Steganografia

Steganografia

La **steganografia** è la pratica di nascondere un messaggio all'interno di un altro messaggio o di un oggetto fisico, in modo che il messaggio nascosto non sia percepito come tale. A differenza della crittografia, che maschera il contenuto del messaggio, la steganografia cerca di nascondere l'esistenza stessa del messaggio.

Tipologie di Steganografia

1. **Steganografia in Immagini**
   - **Least Significant Bit (LSB)**: Una delle tecniche più comuni, che nasconde i dati nei bit meno significativi dei pixel di un'immagine. Cambiando solo gli ultimi bit, le alterazioni sono praticamente invisibili all'occhio umano.
   - **Mascheramento dei Canali di Colore**: Utilizza i canali di colore (RGB) di un'immagine per nascondere informazioni. Ad esempio, si possono modificare i valori del canale blu di un'immagine senza alterarne visibilmente l'aspetto.
2. **Steganografia in Audio**
   - **LSB in Audio**: Simile alla steganografia in immagini, ma applicata ai campioni audio. I bit meno significativi del file audio vengono alterati per nascondere il messaggio.
   - **Modulazione di Ampiezza o Frequenza**: Le variazioni impercettibili dell'ampiezza o della frequenza del segnale audio possono essere usate per trasmettere dati.
3. **Steganografia in Video**
   - **Steganografia nei Frame**: I dati possono essere nascosti in uno o più frame di un video, utilizzando tecniche simili a quelle dell'immagine.
   - **Modulazione dei Pixel**: Modificare i pixel nei frame video per inserire dati, mantenendo il video visivamente intatto.
4. **Steganografia in Testo**
   - **Nascondere Dati nei Testi**: Tecniche come l'uso di caratteri speciali, spazi bianchi, o alterazioni di formato per nascondere informazioni all'interno di un testo normale.
   - **Tecniche di Codifica**: Utilizzo di schemi di codifica e decodifica per incorporare messaggi all'interno di testi. Ad esempio, ogni lettera del testo può essere sostituita da una stringa di lettere che rappresentano un messaggio nascosto.
5. **Steganografia in Metadati**
   - **Exif Metadata**: I dati possono essere nascosti nei metadati di un file, come quelli delle immagini JPEG che contengono informazioni come data, ora, e dettagli della

fotocamera.
    - **File Metadata**: Utilizzare i metadati di documenti e altri file per nascondere informazioni, come nel caso dei file PDF o dei documenti Word.
6. **Steganografia su Reti**
    - **Protocollo di Comunicazione**: Nascondere dati all'interno di pacchetti di rete o di protocolli di comunicazione. Ad esempio, inserendo dati nei campi di un pacchetto IP.

Considerazioni

- **Visibilità**: La steganografia si basa sulla non rilevabilità del messaggio nascosto. Il messaggio deve essere invisibile o difficile da riconoscere.
- **Capacità di Memoria**: La quantità di dati che può essere nascosta dipende dalla tecnica e dal supporto utilizzato.
- **Sicurezza**: Anche se la steganografia può mascherare la presenza di un messaggio, non lo cripta. Combinare la steganografia con la crittografia aumenta la sicurezza.

Importa codificando

```
from PIL import Image

def hide_character(image_path, char, output_image_path):
    img = Image.open(image_path)
    binary_char = format(ord(char), '08b')  # Converti il carattere in binario
(8 bit)

    pixels = list(img.getdata())
    r, g, b = pixels[0]  # Prendi il primo pixel dell'immagine

    # Nascondi i primi 3 bit nei canali R, G, B rispettivamente
    r = (r & 0xF8) | int(binary_char[0:3], 2)
    g = (g & 0xF8) | int(binary_char[3:6], 2)
    b = (b & 0xFC) | int(binary_char[6:8], 2)

    # Aggiorna il pixel
    pixels[0] = (r, g, b)
    img.putdata(pixels)

    img.save(output_image_path)
    print(f"Carattere '{char}' nascosto nel primo pixel dell'immagine.")
```

```
image_path = "immagine_input.png"
output_image_path = "immagine_output.png"
char = "A"

hide_character(image_path, char, output_image_path)
```

Decodifica

```
from PIL import Image

def retrieve_character(image_path):
    img = Image.open(image_path)
    r, g, b = img.getdata()[0]  # Prendi il primo pixel dell'immagine

    # Estrarre i bit dal pixel
    binary_char = f"{r & 0x07:03b}{g & 0x07:03b}{b & 0x03:02b}"

    char = chr(int(binary_char, 2))
    return char

image_path = "immagine_output.png"
hidden_char = retrieve_character(image_path)
print("Carattere estratto:", hidden_char)
```

Spiegazione codice per nascondere un singolo carattere in un pixel

```
from PIL import Image
```

- `from PIL import Image`: Questo importa la libreria Python Imaging Library (PIL), che fornisce strumenti per lavorare con immagini. La libreria PIL (o il suo fork, Pillow) è comunemente usata per caricare, manipolare e salvare immagini in Python.

```
def hide_character(image_path, char, output_image_path):
```

- `def hide_character(image_path, char, output_image_path):` : Questa è la definizione della funzione `hide_character`, che accetta tre argomenti:

- `image_path`: il percorso dell'immagine in cui nascondere il carattere.
- `char`: il carattere da nascondere.
- `output_image_path`: il percorso in cui salvare l'immagine modificata.

```
    img = Image.open(image_path)
```

- `img = Image.open(image_path)`: Questo carica l'immagine specificata da `image_path` e la memorizza nella variabile `img`.

```
    binary_char = format(ord(char), '08b')
```

- `binary_char = format(ord(char), '08b')`:
  - `ord(char)` converte il carattere `char` nel suo valore ASCII corrispondente (un numero intero).
  - `format(..., '08b')` converte questo numero intero in una stringa binaria di 8 bit (inclusi eventuali zeri iniziali). Ad esempio, il carattere `'A'` diventa `'01000001'`.

```
    pixels = list(img.getdata())
```

- `pixels = list(img.getdata())`: Questo estrae i dati dei pixel dell'immagine sotto forma di una lista. Ogni pixel è rappresentato come una tupla di valori RGB (Red, Green, Blue), ad esempio `(255, 0, 0)`.

```
    r, g, b = pixels[0]
```

- `r, g, b = pixels[0]`: Qui stiamo prendendo il primo pixel dall'immagine (che è una tupla di tre valori) e assegnando i valori rispettivamente alle variabili `r`, `g`, e `b` (i valori rosso, verde e blu del primo pixel).

```
    r = (r & 0xF8) | int(binary_char[0:3], 2)
    g = (g & 0xF8) | int(binary_char[3:6], 2)
    b = (b & 0xFC) | int(binary_char[6:8], 2)
```

- **Queste tre righe** servono per nascondere i bit del carattere binario nei canali di colore RGB del pixel:
  - `r & 0xF8`: L'operatore `&` è un AND bit-a-bit. `0xF8` è una maschera binaria (`11111000` in binario), che mantiene i 5 bit più significativi del canale rosso `r` e azzera i 3 meno significativi.

- `int(binary_char[0:3], 2)` converte i primi 3 bit del carattere binario in un numero intero.
- `r = (r & 0xF8) | ...`: L'operatore `|` è un OR bit-a-bit. Combina i 5 bit significativi originali di `r` con i 3 bit del carattere nascosto.
- La stessa logica si applica a `g` e `b`, ma con 3 bit per `g` e 2 bit per `b`.

```
    pixels[0] = (r, g, b)
```

- `pixels[0] = (r, g, b)`: Aggiorna il primo pixel dell'immagine con i nuovi valori RGB modificati.

```
    img.putdata(pixels)
```

- `img.putdata(pixels)`: Questo inserisce nuovamente i dati modificati nell'immagine.

```
    img.save(output_image_path)
    print(f"Carattere '{char}' nascosto nel primo pixel dell'immagine.")
```

- `img.save(output_image_path)`: Salva l'immagine modificata con il carattere nascosto al percorso specificato in `output_image_path`.
- `print(...)`: Stampa un messaggio di conferma indicando che il carattere è stato nascosto.

Spiegazione codice per recuperare un singolo Carattere da un pixel

```
from PIL import Image
```

- Stessa importazione come prima.

```
def retrieve_character(image_path):
    img = Image.open(image_path)
```

- `def retrieve_character(image_path):`: Questa funzione accetta un argomento, `image_path`, che è il percorso dell'immagine da cui recuperare il carattere nascosto.

```
    r, g, b = img.getdata()[0]
```

- `r, g, b = img.getdata()[0]`: Prende il primo pixel dell'immagine e ne estrae i valori RGB.

```
    binary_char = f"{r & 0x07:03b}{g & 0x07:03b}{b & 0x03:02b}"
```

- `binary_char = f"{r & 0x07:03b}{g & 0x07:03b}{b & 0x03:02b}"`:
  - `r & 0x07`: Estrae i 3 bit meno significativi di `r` (usando la maschera `0x07` che è `00000111` in binario).
  - `g & 0x07`: Estrae i 3 bit meno significativi di `g`.
  - `b & 0x03`: Estrae i 2 bit meno significativi di `b`.
  - `f"{...}"`: Crea una stringa binaria concatenando i bit estratti da `r`, `g` e `b`.

```
    char = chr(int(binary_char, 2))
    return char
```

- `chr(int(binary_char, 2))`:
  - `int(binary_char, 2)` converte la stringa binaria `binary_char` in un numero intero.
  - `chr(...)` converte questo numero intero nel carattere corrispondente.
  - Infine, il carattere viene restituito dalla funzione.

Convertire l'Immagine in un Formato Supportato

Se stai utilizzando Steghide e hai un'immagine in formato PNG, potresti doverla convertire in un formato supportato come JPEG o BMP. Di seguito sono riportati i passaggi per farlo utilizzando ImageMagick.

1. Installa ImageMagick

Prima di tutto, devi installare ImageMagick se non lo hai già fatto. Usa il seguente comando per l'installazione su Debian/Parrot OS:

```
sudo apt-get install imagemagick
```

2. Converti l'Immagine PNG in JPEG

Una volta installato ImageMagick, puoi convertire il tuo file immagine PNG in JPEG con il comando seguente:

```
convert iRONuNi.png iRONuNi.jpg
```

3. Esegui Steghide con il Nuovo File JPEG

Ora puoi usare Steghide per nascondere un messaggio nel nuovo file JPEG. Usa il comando seguente per inserire il messaggio nel file JPEG:

```
steghide embed -cf iRONuNi.jpg -ef secret.txt
```

4. Estrai il Messaggio

Per estrarre il messaggio dal file JPEG, usa il comando seguente:

```
steghide extract -sf iRONuNi.jpg -xf secret.txt
```

Collegamenti ai File

Puoi inserire collegamenti ai tuoi file direttamente nella nota per un facile accesso. Ad esempio:

- **Immagine JPEG**: iRONuNi.jpg
- **File di Testo Segreto**: secret.txt

Assicurati che i file `iRONuNi.jpg` e `secret.txt` si trovino nella stessa cartella della tua nota Obsidian o in una sottocartella pertinente. In caso contrario, modifica i percorsi dei collegamenti in base alla loro posizione.

Esempio Completo in Obsidian

```
# Convertire l'Immagine in un Formato Supportato

Se stai utilizzando Steghide e hai un'immagine in formato PNG, potresti
doverla convertire in un formato supportato come JPEG o BMP. Di seguito sono
```

```
riportati i passaggi per farlo utilizzando ImageMagick.

## 1. Installa ImageMagick

Prima di tutto, devi installare ImageMagick se non lo hai già fatto. Usa il
seguente comando per l'installazione su Debian/Parrot OS:

```bash
sudo apt-get install imagemagick
```

2. Converti l'Immagine PNG in JPEG

Una volta installato ImageMagick, puoi convertire il tuo file immagine PNG in JPEG con il comando seguente:

```
convert iRONuNi.png iRONuNi.jpg
```

3. Esegui Steghide con il

Collegamenti ai File

- **Immagine JPEG**: iRONuNi.jpg
- **File di Testo Segreto**: secret.txt

Riepilogo

1. **Installa ImageMagick**: `sudo apt-get install imagemagick`
2. **Converti l'immagine**: `convert iRONuNi.png iRONuNi.jpg`
3. **Nascondi il messaggio**: `steghide embed -cf iRONuNi.jpg -ef secret.txt`
4. **Estrai il messaggio**: `steghide extract -sf iRONuNi.jpg -xf secret.txt`
```