Padding Oracle Attack

Mohammad Abu-Shelbaia
1200198

***Abstract*— The "Padding Oracle Attack Lab" is a hands-on practical exercise that exploits a vulnerability in a specific encryption algorithm. This lab focuses on demonstrating the weakness of the padding scheme employed by the encryption algorithm, which can lead to a successful attack known as the padding oracle attack.**

I. Introduction

Padding Oracle Attack is an attack that uses the padding validation of an encrypted message to decrypt the ciphertext. Varying message lengths get padded to be compatible with the block cipher encryption algorithm, the attack depends on the Oracle server which responds to queries about whether a message is padded correctly or not. The attack is mostly associated with CBC mode decryption used with block ciphers. [1]

A. Cipher Block Chaining (CBC)

block cipher mode of operation that provides information security such as confidentiality or authenticity. It is an algorithm that uses a block cipher to securely transform amounts of data larger than a block. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. [2]

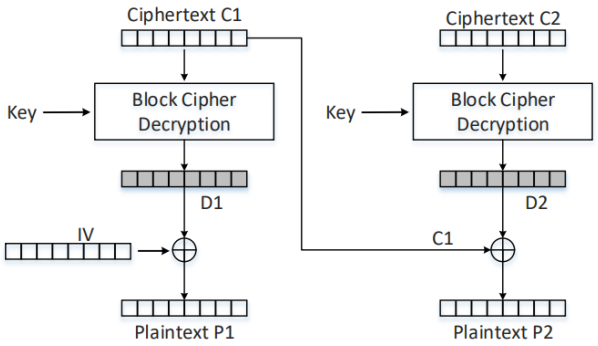

Figure 1: CBC Decryption Diagram

B. PKCS5 Padding

If the size of each block is denoted as B, the process involves adding N padding bytes to the input to reach the next multiple of B, taking into account the following conditions. When the input length is not an exact multiple of B, N padding bytes, each having the value of N, are appended. However, if the input length is already a multiple of B, B bytes of value B are added. This ensures that padding, ranging from one to B bytes in length, is consistently applied without any ambiguity. Once the decryption is complete, a verification step is performed by checking that the last N bytes of the decrypted data all have the value N, where N is greater than 1 but not exceeding B. If this condition is met, N bytes are removed; otherwise, a decryption error is thrown. [3]

II. Procedure

A. Getting Familiar With Padding

This task is a demonstration of how PKCS5 Padding works, for this task we will encrypt a message of variable lengths using AES-128, and decrypt them with '-nopad' flag to observe the appended padding. We can run the `Makefile` using `make run` command.

```
run:
	@echo "[5 Bytes] 12345"
	@echo -n "12345" | openssl enc -aes-128-cbc -e -k hi -md sha256 -pbkdf2 | openssl enc -aes-128-cbc -d -nopad -k hi -md sha256 -pbkdf2 | xxd

	@echo "\n[10 Bytes] 0123456789"
	@echo -n "0123456789" | openssl enc -aes-128-cbc -e -k hi -md sha256 -pbkdf2 | openssl enc -aes-128-cbc -d -nopad -k hi -md sha256 -pbkdf2 | xxd

	@echo "\n[16 Bytes] 0123456789ABCDEF"
	@echo -n "0123456789ABCDEF" | openssl enc -aes-128-cbc -e -k hi -md sha256 -pbkdf2 | openssl enc -aes-128-cbc -d -nopad -k hi -md sha256 -pbkdf2 | xxd
```

Snippet 1: Padding Task

```
mabushelbaia@LegionY540:~
~/oracle-padding-attack main +6 -11 [$!] via v3.10.6
make run
[5 Bytes] 12345
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........

[10 Bytes] 0123456789
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......

[16 Bytes] 0123456789ABCDEF
00000000: 3031 3233 3435 3637 3839 4142 4344 4546  0123456789ABCDEF
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................

~/oracle-padding-attack main +6 -11 [$!] via v3.10.6
date && whoami
Wed Jul  5 02:54:57 IDT 2023
mabushelbaia
```

Figure 2: Padding Output

As we can see in Figure 2, when a message has a size of 5 bytes, 11 bytes of padding are appended, each with a value of '0x0b' (equivalent to 11 in decimal). For a 10-byte message, 6 bytes of padding are appended, each with a value of '0x06'.

Lastly, for a message with a size of 16 bytes, a full block of padding is added, with each byte having a value of '0x10' (equivalent to 16 in decimal), as 16 bytes is a multiple of the block size.

B. Padding Oracle Attack (Level 1)

In this task, we are going to decrypt the last few Bytes of the plain text manually. To get the last byte of the second block we have to change the last byte of the first cipher-text, since

$$P_{32} = C_{16} \oplus D_{32} \tag{1}$$

we will loop through all the values of C_{16} until we land at a value at which we get a MAC error instated of a padding error. Knowing the value that gives us the correct padding we can decrypt the plain text according to the following equation:

$$\hat{P_{32}} = \hat{C_{16}} \oplus D_{32} \tag{2}$$

where $\hat{P_{32}}$ is the value at which we get a correct padding, and since we know that

$$D_{32} = C_{16} \oplus P_{32} \tag{3}$$

we can show that

$$P_{32} = \hat{P_{32}} \oplus C_{16} \oplus \hat{C_{16}} \tag{4}$$

```python
oracle = PaddingOracle('10.9.0.80', 5000)
IV, C1, C2 = oracle.ctext[:16], oracle.ctext
  ↪  [16:32], oracle.ctext[32:48]
C1_copy, IV_copy = bytearray(16), bytearray(16)
P = bytearray(32)
for i in range(2**8):
    C1_copy[15] = i
    status = oracle.decrypt(IV + C1_copy + C2)
    if status == "Valid":
        break
P[31] = 1 ^ C1_copy[15] ^ C1[15]
print(f"P[31]: {hex(P[31])}")
print(f"C1'[15]: {hex(C1_copy(15)}")
# Output:
#    P[31]: 0x3
#    C1'[15]: 0xcf
```

Snippet 2: Decrypting the last byte of the plain text

To get the second to last byte we need to adjust the padding of the first 2-bytes, looping through all possible values of the deemed impossible with the complexity arising exponentially, so we will use another way which is to manipulate the last byte so we can repeat previous steps. First, we need to manipulate the output of the last byte to 0x02, so the server would go and check the second-byte padding we can do so by manipulating the last byte of the first block as shown below:

$$\hat{\hat{C_{16}}} = P_{32} \oplus C_8 \oplus \hat{\hat{P_{32}}} \tag{5}$$

where $\hat{\hat{P_{32}}}$ is the value we want present in P_{32}.

```python

oracle = PaddingOracle('10.9.0.80', 5000)
IV, C1, C2 = oracle.ctext[:16], oracle.ctext
  ↪  [16:32], oracle.ctext[32:48]
C1_copy, IV_copy = bytearray(16), bytearray(16)
P = bytearray(32)
P[31] = 0x03
C1_copy[15] = C1[15] ^ P[31] ^ 0x02
for i in range(2**8):
    C1_copy[14] = i
    status = oracle.decrypt(IV + C1_copy + C2)
    if status == "Valid":
        break
P[30] = 0x02 ^ C1_copy[14] ^ C1[14]
print(f"P[30]: {hex(P[30])}")
print(f"C1'[14]: {hex(C1_copy[14])}")
# Output:
#    P[30]: 0x3
#    C1'[14]: 0x39
```

Snippet 3: Decrypting the second to last byte of the plain text

Figure 3: Decrypting the last two bytes from the plain text

In order to expedite and automate the process of decrypting the values, manual byte-by-byte manipulation of the ciphertext is indeed impractical and time-consuming, especially for large messages with numerous blocks. Therefore, an automated approach becomes necessary to handle the decryption of larger ciphertexts. In the next section, we will explore a more efficient method to automate the process, enabling us to decrypt the plaintext values effectively and save considerable time and effort.

C. Padding Oracle Attack (Level 2)

In this part, we wrote a small script to send queries to the server automatically according to the previous responses, we ran the program on both servers `10.9.0.80:5000` and `10.9.0.80:6000`, and then compared the result we got with the provided plain text.

```python
oracle = PaddingOracle('10.9.0.80', 5000)
IV, C1, C2 = oracle.ctext[:16], oracle.ctext
  ↪  [16:32], oracle.ctext[32:48]
P = bytearray(32)
C1_copy, IV_copy = bytearray(16), bytearray(16)
for k in range(1, 17):
    for i in range(256):
        C1_copy[16 - k] = i
```

```
        status = oracle.decrypt(IV+ C1_copy + C2)
        if status == "Valid":
            break

    P[32 - k] = C1_copy[16 - k] ^ C1[16 - k] ^ k
    for j in range(1, k+1):
        C1_copy[16 - j] = C1[16 - j] ^ (k + 1) ^
        ↪  P[32 - j]

    for i in range(256):
        IV_copy[16 - k] = i
        status = oracle.decrypt(IV_copy + C1)
        if status == "Valid":
            break

    P[16 - k] = IV_copy[16 - k] ^ IV[16 - k] ^ k
    for j in range(1, k+1):
        IV_copy[16 - j] = IV[16 - j] ^ (k + 1) ^
        ↪  P[16 - j]
```

Snippet 4: Decrypting the plain text automatically

A small notice to decrypt the first block, since we don't have any previous cipher block we change the values of the IV instated.

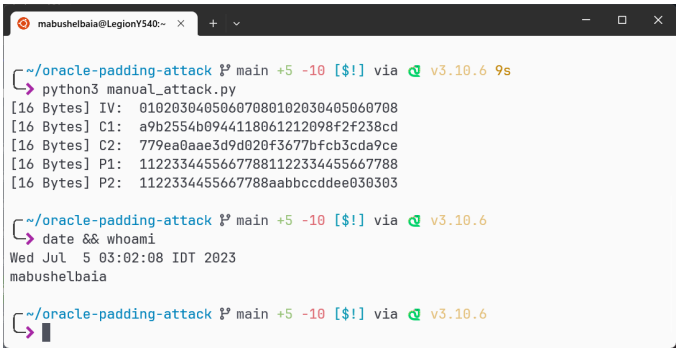

Figure 4: Plain Text retrieved from the server on port 5000

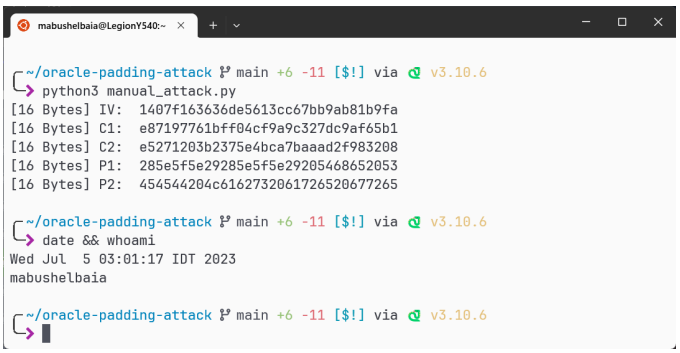

Figure 5: Plain Text retrieved from the server on port 6000

Figure 6: Server Status

Comparing the results in (Figure: 4) and the provided plain text, we conclude that our script is running correctly, and by referring to (Figure: 6) we can see that we retrieved the plain text with one connection to the server.

III. Conclusion

In this lab, we explored the Padding Oracle Attack, which exploits vulnerabilities in the padding scheme of an encryption algorithm. By leveraging the padding validation responses from the Oracle server, we were able to decrypt the ciphertext. Through manual decryption of the last few bytes of the plaintext and an automated approach using a script, we successfully recovered the original message. This exercise highlights the importance of secure padding schemes and the potential risks associated with inadequate implementations. By understanding the weaknesses exposed by the Padding Oracle Attack, we can strengthen encryption protocols to enhance overall data security.

References

[1] "Padding oracle attack," https://en.wikipedia.org/wiki/Padding$_oracle_attack$ [Online; accessed July 5, 2023].

[2] "Block cipher mode of operation — Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Block$_cipher_mode_of_operation$ [Online; accessed 5-July-2023].

[3] "Padding schemes in public key infrastructure (pki)," https://www.cryptosys.net/pki/manpki/pki$_paddingschemes.html$ [Online; accessed July 5, 2023].