

Automatic Detection of Steganographic Content

Author: James Taylor

Supervisor: Michael Dailey

Moderator: Irena Spasic



Student number: C1129788

Module code: CM3203

Module title: One Semester Individual Project

Credits available: 40

Acknowledgements

I would to express my thanks to Mike Dailey for supervising my project, providing flexible meeting times and helpful advice when needed. I would also like to thank my girlfriend Marta, for proof-reading and putting up with me during this stressful time.

Abstract

There exists a fairly large gap in the market for a unified solution to the problem of steganalysis, as most tools are extremely user-unfriendly and exist for a single purpose only. This project has successfully created a piece of software that fills this gap. It can detect least significant bit, SpamMimic and alternate data stream based steganography. It is also open-source and extensible, with a database to store the MD5 hashes of previously detected files.

Keywords: steganography, steganalysis, automatic, security, forensics

Table of Contents

1	Introduction.....	10
1.1	Steganography.....	10
1.2	Steganalysis.....	11
1.3	Intended project audience and beneficiaries.....	12
1.4	Project scope.....	13
1.5	Project aims and objectives.....	14
2	Background.....	15
2.1	Problem identification.....	15
2.2	Likely stakeholders.....	16
2.3	Market research of similar existing solutions.....	17
2.3.1	StegDetect.....	17
2.3.2	StegSecret.....	17
2.3.3	VSL (Virtual Steganographic Laboratory).....	17
2.4	Potential tools.....	18
2.5	Detailed project outline.....	18
3	Specification and Design.....	20
3.1	Business model.....	20
3.1.1	Functional requirements.....	20
3.1.2	Non-functional requirements.....	22

3.2	Initial UI mock-ups	23
3.3	Data flow.....	28
3.3.1	Use case diagrams.....	28
3.3.2	Activity diagrams.....	30
3.4	Class diagrams	32
3.5	Final UI designs	35
3.5.1	Changes from initial mock-ups.....	38
3.5.2	Design ideas	38
4	Implementation	39
4.1	Methodology.....	39
4.1.1	Agile.....	39
4.2	Libraries and Tools.....	40
4.2.1	External libraries.....	40
4.2.2	Internal libraries	41
4.3	Steganalysis methods	42
4.3.1	Least significant bit embedding.....	42
4.3.2	SpamMimic.....	49
4.3.3	Alternate data streams.....	51
4.4	Plugin architecture	51
4.5	Database.....	52

4.6	Problems encountered.....	53
5	Results and Evaluation.....	55
5.1	Functionality testing.....	55
5.1.1	LSB detection within images.....	55
5.1.2	LSB detection in audio files.....	57
5.1.3	SpamMimic detection.....	57
5.1.4	ADS detection.....	58
5.1.5	Click-through testing.....	60
5.2	Usability testing.....	70
5.3	Evaluation of system requirements.....	72
5.3.1	Functional requirements.....	72
5.3.2	Non-functional requirements.....	73
6	Future Work.....	74
7	Conclusion.....	77
8	Reflection on learning.....	79
9	References.....	81

Table of figures

Figure 1- Percentage of steganographic software tools that hide data in various digital media...	12
Figure 2 – UI title page mock-up.....	24
Figure 3 – UI new case page mock-up.....	24
Figure 4 – UI load case page mock-up	25
Figure 5 – UI analysis page mock-up	26
Figure 6 – UI results page mock-up.....	26
Figure 7 – UI plugins page mock-up	27
Figure 8 – UI help page mock-up	27
Figure 9 - Create new case (use case).....	28
Figure 10 - Load existing case (use case)	28
Figure 11 - Manage plugins (use case)	29
Figure 12 - View help (use case).....	29
Figure 13 - Create new case (activity diagram)	30
Figure 14 - Load existing case (activity diagram)	31
Figure 15 - Manage plugins (activity diagram)	31
Figure 16 - View help (activity diagram).....	32
Figure 17 - Class diagram for steganalysis attacks.....	33
Figure 18 - Class diagram for the plugin architecture	33
Figure 19 - Class diagram for the results presentation	34
Figure 20 - Class diagram for the GUI	34
Figure 21 - Final welcome screen.....	35
Figure 22 - Final new case screen.....	36

Figure 23 - Final analysis screen	36
Figure 24 - Final manage plugins screen	37
Figure 25 - Final results screen.....	37
Figure 26 - Agile methodology.....	40
Figure 27 – From left to right, cover file, recovered image (2 nd bit-plane), recovered image (1 st bit-plane or LSB)	43
Figure 28 - Left to right, original image, 5kb random data embedded, embedded poem "if", section of 3 rd image.....	44
Figure 29 - Quadratic equation from SPA attack	47
Figure 30 - False-alarm boundary for SPA detections in the red band	48
Figure 31 - Plot to show SpamMimic cluster on the right and normal documents on the left	50
Figure 32 - Code snippet to classify unknown document as SpamMimic or not	51
Figure 33 - Graph to show the performance of LSB detector in images	56
Figure 34 - Graph to show the performance of the audio LSB detector.....	57
Figure 35 - Graph to show the performance of the SpamMimic detector	58
Figure 36 - DOS commands for creating an ADS	59
Figure 37 – Graph to show the performance of the ADS detector.....	60
Figure 38 - Welcome screen.....	61
Figure 39 - New case screen (with text)	62
Figure 40 - New case error message	62
Figure 41 - Blank analysis screen	63
Figure 42 - Start analysis error message	63
Figure 43 - Browse files error message	64

Figure 44 - Analysis screen with content loaded	64
Figure 45 - Items detected from the database	65
Figure 46 - Analysis screen, after analysis has completed.....	65
Figure 47 - Visual attack success message.....	66
Figure 48 – 1 st and 2 nd bit-planes of the stego images	66
Figure 49 - Results message	67
Figure 50 - Results screen.....	67
Figure 51 - Detailed program output in the results file.....	67
Figure 52 - Default mail client launch	68
Figure 53 - Report bug email.....	68
Figure 54 - Help dialogue	69
Figure 55 - Plugins page	70

1 Introduction

This section will give the reader the necessary knowledge to better understand the project, as well as clarify the scope and aims of the project. In order to properly introduce the project, and its goal of automated steganalysis, it is first necessary to introduce the concept of steganography.

1.1 Steganography

Steganography is the art and science of hiding information. It comes from the Greek words '*steganos*' and '*graphein*' which, when translated, literally mean '*covered writing*' (Johnson, et al., 2001). The purpose of steganography is to conceal the very existence of a message or data from a third party. This approach differs from cryptography, which aims to make a communication unreadable but does not attempt to hide that the communication took place. Steganography operates on the assumption of security-through-obscurity by concealing a message within some inconspicuous cover medium, making it inherently difficult to detect. However, unlike cryptography, once the existence of steganographic content within a file is known, it is often relatively easy to extract as long as the tool or algorithm used to hide the message is also known.

Furthermore, steganography is not a new technique to conceal the existence of a message. One of the first recorded uses of steganography was in fact in ancient Greece, where messages had been known to be tattooed onto the shaven heads of slaves. The sender (the slave's master) would then wait for the slave's hair to grow back, effectively concealing the message. The slave would then be sent to deliver the message. In the instance that a slave was captured, unless the captor knew exactly where to look, the message would not be found (Siper, et al., 2005).

There are many more examples of steganography throughout history. For example, in WWII the German espionage community used microdots (essentially an extremely small photograph of some text, normally around 1mm across) embedded in paper and then covered in adhesive to transmit messages amongst each other (Siper, et al., 2005). Another example is that in 1966 Jeremiah Denton blinked his eyes in Morse Code during a televised press conference, which he was forced into when he was an American prisoner of war in a North Vietnamese POW camp, to spell out the word 'torture' (The U.S. National Archives and Records Administration, 2016). Although steganography has been used for hundreds, if not thousands of years, the digital revolution and the creation of new digital mediums has opened up new possibilities in the field.

The science of detecting and attacking steganography is called steganalysis. Its primary objective is to determine if a suspect message or file contains steganographic content, and where possible, extract that content.

1.2 Steganalysis

There are many ways to attempt to perform steganalysis on a file. Traditional methods carried out by forensic technicians often revolve around looking for tell-tale markers of steganography manually. For example, discovering a deleted file that is exactly the same as the suspect file in question but smaller in size would imply the original file was modified and then deleted to hide evidence (Richer, 2003). Another method described by (Richer, 2003) involves visual detection of anomalies within image files, looking for small distortions to reveal hidden information. However, when dealing with large sets of files, this type of analysis becomes tiresome and difficult. This is where statistical analysis comes in.

Statistical steganalysis is a relatively new academic discipline with few articles appearing before the late 90's and most of the literature focusing on steganographic content in images, as this

is the most prominent type of cover media in steganography. *Figure 1* shows the abundance of tools available in image steganography, out of a total of 3011 tools tested (Fridrich, 2010).

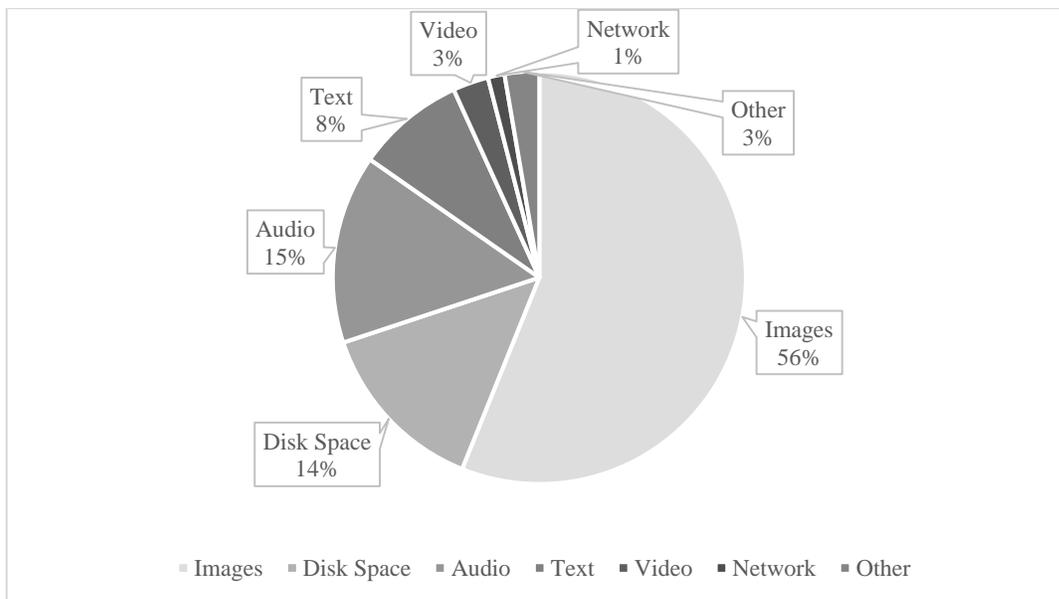


Figure 1- Percentage of steganographic software tools that hide data in various digital media

1.3 Intended project audience and beneficiaries

The system is intended to be used in a digital forensic environment to assist with forensic investigations in order to avoid the problem of seemingly innocuous files slipping through-the-cracks and crucial evidence being missed. This is especially relevant now, since a recent amendment to the RIPA act part three can force individuals to reveal the ‘key’ to encrypted files during an investigation or face imprisonment (The National Archives, 2000).

This could mean that criminals looking to keep their communications secret could start turning more and more to steganography, as it could be argued that if content is first encrypted and then embedded into a cover file, it would be difficult for a forensic analyst to say that without a doubt that *encrypted* content is hidden within a file and not just random data. It could also mean that the encrypted content goes unnoticed altogether.

1.4 Project scope

It is difficult to clearly define the scope of the project before actually implementing it, as the aim of the project is to produce a system that is able to detect as many forms of steganography as is possible in the given timeframe. However, some methods require more research and implementation time than others. Originally, the aim as stated in the initial report was to implement between five and seven methods of steganalysis. This seemed reasonable, but after much research I believe that this will be unachievable in the given timeframe. I also believe that the system should focus on the most prevalent and accessible forms of steganography and since the vast majority of tools focus on images, text, audio and the file system as cover media, I will reduce the scope of the project to focus on implementing at least one method of steganalysis that targets each of these areas.

There are two main categories of steganalysis algorithms; blind and targeted. Blind steganalysis often focuses on collecting large amounts of steganographic and natural files, and then using an SVM classifier to determine whether an unseen file is steganographic or not. This method is extremely flawed however, as it requires extremely large amounts of training data to be a variable choice. Hence, this project will focus on researching, collecting and potentially creating targeted attacks only.

There are multiple categories of targeted attacks, some of which are easier to carry out than others. The categories are as follows: stego-only attack, known cover attack, known message attack, chosen-stego attack, chosen message attack, known-stego attack (Kessler, 2004). This project will focus on the first category of steganalysis attacks, the stego-only attack, in which only the stego-object is available for analysis. This is the most difficult category of attack since it often requires

statistical analysis. This is due to the fact that only the stego-object is available for analysis and one first has to first determine if the object in question is steganographic.

1.5 Project aims and objectives

This project aims to produce a software system that will focus solely on steganalysis, applying various targeted attacks across a multitude of file types to effectively determine if that file contains a specific type of steganography. If this is the case, the system will attempt to automatically retrieve the hidden content where possible.

2 Background

This section will describe the problem this project aims to address in greater detail. It will show that there is a need for a unified solution to the problem of steganalysis and identify likely stakeholders. It will also suggest tools that will likely be of use to complete the project.

2.1 Problem identification

Since the digital revolution, steganography has become increasingly ever more complex and even harder to detect. One of the most common forms of steganography in the digital age is some variation of the least significant bit method (LSB), normally applied to images but sometimes to audio and even video files. This is probably because it is the simplest type of steganography to implement. The method involves encoding a message into the lowest or least significant bits of some digital media, which allows a message to be sent while having minimal effect on the digital media in question. If the user has been clever about their actions, LSB steganography can easily be missed during a forensic investigation as long as the investigator is not directly looking for the existence of LSB steganography.

However, there now exist some forms of steganography that are ‘steganalysis aware’ (Orsdemir, et al., 2008) meaning that they attempt to mask the statistical impact they have on a file, making these methods difficult to detect even if the investigator suspects steganographic content exists. The main problem this project aims to address is that there are now thousands of relatively easy to use, free steganography programs that allow the user to mask information in various different cover media but very few *true* steganalysis programs. In a survey of one hundred and eleven different programs for both steganography and steganalysis, it was found that there

were one hundred and three programs for steganography but only eight for steganalysis (Hayati, et al., 2008).

Out of those eight tools, three are designed to destroy any message concealed, not detect or retrieve it, which could be useful in law enforcement; for example, eliminating terrorist communications, but not particularly in forensics. Out of the remaining five, three are specifically designed to deal solely with images, one for real time packet analysis to detect IP based steganography and one that can detect both audio and image steganography. However, most of these tools use dictionary attacks to try to guess the key in the headers of certain files that employ JPEG specific forms of steganography, which often produce bad results if a complex passphrase is chosen. Only StegDetect uses statistical attacks to determine if a file is steganographic or not and that focuses solely on images. This shows that there is a market for an all-in-one steganalysis system that provides a user-friendly interface and can combat multiple forms of steganography.

2.2 Likely stakeholders

Parties that would likely be interested in a software such as the one this project aims to create would be either digital forensics companies or law enforcement agencies looking to determine whether suspicious files contain hidden information. It also seems that some government agencies and militaries are taking an interest in steganography and steganalysis due to the increased threat of terrorism. In the extract below *“Technology and Terrorism”*, David Clarke says:

“The United States military also maintains a keen interest in research into new steganography software and applications and the development of new steganalysis tools”

(Clarke, 2004)

It could be argued that this is also backed up by Wired magazine who report that Neil Johnson, a steganalysis researcher at George Mason University, has had his work underwritten by the National Security Agency (McCullagh, 2001).

2.3 Market research of similar existing solutions

The existing solutions relevant to the problem area are unsuitable or insufficient in this particular case. This is due to the fact that they all focus solely on images and the vast majority of the systems are all command line driven tools as opposed to full programs. StegSecret and VSL both have GUI's but they are far from user friendly.

2.3.1 StegDetect

StegDetect is a command line steganalysis tool that uses statistical attacks on known JPEG steganographic schemes, all in the spatial domain other than AppendX which operates on end of file (EOF) steganography. It can detect the presence of embedding with software such as JSteg, JPHide, Invisible Secrets, OutGuess 01.3b, F5 (header analysis), AppendX and Camouflage.

Available at <http://www.outguess.org/detection.php>

2.3.2 StegSecret

StegSecret seems to be almost identical in every way to the above mentioned tool StegDetect other than the fact is has a very unfriendly graphical user interface (GUI).

Available at <http://stegsecret.sourceforge.net/#downloads>

2.3.3 VSL (Virtual Steganographic Laboratory)

VSL focuses only on images and is more about steganography than steganalysis, as its only method of steganalysis is RS analysis which traces bit modification patterns in order to detect LSB steganography. It also has a binary similarity measures SVM used in blind steganalysis. However,

the design of this system is more along the lines of what this project aims to create as it is extensible, with a simple GUI interface.

Available at <http://vsl.sourceforge.net>

2.4 Potential tools

The system this project proposes is going to be written in Python, specifically Python 2.7 and, due to the complex nature of the problem, a lot of existing libraries are going to have to be used alongside the ‘vanilla’ Python modules. Therefore, it was decided that this project would be developed using Anaconda, which is basically a scientific distribution of Python with over four hundred libraries commonly used in science, mathematics, engineering, and data analysis built in. Anaconda can be obtained here <https://www.continuum.io/downloads>.

Some of these libraries such as SciPy and Pil (python imaging library) will be invaluable to this project as SciPy has OpenCV which is also a great image processing library. It also gives access to Wav file manipulation with the ‘scipy.io.wavfile’ module. Also included within the SciPy library is SKlearn, which is great for machine learning and could prove to be extremely useful. In terms of steganalysis, the only python library available at the time of writing was PyADS, written by Robin David and released under the MIT license, for detecting and retrieving alternate data streams. Every other method of steganalysis used in the system will have to be written from scratch. In order to build the GUI and make sure that it is cross platform compatible I am going to be using a python wrapper of the Qt Framework called PyQt. I will also be using the ‘*Qt Designer*’ tool to help build the GUI, allowing for relatively quick changes and rapid prototyping.

2.5 Detailed project outline

In order to demonstrate the achievement of the aim stated in the introduction, this project has identified steganalysis software currently available and the downfalls of these programs. The

intended system will focus solely on steganalysis, gathering as many existing targeted attacks as possible in the timeframe and creating new attacks where none already exist. It will create a framework for extensibility and expansion as steganography is such a large, complex and ever changing field. This project will produce a robust software solution intended to be used in forensic investigations and, with thorough tests to show the detection ability, it will demonstrate on what basis it is judged an improvement over existing solutions.

3 Specification and Design

This section covers, in detail, what the proposed system should do and will describe how this project is going to achieve these aims.

3.1 Business model

The business model that this project aims to adopt for the software is that of the open-source model. This means that unlike proprietary software that has restrictive copyright licenses with the intention of making a profit, open-source software can be given away for no charge and not only can be, but is intended to be, modified by the community (Open Source, 2016).

3.1.1 Functional requirements

1. The system should be able to detect the use of steganography in files.
 - a. Importance: High
 - b. Description: The system should use statistical properties of the file in question to determine whether it contains steganographic information.
 - c. Benefit: The system could be classed as a steganalysis tool.
 - d. Risk: None
 - e. Acceptance criteria:
 - i. In order for the system to be defined as a steganalysis tool it must be able to detect steganography in at least one file type.
 - ii. In order for the system to be deemed acceptable it must be able to detect more than one form of steganography.
2. The system should be able to retrieve hidden content where possible.
 - a. Importance: High

- b. Description: Once the method of steganography has been discovered the tool will exploit weaknesses in that method to recover the concealed content.
 - c. Benefit: The system would be able to not only tell if a file was steganographic, but could also retrieve the hidden message. This could help law enforcement obtain a warrant, as getting a warrant on a statistic alone is often difficult.
 - d. Risk: None
 - e. Acceptance criteria:
 - i. In order for the system to be deemed acceptable it must have methods to extract messages from more than one form of steganography.
3. The system should be extensible.
- a. Importance: High.
 - b. Description: Although this would normally be a non-functional requirement as it measures the operation of a system as opposed to specific behaviour. In this system it is integral to the outcome of the project being deemed a success and is therefore a functional requirement.
 - c. Benefit: Allowing new techniques to be added to the existing base attacks, keeping the system current in an ever-changing environment.
 - d. Risk: The system could be deemed ‘stale’ without it.
 - e. Acceptance criteria:
 - i. The system should create a framework for extension, thereby allowing a user to correctly implement and execute their own plugins.
 - ii. If the plugins are compliant with the system’s specified design, it should be able to run multiple plugins and keep normal operation.

4. The system should have a GUI.
 - a. Importance: High
 - b. Description: The system must have a clear, user-friendly GUI.
 - c. Benefit: A GUI would help the system to differentiate itself from other steganalysis tools on the market.
 - d. Risk: Without a GUI the software might not appeal to as many users in the market.
 - e. Acceptance criteria:
 - i. In order for the system to be deemed acceptable it must have a GUI.
5. The system should create and maintain a database of MD5 hashes of files found to contain steganographic content.
 - a. Importance: Medium
 - b. Description: The reason for storing the MD5 hash and not the filename is that the filename might get changed but the file itself could go untouched.
 - c. Benefit: Having a database and checking if a file exists in that database before it is processed for analysis would avoid processing the same file multiple times, as files with steganographic content are often passed around.
 - d. Risk: Without a GUI the software might not appeal to as many users in the market.
 - e. Acceptance criteria:
 - i. A local database should be enough for the purpose of this project.

3.1.2 Non-functional requirements

A non-functional requirement specifies criteria that can be used to judge the operation of a system, rather than specific behaviours.

1. The system should have a high degree of usability.

- a. Importance: High.
 - b. Description: The system should be user friendly and self-explanatory.
 - c. Benefit: If the system is easy enough to use, no complex training would be required.
 - d. Risk: If the system is not usable, it will not be easily adopted by the community.
 - e. Acceptance criteria:
 - i. The system should be easy enough for someone with a basic level of computer interaction experience to use with minimal instruction.
2. The system should be measurably testable.
- a. Importance: High.
 - b. Description: Structured tests that have predictable results should be able to be run.
 - c. Benefit: The benefit of this is that I will be able to get a clear understanding of the strengths and weaknesses of my system.
 - d. Risk: Without the ability to carry out controlled tests, I would not be able to test its worth.
 - e. Acceptance criteria:
 - i. Being able to run controlled tests that produce meaningful results in terms of successful detection rate.

3.2 Initial UI mock-ups

All mock-ups were designed in Balsamiq under a thirty-day trial license. Balsamiq is a piece of software that allows rapid UI mock-up development. The title page is basically just a

welcome screen and a first point of contact for the user. It will also allow the user to access any area they like. For that reason, all cancel buttons will reset the system and return the user here.

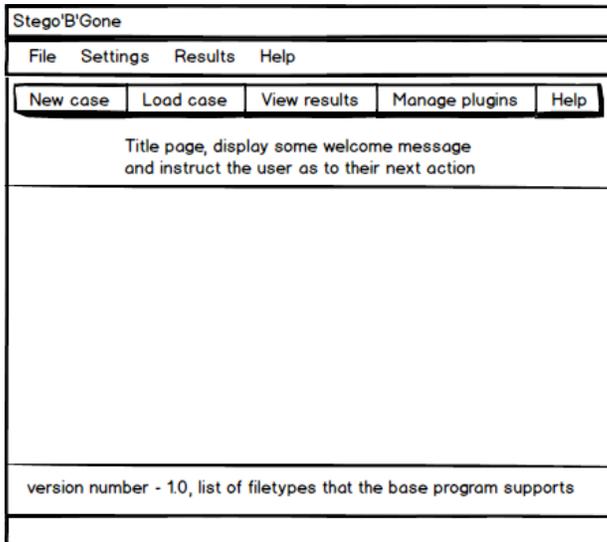


Figure 2 – UI title page mock-up

The 'new case' page will allow the user to enter identifiable information to make the generated report more readable and identifiable for each case.

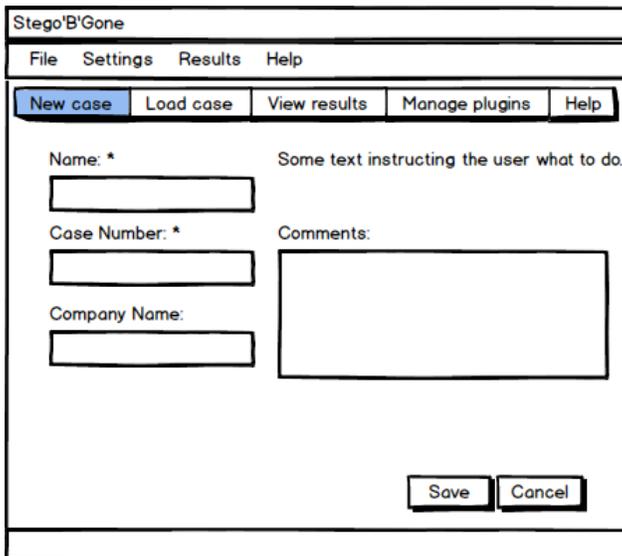


Figure 3 – UI new case page mock-up

In order to achieve a greater feeling of continuity, the *'load case'* page will allow a user to load in a previously generated HTML or XML results file for viewing inside the software. If the file is not a HTML file they will be returned to the title screen.

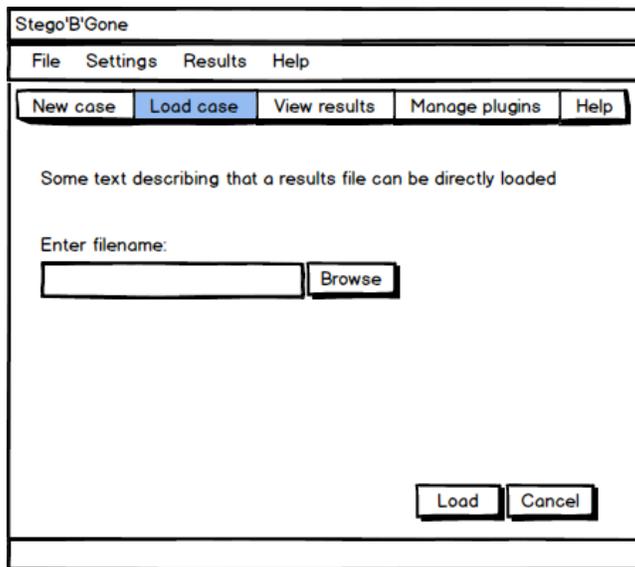


Figure 4 – UI load case page mock-up

The *'analysis page'* will contain the main functionality of the system since this is where the actual steganalysis tasks will take place. The initial plan is to have a button that the user can click to either read in a file or a directory and then, begin automatically processing the files and applying the various targeted attacks that apply to that specific file. There will be some form of output to notify the user that the system is processing tasks, probably in the form of a progress bar or some printed statement. This will also allow the user to estimate roughly how long the system will take to finish processing.

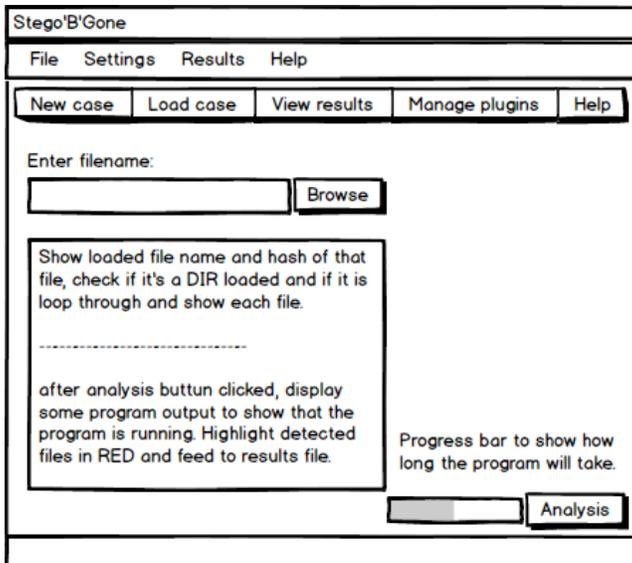


Figure 5 – UI analysis page mock-up

The 'results page' will allow a user to view results files within the software as opposed to opening the file externally. This will give the system better *'flow'* and an increased sense of continuity.

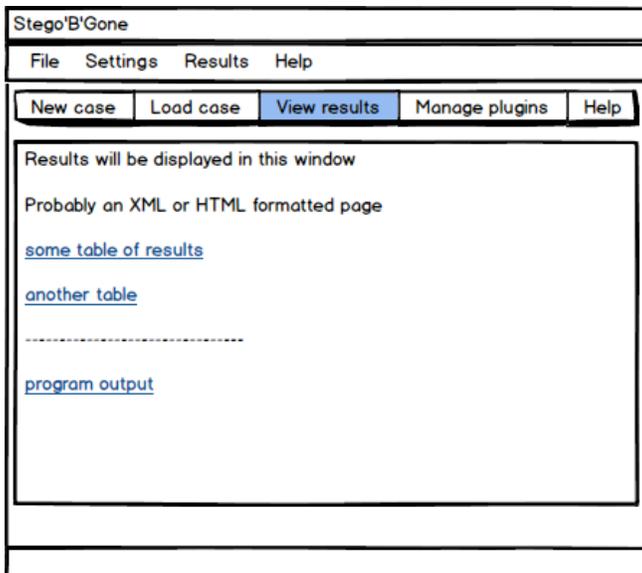


Figure 6 – UI results page mock-up

The *'plugins page'* will allow a user to load in a new plugin. It will check if that plugin is compliant and if it is, add it to the list of existing plugins. The user will then be able to activate or deactivate plugins.

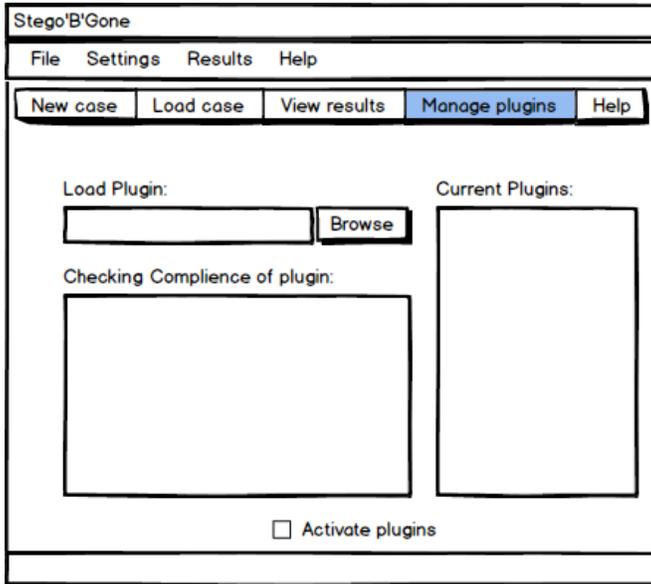


Figure 7 – UI plugins page mock-up

The *'help page'* will act as a user guide and detail the usage of the system. It will also allow the user to submit bug reports or general suggestions.

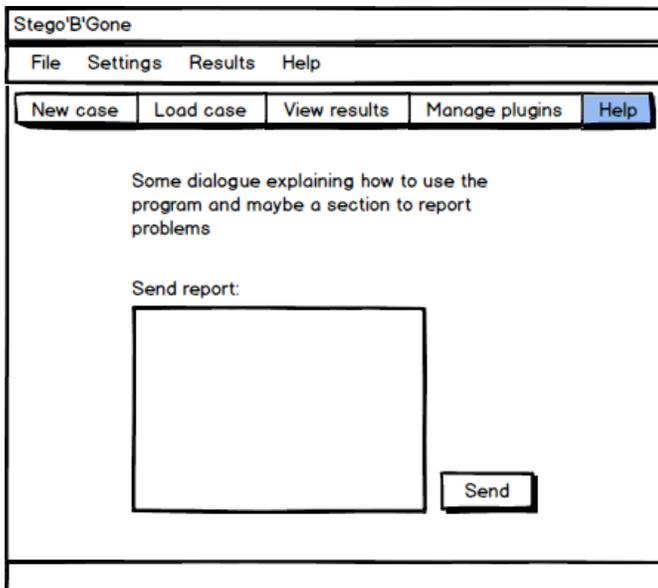


Figure 8 – UI help page mock-up

3.3 Data flow

Although a detailed set of requirements has been provided above, in order to best describe the flow of data in the system, use case diagrams and activity diagrams will be presented to demonstrate it pictorially.

3.3.1 Use case diagrams

A use case diagram is a graphical depiction of the interactions between elements within a system and, in this particular instance, they are used to show the relationships among the actors (users of the system) and each use case.

3.3.1.1 New case

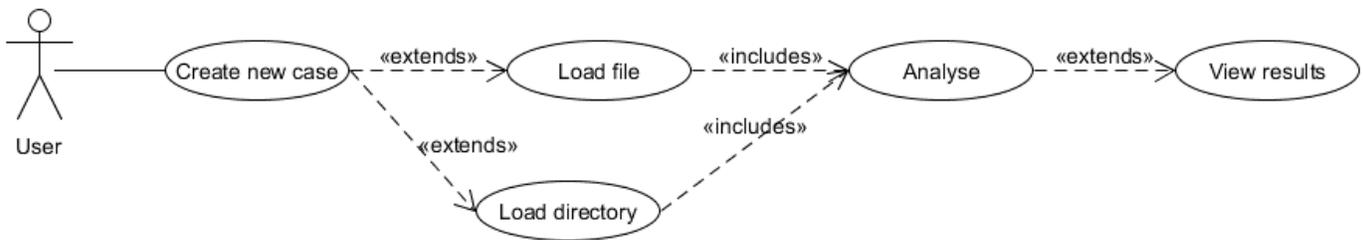


Figure 9 - Create new case (use case)

This use case is necessary for the core functionality, as if the user cannot interact with the system to create a new case and analyse files then the system would not be deemed a success.

3.3.1.2 Load case

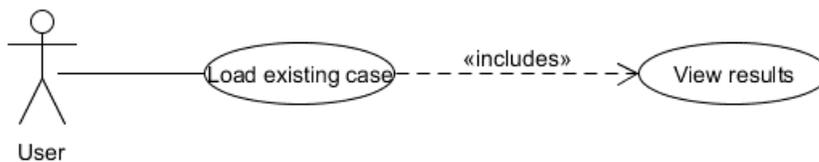


Figure 10 - Load existing case (use case)

The ability to load an existing case is not necessary for the functionality of the system but creates a more consistent piece of software and is thus necessary for professionalism.

3.3.1.3 Manage plugins

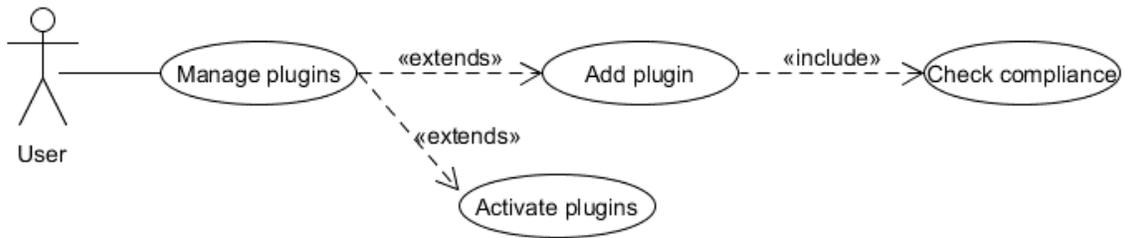


Figure 11 - Manage plugins (use case)

The manage plugins use case is necessary for the main functionality of the system as the user has to have the ability to add new methods of steganalysis if they wish. Without this interaction the system would become obsolete.

3.3.1.4 View help

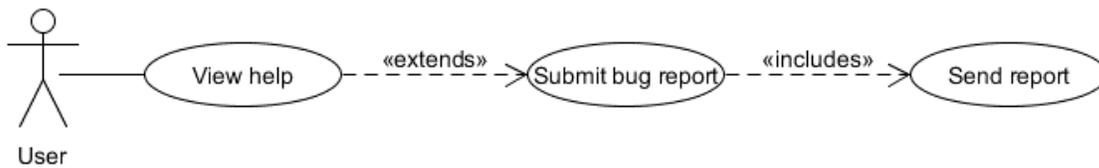


Figure 12 - View help (use case)

The view help use case is not necessary for the operation of the system but presents the user with a detailed overview of exactly how to use the software. This interaction will also show the user the licensing information.

3.3.2 Activity diagrams

An activity diagram is a graphical representation of procedural system activities and contains support for choice and iteration. They are considered a variant of the state chart diagram.

The following activity diagrams attempt to describe the flow of data in each particular use case.

3.3.2.1 *New case*

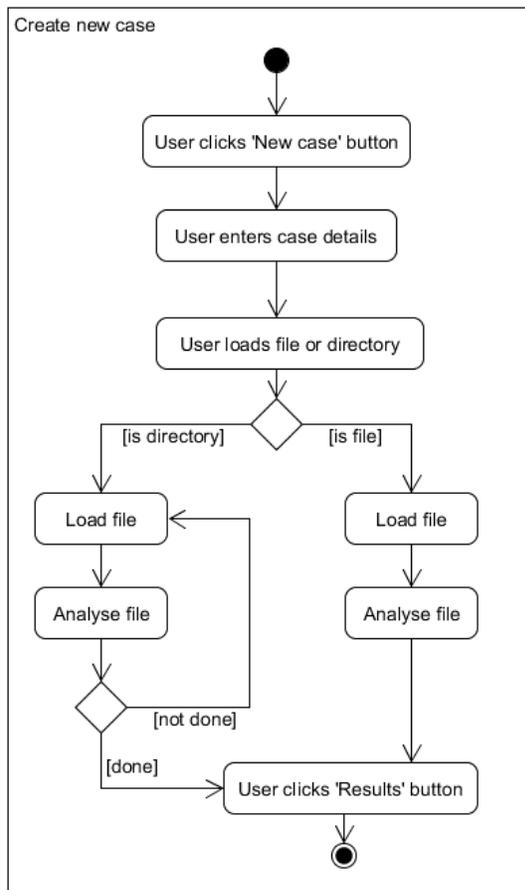


Figure 13 - Create new case (activity diagram)

3.3.2.2 Load case

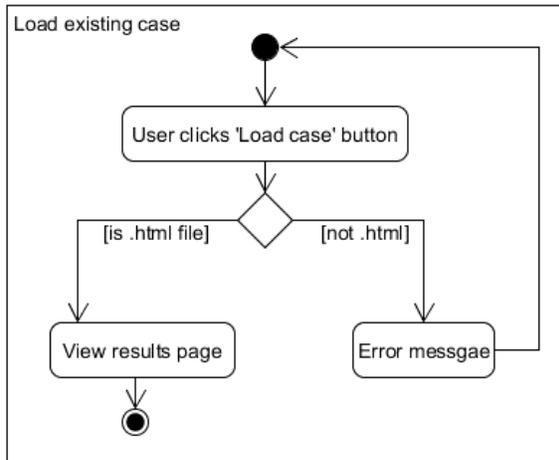


Figure 14 - Load existing case (activity diagram)

3.3.2.3 Manage plugins

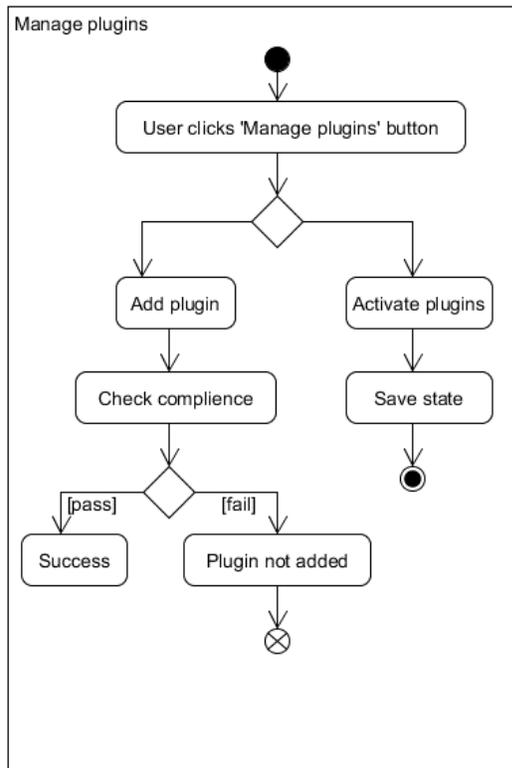


Figure 15 - Manage plugins (activity diagram)

3.3.2.4 View help

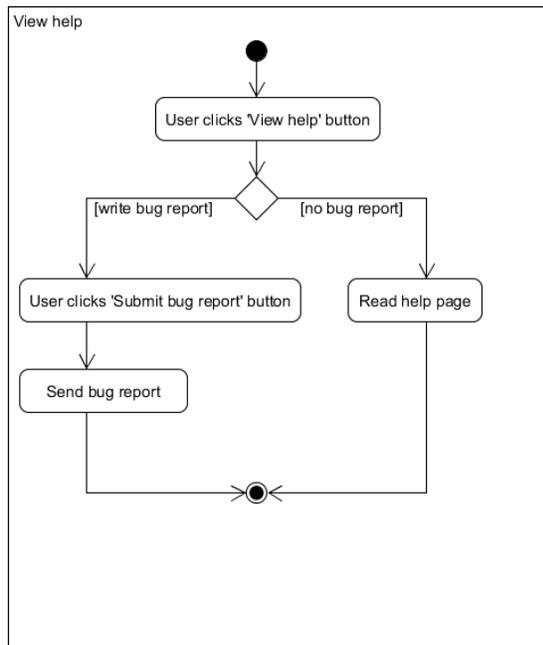


Figure 16 - View help (activity diagram)

3.4 Class diagrams

The following class diagrams attempt to express the static architecture of the system. This section shows each of the classes intended to be implemented and the propose of each class, as well as describing their relationships to each other.

The figure below shows the intended structure for each steganalysis attack, in which each attack implements the abstract class or interface of *Attack*. This is done to ensure compliance when running plugins, as the plugin loader could simply check if the plugin is question is an instance of the attack interface.

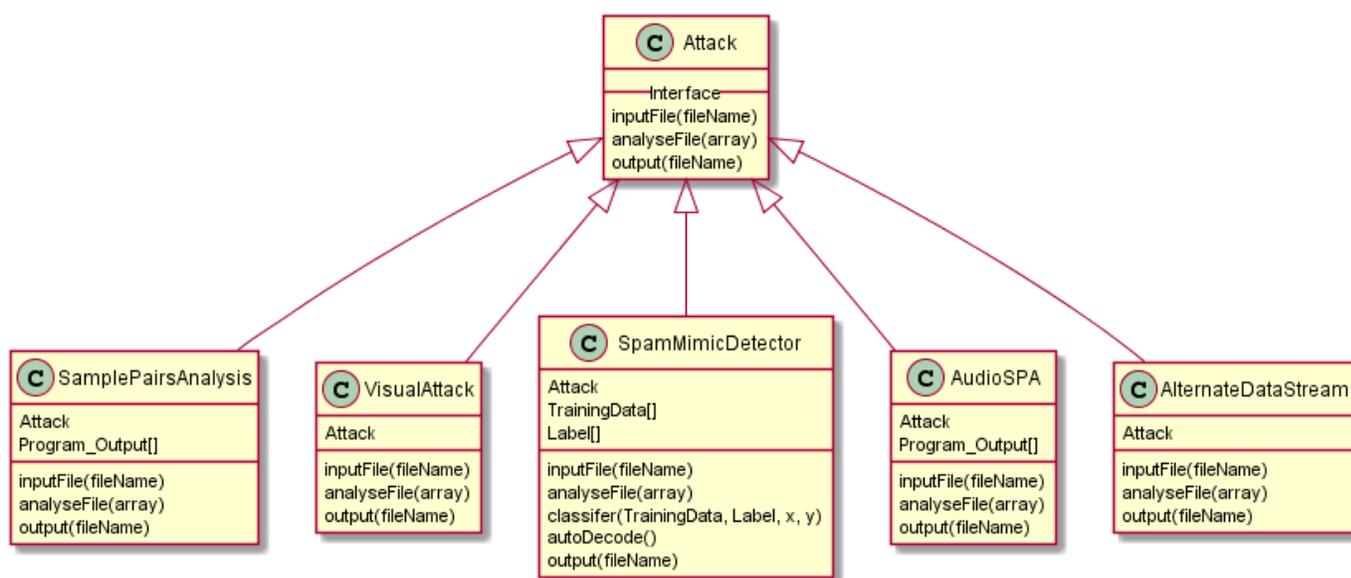


Figure 17 - Class diagram for steganalysis attacks

The plugin class structure is intended such that the *PossiblePlugins* class looks for plugins in a given directory. The *LoadPlugin* class is used to get human-readable names from the *PossiblePlugins* list of plugins and display it in the GUI. The *RunPlugin* class determines if the each of the plugins from *PossiblePlugins* have been correctly written and if so it will then run them.

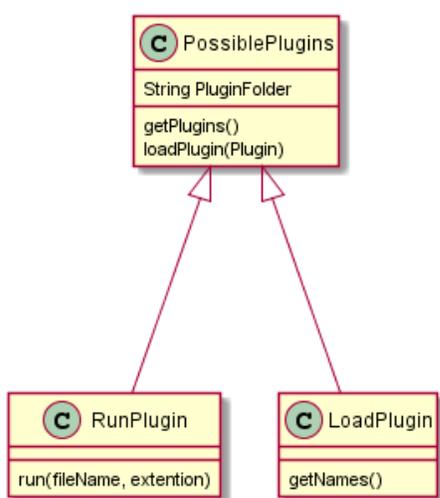


Figure 18 - Class diagram for the plugin architecture

The results class is intended to be passed data to from the GUI or *'MainWindow'* class. The data it will be passed will be all the user-input such as the investigator name and case number etc. as well as the program output from each attack. This class will be used to then display this information in a more human-readable manner.



Figure 19 - Class diagram for the results presentation

The *'MainWindow'* class implements the *'Ui_MainWindow'* class, which is automatically generated from the Qt Designer program. The MainWindow class will also have methods to control the various aspects of functionality of the rest of the program, such as the database checks and creation, as well as starting the thread for each attack.

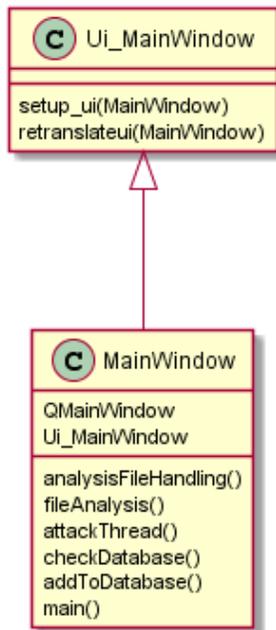


Figure 20 - Class diagram for the GUI

3.5 Final UI designs

The final UI designs were created as template files in 'Qt Designer' and then converted into Python code using the 'pyuic' program. I decided to use the Qt Framework as it is widely used in developing software intended to be run on multiple platforms with minimal (sometimes no) changes to the underlying code, whilst still maintaining the speed and capabilities of a native software program (Qt Framework, 2016). The Qt Designer program also allows for rapid prototyping, using a drag-and-drop interface. This makes complex interfaces much easier to handle and allows for more time spent developing back-end code.

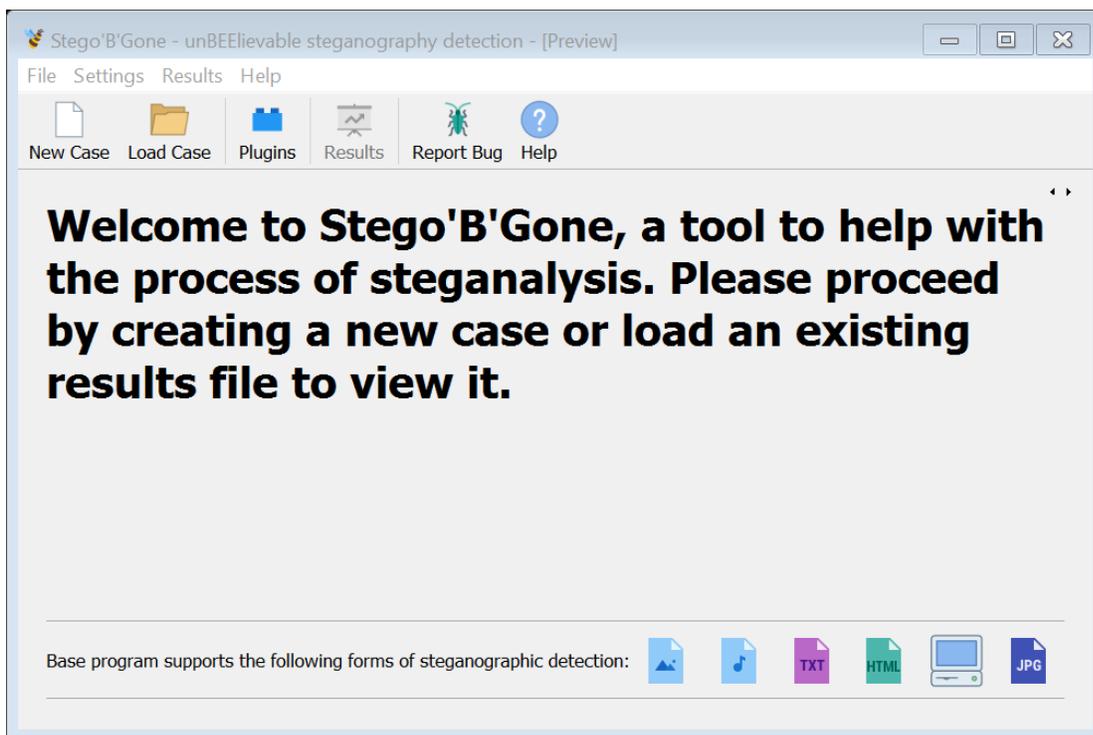


Figure 21 - Final welcome screen

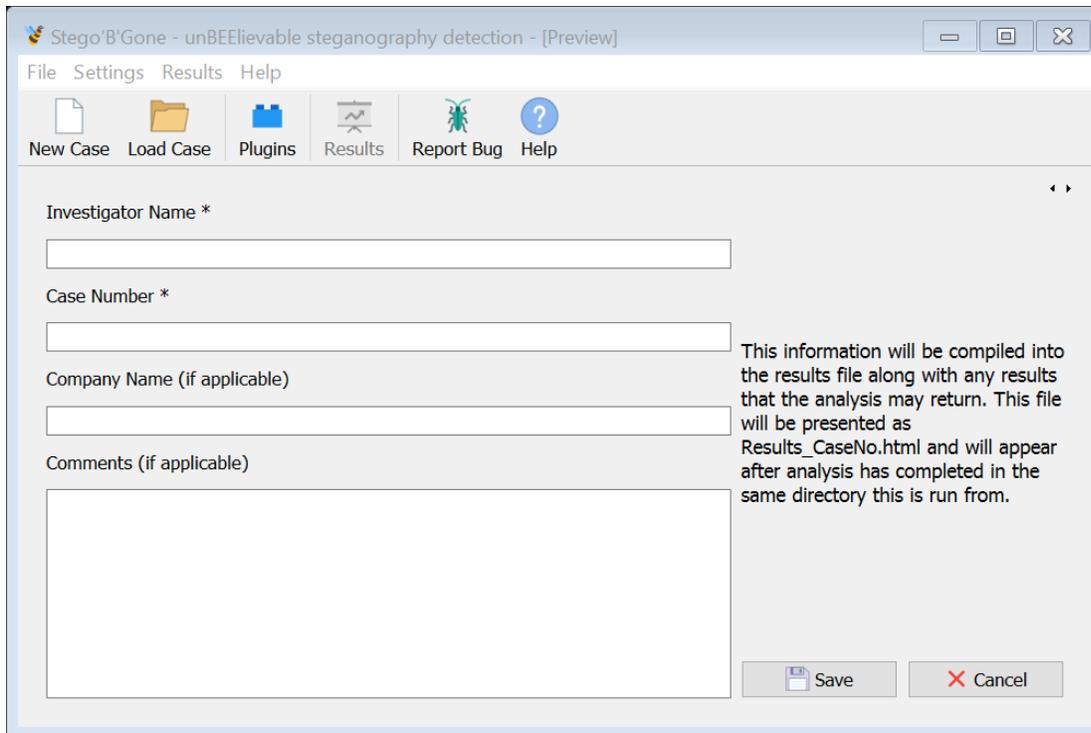


Figure 22 - Final new case screen

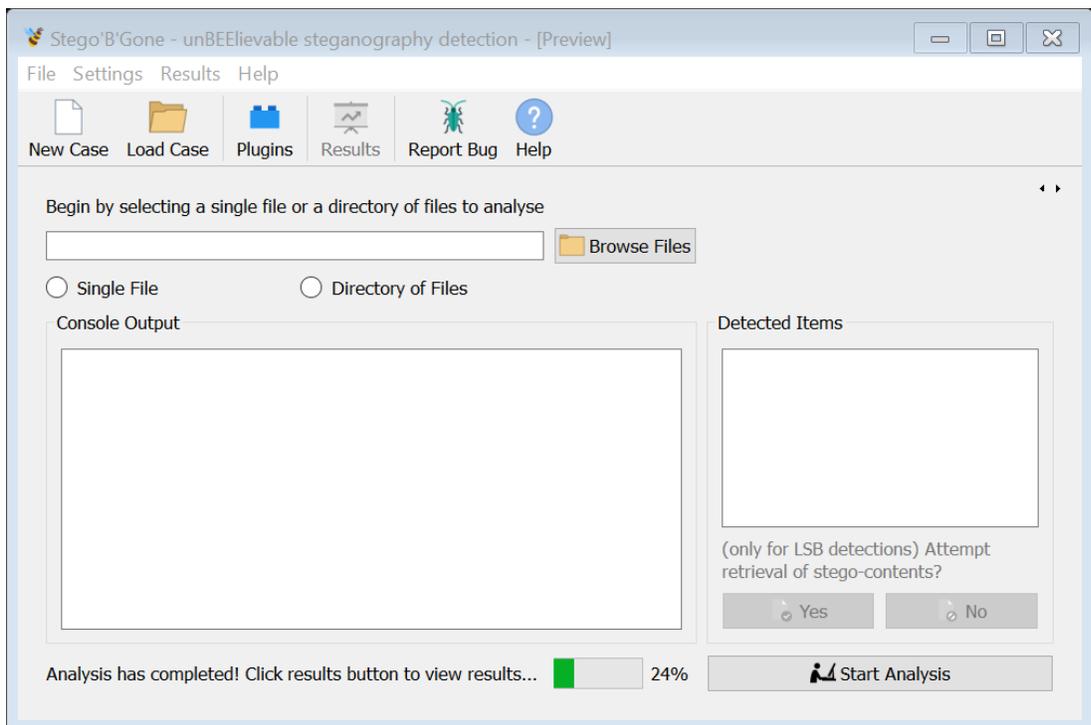


Figure 23 - Final analysis screen

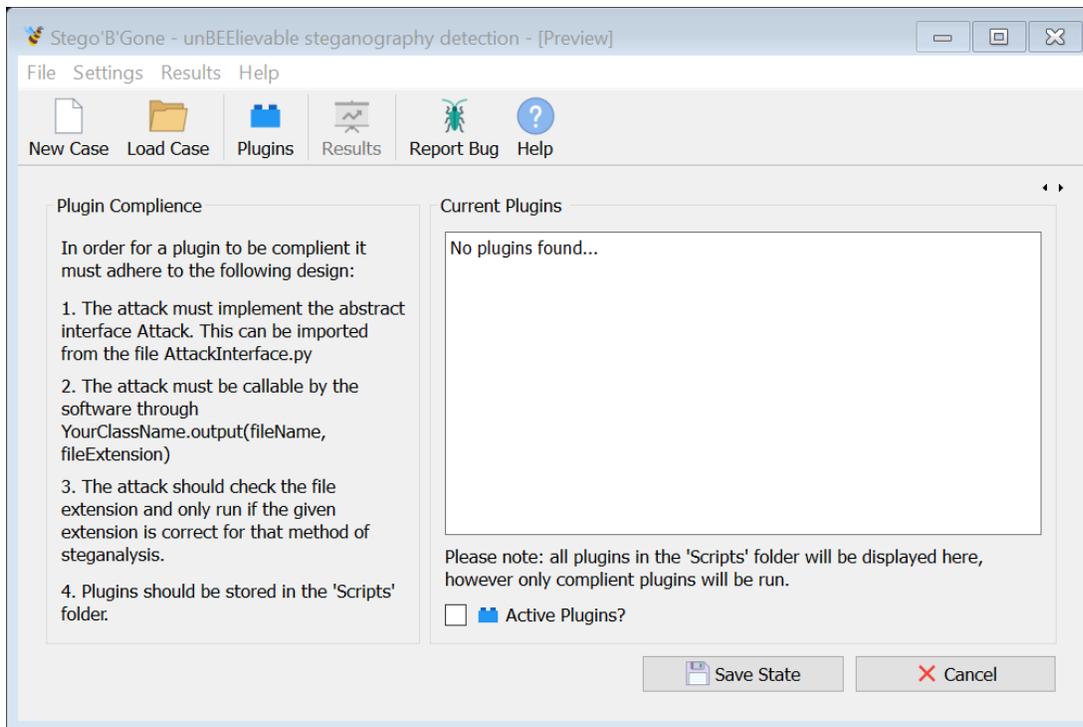


Figure 24 - Final manage plugins screen

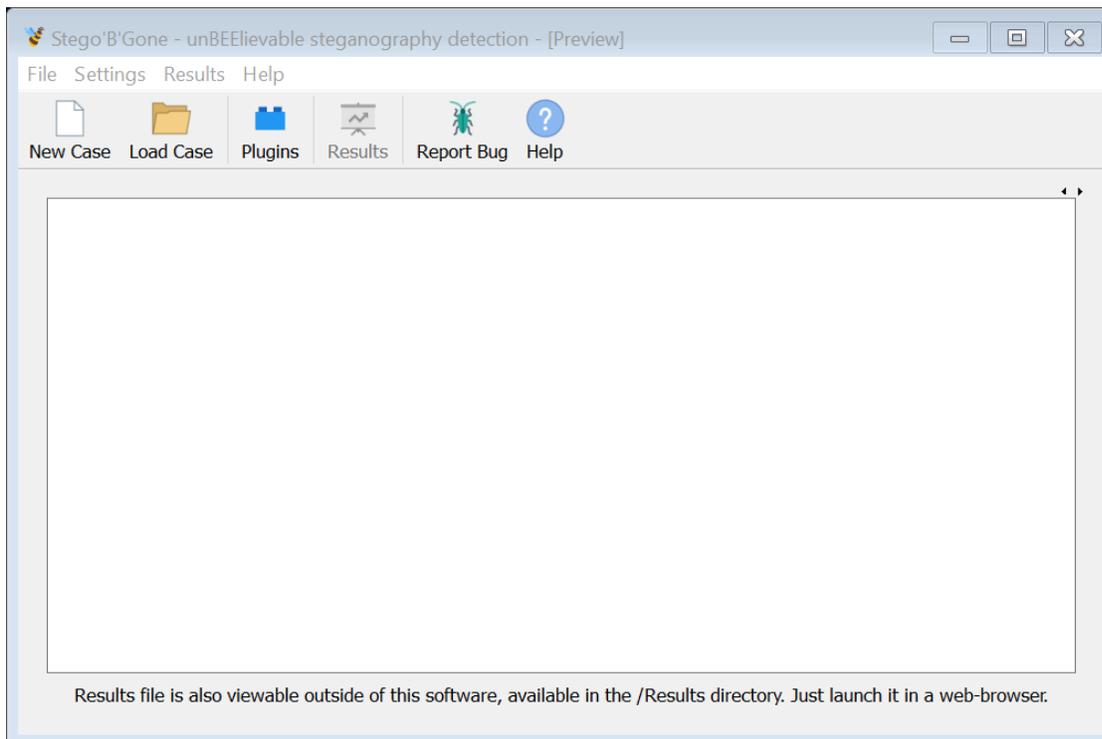


Figure 25 - Final results screen

All icons used in the software were obtained from the icons8 web application, found at <https://icons8.com/>. It allows the free download and use of their icons under the ‘*Creative Commons Attribution-NoDerivs 3.0 Unported*’ licence, requiring only a link to their website in the about or help section in the application.

3.5.1 Changes from initial mock-ups

Instead of having a screen for the ‘*View help*’ section as initially proposed with a section to report a bug, I decided to split the two up, as after testing it, I felt that having two separate buttons allowed for a better flow. In the actual system, this is going to be coded such that when the ‘*Help*’ button is clicked, it will produce a pop-up with helpful keyboard shortcuts and useful information about the system, as well as containing the licence information.

Rather than having a bug submission area within the software, the ‘*Report Bug*’ button will simply bring up the default mail client on the user’s machine. This will create a new email with a predefined subject line containing information such as the version number of the software and the host OS, in order to make debugging problems easier. Another change from the initial mock-ups is that the ‘*Load case*’ button will simply bring up a file explorer dialogue. This will allow the user to load a .html file directly into the results page for viewing within the software as opposed to having its own page for a small amount of functionality.

3.5.2 Design ideas

After much deliberation I chose the name “*Stego’B’Gone*” for the software as I wanted something memorable but also relevant. The ‘B’ in the middle of the name implies a ‘*bee*’ hence the bee logo and the tagline, *unBEElievable steganography detection*.

4 Implementation

The below section describes the realisation of the proposed system as outlined in the specification and design.

4.1 Methodology

When choosing a development methodology, two main techniques were considered: waterfall and agile. Of these two, the agile method seemed more appropriate for this project as the waterfall method does not allow for modifications to the original specification. Furthermore, due to the iterative nature of agile, a new method of steganalysis could be created and then merged with the existing functionality of the system.

4.1.1 Agile

In terms of a software development methodology, the agile methodology made the most sense for this particular project as it is designed to handle unpredictability (Anon., 2008). Moreover, it allowed the development of one method of steganalysis and the ability to test it independently, then integrate it into the system and test again, moving on to the next method of steganalysis. Considering that the number of steganalysis methods that were intended to be implemented were unknown at the start of the project, this method, as opposed to another method such as the waterfall model, was the only viable choice.

This method also applied to the various other areas of functionality that I wished to develop for this project such as the database and the plugin architecture. The database back-end was created outside of the system in a test script, once initial testing had been completed and it appeared to work as expected, it was added in to the existing functionality. The same goes for the plugin

architecture; it was created independently, tested and then merged in with the existing functionality. A diagram of the agile development methodology can be seen in the figure below.

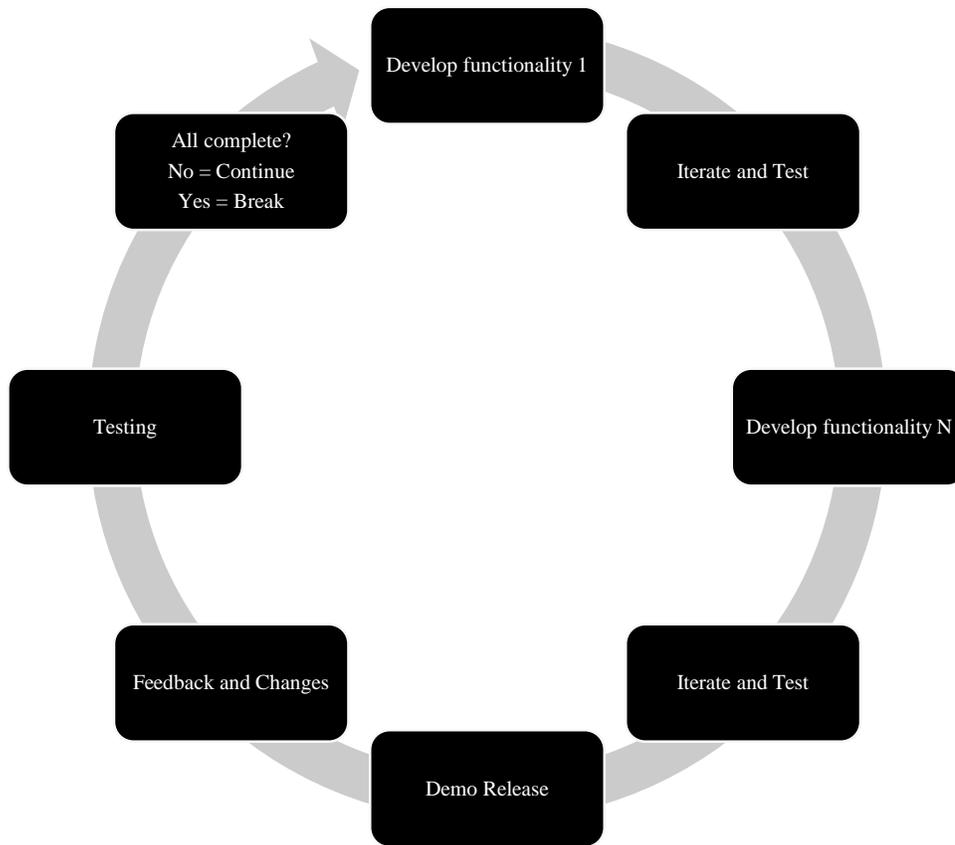


Figure 26 - Agile methodology

4.2 Libraries and Tools

Throughout the project I used various libraries, both internal and external, in order to achieve the aims of the project. These libraries are outlined below in order for the reader to have a better understanding of the specific purpose of each one.

4.2.1 External libraries

The following libraries are *not* included in the standard 'vanilla' Python 2.7 distribution.

SCIPY: SciPy is a fantastic library for scientific computing. It allowed for the quick implementation of machine learning with SKlearn and also gave easy access to .wav file manipulation.

PIL: The Python imaging library allowed for easy access to various image file types, which was extremely useful in the LSB steganalysis techniques. It also allowed me to rebuild an image from an array of pixel data in the visual attack against LSB.

PYADS: PyADS is a free to use library written by Robin David which allows the manipulation of NTFS Alternate Data Streams (ADS) of files and directories. It is licensed under the MIT license, which states:

“Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software” – MIT license

PYQT: PyQt is a Python wrapper around the C++ Qt Framework, that allows the Qt Framework to be used easily in Python. Qt is designed to be used in developing software that is intended to be run on multiple platforms.

REQUESTS: The requests module allows extremely fast and easy manipulation of HTTP requests to websites.

HTML: HTML is a module written by Philippe Lagadec and distributed under the CeCILL open-source license. It is designed to make the generation of HTML tables within Python extremely easy.

NUMPY: NumPy is a well-known Python library which can be used as an efficient multi-dimensional container of generic data or as a powerful N-dimensional array object.

4.2.2 Internal libraries

The following libraries are included in the standard Python 2.7 distribution.

HASHLIB: Hashlib allows for the easy generation of multiple different types of hashes, such as SHA1 and MD5.

SQLITE3: SQLite3 is a great in-built python library for dealing with databases and handling SQL queries.

IMP: The Imp module is used in searching for and loading modules and was extremely useful in the development of the plugin architecture.

INSPECT: The Inspect module allows for easy information retrieval of live objects such as modules, classes and methods.

ABC: ABC stands for abstract base class, and provides the infrastructure for creating an abstract base class. This is especially useful if you want to create an interface in Python.

RE: This module provides regular expressions for pattern matching within strings and is extremely similar to the Perl regex.

4.3 Steganalysis methods

From the initial research conducted at the start of the project presented in *figure 1*, it was evident that the main areas of steganography that the project needed to address were those in images, text, audio and the file system. Therefore, I decided to narrow the scope of the project to focus on at least one attack against each of these areas.

The methods of steganalysis that were eventually implemented are outlined below.

4.3.1 Least significant bit embedding

LSB embedding is the most prominent form of modern steganography and can be applied to a multitude of cover medium with minimal effect on the file. The most common form of LSB embedding is within images and this is where my first focus was. I researched several methods for detection of LSB steganography within images and the three most prominent methods were the

visual attack, chi-squared test of randomness (sometimes called χ^2) and the sample pairs analysis attack. These attacks are discussed in more detail below.

4.3.1.1 Visual attack

The visual attack is the simplest form of LSB embedding detection. The basis of this attack is to enhance the colours in the LSB of each pixel. It involves checking the LSB of each pixel in an image. If the LSB of that pixel is a 0, it remains untouched but if the LSB of the pixel is a 1 then the colour value for that pixel is amplified to its maximum intensity of 255. This technique can also be applied to the next most-LSB as some embedding programs embed into two bit-planes, as can be seen from the figure below. These recovered images could then be combined to produce the hidden file.



Figure 27 – From left to right, cover file, recovered image (2nd bit-plane), recovered image (1st bit-plane or LSB)

Image obtained from the steganography Wikipedia page which can be found here:
https://en.wikipedia.org/wiki/Steganography#/media/File:Steganography_original.png

This will intensify the pixels of the original image (if there is in fact an image embedded) and should make the original image visible to the human eye if the image has been embedded linearly. If the image has not been embedded linearly, visual patterns should still emerge but the image will not be recoverable unless the investigator knows the path of insertion. Visual patterns will also appear if text has been inserted into the image, see *figure 28*:



Figure 28 - Left to right, original image, 5kb random data embedded, embedded poem "if", section of 3rd image

Original image (left) and original images with embedding obtained from <http://www.guillermi2.net/stegano/tools/> for testing my program. However, the images displayed above (other than the original on the left) are the output of my program.

This effect can be seen from the above set of images. The second image has 5kb of random data embedded and, as expected, the visual attack produces lots of randomly placed colour blocks.

With the image on the right (zoomed in on the far right) that contains the poem 'If' by Rudyard Kipling however, we can start to see vertical lines emerging. This is due to a property of ASCII text in which lower case characters share three out of the eight bits that make them up. As one bit can be hidden in each of the three colour bands, this means that similar pixels will appear every $\left(\frac{8}{3}\right) * 3$ pixel, which is every eight pixels, and hence the repeating vertical lines of similar colour.

Therefore, even this simple attack can produce fairly decent results. It does however require a human to look at each file output by the program and determine whether there is some visible pattern there. If the image file is rather large this process can also take quite a while, making it not particularly viable as a method of steganography detection. However, it is a reasonably good method for LSB retrieval and that is what was eventually decided upon for this system. The system

would need to detect the presence of LSB steganography by some more autonomous means and then use this method to attempt retrieval of the stego-artifact, and as such, this next section will discuss statistical attacks that can be used in detecting the presence of LSB steganography.

4.3.1.2 Chi-squared test

The chi-squared test of randomness is based on the differences between the observed properties of a file and the expected properties. The attack was first proposed by Andreas Westfeld (Westfeld & Pfitzmann, 2006).

This is written as:

$$\chi^2 = \sum \frac{(o - e)^2}{e}$$

Where o is observed value and e is the expected value.

This can be used to detect the presence of encrypted information hidden within an image, since the expected probability of a LSB embedding within a natural image being either 0 or 1 will not be 0.5 as there are often similar colours neighboring each other. However, an image file with encrypted content embedded should have an area or areas in which the observed probability of a LSB being either 0 or 1 is much closer to 0.5 which would indicate that encrypted content is embedded in the file.

Due to its highly specialised design, the chi-squared test is not the most appropriate test for a system that is intending to detect steganography in many different cases. A more advanced attack will now be discussed. This brings me on to sample pairs analysis, a technique first proposed in the paper '*Reliable Detection of LSB Steganography in Color and Grayscale Images*' (Fridrich, et al., 2008).

4.3.1.3 *Sample pairs analysis*

Sample pairs analysis (SPA) is the simplest case of RS analysis. It works by utilizing the fact that neighboring pixels in natural images exhibit strong correlations. Therefore, if you were to take a pair of horizontally neighboring pixels, say (r, s) , from the set of all pixels P , the larger the difference between $|r - s|$ is, the higher the probability that at least one element in that pair has been modified. The algorithm starts when the pair (r, s) is added to one of the primary sets X, Y or Z based on the following conditions:

$$X = (r < s \text{ AND } s \text{ is even}) \text{ OR } (r < s \text{ AND } s \text{ is odd})$$

$$Y = (r > s \text{ AND } s \text{ is even}) \text{ OR } (r > s \text{ AND } s \text{ is odd})$$

$$Z = (r = s)$$

With LSB embedding, the count of pixel pairs in X will decrease, but the count of pairs in Y will increase. A similar analysis can be carried out if the pair (r, s) has $r > s$ because the situation is complementary. X will still decrease and Y will still increase.

Due to the sets X and Y being symmetrical, the cardinalities of X and Y should be the same, because in natural images it should be equally likely to have either $r < s$ and $r > s$, regardless of the fact that s is even or odd, meaning that if content has been embedded then the difference between $|X| - |Y|$ should not be 0. There are four possibilities for a modification pattern of a given pixel pair (r, s) under LSB embedding can have, which are outlined below:

1. r and s unmodified or 00
2. r is modified or 10
3. s is modified or 01
4. r and s are modified or 11

LSB embedding should make a pair move from its primary set to another set. Under the assumption that the stego-content is embedded over a random path through the image, then each pixel in the image is equally likely to have been modified. This means that the expected number of pixels modified with a specific modification pattern is the same for every primary set.

The next goal is to come up with an equation for the change rate β using only the cardinalities of the primary sets which can be directly calculated from the cover image. This can be seen from the below code snippet from the ImageSteganalysis.py module in the method *'SamplePairsAnalysis.analyseFile()'*. In the code shown in *figure 29*, β is represented as b . The variable w is the width of the image and h is the height. The statement to check if a is greater than 0 is just to avoid a division by zero error. The variable K is basically the union of the sets W and Z . The final estimate of the change rate is taken from the lesser root of the quadratic equation (Fridrich, 2010).

```
a = 2 * K
b = 2 * (2 * x - w * (h - 1))
c = y - x

D = math.pow(b, 2) - (4 * a * c)

if a > 0:
    p1 = (-b + math.sqrt(D)) / (2 * a)
    p2 = (-b - math.sqrt(D)) / (2 * a)
else:
    p1 = -1
    p2 = -1

return min(p1, p2).real
```

Figure 29 - Quadratic equation from SPA attack

If $K = 0$ then it is impossible to estimate β , but because K is the number of pixel pairs where both the values belong to the same LSB pair, this would only happen with an extremely small probability in natural images. The code I developed for this attack was based on pseudo-code from Jessica Fridrich's book *'Steganography in Digital Media'* (Fridrich, 2010) and it

required a lot of trial and error on my behalf as I struggled to fully understand the mathematics in this particular attack.

4.3.1.3.1 Images

In order to use this attack on images, I first developed a method to detect the image type, such as RGB, RGBA or grayscale using the python imaging library. I then wrote a method to split a colour image into its separate colour bands for RGB images and return an error for other image formats. Once the image was split into colour bands, I then returned a *'pixel map'* for each band, then attacked them separately.

The value 0.025 in my code is described by Fridrich as *"the constant γ is the threshold on the test statistic β set to achieve $P(FA) < \epsilon FA$, where ϵFA is a bound on the false-alarm rate"* (Fridrich, 2010). Again, this was mostly set by trial and improvement in my code. This allows the system to detect stego-content that has been embedded in as little as 0.05 bits per pixel (bpp) of the cover image, as the bpp is roughly twice the estimated change-rate. If this threshold is lowered, the system starts producing a lot more false-alarms, and if raised, the system starts to miss cover images with LSB embedding containing a small amount of data.

This can be seen in the below code snippet from the *'SamplePairsAnalysis.output()'* method in the *'ImageSteganalysis.py'* module.

```
def output(self, image):
    r, g, b = self.inputFile(image)
    isStego = False
    # 0.025 is the lowest change-rate threshold that SPA is effective to
    if self.analyseFile(r) > 0.025:
        program_output.append("%s contains steganographic content in the Red plane" % image)
        program_output.append("Estimated change-rate = %s" % self.analyseFile(r))
        program_output.append("Estimated embedded message in bpp = %s" % ("%2f" % (2 *
self.analyseFile(r))))
        program_output.append("=====")
        isStego = True
```

Figure 30 - False-alarm boundary for SPA detections in the red band

4.3.1.3.2 Audio

In order to use this attack on audio files, I realised it would only work on dual channel or stereo audio as it essentially requires a two-dimensional array to grab *'sample pairs'*. I had to use the SciPy module, specifically the *'scipy.io.wavfile'* module to split a wav file into its separate bands and then apply the same algorithm to it. The audio file's stego-threshold (the point at which the file is deemed steganographic) was much harder to tweak than with images. This was due to the fact that most .wav files seem to have some effects applied to them already and these modifications seemed to be affecting the change-rate giving many more false-positives.

4.3.2 SpamMimic

Not much is known about the inner workings of SpamMimic as it is a closed source system. It seems to almost magically convert a secret message to some seemingly innocuous spam and then convert it back to the message again. Much of the research involved reading through many forum posts with speculative comments as to the inner workings of the system and attempts to decode it, none of which were successful. What I did manage to find is that the system is likely based on the work of Peter Wayner (Wayner, 1992) and his mimic functions. It seems likely that the system works by following his method of *'context free probabilistic grammar'* and then uses Huffman tree encoding in some way. However, in order to detect the presence of SpamMimic in an email or document, one does not need to know the inner workings of the encoding, only the statistical properties of the text that make it identifiable.

My plan was to find a way of representing the text on a graph, giving it an x and y value. Then plotting various SpamMimic texts of different lengths alongside regular spam messages, taken from an archive of known spam messages, and seeing if a cluster would form. I could then

train a binary support vector machine (SVM) to classify an unseen document based on knowledge of the statistical properties of known SpamMimic documents.

In order to detect the presence of SpamMimic embedding, I first had to encode and analyse many short messages, gradually increasing the length of the message and looking at the effect this had on the cover text. Eventually I realised that, as with block encryption methods like AES, the output had a block-size. Once the input was over a certain length, the output would create another block denoted by the capitalised word 'Dear'. I also discovered that there are certain markers in the text that when divided by the block-size always fall between a certain range. The marker in question is the '!' marker. This divided by the block-size will be the x value.

To get the remaining y value for plotting, I found that counting the unique words in the text, and similarly dividing that by the block-size, also gave a value that was unique to SpamMimic documents. Plotting these values on a graph formed a specific cluster of SpamMimic documents to the right hand side of the graph, with normal spam messages dotted around the graph which can be seen from the figure below.

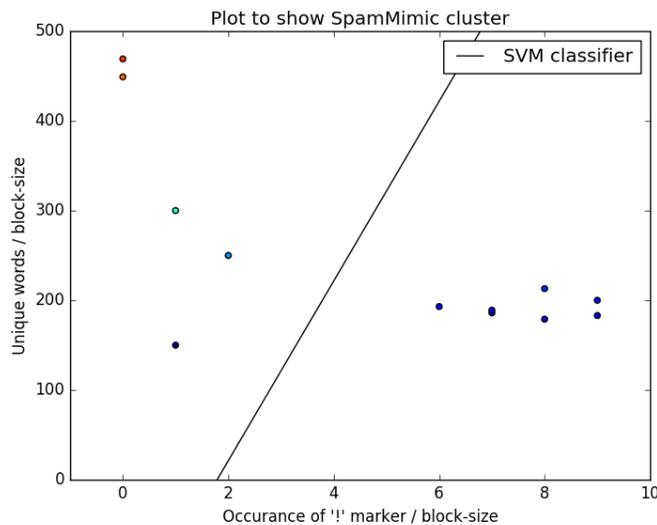


Figure 31 - Plot to show SpamMimic cluster on the right and normal documents on the left

By giving the training data a label (1 for SpamMimic, 0 for not SpamMimic) to each pair of values collected from various documents, I was able to train an SVM to accurately classify an unknown document as either SpamMimic or not which can be seen from the following code snippet.

```
# custom method for this particular attack
def classifier(self, data, dataLabel, x, y):
    # build SVM
    clf = svm.SVC(kernel='linear', C = 1.0)
    clf.fit(data, dataLabel)
    # predictions
    return int((clf.predict([x,y])))
```

Figure 32 - Code snippet to classify unknown document as SpamMimic or not

4.3.3 Alternate data streams

In order to detect alternate data streams, I used the library PyADS which I previously mentioned in the external libraries section of the paper. I had to create an instance of the *ADS()* class and then simply call the method *containsStreams()* to see if the given file contained an ADS. If the file did contain a stream, I then called the *getStreams()* method in order to get the content of the stream. This however, returns many false-positives as most files downloaded from the internet have a *[ZoneTransfer] zone id = 3* ADS. This is a security precaution introduced in XP SP2 to identify that a file has come from another computer, with *zone id = 3* specifically being the internet (Sanderson Forensics, 2006).

To be able to account for this, I simply ran a check to see if the stream contained the string *[ZoneTransfer]*. If it did not then I would recover the stream, thus eliminating most of the false positives.

4.4 Plugin architecture

In order to create the plugin architecture and make the system extensible, I required three specific modules; *imp*, *inspect* and *ABC* (abstract base class).

I first had to create a way of searching for plugins within the designated plugin folder. I did this using the method *'find_module'* from the `imp` module. This allowed me to search for valid Python modules, but if a Python file had already been run the directory would contain the `.py` and `.pyc` (compiled) Python files. Which means that each module would be run twice. In order to combat this, as each module was found I decided to split the extension from the file name. This allowed me to filter out any `.pyc` files.

I next needed a way to *'live'* load modules. This was done through the *'load_module'* method within the `imp` module, which allows a module previously discovered with *'find_module'* to be loaded. This then returns a module object. To be able to then check that the classes in this module were correctly implemented (adhering to the defined plugin structure) before they were run, I used the `inspect` module to get the classes in the plugin and check if the classes were an instance of the attack interface I built using the `ABC` module. If the class was in fact an instance of the `Attack` class, then it could be run by simply calling the *'Attack.output(filename, extension)'* method.

4.5 Database

To be able to achieve the aim of keeping a record of files that contained steganographic content, so that if they were to be seen again they wouldn't have to be re-analysed by the system, I had to implement basic database functionality. Instead of saving the file directly (as this could potentially have a huge storage impact) or the file name, I decided to save the MD5 hash of the file. This is because the file name could change but the file may remain untouched.

The way that this was eventually implemented was that, as each file was loaded into the system an MD5 hash was taken using the `Hashlib` library. If it was the first run through then a new database would be created at this point. If the database already existed, a check was then run

against the database for the hash of each of the files. If files were found to match the given hash in the database, then those files would be removed from the *'analysis'* text box and added to the *'detected'* text box, along with the method of steganography previously found, saving them being analysed again. I used the module SQLite3 to implement much of the database structure.

4.6 Problems encountered

Throughout the development of this project I encountered many difficulties, mostly due to my over-ambitious aims and complex subject matter. I quickly realised that my intended goal of being able to detect and retrieve the content from around seven methods of steganography was going to be extremely difficult in the given timeframe, as steganalysis was a very new field to me.

The main problem with most of the methods of steganalysis is they require an extensive knowledge of advanced level statistics and I struggled to understand how many of the attacks worked, even at a high level. It also seemed that only information available about certain attacks (especially in the more complex attacks) was in academic papers, that were often extremely difficult to understand and included very advanced mathematics.

This project was also the first time I had attempted to create a fairly large GUI that was not just a few buttons. I remembered using Tkinter in my first year for GUI development in Python and I could not fully understand it, so I wanted to avoid that at all costs. It took quite a while before I discovered the Qt framework, but it seemed much easier to use so I decided to learn it and apply it to this project. However, learning the syntax of PyQt and how the code interacted with Python code also took a lot longer than I had initially expected and therefore increased the development time.

I also struggled with multi-threading, as this was the first time I had properly implemented a complex multi-threaded system. Initially, I built the entire system without threads and then, when

I started merging the GUI and actual backend code together, I quickly realised I was going to have to at least have two threads, one for the GUI and one for the backend in order to stop the GUI from becoming unresponsive. This meant I had to completely redesign the code structure and added about a week to my development time. Something that I particularly got stuck with was the *'signal and slot'* mechanism in PyQt for sending data back and forth between threads. After about three days of struggling, I gave up and decided to use global variables to pass information instead. This can sometimes be dangerous and can cause variables to sometimes have unexpected values. Therefore, I had to make the GUI wait for each attack thread from finishing its processing before *'repainting'* the GUI. With large files this could cause a problem and could have the same effect as not making the system threaded in the first place.

5 Results and Evaluation

This section describes the process of testing the system and the evaluation of its performance in these tests. In the initial report, the moderator made a comment that they were unclear as to how the system would be tested. The system will be tested in three ways. Functionality testing to quantify the effectiveness of the system, click-through testing, in order to demonstrate that the system behaves as expected and finally usability testing to determine how easy the system is to use.

5.1 Functionality testing

In order to test the functionality of the system, I have devised several controlled tests. These tests are designed to assess the functional correctness of each steganalysis attack by having a set of known-natural files and known-stego files. All other methods of steganalysis will be switched off, allowing sole testing of the attack in question. This will allow me to easily see false-positives and false-negatives. I have also created tests for the other functional aspects of the system, such as the database and plugin architecture. These are simply ‘*click-through*’ tests (using the program) to check that everything is working as expected. These are also described in the section below.

5.1.1 LSB detection within images

To determine the success of the project in detecting LSB insertions in images, the system will be tested with various images downloaded from the internet through the creative commons image search, found at <https://search.creativecommons.org/>. Some of these images will then have a relatively small image embedded in them, while others will not. They will be clearly labeled as *naturalN.png* and *stegoN.png* respectively, where *N* represents the corresponding number. The image embedding will be performed with the PySteg library, also written by Robin David and

released use under the MIT license. A graph will then be plotted to show the number of correct and incorrect predictions made by the system.

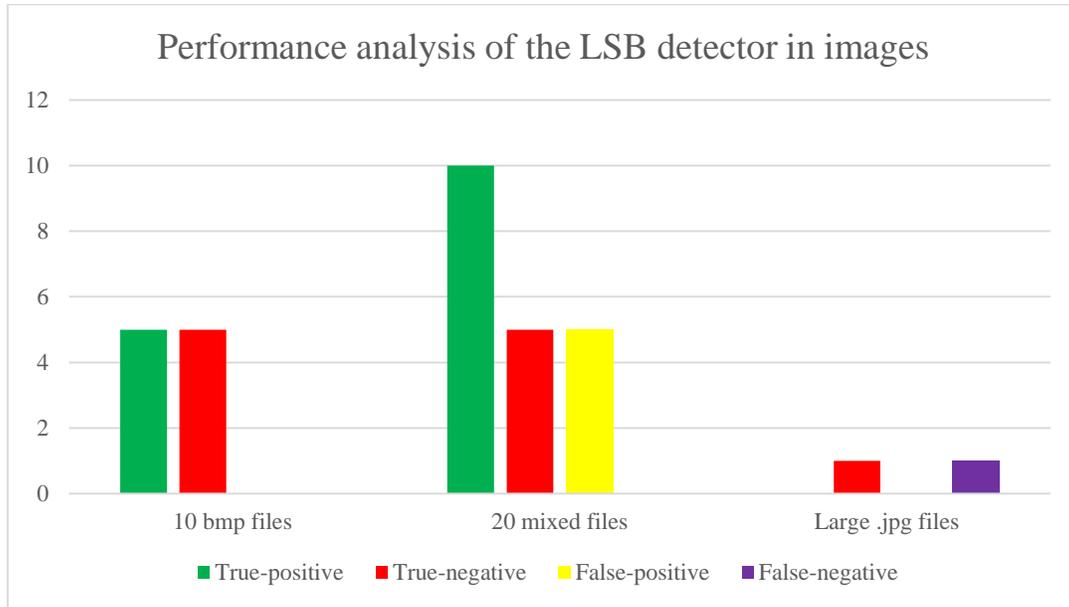


Figure 33 - Graph to show the performance of LSB detector in images

Number of files	Type of files	File size	Time to process in seconds	Highest false-positive change-rate	Lowest true-positive change-rate
10	.bmp	202kb – 1.18mb	14.4139997959	n/a	n/a
20	Various	7.93kb – 1.18mb	18.8990001678	0.2 BPP	0.33 BPP
2	.jpg	1.15 – 23.7mb	94.6979999542	n/a	n/a

With some *'tinkering'* it appears that the optimal detector rate is around 0.025 bits per pixel (BPP). If this value is changed even slightly, the correct rate of detection starts to drop. The false positives and false negative in the graph are down to the way that SPA works. If an image is particularly 'noisy' the detector tends to have a false-positive bias. Whereas if an image is rather large with a small amount of data embedded, it is likely that the detector would not pick up the stego-content.

5.1.2 LSB detection in audio files

In order to test the performance of the LSB in audio files, I am using a script written by Fred Hatful called PyHide, that allows the LSB insertion of binary data into audio files. The files with embedding will then be tested alongside their natural counterparts, in the same way that the image files were, to determine the detection rate of the system.

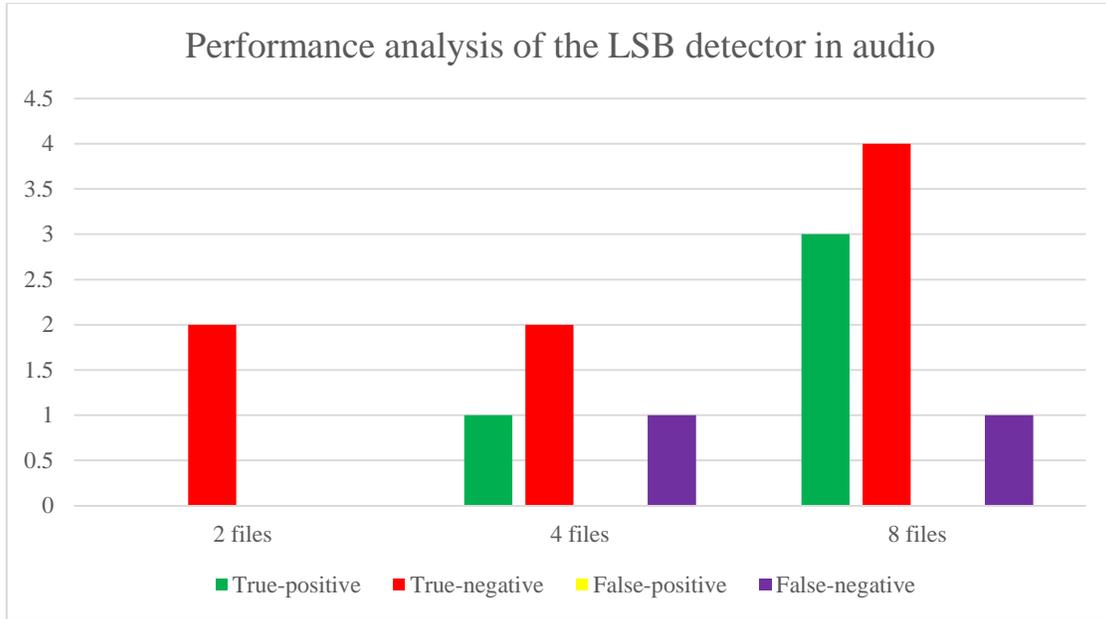


Figure 34 - Graph to show the performance of the audio LSB detector

Number of files	Type of files	File size	Time to process in seconds
2	.wav	1.34 – 1.93mb	4.73900008202
4	.wav	1.34 – 1.66mb	14.8770000935
8	.wav	1.34 – 5.88mb	40.3289999962

The optimal detector change rate seems to be around 0.06 bits per sample (BPS). More than this value and many more false-negatives arise, less than this value and many more false-positives arise.

5.1.3 SpamMimic detection

In order to test the system for how well it can detect SpamMimic documents, several SpamMimic documents will be produced, each with encoded messages of various lengths. Several

'natural spam' files will also be gathered. These files can be found in the 'spam archive' which is located at <http://www.antespam.co.uk/spam-resource/>. Again, similarly to the previous tests, these files will be tested in conjunction with the other files to see how well the detector performs.

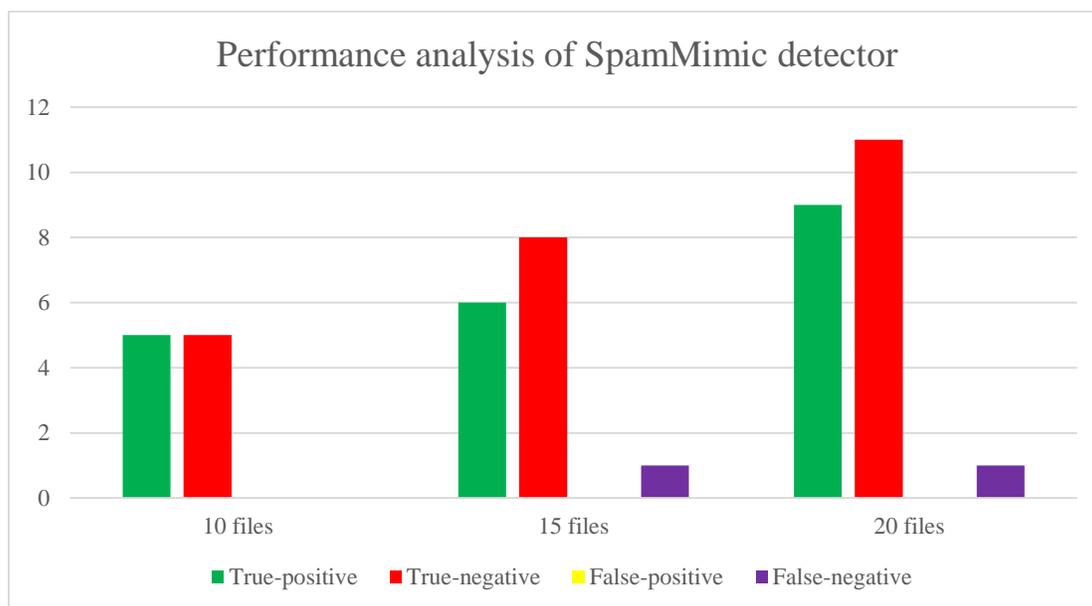


Figure 35 - Graph to show the performance of the SpamMimic detector

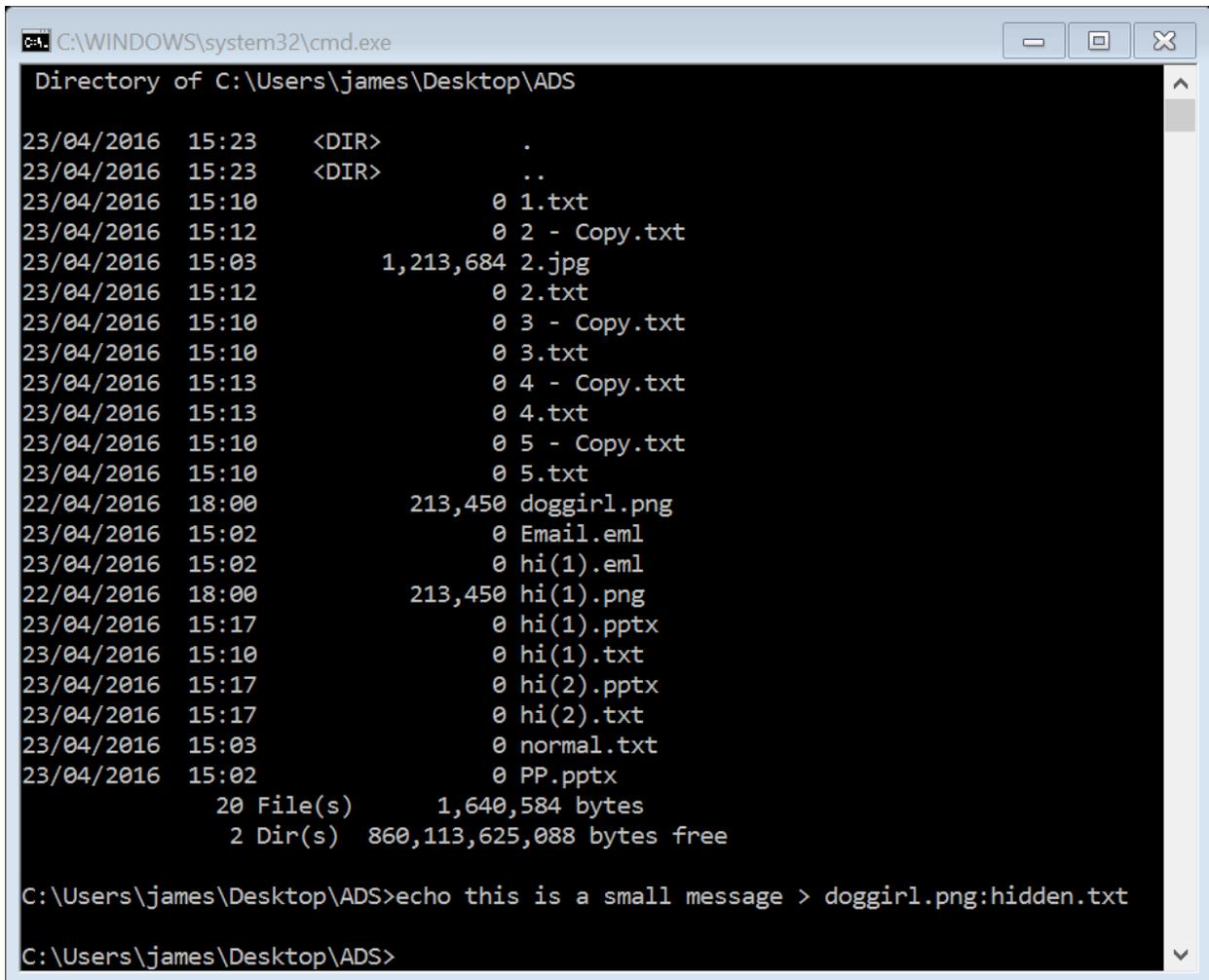
Number of files	Type of files	File size	Time to process in seconds	Message recovery	Message Encrypted?
10	.txt	2 – 6kb	2.68000006676	100%	No
15	Various	2 – 8kb	2.92100000381	85.71%	Yes
20	Various	1 – 10kb	4.00900006294	90%	Yes

The false-negative that was returned was due to the message being encrypted and, because of an error in the coding, it meant that it was trying to retrieve a message that was null causing an error and giving a false result.

5.1.4 ADS detection

To evaluate the performance of the ADS detector, as with the previous tests, all other methods of steganalysis were turned off. The system was fed a directory of files, some with ADS

and others without. All of the streams were created in the DOS prompt by simply running the following command:



```
C:\WINDOWS\system32\cmd.exe
Directory of C:\Users\james\Desktop\ADS
23/04/2016  15:23    <DIR>          .
23/04/2016  15:23    <DIR>          ..
23/04/2016  15:10                0 1.txt
23/04/2016  15:12                0 2 - Copy.txt
23/04/2016  15:03           1,213,684 2.jpg
23/04/2016  15:12                0 2.txt
23/04/2016  15:10                0 3 - Copy.txt
23/04/2016  15:10                0 3.txt
23/04/2016  15:13                0 4 - Copy.txt
23/04/2016  15:13                0 4.txt
23/04/2016  15:10                0 5 - Copy.txt
23/04/2016  15:10                0 5.txt
22/04/2016  18:00           213,450 doggirl.png
23/04/2016  15:02                0 Email.eml
23/04/2016  15:02                0 hi(1).eml
22/04/2016  18:00           213,450 hi(1).png
23/04/2016  15:17                0 hi(1).pptx
23/04/2016  15:10                0 hi(1).txt
23/04/2016  15:17                0 hi(2).pptx
23/04/2016  15:17                0 hi(2).txt
23/04/2016  15:03                0 normal.txt
23/04/2016  15:02                0 PP.pptx
                20 File(s)      1,640,584 bytes
                2 Dir(s)  860,113,625,088 bytes free

C:\Users\james\Desktop\ADS>echo this is a small message > doggirl.png:hidden.txt
C:\Users\james\Desktop\ADS>
```

Figure 36 - DOS commands for creating an ADS

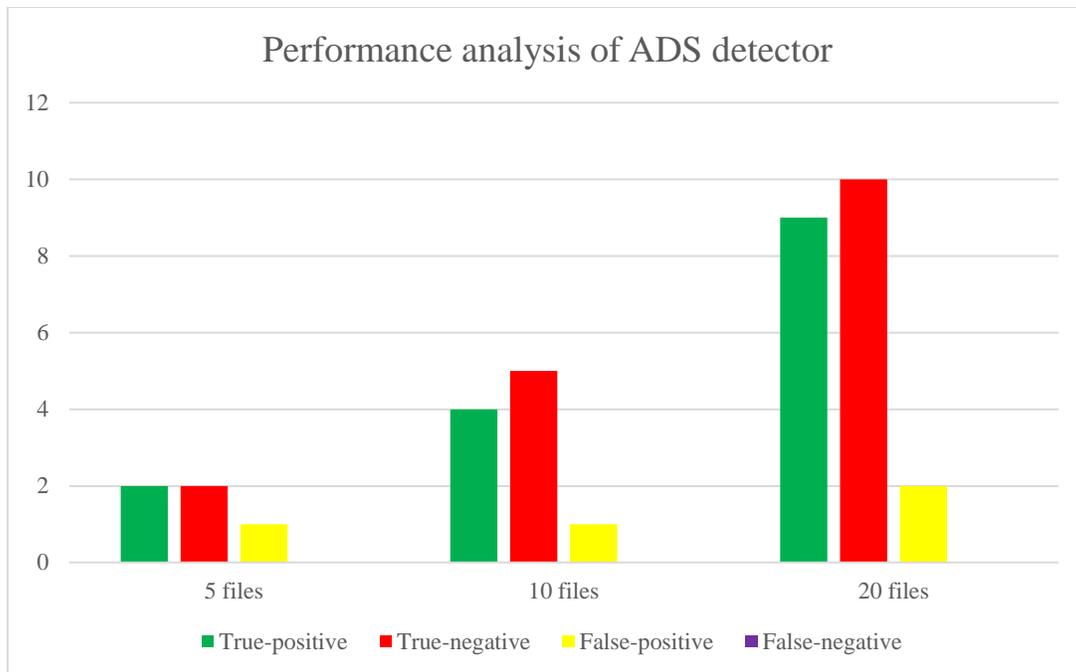


Figure 37 – Graph to show the performance of the ADS detector

Number of files	Type of files	File size	Time to process in seconds	Message recovery	False streams
5	Various	0 – 202kb	0.0320000648499	100%	Yes - 1
10	Various	0 – 202kb	0.0590000152588	100%	Yes - 2
20	Various	0 – 202kb	0.111999988556	100%	Yes - 2

The performance of the detector can be seen above. The false positives were down to the fact that email files (.eml) contain some sort of ADS used by mail programs. The ADS starts to fall down if a file is added to the stream rather than a text based message. The detector will be able to tell that ADS exists and the name of that ADS but will not be able to recover the contents.

5.1.5 Click-through testing

This section is to show that the system as a whole works as expected. This will be demonstrated with screen captures of the various aspects of the program in use and short explanations of each aspect of the program. This is down to a limitation in the PyADS library, in that it will only recover text-based messages.

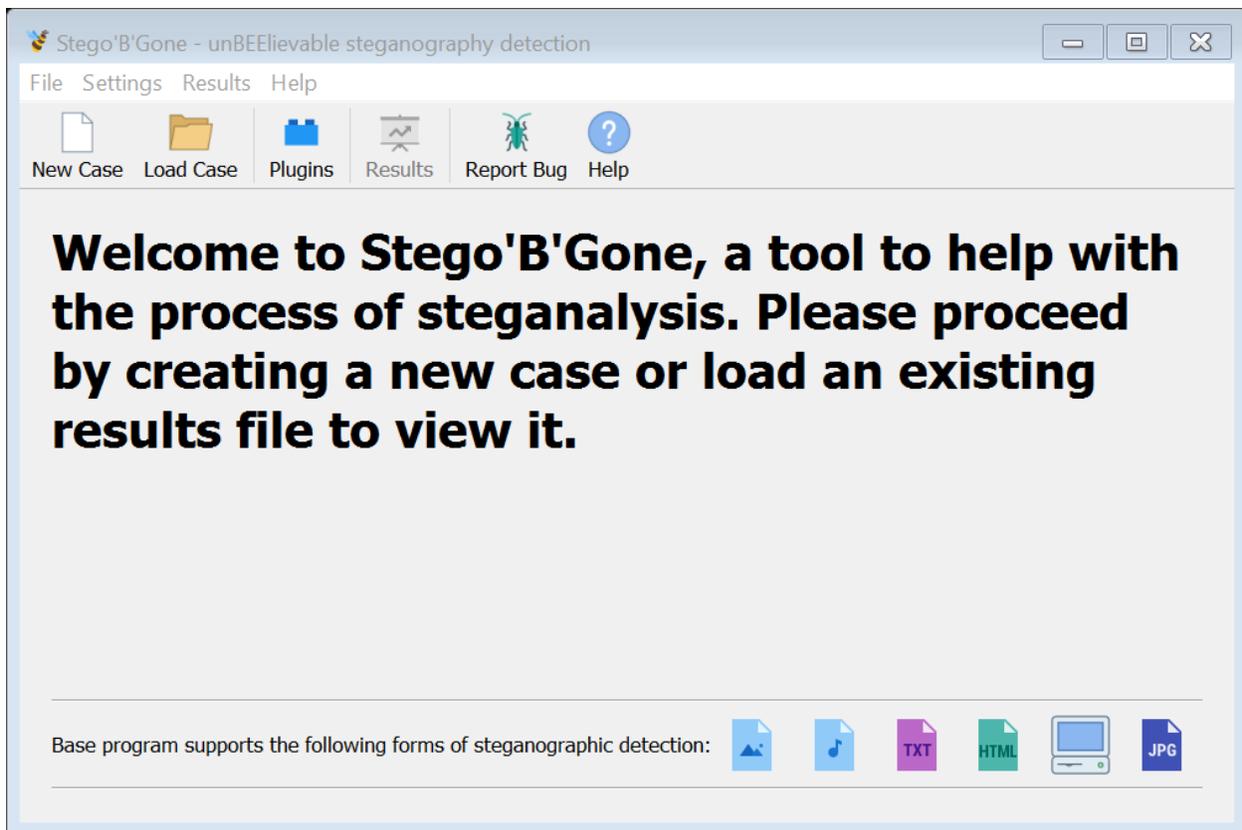


Figure 38 - Welcome screen

The welcome screen has not changed at all since it was designed and therefore requires no explanation.

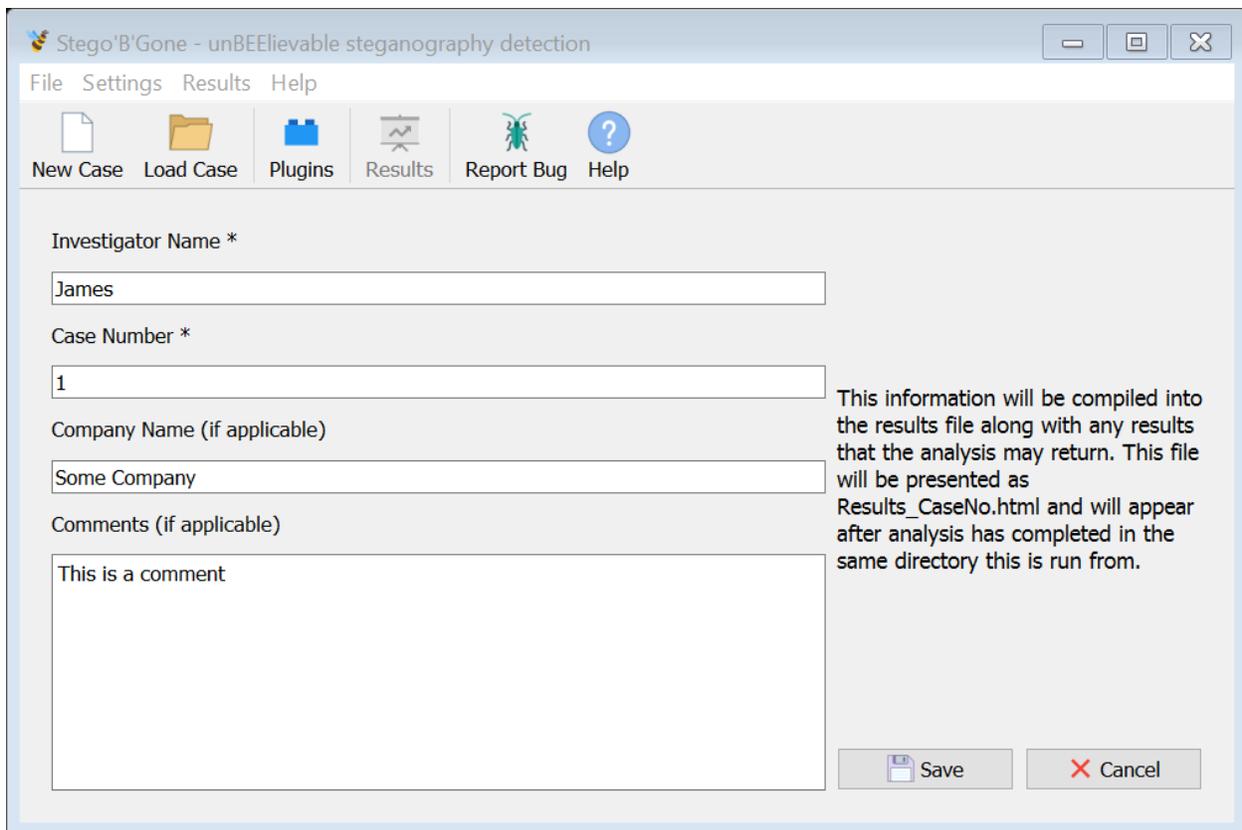


Figure 39 - New case screen (with text)

If the required fields (marked with an asterisk) are left blank when the user tries to save the case, the user receives the below error message:

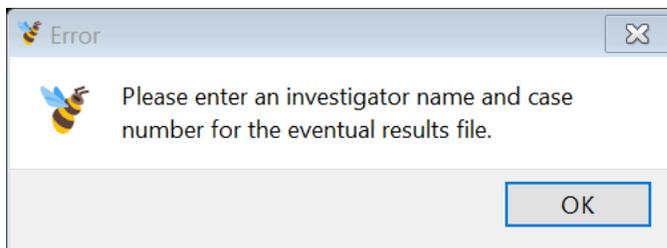


Figure 40 - New case error message

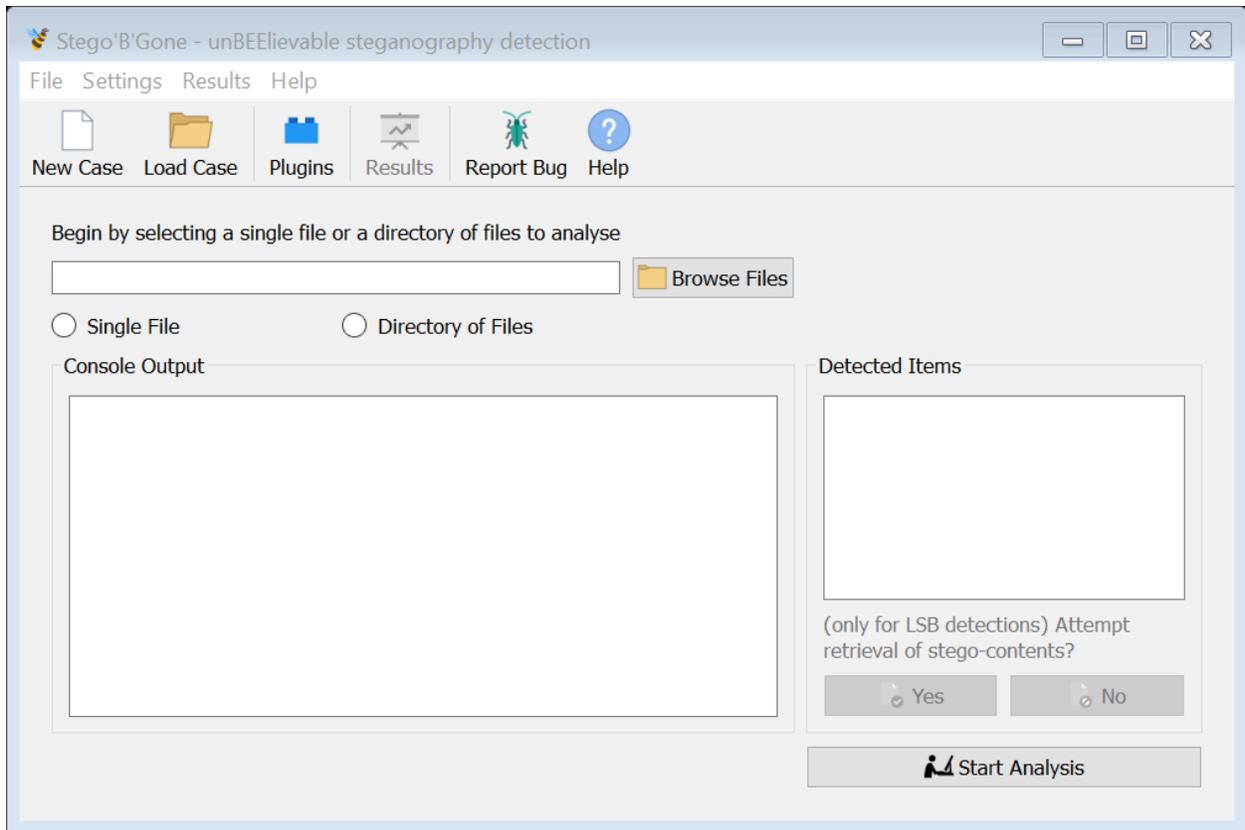


Figure 41 - Blank analysis screen

Once a new case has successfully been created, the user is presented with the analysis screen above. If the user tries to click 'Start analysis' or 'Browse files' they are presented with the following error messages.

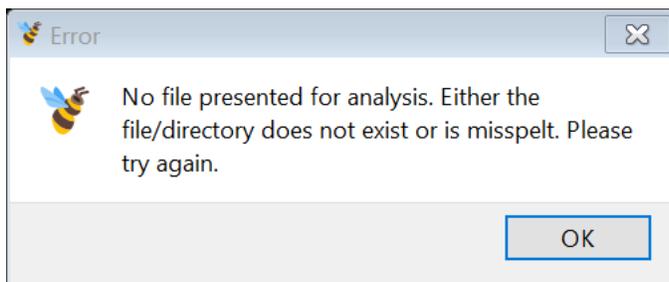


Figure 42 - Start analysis error message

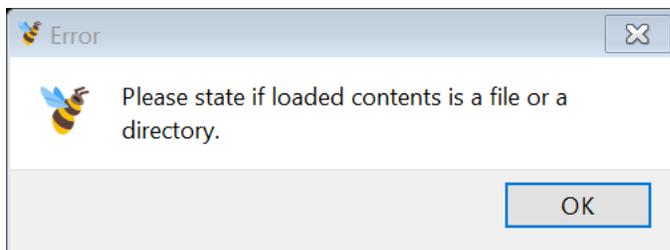


Figure 43 - Browse files error message

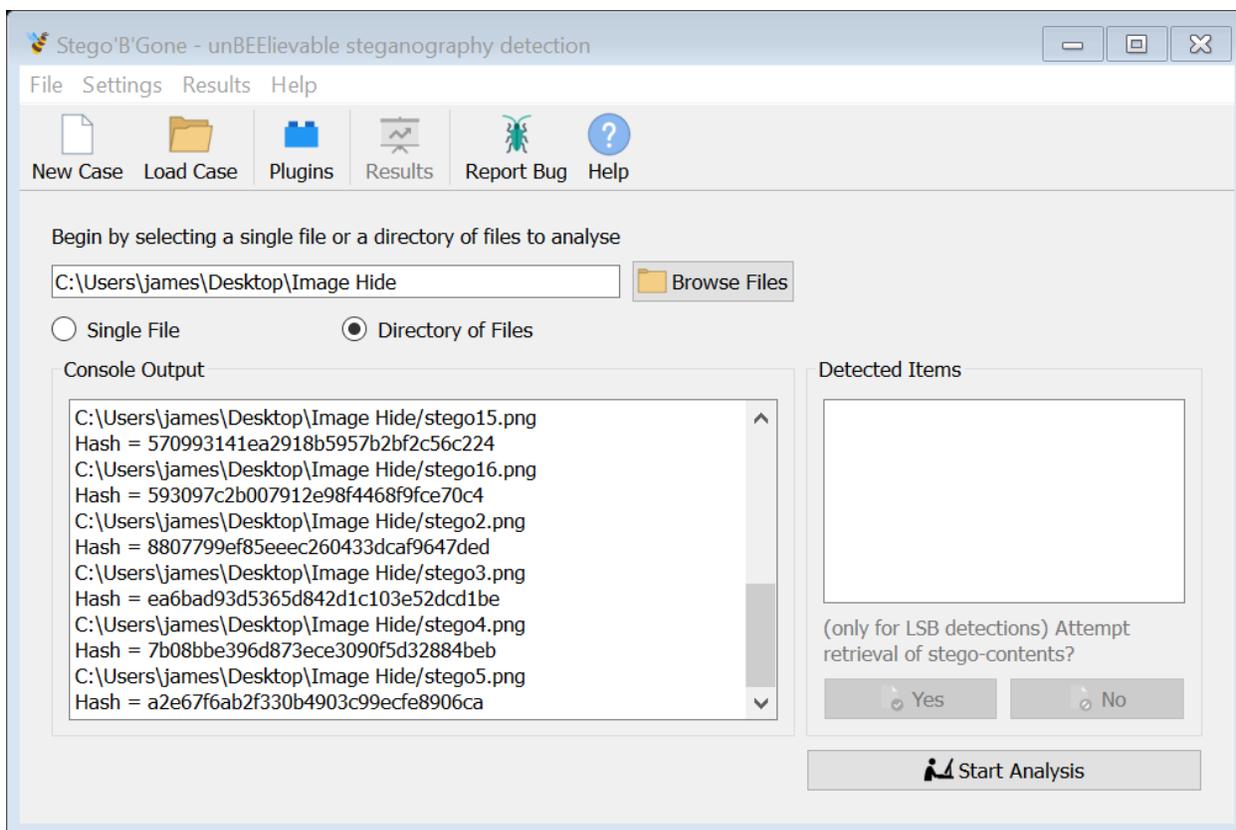


Figure 44 - Analysis screen with content loaded

The above screen shows the file names and their path, followed by the MD5 hash value of that file. It is at this point that the system looks for the database file. If the database file does not exist, it gets created. On the other hand, if the file already exists, the system searches for each of the hash values in the database. Any values that match a value in the database will be displayed in the 'Detected items' window, as can be seen below.

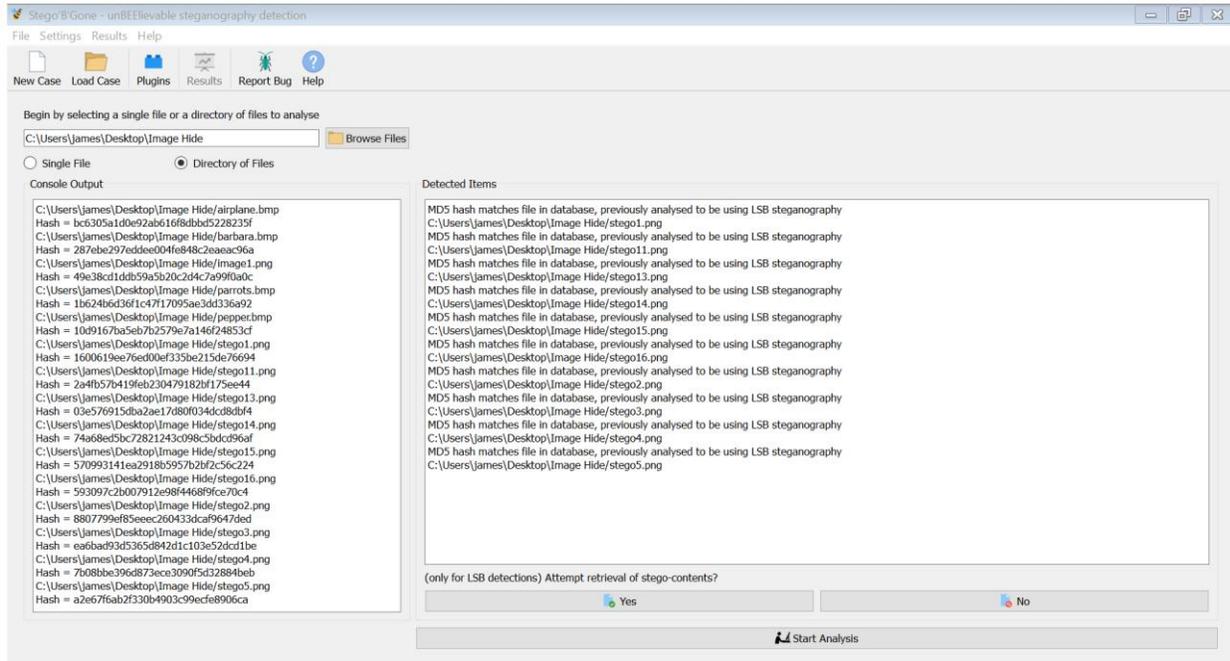


Figure 45 - Items detected from the database

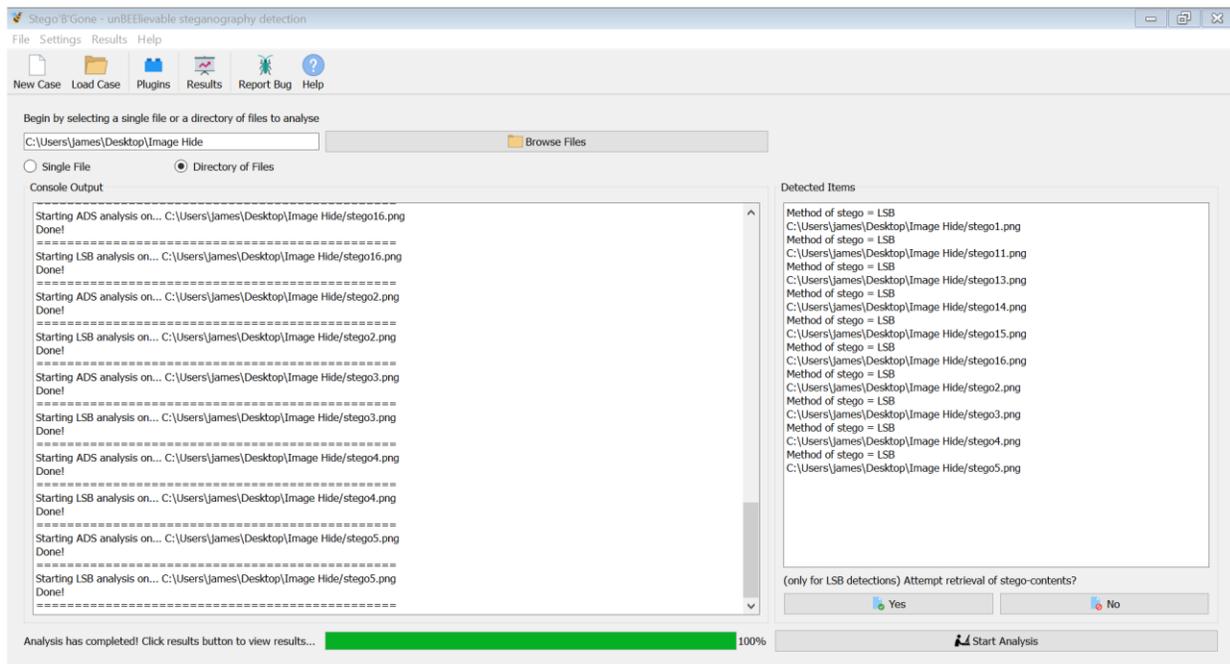


Figure 46 - Analysis screen, after analysis has completed

The above screen capture shows the analysis screen once the analysis process has completed. Notice that the 'Results' button is now active, which means that the results file can now be loaded. As the system progresses through the list of files, it also updates a progress bar to

give the user some visual feedback. Any files that are detected to contain stego-content are appended to the list of detected items and added to the database along with the method of stego detected.

If the detected items are image files and are found to contain LSB steganography, a visual attack can be launched in order to attempt the retrieval of the hidden message or image. However, this does not have a high probability of returning anything of use to the investigator, unless the image has been modified linearly to contain another image or text. If the user chooses to launch the attack, once it has completed they are presented with the following message in *figure 47*.

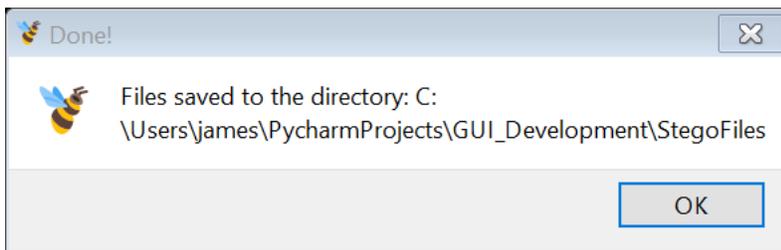


Figure 47 - Visual attack success message

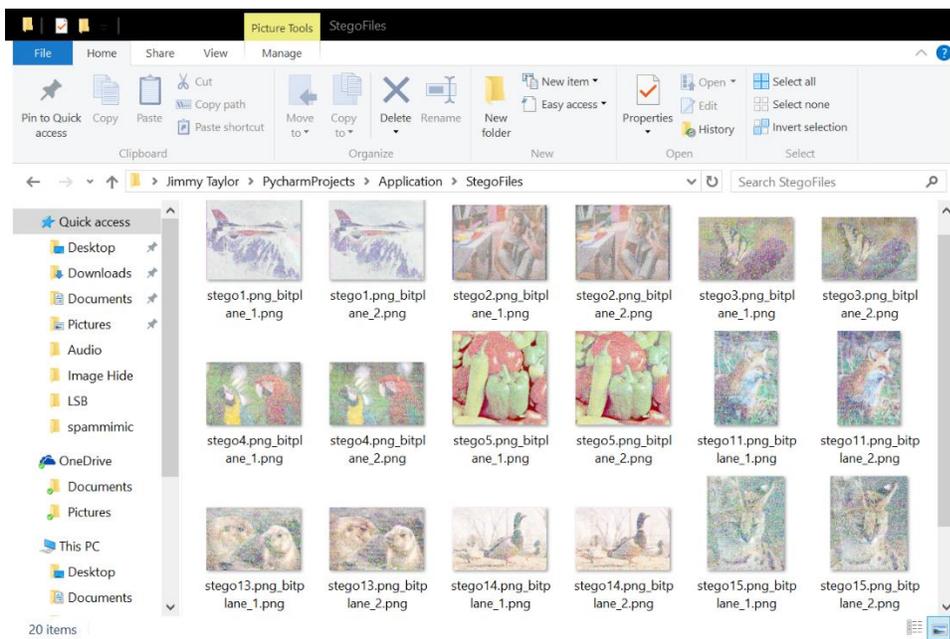


Figure 48 – 1st and 2nd bit-planes of the stego images

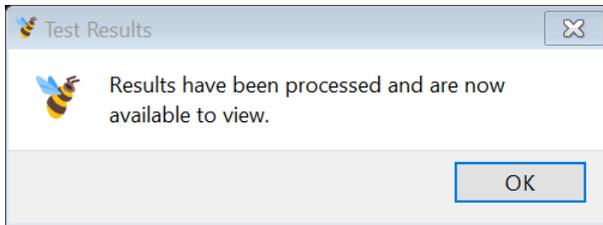


Figure 49 - Results message

Once the analysis has completed, the user can choose to view the results file in the software or view it in a browser, meaning they do not have to load the software again. Viewing the results in the software is displayed in *figure 50*. The results operate on a traffic light system for easy viewing, green for natural, red for stego and yellow for an error.

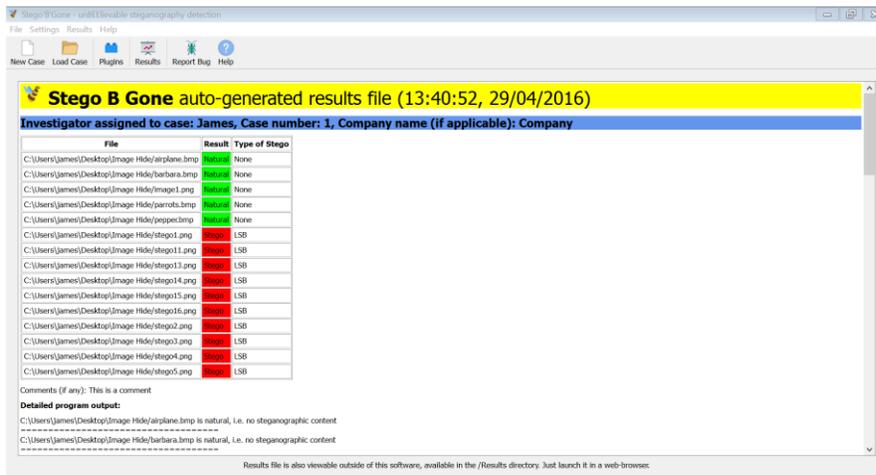


Figure 50 - Results screen

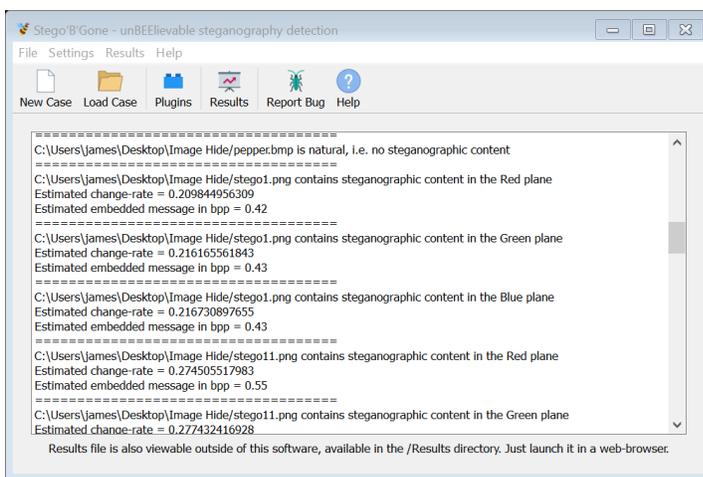


Figure 51 - Detailed program output in the results file

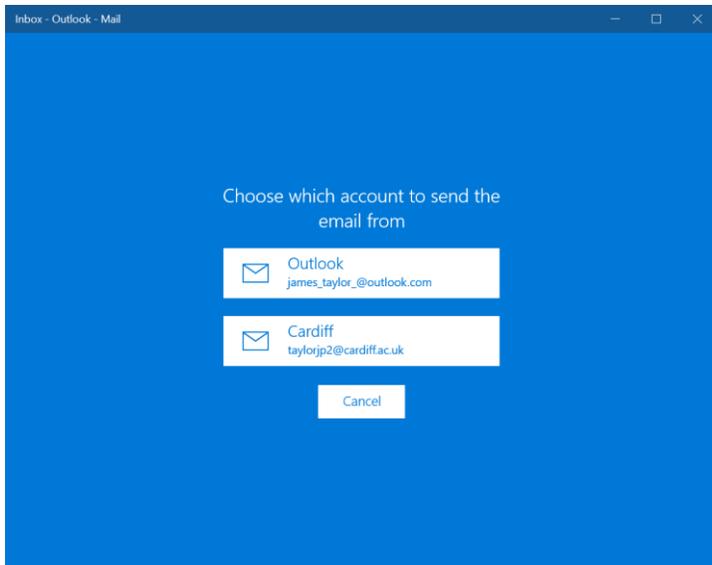


Figure 52 - Default mail client launch

When the 'Bug report' button is clicked it launches the default mail client, as can be seen from the above figure. It then creates an email with the software version as the subject and a polite message, asking the user to submit their bug, which can be seen in the figure below.

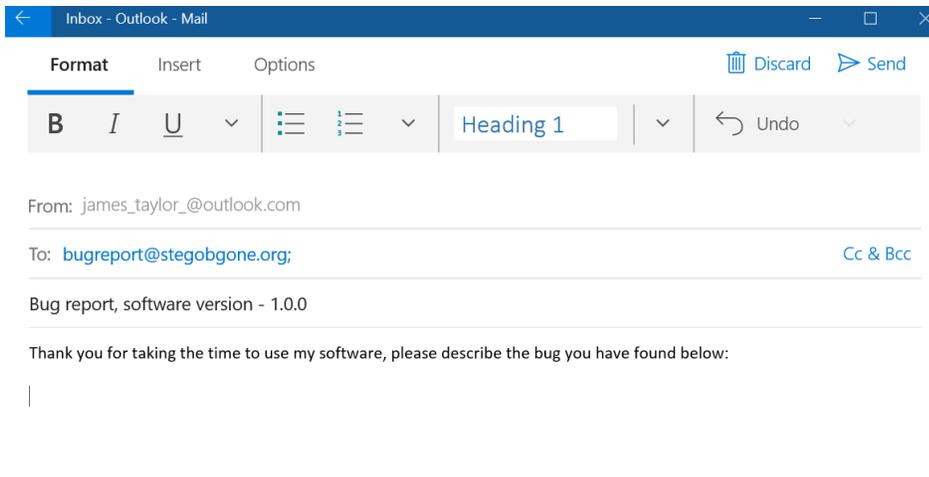


Figure 53 - Report bug email

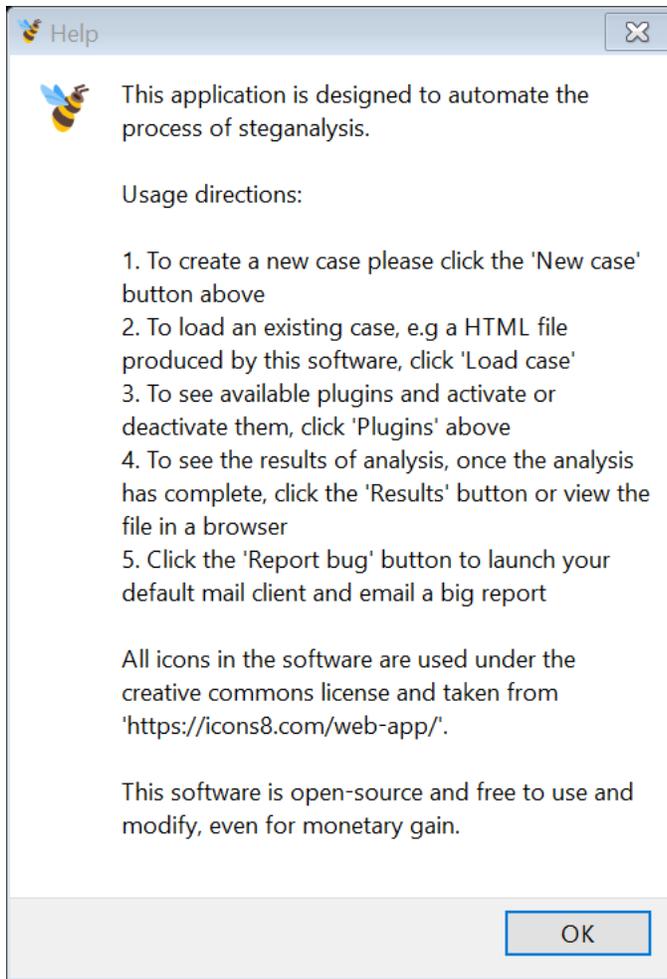


Figure 54 - Help dialogue

If the user clicks the 'Help' button they are presented with the above help dialogue. This contains simple usage directions and a link to the icons used in the software, which is required by the licence to use said icons.

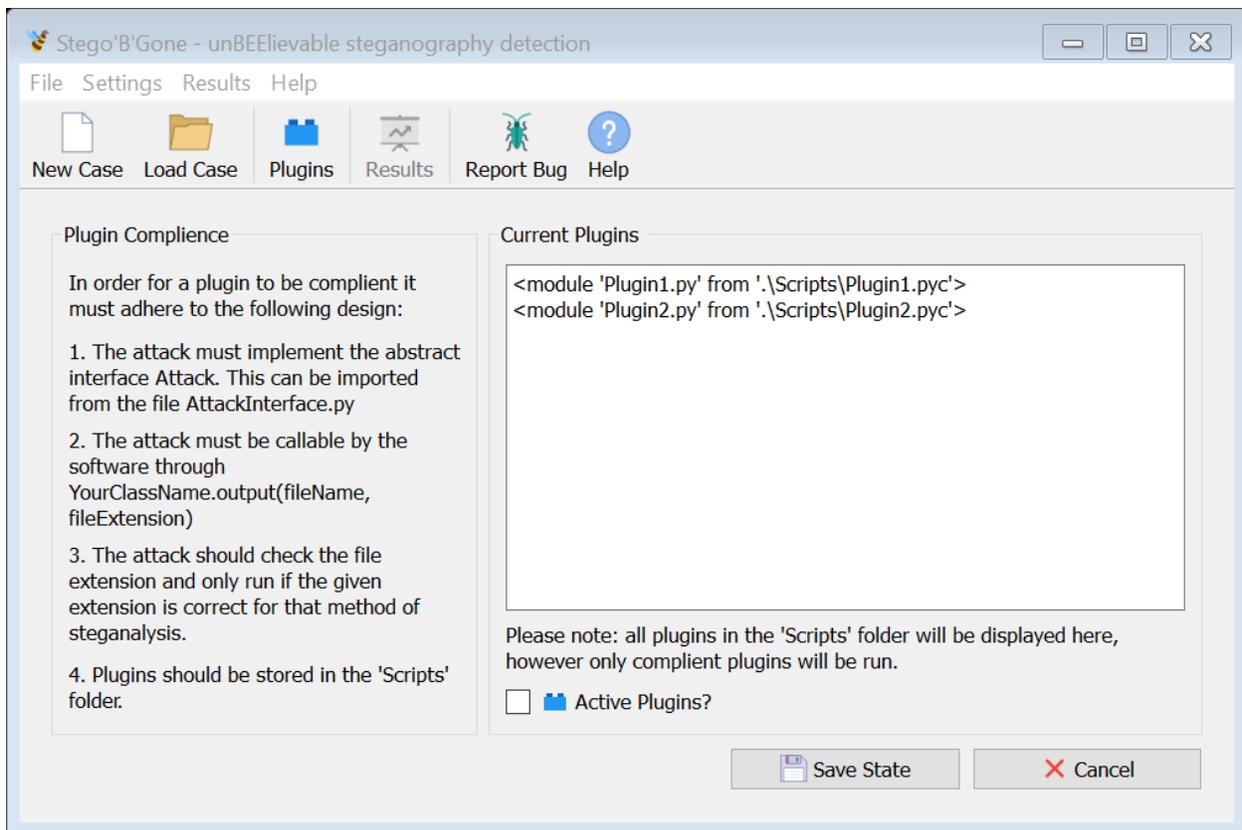


Figure 55 - Plugins page

Clicking on the 'Plugins' button brings the user to the above screen, from there they can choose to activate or deactivate plugins. This state is persistent and will remain even after the software is closed.

5.2 Usability testing

This section describes the way in which the usability of the system was tested. The system was tested by five individuals with varying levels of computer literacy, in a hands-on, task driven test. This style of test was used to demonstrate the ease of performing mission-critical tasks, as opposed to interviewing users and asking for comments on the system. Bias could not be completely eliminated in these tests as the users were either close friends or family. However, the rating system was anonymous to make the users more comfortable to give negative feedback if they felt it was necessary. Each user was then given ten minutes with the system, given only a

short explanation of steganography and steganalysis. They were then expected to perform the main tasks the system was designed to handle. These tasks were as follows:

1. Create a new case and;
 - a. Analyse a single file
 - b. Analyse a directory of files
2. Load an existing case
3. Activate plugins
4. View the help page
5. Submit a bug report

Each user was then asked to rank how easy they felt each task was on a scale of one to five.

Five being very easy and one being very difficult. The results were as follows:

Table 1 - Usability test data

Task no.	User 1	User 2	User 3	User 4	User 5	Average
1a	4	3	5	4	3	3.8
1b	4	3	5	4	3	3.8
2	5	4	5	5	4	4.6
3	5	5	5	5	4	4.8
4	5	5	5	5	4	4.8
5	5	5	5	5	4	4.8
Total						4.43333333333333

From the above table, it is clear to see that with an overall average of ~4.4 out of 5, the system has met its non-functional requirement of being very easy to use, even for someone with a basic level computer literacy. This is mainly due to the fact that the layout of the system is consistent and resembles many other pieces of software, with the familiar menu-bar and tool-bar structure.

5.3 Evaluation of system requirements

The following section will discuss if the system has met its specified requirements and if so, it will also explain to what extent it has met these requirements.

5.3.1 Functional requirements

Requirement 1: The system should be able to detect the use of steganography in files. This requirement has indeed been met since the system is able to detect three different types of steganography in four different cover mediums.

Requirement 2: The system should be able to retrieve hidden content where possible. This requirement has been met as the system can automatically retrieve the contents of both ADS and SpamMimic documents once they have been detected. The system also has a method for extracting the LSB planes from images with the visual attack. However, as described in the implementation, this attack is only viable if the information has been stored linearly.

Requirement 3: The system should be extensible. This requirement has been met as the system operates a plugin based architecture enabling the expansion of its base steganalysis techniques.

Requirement 4: The system should have a GUI. The system has a well-developed, easy to use GUI and therefore meets this requirement.

Requirement 5: The system should create and maintain a database of MD5 hashes of files found to contain steganographic content. This requirement has been met as the system creates a local database if one does not already exist. Nevertheless, if one does already exist, once a file has been loaded in, the system checks the MD5 of the file against the files in the database. If there is a match that file is flagged as steganographic before any analysis has taken place.

5.3.2 Non-functional requirements

Requirement 1: The system should maintain a high degree of usability. This requirement has been met according to the usability testing carried out by five anonymous users of varying levels of computer literacy.

Requirement 2: The system should be measurably testable. This requirement has been met as can be seen from the functionality testing. These tests show that the testability of the system is measurable, as the system has a visual outcome for input and that outcome (or deliverable) can be deemed correct or incorrect.

6 Future Work

Throughout the course of the project, my ideas have grown and developed way beyond anything I could have hoped to achieve in the available time. Given a longer timeframe and using the knowledge acquired over the course of the project, I feel that I could improve the final system in many ways.

The first and most obvious way to improve the system would be to gather or create more targeted attacks against various steganographic systems and implement them. Some of the steganographic techniques that could be exploited in the future might be; discrete cosine transform (DCT) embedding, GIF pallet embedding, discrete wavelet transform embedding, echo hiding and EOF steganography.

DCT embedding in the transform-domain of JPEG images from steganography programs such as JSteg, JPHide and OutGuess is a very popular method of steganography as JPEG images are so widely used. This method of steganography follows the LSB insertion method but instead of embedding into the image directly, the data is embedded into the DCT of the JPG coefficients. A detector of this type of steganography could be built by utilising the quantitative attack described by Jessica Fridrich (Fridrich, 2010).

Another targeted attack that could be implemented against image files (GIF files specifically) would be an attack against GIF pallet embedding by programs such as GIFShuffle. This method again follows the same principle of LSB insertion but this time into the indices of sorted colour palette of the images. GIFShuffle relies on the fact that reordering the colours in the colour map of a GIF file does not affect the way the image is displayed (Kwan, 2010). At the time of writing, I could not find any detection methods for this type of steganography.

There are also several more complex methods of steganography also available for audio other than LSB embedding. The first of which is similar to the DCT embedding in JPG images. It involves embedding into the LSB of the discrete wavelet transform (DWT) coefficients of an audio signal. There have been attacks proposed against this method of steganography, although, they were far too complicated for the timeframe of this project. One such attack is based on the principle component analysis of statistical moments (Fu, et al., 2007).

Another method of audio steganography is echo hiding. This involves adding a short echo to the cover signal, whose parameters can be manipulated to conceal data. The initial amplitude, offset and decay rate of the signal could be modified so it is not audible by the human ear but can contain small amounts of information. Again, there are attacks against this method but they are also extremely complex and require an advanced knowledge of signal processing. The attack in question is based on the principle of sliding windowed cepstrum, which is the result of taking the Inverse Fourier Transform (IFT) of the logarithm of the estimated spectrum of a signal (Xie, et al., 2011).

A much simpler form of steganalysis that could be added to the system would be a detector of '*end of file*' (EOF) based steganography by programs such as AppendX or Camouflage. This method would simply require a list of EOF hex markers for different file types. The detector would then flag the file as potentially steganographic if any data existed beyond the file's EOF marker.

The possibilities for future development in targeted attacks are almost infinite, however there are other ways in which the system could be improved. One such way would be to host a database of found steganographic files on the internet, instead of locally. This would create a much larger base of known files and could potentially become highly useful to law enforcement agencies. This is because files with embedded steganographic content are often passed around. Another way

the system itself could also be improved is to connect to a database of plugins, similar to the previously discussed internet hosted database containing the hash values of detected files. This would allow users to pick from a growing list of steganalysis techniques for their specific purpose. The system could also allow users to upload plugins that they have written to the database for use by others in the community. This functionality could also include a rating system, where the highest ranked steganalysis techniques are given a higher place in the list of plugins.

7 Conclusion

This section is a short summary of the overall project, if it has met the initial goals that were set and to what extent these aims have been met.

This project has successfully delivered a piece of fully functional software that focuses purely on steganalysis. It also allows the automatic detection and retrieval (when possible) of steganographic content from within various file types. However, due to my keen interest but lack of knowledge in the subject, the scope of the project in the initial report was to have a bank of between five and seven steganalysis techniques for various types of steganography. I quickly realised that this objective was unrealistic, as the sheer amount of time it takes to research some of these techniques, let alone implement them, would mean that I would have definitely run over the timeframe. Instead, I aimed to have at least one method of steganalysis for the four main cover mediums being text, audio, images and file system.

The development stage of the project went very well in general. However, there were some steep learning curves when it came to certain aspects of the program. Namely learning PyQt, creating a threaded application, and the research involved in each method of steganalysis. However, these were all overcome and I feel that I have developed professionally because of these difficulties.

As can be seen in the results section, the project has met all the proposed requirements; it can detect several forms of steganography in the most often used cover media (images, text, audio and file system). Each of the detectors works very well, with only a small number of false alarms. The system is also extensible and allows the addition and persistent activation of plugins. Furthermore, it has a database in an attempt to not analyse files multiple times and also has a user-friendly, simplistic GUI as can be seen from the usability testing in the results section.

In conclusion, the project has produced a piece of software that has met all in the initial aims and can be built upon in the future by adding new methods of steganalysis, due to the extensible design. It is because of this that I believe this project has been a success.

8 Reflection on learning

This section details the impact that this project has had on my overall development. It details what I have learned, in terms of both soft (transferable) and hard (technical) skills.

In terms of the scope of this project, it is the largest, most complex and ambitious project I have ever undertaken. It has stretched me technically, creatively and emotionally. The subject matter is incredibly interesting and the applications of steganalysis and steganography in the real world are exciting. If I had narrowed the scope and focused on a particular area to attack, such as images, it would have allowed me to get a deeper understanding of that particular area of steganography. However, as much of the research is on images alone, attempting to collate different attacks for different methods into one unified system gave me a niche area to work in. This also gave a fairly good overview of steganography and steganalysis in general.

The technical skills I have gained from completing this project have enhanced my professional development greatly. I have developed a deep knowledge of, and passion for, Python and the *'Pythonic'* way of writing code, where simplicity is key. I learned a lot about image processing, through the LSB steganalysis and I also got to try my hand at machine learning to develop a detector for SpamMimic. Over the course of the project, I have had a great level of exposure to many different file-types and their construction, as well as areas in which information can be hidden.

I have also gained several transferable skills from undertaking this project. A comment brought up by the moderator in my initial report was that my time-plan should have been more *'fine grained'* and I whole heartedly agree with this, as this project really stretched my time management skills. For instance, in my initial plan I had allocated a week to the entire development

of the GUI and in reality each aspect of the GUI development took nearly a week to complete and really ate into my steganalysis development time. Other transferable skills I have improved during this project are my communications and project management skills. Over the course of the project I had bi-weekly meetings with my supervisor. This allowed me to discuss my progress and the next course of action at each step of the project along the way. I tried to treat these meetings as *'iterations'* from the agile methodology, providing my supervisor with a description of what I had implemented and then discuss where we thought it should go from there. This means that if I ever went off-track, I could quickly redirect the project.

I feel that the experience I have gained from this project can be carried on to other projects in the future and on to my professional career. I have learned from the mistakes I made in this project and I know how I can improve upon these in the future. One of the many examples would be keeping weekly or even daily logs of changes I had made to the system. This would make for much easier documentation writing, as trying to recall things retrospectively is often extremely time consuming and difficult. Another would be to focus more time and energy on the planning stages of a large project such as this. Not just dive right in, start coding and hope for the best, which would be my normal methodology.

Overall, this project has taught me a great deal. It has given me project management skills and greatly improved my programming ability. It has also peaked my interest in the fields of steganography and steganalysis alike. Giving me the knowledge and opportunity to delve deeper into these fields. Completing a security-focused project has also solidified my choice of pursuing a career in the security industry, as I find the field both challenging and interesting.

9 References

Anon., 2008. *Agile Methodology*. [Online]

Available at: <http://agilemethodology.org/>

[Accessed 18 4 2016].

Clarke, D., 2004. *Technology and Terrorism*. s.l.:Transaction Publishers.

Fridrich, J., 2010. Quantitative attack on JSteg. In: *Steganography in Digital Media: Principles, Algorithms and Applications*. s.l.:Cambridge University Press, pp. 66-68.

Fridrich, J., 2010. Selected targeted attacks. In: *Steganography in Digital Media: Principles, Algorithms and Applications*. s.l.:s.n., p. 226.

Fridrich, J., 2010. *Steganography in Digital Media: Principles, Algorithms and Applications*. Cambridge: Cambridge University Press.

Fridrich, J., Goljan, M. & Du, R., 2008. *Reliable Detection of LSB Steganography in Color and Grayscale Images*. [Online]

Available at: http://parsys.informatik.uni-oldenburg.de/~stego/workshopStatistik/g1/RS_Steganalysis.pdf

[Accessed 19 04 2016].

Fu, J.-W., QI, Y.-C. & Yuan, J.-S., 2007. *Wavelet domain audio steganalysis based on statistical moments and PCA*. Beijing, International Conferance of Wavelet Analysis and Pattern Recognition.

Hayati, P., Potdar, V. & Chang, E., 2008. *pedramhayati.com*. [Online]

Available at:

http://www.pedramhayati.com/images/docs/survey_of_steganography_and_steganalytic_t

ools.pdf

[Accessed 13 04 2016].

Johnson, N. F., Duric, Z. & Jajodia, S., 2001. *Information Hiding: Steganography and Watermarking - Attacks and Countermeasures*. s.l.:Kluwer Academic Publishers.

Kessler, G. C., 2004. An Overview of Steganography for the Computer Forensics Examiner. *Forensic Science Communications*, 6(3).

Kwan, M., 2010. *darkside.com.au*. [Online]

Available at: <http://www.darkside.com.au/gifshuffle/description.html>

[Accessed 21 04 2016].

McCullagh, D., 2001. *Secret Messages Come in .Wavs*, s.l.: Wired Magazine.

Open Source, 2016. *opensource.com*. [Online]

Available at: <https://opensource.com/resources/what-open-source>

[Accessed 14 04 2016].

Orsdemir, A., Altun, H. O., Sharma, G. & Bocko, M. F., 2008. *University of Rochester*. [Online]

Available at:

http://www.ece.rochester.edu/~gsharma/papers/OrsdemirSteganoAwareStegoEI2008_6819_41.pdf

[Accessed 13 04 2016].

Petitcolas, F. A. P. & Katzenbeisser, S., 2000. *Information Hiding: Techniques for Steganography and Digital Watermarking*. Boston: Artech House.

Qt Framework, 2016. *qt*. [Online]

Available at: www.qt.io/qt-framework/

[Accessed 16 04 2016].

Richer, P., 2003. *SANS Institute*. [Online]

Available at: <https://www.sans.org/reading-room/whitepapers/steganography/steganalysis-detecting-hidden-information-computer-forensic-analysis-1014>

[Accessed 13 04 2016].

Sanderson Forensics, 2006. *sandersonforensics.com*. [Online]

Available at: <http://www.sandersonforensics.com/Files/ZoneIdentifier.pdf>

[Accessed 20 04 2016].

Siper, A., Farley, R. & Lombardo, C., 2005. *Pace University*. [Online]

Available at: <http://www.csis.pace.edu/~ctappert/srd2005/d1.pdf>

[Accessed 11 04 2016].

The National Archives, 2000. *legislation.gov*. [Online]

Available at: <http://www.legislation.gov.uk/ukpga/2000/23/part/III>

[Accessed 13 04 2016].

The U.S. National Archives and Records Administration, 2016. *National Archives*. [Online]

Available at: <https://www.archives.gov/exhibits/eyewitness/html.php?section=8>

[Accessed 11 04 2016].

Wayner, P., 1992. MIMIC FUNCTIONS. *Cryptologia*, 16(3), pp. 193-214.

Westfeld, A. & Pfitzmann, A., 2006. *ece.cmu.edu*. [Online]

Available at: <https://users.ece.cmu.edu/~adrian/487-s06/westfeld-pfitzmann-ihw99.pdf>

[Accessed 14 04 2016].

Xie, C., Cheng, Y. & Chen, Y., 2011. An active steganalysis approach for echo hiding based on Sliding Windowed Cepstrum. *Signal Processing*, 91(4), pp. 877-889.

