FINDING PASSWORD IN EXECUTABLE USING GDB

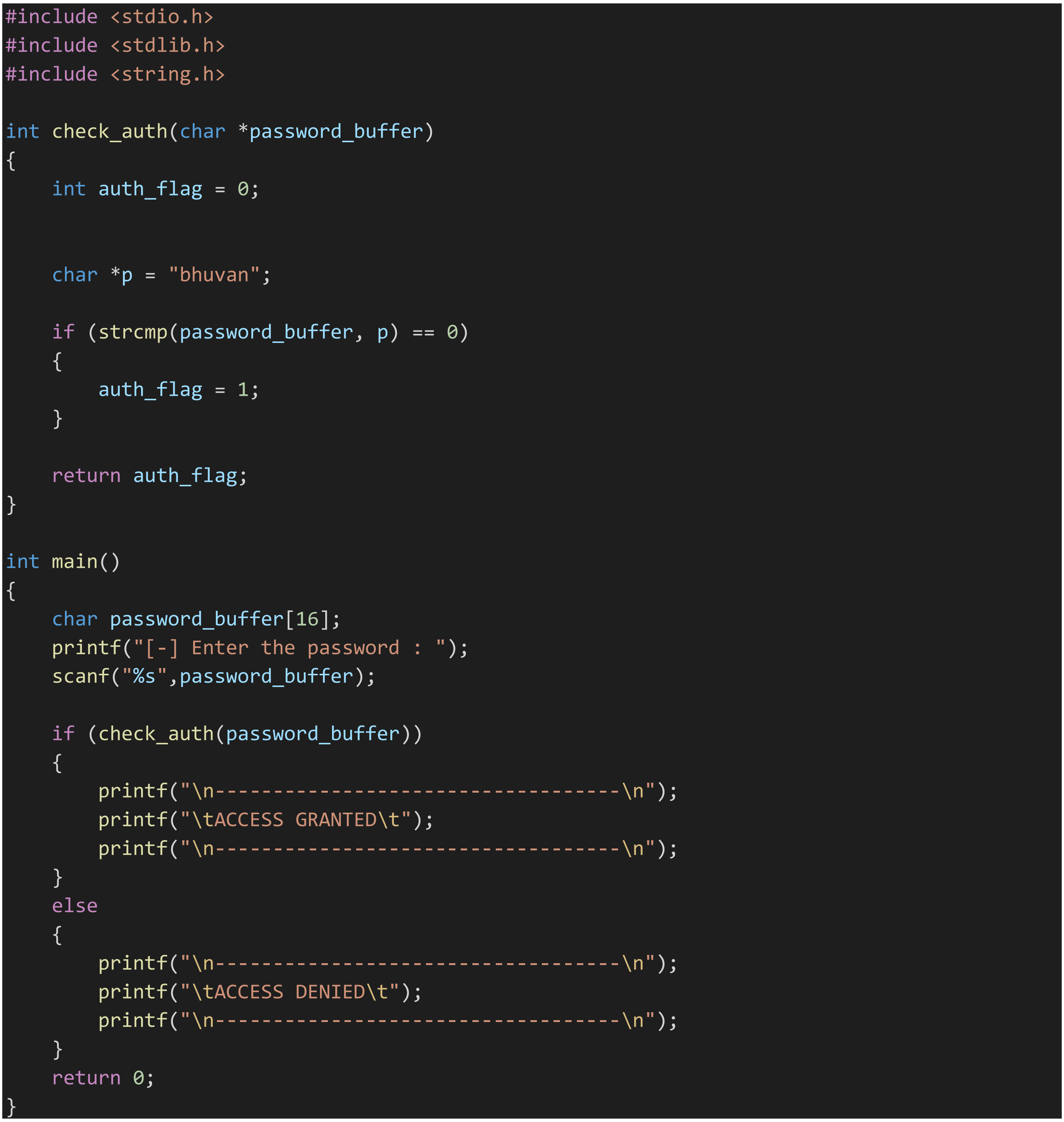
SUBJECT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

SUBJECT CODE: CS6008

MODULE: 1

NAME	BHUVANESHWAR S
REG.NO	2019103513
DATE	05/04/2022

Filename : auth.c



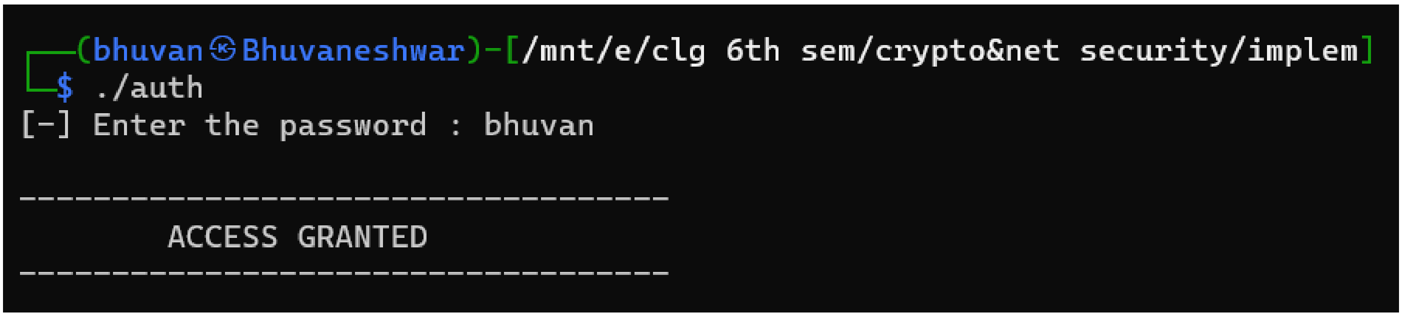
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_auth(char *password_buffer)
{
    int auth_flag = 0;


    char *p = "bhuvan";

    if (strcmp(password_buffer, p) == 0)
    {
        auth_flag = 1;
    }

    return auth_flag;
}

int main()
{
    char password_buffer[16];
    printf("[-] Enter the password : ");
    scanf("%s",password_buffer);

    if (check_auth(password_buffer))
    {
        printf("\n-----------------------------------\n");
        printf("\tACCESS GRANTED\t");
        printf("\n-----------------------------------\n");
    }
    else
    {
        printf("\n-----------------------------------\n");
        printf("\tACCESS DENIED\t");
        printf("\n-----------------------------------\n");
    }
    return 0;
}
```

The program accepts a password from a command line argument and then call function check_auth().This function compare the password whether it is equal it return 1 otherwise it return 0.

Compile this program using this command : gcc -g -o auth auth.c

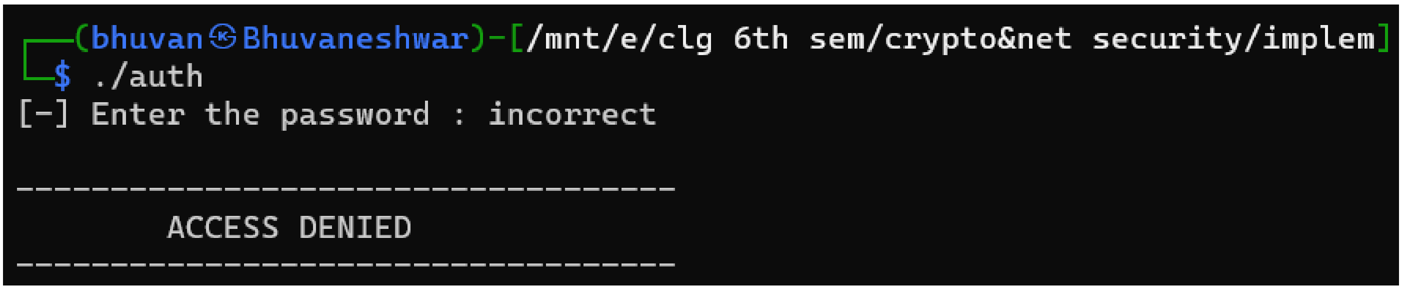
Then run this program : ./auth

When user type correct password



```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/implem]
└─$ ./auth
[-] Enter the password : bhuvan

-----------------------------------
        ACCESS GRANTED
-----------------------------------
```

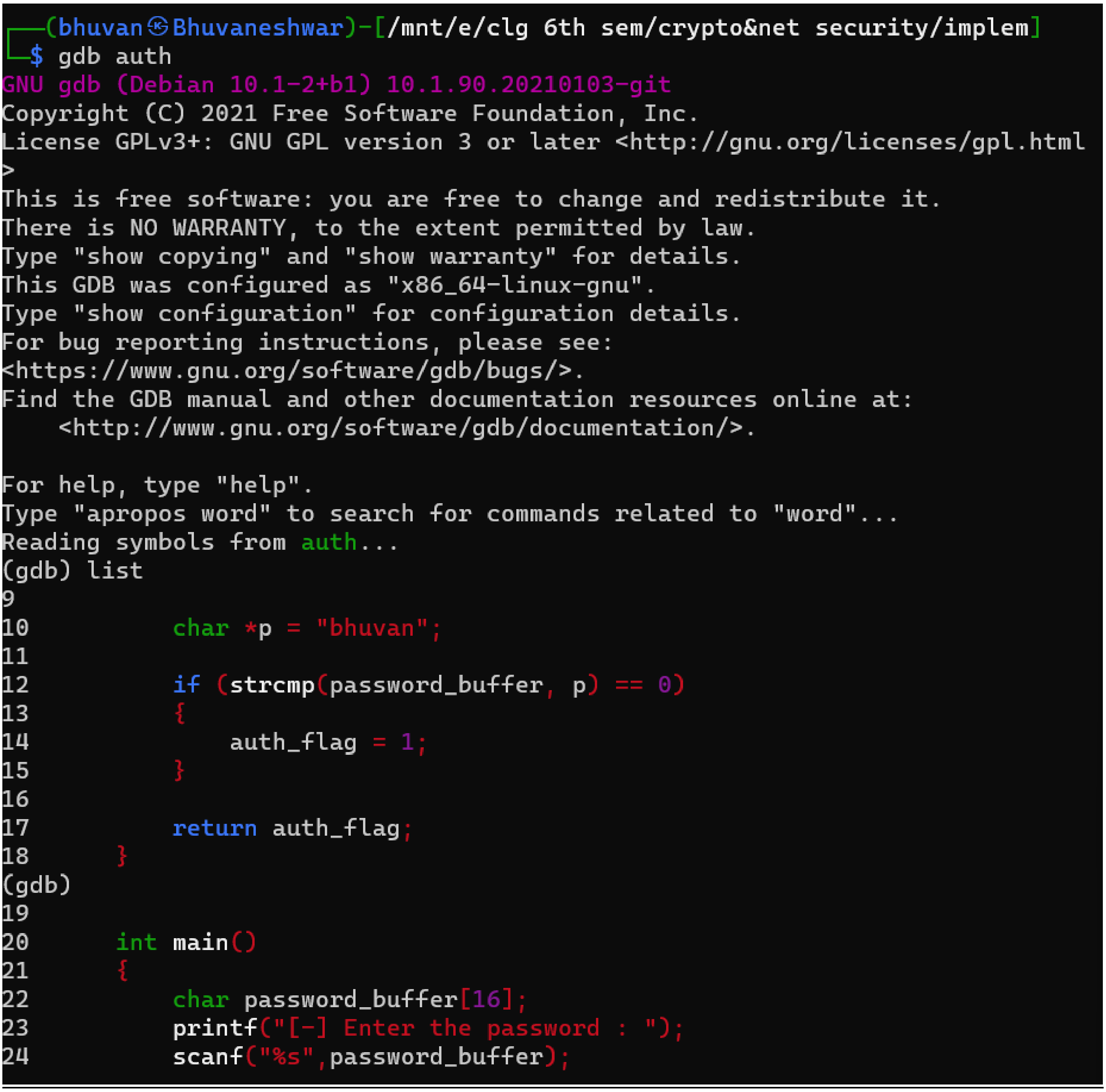
When user type incorrect password.



```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/implem]
└─$ ./auth
[-] Enter the password : incorrect

-----------------------------------
        ACCESS DENIED
-----------------------------------
```

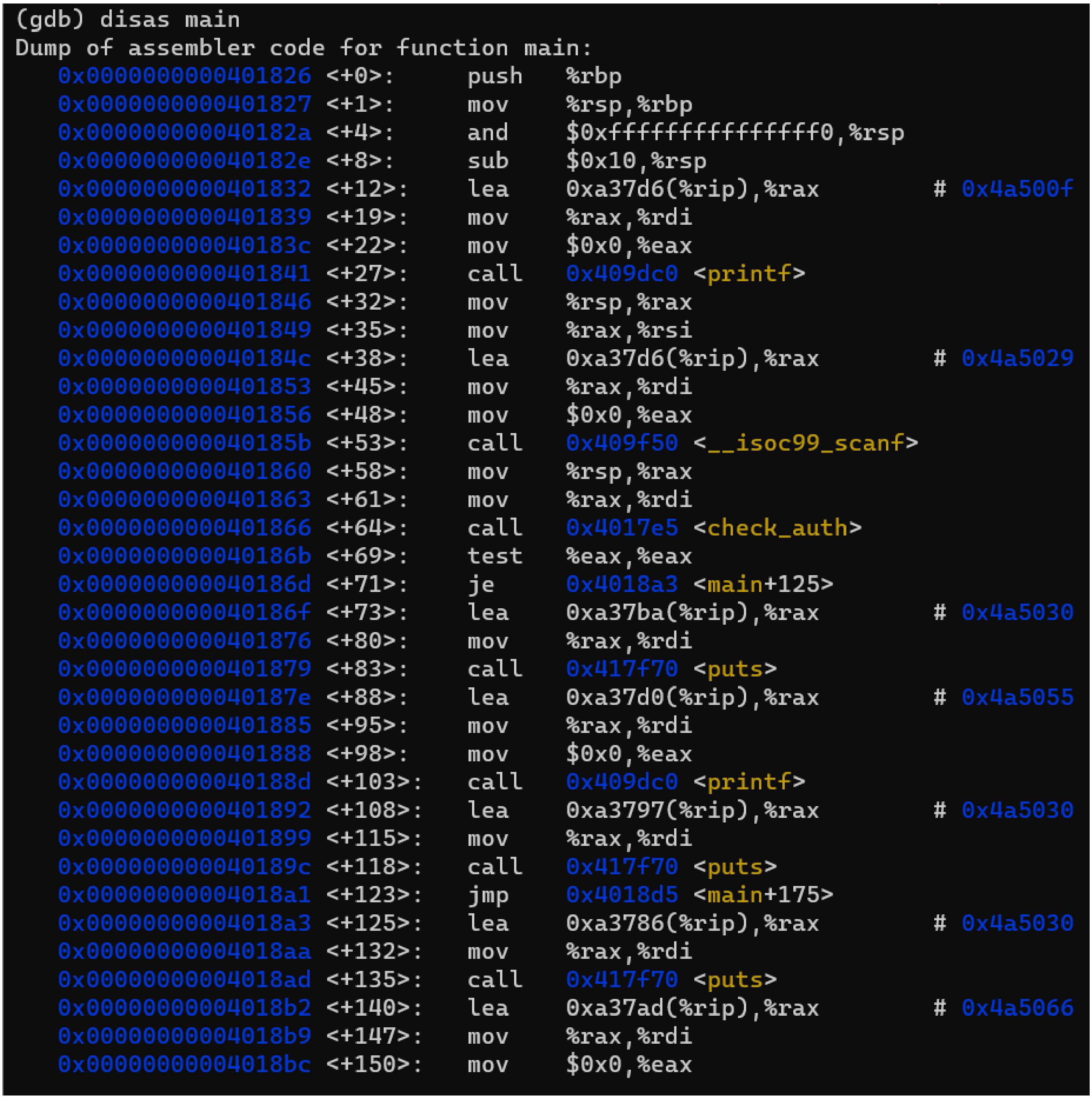
i) CRACKING USING GDB DEBUG

Let start with gdb debugging using this command : **gdb auth**



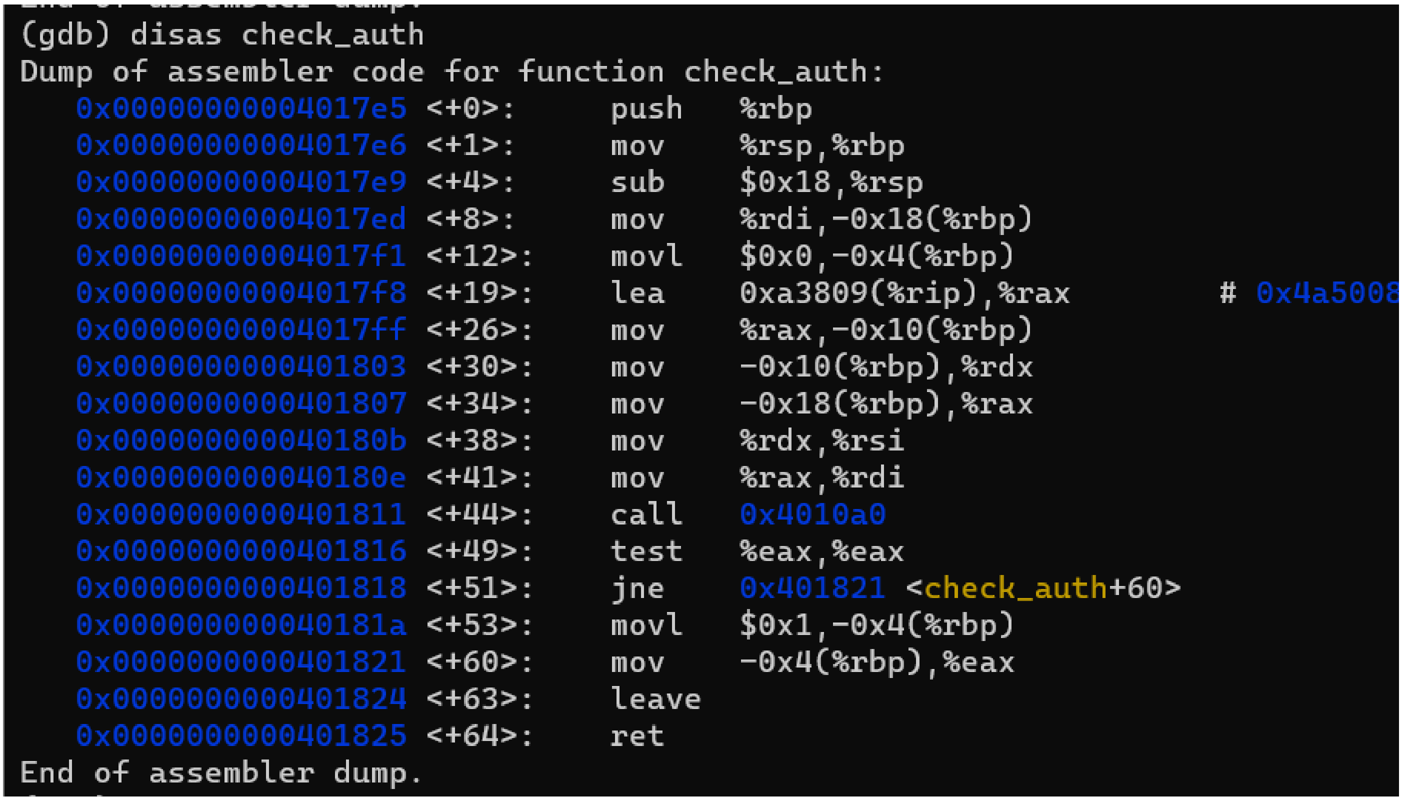
```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/implem]
└─$ gdb auth
GNU gdb (Debian 10.1-2+b1) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html
>
This is free software: you are free 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 auth...
(gdb) list
9
10              char *p = "bhuvan";
11
12              if (strcmp(password_buffer, p) == 0)
13              {
14                  auth_flag = 1;
15              }
16
17              return auth_flag;
18          }
(gdb)
19
20          int main()
21          {
22              char password_buffer[16];
23              printf("[-] Enter the password : ");
24              scanf("%s",password_buffer);
```

Disassemble main

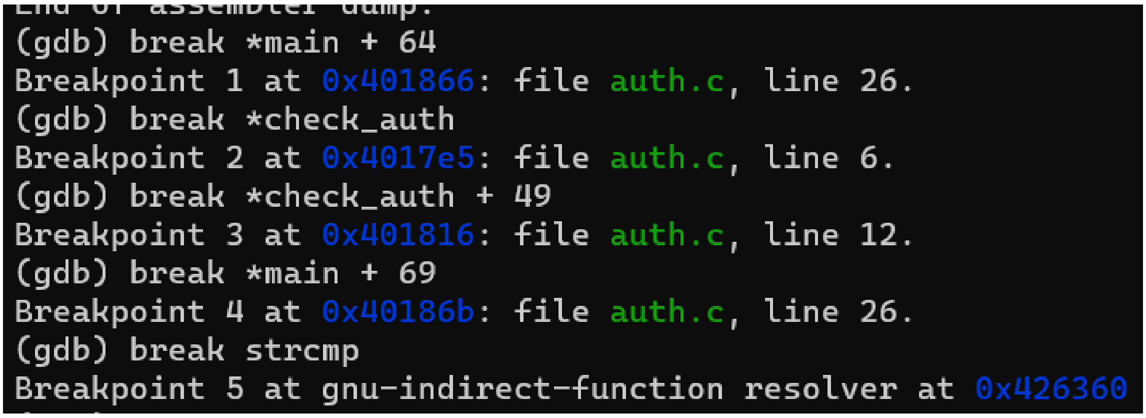


```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000401826 <+0>:     push   %rbp
   0x0000000000401827 <+1>:     mov    %rsp,%rbp
   0x000000000040182a <+4>:     and    $0xfffffffffffffff0,%rsp
   0x000000000040182e <+8>:     sub    $0x10,%rsp
   0x0000000000401832 <+12>:    lea    0xa37d6(%rip),%rax        # 0x4a500f
   0x0000000000401839 <+19>:    mov    %rax,%rdi
   0x000000000040183c <+22>:    mov    $0x0,%eax
   0x0000000000401841 <+27>:    call   0x409dc0 <printf>
   0x0000000000401846 <+32>:    mov    %rsp,%rax
   0x0000000000401849 <+35>:    mov    %rax,%rsi
   0x000000000040184c <+38>:    lea    0xa37d6(%rip),%rax        # 0x4a5029
   0x0000000000401853 <+45>:    mov    %rax,%rdi
   0x0000000000401856 <+48>:    mov    $0x0,%eax
   0x000000000040185b <+53>:    call   0x409f50 <__isoc99_scanf>
   0x0000000000401860 <+58>:    mov    %rsp,%rax
   0x0000000000401863 <+61>:    mov    %rax,%rdi
   0x0000000000401866 <+64>:    call   0x4017e5 <check_auth>
   0x000000000040186b <+69>:    test   %eax,%eax
   0x000000000040186d <+71>:    je     0x4018a3 <main+125>
   0x000000000040186f <+73>:    lea    0xa37ba(%rip),%rax        # 0x4a5030
   0x0000000000401876 <+80>:    mov    %rax,%rdi
   0x0000000000401879 <+83>:    call   0x417f70 <puts>
   0x000000000040187e <+88>:    lea    0xa37d0(%rip),%rax        # 0x4a5055
   0x0000000000401885 <+95>:    mov    %rax,%rdi
   0x0000000000401888 <+98>:    mov    $0x0,%eax
   0x000000000040188d <+103>:   call   0x409dc0 <printf>
   0x0000000000401892 <+108>:   lea    0xa3797(%rip),%rax        # 0x4a5030
   0x0000000000401899 <+115>:   mov    %rax,%rdi
   0x000000000040189c <+118>:   call   0x417f70 <puts>
   0x00000000004018a1 <+123>:   jmp    0x4018d5 <main+175>
   0x00000000004018a3 <+125>:   lea    0xa3786(%rip),%rax        # 0x4a5030
   0x00000000004018aa <+132>:   mov    %rax,%rdi
   0x00000000004018ad <+135>:   call   0x417f70 <puts>
   0x00000000004018b2 <+140>:   lea    0xa37ad(%rip),%rax        # 0x4a5066
   0x00000000004018b9 <+147>:   mov    %rax,%rdi
   0x00000000004018bc <+150>:   mov    $0x0,%eax
```

Disassemble check_auth



```
(gdb) disas check_auth
Dump of assembler code for function check_auth:
   0x00000000004017e5 <+0>:     push   %rbp
   0x00000000004017e6 <+1>:     mov    %rsp,%rbp
   0x00000000004017e9 <+4>:     sub    $0x18,%rsp
   0x00000000004017ed <+8>:     mov    %rdi,-0x18(%rbp)
   0x00000000004017f1 <+12>:    movl   $0x0,-0x4(%rbp)
   0x00000000004017f8 <+19>:    lea    0xa3809(%rip),%rax        # 0x4a5008
   0x00000000004017ff <+26>:    mov    %rax,-0x10(%rbp)
   0x0000000000401803 <+30>:    mov    -0x10(%rbp),%rdx
   0x0000000000401807 <+34>:    mov    -0x18(%rbp),%rax
   0x000000000040180b <+38>:    mov    %rdx,%rsi
   0x000000000040180e <+41>:    mov    %rax,%rdi
   0x0000000000401811 <+44>:    call   0x4010a0
   0x0000000000401816 <+49>:    test   %eax,%eax
   0x0000000000401818 <+51>:    jne    0x401821 <check_auth+60>
   0x000000000040181a <+53>:    movl   $0x1,-0x4(%rbp)
   0x0000000000401821 <+60>:    mov    -0x4(%rbp),%eax
   0x0000000000401824 <+63>:    leave
   0x0000000000401825 <+64>:    ret
End of assembler dump.
```



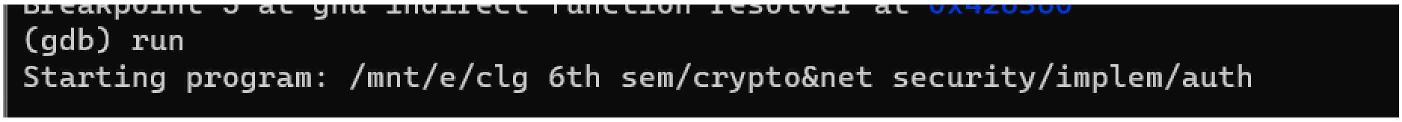
```
(gdb) break *main + 64
Breakpoint 1 at 0x401866: file auth.c, line 26.
(gdb) break *check_auth
Breakpoint 2 at 0x4017e5: file auth.c, line 6.
(gdb) break *check_auth + 49
Breakpoint 3 at 0x401816: file auth.c, line 12.
(gdb) break *main + 69
Breakpoint 4 at 0x40186b: file auth.c, line 26.
(gdb) break strcmp
Breakpoint 5 at gnu-indirect-function resolver at 0x426360
```

Break points are set **as *main+64,*check_auth, check_auth + 49, *main + 69, strcmp**.

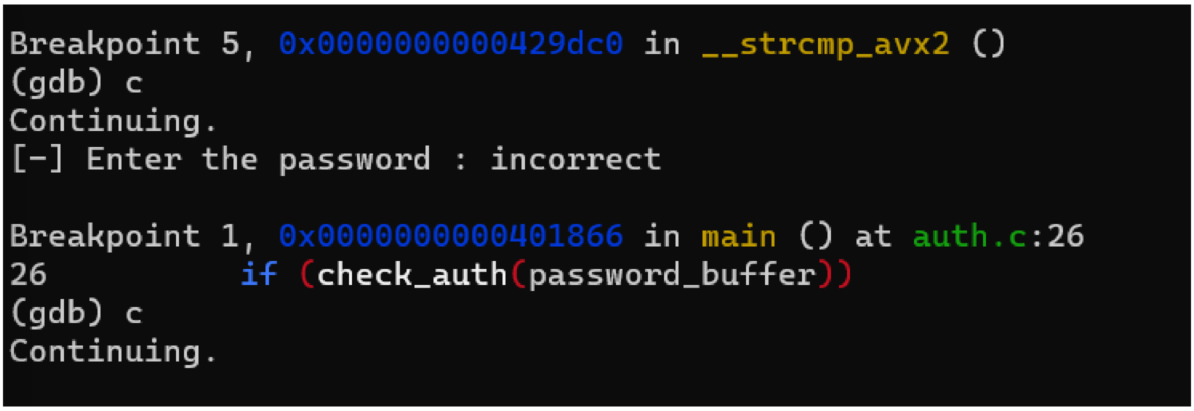
Where

- ***main + 64** => call functions check_auth() to compare password
- ***main + 69** => return an integer from check_auth() function to check whether it is correct or not.
- ***check_auth + 49** => retrun an integer value from strcmp() function to compare the both string
- **Strcmp** => call strcmp() function through “string.h” library

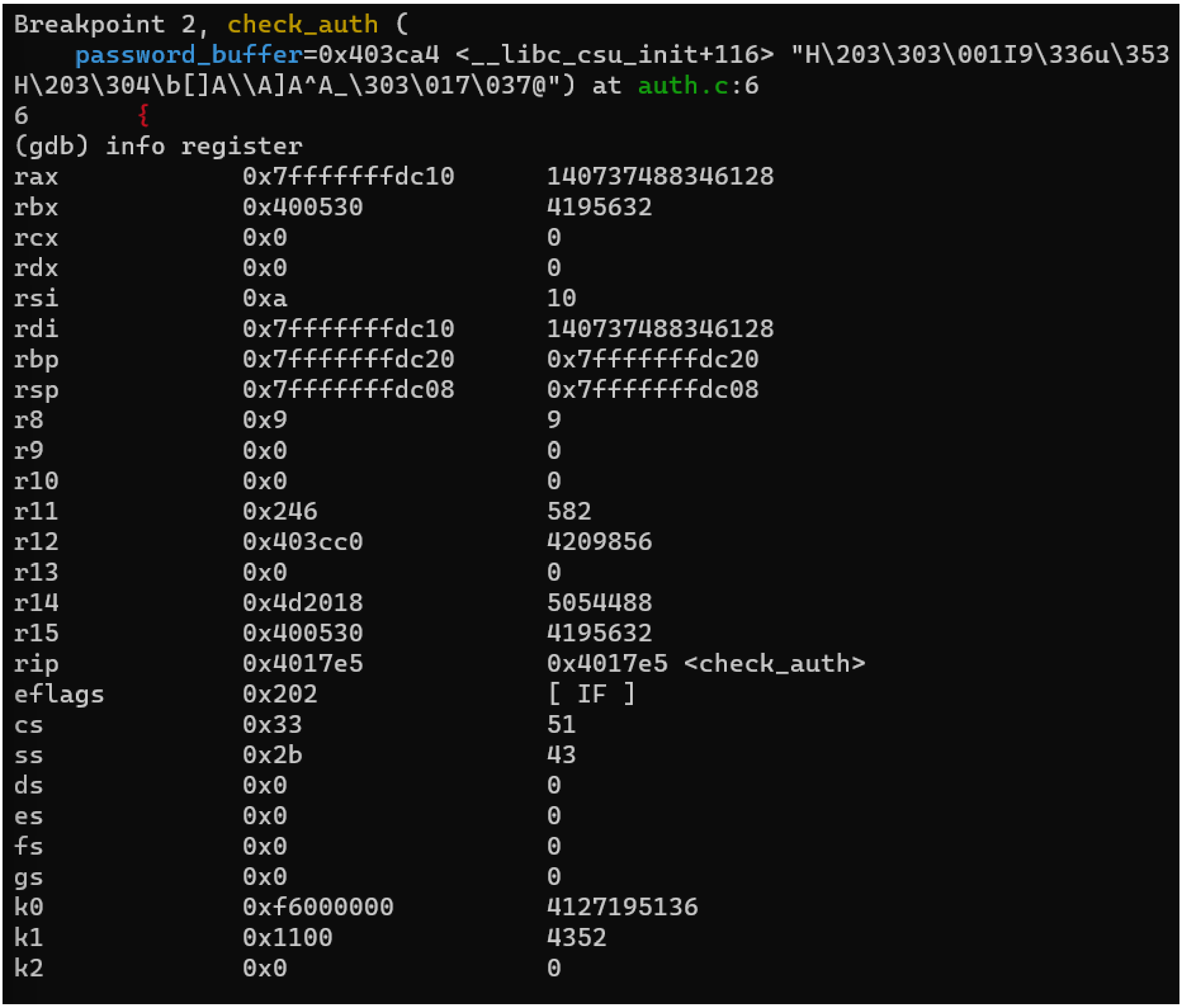
Now let start run command : run



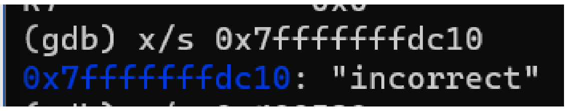
```
(gdb) run
Starting program: /mnt/e/clg 6th sem/crypto&net security/implem/auth
```



```
Breakpoint 5, 0x0000000000429dc0 in __strcmp_avx2 ()
(gdb) c
Continuing.
[-] Enter the password : incorrect

Breakpoint 1, 0x0000000000401866 in main () at auth.c:26
26            if (check_auth(password_buffer))
(gdb) c
Continuing.
```



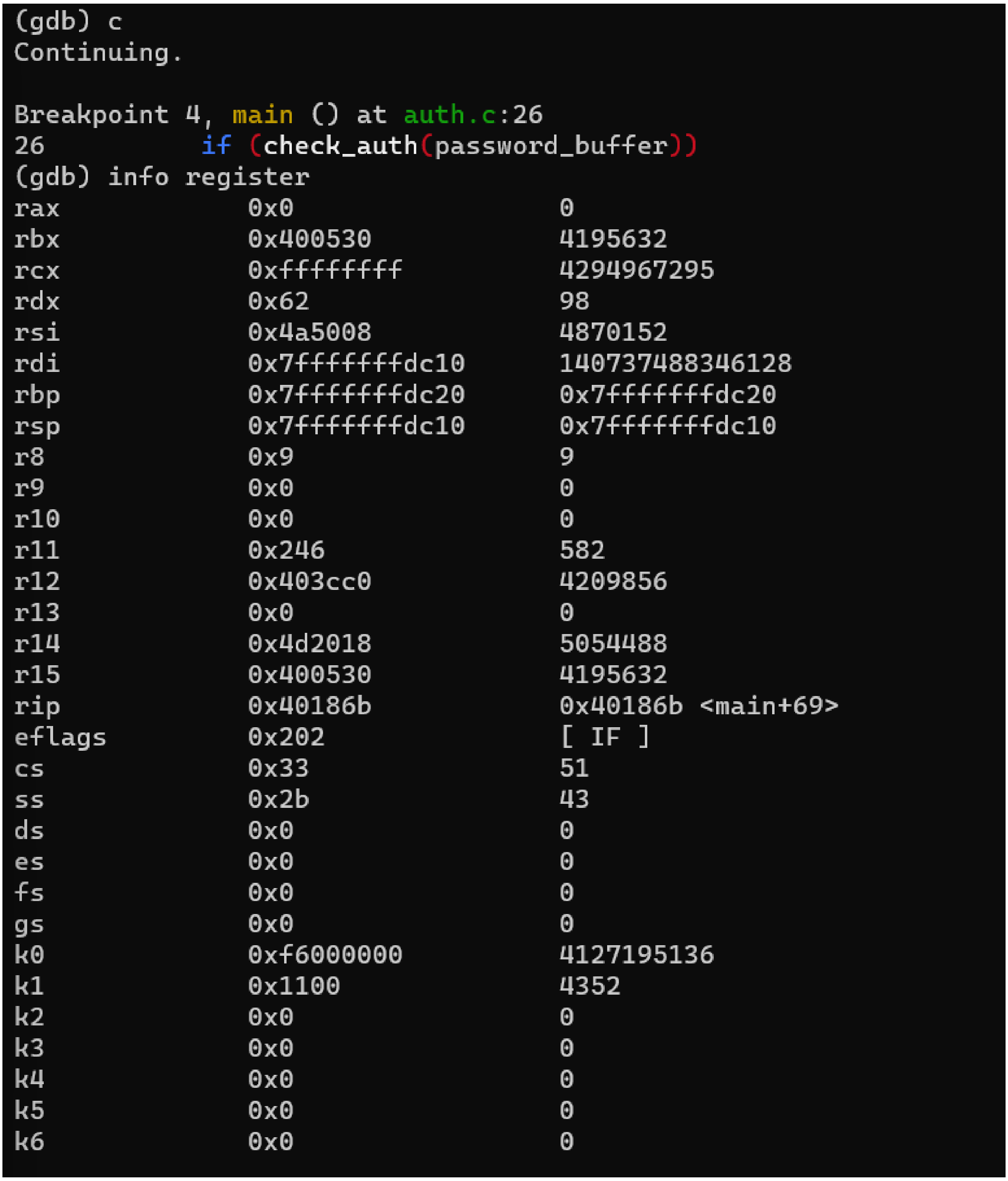
```
Breakpoint 2, check_auth (
    password_buffer=0x403ca4 <__libc_csu_init+116> "H\203\303\001I9\336u\353
H\203\304\b[]A\\A]A^A_\303\017\037@") at auth.c:6
6         {
(gdb) info register
rax            0x7fffffffdc10      140737488346128
rbx            0x400530            4195632
rcx            0x0                 0
rdx            0x0                 0
rsi            0xa                 10
rdi            0x7fffffffdc10      140737488346128
rbp            0x7fffffffdc20      0x7fffffffdc20
rsp            0x7fffffffdc08      0x7fffffffdc08
r8             0x9                 9
r9             0x0                 0
r10            0x0                 0
r11            0x246               582
r12            0x403cc0            4209856
r13            0x0                 0
r14            0x4d2018            5054488
r15            0x400530            4195632
rip            0x4017e5            0x4017e5 <check_auth>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0xf6000000          4127195136
k1             0x1100              4352
k2             0x0                 0
```



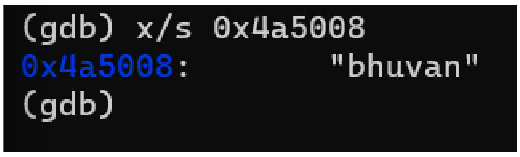
```
(gdb) x/s 0x7fffffffdc10
0x7fffffffdc10: "incorrect"
```

$rax stores the values from user_input password

Continue the program



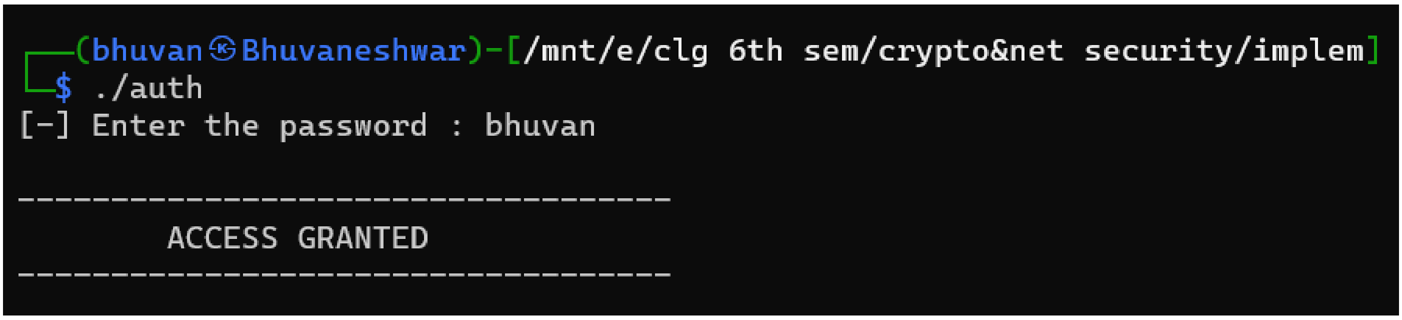
```
(gdb) c
Continuing.

Breakpoint 4, main () at auth.c:26
26          if (check_auth(password_buffer))
(gdb) info register
rax            0x0                 0
rbx            0x400530            4195632
rcx            0xffffffff          4294967295
rdx            0x62                98
rsi            0x4a5008            4870152
rdi            0x7fffffffdc10      140737488346128
rbp            0x7fffffffdc20      0x7fffffffdc20
rsp            0x7fffffffdc10      0x7fffffffdc10
r8             0x9                 9
r9             0x0                 0
r10            0x0                 0
r11            0x246               582
r12            0x403cc0            4209856
r13            0x0                 0
r14            0x4d2018            5054488
r15            0x400530            4195632
rip            0x40186b            0x40186b <main+69>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0xf6000000          4127195136
k1             0x1100              4352
k2             0x0                 0
k3             0x0                 0
k4             0x0                 0
k5             0x0                 0
k6             0x0                 0
```



```
(gdb) x/s 0x4a5008
0x4a5008:       "bhuvan"
(gdb)
```

Finally we crack the password through $rbp address.

Run again to program using “bhuvan” as user input

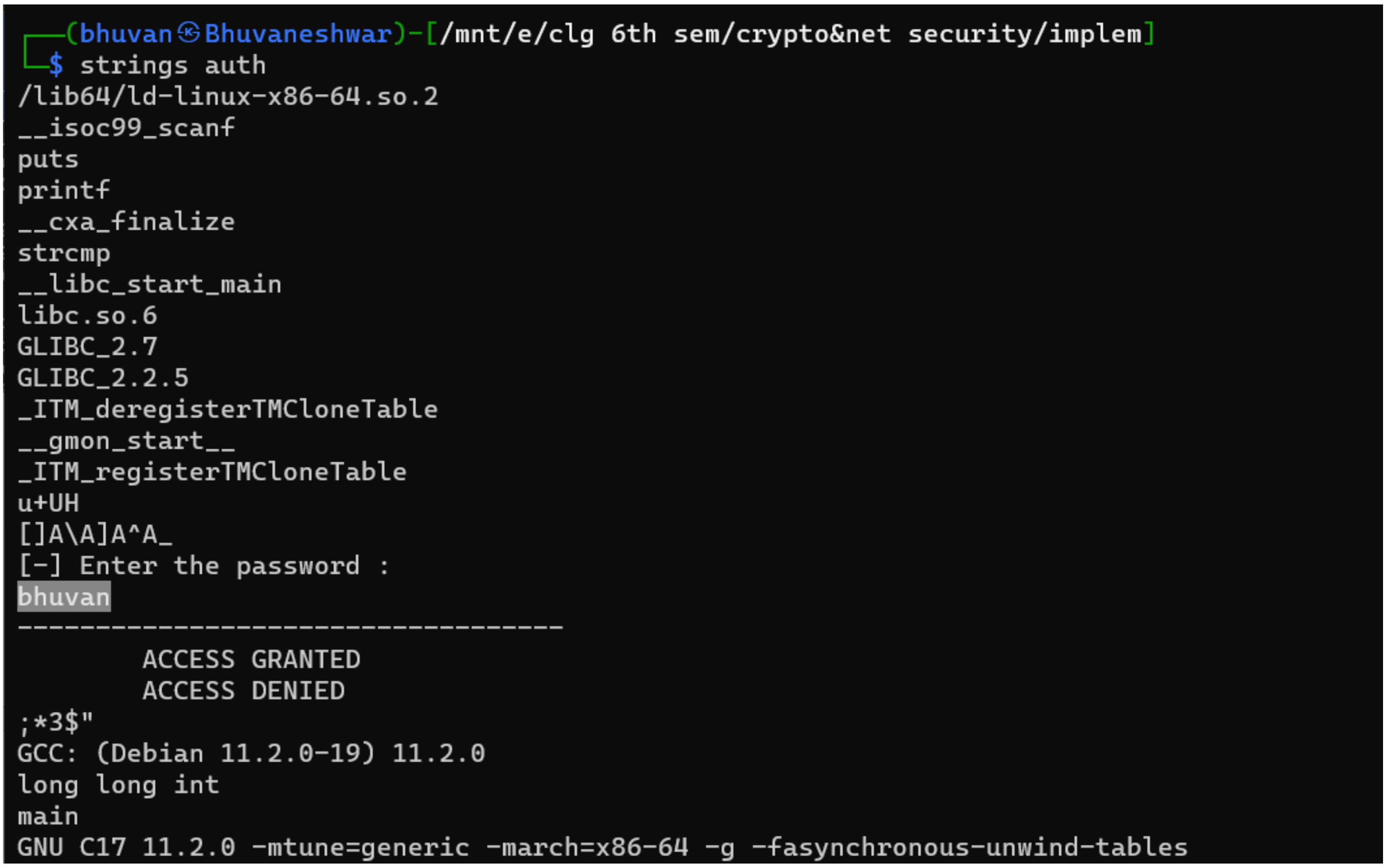


```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/implem]
└─$ ./auth
[-] Enter the password : bhuvan

-----------------------------------
        ACCESS GRANTED
-----------------------------------
```

Finally we cracked the password using gdb command.

ii) USING STRING COMMAND

A string is any sequence of 4 or more printable characters that end with a new line or null characters. Only if developer mentioned password in the code can see via strings in terminal, otherwise we can't see the password.



```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/implem]
└─$ strings auth
/lib64/ld-linux-x86-64.so.2
__isoc99_scanf
puts
printf
__cxa_finalize
strcmp
__libc_start_main
libc.so.6
GLIBC_2.7
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u+UH
[]A\A]A^A_
[-] Enter the password :
bhuvan
-----------------------------------
        ACCESS GRANTED
        ACCESS DENIED
;*3$"
GCC: (Debian 11.2.0-19) 11.2.0
long long int
main
GNU C17 11.2.0 -mtune=generic -march=x86-64 -g -fasynchronous-unwind-tables
```