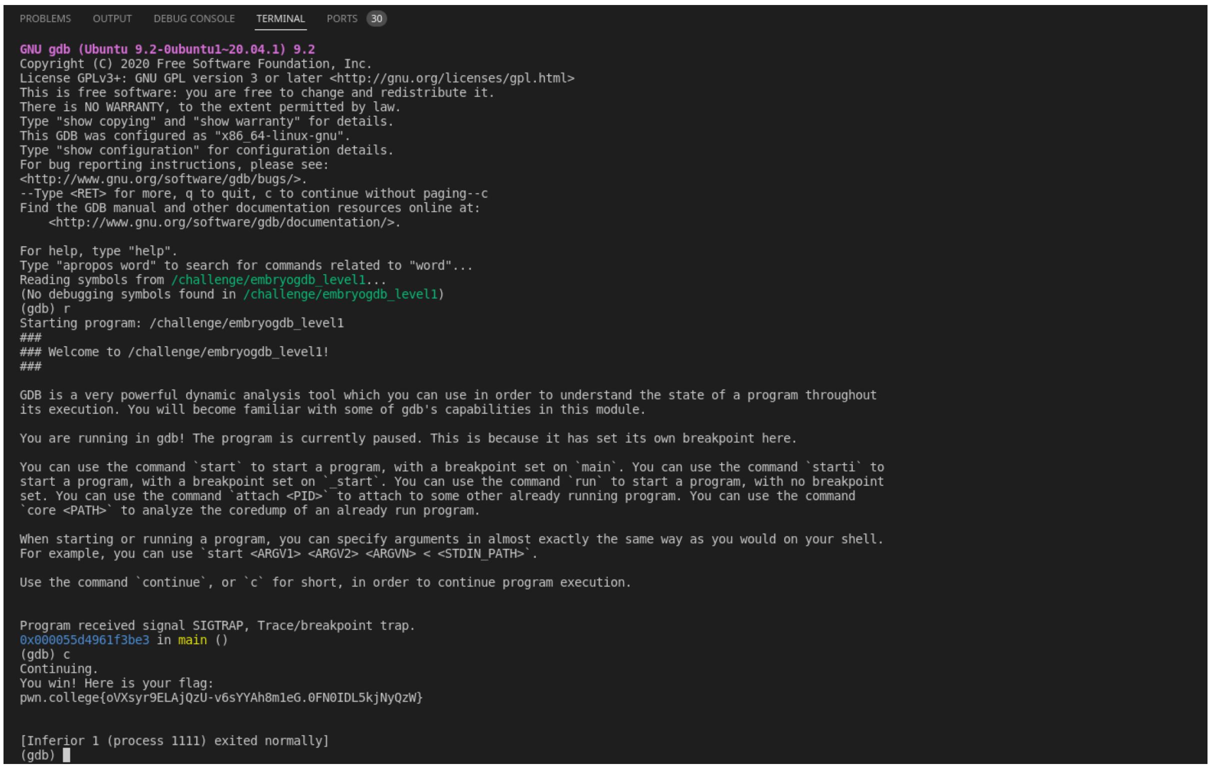
PWN COLLEGE GDB MODULE

CHALLENGES

BY JAIFIN B ALOOR

Level 1

I used /challenges/embryogdb_level1 to start the challenge. Then i used r command to run the program. There was a breakpoint at main and i used c command to countinue and i got the flag.

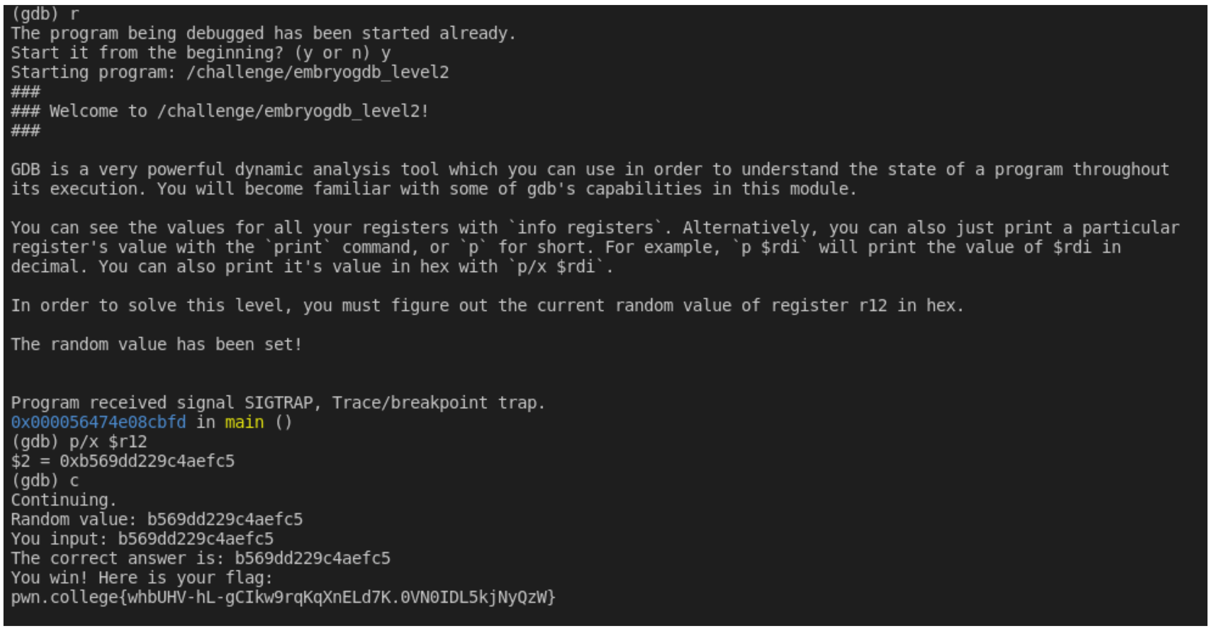


```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show 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/>.
--Type <RET> for more, q to quit, c to continue without paging--c
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 /challenge/embryogdb_level1...
(No debugging symbols found in /challenge/embryogdb_level1)
(gdb) r
Starting program: /challenge/embryogdb_level1
###
### Welcome to /challenge/embryogdb_level1!
###

GDB is a very powerful dynamic analysis tool which you can use in order to understand the state of a program throughout
its execution. You will become familiar with some of gdb's capabilities in this module.

You are running in gdb! The program is currently paused. This is because it has set its own breakpoint here.

You can use the command `start` to start a program, with a breakpoint set on `main`. You can use the command `starti` to
start a program, with a breakpoint set on `_start`. You can use the command `run` to start a program, with no breakpoint
set. You can use the command `attach <PID>` to attach to some other already running program. You can use the command
`core <PATH>` to analyze the coredump of an already run program.

When starting or running a program, you can specify arguments in almost exactly the same way as you would on your shell.
For example, you can use `start <ARGV1> <ARGV2> <ARGVN> < <STDIN_PATH>`.

Use the command `continue`, or `c` for short, in order to continue program execution.


Program received signal SIGTRAP, Trace/breakpoint trap.
0x000055d4961f3be3 in main ()
(gdb) c
Continuing.
You win! Here is your flag:
pwn.college{oVXsyr9ELAjQzU-v6sYYAh8m1eG.0FN0IDL5kjNyQzW}


[Inferior 1 (process 1111) exited normally]
(gdb) 
```

Level 2

P command is used for printing stuff. $reg means the value stored in the register reg.

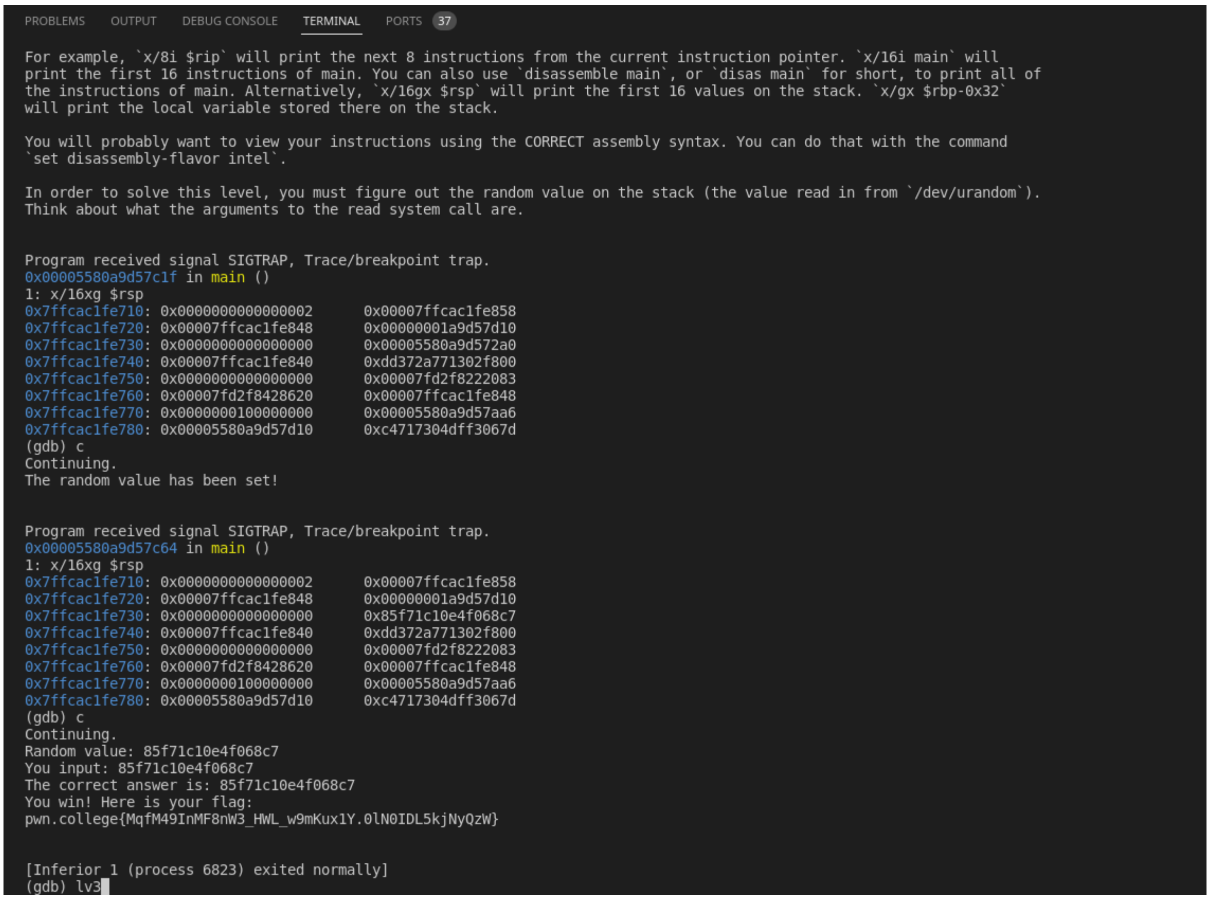
So p $reg prints the value stored in the register reg. Use p/x to print the value in the register in hex. I ran the program first with r and then printed the random value in r12 register with p/x $r12. Then i typed c to countinue and i got the flag.



```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /challenge/embryogdb_level2
###
### Welcome to /challenge/embryogdb_level2!
###

GDB is a very powerful dynamic analysis tool which you can use in order to understand the state of a program throughout
its execution. You will become familiar with some of gdb's capabilities in this module.

You can see the values for all your registers with `info registers`. Alternatively, you can also just print a particular
register's value with the `print` command, or `p` for short. For example, `p $rdi` will print the value of $rdi in
decimal. You can also print it's value in hex with `p/x $rdi`.

In order to solve this level, you must figure out the current random value of register r12 in hex.

The random value has been set!


Program received signal SIGTRAP, Trace/breakpoint trap.
0x000056474e08cbfd in main ()
(gdb) p/x $r12
$2 = 0xb569dd229c4aefc5
(gdb) c
Continuing.
Random value: b569dd229c4aefc5
You input: b569dd229c4aefc5
The correct answer is: b569dd229c4aefc5
You win! Here is your flag:
pwn.college{whbUHV-hL-gCIkw9rqKqXnELd7K.0VN0IDL5kjNyQzW}
```

Level 3

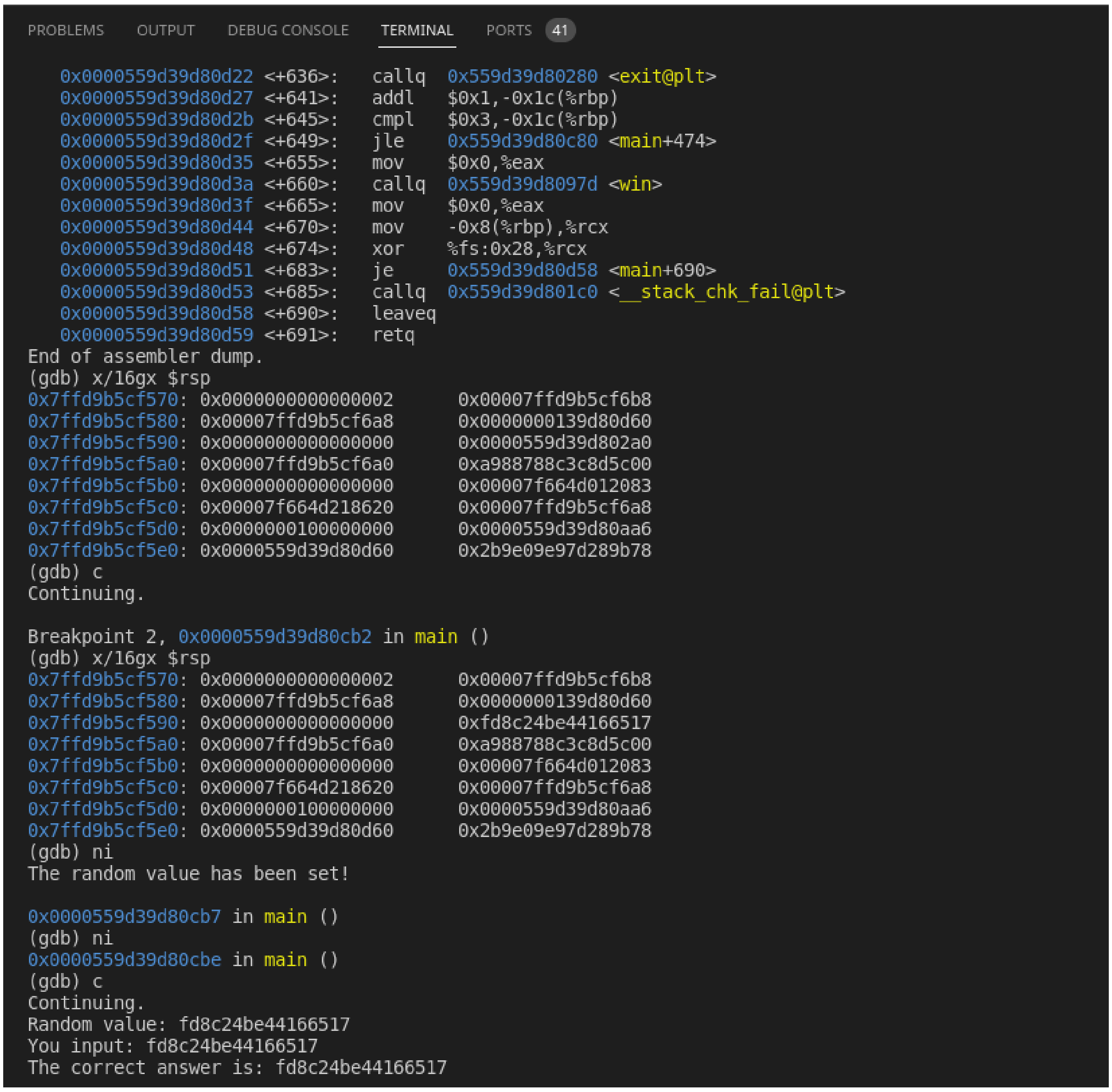
First i set a breakpoint after a random value has been set. Then i used x/16xg $rsp to examine the 16 gaint hex vlaues at the stack pointer. Then i typed c to countinue the program and the random value has been set. Then i used x/16gx again to see which of the values has been changed and only one value has been changed. So that was the random value. And thats how i got the flag.



```
For example, `x/8i $rip` will print the next 8 instructions from the current instruction pointer. `x/16i main` will
print the first 16 instructions of main. You can also use `disassemble main`, or `disas main` for short, to print all of
the instructions of main. Alternatively, `x/16gx $rsp` will print the first 16 values on the stack. `x/gx $rbp-0x32`
will print the local variable stored there on the stack.

You will probably want to view your instructions using the CORRECT assembly syntax. You can do that with the command
`set disassembly-flavor intel`.

In order to solve this level, you must figure out the random value on the stack (the value read in from `/dev/urandom`).
Think about what the arguments to the read system call are.


Program received signal SIGTRAP, Trace/breakpoint trap.
0x00005580a9d57c1f in main ()
1: x/16xg $rsp
0x7ffcac1fe710: 0x0000000000000002      0x00007ffcac1fe858
0x7ffcac1fe720: 0x00007ffcac1fe848      0x00000001a9d57d10
0x7ffcac1fe730: 0x0000000000000000      0x00005580a9d572a0
0x7ffcac1fe740: 0x00007ffcac1fe840      0xdd372a771302f800
0x7ffcac1fe750: 0x0000000000000000      0x00007fd2f8222083
0x7ffcac1fe760: 0x00007fd2f8428620      0x00007ffcac1fe848
0x7ffcac1fe770: 0x0000000100000000      0x00005580a9d57aa6
0x7ffcac1fe780: 0x00005580a9d57d10      0xc4717304dff3067d
(gdb) c
Continuing.
The random value has been set!


Program received signal SIGTRAP, Trace/breakpoint trap.
0x00005580a9d57c64 in main ()
1: x/16xg $rsp
0x7ffcac1fe710: 0x0000000000000002      0x00007ffcac1fe858
0x7ffcac1fe720: 0x00007ffcac1fe848      0x00000001a9d57d10
0x7ffcac1fe730: 0x0000000000000000      0x85f71c10e4f068c7
0x7ffcac1fe740: 0x00007ffcac1fe840      0xdd372a771302f800
0x7ffcac1fe750: 0x0000000000000000      0x00007fd2f8222083
0x7ffcac1fe760: 0x00007fd2f8428620      0x00007ffcac1fe848
0x7ffcac1fe770: 0x0000000100000000      0x00005580a9d57aa6
0x7ffcac1fe780: 0x00005580a9d57d10      0xc4717304dff3067d
(gdb) c
Continuing.
Random value: 85f71c10e4f068c7
You input: 85f71c10e4f068c7
The correct answer is: 85f71c10e4f068c7
You win! Here is your flag:
pwn.college{MqfM49InMF8nW3_HWL_w9mKux1Y.0lN0IDL5kjNyQzW}


[Inferior 1 (process 6823) exited normally]
(gdb) lv3
```

Level 4

I set 2 breakpoints befour and after the random value has been set.i run the program first. Then i typed x/16gx $rsp to get the 16 giant hex values at the rsp. Then i typed c to sountunue the program and the random value has been set and at the next breaakpoint again i did x/16gx $rsp and found out the random value. My input is the correct answer but i didnt get the flag though.



```
   0x0000559d39d80d22 <+636>:   callq  0x559d39d80280 <exit@plt>
   0x0000559d39d80d27 <+641>:   addl   $0x1,-0x1c(%rbp)
   0x0000559d39d80d2b <+645>:   cmpl   $0x3,-0x1c(%rbp)
   0x0000559d39d80d2f <+649>:   jle    0x559d39d80c80 <main+474>
   0x0000559d39d80d35 <+655>:   mov    $0x0,%eax
   0x0000559d39d80d3a <+660>:   callq  0x559d39d8097d <win>
   0x0000559d39d80d3f <+665>:   mov    $0x0,%eax
   0x0000559d39d80d44 <+670>:   mov    -0x8(%rbp),%rcx
   0x0000559d39d80d48 <+674>:   xor    %fs:0x28,%rcx
   0x0000559d39d80d51 <+683>:   je     0x559d39d80d58 <main+690>
   0x0000559d39d80d53 <+685>:   callq  0x559d39d801c0 <__stack_chk_fail@plt>
   0x0000559d39d80d58 <+690>:   leaveq
   0x0000559d39d80d59 <+691>:   retq
End of assembler dump.
(gdb) x/16gx $rsp
0x7ffd9b5cf570: 0x0000000000000002      0x00007ffd9b5cf6b8
0x7ffd9b5cf580: 0x00007ffd9b5cf6a8      0x0000000139d80d60
0x7ffd9b5cf590: 0x0000000000000000      0x0000559d39d802a0
0x7ffd9b5cf5a0: 0x00007ffd9b5cf6a0      0xa988788c3c8d5c00
0x7ffd9b5cf5b0: 0x0000000000000000      0x00007f664d012083
0x7ffd9b5cf5c0: 0x00007f664d218620      0x00007ffd9b5cf6a8
0x7ffd9b5cf5d0: 0x0000000100000000      0x0000559d39d80aa6
0x7ffd9b5cf5e0: 0x0000559d39d80d60      0x2b9e09e97d289b78
(gdb) c
Continuing.

Breakpoint 2, 0x0000559d39d80cb2 in main ()
(gdb) x/16gx $rsp
0x7ffd9b5cf570: 0x0000000000000002      0x00007ffd9b5cf6b8
0x7ffd9b5cf580: 0x00007ffd9b5cf6a8      0x0000000139d80d60
0x7ffd9b5cf590: 0x0000000000000000      0xfd8c24be44166517
0x7ffd9b5cf5a0: 0x00007ffd9b5cf6a0      0xa988788c3c8d5c00
0x7ffd9b5cf5b0: 0x0000000000000000      0x00007f664d012083
0x7ffd9b5cf5c0: 0x00007f664d218620      0x00007ffd9b5cf6a8
0x7ffd9b5cf5d0: 0x0000000100000000      0x0000559d39d80aa6
0x7ffd9b5cf5e0: 0x0000559d39d80d60      0x2b9e09e97d289b78
(gdb) ni
The random value has been set!

0x0000559d39d80cb7 in main ()
(gdb) ni
0x0000559d39d80cbe in main ()
(gdb) c
Continuing.
Random value: fd8c24be44166517
You input: fd8c24be44166517
The correct answer is: fd8c24be44166517
```

CHALLENGE FLAGS

lv1 = pwn.college{oVXsyr9ELAjQzU-v6sYYAh8m1eG.0FN0IDL5kjNyQzW}

lv2 = pwn.college{whbUHV-hL-gCIkw9rqKqXnELd7K.0VN0IDL5kjNyQzW}

lv3 = pwn.college{MqfM49InMF8nW3_HWL_w9mKux1Y.0lN0IDL5kjNyQzW}