Prodigy InfoTech Cyber Security Report

Week 2: Pixel Manipulation for Image Encryption

Full Name: Sairaj Turalkar

Date: 08/06/2024

1. Introduction

In the second week of my cybersecurity internship at Prodify Infotech, I was assigned to develop a simple image encryption tool using pixel manipulation. This project involved creating a Python program to perform encryption and decryption of images by swapping pixel values or applying basic mathematical operations to each pixel.

2. Objective

The objective of this task was to build a tool that could encrypt and decrypt images using pixel manipulation techniques. The tool should allow users to specify an image and an encryption key to perform the operations.

3. Methodology

3.1. Python Program

The implementation includes three main functions: encrypt_image, decrypt_image, and main.

Encrypt Image Function

The encrypt_image function takes an image path and a key as input. It loads the image, converts it to a numpy array, and applies a shuffling algorithm to encrypt the pixel values. The encrypted image is then saved.

```
def encrypt_image(image_path, key):
    img = Image.open(image_path)
    img_array = np.array(img)
```

```
    encrypted_array = np.copy(img_array)
    np.random.seed(key)
    np.random.shuffle(encrypted_array.flat)

    encrypted_img = Image.fromarray(encrypted_array)
    encrypted_img.save("encrypted_image.png")
    print("Image encrypted and saved as encrypted_image.png")
```

Decrypt Image Function

The decrypt_image function reverses the encryption process. It takes the encrypted image path and the key, then restores the original pixel arrangement using the same key.

```
def decrypt_image(encrypted_image_path, key):
    encrypted_img = Image.open(encrypted_image_path)
    encrypted_array = np.array(encrypted_img)

    decrypted_array = np.copy(encrypted_array)
    flat_indices = np.arange(decrypted_array.size)
    np.random.seed(key)
    np.random.shuffle(flat_indices)
    decrypted_array.flat[flat_indices] = encrypted_array.flat

    decrypted_img = Image.fromarray(decrypted_array)
    decrypted_img.save("decrypted_image.png")
    print("Image decrypted and saved as decrypted_image.png")
```

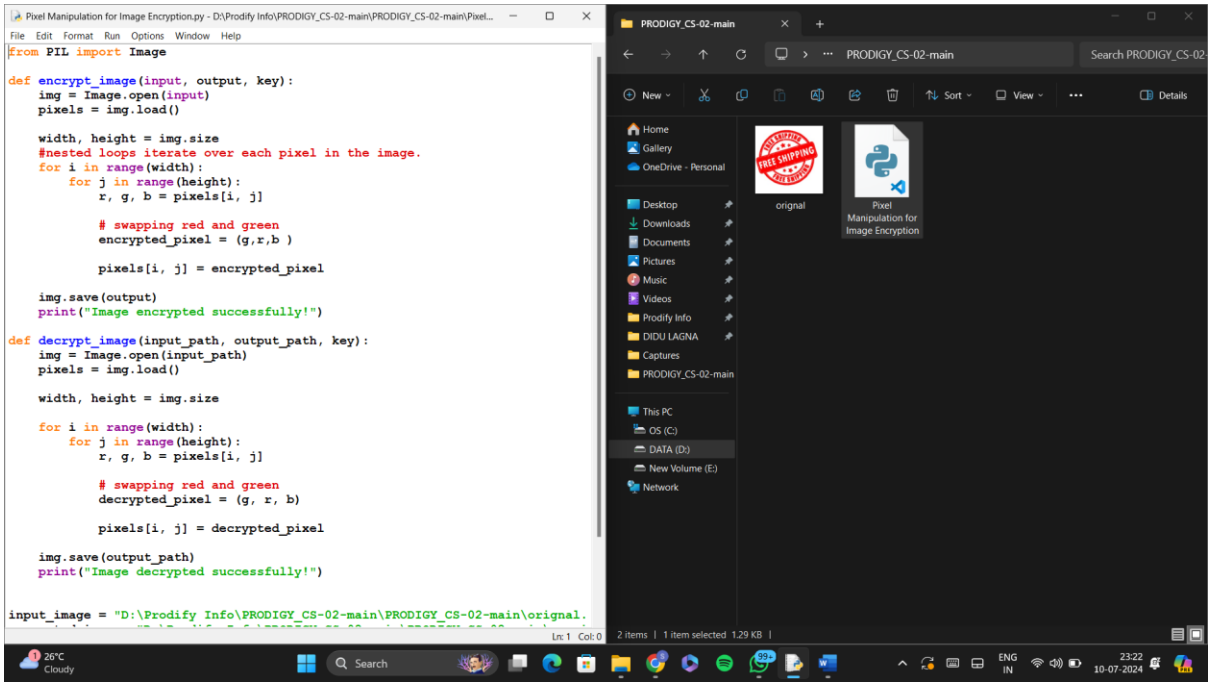
Main Function

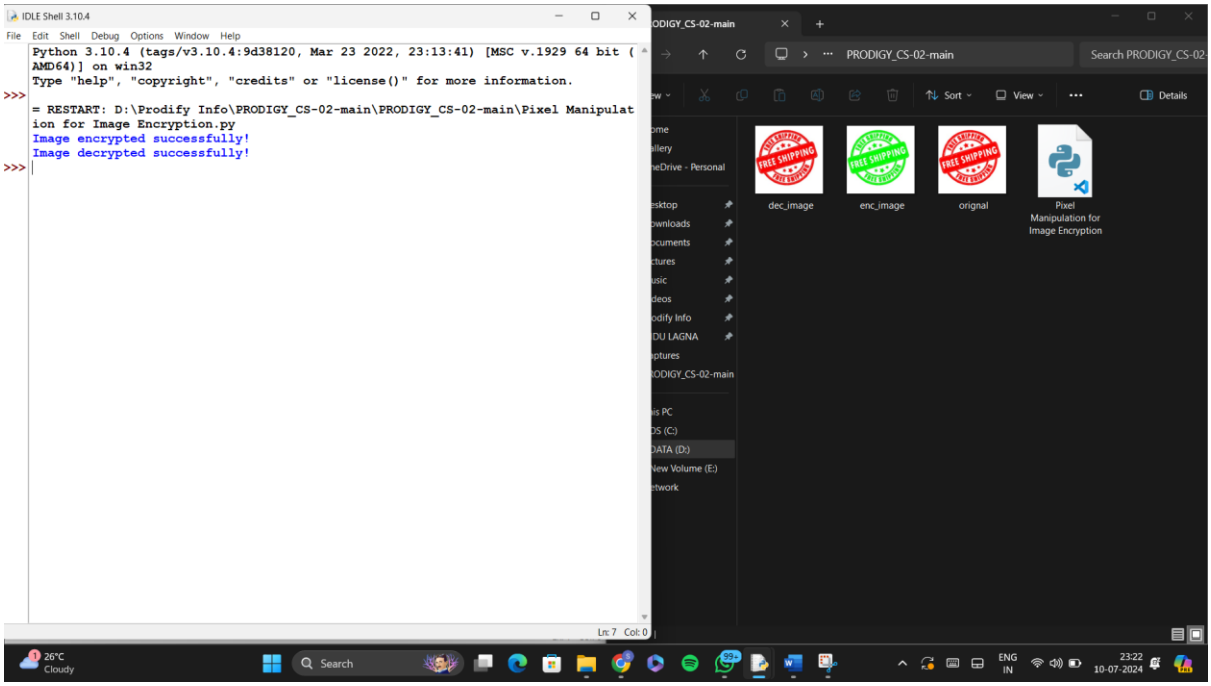
The main function serves as the user interface, allowing users to choose between encryption and decryption, specify the image path, and provide an encryption key.

```
def main():
    while True:
        choice = input("Do you want to (e)ncrypt or (d)ecrypt an image? ")
        image_path = input("Enter the image path: ")
        key = int(input("Enter the encryption key (a number): "))

        if choice.lower() == 'e':
            encrypt_image(image_path, key)
        elif choice.lower() == 'd':
            decrypt_image(image_path, key)
        else:
            print("Invalid choice. Please choose 'e' for encryption or 'd' for decryption.")

        repeat = input("Do you want to perform another operation? (if yes press 'y' / if no press 'n'): ")
        if repeat.lower() != 'y':
            print("Goodbye!")
            break

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

4. Testing and Results

To ensure the accuracy of the implementation, the program was tested with various images and keys. The encrypted and decrypted images matched the expected outcomes, confirming the tool's functionality.





5. Conclusion

This task provided valuable insights into pixel manipulation techniques for image encryption. Implementing this tool in Python enhanced my understanding of image processing and cryptographic principles. The experience gained from this project will be beneficial for more advanced cybersecurity endeavors.

6. References

- Python Imaging Library (PIL): Pillow Documentation