

אבטחת תקשורת - מטלה 2

מגישים: רועי חשאי ויתיר גרוס

2.....	הקדמה
2.....	שלב 1
2.....	שלב 2
3.....	שלב 3
3.....	שלב 4
4.....	שלב 5
5.....	שלב 6
6.....	שלב 7
7.....	שלב 8
7.....	שלב 9
8.....	שלב 10
9.....	שלב 11
10.....	שלב 12
12.....	תוכנית POA

הקדמה

בדוח זה נסביר תחילה כל חלק וכל שלב מתוך המטלה. לבסוף נראה את הקוד השלם שמבצע את ההתקפה וניתן מס דוגמאות הרצאה שימחישו את נכונות הסקריפט ואת היכולת לשבור הצפנות ע"י padding POA (oracle attack)

שלב 1

```
# ===== STEP 1 =====  
from Cryptodome.Cipher import DES  
from Cryptodome.Util.Padding import pad,unpad
```

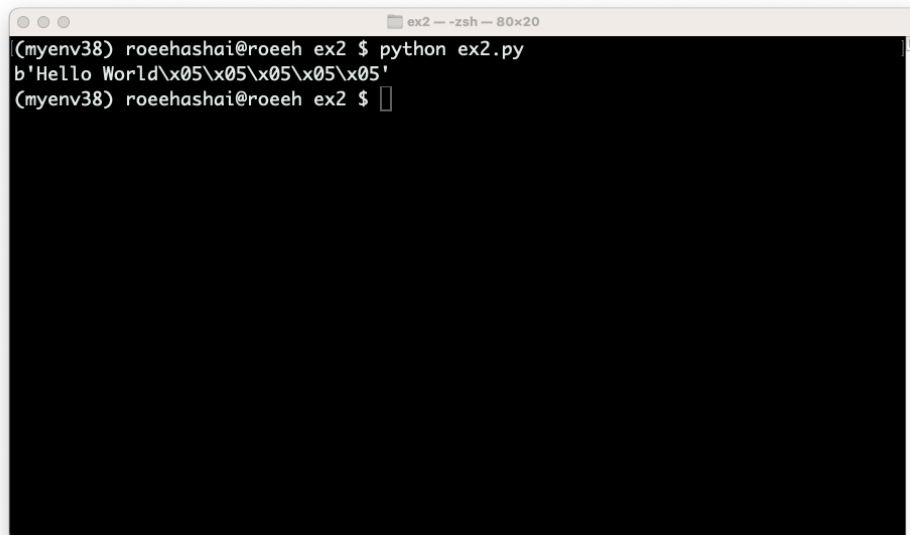
תחילה נייבא את הספריות המתאימות. נשים לב שאלו ספריות לא בטוחות כי הם יאפשרו לנו ממש לעשות את padding ולשלוט על דברים שבספריה שראינו בכיתה לא יכלנו לשלוט עליהם אך זה כמובן לשם התרגיל. ע"י DES אנחנו נוכל להצפין במוד CBC וע"י pad,unpad נוכל לעשות את padding שראינו בכיתה שהוא בשיטת PKCS7

שלב 2

כעת נרפד את המחרוזת Hello World ונדפיס את מה שיצא. לשם כך נשתמש בפונק' pad

```
# ===== STEP 2 =====  
plain_text = b"Hello World"  
padded_text = pad(plain_text, DES.block_size)  
print(padded_text)
```

ונקבל



```
ex2 -- zsh -- 80x20  
(myenv38) roeehashai@roeeh ex2 $ python ex2.py  
b'Hello World\\x05\\x05\\x05\\x05\\x05'  
(myenv38) roeehashai@roeeh ex2 $
```

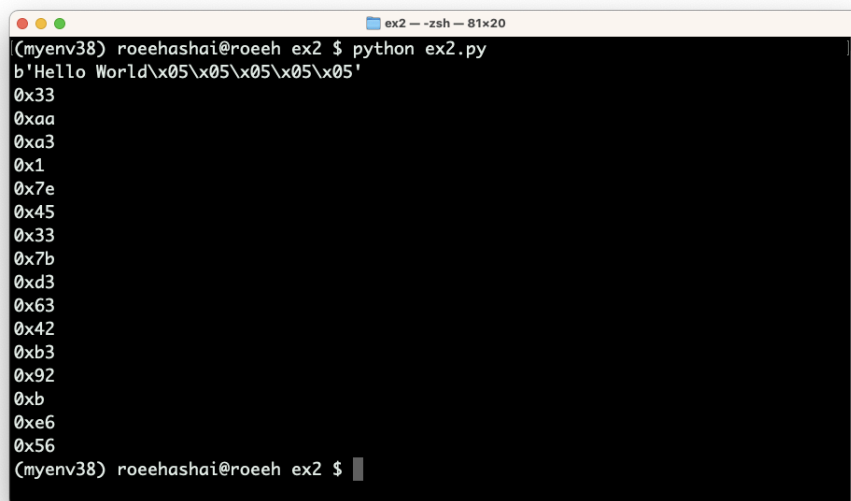
נזכיר כי בDES mode CBC גודל מפתח הוא 8 בתים וגודל כל בלוק הוא גם 8 בתים. ולכן המחרוזת Hello World אשר באורך 11 בתים תצטרך לקבל ריפוד של 5 בתים. כיוון שאנחנו בשיטת PKCS7 אנחנו נקבל ריפוד כזה ש5 הבתים האחרונים ירופדו במס הבתים שריפדנו (5 בתים) כלומר 0x05 למשך 5 בתים.

שלב 3

כעת אנחנו נצפין את ההודעה המרופדת משלב 2 בהצפנת DES mode CBC לשם כך נגדיר מפתח כדרוש נגדיר iv שכולו אפסים בגודל שמונה בתים כגודל בלוק נצפין ונדפיס

```
# ===== STEP 3 =====
key = b"poaisfun"
iv = b"\x00\x00\x00\x00\x00\x00\x00\x00"
cipher = DES.new(key, DES.MODE_CBC, iv)
ciphertext = cipher.encrypt(padded_text)
for byte in ciphertext:
    print(hex(byte))
```

נריץ ונקבל



```
ex2 -- zsh -- 81x20
(myenv38) roeehashai@roeeh ex2 $ python ex2.py
b'Hello World\x05\x05\x05\x05'
0x33
0xaa
0xa3
0x1
0x7e
0x45
0x33
0x7b
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
(myenv38) roeehashai@roeeh ex2 $
```

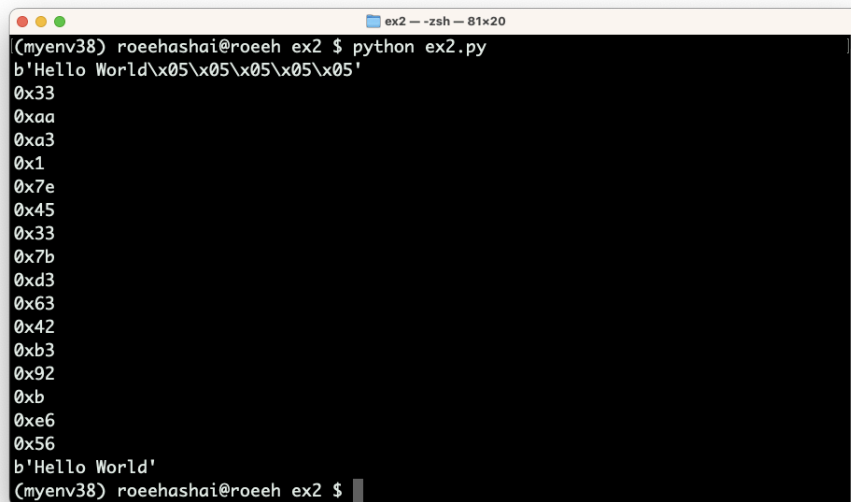
ניתן לראות את ההצפנה של הטקסט בייט אחר בייט כמו שמתואר בתרגיל וזהה לפלט הצפוי.

שלב 4

כעת נראה שניתן לחזור אחורה מהסייפר טקסט ולקבל את הטקסט המקורי בלי הריפוד. לשם כך נכתוב את השורות הבאות

```
# ===== STEP 4 =====
cipher = DES.new(key, DES.MODE_CBC, iv)
plain_text_padded = cipher.decrypt(ciphertext)
plain_text = unpad(plain_text_padded, DES.block_size)
print(plain_text)
```

קודם נקבל את הטקסט אבל מרופד עם 0x05 שכתוב ב5 הבתים האחרונים, לאחר מכן נוריד את הריפוד ע"י קריאה לפונק unpad אשר במקרה של ריפוד תקין מורידה את הריפוד. כיוון שריפדנו ע"י הפונק pad אז וודאי שנקבל ריפוד תקין ולכן בקריאה לפונק unpad נקבל את הטקסט המקורי ללא כל שינוי. ולכן נריץ ונקבל



```
ex2 -- zsh -- 81x20
(myenv38) roeehashai@roeeh ex2 $ python ex2.py
b'Hello World\x05\x05\x05\x05\x05'
0x33
0xaa
0xa3
0x1
0x7e
0x45
0x33
0x7b
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
b'Hello World'
(myenv38) roeehashai@roeeh ex2 $
```

ניתן לראות כי הטקסט המקורי פוענח בהצלחה.

שלב 5

נבנה פונק Xor אשר בהמשך תשמש אותנו לחישוב הבתים במתקפה. פונק אשר מקבלת שלושה מספרים ומחזירה את תוצאות Xor בין שלושתם.

```
# ===== STEP 5 =====
def xor(a, b, c):
    """
    input: a,b,c (int)
    output: a XOR b XOR c (bytes)
    """
    return bytes([a ^ b ^ c])

print(xor(0,0,0))
print(xor(0,0,1))
print(xor(0,1,0))
print(xor(0,1,1))
print(xor(1,0,0))
print(xor(1,0,1))
print(xor(1,1,0))
print(xor(1,1,1))
```

נשים לב שיש פה עניין של לקבל טיפוס int ולהחזיר byte לפי הדרישה של התרגיל ולכן נחזיר מערך בטיפ עם בייט בודד שהוא יהיה התשובה כדי לקבל תשובה בבתים ולא כints. עניין של טיפוסים והמרות של פייטון. נריץ ונקבל

```
ex2 -- zsh -- 81x20
0x33
0x7b
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
b'Hello World'
b'\x00'
b'\x01'
b'\x01'
b'\x00'
b'\x01'
b'\x00'
b'\x00'
b'\x01'
(myenv38) roeehashai@roeeh ex2 $
```

שלב 6

נכתוב כעת את הפונק אשר תאפשר לנו המשך לבצע את ההתקפה. הפונק oracle היא פונק אשר מחזירה אמת או שקר לאם יש ריפוד חוקי לטקסט. כלומר היא מקבל סייפרטקסט מוצפן מבצעת את הפענוח. לה כמובן יש את המפתח ומחזירה אמת או שקר רק אם הפענוח יצר מחרוזת אשר יש לה ריפוד חוקי. מה זה ריפוד חוקי? בשיטת PKCS7 ראינו שמרפדים בכמות בתים שמתחלקת ב8 וכמות הבתים שריפדנו זה מה שכותבים בבתים שרופדו. ולכן Hello World\x05\x05\x05\x05\x05\x05 ויש 5 בתים שכתוב בהם הערך 5. לעומת Hello World!\x05\x05\x05\x05\x05\x05 רואים שלכאורה יש 5 בתים של ריפוד אבל בפועל יש רק 4 בתים שיש בהם את הערך 0x05. וזה לא חוקי. נראה את הפונק

```
# ===== STEP 6 =====
def oracle(ciphertext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    try:
        unpad(decrypted_text, DES.block_size)
        return True
    except ValueError:
        return False

print(oracle(ciphertext, key, iv)) # True
bad_ciphertext = DES.new(key, DES.MODE_CBC, iv).encrypt(b"0123456789abcde\x05")
print(oracle(bad_ciphertext, key, iv)) # False
```

הפונק שכתבתנו מפענחת את הטקסט, מנסה לבצע את unpadding ולהוריד את padn ותצליח רק במקרה שהוא חוקי. כלומר בדיוק ההתנהגות שאנחנו רוצים. וכאשר נריץ על הטקסט החוקי מלפני נקבל אמת לעומת הטקסט 0123456789abcde\x05 אשר מצפה לריפוד של 5 בתים ומקבל רק בייט אחד של ריפוד נכשל. נריץ

```
ex2 -- -zsh -- 81x20
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
b'Hello World'
b'\x00'
b'\x01'
b'\x01'
b'\x00'
b'\x01'
b'\x00'
b'\x00'
b'\x01'
True
False
(myenv38) roeeshai@roeesh ex2 $
```

שלב 7

בשלב זה ניצור משתנה בשם c. משתנה זה הוא למעשה השרשור של הבלוק שעליו עושים את brouten force והבלוק C2 מהסייפר שאותו נשבור. כלומר לפני הסימונים שראינו בכיתה c=Xj||C2

```
# ===== STEP 7 =====
c = b'\x00' * 8 + ciphertext[8:]
for b in c:
    print(hex(b))
```

כאשר נריץ נקבל

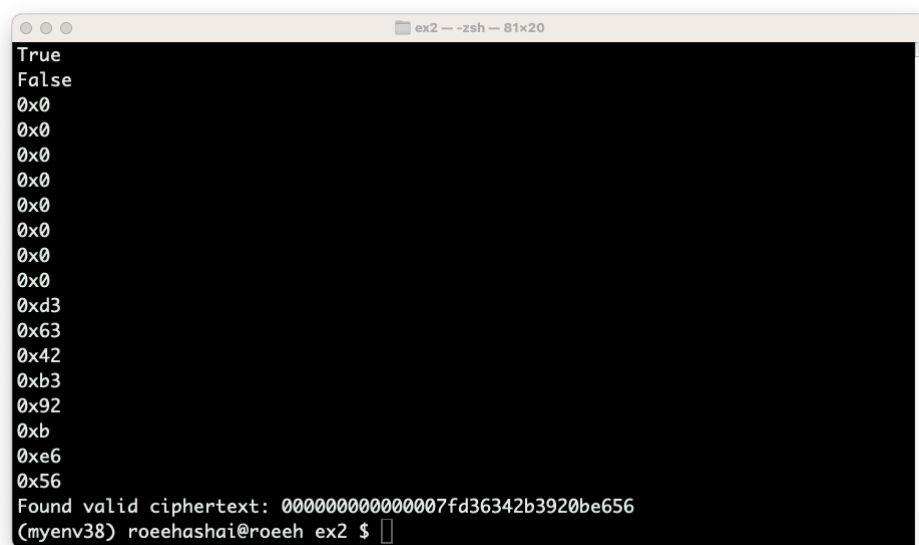
```
ex2 -- -zsh -- 81x20
b'\x01'
True
False
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
(myenv38) roeeshai@roeesh ex2 $
```

שלב 8

כעת אנחנו נשלח את c ללואה שבכל פעם תגדיל את הערך ההתחלתי של הבייט האחרון ב1. זה יהיה הבסיס לחלק של brute force במתקפת padding oracle שנבצע בהמשך.

```
# ===== STEP 8 =====  
c = bytearray(c)  
for i in range(256):  
    c[7] = i  
    if oracle(c, key, iv):  
        print("Found valid ciphertext:", c.hex())  
        break
```

ראשית נמיר את c מלפני לטיפוס שהוא מערך בתים mutable כדי שנוכל לעשות על הבייט השמיני brout force. ואז נבדוק את כל האפשרויות מ0 עד 255 עבור הערך של הבייט האחרון. במצב הזה כדי לקבל מהאורקל אמת נצטרך שלאחר הפענוח ערך הבייט האחרון יהיה 0x01 כדי שהטקסט יחשב טקסט עם ריפוד חוקי. ולכן כמו שראינו בכיתה כאשר נבדוק את כל האפשרויות לערך של בייט נקבל בטוח שתוצאות xor של אחד מהם (הxor המגיע מהפענוח של DES) יניב טקסט שהבייט האחרון שלו הוא בעל הערך 0x01 כלומר ריפוד חוקי ויחזיר עליו אמת. אנחנו רק צריכים להבין מיהו הבייט של Xj אשר נותן את הריפוד החוקי הזה. כעת נריץ ונקבל את הc שגורם לאורקל להחזיר אמת כי לאחר הפענוח שלו הבייט הימני ביותר הוא 0x01.



```
True  
False  
0x0  
0x0  
0x0  
0x0  
0x0  
0x0  
0x0  
0x0  
0xd3  
0x63  
0x42  
0xb3  
0x92  
0xb  
0xe6  
0x56  
Found valid ciphertext: 0000000000000007fd36342b3920be656  
(myenv38) roeeshai@roeesh ex2 $
```

רואים שכאשר הבייט האחרון של Xj הוא 0x7f אנחנו נקבל אמת מהאורקל כלומר נקבל ריפוד חוקי שהבייט האחרון של הפענוח הוא עם הערך 0x01 (ולכן האורקל החזיר אמת).

שלב 9

לאחר שאנו יודעים מהו הערך של הבייט החיפשונו נוכל לגלות את הערך של הבייט השמיני בטקסט! וזה כמובן מבלי לפענח את הטקסט עם המפתח הסודי. כל שעלינו לעשות הוא להשתמש בנוסחה שראינו בכיתה שכרגע היא משוואה בנעלם אחד

$$P_i[x] = P'_2[x] \oplus C_{i-1}[x] \oplus X_j[x]$$

את הערך של P'_2 אנחנו יודעים כי מזה שהאורקל החזיר אמת זה כנראה כי הביית האחרון הוא $0x01$. את C_{i-1} אנחנו גם מכירים כי יש לנו את הסייפרטקסט המקורי ואת X_j אנחנו יודעים כי עשינו עליו את brute force ולכן כל מה שנותר הוא לגלות את P_i שזה הביית השמיני בטקסט המקורי!

```
# ===== STEP 9 =====
# P'_2[x] ^ C_{i-1}[x] ^ X_j[x]
P_2_7 = xor(1, ciphertext[7], c[7])
print(P_2_7)
```

ונקבל

```
False
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
Found valid ciphertext: 000000000000007fd36342b3920be656
b'\x05'
(myenv38) roeeshai@roeesh ex2 $
```

שלב 10

כעת כדי להמשיך את המתקפה עלינו להבין מהו צריך להיות הערך של הבלוק האחרון ב X_j על מנת שלאחר הפענוח שלו יהיה שם $0x02$ כדי שבאפזה הבאה שנעשה brute force על הביית השני מימין כלומר ביית 7 אנחנו נקבל אמת מהאורקל וזה יקרה כאשר שני הבתים האחרונים מימין ערכם הוא $0x02$ וכך נוכל להמשיך לחשוף את שאר הבתים של הטקסט. נשתמש באותה משוואה שהצגנו לעיל רק עם העברת אגפים קטנה כדי שנוכל לחלץ את X_j . ולכן נכתוב

```
# ===== STEP 10 =====
# X_j[x] = P'_2[x] ^ C_{i-1}[x] ^ P_2[x]
X_j_7 = xor(2, ciphertext[7], P_2_7[0])[0]
c[7] = X_j_7
print(c.hex())
```

כאשר נריץ את זה נקבל את הערך c_7 שצריך להיות שם. כלומר אם נשים ב X_j את c_7 אז לאחר הפענוח נקבל ערך הביית האחרון הוא $0x02$ ואז כאשר נעשה brute force על הביית השני מימין נקבל אמת כאשר ערכו הוא לא $0x01$ אלא $0x02$.


```
ex2 -- -zsh -- 81x20
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0xd3
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
Found valid ciphertext: 00000000000007fd36342b3920be656
b'\x05'
00000000000007cd36342b3920be656
(myenv38) roeeshai@roeesh ex2 $
```

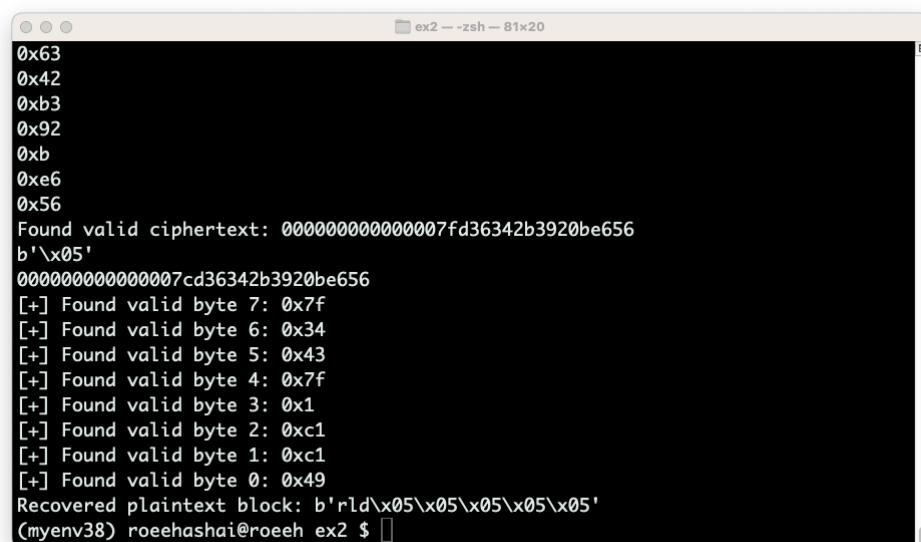
שלב 11

נהפוך את שלבים 8-10 ללואה על מנת שנוכל לחשוף את כל הבלוק. נחשוף בייט באופן שראינו לעיל וככה נחשוף את כל הבלוק. נשים לב שאנחנו צריכים לרוץ בלולאה של כמות הבתים כלומר 8. לחשוף את הטקסט באינדקס המתאים ואז להבין מה ערך הביט שצריך לשים בXj כדי שבסיבוב הבא נוכל לחשוף את הביט הבא. נשים לב שגם זה צריך לקרות בתוך לולאה כי באיטרציה הראשונה חושפים את הביט הראשון(מימין) ואז צריך לקבע בלוק אחד מימין לערך 0x02. לאחר מכן בפאזה הבאה לאחר מציאת הביט השני מימין בטקסט נצטרך לקבע את ה2 בתים הימניים בXj בערך 0x03. ואז 3 בתים ימניים ביותר בערך 0x04 וכך הלאה עד לחשיפת הבלוק. כלומר יש לולאה פנימית שמבצעת את התיקונים האלה. נתבונן על הקוד

```
# ===== STEP 11 =====
C1 = ciphertext[:8]
C2 = ciphertext[8:]
XJ = bytearray(8)
XJ_C2 = XJ + bytearray(C2)
plaintext = bytearray(8)
for byte_idx in reversed(range(8)):
    padding_val = 8 - byte_idx
    # Step 1: Brute-force byte at byte_idx in XJ
    for byte_val in range(256):
        XJ_C2[byte_idx] = byte_val
        if oracle(bytes(XJ_C2), key, iv):
            print(f"[+] Found valid byte {byte_idx}: {hex(byte_val)}")
            break
    # Step 2: Recover plaintext byte at byte_idx in P2
    plaintext[byte_idx] = xor(padding_val, C1[byte_idx], XJ_C2[byte_idx])[0]
```

```
# Step 3: Update all known bytes to match new padding value
next_pad = padding_val + 1
for i in range(8 - byte_idx):
    pos = 7 - i
    XJ_C2[pos] = xor(next_pad, C1[pos], plaintext[pos])[0]
print("Recovered plaintext block:", bytes(plaintext))
```

תחילה נגדיר את C1, C2 הבלוקים של הסייפר. אנחנו נחשוף את הבלוק C2. נגדיר את השרשור של הבלוק עם הXj ההתחלתי שכולו אפסים ונגדיר מערך בתים מאופס של הטקסט שנגלה במהלך הלולאות. כעת נבצע לולאה לכל בייט בבלוק, ישנם 8 כי מדובר בDES mode CBC. נחשב את padding הצפוי לאיטרציה הנ"ל. כלומר מהו המס' שיהיה ב'P' שיחושב ע"י האורקל בבית המתאים כאשר האורקל יחזיר אמת. בהתחלה 0x01 ואז 0x02 וכך הלאה. לאחר מכן נראה את brute force כמו שראינו בשלב 8. ואז מהxor והמשוואה שראינו השלה 9 נוכל לגלות את הבייט המתאים בטקסט. נשמור אותו. ולבסוף נתקן את הבתים הדרושים בXj כדי שנוכל להמשיך לשבור את המשך הסייפר משמאל. ואת זה נעשה בלולאה האחרונה מה שיש מתחת להערה step 3. כאשר נריץ נקבל



```
0x63
0x42
0xb3
0x92
0xb
0xe6
0x56
Found valid ciphertext: 000000000000007fd36342b3920be656
b'\x05'
000000000000007cd36342b3920be656
[+] Found valid byte 7: 0x7f
[+] Found valid byte 6: 0x34
[+] Found valid byte 5: 0x43
[+] Found valid byte 4: 0x7f
[+] Found valid byte 3: 0x1
[+] Found valid byte 2: 0xc1
[+] Found valid byte 1: 0xc1
[+] Found valid byte 0: 0x49
Recovered plaintext block: b'rld\x05\x05\x05\x05\x05'
(myenv38) roeeshai@roeesh ex2 $
```

ניתן בהדפסות לראות את הבתים שחזרו אמת מהאורקל לכל פאזה שבאמצעותם גילינו את הבתים המתאימים בטקסט. לאחר מכן ניתן לראות כי הצלחנו לשחזר את הבלוק השני בplaintext! מבלי כמובן להשתמש בפענוח ובמפתח הסודי.

שלב 12

בשלב זה נרחיב ונעבור מלחשוף בלוק אחד לחשיפת כל הסייפר. נעשה זה בשיטה של חשיפת בלוק בלוק. ישנם שני שינויים שיש לבצע לשם כך. ראשית, לכל בלוק בסייפר נעשה את הלוגיקה שהראנו לעיל. שנית, נצטרך להבדיל בין בלוק ראשון לסייפר לבין השאר כי הבלוק הראשון בסייפר קודמו הוא iv בשונה משאר הבלוקים שקודמם הם בלוקים קודמים בסייפר. נראה את הקוד

```
# ===== STEP 12 =====
blocks_cnt = len(ciphertext) // DES.block_size
```

```

plaintext_blocks = []
for i in range(blocks_cnt):
    block_start = i * DES.block_size
    block_end = block_start + DES.block_size
    block_ciphertext = ciphertext[block_start:block_end]
    if i == 0:
        C_prev = iv
    else:
        C_prev = ciphertext[block_start - DES.block_size:block_start]
    XJ = bytearray(8)
    XJ_C2 = XJ + bytearray(block_ciphertext)
    plaintext_block = bytearray(8)
    for byte_idx in reversed(range(8)):
        padding_val = 8 - byte_idx
        # Step 1: Brute-force byte at byte_idx in XJ
        for byte_val in range(256):
            XJ_C2[byte_idx] = byte_val
            if oracle(bytes(XJ_C2), key, iv):
                print(f"[+] Found valid byte {byte_idx} in block {i}: {hex(byte_val)}")
                break
        # Step 2: Recover plaintext byte at byte_idx in P2
        plaintext_block[byte_idx] = xor(padding_val, C_prev[byte_idx],
XJ_C2[byte_idx])[0]
        # Step 3: Update all known bytes to match new padding value
        next_pad = padding_val + 1
        for j in range(8 - byte_idx):
            pos = 7 - j
            XJ_C2[pos] = xor(next_pad, C_prev[pos], plaintext_block[pos])[0]
    plaintext_blocks.append(bytes(plaintext_block))
print("Full plaintext:", b''.join(plaintext_blocks))

```

ראשית נחשב את מס הבלוקים שיש לפענח. לכל בלוק נבצע את הלוגיקה מלעיל. קודם נחשב את ההיסט של הבלוק הנוכחי. כדי להרכיב מתוך הסייפר טקסט את הבלוק סייפר שאותו שוברים לאיטרציה הנוכחית. מיד לאחר מכן נבדיל בין הבלוק הקודם לבין אם מדובר בבלוק בראשון או בכל אחד אחר. ואז נבצע ממש את הלולאות שראינו בשלב 11. אם נריץ את זה על הטקסט ההתחלתי נקבל

```
ex2 - zsh - 81x20
[+] Found valid byte 0: 0x49
Recovered plaintext block: b'rld\x05\x05\x05\x05'
[+] Found valid byte 7 in block 0: 0x6e
[+] Found valid byte 6 in block 0: 0x55
[+] Found valid byte 5 in block 0: 0x23
[+] Found valid byte 4 in block 0: 0x6b
[+] Found valid byte 3 in block 0: 0x69
[+] Found valid byte 2 in block 0: 0x6a
[+] Found valid byte 1 in block 0: 0x62
[+] Found valid byte 0 in block 0: 0x40
[+] Found valid byte 7 in block 1: 0x7f
[+] Found valid byte 6 in block 1: 0x34
[+] Found valid byte 5 in block 1: 0x43
[+] Found valid byte 4 in block 1: 0x7f
[+] Found valid byte 3 in block 1: 0x1
[+] Found valid byte 2 in block 1: 0xc1
[+] Found valid byte 1 in block 1: 0xc1
[+] Found valid byte 0 in block 1: 0x49
Full plaintext: b'Hello World\x05\x05\x05\x05'
(myenv38) roeeshai@roeesh ex2 $
```

ואכן הצלחנו לחשוף ולשבור את הסייפרטקסט לפענח את הטקסט מבלי להתשמש במפתח ע"י התקפת padding oracle!

תוכנית POA

בשלב זה נהפוך את כל מה שראינו בשלבים הקודמים לתוכנית אשר מקבל מה args את הסייפר את המפתח ואת iv ומפעילה את על הלוגיקות והפונק שראינו בתרגיל כדי לשבור את הסייפר.

```
import sys
def padding_oracle_attack(ciphertext, key, iv):
    blocks_cnt = len(ciphertext) // DES.block_size
    plaintext_blocks = []
    for i in range(blocks_cnt):
        block_start = i * DES.block_size
        block_end = block_start + DES.block_size
        block_ciphertext = ciphertext[block_start:block_end]
        if i == 0:
            C_prev = iv
        else:
            C_prev = ciphertext[block_start - DES.block_size:block_start]
        XJ = bytearray(8)
        XJ_C2 = XJ + bytearray(block_ciphertext)
        plaintext_block = bytearray(8)
        for byte_idx in reversed(range(8)):
            padding_val = 8 - byte_idx
            # Step 1: Brute-force byte at byte_idx in XJ
            for byte_val in range(256):
                XJ_C2[byte_idx] = byte_val
```

```

        if oracle(bytes(XJ_C2), key, iv):
            print(f"[+] Found valid byte {byte_idx} in block {i}:
{hex(byte_val)}")
            break

        # Step 2: Recover plaintext byte at byte_idx in P2
        plaintext_block[byte_idx] = xor(padding_val, C_prev[byte_idx],
XJ_C2[byte_idx])[0]

        # Step 3: Update all known bytes to match new padding value
        next_pad = padding_val + 1
        for j in range(8 - byte_idx):
            pos = 7 - j
            XJ_C2[pos] = xor(next_pad, C_prev[pos], plaintext_block[pos])[0]
        plaintext_blocks.append(bytes(plaintext_block))
    return b''.join(plaintext_blocks)

def main():
    if len(sys.argv) != 4:
        print("Usage: python ex2.py <ciphertext> <key> <iv>")
        sys.exit(1)

    ciphertext = bytes.fromhex(sys.argv[1])
    key = bytes.fromhex(sys.argv[2])
    iv = bytes.fromhex(sys.argv[3])
    plaintext = padding_oracle_attack(ciphertext, key, iv)
    print("Recovered plaintext:", plaintext)
    print("Textual representation:", plaintext.decode('utf-8', errors='ignore'))

if __name__ == "__main__":
    main()

```

ואם נריץ כעת את הקוד עם הסייפר You can't break me, I'm a padding oracle! נקבל ריפוד של

```
b"You can't break me, I'm a padding oracle!\x07\x07\x07\x07\x07\x07\x07"
```

ההצפנה של הטקסט יהיה

```
ff201a1db9d4314f1f7a7c996c25e6070c41be20362db832e16ffd6c3a7853b20ac400fc03fef7c49b7778
bfc6c93a48
```

ואם נכניס את אותו המפתח ממקודם שהיה poaisfun שערכו ב hex הוא 706f61697366756e יחד עם iv שכולו אפסים. כלומר נריץ:

```
(myenv38) roeeshai@roeesh ex2 $ python ex2.py
ff201a1db9d4314f1f7a7c996c25e6070c41be20362db832e16ffd6c3a7853b20ac400fc03fef7c49b
7778bfc6c93a48 706f61697366756e 0000000000000000
```

נקבל את הפלט

```
ex2 - -zsh - 81x20
[+] Found valid byte 7 in block 4: 0xd6
[+] Found valid byte 6 in block 4: 0x3d
[+] Found valid byte 5 in block 4: 0x18
[+] Found valid byte 4 in block 4: 0x5f
[+] Found valid byte 3 in block 4: 0x1b
[+] Found valid byte 2 in block 4: 0x94
[+] Found valid byte 1 in block 4: 0x48
[+] Found valid byte 0 in block 4: 0x8e
[+] Found valid byte 7 in block 5: 0xc2
[+] Found valid byte 6 in block 5: 0xf2
[+] Found valid byte 5 in block 5: 0xfa
[+] Found valid byte 4 in block 5: 0x0
[+] Found valid byte 3 in block 5: 0xfe
[+] Found valid byte 2 in block 5: 0x1
[+] Found valid byte 1 in block 5: 0xc4
[+] Found valid byte 0 in block 5: 0x23
Recovered plaintext: b"You can't break me, I'm a padding oracle!\x07\x07\x07\x07\x07\x07\x07"
Textual representation: You can't break me, I'm a padding oracle!
(myenv38) roeeshai@roeesh ex2 $
```

כדרוש!