AESWT PoC 2

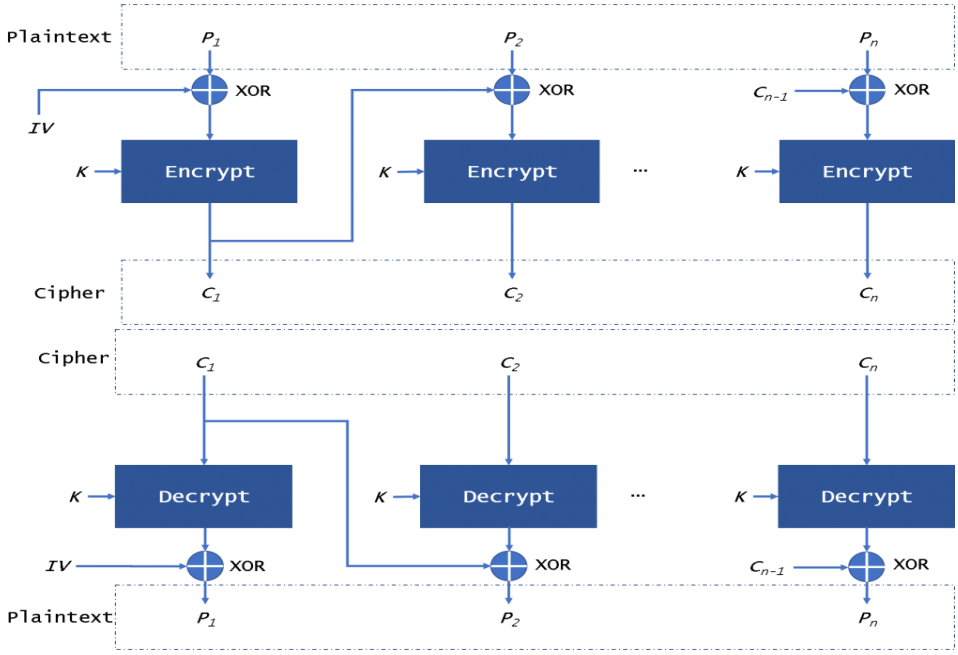
Luigi Dell'Eva, 27/03/2024.

Background

The challenge is accessible through a shell uisng the command `nc cyberchallenge.disi.unitn.it 50301` and implements a Proof of Concept program for AESWT, a faster successor to JWT (JSON Web Tokens), for website authentication. The user can perform two action: **create token** where the user is asked to provide a username and a description, then a token is printed and **login** where the user is asked to provide a token (generated within the same session) and the server will respond with the username.

Advanced Encryption Standard, also known as Rijndael algorithm, is a symmetric encryption algorithm that uses a block cipher to encrypt and decrypt data. The algorithm uses a key of a certain length (128, 192, or 256 bits) to encrypt and decrypt data in blocks of 128 bits (16 Bytes). [1] In case the data is not a multiple of 16 Bytes, the data is padded with to reach the block size.

AES algorithm provides five modes of operation which were standardized: ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback), and CTR (Counter). Since the challenge utilizes CBC mode, i will focus on this mode. In **CBC mode**, the plaintext is divided into blocks of 128 bits and each block is XORed with the previous ciphertext block before being encrypted. The first block is XORed with an initialization vector (IV) before being encrypted. The decriptyon is the reverse process. [2]

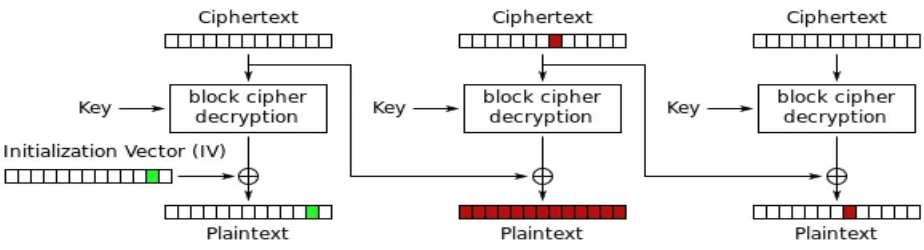


CBC is vulnerable to **Padding oracle** and **Bit flipping** attacks. Padding oracle attacks exploit the fact that the server responds with a different message depending on the padding of the decrypted message. Bit flipping attacks exploit the fact that the XOR operation is commutative and associative, so if an attacker can control the plaintext, he can control the ciphertext. [3,4]

The above modes do not provide integrity and authenticity of the data, so it is recommended to use a **Message authenticaton** technique such as HMAC (Hash-based Message Authentication Code) to ensure it (for example CBC-MAC). [5,6]

Vulnerability

Bit flipping is possible due to how CBC chaining works, modifications to the ciphertext block can lead to predictable changes in the decrypted plaintext at the corresponding location. An attacker can exploit this to alter specific bytes in the message. [4]



Cipher Block Chaining (CBC) mode decryption

Solution

To forge the tampered key I decided to use the Bit Flipping vulnerability which consists in finding a character that replaced within the provided key will generate the character i need to login into the application.

I know that `A (previous cipher block) XOR B (current cipher) = C (plaintext)` which can be rewritten as `A XOR decrypt(B) = C`. Now, I need to find `decrypt(B) = C XOR A`. Once I get `decrypt(B)`, I can utilize it to find a character that replaced into A will transform the corresponding character in C into Y. So, I need to find a character (?) that XORed with `decrypt(B)` will give me Y (`? XOR decrypt(B) = Y -> ? = Y XOR decrypt(B)`).

Providing an input with user `admin` and description `I am a boss` was not possible since the presence of the `admin` string would have triggered the check. The first thought was to replace a character of `admin` (i.e. `a` with `b`). However, those input would have generated a two-block ciphertext, and the bit flippping would have corrupted the first block, making the decryption fail. Additionally special characters such as `&` and `=` were not allowed.

Therefore, I needed to find an input that would help me to bit flip (knowing that some block will be corrupted) and at the same time obtain a string that when decrypted and given input to the check works. I found that description as `I am a boss` and `bdminXdescaaaaaaaaaaaaaaaaXdescXI am a boss` fit the problem.

Now I needed to change the char at position:

- block 2 pos 6 -> b to a
- block 2 pos 11 -> X to &
- block 3 pos 0 -> X to &
- block 3 pos 5 -> X to =

```
from Crypto.Cipher import AES
from pwn import *

def bitFlip(token, pos1, pos2, src_char, dest_char):
    dec_B = hex(int("0x"+token[pos1:pos2], 16) ^ int(src_char, 16))
    replacement_hex = hex(int(dest_char, 16) ^ int(dec_B, 16))[2:]
    return token[:pos1] + replacement_hex + token[pos2:]

conn = remote('cyberchallenge.disi.unitn.it', 50301)
#conn = process(['python3', 'challenge.py'])

conn.sendlineafter(b'> ', b'1')
conn.sendlineafter(b'> ', b'bdminXdescaaaaaaaaaaaaaaaaXdescXI am a boss')
conn.sendlineafter(b'> ', b'I am a boss')

conn.recvuntil(b':')
token = conn.recvline().strip().decode()
print(token, type(token))

# Transform b to a
token = bitFlip(token, 44, 46, '0x62', '0x61')
# Transform X to &
token = bitFlip(token, 54, 56, '0x58', '0x26')
# Transform X to &
token = bitFlip(token, 96, 98, '0x58', '0x26')
# Transform X to =
token = bitFlip(token, 106, 108, '0x58', '0x3d')

conn.sendlineafter(b'> ', b'2')
conn.sendlineafter(b'> ', token.encode())

recvText = conn.recvall(timeout=1)
print(recvText.strip().decode())
```


[1] NIST Advanced Encryption Standard (AES): https://www.nist.gov/publications/advanced-encryption-standard-aes

[2] AES mode of operation: https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/

[3] Padding oracle attack: https://en.wikipedia.org/wiki/Padding_oracle_attack

[4] Bit flipping attack: https://zhangzeyu2001.medium.com/attacking-cbc-mode-bit-flipping-7e0a1c185511

[5] Block cipher mode of operation: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[6] CBC-MAC: https://en.wikipedia.org/wiki/CBC-MAC

[7] Bit flipping logic: https://youtu.be/QG-z0r9afIs?si=nTeR41RW-XyJguuh