Padding Oracle Attack by Md. Mehedi Faruk

Introduction

Symmetric encryption like AES in Cipher Block Chaining [CBC] mode is used to secure data. If it's used without proper authentication, it's vulnerable to attacks like 'Padding Oracle Attack'. This report describes a padding oracle attack vulnerability in a quote distribution website **[cbc.syssec.dk]** that uses AES-CBC to encrypt authentication tokens.Website only provide a quote if a valid token is provided in a cookie. Here, the token is an encryption of a message containing a secret part. My task was to recover the secret part of a message from a given token and create a new valid ciphertext for a desired plaintext without accessing the key.
Helpful error messages make it possible due to leak information about padding during decryption. This report will show details about the attack along with a retrieved quote from the given site.

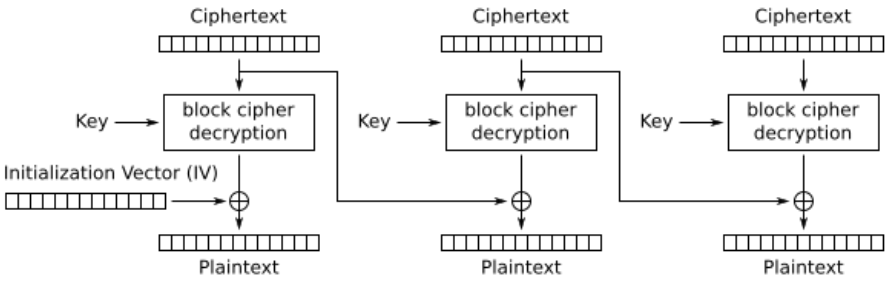
Encryption of AES-CBC:

Plaintext in 16-byte blocks.

$C_0 = IV$

$C_i = Enc(P_i \oplus C_{i-1})$

Decryption of AES-CBC:



Cipher Block Chaining (CBC) mode decryption

Image Source : Wikipidia

Attack Process:

Getting the token:
I need to get the token 'authtoken' which is not but a cookie. The cookie is encrypted and has two parts: an IV + an encrypted message.

```
def requesting_for_token(self, address):
        res = requests.get(address)
        return res.cookies.get_dict().get("authtoken")
```

Processing the token:
splits the token into two parts. The first 16 characters are the IV, and everything after that is the encrypted message. I converted these from text to actual bytes, and everything is properly aligned in 16-byte blocks now.

```
def iv_ct_tobyte(self, token, blocksize):
        iv = token[:blocksize]
        ciphertext = token[blocksize:]
        assert len(iv) == blocksize and len(ciphertext) % blocksize == 0
        iv_bytes = bytes.fromhex(iv)
        ciphertext_bytes = bytes.fromhex(ciphertext)
        return (iv_bytes, ciphertext_bytes)
```

Server Responses:
When I send modified tokens back to the server, it can give possible responses: Like "Padding is incorrect." "PKCS#7 padding is incorrect." Or it accepts the token without any error. These responses are very important because they tell me if I'm guessing correctly.

Block by Block attack:
For every block, I need to start with the last byte and try all possible values 0-255 to see the server response. Using server response to find the correct value. Send a modified token to the server, and the server acts like an oracle by showing if padding is valid or not.

1. Original: IV + Encrypted_Block → Decrypted → Valid/Invalid Padding
2. We control: IV (can modify all bytes)
3. Goal: Find value that gives valid padding (0x01)
4. Process:
   - Try IV values until padding is valid
   - When found: Decrypted_Byte ⊕ IV_Value = 0x01
   - Therefore: Decrypted_Byte = IV_Value ⊕ 0x01
   - Original_Byte = Decrypted_Byte ⊕ Original_IV

```
def single_block(self, block, address):
        print("\nDecrypting block: ", end="")
        iv_zero = [0] * self.BLOCK_SIZE
        for valid_pad in range(1, self.BLOCK_SIZE + 1):
            print("", end="", flush=True)
            iv_pad = []
            for b in iv_zero:
                iv_pad.append(valid_pad ^ b)
            for element_cand in range(256):
                iv_pad[-valid_pad] = element_cand
                iv = bytes(iv_pad)
                if self.oracle_for_attack(iv, block, address):
                    if valid_pad == 1:
                        iv_pad[-2] ^= 1
                        iv = bytes(iv_pad)
                        if not self.oracle_for_attack(iv, block, address):
                            continue
                    break
            else:
                raise Exception("No valid padding byte found")
            iv_zero[-valid_pad] = element_cand ^ valid_pad
        return bytes(iv_zero)


def xor_block(iv, dec):#Xor Operation
    reply_from_server = b""
    for i in range(len(iv)):
        reply_from_server += bytes([iv[i] ^ dec[i]])
```

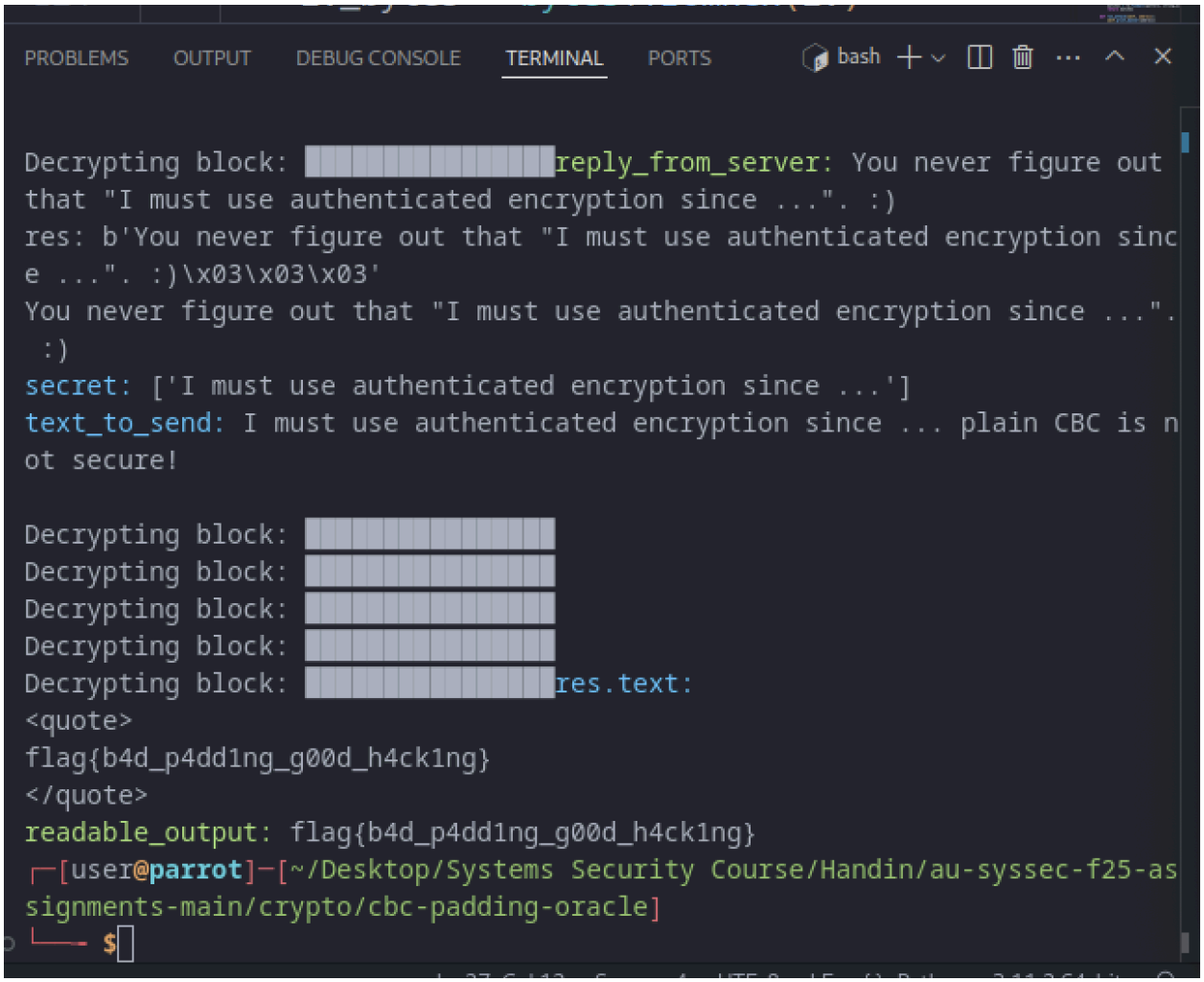
Forge New Token:
A new plaintext message is created by inserting "plain CBC is not secure!" to the secret value. Creating a new token containing IV and ciphertext for the padded plaintext using the padding oracle.The forged token is created by concatenating the IV and ciphertext.

```
def oracle_for_attack(iv, block, address):
    # Combine modified IV and block
    new_ciphertext = iv + block

    # Send to server and get response
    res = sending_short_token(new_ciphertext, address)

    # Check server response
    return (res.text != "Padding is incorrect.") and (
        res.text != "PKCS#7 padding is incorrect."
    )
```

Submit Forged Token:
The forged token is submitted to the /quote/ url. The server's response, containing the quote, is printed as the final output.

```
def sending_short_token(token, address):
    return requests.get(f"{address}/quote/",
                        cookies={"authtoken": token.hex()})
```

Quote retrieved from the server:



Why does this attack work?

CBC mode XORs each block with previous ciphertext along with padding validation and leaked information, allowing us to modify the IV to test every byte. Server reveals padding validity, and we can build plaintext byte by byte using those leaks.

References : Firebird Internal CTF 2023 Writeup
Padding oracle attack - Wikipedia