. This method returns the direct download url.

parseXML(*xml*)
This method parses XML data with the BeautifulSoup module.

sendspace_url = 'http://api.sendspace.com/rest/'

upload(*img*)
The upload method uploads an image to SendSpace. This method takes an image as a BytesIO object. This method returns the download and delete urls as a tuple.

uploadImage(*upl_url*, *max_size*, *upl_id*, *upl_extra_info*, *img*)
The uploadImage method sends an image file for upload to SendSpace. This method requires the upload

url and extra info from the SendSpace connection and the image (BytesIO object) as parameters. This method returns the direct download URL and delete URL of the image as a tuple.

Web_Connection.proxy_list module

The list of possible https and http proxies to use with *sendspace*. Proxies are necessary on the DREN at USMA. The DREN blocks many file-sharing websites, such as *sendspace*.

`Web_Connection.proxy_list.`**`proxy`**`()`

Proxy without SSL.

`Web_Connection.proxy_list.`**`ssl_proxy`**`()`

The SSL proxy is a free US proxy https://165.139.149.169:3128.

Index