Background

The challenge is a cryptographic exercise that centers on exploiting the vulnerabilities of AES-CBC encryption [1] [3]. One inherent characteristic of CBC mode is its malleability, which means that an attacker can manipulate the ciphertext by performing bit-flipping attacks [2] to induce controlled changes in the decrypted plaintext. This is possible because alterations to one ciphertext block affect the decryption of the subsequent block, allowing an adversary to modify specific parts of the plaintext without knowing the encryption key.
The vulnerability arises because AES-CBC does not provide any message integrity checks. Without incorporating additional measures, such as a Message Authentication Code (MAC) [5] or employing authenticated encryption modes like AES-GCM [4], the system remains vulnerable to modification attacks even when the encryption algorithm itself is considered secure. The lack of integrity verification permits unauthorized alterations that can result in significant security issues.
These vulnerabilities can have severe consequences, including unauthorized data manipulation, loss of message integrity, and exposure of sensitive information. The absence of proper integrity checks can allow attackers to alter configuration data, tamper with transaction details, or inject malicious commands.
Countermeasures include using authenticated encryption schemes or robust integrity checks, essential to mitigate the mentioned risks and ensure confidentiality and integrity of the data.

Vulnerability

In the challenge, a string representing a zoo gets created. The format of the plaintext, where fixed tokens (i.e., "pet=", delimiters, and padding elements) are concatenated with user-supplied data, creates a predictable layout.
This string of animal names is then encrypted using AES in CBC mode with a fixed Initialization Vector, no integrity checks and a well-defined (and known) plaintext structure, enabling for the retrieval of the flag.
The injection point is in the same place for both versions, but the string concatenation and sanitization differs one from the other.

Solution

The program will return the flag if in the zoo there are some specific animals, but it performs sanitization before creating the zoo string. The block size for CBC is 16 bytes for both challenges. In the first version, it needs to contain both *Ferris* and *rubberduck*. Since sanitization checks if either of the animals is inserted by the user, the solution is to insert some similar names and perform a bit-flip attack to change some characters during decryption. A possible block configuration can be the one found in `first_exploit` function in Listing 1.
CBC decryption works as follow: $P_i = D_K(C_i) \oplus C_{i-1}$, meaning that if a portion of the previous chipertext block C_{i-1} gets purposefully changed, it would cause the output of $D_K(C_i)$ to get XORed with the tampered portion.
In the first version, by flipping some bits of the IV, the first block `pet=Ferriy|pet=u` can be changed to obtain `pet=Ferris|pet=r` which solves the challenge.
The second version is a bit more cumbersome as on top of requiring *BillTheRock* to also be in the zoo to obtain the flag, it limits the number of bytes that can differ in the ciphertext to less than 5 (as the ciphertext gets hex-encoded), restricting the possible tampering.
This can be bypassed by flipping only either the low or high nibble of some ciphertext characters, so only half of the hex-version changes. The animal names have need to contain specific characters that differ from the ones that need to be obtained only by a nibble.

A possible block configuration can be the one in `second_exploit` function in Listing 1.

```python
from pwn import remote

BLOCK_SIZE = 16

def first_exploit(r):
    r.sendlineafter(b'> ', b'1')
    r.sendlineafter(b'> ', b'Ferriy')      # Blocks configuration
    r.sendlineafter(b'> ', b'uubberduck') # "IVIVIVIVIVIVIVIV"
                                          # "pet=Ferriy|pet=u"
                                          # "ubberduck0000000"
    r.recvuntil(b': ')
    iv_ct = bytearray.fromhex(r.recvline().strip().decode())
    iv = iv_ct[:BLOCK_SIZE]; ct = iv_ct[BLOCK_SIZE:]

    delta1 = ord('y') ^ ord('s')
    iv[9] ^= delta1
    delta2 = ord('u') ^ ord('r')
    iv[-1] ^= delta2

    modified_ct = bytes(iv) + bytes(ct)

    r.sendlineafter(b'> ', b'2')
    r.sendlineafter(b'> ', modified_ct.hex().encode())

    r.recvline(); r.recvline()
    print(f"First flag: {r.recvline().decode().strip()}") # Flag

def second_exploit(r):
    r.sendlineafter(b'> ', b'1')
    r.sendlineafter(b'> ', b'A' * 23)
    r.sendlineafter(b'> ', b',pet;vubberduck,AAAAAAAAAAAAAAA')
    # Blocks configuration
    # "IVIVIVIVIVIVIVIV" Initialization Vector
    # "pet=BillTheRock|" Must not change
    # "pet=AAAAAAAAAAAA" First animal used for bitflips
    # "AAAAAAAAAAA|pet=" ...
    # "Xpet;vubberduckX" -> |pet=rubberduck|
    # "AAAAAAAAAAAAAAAA" Second part of second animal used for bitflips
    # "|pet=terristhepr" -> |pet=Ferris|hepr
    # "ill0000000000000"

    r.recvuntil(b': ')
    iv_ct = bytearray.fromhex(r.recvline().strip().decode())
    iv = iv_ct[:BLOCK_SIZE]; ct = iv_ct[BLOCK_SIZE:]

    ct[32] ^= (0x2 ^ 0x7) << 4   # high nibble bit flip:  , (0x2c) -> | (0x7c)
    ct[36] ^= (0xb ^ 0xd)        # low nibble bit flip:   ; (0x3b) -> = (0x3d)
    ct[37] ^= (0x6 ^ 0x2)        # low nibble bit flip:   v (0x76) -> r (0x72)
    ct[47] ^= (0x2 ^ 0x7) << 4   # high nibble bit flip:  , (0x2c) -> | (0x7c)
    delta3 = ord('t') ^ ord('F') # full byte bit flip:    t (0x74) -> F (0x46)
    ct[69] ^= delta3
    ct[75] ^= (0x4 ^ 0xc)        # low nibble bit flip:   t (0x74) -> | (0x7c)

    modified_ct = bytes(iv) + bytes(ct)
    r.sendlineafter(b'> ', b'2')
    r.sendlineafter(b'> ', modified_ct.hex().encode())

    r.recvline()
    print(f"Second flag: {r.recvline().decode().strip()}") # Flag


```

```python
# ======== First version ========
r1 = remote('cyberchallenge.disi.unitn.it', 50100)
first_exploit(r1)

# ======== Second version ========
r2 = remote('cyberchallenge.disi.unitn.it', 50101)
second_exploit(r2)
```

Listing 1: Solution code

References

[1] Wikipedia. AES. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

[2] Wikipedia. Bit-flipping. https://en.wikipedia.org/wiki/Stream_cipher_attacks#Bit-flipping_attack.

[3] Wikipedia. CBC. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

[4] Wikipedia. Gcm. https://en.wikipedia.org/wiki/Galois/Counter_Mode.

[5] Wikipedia. MAC. https://en.wikipedia.org/wiki/Message_authentication_code.