




























































Esteganografía en retos CTF



























Contenido

Introducción a los Retos de Esteganografía en CTF	7
🧠 ¿Qué es?	8
🔍 ¿Cómo funciona?	8
✂️ Características técnicas	8
🔍 Uso en CTF	9
🔧 Cómo detectarlo y extraerlo	9
🐍 Ejemplo en Python	9
✅ Resumen rápido	10
📷 Esteganografía en Metadatos (EXIF)	11
🧠 ¿Qué es?	11
🔍 ¿Cómo funciona?	11
✂️ Características técnicas	11
🔍 Uso en CTF	12
🔧 Cómo detectarlo y extraerlo	12
🐍 Ejemplo en Python	12
✅ Resumen rápido	13
🎨 Esteganografía en el Dominio Espacial	14
🧠 ¿Qué es?	14
🔍 ¿Cómo funciona?	14
✂️ Características técnicas	14
🔍 Uso en CTF	15
🔧 Cómo detectarlo y extraerlo	15
🐍 Ejemplo en Python	15
✅ Resumen rápido	16
📊 Esteganografía en el Dominio de la Frecuencia (DCT / JPEG)	17
🧠 ¿Qué es?	17
🔍 ¿Cómo funciona?	17
✂️ Características técnicas	17
🔍 Uso en CTF	18
🔧 Cómo detectarlo y extraerlo	18

	Ejemplo en Python	18
	Resumen rápido	19
	Esteganografía mediante Técnicas de Codificación de Imagen	20
	¿Qué es?	20
	¿Cómo funciona?	20
	1. Canales de color (RGBA)	20
	2. Paleta de colores	20
	3. Estructura y formato del archivo	21
	Características técnicas	21
	Uso en CTF	21
	Cómo detectarlo y extraerlo	21
	Ejemplo en Python	22
	Resumen rápido	22
	Esteganografía mediante Acrósticos Visuales	23
	¿Qué es?	23
	¿Cómo funciona?	23
	Características técnicas	23
	Uso en CTF	24
	Cómo detectarlo y extraerlo	24
	Ejemplo práctico (detección de puntos y rayas)	24
	Resumen rápido	25
	Esteganografía en Códigos QR	26
	¿Qué es?	26
	¿Cómo funciona?	26
	Características técnicas	27
	Uso en CTF	27
	Cómo detectarlo y extraerlo	27
	Ejemplo básico de idea en Python	27
	Resumen rápido	28
	¿Qué es?	29
	¿Cómo funciona?	29
	Características técnicas	29
	Uso en CTF	30

	Cómo detectarlo y extraerlo	30
	Ejemplo práctico en Python	30
	Resumen rápido	31
	Esteganografía en el Espectrograma.....	32
	¿Qué es?	32
	¿Cómo funciona?	32
	Características técnicas	32
	Uso en CTF	33
	Cómo detectarlo y extraerlo	33
	Ejemplo de visualización rápida con Python	33
	Resumen rápido	34
	Esteganografía por Manipulación de la Fase de Audio	35
	¿Qué es?	35
	¿Cómo funciona?	35
	Características técnicas	35
	Uso en CTF	36
	Cómo detectar y extraerlo	36
	Ejemplo simplificado en Python.....	36
	Resumen rápido	37
	Esteganografía en Archivos de Vídeo	38
	¿Qué es?	38
	¿Cómo funciona?	38
	1. Ocultación en Frames de Vídeo.....	38
	2. Ocultación en Pista de Audio del Vídeo	38
	3. Manipulación de Metadatos o Estructura	39
	Características técnicas	39
	Uso en CTF	39
	Cómo detectar y extraerlo	39
	Ejemplo básico para extraer frames con Python (OpenCV)	40
	Resumen rápido	40
	Esteganografía en Archivos de Texto	41
	¿Qué es?	41
	Métodos principales	41

1. Espacios en blanco y tabulaciones (Whitespace Steganography)...	41
2. Caracteres no imprimibles / Unicode oculto	41
3. Acrósticos	42
4. Manipulación tipográfica.....	42
 Características técnicas	42
 Uso en CTF	42
 Cómo detectarlo	43
 Ejemplo en Python para detectar ZWSP	43
 Resumen rápido	43
 Esteganografía en Archivos Binarios.....	44
 ¿Qué es?	44
 ¿Cómo funciona?	44
 Características técnicas	45
 Uso en CTF	45
 Cómo detectar y extraerlo	45
 Ejemplo básico en Python para detectar secuencias largas de 00 ..	46
 Resumen rápido	46
 Esteganografía en Archivos Comprimidos (ZIP, RAR, etc.)	47
 ¿Qué es?	47
 ¿Cómo funciona?	47
1. Comentarios del archivo comprimido.....	47
2. Archivos ocultos o renombrados	47
3. Manipulación de la estructura interna	47
 Características técnicas	48
 Uso en CTF	48
 Cómo detectar y extraerlo	48
 Ejemplo básico en Python para leer comentario ZIP	49
 Resumen rápido	49
 Esteganografía en Archivos PDF	50
 ¿Qué es?	50
 ¿Cómo funciona?	50
1. Metadatos ocultos.....	50
2. Objetos invisibles o no referenciados	50
3. Uso de campos de formulario.....	50

4. Manipulación de flujos de datos	50
5. Imágenes o elementos con esteganografía interna	50
 Características técnicas	51
 Uso en CTF	51
 Cómo detectar y extraerlo	51
 Ejemplo básico con pdf-parser.py (Python)	52
 Resumen rápido	52
 Sonic Visualiser (Análisis de Espectro de Audio)	53
 ¿Qué es?	53
 ¿Cómo funciona?	53
 Características técnicas	54
 Uso en CTF	54
 Herramientas y comandos	54
 Ejemplo en CTF	54
 Resumen rápido	55
 Detección de Tonos DTMF (Dual-Tone Multi-Frequency)	56
 ¿Qué es?	56
 ¿Cómo funciona?	56
 Características técnicas	57
 Herramientas útiles	57
 Ejemplo en CTF	57
 Resumen rápido	58
 Phone Keypad Cipher	59
 ¿Qué es?	59
 ¿Cómo funciona?	59
 Tabla clásica de mapeo (Multi-Tap)	60
 Procedimiento en CTF	60
 Herramientas útiles	60
 Ejemplo en CTF	61
 Resumen rápido	61

Introducción a los Retos de Esteganografía en CTF

Los **retos de esteganografía** en los CTF (Capture The Flag) son desafíos que ponen a prueba la capacidad del participante para descubrir mensajes, archivos o información oculta dentro de otros archivos o medios digitales. A diferencia de la criptografía, que se centra en ocultar el contenido del mensaje a través de cifrados, la esteganografía busca ocultar la **existencia misma** del mensaje, camuflándolo dentro de elementos aparentemente inocuos.

En los CTF, los retos de esteganografía pueden involucrar imágenes, audios, vídeos, archivos PDF, documentos, archivos binarios o incluso códigos QR, entre otros. El objetivo es identificar la presencia de información oculta, descubrir el método utilizado para esconderla y extraerla correctamente.

Estos desafíos fomentan la combinación de habilidades técnicas en análisis forense digital, manipulación de archivos, programación, y un conocimiento sólido de formatos y técnicas avanzadas de ocultación. Además, enseñan a pensar de manera creativa y detallada para detectar patrones o anomalías que a simple vista pasan desapercibidas.

La esteganografía en CTF no solo es una práctica divertida y educativa, sino que también tiene aplicaciones reales en áreas como la seguridad informática, la privacidad, y la lucha contra la censura, donde comunicar mensajes secretos de forma imperceptible puede ser crucial.

Esteganografía en imágenes:

Esteganografía LSB (Least Significant Bit)

¿Qué es?

La técnica **LSB** consiste en ocultar información dentro de un archivo digital (normalmente imágenes, pero también audio o vídeo) modificando el **bit menos significativo** de cada byte de los datos.

El bit menos significativo (Least Significant Bit) es el que menos afecta al valor final del byte, por lo que su alteración es prácticamente imperceptible para el ojo o el oído humano.

¿Cómo funciona?

1. Elegir el **archivo portador** (por ejemplo, una imagen BMP o PNG sin compresión con pérdida).
2. **Convertir el mensaje a bits**.
3. **Reemplazar el último bit** de cada byte del archivo con los bits del mensaje.
4. El cambio es tan pequeño que no altera la apariencia o el sonido del archivo de forma perceptible.

Ejemplo con un byte:

- Pixel original (rojo): 10110110 (182 decimal)
 - Sustituyendo el LSB: 10110111 (183 decimal)
- La diferencia de color es imperceptible.

Características técnicas

- **Método:** modificación directa de bits de datos.
- **Requisitos:** formatos que soporten datos sin compresión destructiva (BMP, PNG, WAV, etc.).
- **Capacidad:** depende del tamaño del archivo portador y del número de bits modificados por píxel (1 o más bits por canal).
- **Seguridad:** alta para ocultación visual, baja frente a análisis estadístico.



Uso en CTF

- Muy común en retos de **forensics** o **steganography**.
- Suelen darte una imagen aparentemente normal.
- Herramientas como **zsteg**, **steghide**, **stegsolve**, **binwalk** o scripts de Python ayudan a extraer datos.
- A veces se combina con compresión, cifrado o formatos no estándar para despistar.



Cómo detectarlo y extraerlo

- Analizar los valores RGB de cada píxel y ver si hay patrones en los bits menos significativos.
- Usar herramientas automatizadas de esteganálisis.
- Comparar histogramas de colores con y sin LSB alterado.
- Revisar canales alfa o datos ocultos en metadatos.



Ejemplo en Python

```
from PIL import Image

def extract_lsb(image_path):
    img = Image.open(image_path)
    bits = ""
    for pixel in img.getdata():
        for channel in pixel[:3]: # R, G, B
            bits += str(channel & 1)
    # Convertir bits a texto (ASCII)
    chars = [chr(int(bits[i:i+8], 2)) for i in range(0,
len(bits), 8)]
    return "".join(chars)

mensaje = extract_lsb("imagen.png")
print("Mensaje oculto:", mensaje)
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía digital (bit-level)
Clave	No obligatoria, pero puede usarse
Capacidad	Alta (dependiendo del tamaño del portador)
Seguridad	Baja ante esteganálisis técnico
Uso en CTF	Forensics, stegano, retos de extracción

Esteganografía en Metadatos (EXIF)

¿Qué es?

Los **metadatos EXIF** (Exchangeable Image File Format) son información incrustada dentro de archivos de imagen (como JPEG o TIFF) que describen detalles técnicos y contextuales de la fotografía:

- Modelo de cámara
- Fecha y hora
- Configuración de exposición
- Coordenadas GPS
- Autor, copyright, comentarios, etc.

En esteganografía, estos campos pueden usarse para **ocultar mensajes, contraseñas o pistas** sin modificar visiblemente la imagen.

¿Cómo funciona?

1. **Seleccionar un archivo de imagen** (normalmente JPEG, aunque PNG y TIFF también pueden tener metadatos).
2. **Editar los campos EXIF** usando herramientas para insertar texto o datos binarios.
3. El archivo resultante se ve igual, pero conserva la información oculta.
4. El receptor extrae los metadatos para recuperar el mensaje.

Ejemplo:

- Campo Artist: “CTF{Flag_Oculto_Aqui}”
- Campo UserComment: binarios codificados en Base64.

Características técnicas

- **Método:** escritura de datos en campos EXIF estándar o personalizados.
- **Requisitos:** formato de imagen compatible con EXIF.
- **Capacidad:** limitada por el tamaño de los campos EXIF (normalmente unos pocos KB).
- **Seguridad:** muy baja; cualquiera con un visor de metadatos puede verlos.



Uso en CTF

- Muy frecuente en retos de **OSINT** o **steganografía básica**.
- Posible encontrar:
 - Coordenadas GPS que apuntan a un lugar clave.
 - Comentarios con hashes o contraseñas.
 - Flags directamente escritas en texto.
- Herramientas recomendadas:
 - `exiftool` (CLI)
 - `exiv2`
 - Visores online como *metapicz.com* o *exif.regex.info*



Cómo detectarlo y extraerlo

- Ejecutar:
- `exiftool imagen.jpg`

para listar todos los metadatos.
- Buscar campos sospechosos como `UserComment`, `Description`, `Copyright`, `Artist`.
- Si hay datos codificados, probar con Base64, Hex, o cifrados simples.



Ejemplo en Python

```
import piexif

# Leer metadatos EXIF
exif_dict = piexif.load("imagen.jpg")
user_comment = exif_dict["Exif"].get(piexif.ExifIFD.UserComment)

if user_comment:
    print("Comentario oculto:",
          user_comment.decode(errors="ignore"))
else:
    print("No hay comentario oculto")
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en metadatos
Clave	No necesaria
Capacidad	Baja (unos pocos KB)
Seguridad	Muy baja (visible con cualquier visor)
Uso en CTF	Búsqueda de pistas, flags o coordenadas



Esteganografía en el Dominio Espacial



¿Qué es?

La esteganografía en el **dominio espacial** consiste en ocultar información modificando directamente los valores de los píxeles de una imagen en su forma original, sin transformaciones matemáticas como en el dominio de la frecuencia (por ejemplo, DCT en JPEG).

Aquí, los datos se insertan cambiando **color, brillo o componentes de los píxeles** de forma que el ojo humano no perciba las alteraciones.



¿Cómo funciona?

Existen varias técnicas dentro del dominio espacial, siendo la más conocida **LSB (Least Significant Bit)**, pero también hay otras más visibles o sutiles:

1. Modificación directa de píxeles

Cambiar ligeramente los valores RGB o de brillo para codificar bits de información.

Ejemplo: si el bit es 1, sumar +1 al valor de un canal; si es 0, dejarlo igual.

2. Codificación por patrones

Asignar colores o variaciones específicas a ciertos píxeles que actúan como “marcas” en posiciones conocidas.

3. Cambio por bloques o regiones

Alterar el color o la luminosidad de zonas completas para representar información más robusta.



Características técnicas

- **Método:** manipulación directa de la matriz de píxeles.
- **Requisitos:** acceso a la imagen sin compresión con pérdida (PNG, BMP, TIFF) para evitar distorsiones.
- **Capacidad:** media-alta, según la resolución y el tipo de modificación.
- **Seguridad:** baja ante análisis de diferencias o histogramas.

Uso en CTF

- Muy común en retos **stegano** donde:
 - Una imagen aparentemente normal esconde un patrón.
 - El reto puede requerir comparar la imagen original con la modificada.
 - A veces la flag está escrita como texto muy tenue o en colores casi idénticos al fondo.
- Herramientas útiles:
 - **StegSolve** (visualización por capas y canales)
 - **GIMP/Photoshop** (aumentar contraste, brillo, saturación)
 - Scripts en Python con Pillow o OpenCV.

Cómo detectarlo y extraerlo

1. **Manipulación visual:**
 - Ajustar brillo, contraste, inversión de colores.
 - Separar canales RGB y analizar individualmente.
2. **Comparación de imágenes:**
 - Restar la imagen de un supuesto original para ver diferencias.
3. **Análisis de histogramas:**
 - Detectar patrones anómalos en la distribución de colores.

Ejemplo en Python

```
from PIL import Image
import numpy as np

# Cargar imagen y convertir a array
img = Image.open("imagen.png")
arr = np.array(img)

# Aumentar contraste artificialmente para resaltar posibles
datos ocultos
arr = np.clip((arr - 128) * 4 + 128, 0, 255)

Image.fromarray(arr.astype('uint8')).save("imagen_modificada.png")
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en píxeles (dominio espacial)
Clave	No siempre necesaria
Capacidad	Media-Alta
Seguridad	Baja ante análisis técnico
Uso en CTF	Patrones ocultos, mensajes en colores o brillo



Esteganografía en el Dominio de la Frecuencia (DCT / JPEG)



¿Qué es?

En lugar de modificar los píxeles directamente (dominio espacial), este método **oculta la información en las frecuencias** que componen la imagen.

Esto se hace usando transformadas matemáticas como la **Transformada de Coseno Discreto (DCT)**, que es la base de la compresión JPEG.

- La DCT descompone la imagen en componentes de **baja frecuencia** (detalle grueso) y **alta frecuencia** (detalle fino).
- Los datos se insertan modificando **coeficientes de frecuencia**, normalmente en las medias y altas, ya que los cambios ahí son menos perceptibles al ojo humano.



¿Cómo funciona?

1. **División en bloques**
El JPEG divide la imagen en bloques de 8×8 píxeles.
2. **Aplicación de la DCT**
Cada bloque se transforma en una matriz de coeficientes que representan distintas frecuencias.
3. **Inserción de datos**
Se modifican ciertos coeficientes (no los más bajos ni los más altos) para codificar bits.
4. **Compresión y guardado**
El archivo JPEG se guarda con los datos ocultos, visualmente casi idéntico al original.



Características técnicas

- **Método:** modificación de coeficientes de frecuencia en la DCT.
- **Requisitos:** formato JPEG u otros que usen compresión con transformadas.
- **Capacidad:** media, pero mayor resistencia a compresión y ruido que el dominio espacial.

- **Seguridad:** media-alta; no es evidente a simple vista, pero detectables con análisis estadístico.



Uso en CTF

- Más raro que el LSB porque requiere herramientas específicas.
- Pistas típicas:
 - Un JPEG que conserva calidad pero tiene tamaño sospechosamente grande.
 - Un reto que menciona “frecuencia”, “coseno” o “coeficientes”.
- Herramientas útiles:
 - **StegHide**
 - **OutGuess**
 - **F5 Steganography**
 - Scripts con `scipy.fftpack` para extraer la DCT.



Cómo detectarlo y extraerlo

1. Usar herramientas como:
2. `steghide extract -sf imagen.jpg`
3. Analizar los coeficientes DCT y compararlos con una versión limpia.
4. Revisar anomalías estadísticas en frecuencias medias.



Ejemplo en Python

```
import cv2
import numpy as np

# Leer imagen en escala de grises
img = cv2.imread("imagen.jpg", 0)

# Dividir en bloques de 8x8 y aplicar DCT
h, w = img.shape
for i in range(0, h, 8):
    for j in range(0, w, 8):
        bloque = img[i:i+8, j:j+8]
        dct_bloque = cv2.dct(np.float32(bloque))
        print(dct_bloque) # Aquí se podrían buscar alteraciones
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en el dominio de la frecuencia
Clave	Suele requerirla (dependiendo del método)
Capacidad	Media
Seguridad	Media-Alta
Uso en CTF	Reto avanzado con JPEG, uso de StegHide / OutGuess

Esteganografía mediante Técnicas de Codificación de Imagen

(Canales de color, paletas y formato de archivo)

¿Qué es?

Este tipo de esteganografía se basa en **usar las características técnicas de la codificación de imágenes** para ocultar datos. En lugar de esconder bits en el brillo o la frecuencia, se aprovechan **los canales de color, la estructura del archivo o la paleta de colores** para introducir información.

Es habitual en formatos como:

- **PNG** (canales RGBA)
- **GIF** (paletas de colores indexadas)
- **BMP** (canales sin compresión)
- **TIFF** (múltiples capas o etiquetas)

¿Cómo funciona?

1. Canales de color (RGBA)

- Cada píxel tiene valores para **Rojo (R)**, **Verde (G)**, **Azul (B)** y, en PNG, **Alfa (A)** para transparencia.
- Un atacante puede ocultar información en un canal poco perceptible (por ejemplo, modificar mínimamente el azul o el alfa).
- Ejemplo: si el canal alfa es completamente opaco (255), cambiar ligeramente su valor para codificar bits no será visible.

2. Paleta de colores

- En imágenes **indexadas** (GIF, PNG de 8 bits), cada píxel guarda un índice que apunta a un color en la paleta.
- Cambiando el orden de la paleta o sustituyendo colores con valores muy similares, se pueden codificar datos.
- Ventaja: visualmente imperceptible, pero almacena mucha información en el archivo.

3. Estructura y formato del archivo

- Algunos formatos permiten **campos extra** o **bloques no usados** donde se puede insertar texto.
- Ejemplo: en PNG, los chunks *tEXt* o *zTXt* pueden contener datos arbitrarios.
- En GIF, se pueden crear frames adicionales o datos de control con mensajes.



Características técnicas

- **Método:** manipulación de canales de color, paletas o campos propios del formato.
- **Requisitos:** conocimiento del estándar del formato.
- **Capacidad:** media-alta (depende del número de colores y tamaño de la imagen).
- **Seguridad:** media; requiere inspección binaria o de canales para detectarlo.



Uso en CTF

- Muy común en retos donde:
 - El PNG tiene transparencia sospechosa.
 - El GIF tiene muchos colores casi idénticos.
 - La paleta de colores muestra un patrón al visualizarse como datos crudos.
- Herramientas útiles:
 - **StegSolve** (permite ver cada canal por separado).
 - **zsteg** (para PNG y BMP).
 - **ImageMagick** (`convert imagen.png -separate canal.png`).
 - Visores hexadecimales para inspeccionar chunks.



Cómo detectarlo y extraerlo

1. Separar los canales de color:

```
convert imagen.png -channel R -separate r.png
convert imagen.png -channel G -separate g.png
convert imagen.png -channel B -separate b.png
convert imagen.png -alpha extract a.png
```

2. Analizar la paleta con:

```
identify -verbose imagen.gif
```

3. Buscar chunks ocultos en PNG:

```
zsteg imagen.png
```

Ejemplo en Python

```
from PIL import Image
```

```
img = Image.open("imagen.png")  
r, g, b, a = img.split() # Separar canales RGBA  
r.save("canal_rojo.png")  
a.save("canal_alpha.png")
```

Resumen rápido

Elemento	Valor
Tipo	Esteganografía en codificación de imagen
Clave	No siempre necesaria
Capacidad	Media-Alta
Seguridad	Media; requiere análisis de canales y estructura interna
Uso en CTF	Transparencia oculta, colores casi idénticos, chunks extra



Esteganografía mediante Acrósticos Visuales

(Patrones ocultos en el contenido visual, no en los datos digitales)



¿Qué es?

A diferencia de la esteganografía digital (que modifica bits de un archivo), los **acrósticos visuales** esconden la información en el **contenido visible de la imagen**. No requieren manipulación técnica del archivo, sino que el mensaje está “dibujado” o codificado dentro de la propia escena.

El nombre proviene del concepto de *acróstico* en literatura (mensaje oculto en letras iniciales), pero aquí se aplica a **patrones visuales**.



¿Cómo funciona?

1. Patrones intencionales

El autor crea una imagen donde ciertos elementos tienen un significado oculto.

2. Uso de códigos visuales

- **Morse:** puntos y rayas representados por objetos (flores grandes = raya, flores pequeñas = punto).
- **Binario:** posición de elementos (árbol a la izquierda = 0, a la derecha = 1).
- **Letras o números:** formados por montañas, rocas, nubes o edificios.

3. Interpretación humana o con IA

El reto está en observar y reconocer que **la disposición o el tamaño no son aleatorios**.



Características técnicas

- **Método:** patrones semánticos o geométricos dentro de la imagen.
- **Requisitos:** solo la imagen (no necesita metadatos).
- **Capacidad:** baja-media (limitada por la complejidad del patrón).
- **Seguridad:** alta contra análisis automatizado, baja contra observador humano entrenado.

Uso en CTF

En CTFs, este tipo suele aparecer en:

- Retos de *OSINT* o *forensics* donde hay que “mirar más allá”.
- Desafíos de arte digital, donde la pista está en el propio dibujo.
- Imágenes con patrones repetitivos que parecen decorativos pero representan datos.

Ejemplos comunes:

- Nubes formando letras.
- Ventanas iluminadas que forman números binarios.
- Farolas que indican puntos de un polígono (y luego se interpretan como coordenadas).

Cómo detectarlo y extraerlo

1. **Mirada analítica**
 - Buscar repetición en tamaños, posiciones o colores.
 - Preguntarse: “¿Esto podría ser un código?”
2. **Conversión a esquema**
 - Pasar la imagen a blanco y negro o resaltado de contornos para aislar formas.
 - Numerar elementos y mapearlos en un diagrama.
3. **Traducción**
 - Si parece morse: transcribir puntos y rayas.
 - Si es binario: agrupar en bytes y decodificar.
 - Si es geométrico: usar coordenadas para letras (ej. cifrado Polybius).

Ejemplo práctico (detección de puntos y rayas)

```
from PIL import Image
import cv2
import numpy as np

# Cargar imagen y convertir a escala de grises
img = cv2.imread('paisaje.png', cv2.IMREAD_GRAYSCALE)

# Umbral para detectar objetos
```

```
_, thresh = cv2.threshold(img, 200, 255, cv2.THRESH_BINARY)

# Encontrar contornos (flores, estrellas, etc.)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

for c in contours:
    area = cv2.contourArea(c)
    if area < 50:
        print("Punto detectado")
    else:
        print("Raya detectada")
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía visual no digital
Clave	No necesaria
Capacidad	Baja-Media
Seguridad	Alta contra análisis automático, baja contra ojo humano
Uso en CTF	Retos de observación, patrones artísticos

Esteganografía en Códigos QR

(Ocultación de información dentro de códigos QR)

¿Qué es?

La **esteganografía en códigos QR** consiste en ocultar datos secretos dentro de un código QR que a simple vista parece normal y funciona para almacenar información (como URLs, texto, etc.). Se aprovechan las propiedades del código QR para esconder mensajes adicionales sin afectar su escaneabilidad.

¿Cómo funciona?

1. Estructura básica de un QR

- Un código QR está formado por una matriz de módulos (cuadrados negros y blancos) que codifican datos.
- Tiene zonas fijas (marcadores de posición, sincronización) y zonas de datos.

2. Ocultación de datos

- Se manipulan módulos específicos de la matriz para codificar información oculta.
- Técnicas comunes:
 - **Bits escondidos en zonas de corrección de errores:** el QR puede tolerar errores; se modifican módulos que se pueden “corregir” al escanear, pero que en realidad codifican un mensaje secreto.
 - **Codificación en niveles de gris o colores:** usar diferentes tonos o colores en módulos para codificar bits extras.
 - **Superposición de datos:** combinar varios mensajes, donde uno es visible y otro es el mensaje oculto.

3. Lectura dual

- Un lector QR normal devuelve el mensaje “visible”.
- Un lector especializado o un análisis de la imagen extrae el mensaje oculto.



Características técnicas

- **Robustez:** el QR tiene corrección de errores (hasta ~30%), lo que permite modificar módulos sin perder funcionalidad.
- **Capacidad:** limitada, depende del nivel de corrección de errores y tamaño del QR.
- **Técnicas avanzadas:** uso de colores o microvariaciones en módulos para codificar más información.



Uso en CTF

En retos CTF, la esteganografía en QR puede aparecer como:

- Un QR aparentemente normal que apunta a una URL o texto inocuo.
- El mensaje oculto está en las diferencias sutiles de módulos, colores o patrones que requieren análisis gráfico.
- A veces se entregan varias imágenes QR para comparar.



Cómo detectarlo y extraerlo

1. **Comparar con un QR estándar**
 - Si tienes el mensaje visible, crear el QR original y comparar con el entregado para detectar diferencias.
2. **Análisis visual detallado**
 - Usar zoom para detectar módulos con tonos o formas diferentes.
 - Analizar posibles patrones.
3. **Herramientas especializadas**
 - Software de análisis de QR o scripts personalizados que extraen bits ocultos.



Ejemplo básico de idea en Python

```
import qrcode
from PIL import Image

# Crear QR estándar
qr =
qrcode.QRCode(error_correction=qrcode.constants.ERROR_CORRECT_H)
qr.add_data("Mensaje visible")
```

```

qr.make()
img = qr.make_image(fill_color="black",
back_color="white").convert('RGB')

# Modificar píxeles para ocultar datos (ejemplo muy simple)
pixels = img.load()
width, height = img.size

# Ejemplo: modificar algunos módulos para ocultar bits
for x in range(0, width, 10):
    for y in range(0, height, 10):
        r, g, b = pixels[x, y]
        # Cambiar ligeramente el color para ocultar datos (solo
demostrativo)
        pixels[x, y] = (r, g, (b+10) % 256)

img.show()

```

Resumen rápido

Elemento	Valor
Tipo	Esteganografía en código QR
Clave	A veces necesaria (para decodificar mensaje oculto)
Capacidad	Baja-media (depende de corrección de errores y tamaño QR)
Seguridad	Media; requiere análisis detallado para descubrir
Uso en CTF	QR aparentemente normal con mensaje adicional escondido



Esteganografía en audio:



Esteganografía LSB en Audio

(Least Significant Bit aplicado a señales de sonido)



¿Qué es?

La **esteganografía LSB en audio** consiste en ocultar información modificando el **bit menos significativo** de cada muestra de audio digital.

Como el bit LSB influye muy poco en el valor final, el cambio es casi imperceptible al oído humano.

Este método es análogo al LSB en imágenes, pero trabaja sobre el **flujo temporal de muestras sonoras** en lugar de píxeles.



¿Cómo funciona?

1. Digitalización del audio

- Un archivo de audio (ej. WAV) se compone de **muestras** que representan la amplitud de la onda sonora en intervalos de tiempo.
- Cada muestra se guarda con cierta profundidad de bits (8, 16, 24 bits, etc.).

2. Modificación del LSB

- El bit menos significativo de cada muestra se sustituye por un bit del mensaje secreto.
- Esto introduce un cambio tan pequeño que es imperceptible.

3. Reconstrucción del audio

- El archivo resultante suena casi igual, pero lleva embebido el mensaje.



Características técnicas

- **Método:** manipulación directa de los bits LSB en las muestras de audio.

- **Formato ideal:** WAV o AIFF (sin compresión), ya que formatos como MP3 pueden destruir el mensaje.
- **Capacidad:** depende de la tasa de muestreo y profundidad de bits.
Ejemplo: un WAV estéreo 44.1 kHz / 16 bits puede ocultar miles de caracteres en pocos segundos.
- **Seguridad:** vulnerable a recodificación y filtrado; robustez baja.

Uso en CTF

En retos CTF, el LSB en audio se presenta así:

- Un archivo WAV aparentemente normal.
- El reto pide “escuchar con otros oídos” → pista de que hay *stego*.
- Puede usarse para ocultar:
 - Texto plano.
 - Otros archivos binarios (ej. PNG oculto dentro del audio).

Cómo detectarlo y extraerlo

1. **Analizar formato**
 - Confirmar que es WAV/AIFF y no comprimido.
 - Revisar la tasa de bits.
2. **Inspeccionar bits**
 - Extraer el LSB de cada muestra.
 - Agruparlos en bytes y convertir a texto o binario.

Ejemplo práctico en Python

```
import wave

# Abrir archivo de audio
with wave.open("audio.wav", "rb") as audio:
    frames = audio.readframes(audio.getnframes())
    samples = list(frames)

# Extraer LSB de cada byte
bits = [str(sample & 1) for sample in samples]
```

```
# Convertir bits a texto
mensaje = "".join(
    chr(int("".join(bits[i:i+8]), 2))
    for i in range(0, len(bits), 8)
)

print("Mensaje oculto:", mensaje)
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en dominio temporal (audio)
Clave	No siempre necesaria
Capacidad	Alta (dependiendo de la duración y calidad del audio)
Seguridad	Baja ante compresión o edición
Uso en CTF	Archivos WAV sospechosos con datos ocultos en las muestras



Esteganografía en el Espectrograma

(Ocultación de datos en el dominio de la frecuencia)



¿Qué es?

La **ocultación en el espectrograma** consiste en modificar las frecuencias de un archivo de audio de forma que, al visualizarlo en un **espectrograma**, aparezca un patrón, texto o imagen. Normalmente se usan:

- Frecuencias inaudibles para humanos (por encima de ~20 kHz o por debajo de ~20 Hz).
- Señales integradas en el “ruido” del archivo.

Esta técnica es visual: el mensaje no se percibe escuchando, sino mirando el espectro.



¿Cómo funciona?

1. **Conversión a dominio de frecuencia**
 - El audio se analiza con la **Transformada de Fourier** para obtener las frecuencias presentes a lo largo del tiempo.
2. **Inserción de información**
 - Se modifican amplitudes específicas de ciertas frecuencias para dibujar formas o codificar bits.
 - Esto puede hacerse con software de edición espectral (ej. *Sonic Visualiser*, *Audacity* con *Spectrogram View*).
3. **Codificación visual o binaria**
 - Visual: texto, logotipos, QR, figuras.
 - Binaria: patrones que representan bits que luego se decodifican.



Características técnicas

- **Método:** manipulación de componentes de frecuencia.
- **Formatos ideales:** WAV, FLAC, AIFF (sin compresión destructiva).
- **Ventajas:** difícil de detectar si no se visualiza el espectro.

- **Limitaciones:** vulnerable a filtrado de frecuencias, recodificación con pérdida.

Uso en CTF

En retos CTF, el patrón suele ser:

- Te dan un archivo de audio “ruidoso” o con sonidos extraños.
- Al visualizar el espectrograma, aparece:
 - Texto claro (ej. *FLAG{...}*).
 - Un código QR que hay que escanear.
 - Coordenadas o símbolos.

Cómo detectarlo y extraerlo

1. **Abrir el audio en un analizador espectral:**
 - *Audacity* → Vista espectrograma (*Spectrogram View*).
 - *Sonic Visualiser*.
 - *Spek* (rápido para inspecciones iniciales).
2. **Ajustar la escala y el rango de frecuencias** para ver detalles ocultos:
 - Zoom en frecuencias altas o bajas.
 - Cambiar paleta de colores para resaltar patrones.
3. **Leer o extraer la información visual:**
 - Texto a simple vista.
 - Imagen capturada y procesada.
 - Decodificación de patrones binarios.

Ejemplo de visualización rápida con Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
from scipy.signal import spectrogram

# Cargar audio
rate, data = wavfile.read('audio.wav')

# Si es estéreo, convertir a mono
if data.ndim > 1:
```

```

data = data.mean(axis=1)

# Generar espectrograma
f, t, Sxx = spectrogram(data, rate)
plt.pcolormesh(t, f, 10 * np.log10(Sxx), shading='gouraud')
plt.ylabel('Frecuencia [Hz]')
plt.xlabel('Tiempo [s]')
plt.ylim(0, 24000) # Rango audible + un poco más
plt.show()

```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en dominio de frecuencia (audio)
Clave	No siempre necesaria
Capacidad	Media-alta (visual o binaria)
Seguridad	Media (resiste cambios leves, pero no filtrado fuerte)
Uso en CTF	Muy común; espectrogramas con texto, QR o símbolos ocultos

Esteganografía por Manipulación de la Fase de Audio

(Ocultación de datos modificando la fase de la señal sonora)

¿Qué es?

Este método consiste en **ocultar información modificando la fase de las ondas de sonido** en una señal de audio digital.

Mientras que la amplitud determina el volumen y la frecuencia el tono, la **fase** es el desplazamiento temporal de la onda. Cambiar la fase puede ser imperceptible para el oído humano si se hace cuidadosamente, pero permite codificar datos.

¿Cómo funciona?

1. **Análisis de la señal**
 - Se toma la señal de audio y se aplica una transformada, generalmente la **Transformada Discreta de Fourier (DFT)** o la **Transformada Rápida de Fourier (FFT)**, para separar las componentes en frecuencia, amplitud y fase.
2. **Modificación de la fase**
 - Se alteran los ángulos de fase de ciertas frecuencias específicas para codificar bits del mensaje.
 - La amplitud se mantiene para que el audio suene igual.
3. **Reconstrucción**
 - Se realiza la transformada inversa para obtener la señal temporal con la fase modificada.
 - El resultado es un archivo de audio que suena igual pero lleva el mensaje escondido en la fase.

Características técnicas

- **Método:** manipulación en dominio de frecuencia, enfocándose en fase, no en amplitud.
- **Ventajas:** difícil de detectar con análisis convencional (no altera amplitud ni espectro).
- **Desventajas:** más complejo de implementar y extraer que LSB o espectrograma.

- **Robustez:** alta contra compresión y ruido, pero sensible a cambios de tiempo o recortes.

Uso en CTF

Menos frecuente que LSB o espectrogramas, pero aparece en retos avanzados de esteganografía en audio donde:

- El audio parece normal y no se detectan cambios en amplitud o espectrograma.
- Se requiere analizar la **fase de frecuencias** para extraer datos.
- Se usan herramientas especializadas o scripts para analizar la fase.

Cómo detectar y extraerlo

1. **Transformada de Fourier**
 - Calcular FFT para obtener magnitud y fase.
2. **Comparar fase con referencia**
 - Si se tiene el audio original sin estego, se puede comparar fases.
 - Buscar cambios significativos codificados en patrones.
3. **Decodificación de bits**
 - Mapear ciertos cambios de fase a bits 0 o 1.
 - Reconstruir mensaje a partir de secuencia de bits.

Ejemplo simplificado en Python

```
import numpy as np
from scipy.io import wavfile
from scipy.fft import fft, ifft

# Leer audio
rate, data = wavfile.read('audio.wav')
if data.ndim > 1:
    data = data.mean(axis=1)

# FFT
freq_data = fft(data)
```

```
# Extraer fase
phase = np.angle(freq_data)

# Aquí se analizaría/modificaría la fase para ocultar o extraer
datos
# Ejemplo: detectar cambios en fase para decodificar bits

# (Este paso requiere conocer esquema de codificación)

# Reconstruir audio (sin cambios en este ejemplo)
reconstructed = ifft(freq_data).real.astype(np.int16)

# Guardar o reproducir reconstructed
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en dominio de frecuencia (fase de audio)
Clave	Generalmente sí, o referencia para comparación
Capacidad	Media
Seguridad	Alta, difícil de detectar con análisis común
Uso en CTF	Retos avanzados de esteganografía en audio



Esteganografía en otros archivos:



Esteganografía en Archivos de Vídeo

(Ocultación de datos en frames o pistas de audio de un vídeo)



¿Qué es?

La **esteganografía en vídeos** consiste en ocultar información dentro de un archivo de vídeo, usando ya sea:

- Modificación de **frames (imágenes)** individuales, alterando píxeles o colores.
- Ocultación en la **pista de audio** del vídeo.
- Manipulación de metadatos o estructuras internas del contenedor multimedia.



¿Cómo funciona?

1. Ocultación en Frames de Vídeo

- **LSB en píxeles de frames:**
Modificar el bit menos significativo de algunos píxeles de frames seleccionados para insertar bits del mensaje.
- **Manipulación en el dominio de la frecuencia:**
Aplicar transformadas (DCT, DWT) a los frames y ocultar datos en coeficientes menos perceptibles, como en JPEG o MPEG.
- **Cambios imperceptibles en colores:**
Ajustar ligeramente valores RGB o canales de color sin que se note visualmente.

2. Ocultación en Pista de Audio del Vídeo

- Aplicar técnicas de esteganografía en audio (LSB, espectrograma, fase) en la pista sonora asociada al vídeo.

3. Manipulación de Metadatos o Estructura

- Insertar datos en campos no visibles o poco usados del contenedor de vídeo (como MKV, MP4), sin alterar la reproducción.



Características técnicas

- **Capacidad:** alta debido a gran cantidad de frames y datos de audio.
- **Imperceptibilidad:** bien diseñada, no se nota diferencia visual ni auditiva.
- **Robustez:** varía; puede perderse con recompresión o edición.
- **Formato común:** MP4, AVI, MKV, MOV.



Uso en CTF

- Vídeos con apariencia normal que contienen mensajes ocultos en imágenes o audio.
- A veces se entregan múltiples vídeos para comparar o pistas para analizar frames específicos.
- Se puede necesitar extracción frame a frame o analizar la pista de audio separadamente.



Cómo detectar y extraerlo

1. **Extraer frames**
 - Usar *ffmpeg* o software para extraer imágenes de vídeo:
 - `ffmpeg -i video.mp4 frame_%04d.png`
 - Analizar frames individualmente con técnicas como LSB, comparando con originales si existen.
2. **Extraer audio**
 - Extraer pista de audio para analizar con técnicas de esteganografía en audio:
 - `ffmpeg -i video.mp4 -q:a 0 -map a audio.wav`
3. **Analizar metadatos**
 - Usar herramientas como *exiftool* para revisar metadatos y datos ocultos.
4. **Uso de software especializado**

- Herramientas para esteganálisis multimedia, como *StegExpose*, *StegSolve* (más para imágenes), o scripts personalizados.

Ejemplo básico para extraer frames con Python (OpenCV)

```
import cv2

video_path = 'video.mp4'
cap = cv2.VideoCapture(video_path)
frame_num = 0

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imwrite(f'frame_{frame_num:04d}.png', frame)
    frame_num += 1

cap.release()
print("Frames extraídos.")
```

Resumen rápido

Elemento	Valor
Tipo	Esteganografía en vídeo (frames, audio, metadatos)
Clave	Depende del método y clave del mensaje
Capacidad	Alta
Seguridad	Media-alta (depende de técnica y recompresión)
Uso en CTF	Muy común; vídeos aparentemente normales con datos ocultos



Esteganografía en Archivos de Texto

(Ocultación en espacios en blanco, caracteres no imprimibles o acrósticos)



¿Qué es?

La **esteganografía en texto** consiste en ocultar información dentro de un archivo de texto plano, sin alterar significativamente su contenido visible.

La manipulación puede ser en el **contenido visual** o en los **caracteres invisibles** que el lector normal no percibe.



Métodos principales

1. Espacios en blanco y tabulaciones (Whitespace Steganography)

- Se añade información **en los espacios al final de cada línea** (EOL) o en secuencias de espacios/tabs dentro del texto.
- La codificación se puede hacer, por ejemplo:
 - **Espacio** = 0
 - **Tabulación** = 1
- Herramienta famosa: **SNOW** (Steganographic Nature Of Whitespace).



Ejemplo:

Hola mundo[_]
Esto es un texto[Tab]

(Los espacios y tabs aquí codifican bits de un mensaje oculto).

2. Caracteres no imprimibles / Unicode oculto

- Se usan caracteres como **zero-width space (ZWSP)** U+200B, **zero-width non-joiner (ZWNJ)**, o **zero-width joiner (ZWJ)**.
- No se ven, pero están en el archivo y pueden representar bits.
- Ejemplo: un carácter ZWSP = 0, un ZWNJ = 1.

3. Acrósticos

- Usar la **primera letra** (o la última) de cada línea o párrafo para formar un mensaje oculto.
- También se pueden usar **palabras iniciales** para transmitir un código.

💡 Ejemplo:

```
**C**uando  
**T**odos  
**F**allan
```

(Iniciales = "CTF")

4. Manipulación tipográfica

- Uso intencionado de variantes Unicode visualmente idénticas (homoglyphs) para representar bits.
- Ejemplo: letra a normal vs. a cirílica (parecen iguales, pero son diferentes códigos).

✂ Características técnicas

Característica	Detalle
Capacidad	Baja-media, depende del método
Robustez	Baja frente a edición de texto o cambios de formato
Imperceptibilidad	Alta si se usan espacios, Unicode invisible o acrósticos
Riesgo	Fácil de romper si se sospecha y se usa un editor hex/Unicode

🔍 Uso en CTF

- Archivos .txt aparentemente normales que contienen **espacios invisibles** o patrones raros.
- Mensajes ocultos en las **primeras letras** de líneas o párrafos.
- Textos con caracteres **Unicode extraños** detectables con un editor hexadecimal.

Cómo detectarlo

1. Mostrar caracteres invisibles

- Usar `cat -A archivo.txt` en Linux para mostrar tabs y espacios.
- En Python:
- `with open("archivo.txt", "rb") as f:`
- `print(f.read())`

2. Buscar Unicode oculto

- Usar scripts que identifiquen caracteres con código >127 o invisibles.

3. Analizar acrósticos

- Extraer sistemáticamente primeras/últimas letras y comprobar si forman un mensaje.

Ejemplo en Python para detectar ZWSP

```
with open("archivo.txt", "r", encoding="utf-8") as f:  
    data = f.read()
```

```
zwsp = "\u200B"  
if zwsp in data:  
    print(f"Se encontraron {data.count(zwsp)} caracteres Zero-  
Width Space.")
```

Resumen rápido

Elemento	Valor
Tipo	Esteganografía en texto
Clave	Mapeo de espacios/tabs o letras de acróstico
Capacidad	Baja a media
Seguridad	Baja (fácil de alterar si se formatea el texto)
Uso en CTF	Muy común en retos de forense y esteganografía con .txt sospechosos



Esteganografía en Archivos Binarios

(Ocultación en zonas no utilizadas o “relleno” dentro del archivo)



¿Qué es?

La esteganografía en archivos binarios consiste en ocultar datos secretos dentro de un archivo ejecutable, imagen, o cualquier archivo binario, aprovechando espacios **no utilizados, relleno, o áreas reservadas** que no afectan el funcionamiento o la integridad del archivo.



¿Cómo funciona?

1. Zonas no utilizadas o relleno

- Muchos formatos binarios incluyen espacios de relleno (padding) para alinear datos, o áreas reservadas que no son críticas.
- Se pueden insertar datos en estas zonas sin alterar la funcionalidad del archivo.
- Ejemplo: en archivos ejecutables, existen segmentos o secciones vacías.

2. Espacios entre segmentos

- Entre bloques de datos o instrucciones pueden quedar huecos de bytes nulos (00) o valores constantes, ideales para ocultar bits.

3. Metadatos internos

- Algunos archivos binarios tienen campos reservados o metadatos extensibles donde se puede incrustar información.

4. Modificación controlada de bytes

- Alterar bytes que no afectan la ejecución, como códigos no usados o secciones padding, para codificar datos binarios.

Características técnicas

Característica	Detalle
Capacidad	Media a alta (depende del archivo y zonas disponibles)
Robustez	Alta, si se usa correctamente; no afecta funcionamiento
Imperceptibilidad	Alta, porque los datos no se ven ni alteran el comportamiento
Riesgo	Edición accidental puede borrar datos ocultos

Uso en CTF

- Archivos .exe, .dll, .bin o incluso imágenes con zonas de relleno.
- Pistas para examinar secciones vacías o bytes repetitivos.
- A veces se entregan con instrucciones para buscar “datos en zonas de relleno”.

Cómo detectar y extraerlo

1. Análisis hexagonal

- Abrir archivo con *hex editor* (HxD, bless, etc.).
- Buscar bloques grandes de bytes repetidos (00, FF) o patrones extraños.

2. Comparar con archivo original

- Si se tiene un archivo base, comparar byte a byte para encontrar diferencias o datos extra.

3. Uso de herramientas forenses

- *binwalk* puede analizar estructuras internas y detectar datos embebidos.

4. Extraer segmentos específicos

- Extraer rangos de bytes sospechosos y analizar contenido (texto, binario, etc.).

Ejemplo básico en Python para detectar secuencias largas de 00

```
with open("archivo.bin", "rb") as f:
    data = f.read()

long_zeros = []
count = 0
start = None

for i, byte in enumerate(data):
    if byte == 0x00:
        if count == 0:
            start = i
            count += 1
        else:
            if count > 10: # Secuencia larga arbitraria
                long_zeros.append((start, count))
            count = 0

if long_zeros:
    print("Secuencias largas de ceros encontradas en:")
    for s in long_zeros:
        print(f"Offset: {s[0]}, Longitud: {s[1]}")
else:
    print("No se encontraron secuencias largas de ceros.")
```

Resumen rápido

Elemento	Valor
Tipo	Esteganografía en zonas no usadas o relleno en binarios
Clave	Conocer estructura interna del archivo para no corromper
Capacidad	Media-alta
Seguridad	Alta, difícil de detectar sin análisis forense
Uso en CTF	Muy usada en retos con archivos binarios o ejecutables



Esteganografía en Archivos Comprimidos (ZIP, RAR, etc.)

(Ocultación de datos en comentarios y estructura interna del archivo)



¿Qué es?

La esteganografía en archivos comprimidos consiste en esconder información oculta dentro de un archivo ZIP, RAR, 7z u otro formato, sin alterar la posibilidad de descomprimirlo normalmente. Se aprovechan características específicas del formato:

- Comentarios en el archivo comprimido
- Estructura interna con campos reservados o padding
- Archivos ocultos o renombrados dentro del contenedor



¿Cómo funciona?

1. Comentarios del archivo comprimido

- Muchos formatos (ZIP, RAR) permiten incluir un comentario de texto al archivo comprimido.
- En este comentario se puede esconder texto plano o incluso datos codificados (Base64, hex).
- El comentario no afecta la descompresión ni la integridad del archivo.

2. Archivos ocultos o renombrados

- Insertar archivos con nombres inusuales o extensiones falsas dentro del archivo comprimido.
- Se pueden renombrar archivos con extensiones dobles para despistar.

3. Manipulación de la estructura interna

- Algunos campos o metadatos en el encabezado del archivo comprimido permiten almacenamiento extra.
- Campos de relleno o zonas no usadas pueden ser modificadas para esconder datos binarios.

Características técnicas

Característica	Detalle
Capacidad	Baja a media (dependiendo del tamaño de comentarios o archivos)
Robustez	Alta, no afecta la descompresión
Imperceptibilidad	Alta, el archivo funciona normalmente
Riesgo	Fácil de detectar si se analiza con herramientas adecuadas

Uso en CTF

- Archivos ZIP o RAR que se entregan como pistas con comentarios o archivos sospechosos.
- Mensajes ocultos en comentarios o archivos con nombres raros dentro del archivo comprimido.
- A veces se usa para pasar mensajes secretos o pequeñas claves dentro de un contenedor.

Cómo detectar y extraerlo

1. Leer comentarios del archivo

- En ZIP:
- `unzip -z archivo.zip`
- En RAR:
- `unrar l archivo.rar`

Luego buscar en la salida el comentario o metadatos.

2. Listar archivos ocultos o extraños

- Usar comandos normales para listar contenido:
- `unzip -l archivo.zip`
- `rar l archivo.rar`
- Buscar nombres o tamaños inusuales.

3. Herramientas especializadas

- *binwalk* para analizar estructura binaria.
- *7-Zip* para examinar contenido oculto.

- Herramientas específicas para metadatos (como *zipinfo*).

Ejemplo básico en Python para leer comentario ZIP

```
import zipfile

with zipfile.ZipFile('archivo.zip', 'r') as zipf:
    comment = zipf.comment
    if comment:
        print("Comentario oculto en ZIP:")
        print(comment.decode('utf-8'))
    else:
        print("No hay comentario en el ZIP.")
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en comentarios y estructura interna
Clave	Acceso al archivo comprimido y lectura de metadatos
Capacidad	Baja-media
Seguridad	Media, detectable con herramientas
Uso en CTF	Común en retos con archivos ZIP o RAR



Esteganografía en Archivos PDF

(Ocultación de datos en la estructura compleja y metadatos del PDF)



¿Qué es?

Los archivos PDF son documentos con una estructura compleja que incluye texto, imágenes, objetos, metadatos y scripts. Esta complejidad permite ocultar información en varias partes del archivo sin afectar la visualización del documento.



¿Cómo funciona?

1. Metadatos ocultos

- Los PDFs contienen metadatos (autor, título, palabras clave).
- Se pueden añadir metadatos personalizados con mensajes ocultos o datos codificados.

2. Objetos invisibles o no referenciados

- PDFs están formados por múltiples objetos (texto, imágenes, fuentes).
- Es posible incluir objetos no referenciados ni mostrados que contengan información oculta.

3. Uso de campos de formulario

- Campos de formulario (checkbox, texto) con contenido invisible o muy pequeño.

4. Manipulación de flujos de datos

- Modificar flujos de texto o imágenes con bits específicos para ocultar información.

5. Imágenes o elementos con esteganografía interna

- Insertar imágenes que contengan datos ocultos (LSB en imágenes embebidas).



Características técnicas

Característica	Detalle
Capacidad	Media a alta (depende del método y tamaño del PDF)
Robustez	Media (puede dañarse si se re-optimiza el PDF)
Imperceptibilidad	Alta, dado que el documento se ve normal
Riesgo	Puede detectarse con análisis profundo de estructura interna



Uso en CTF

- Archivos PDF con mensajes ocultos en metadatos o en objetos no visibles.
- PDFs con imágenes o formularios que esconden pistas.
- Uso frecuente para ocultar datos cuando se desea enviar documentos “normales” con información extra.



Cómo detectar y extraerlo

1. Inspeccionar metadatos

- Usar `pdftinfo` (Linux) o herramientas como *ExifTool*.

2. Extraer objetos PDF

- Usar `pdf-parser.py` (herramienta de Didier Stevens) para listar y examinar objetos PDF.
- Buscar objetos no referenciados o sospechosos.

3. Revisar contenido de campos de formulario

- Abrir el PDF en lectores avanzados y examinar campos ocultos.

4. Extraer imágenes y analizarlas

- Extraer imágenes con `pdfimages` y analizarlas con técnicas LSB u otras.

Ejemplo básico con pdf-parser.py (Python)

```
pdf-parser.py -s "stream" archivo.pdf
```

Esto lista todos los objetos stream, donde pueden estar datos ocultos.

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en metadatos, objetos invisibles, formularios, imágenes embebidas
Clave	Conocimiento de la estructura PDF y uso de herramientas específicas
Capacidad	Media-alta
Seguridad	Media (puede perderse en re-guardados o ediciones)
Uso en CTF	Muy común para esconder mensajes y archivos dentro de PDFs

Sonic Visualiser (Análisis de Espectro de Audio)

(Herramienta para descubrir mensajes ocultos en espectrogramas de audio)

¿Qué es?

Sonic Visualiser es un software gratuito y de código abierto utilizado para analizar archivos de audio en profundidad, permitiendo visualizar su contenido en el dominio de la frecuencia y del tiempo.

En retos CTF, es una herramienta clave para encontrar mensajes ocultos en **espectrogramas**, ya que muchos retos de esteganografía en audio esconden texto, dibujos o códigos QR en la representación gráfica de las frecuencias.

¿Cómo funciona?

1. **Carga del audio:** Abrimos el archivo de audio (WAV, MP3, FLAC, etc.) en Sonic Visualiser.
2. **Visualización del espectrograma:** Seleccionamos la opción de espectrograma, que muestra las frecuencias en el eje vertical y el tiempo en el eje horizontal, con la intensidad representada por colores.
3. **Búsqueda de patrones ocultos:** Inspeccionamos la imagen resultante en busca de:
 - Texto oculto (mensajes escritos)
 - Dibujos (símbolos, logotipos)
 - Códigos QR
4. **Interpretación y extracción:** Una vez identificado el patrón, podemos:
 - Leerlo directamente si es texto.
 - Extraerlo como imagen para un procesamiento posterior.



Características técnicas

Característica	Detalle
Tipo de ocultación	Esteganografía en el dominio de la frecuencia
Requerimientos	Archivo de audio con datos ocultos en su espectro
Visibilidad	Alta si se usan frecuencias audibles, baja si está en el rango inaudible
Capacidad	Limitada por la duración y el rango de frecuencias



Uso en CTF

- **Mensajes en espectrogramas:** Textos codificados en bandas de frecuencia.
- **Dibujos o logotipos:** Figuras creadas mediante síntesis de frecuencias.
- **Códigos QR ocultos:** Codificados como imagen espectral que luego puede escanearse.
- **Pistas direccionales:** Frecuencias que guían al siguiente paso del reto.



Herramientas y comandos

- **Sonic Visualiser:** Disponible para Windows, macOS y Linux.
 - [Página oficial](#)
 - Una vez abierto el audio:
 1. Pane → Add Spectrogram
 2. Ajustar rango de frecuencias y colores para mayor contraste.
 3. Usar zoom y desplazamiento para inspeccionar detalles.



Ejemplo en CTF

Un reto puede darte un archivo mensaje.wav. A simple oído, suena como ruido blanco.

Al abrirlo en Sonic Visualiser y activar el espectrograma, aparece un texto que dice:

```
flag{hidden_in_the_noise}
```

✓ Resumen rápido

Elemento	Valor
Tipo	Esteganografía en audio, dominio de la frecuencia
Clave	Analizar el espectro con la herramienta adecuada
Capacidad	Baja-media (ideal para mensajes cortos o imágenes pequeñas)
Detección	Fácil con herramientas de espectrograma
Uso en CTF	Muy común para mensajes escondidos en ruido

Detección de Tonos DTMF (Dual-Tone Multi-Frequency)

(Identificación de dígitos y comandos transmitidos por tonos telefónicos)

¿Qué es?

DTMF es un sistema de señalización usado en telefonía para enviar información a través de la pulsación de teclas. Cada número o símbolo que pulsas en un teléfono genera **dos tonos simultáneos** (uno de una frecuencia baja y otro de una frecuencia alta).

En retos CTF, los DTMF pueden usarse para ocultar mensajes o códigos en un archivo de audio que suena como pitidos telefónicos. Detectarlos y decodificarlos permite revelar el texto o número oculto.

¿Cómo funciona?

1. Frecuencias usadas

Los DTMF están organizados en una matriz de 4 filas (frecuencias bajas) y 4 columnas (frecuencias altas):

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

2. Proceso de detección en CTF

- Abrir el archivo de audio sospechoso.
- Escuchar si tiene pitidos cortos y distintos.
- Usar herramientas de análisis de espectro (Audacity, Sonic Visualiser, multímetros FFT) para identificar las frecuencias de cada tono.
- Mapear cada par de frecuencias a su número o letra usando la tabla DTMF.

3. Extracción del mensaje

- Escribir la secuencia de dígitos/letras detectados.
- Si forman un texto, puede ser la flag o una pista.
- Si parecen números de teléfono o códigos, podrían llevar a la siguiente fase.

Características técnicas

Característica	Detalle
Tipo de ocultación	Mensaje en audio mediante pares de frecuencias
Rango de frecuencias	697 Hz a 1633 Hz
Capacidad	Muy baja (números/códigos cortos)
Visibilidad	Fácil de oír, requiere identificación de pares de tonos
Detección	Manual o con herramientas automáticas

Herramientas útiles

- **Audacity:**
 - Abrir audio → Análisis → Trazar espectro para cada tono.
- **Multimon-ng (Linux):**
 - `multimon-ng -t wav -a DTMF archivo.wav`
- **DTMF.io (web):**
 - Subir audio y obtener la decodificación automática.
- **Python + scipy:**
 - Usar FFT para detectar frecuencias dominantes y mapearlas.

Ejemplo en CTF

Archivo: `phone_call.wav`

Al escucharlo, suena como una secuencia de pitidos cortos.

Tras analizarlo con **multimon-ng**, obtenemos:

5 1 2 8 3 4

La flag del reto es:

`flag{512834}`

✓ Resumen rápido

Elemento	Valor
Tipo	Audio → Frecuencias → Números
Clave	Identificar dos frecuencias simultáneas por tono
Capacidad	Baja, ideal para datos cortos
Detección	Fácil con herramientas de espectro o decodificadores DTMF
Uso en CTF	Muy común en retos de audio relacionados con telefonía



Phone Keypad Cipher

(Codificación de texto usando las letras asociadas a las teclas de un teléfono)



¿Qué es?

Antes de los smartphones, los teléfonos móviles y fijos con teclado numérico incluían **varias letras en cada número** (para escribir SMS usando pulsaciones repetidas). Este sistema puede usarse como un método de codificación donde cada número representa una letra según la posición en la tecla.

En retos CTF, el mensaje puede aparecer como una secuencia de números (ejemplo: 7777 33 666 555) que corresponde a texto si se mapea correctamente.



¿Cómo funciona?

Existen dos variantes principales:

1. Multi-Tap SMS (pulsaciones repetidas)

Cada número representa varias letras dependiendo de cuántas veces se pulse:

- 2 → A (1 vez), B (2 veces), C (3 veces)
- 7 → P (1 vez), Q (2 veces), R (3 veces), S (4 veces)

Ejemplo:

7777 33 666 555
S E O L

2. T9 Predictive Text

Cada palabra se escribe pulsando solo una vez por letra y un diccionario interno adivina la palabra. En CTF, rara vez usan T9 porque requiere diccionario, pero puede aparecer como clave oculta.

Tabla clásica de mapeo (Multi-Tap)

Número	Letras
1	(a veces vacío o signos)
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P Q R S
8	T U V
9	W X Y Z
0	Espacio

Procedimiento en CTF

1. Identificar si la secuencia de números parece un patrón de pulsaciones telefónicas.
 - Suele tener repeticiones (ej: 222 666 555).
 - Puede estar separada por espacios o comas.
2. Usar la tabla para convertir:
 - Contar cuántas veces se repite cada número antes de que cambie.
 - Mapear a la letra correspondiente.
3. Unir el texto y buscar la flag.

Herramientas útiles

- DCode.fr - Phone Keypad Cipher
- CyberChef - From Phone Keypad
- Scripts en Python para contar repeticiones.

Ejemplo en CTF

Entrada:

7777 33 666 555 0 33 666 8

Decodificación:

S E O L _ E O T

Flag:

flag{SEOL_EOT}

Resumen rápido

Elemento	Valor
Tipo	Sustitución basada en pulsaciones de teclado
Clave	Tabla de letras por tecla (multi-tap)
Capacidad	Media (texto plano, cualquier palabra)
Detección	Fácil si hay secuencias repetidas
Uso en CTF	Muy común en retos con números repetidos



Herramientas comunes para resolver retos de esteganografía:

- **StegHide:** Una herramienta popular para ocultar y extraer datos de archivos de imagen y audio. A menudo requiere una contraseña.
- **StegSolve:** Una herramienta gráfica de Java muy útil para el estegoanálisis de imágenes, que permite visualizar los diferentes canales de color, realizar filtros y análisis de bits.
- **Binwalk:** Ideal para buscar archivos incrustados dentro de otros archivos (por ejemplo, un archivo ZIP dentro de una imagen).
- **ExifTool:** Para analizar y manipular metadatos de una gran variedad de formatos.
- **Strings:** Permite extraer cadenas de texto legibles de un archivo binario.
- **Editores hexadecimales (como GHex o HxD):** Fundamentales para examinar la estructura de un archivo a nivel de bytes y buscar patrones o datos ocultos.
- **Editores de audio (como Audacity):** Útiles para visualizar espectrogramas y analizar archivos de audio en busca de datos ocultos.

En general, los retos de esteganografía en CTF requieren un enfoque metódico y creativo, a menudo combinando el uso de varias herramientas para descubrir la información oculta.

