Steg

```
#!/usr/bin/env python3
"""
Steganography Extraction Tool

This script extracts hidden data from images using various steganography techniques.
Usage: python steg_extract.py <image_file>
"""

import sys
import os
import numpy as np
from PIL import Image
import binascii
import re
import zlib
import struct
from bitstring import BitArray

def extract_lsb(image_path, bit_depth=1):
    """Extract data hidden using LSB (Least Significant Bit) steganography."""
    try:
        img = Image.open(image_path)
        pixels = np.array(img)

        # Flatten the pixel array and extract LSBs
        flat_pixels = pixels.flatten()

        # Get the least significant bits
        bits = ""
        for pixel in flat_pixels:
            # Extract the specified number of least significant bits
            for i in range(bit_depth):
                bits += str((pixel >> i) & 1)

        # Convert bits to bytes
        bytes_data = BitArray(bin=bits).bytes
```

```
        # Try to find printable text
        printable_data = ""
        for i in range(len(bytes_data)):
            char = bytes_data[i:i+1]
            if 32 <= ord(char) <= 126 or ord(char) in (10, 13, 9):  # Printable ASCII or
newline/tab
                printable_data += char.decode('ascii', errors='ignore')
            else:
                printable_data += '.'

        return {
            'raw_bits': bits[:100] + "...",  # First 100 bits
            'raw_bytes': binascii.hexlify(bytes_data[:50]).decode('ascii') + "...",  # First
50 bytes
            'possible_text': printable_data[:1000]  # First 1000 printable chars
        }
    except Exception as e:
        return {'error': f"LSB extraction failed: {str(e)}"}

def extract_metadata(image_path):
    """Extract metadata from the image that might contain hidden information."""
    try:
        img = Image.open(image_path)
        metadata = {}

        # Extract EXIF data if available
        if hasattr(img, '_getexif') and img._getexif():
            metadata['exif'] = str(img._getexif())

        # Extract other metadata
        metadata['format'] = img.format
        metadata['mode'] = img.mode
        metadata['info'] = str(img.info)

        return metadata
    except Exception as e:
        return {'error': f"Metadata extraction failed: {str(e)}"}

def extract_color_plane(image_path):
    """Extract data from color planes separately to find potential hidden information."""
```

```
    try:
        img = Image.open(image_path)
        if img.mode != 'RGB' and img.mode != 'RGBA':
            return {'error': "Not an RGB/RGBA image"}

        planes = {}
        pixels = np.array(img)

        # Extract red, green, blue planes
        if img.mode == 'RGB' or img.mode == 'RGBA':
            planes['red'] = pixels[:,:,0]
            planes['green'] = pixels[:,:,1]
            planes['blue'] = pixels[:,:,2]

        # Check for unusual patterns in each plane
        results = {}
        for plane_name, plane_data in planes.items():
            # Look for unusual distributions (e.g., even/odd patterns)
            even_count = np.sum(plane_data % 2 == 0)
            odd_count = np.sum(plane_data % 2 == 1)

            # If there's a significant imbalance, it might indicate steganography
            results[f"{plane_name}_analysis"] = {
                'even_pixels': even_count,
                'odd_pixels': odd_count,
                'imbalance': abs(even_count - odd_count) / (even_count + odd_count)
            }

            # Extract LSB from this color plane only
            bits = "".join([str(p & 1) for p in plane_data.flatten()])
            results[f"{plane_name}_lsb_sample"] = bits[:100] + "..."

        return results
    except Exception as e:
        return {'error': f"Color plane extraction failed: {str(e)}"}

def extract_hidden_files(image_path):
    """Look for embedded files using common signatures/headers."""
    try:
        with open(image_path, 'rb') as f:
            data = f.read()
```

```
        # Common file signatures to look for
        file_signatures = {
            b'\x50\x4B\x03\x04': 'ZIP',
            b'\x52\x61\x72\x21\x1A\x07': 'RAR',
            b'\x25\x50\x44\x46': 'PDF',
            b'\xFF\xD8\xFF': 'JPG',
            b'\x89\x50\x4E\x47': 'PNG',
            b'\x47\x49\x46\x38': 'GIF',
            b'\x7F\x45\x4C\x46': 'ELF',
            b'\xD0\xCF\x11\xE0': 'MS Office',
            b'\x50\x4B\x05\x06': 'ZIP (empty)',
            b'\x1F\x8B\x08': 'GZIP',
            b'\x42\x5A\x68': 'BZ2',
            b'\x75\x73\x74\x61\x72': 'TAR',
            b'\x49\x44\x33': 'MP3',
            b'\x4D\x5A': 'EXE',
        }

        found_files = []
        for signature, filetype in file_signatures.items():
            # Find all occurrences of the signature
            offsets = [m.start() for m in re.finditer(re.escape(signature), data)]
            for offset in offsets:
                found_files.append({
                    'type': filetype,
                    'offset': offset,
                    'signature': binascii.hexlify(signature).decode('ascii')
                })

        return found_files
    except Exception as e:
        return {'error': f"Hidden file extraction failed: {str(e)}"}

def extract_parity_steganography(image_path):
    """Check for parity-based steganography."""
    try:
        img = Image.open(image_path)
        pixels = np.array(img)

        # Count the parity of pixels in each row and column
```

```
        row_parity = np.sum(pixels.sum(axis=2) % 2, axis=1) % 2
        col_parity = np.sum(pixels.sum(axis=2) % 2, axis=0) % 2

        # Convert to binary strings (potentially hidden messages)
        row_message = "".join([str(int(bit)) for bit in row_parity])
        col_message = "".join([str(int(bit)) for bit in col_parity])

        return {
            'row_parity_bits': row_message,
            'col_parity_bits': col_message
        }
    except Exception as e:
        return {'error': f"Parity steganography extraction failed: {str(e)}"}

def extract_hidden_text(image_path):
    """Extract text from the image using several methods."""
    try:
        with open(image_path, 'rb') as f:
            data = f.read()

        # Look for ASCII/UTF-8 text patterns
        possible_strings = []
        ascii_regex = rb'[ -~\r\n\t]{8,}'  # 8+ printable ASCII chars
        for match in re.finditer(ascii_regex, data):
            possible_strings.append(match.group(0).decode('ascii', errors='ignore'))

        return {
            'possible_strings': possible_strings[:20]  # Return first 20 found strings
        }
    except Exception as e:
        return {'error': f"Text extraction failed: {str(e)}"}

def analyze_bit_distribution(image_path):
    """Analyze bit distribution for statistical anomalies."""
    try:
        img = Image.open(image_path)
        pixels = np.array(img)

        # Analyze distribution of each bit position
        bit_counts = []
        for bit_pos in range(8):
```

```
            mask = 1 << bit_pos
            bit_count = np.sum((pixels & mask) > 0)
            bit_counts.append(bit_count)

        total_bits = pixels.size * 8
        bit_frequencies = [count / total_bits for count in bit_counts]

        # Calculate deviation from expected 0.5 frequency
        deviations = [abs(freq - 0.5) for freq in bit_frequencies]

        return {
            'bit_frequencies': bit_frequencies,
            'deviations': deviations,
            'analysis': "High deviation in LSBs may indicate steganography"
        }
    except Exception as e:
        return {'error': f"Bit distribution analysis failed: {str(e)}"}

def extract_stegano_data(image_path):
    """Main function to extract steganographic data from an image."""
    results = {
        'filename': os.path.basename(image_path),
        'filesize': os.path.getsize(image_path)
    }

    # Run all extraction methods
    results['lsb_extraction'] = extract_lsb(image_path)
    results['lsb_extraction_2bit'] = extract_lsb(image_path, bit_depth=2)
    results['metadata'] = extract_metadata(image_path)
    results['color_planes'] = extract_color_plane(image_path)
    results['hidden_files'] = extract_hidden_files(image_path)
    results['parity_data'] = extract_parity_steganography(image_path)
    results['text_data'] = extract_hidden_text(image_path)
    results['bit_distribution'] = analyze_bit_distribution(image_path)

    return results

def save_extracted_data(results, original_image_path):
    """Save extracted data to files."""
    base_name = os.path.splitext(os.path.basename(original_image_path))[0]
    output_dir = f"{base_name}_extracted"
```

```
    # Create directory if it doesn't exist
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Save main results as text
    with open(f"{output_dir}/results.txt", 'w') as f:
        f.write(f"Steganography Extraction Results for {results['filename']}\n")
        f.write(f"File size: {results['filesize']} bytes\n\n")

        # Write metadata
        f.write("=== METADATA ===\n")
        for k, v in results['metadata'].items():
            f.write(f"{k}: {v}\n")

        # Write LSB extraction results
        f.write("\n=== LSB EXTRACTION ===\n")
        f.write(f"LSB sample: {results['lsb_extraction']['raw_bits']}\n")
        f.write(f"Bytes: {results['lsb_extraction']['raw_bytes']}\n")
        f.write(f"Possible text:\n{results['lsb_extraction']['possible_text']}\n")

        # Write 2-bit LSB extraction
        f.write("\n=== 2-BIT LSB EXTRACTION ===\n")
        f.write(f"LSB sample: {results['lsb_extraction_2bit']['raw_bits']}\n")
        f.write(f"Bytes: {results['lsb_extraction_2bit']['raw_bytes']}\n")
        f.write(f"Possible text:\n{results['lsb_extraction_2bit']['possible_text']}\n")

        # Write color plane analysis
        f.write("\n=== COLOR PLANE ANALYSIS ===\n")
        for k, v in results['color_planes'].items():
            f.write(f"{k}: {v}\n")

        # Write hidden files
        f.write("\n=== POSSIBLE HIDDEN FILES ===\n")
        for file_info in results['hidden_files']:
            f.write(f"Type: {file_info['type']}, Offset: {file_info['offset']}, Signature:
{file_info['signature']}\n")

        # Write parity data
        f.write("\n=== PARITY STEGANOGRAPHY ===\n")
        f.write(f"Row parity: {results['parity_data']['row_parity_bits']}\n")
```

```
        f.write(f"Column parity: {results['parity_data']['col_parity_bits']}\n")

        # Write found text strings
        f.write("\n=== POSSIBLE HIDDEN TEXT ===\n")
        for s in results['text_data']['possible_strings']:
            f.write(f"{s}\n")
            f.write("---\n")

        # Write bit distribution analysis
        f.write("\n=== BIT DISTRIBUTION ANALYSIS ===\n")
        f.write("Bit position frequencies (0-7, LSB to MSB):\n")
        for i, freq in enumerate(results['bit_distribution']['bit_frequencies']):
            f.write(f"Bit {i}: {freq:.4f} (deviation:
{results['bit_distribution']['deviations'][i]:.4f})\n")

    # If we found potential embedded files, try to extract them
    if results['hidden_files']:
        with open(original_image_path, 'rb') as f:
            data = f.read()

        for i, file_info in enumerate(results['hidden_files']):
            # Create a name for the extracted file
            ext = file_info['type'].lower().split()[0]  # Use the first word of the type as
extension
            output_file = f"{output_dir}/extracted_file_{i}.{ext}"

            # Get start position from offset
            start_pos = file_info['offset']

            # Write the data to a file, up to 10MB maximum
            with open(output_file, 'wb') as out_f:
                out_f.write(data[start_pos:start_pos + 10*1024*1024])

    return output_dir

def main():
    if len(sys.argv) != 2:
        print(f"Usage: {sys.argv[0]} <image_file>")
        sys.exit(1)

    image_path = sys.argv[1]
```

```
    if not os.path.exists(image_path):
        print(f"Error: File '{image_path}' not found.")
        sys.exit(1)

    print(f"Analyzing {image_path} for steganographic data...")
    results = extract_stegano_data(image_path)

    # Save results to files
    output_dir = save_extracted_data(results, image_path)
    print(f"Analysis complete. Results saved to {output_dir}/")

if __name__ == "__main__":
    main()
```

Revision #1

Created 2025-11-25 18:07:34 UTC by David Rizzo

Updated 2025-11-25 18:07:42 UTC by David Rizzo