USER

Code:

```
import string

LOWERCASE_OFFSET = ord("a")
ALPHABET = string.ascii_lowercase[:16]

def b16_encode(plain):
enc = ""
for c in plain:
binary = "{0:08b}".format(ord(c))
enc += ALPHABET[int(binary[:4], 2)]
enc += ALPHABET[int(binary[4:], 2)]
return enc

def shift(c, k):
t1 = ord(c) - LOWERCASE_OFFSET
t2 = ord(k) - LOWERCASE_OFFSET
return ALPHABET[(t1 + t2) % len(ALPHABET)]

flag = "redacted"
assert all([c in "abcdef0123456789" for c in flag])

key = "redacted"
assert all([k in ALPHABET for k in key]) and len(key) < 15

b16 = b16_encode(flag)
enc = ""
for i, c in enumerate(b16):
enc += shift(c, key[i % len(key)])
print(enc)
```

New Vignere
Author: madStacks

Description
Another slight twist on a classic, see if you can recover the flag. (Wrap with picoCTF{})
eljodmjdjcnfcdmgbleojbgngojkkdpimebgeigpdkjpmgngpfpgelemjoglghjd new_vignere.py

ASSISTANT

Identify Vulnerability:

The given code appears to implement a simple Vigenère cipher, a series of interwoven Caesar ciphers, based on the letters of a keyword. However, there is no evident vulnerability in this code. It is simply a cipher implementation.

However, one thing to note is that Vigenère ciphers can be vulnerable to Kasiski examination or frequency analysis if the key length is short. Additionally, if the key is significantly shorter than the plaintext, then multiple Caesar ciphers will be applied in repeating patterns, which can make the cipher easier to break.

In this context, to break this code you could apply Kasiski examination. In Kasiski Examination, you're looking for repeated sequences of more than 3 characters. When you find one of these, you're hoping that both occurrences were encrypted using the same few letters of the key.

Exploit Vulnerability:

```python
def calculate_prob(el1, el2, corpus):
    try:
        prob1 = corpus[el1] / sum(corpus.values())
        prob2 = corpus[el2] / sum(corpus.values())
        return prob1 * prob2
    except:
        return None
corpus = {}
text = "eljodmjdjcnfcdmgbleojbgngojkkdpimebgeigpdkjpmgngpfpgelemjoglghjd"
for i in range(len(text)):
    for j in range(i + 3, len(text), 3):
        str1 = text[i:j]
        str2 = text[j : j + (j - i)]
        if str1 == str2 and (str1 not in no_doubles):
            print(str1)

maximal_prob = 0
possible_key_length = 0
corpus = {}
for i in range(len(text)):
    for j in range(i + 3, len(text), 3):
        str1 = text[i:j]
        str2 = text[j : j + (j - i)]
        if str1 == str2:
            try:
                prob = calculate_prob(str1[0], str1[1], corpus)
                if prob > maximal_prob:
                    for letter in range(ord('a'), ord('z') + 1):
                        if (ord(str1[0]) - letter) % 16 == (ord(str1[1]) - letter) % 16 and letter > maximal_prob:
```

maximal_prob = letter
possible_key_length = j - i
except:
pass```

This is just an exercise to determine how we could start to break this sort of encryption. Since we do not have a definitive length of the key and no hint on how (to be simple given directly) we would go on from here.

Fix Vulnerability:

This problem seems to be, as presented, primarily related to traditional paper-and-pencil cryptology techniques. Given the constraints below we can adapt these classic problems such as the Kasiski examination and frequency analysis with some heuristic tricks to find an easier path to exploit the system by focusing on higher probability events and with them in the end deriving or at least giving clues on the original text.

But remember to note, this vulnerability can be primarily mitigated through careful design and security evaluations of your ciphers.

In order to securely implement a cipher in an application a set of standards like OWASP or the like must be considered and followed all along the architecture and the lifecycle.

USER

How to solve the problem

ASSISTANT

Classify the Challenge:

The challenge is a cryptography problem, specifically a Vigenère cipher.

Approach to Solve:

1. Understand the Vigenère cipher and its weaknesses. The Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to encrypt and decrypt messages. One of its main weaknesses is that if the keyword is shorter than the message, it will repeat itself, making it easier to break.

2. Look for repeated patterns in the ciphertext. Since the keyword is likely to be shorter than the message, there may be repeated patterns in the ciphertext that can be used to deduce the length of the keyword.

3. Use the Kasiski examination method to find the length of the keyword. This method involves looking for repeated sequences of letters in the ciphertext and measuring the distance between them. If the distance is a multiple of the keyword length, it may indicate the length of the keyword.

4. Once the length of the keyword is known, use frequency analysis to deduce the actual keyword. Since the Vigenère cipher is a polyalphabetic substitution cipher, each letter in the ciphertext corresponds to a

different letter in the alphabet, depending on the keyword. By analyzing the frequency of letters in the ciphertext, it may be possible to deduce the keyword.

5. Once the keyword is known, use it to decrypt the ciphertext and recover the original message.

Here's a step-by-step solution in Python:
```python
def calculate_prob(el1, el2, corpus):
    try:
        prob1 = corpus[el1] / sum(corpus.values())
        prob2 = corpus[el2] / sum(corpus.values())
        return prob1 * prob2
    except:
        return None

def vigenere_cipher_decrypt(ciphertext, keyword):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    keyword_index = 0
    plaintext = ''
    for char in ciphertext:
        shift = alphabet.index(keyword[keyword_index % len(keyword)]) % 16
        plaintext += alphabet[(alphabet.index(char) - shift) % 16]
        keyword_index += 1
    return plaintext

ciphertext = "eljodmjdjcnfcdmgbleojbgngojkkdpimebgeigpdkjpmgngpfpgelemjoglghjd"

# Find repeated patterns in the ciphertext
for i in range(len(ciphertext)):
    for j in range(i + 3, len(ciphertext), 3):
        str1 = ciphertext[i:j]
        str2 = ciphertext[j : j + (j - i)]
        if str1 == str2:
            print(str1)

# Use Kasiski examination to find the length of the keyword
maximal_prob = 0
possible_key_length = 0
corpus = {}
for i in range(len(ciphertext)):
    for j in range(i + 3, len(ciphertext), 3):
        str1 = ciphertext[i:j]
        str2 = ciphertext[j : j + (j - i)]
        if str1 == str2:
            try:
```

```
                prob = calculate_prob(str1[0], str1[1], corpus)
                if prob > maximal_prob:
                    for letter in range(ord('a'), ord('z') + 1):
                        if (ord(str1[0]) - letter) % 16 == (ord(str1[1]) - letter) % 16 and letter > maximal_prob:
                            maximal_prob = letter
                            possible_key_length = j - i
            except:
                pass

# Use frequency analysis to deduce the actual keyword
keyword = ''
for i in range(possible_key_length):
    max_prob = 0
    letter = ''
    for char in 'abcdefghijklmnopqrstuvwxyz':
        prob = calculate_prob(ciphertext[i], char, corpus)
        if prob > max_prob:
            max_prob = prob
            letter = char
    keyword += letter

# Decrypt the ciphertext using the keyword
plaintext = vigenere_cipher_decrypt(ciphertext, keyword)
print(plaintext)
```

This solution is based on the steps outlined above and uses Python to implement the Kasiski examination and frequency analysis methods to deduce the keyword and decrypt the ciphertext.

USER

That did not work

ASSISTANT