Antara N.J Frisch

Steganography

documentation

|  |  |  |
| --- | --- | --- |
| Version | Description | Date |
| 0.1 | Analyzing and initial draft | 14/05/2025 |
| 0.2 | Realising | 12/06/2025 |

Contents

[Introduction 2](#_Toc200669930)

[Context 2](#_Toc200669931)

[Analysis 3](#_Toc200669932)

[Realisation 3](#_Toc200669933)

[Summary 8](#_Toc200669934)

[Conclusion 9](#_Toc200669935)

# Introduction

Air-gapped systems are not connected to public or unsecured networks such as the internet, making them a favoured option for keeping sensitive data secure. However, despite this isolation, research has shown that these systems are not completely immune to attacks. Malicious code can exploit electromagnetic emissions, and while many studies demonstrate the feasibility of such attacks, fewer address how attackers can evade adaptive detection mechanisms, which often assume static or easily characterized signals.

# Context

This project explores the use of a Raspberry Pi 4 as a covert transmitter, leveraging its electromagnetic emissions to transmit data from an air-gapped system. In its current implementation, the setup transmits raw byte sequences. However, to increase the sophistication and stealth of the communication channel, further complexity of data is required. This will be explored with the concept of steganography.

# Analysis

Steganography is the practice of hiding data within seemingly harmless files like images, audio, or video to avoid detection. It allows secret content such as text files, commands, or code to be concealed without visibly altering the host file. Common techniques include embedding data in the least significant bits of image pixels, which appear identical to the original to human eyes. (TechTarget, 2023)

While steganography has legitimate uses, such as digital watermarking for copyright protection, it can also be exploited for malicious purposes. Cybercriminals use steganography to create "stegware" malware hidden inside media files. These files can bypass traditional security filters because they appear benign. Once opened on a victim's device, the hidden code can extract data, open backdoors, or connect to command and control servers. (TechTarget, 2023)

Attackers might embed malicious payloads in image or audio files attached to emails or uploaded to websites. Since the steganographic data is concealed and sometimes encrypted, detection becomes more difficult. This makes steganography an attractive method for covert communication, data exfiltration, and delivering malware in targeted attacks. (TechTarget, 2023)

Tools like OpenStego, Xiao Steganography, and Image Steganography help both legitimate users and malicious actors hide or extract data. When paired with encryption, steganography offers a powerful way to conceal sensitive or dangerous information. (TechTarget, 2023)

# Realisation

The project began with the challenge of selecting a suitable image one that was both small in file size and visually resembled a burger. The goal was to hide a secret message inside the image using steganography. To meet the technical constraints, the image needed to be minimal in size while still containing enough bytes to store a complete message.

Due to a transfer rate of just 2 bits per second, it was necessary to use a very small image to avoid long transmission times. After some rough calculations, it became clear that at least 72 bits were needed to store the secret message. Since RGB images typically use 3 bytes per pixel (one byte each for Red, Green, and Blue), this meant storing 1 bit in each color channel 3 bits per pixel. Therefore, a 4×6 pixel image (24 pixels total) would yield just enough capacity to store 72 bits, or 9 characters.

Finding such a tiny image that also looked like a burger turned out to be extremely difficult. Image repositories rarely contain images this small, and shrinking existing burger images resulted in visual noise or unusable color data. To work around this, a Python script was used to generate a 4×6 image from scratch. Each pixel's RGB values were chosen to vaguely represent a cheeseburger, keeping the image both meaningful and functional.

A computer screen shot of a program

Description automatically generated

Figure 1 image creation script

Initially, the plan was to use OpenStego, a tool designed for steganographic embedding using Least Significant Bit (LSB) manipulation. However, it quickly became apparent that OpenStego was built to hide entire files—like a .txt document—rather than individual characters or short strings. This created a problem: the file size of even a small .txt file exceeded the 72-bit limit of the image. As a result, the embedding had to be done manually using Python.

A diagram of a computer code

Description automatically generated with medium confidence

Figure 2 LSB diagram

A custom Python script was created to embed the message "404 sauce" into the image. Here's how it works:

The message is first converted to its binary representation, with a null terminator (00000000) added to indicate the end of the message.

These bits are then embedded, one at a time, into the least significant bit of each RGB color channel.

The result is a modified image that visually remains unchanged to the human eye but contains a complete hidden message.

A screen shot of a computer program

Description automatically generated

Figure 3 Embedding message

Since each of the 24 pixels holds 3 bits (one per color channel), the image offers exactly 72 bits of storage just enough to hold 9 characters. Once the message is embedded, the image is saved and ready for decoding.

A colorful stripes on a black background

Description automatically generated

Figure 4 steganography burger

A colorful striped rectangle on a black background

Description automatically generated

Figure 5 burger image

On the receiving side, a separate Python script acts as the decoder. It opens the image, reads each pixel’s RGB values, and extracts the least significant bit from each channel. These bits are then grouped into 8-bit segments (1 byte each) and converted back into characters. The process continues until the decoder encounters the null terminator, signalling the end of the hidden message.

A screen shot of a computer screen

Description automatically generated

Figure 6 Extraction code

For example, the message “404 sauce” appears in binary as:

00110100 # '4' (ASCII 52)

00110000 # '0' (ASCII 48)

00110100 # '4' (ASCII 52)

00100000 # ' ' (space, ASCII 32)

01110011 # 's' (ASCII 115)

01100001 # 'a' (ASCII 97)

01110101 # 'u' (ASCII 117)

01100011 # 'c' (ASCII 99)

01100101 # 'e' (ASCII 101)

This bit-by-bit reconstruction ensures accurate recovery of the original message.

# Summary

This project investigates the covert transmission of data from air-gapped systems by combining electromagnetic emission techniques with steganography. A Raspberry Pi 4 is used as the transmitter, and the project explores embedding hidden messages within small image files to increase stealth and sophistication.

The chosen steganographic method involves hiding binary data within the least significant bits (LSBs) of a custom 4×6 pixel image that resembles a burger. This size was carefully selected to match the system’s extremely limited transmission rate of 2 bits per second, providing just enough capacity (72 bits) to embed a 9-character message.

Attempts to use existing steganography tools like OpenStego proved ineffective for this scale, prompting the development of custom Python scripts. These scripts embedded the message “404 sauce” and later decoded it by reading the LSBs from the image’s pixels.

# Conclusion

This project shows that covert data transmission is achievable even under strict limitations like low bitrates and minimal image size, by using basic steganographic methods. Through manually encoding and decoding data using least significant bit (LSB) techniques, a working proof of concept was successfully created. This underscores the dual nature of such approaches: they offer practical tools for secure, low-profile communication but also present serious risks if misused, particularly in environments where conventional network monitoring cannot detect such activity.

# Bibliography

TechTarget. (2023, September). *techtarget.com*. Retrieved from What is steganography?: https://www.techtarget.com/searchsecurity/definition/steganography