

# Steganography Lab

## Theoretical Aid Document

### Introduction

The practice of steganography predates computers by quite a while. Hiding messages or information within unsuspecting media or physical objects in order to avoid prying eyes seems to be innate to human nature. The dawn of computing and digital media brought forth a new dimension to steganography, as many new different media formats presented themselves as possible vessels for information, be it benign or malign.

Steganography is important to study in cybersecurity, because it is a form of security through obscurity. It is not encryption; it does not hide the meaning of information, it hides its existence. Its main advantage over cryptography, especially when paired with it, is that cryptographed information, while secure, may attract unwanted attention, or even be downright illegal in certain countries. By concealing information in files which often go unattended the information effectively stops being an object of scrutiny. While this may have valid uses, it is important to realize that it is also a very commonplace practice for hiding malware or other types of illegal information.

In this document, which accompanies the set of tasks for this lab, we discuss some theoretical principles of steganography in hopes to aid you in your learning experience.

### Digital steganography & file formats

Digital steganography and file formats go hand in hand. Digital files present themselves as prime targets for steganography, both due to their complexity and to the variety of standards that govern them. Most of the media formats, be they audio, video, images, ..., are simply blobs of binary data that would be illegible to the human eye were it not for the standards that govern them.

File encodings provide structure and meaning to these blobs in the form of byte sequences colloquially referred to as "magic bytes". These encode various metadata about the files and tell applications how to parse them. They are flexible enough to accommodate large quantities of data and metadata, and also flexible enough that the data inside them can be modified cleverly and covertly in order to contain information with a meaning altogether different from the carrier media.

## The JPEG and PNG file formats

The JPEG format, which you will first encounter when solving task 1 of the lab, is a commonly used method of lossy compression for digital images. It shares its name with the authoring committee of the standard, the Joint Photographic Experts Group, and has been the most widely used image compression standards ever since its introduction in 1992. Files encoded in this standard commonly have the .jpg or .jpeg file extension.

More accurately, JPEG is somewhat of an umbrella term, as there exist a couple file formats such as JPEG/JFIF and JPEG/Exif that implement this standard for compression. The MIME type for JPEG is `image/jpeg`, and the actual file format of the `challenge.jpg` you will find in the lab is JPEG/JFIF (JPEG File Interchange Format), a slightly cut-down version of the JIF (JPEG Interchange Format) originally defined by the committee.

For this lab, it is important that you recognize a few of the JFIF magic bytes:

- `FF D8` -> Start of Image (SOI)
- `FF E0 s1 s2 4A 46 49 46 00 ...` - JFIF-APP0, includes "JFIF" encoded in ASCII, terminated by a NULL byte (`s1 s2` are segment length bytes)
- `FF DA` -> Start of Scan, actual image comes after this
- `FF D9` -> End of Image

The PNG (Portable Network Graphics) format is another extremely common image file format you will encounter throughout this lab. Unlike JPEG, PNG uses lossless compression though, meaning that image data is preserved exactly during compression and decompression. This makes PNG especially suitable for tasks requiring pixel-perfect reconstruction, such as diagrams, digital art and, yes, steganographic embedding techniques based on direct pixel manipulation.

The PNG format was introduced in the 90s as an open alternative to the GIF format and has since become another of the most widely used image formats on the web. Files encoded in this format commonly use the .png file extension, and the MIME type for PNG images is `image/png`.

As for the internal structure, PNG files are organized as a sequence of structured data blocks known as chunks. Each chunk contains a length field, a chunk type identifier, the chunk data itself and a CRC (Cyclic Redundancy Check) checksum for integrity guarantees.

Unlike JPEG, PNG files do not use neat marker sequences beginning with the FF byte. Instead the format is strongly structured around these “chunks”. For this lab, you should have these magic bytes in mind:

- `89 50 4E 47 0D 0A 1A 0A` -> PNG file signature/header, contains the "PNG" ASCII string

That 's it. Other chunks are not really critical for the purposes of this lab, but you can easily consult them in the Wikipedia page for PNG.

Because PNG compression is lossless, modifications to individual bits can be preserved exactly when the image is saved/exported. This makes PNG files particularly suitable for least significant bit (LSB) steganography, which you will explore later in this lab.

## Binary analysis tools

For the forensic analyst looking to examine a file's internals, simple media visualizers will not suffice. You must use specialized binary file inspection tools. We bundle a few of them in the lab environment, which should be more than enough for the purposes of the tasks you will have to perform. They are command line interface tools that are documented and easy to use. When in doubt, you should consult the manual or the help text of the commands (appending `-h` or `--help` usually does the trick), but we provide a brief description of them here.

The `file` utility attempts to identify the type of a file by inspecting its contents rather than relying solely on its file extension. It uses a database of known file signatures and magic bytes to determine the most likely format of a file. You will find that for steganographed content, it does not disclose the full meaning of the file - emphasis on **most likely**.

`exiftool` is a more powerful metadata extraction utility capable of reading metadata from a wide variety of media and document formats, including JPEG and PNG, of course. It can display, among others, image dimensions, MIME types, embedded metadata, timestamps, camera information, and countless structured file attributes. While very useful, metadata inspection alone is often insufficient for detecting hidden payloads.

The `xxd` command displays the raw binary contents of a file as hexadecimal types alongside their ASCII representation. This makes it a very powerful asset when searching for magic bytes, file signatures, embedded data, or suspicious trailing content. The ASCII strings are also a nice-to-have since you can search for them if you pipe the output into a pager.

`hexdump` is very similar to `xxd`, converting binary data into a human-readable hexadecimal representation. Different formatting options can make it useful for low-level binary inspection and debugging.

`binwalk` is a much more powerful forensic analysis tool designed to identify and optionally extract embedded files and executable code from binary blobs and firmware images. It scans files for known magic bytes and reports the offsets at which they occur. It can trivialize the challenge a bit, so we recommend experimenting with the other utilities first, especially since fancy tools should not replace careful manual inspection and understanding of the underlying file structures.

The `dd` command utility is a low-level copying tool capable of operating directly on raw byte streams instead of simple files and directories. It has its uses both in digital forensics as in burning images into USBs and a bunch of other problems. You can use it to manually "carve" out embedded files from larger binary containers.

## Steganography fundamentals

Most steganographic procedures consist of two main phases, the embedding and the recovery or extraction.

In the embedding, you take a payload and a carrier medium, apply some embedding algorithm, which can range from simple concatenation to more complex techniques such as LSB encoding or modifying the echo of a sound file (Echo Steganography). The resulting modified carrier containing the hidden information can be referred to as the stego-object.

In the extraction, some algorithm is applied such that it can "reverse" the operations of the original algorithm and extract the payload.

## Notes on capacity, robustness, stealth and algorithm complexity

Steganographic systems will often struggle to balance three competing inherent properties: capacity, robustness and stealth.

Think of capacity as the amount of data that can be hidden within the carrier medium. The capacity depends not only on the carrier's size, but also on the algorithm used for embedding. You will understand this better once you deal with LSB encoding, but simply concatenating an image after another would provide you

with essentially "unlimited" capacity (you will just increase the file size), while encoding it into the already present and limited bits of the carrier would obviously limit your capacity to the available data you can modify without drastically altering the carrier.

Besides keeping the payload within the limits of the carrier, you must also think about the ability of the hidden payload to survive recompression, resizing, filtering and other medium-specific transformations that may be applied to its carrier. Generally speaking, it would be nigh impossible to truly safeguard it against everything but some robust algorithms are highly resilient to this, while others are not.

Stealth is also pretty self-explanatory and is the degree to which the modifications remain undetectable to human observers, forensic analysis or automated detection tools.

Improving one of these properties will probably worsen another, as increasing capacity reduces stealth and robustness will require noticeable modifications and intricate algorithms.

## Stegomalware

Steganography is obviously not limited to benign uses. The craftiness of the methods and the payoff of being undetected appeals to many adversaries who often make use of steganographic techniques to conceal their malicious payloads, configuration files, command-and-control data, among other malware "paraphernalia". This practice is sometimes referred to as *stegomalware*, which is the term we use in this lab.

Examples of stegomalware include:

- malware payloads hidden inside images
- malicious scripts embedded in documents
- command-and-control instructions concealed in social media images
- staged malware delivery systems that retrieve hidden payloads remotely

The advantage of these techniques over simple plain hardware is to make use of the covertness of steganography in order to evade superficial inspection, bypass basic filtering systems, reduce suspicion, or even distribute various different components of malware in separate pieces of media, which are less suspicious than its whole.

Real life examples of stegomalware include:

- Operation Shady RAT - used digital images to hide CnC server addresses
- Zeus/Zbot - masked banking configuration data inside JPEG files exploited via malvertising
- Cerber - concealed executable ransomware code in JPEG files distributed via phishing
- Loki Bot - hid malware installers in JPEG and video files among others.

In this lab, you will experiment with a possible utilization of stegomalware, albeit using an intentionally harmless demonstration payload. This is solely to illustrate how hidden content may be embedded, transmitted, extracted and analyzed.

## Steganalysis

As with most fields in cybersecurity, attackers and defenders go back and forth, each creating and employing techniques to gain the upper hand over the other. In this constantly evolving cycle, steganalysis shows up as the corresponding defense against stegomalware.

Steganalysis is the formal name for the process of identifying, analyzing and potentially extracting hidden information embedded within digital media. Unlike ordinary file inspection however, steganalysis attempts to determine whether a file contains concealed data even when the file appears visually or structurally normal. It adopts a defensive approach by default.

In order to counteract steganography's stealthy techniques, most steganalysis techniques focus not on visible artifacts but on statistical irregularities and inconsistencies introduced during the embedding process. The most basic method of steganalysis would be comparison with the original carrier. If you already have in your possession a copy or an expectation of what some media file might look like, comparing it with the received copy (even with a simple diff command) will immediately show signs of something fishy if the file was tampered with. Steganalysis techniques go far beyond that.

They may broadly target:

- hidden appended data,
- suspicious metadata,
- anomalous file structures,
- or statistical perturbations in the media itself.

# Statistical steganalysis

More advanced steganographic techniques, such as LSB embedding, modify the carrier medium in ways deeper than just appending additional bytes. In these cases, the hidden information is more effectively detected through statistical means that attempt to identify subtle perturbations introduced into the media data.

Natural images exhibit statistical properties arising from sensor noise, lighting conditions, compression artifacts, texture and scene structure. Steganographic embedding of large enough payloads into those images will probably alter these properties in measurable ways, even when the image visually remains unchanged.

Common statistical indicators include:

- variance of residual noise
- entropy measurements
- histogram anomalies
- pixel correlation changes
- frequency-domain inconsistencies

Generally, no single measurement will suffice to prove the presence of hidden data, but the conjunction of them will strongly hint towards tampering with the file.

A common introductory approach to this field would be *residual noise analysis*. The central idea is to estimate the high-frequency components of an image by comparing the original (which may have been tampered with) against a denoised approximation of itself.

The process generally involves applying a denoising filter such as Gaussian blur or similar, computing the absolute difference between the original and denoised images and analyzing the resulting residual image statistically. This residual image will emphasize local pixel variations, noise, edges and potential steganographic perturbations.

Since techniques such as LSB embedding introduce small pseudo-random modifications into pixel values, large numbers of such modifications may alter the statistical distribution of the residual image.

Other heuristics may be applied, such as neighbor correlation or entropy. Natural images tend to exhibit strong correlation between neighboring pixels because adjacent regions of a scene are often visually similar. Steganographic embedding of a payload will often weaken this relationship. Entropy will measure the randomness or unpredictability of the image data. If it has higher entropy, that's a possible indication of tampering.

Unfortunately, statistical steganalysis is inherently probabilistic and often very difficult in practice. It depends on several factors such as payload size, image content and format, compression methods, embedding strategy, post-processing operations,... For these reasons, nowadays modern steganalysis relies heavily on machine learning, large statistical models and combination of multiple detection methods.

## **Further Research**

There are many techniques and tools that could not be fully explored in the scope of this lab. If you seek to expand your knowledge beyond it, we suggest you look into enterprise Content Disarm and Reconstruction (CDR) techniques.

CDR works by essentially stripping files of any data that is not strictly necessary for their functioning, thus removing most hiding spots of Steganographic embeddings. They can do this by stripping metadata, recompressing images, rewriting file structures, etc.