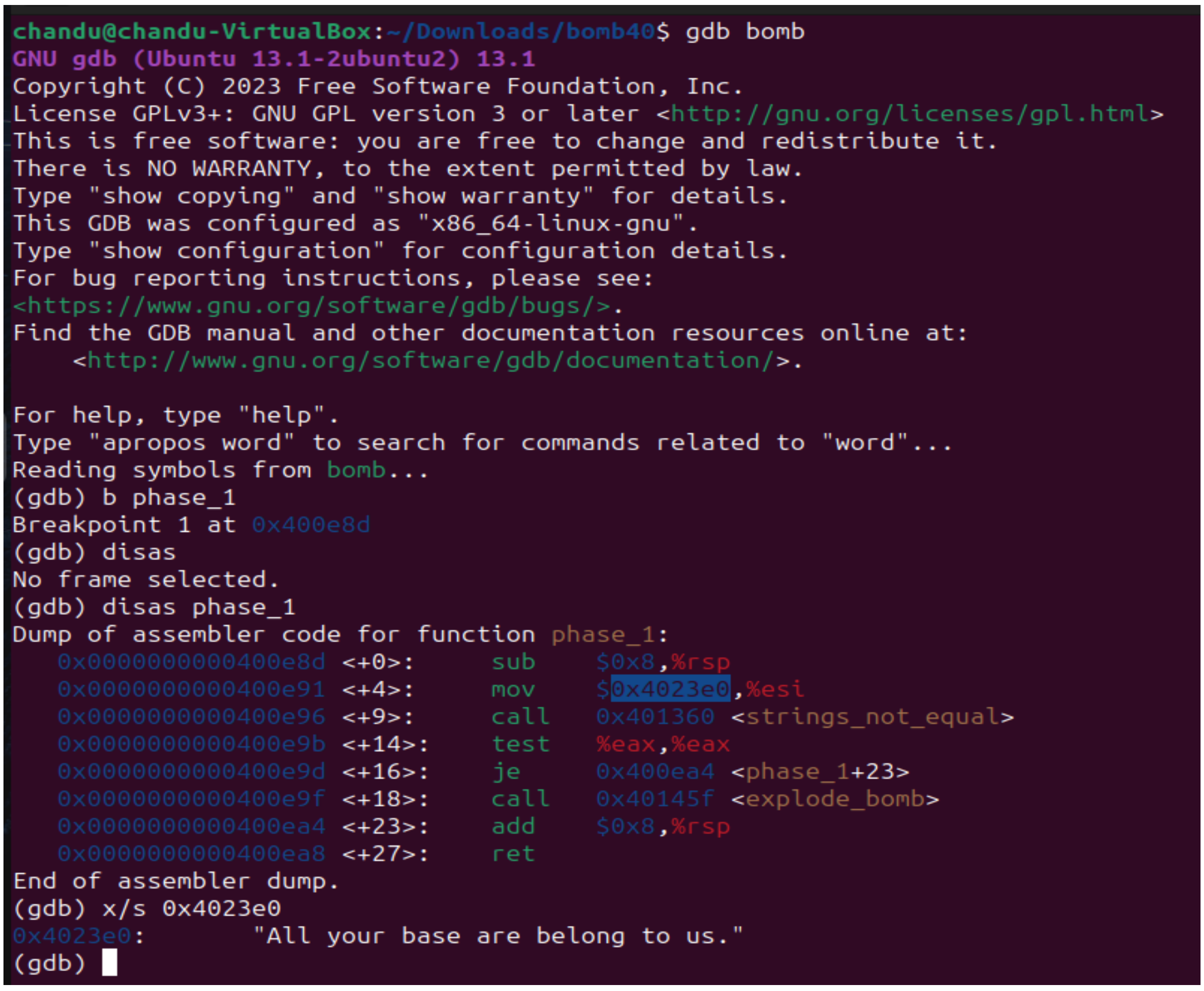
Frequently used gdb bombs:

- objdump -d bomb we use this command to disassemble bomb40.c
- gbd bomb command is used to run the code and to to diffuse the the phases.
- Then break<location> is used to get the breakpoint at particular phase.
- Then we use disas to get the particular breakpoint assembly phase code.
- Then we use i r command to get the info of the registers.
- And also we use n i to go to the next instruction in particular assembly phase.
- x/s 0xAddress
- x/d 0xAddress

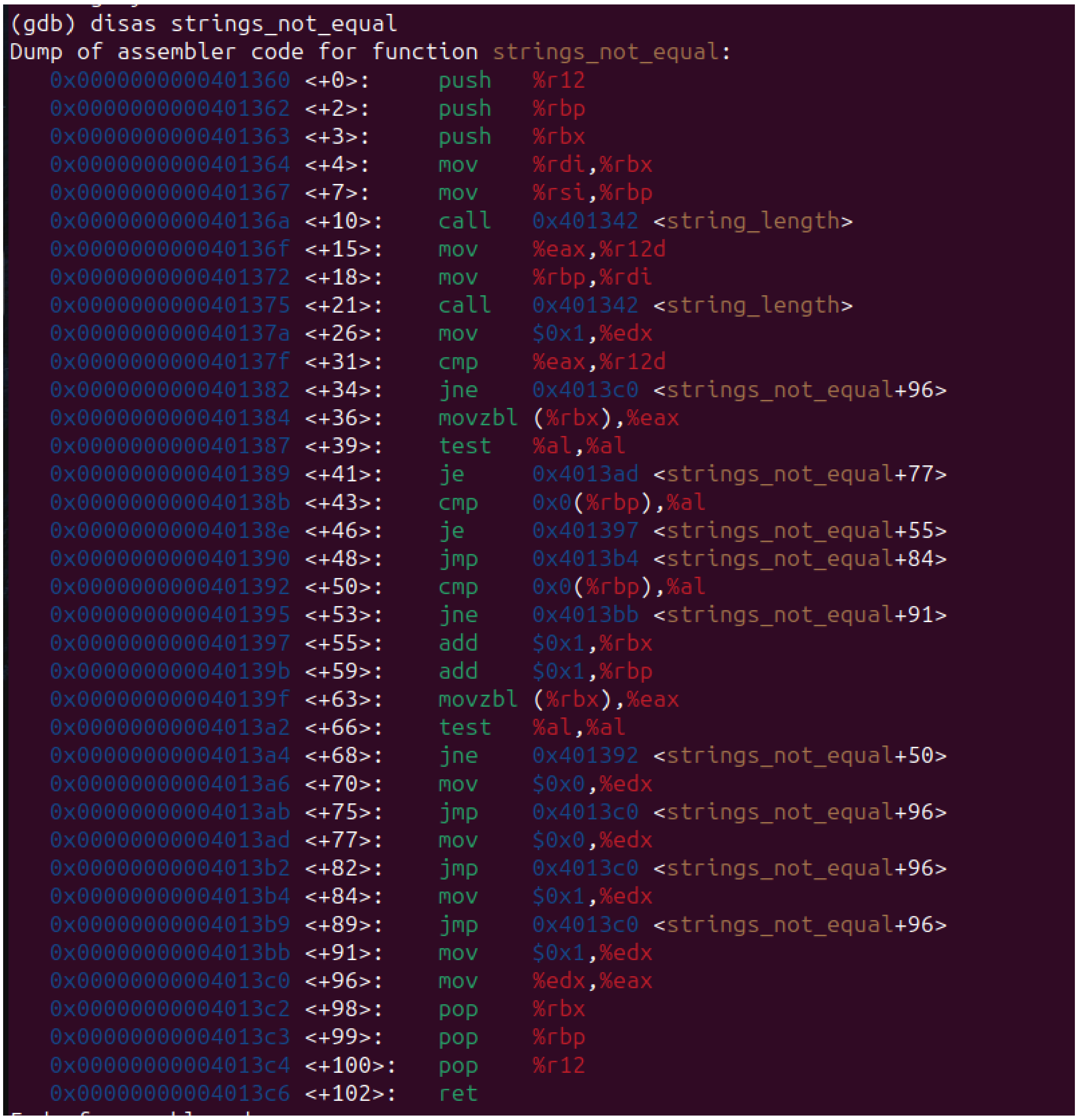
Phase 1:

- Use command “gdb bomb” to start gdb debugger in the terminal
- Then add break point at phase_1 using “break” command.



```
chandu@chandu-VirtualBox:~/Downloads/bomb40$ gdb bomb
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
Copyright (C) 2023 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 bomb...
(gdb) b phase_1
Breakpoint 1 at 0x400e8d
(gdb) disas
No frame selected.
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023e0,%esi
   0x0000000000400e96 <+9>:     call   0x401360 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x40145f <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) x/s 0x4023e0
0x4023e0:       "All your base are belong to us."
(gdb) 
```

- Now run the program and enter any dummy input.
- The program will break at phase1.
- Here I have used disas phase_1 to get the assembly code of phase1
- Then I have used x/s 0x4023e0 to know the what is stored in the esi.
- Check memory $0x402390 (x/s 0x402390),it has a string which is being moved into $esi register.And passes onto string_not_equal.



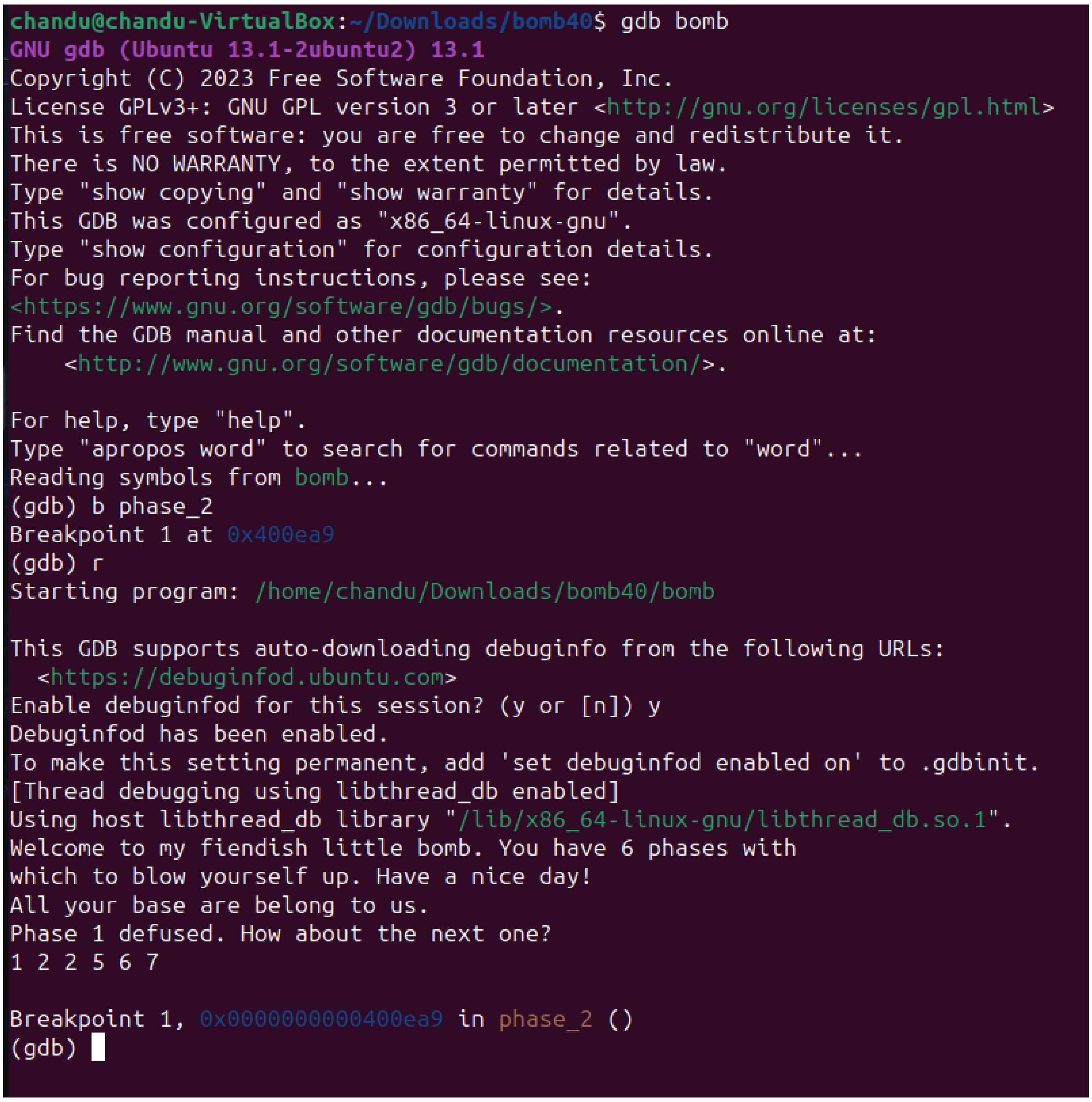
```
(gdb) disas strings_not_equal
Dump of assembler code for function strings_not_equal:
   0x0000000000401360 <+0>:     push   %r12
   0x0000000000401362 <+2>:     push   %rbp
   0x0000000000401363 <+3>:     push   %rbx
   0x0000000000401364 <+4>:     mov    %rdi,%rbx
   0x0000000000401367 <+7>:     mov    %rsi,%rbp
   0x000000000040136a <+10>:    call   0x401342 <string_length>
   0x000000000040136f <+15>:    mov    %eax,%r12d
   0x0000000000401372 <+18>:    mov    %rbp,%rdi
   0x0000000000401375 <+21>:    call   0x401342 <string_length>
   0x000000000040137a <+26>:    mov    $0x1,%edx
   0x000000000040137f <+31>:    cmp    %eax,%r12d
   0x0000000000401382 <+34>:    jne    0x4013c0 <strings_not_equal+96>
   0x0000000000401384 <+36>:    movzbl (%rbx),%eax
   0x0000000000401387 <+39>:    test   %al,%al
   0x0000000000401389 <+41>:    je     0x4013ad <strings_not_equal+77>
   0x000000000040138b <+43>:    cmp    0x0(%rbp),%al
   0x000000000040138e <+46>:    je     0x401397 <strings_not_equal+55>
   0x0000000000401390 <+48>:    jmp    0x4013b4 <strings_not_equal+84>
   0x0000000000401392 <+50>:    cmp    0x0(%rbp),%al
   0x0000000000401395 <+53>:    jne    0x4013bb <strings_not_equal+91>
   0x0000000000401397 <+55>:    add    $0x1,%rbx
   0x000000000040139b <+59>:    add    $0x1,%rbp
   0x000000000040139f <+63>:    movzbl (%rbx),%eax
   0x00000000004013a2 <+66>:    test   %al,%al
   0x00000000004013a4 <+68>:    jne    0x401392 <strings_not_equal+50>
   0x00000000004013a6 <+70>:    mov    $0x0,%edx
   0x00000000004013ab <+75>:    jmp    0x4013c0 <strings_not_equal+96>
   0x00000000004013ad <+77>:    mov    $0x0,%edx
   0x00000000004013b2 <+82>:    jmp    0x4013c0 <strings_not_equal+96>
   0x00000000004013b4 <+84>:    mov    $0x1,%edx
   0x00000000004013b9 <+89>:    jmp    0x4013c0 <strings_not_equal+96>
   0x00000000004013bb <+91>:    mov    $0x1,%edx
   0x00000000004013c0 <+96>:    mov    %edx,%eax
   0x00000000004013c2 <+98>:    pop    %rbx
   0x00000000004013c3 <+99>:    pop    %rbp
   0x00000000004013c4 <+100>:   pop    %r12
   0x00000000004013c6 <+102>:   ret
```

• Here it is checking user input and the string in $esi registers are same,if not explode_bomb will be called in phase1.
• So, the solution key for phase1 is the string that moved into $esi register.
• In my case string is “All your base are belongs to us.”.
• Save this solution keys for phases in a text file.

Phase 2 :

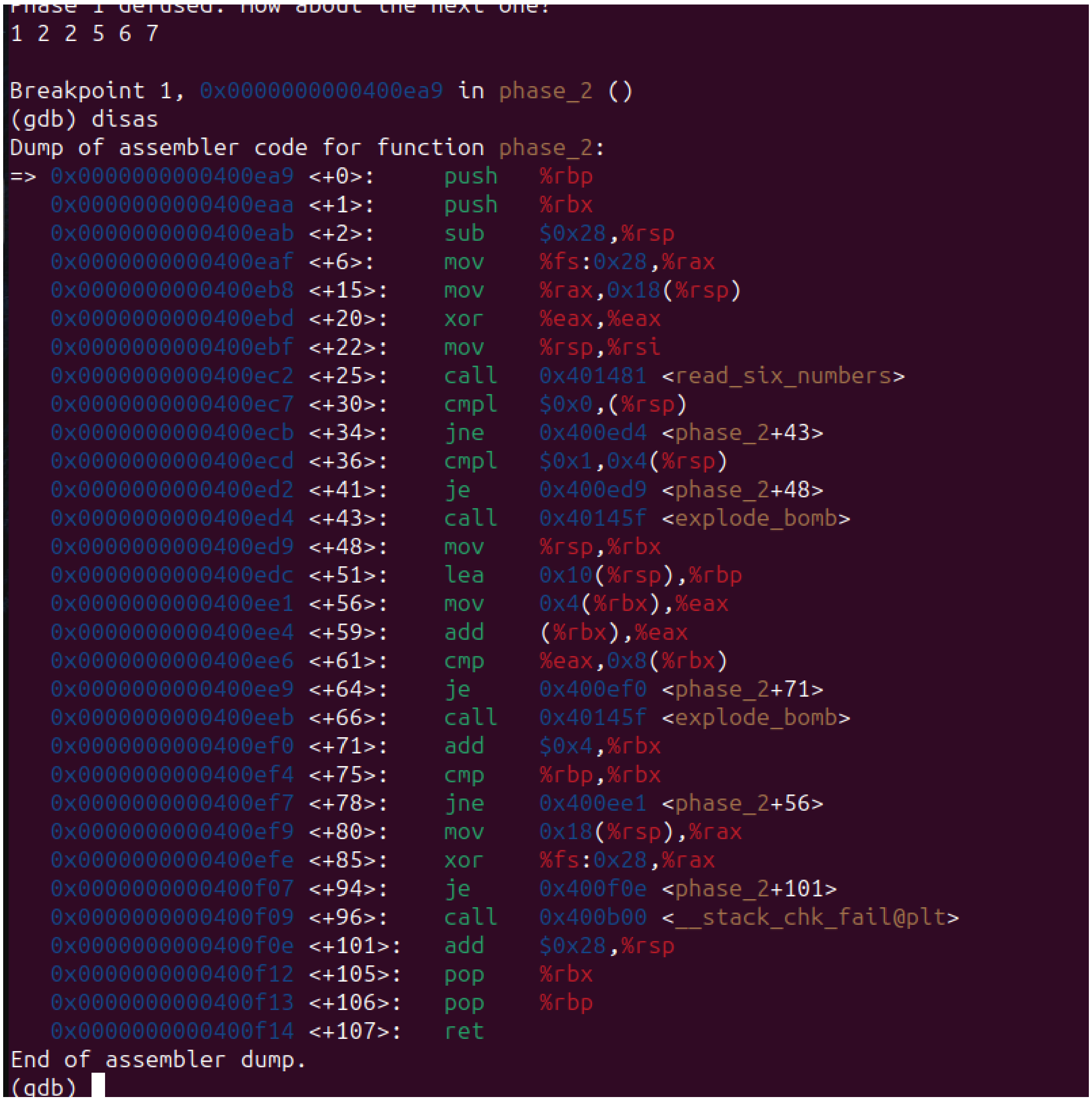
• Start the gdb debugger and add break point at phase_2.

• Run the program with solutions text file as argument.And enter a dummy input for phase2.



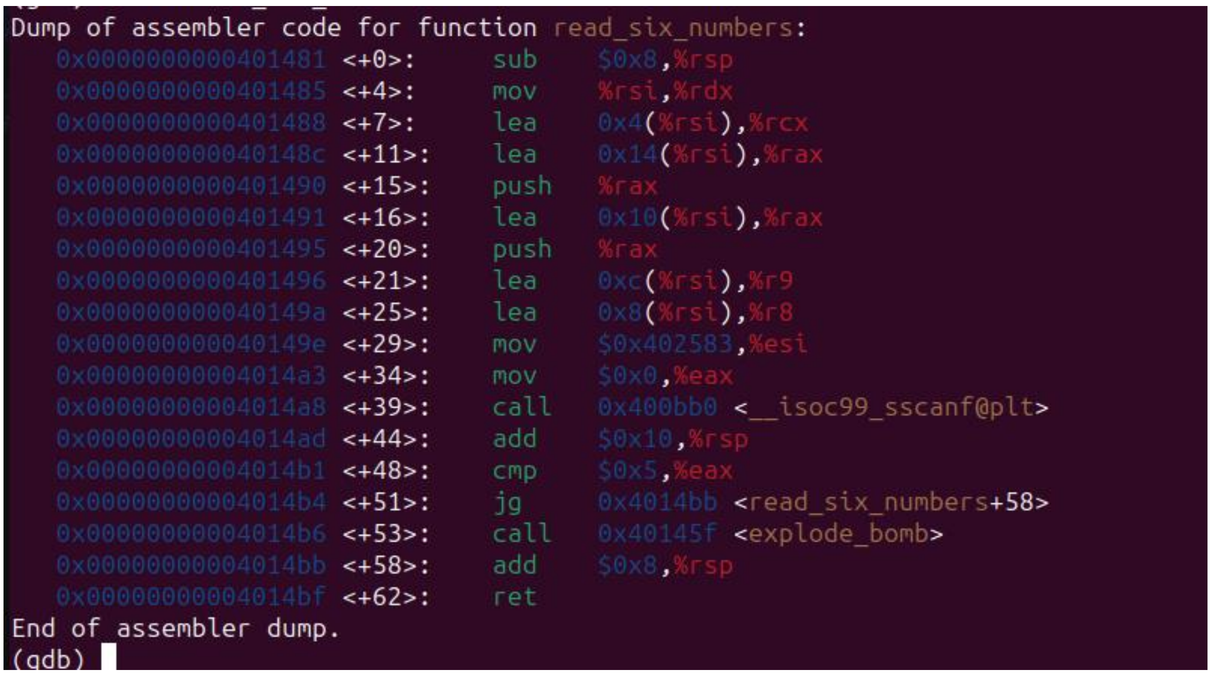
```
chandu@chandu-VirtualBox:~/Downloads/bomb40$ gdb bomb
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO 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 bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400ea9
(gdb) r
Starting program: /home/chandu/Downloads/bomb40/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
All your base are belong to us.
Phase 1 defused. How about the next one?
1 2 2 5 6 7

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) 
```

• Now use disas command to see the assembly code of phase2.



```
Phase 1 defused. How about the next one?
1 2 2 5 6 7

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    call   0x401481 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    call   0x40145f <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    call   0x40145f <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   ret
End of assembler dump.
(gdb) 
```

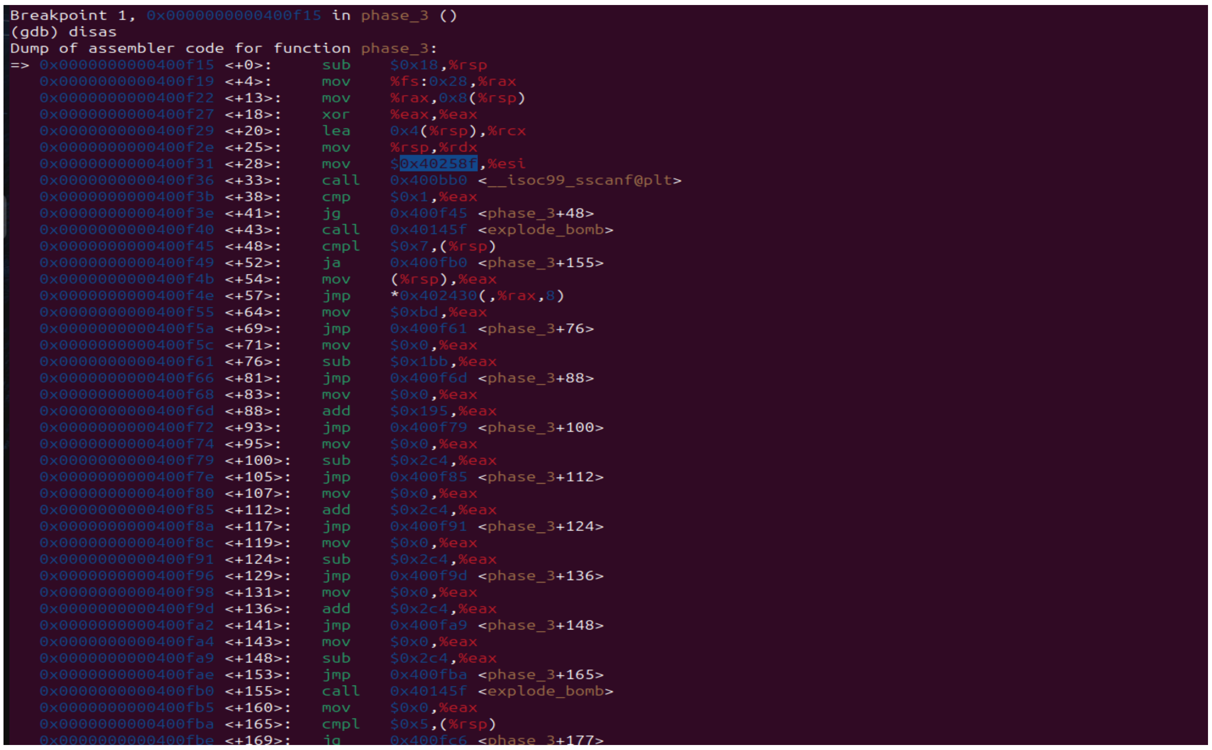
• We can see read_six_numbers function.If we examine assembly code of this function we can conclude that solution format for phase2 is six numbers.



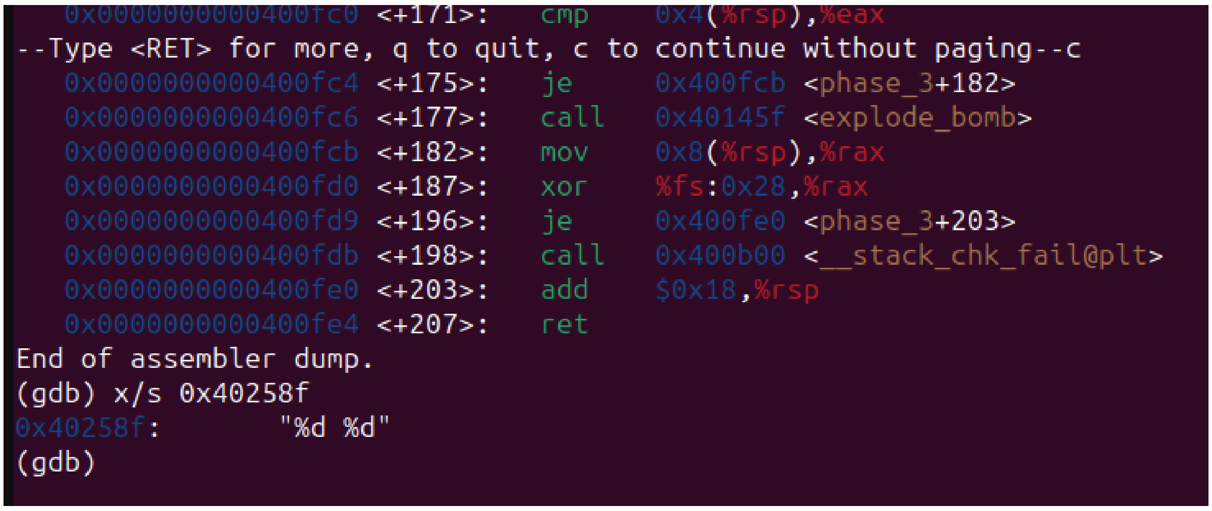
```
Dump of assembler code for function read_six_numbers:
   0x0000000000401481 <+0>:     sub    $0x8,%rsp
   0x0000000000401485 <+4>:     mov    %rsi,%rdx
   0x0000000000401488 <+7>:     lea    0x4(%rsi),%rcx
   0x000000000040148c <+11>:    lea    0x14(%rsi),%rax
   0x0000000000401490 <+15>:    push   %rax
   0x0000000000401491 <+16>:    lea    0x10(%rsi),%rax
   0x0000000000401495 <+20>:    push   %rax
   0x0000000000401496 <+21>:    lea    0xc(%rsi),%r9
   0x000000000040149a <+25>:    lea    0x8(%rsi),%r8
   0x000000000040149e <+29>:    mov    $0x402583,%esi
   0x00000000004014a3 <+34>:    mov    $0x0,%eax
   0x00000000004014a8 <+39>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x00000000004014ad <+44>:    add    $0x10,%rsp
   0x00000000004014b1 <+48>:    cmp    $0x5,%eax
   0x00000000004014b4 <+51>:    jg     0x4014bb <read_six_numbers+58>
   0x00000000004014b6 <+53>:    call   0x40145f <explode_bomb>
   0x00000000004014bb <+58>:    add    $0x8,%rsp
   0x00000000004014bf <+62>:    ret
End of assembler dump.
(gdb)
```

- In phase2 there is loop executing,it is doubling the input in $eax andcomparing it with $rbx+4,now %(rbx),eax is here in every loop eax is increasing by 1 and and the next value is stored in rbx+4.so the answer will be 0 1 1 2 3 5.Here eax is increased by 1 in every loop and added by previous rbx value so that you will get the answer.
- If we check 0 1 1 2 3 5 its worked.

Phase 3:

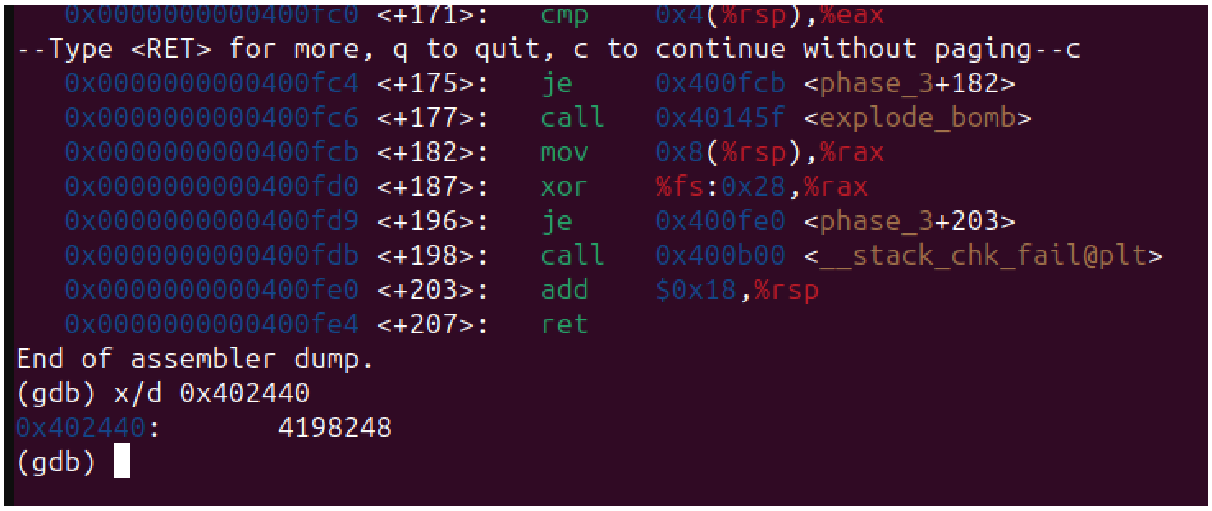


```
Breakpoint 1, 0x0000000000400f15 in phase_3 ()
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x0000000000400f15 <+0>:     sub    $0x18,%rsp
   0x0000000000400f19 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f22 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400f27 <+18>:    xor    %eax,%eax
   0x0000000000400f29 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400f2e <+25>:    mov    %rsp,%rdx
   0x0000000000400f31 <+28>:    mov    $0x40258f,%esi
   0x0000000000400f36 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:    cmp    $0x1,%eax
   0x0000000000400f3e <+41>:    jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:    call   0x40145f <explode_bomb>
   0x0000000000400f45 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:    ja     0x400fb0 <phase_3+155>
   0x0000000000400f4b <+54>:    mov    (%rsp),%eax
   0x0000000000400f4e <+57>:    jmp    *0x402430(,%rax,8)
   0x0000000000400f55 <+64>:    mov    $0xbd,%eax
   0x0000000000400f5a <+69>:    jmp    0x400f61 <phase_3+76>
   0x0000000000400f5c <+71>:    mov    $0x0,%eax
   0x0000000000400f61 <+76>:    sub    $0x1bb,%eax
   0x0000000000400f66 <+81>:    jmp    0x400f6d <phase_3+88>
   0x0000000000400f68 <+83>:    mov    $0x0,%eax
   0x0000000000400f6d <+88>:    add    $0x195,%eax
   0x0000000000400f72 <+93>:    jmp    0x400f79 <phase_3+100>
   0x0000000000400f74 <+95>:    mov    $0x0,%eax
   0x0000000000400f79 <+100>:   sub    $0x2c4,%eax
   0x0000000000400f7e <+105>:   jmp    0x400f85 <phase_3+112>
   0x0000000000400f80 <+107>:   mov    $0x0,%eax
   0x0000000000400f85 <+112>:   add    $0x2c4,%eax
   0x0000000000400f8a <+117>:   jmp    0x400f91 <phase_3+124>
   0x0000000000400f8c <+119>:   mov    $0x0,%eax
   0x0000000000400f91 <+124>:   sub    $0x2c4,%eax
   0x0000000000400f96 <+129>:   jmp    0x400f9d <phase_3+136>
   0x0000000000400f98 <+131>:   mov    $0x0,%eax
   0x0000000000400f9d <+136>:   add    $0x2c4,%eax
   0x0000000000400fa2 <+141>:   jmp    0x400fa9 <phase_3+148>
   0x0000000000400fa4 <+143>:   mov    $0x0,%eax
   0x0000000000400fa9 <+148>:   sub    $0x2c4,%eax
   0x0000000000400fae <+153>:   jmp    0x400fba <phase_3+165>
   0x0000000000400fb0 <+155>:   call   0x40145f <explode_bomb>
   0x0000000000400fb5 <+160>:   mov    $0x0,%eax
   0x0000000000400fba <+165>:   cmpl   $0x5,(%rsp)
   0x0000000000400fbe <+169>:   jg     0x400fc6 <phase_3+177>
```

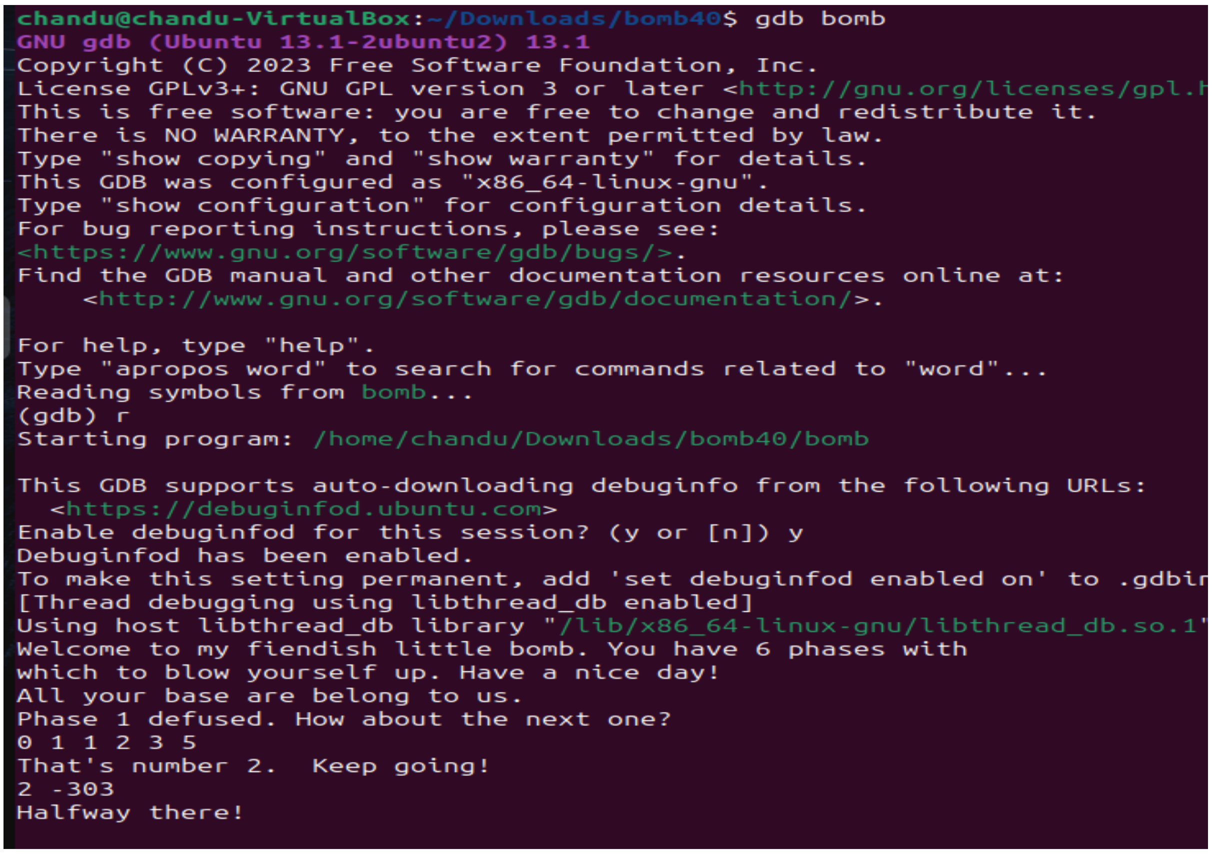


```
   0x0000000000400fc0 <+171>:   cmp    0x4(%rsp),%eax
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400fc4 <+175>:   je     0x400fcb <phase_3+182>
   0x0000000000400fc6 <+177>:   call   0x40145f <explode_bomb>
   0x0000000000400fcb <+182>:   mov    0x8(%rsp),%rax
   0x0000000000400fd0 <+187>:   xor    %fs:0x28,%rax
   0x0000000000400fd9 <+196>:   je     0x400fe0 <phase_3+203>
   0x0000000000400fdb <+198>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fe0 <+203>:   add    $0x18,%rsp
   0x0000000000400fe4 <+207>:   ret
End of assembler dump.
(gdb) x/s 0x40258f
0x40258f:       "%d %d"
(gdb)
```

- Here in x/s 0x40258f you can check there will be two input numbers
- *0x402430(,%rax,8) by calculating it you will get 0x402440 then you will get 4198248 by converting into an hexadecimal number number you will get 400f68 from there to do all subtraction,addition whenever ther is a jump you have to jump there and you to do add or sub upto last the %eax value is changing.
- Here I took %rax as 2 there is a condition like the first input number must be greater than 1 and less than 7.
- When I took as 2 you will get 4198248.
- By this all I got ‘2 -303’where it got worked.

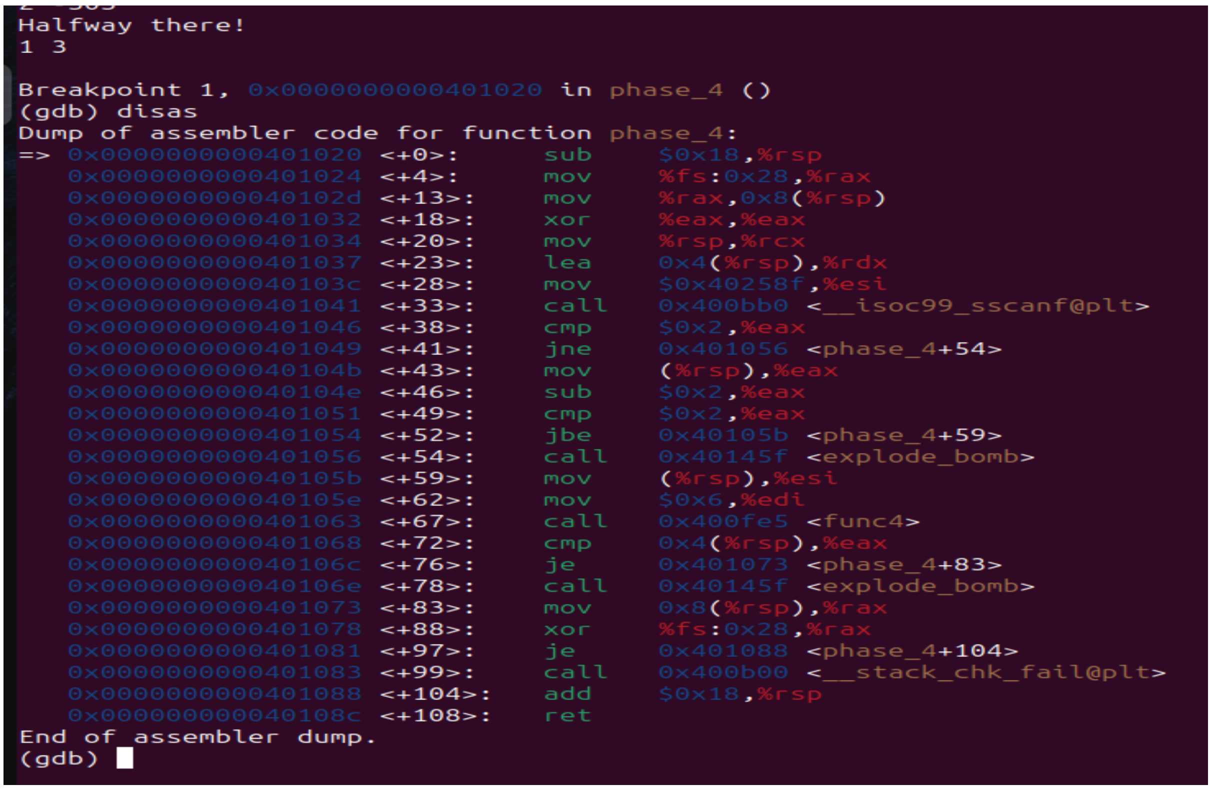


```
   0x0000000000400fc0 <+171>:   cmp    0x4(%rsp),%eax
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400fc4 <+175>:   je     0x400fcb <phase_3+182>
   0x0000000000400fc6 <+177>:   call   0x40145f <explode_bomb>
   0x0000000000400fcb <+182>:   mov    0x8(%rsp),%rax
   0x0000000000400fd0 <+187>:   xor    %fs:0x28,%rax
   0x0000000000400fd9 <+196>:   je     0x400fe0 <phase_3+203>
   0x0000000000400fdb <+198>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fe0 <+203>:   add    $0x18,%rsp
   0x0000000000400fe4 <+207>:   ret
End of assembler dump.
(gdb) x/d 0x402440
0x402440:       4198248
(gdb) 
```



```
chandu@chandu-VirtualBox:~/Downloads/bomb40$ gdb bomb
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.h
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 bomb...
(gdb) r
Starting program: /home/chandu/Downloads/bomb40/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbir
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1"
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
All your base are belong to us.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
2 -303
Halfway there!
```

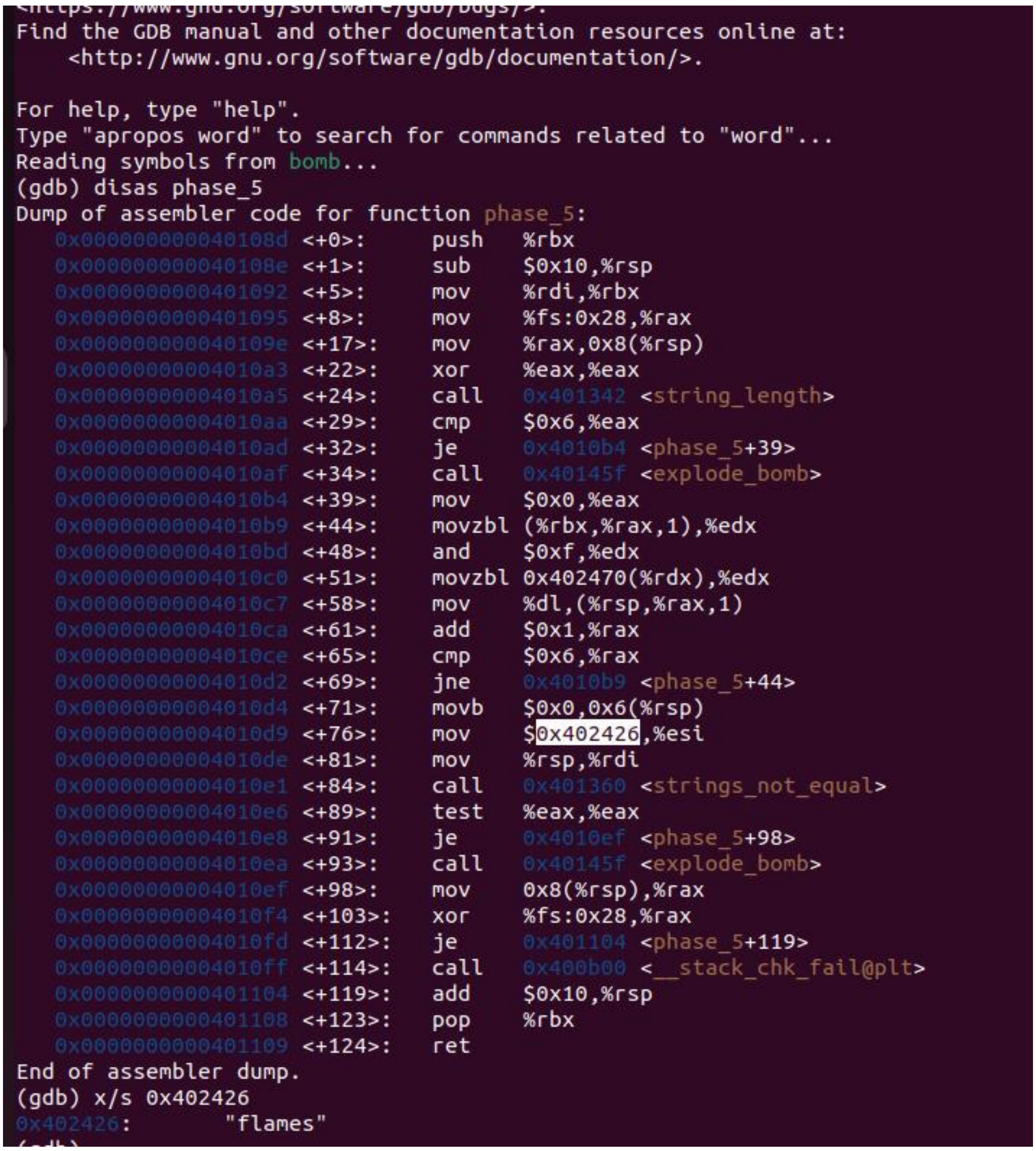
Phase 4:



```
Halfway there!
1 3

Breakpoint 1, 0x0000000000401020 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x0000000000401020 <+0>:     sub    $0x18,%rsp
   0x0000000000401024 <+4>:     mov    %fs:0x28,%rax
   0x000000000040102d <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401032 <+18>:    xor    %eax,%eax
   0x0000000000401034 <+20>:    mov    %rsp,%rcx
   0x0000000000401037 <+23>:    lea    0x4(%rsp),%rdx
   0x000000000040103c <+28>:    mov    $0x40258f,%esi
   0x0000000000401041 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401046 <+38>:    cmp    $0x2,%eax
   0x0000000000401049 <+41>:    jne    0x401056 <phase_4+54>
   0x000000000040104b <+43>:    mov    (%rsp),%eax
   0x000000000040104e <+46>:    sub    $0x2,%eax
   0x0000000000401051 <+49>:    cmp    $0x2,%eax
   0x0000000000401054 <+52>:    jbe    0x40105b <phase_4+59>
   0x0000000000401056 <+54>:    call   0x40145f <explode_bomb>
   0x000000000040105b <+59>:    mov    (%rsp),%esi
   0x000000000040105e <+62>:    mov    $0x6,%edi
   0x0000000000401063 <+67>:    call   0x400fe5 <func4>
   0x0000000000401068 <+72>:    cmp    0x4(%rsp),%eax
   0x000000000040106c <+76>:    je     0x401073 <phase_4+83>
   0x000000000040106e <+78>:    call   0x40145f <explode_bomb>
   0x0000000000401073 <+83>:    mov    0x8(%rsp),%rax
   0x0000000000401078 <+88>:    xor    %fs:0x28,%rax
   0x0000000000401081 <+97>:    je     0x401088 <phase_4+104>
   0x0000000000401083 <+99>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401088 <+104>:   add    $0x18,%rsp
   0x000000000040108c <+108>:   ret
End of assembler dump.
(gdb) 
```

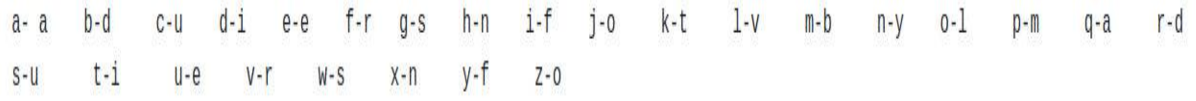
- Here in this code you can check that there is a cmp $0x2,%eax
- Here I took the second input as 2 when I took 2 as a first input then I got the wrong answer so I took it as a second input.
- Then bu using n i command I went to the lastly modified eax value and I checked what is there in an eax by using I r I checked info of registers and found the value as 40.
- Then by keeping 40 2 phase 4 is diffused.

Phase5:



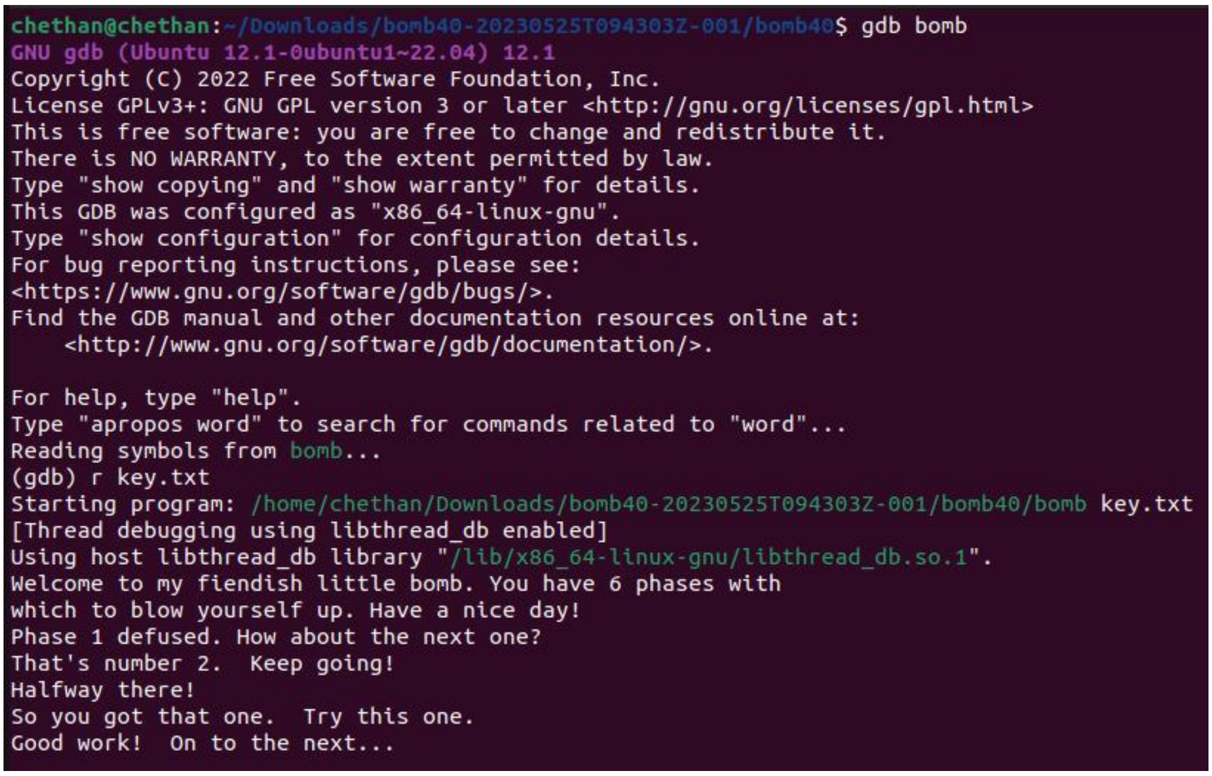
```
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x000000000040108d <+0>:     push   %rbx
   0x000000000040108e <+1>:     sub    $0x10,%rsp
   0x0000000000401092 <+5>:     mov    %rdi,%rbx
   0x0000000000401095 <+8>:     mov    %fs:0x28,%rax
   0x000000000040109e <+17>:    mov    %rax,0x8(%rsp)
   0x00000000004010a3 <+22>:    xor    %eax,%eax
   0x00000000004010a5 <+24>:    call   0x401342 <string_length>
   0x00000000004010aa <+29>:    cmp    $0x6,%eax
   0x00000000004010ad <+32>:    je     0x4010b4 <phase_5+39>
   0x00000000004010af <+34>:    call   0x40145f <explode_bomb>
   0x00000000004010b4 <+39>:    mov    $0x0,%eax
   0x00000000004010b9 <+44>:    movzbl (%rbx,%rax,1),%edx
   0x00000000004010bd <+48>:    and    $0xf,%edx
   0x00000000004010c0 <+51>:    movzbl 0x402470(%rdx),%edx
   0x00000000004010c7 <+58>:    mov    %dl,(%rsp,%rax,1)
   0x00000000004010ca <+61>:    add    $0x1,%rax
   0x00000000004010ce <+65>:    cmp    $0x6,%rax
   0x00000000004010d2 <+69>:    jne    0x4010b9 <phase_5+44>
   0x00000000004010d4 <+71>:    movb   $0x0,0x6(%rsp)
   0x00000000004010d9 <+76>:    mov    $0x402426,%esi
   0x00000000004010de <+81>:    mov    %rsp,%rdi
   0x00000000004010e1 <+84>:    call   0x401360 <strings_not_equal>
   0x00000000004010e6 <+89>:    test   %eax,%eax
   0x00000000004010e8 <+91>:    je     0x4010ef <phase_5+98>
   0x00000000004010ea <+93>:    call   0x40145f <explode_bomb>
   0x00000000004010ef <+98>:    mov    0x8(%rsp),%rax
   0x00000000004010f4 <+103>:   xor    %fs:0x28,%rax
   0x00000000004010fd <+112>:   je     0x401104 <phase_5+119>
   0x00000000004010ff <+114>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401104 <+119>:   add    $0x10,%rsp
   0x0000000000401108 <+123>:   pop    %rbx
   0x0000000000401109 <+124>:   ret
End of assembler dump.
(gdb) x/s 0x402426
0x402426:       "flames"
(gdb)
```

- Examine phase_5 assembly code.it can seen that input should be a string of length 6.
- Now give 6 input character alphabets in order.
- Then identify the mapping of each character.
- Mapping of each character occurs as follows.



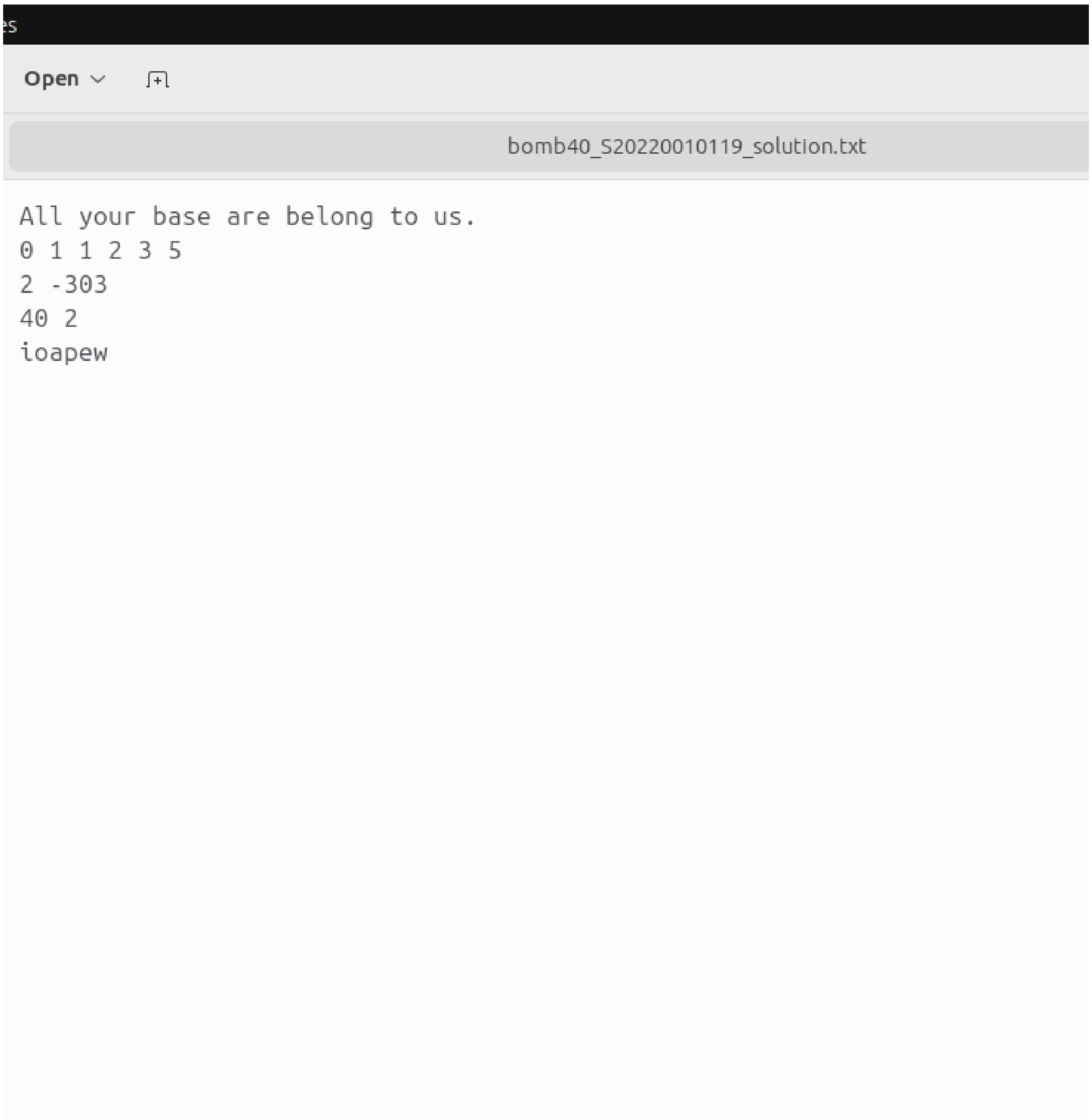
```
a- a   b-d   c-u   d-i   e-e   f-r   g-s   h-n   i-f   j-o   k-t   l-v   m-b   n-y   o-l   p-m   q-a   r-d
s-u    t-i   u-e   v-r   w-s   x-n   y-f   z-o
```

- At memory location 0x402426(use x/s) we have "flames".
- So as per above mapping the input string is "ioapew".



```
chethan@chethan:~/Downloads/bomb40-20230525T094303Z-001/bomb40$ gdb bomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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 bomb...
(gdb) r key.txt
Starting program: /home/chethan/Downloads/bomb40-20230525T094303Z-001/bomb40/bomb key.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

- By this phase_5 is diffused.



```
All your base are belong to us.
0 1 1 2 3 5
2 -303
40 2
ioapew
```

- It is solution.txt file