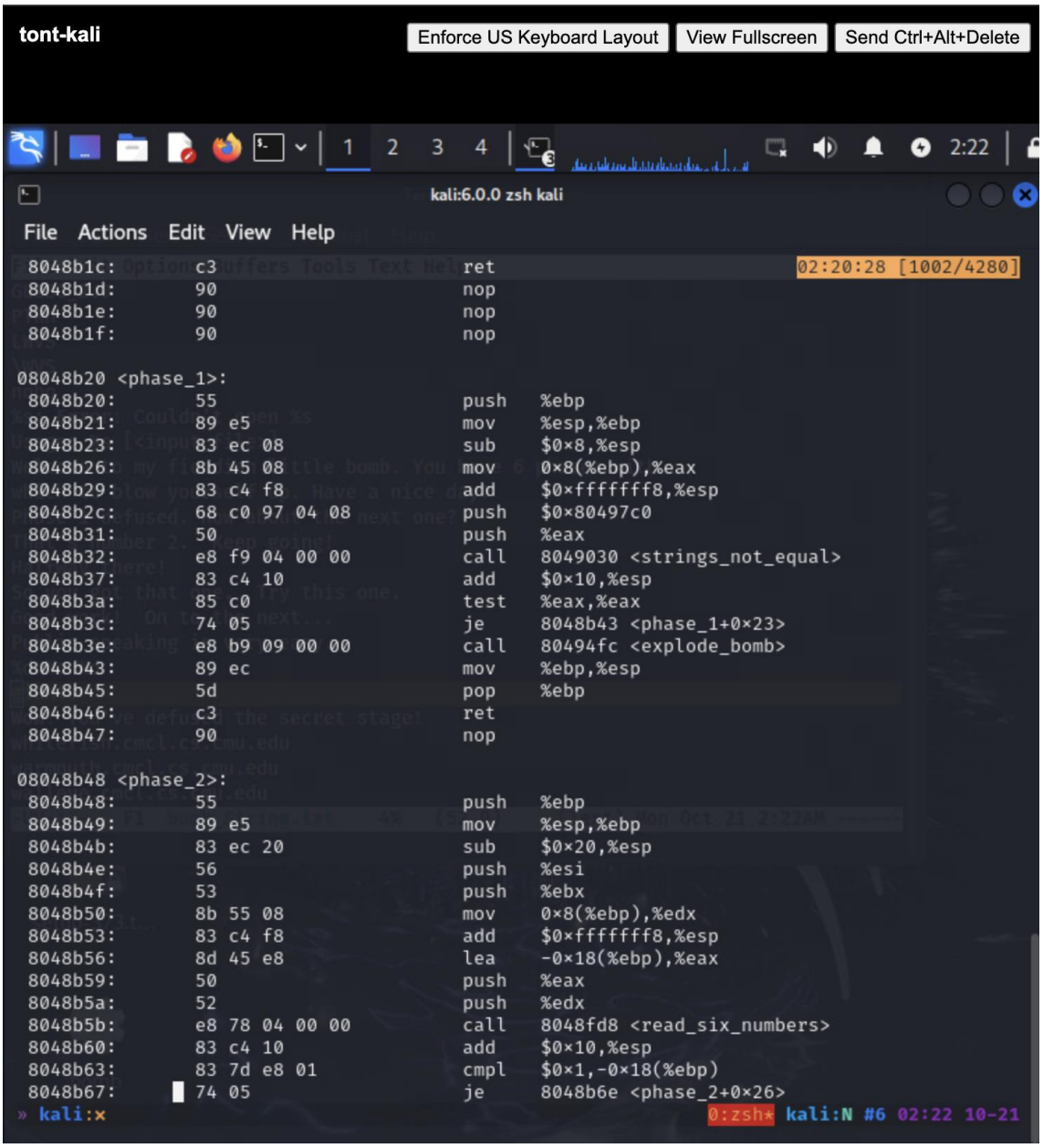
Tan Ton
CS 373

Lab 1 GDB Bomb!!!

Phase 1

Looking at Phase_1, there is a “call <String_not_equal>

I believe the first phase could be testing for a string that compare if they match together, where the address at 0x80497c0 will be the one got compare from eax (user input).



Then, I print out all the strings that might have in the first phase

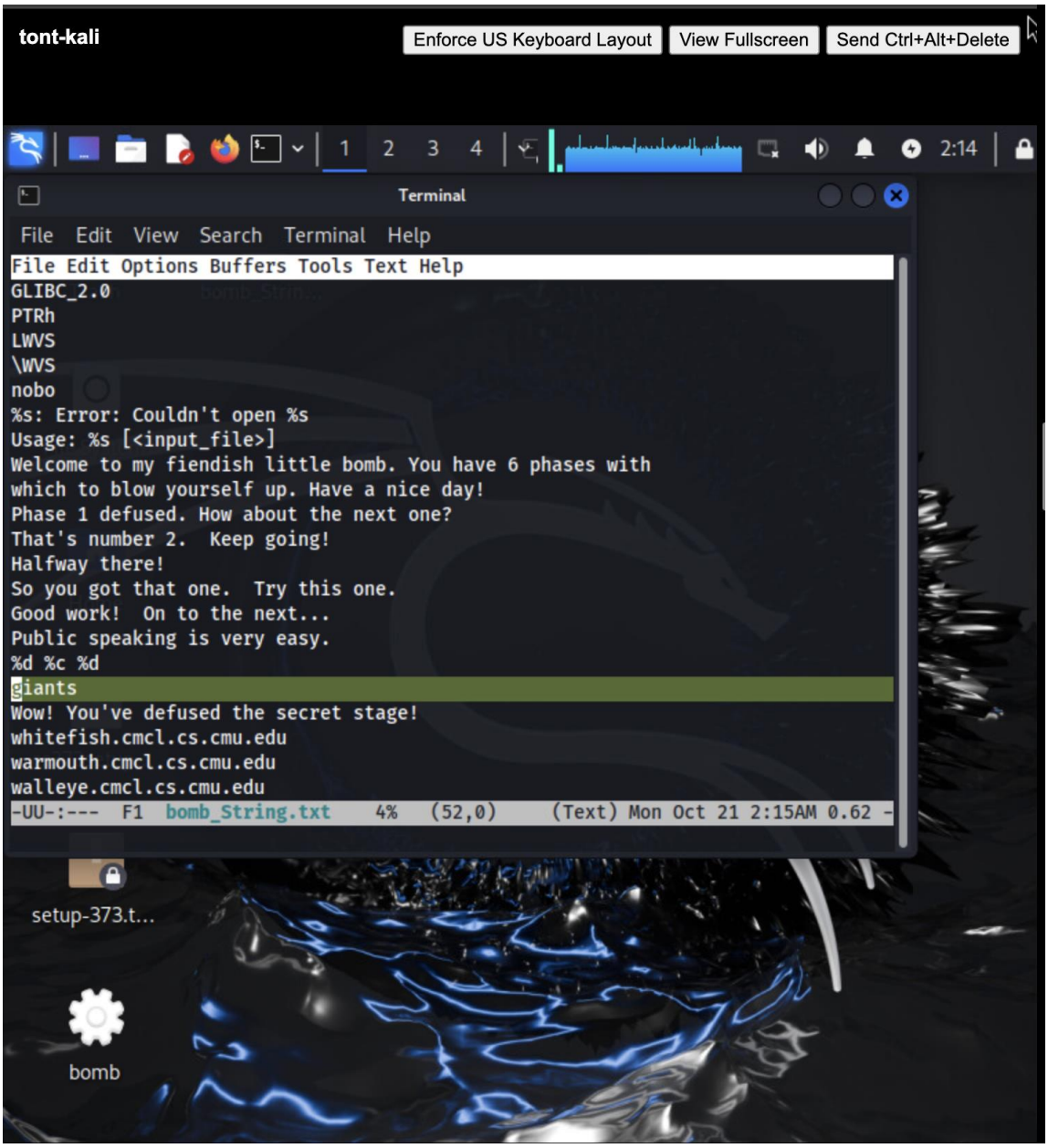
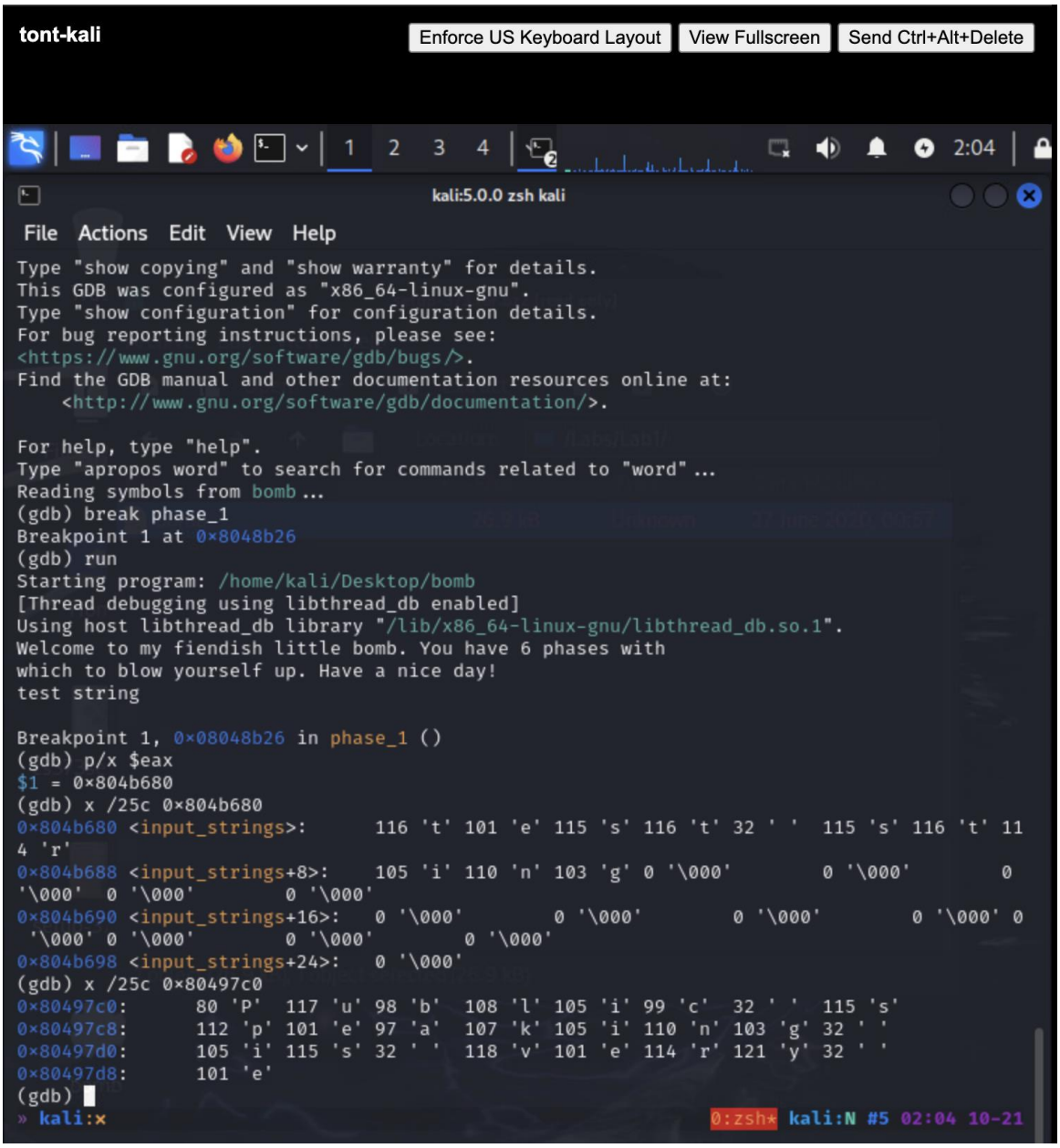


Figure out the only way I can test which string is the answer, i use gdb as a testing tool.

To see if $eax is actually storing user input, I use “test string” as input.

I then use p/x $eax as a way to find the memory address where it stored and check to see if the “test string” stored here.

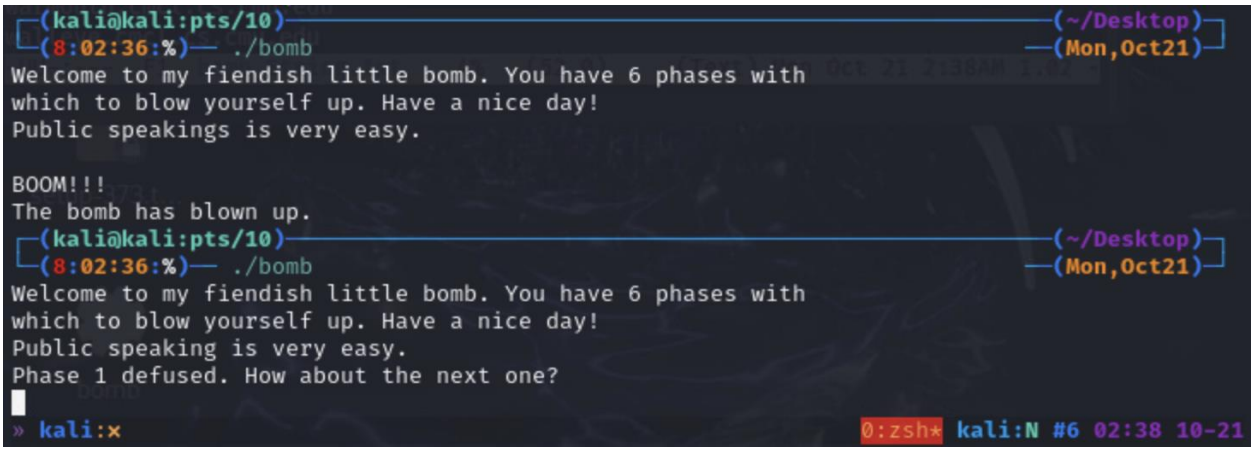
Figure out that I was right, I tested the actual strings that got compared with $eax, which is 0x080497c0.



The string appeared to be “public speaking is ve”

I then move back to figure 1 and see the string “Public speaking is very easy.”

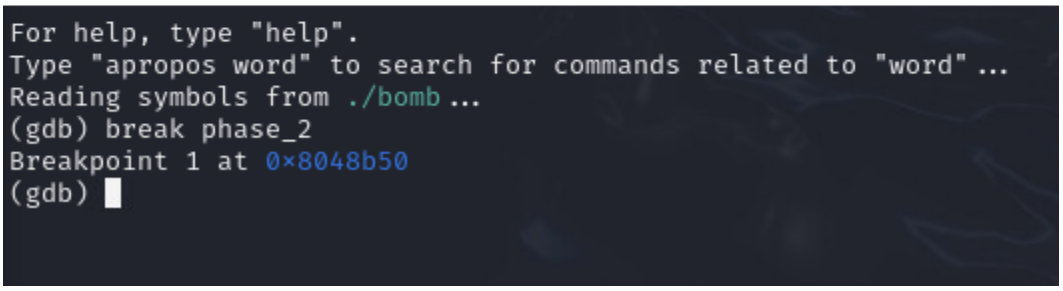
First phase defuses with the strings “Public speaking is very easy.”



Phase 2

Notice in the first phase, there is a call stack “explode bomb>, which makes me figure out that anything happening before the “explode bomb” could be a clue.

<read_six_numbers>, which mean the phase 2 defuse bomb could be related to number



I break the phase 2 as the same way with phase 1 and use the same method as phase 1 to see where numbers got stored at.

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bomb...
(gdb) break phase_2
Breakpoint 1 at 0x8048b50
(gdb) run
Starting program: /home/kali/Desktop/bomb
[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!
Public speaking is very easy.
Phase 1 defused. How about the next one?
12345

Breakpoint 1, 0x08048b50 in phase_2 ()
(gdb)
» kali:x
```

Notice there is a imul, which is to multiply current number with next index. And since 1 has to be accepted or the bomb will explode.

```
Breakpoint 1, 0x08048b50 in phase_2 ()
(gdb) dises
Undefined command: "dises".  Try "help".
(gdb) disas
Dump of assembler code for function phase_2:
   0x08048b48 <+0>:     push   %ebp
   0x08048b49 <+1>:     mov    %esp,%ebp
   0x08048b4b <+3>:     sub    $0x20,%esp
   0x08048b4e <+6>:     push   %esi
   0x08048b4f <+7>:     push   %ebx
=> 0x08048b50 <+8>:     mov    0x8(%ebp),%edx
   0x08048b53 <+11>:    add    $0xfffffff8,%esp
   0x08048b56 <+14>:    lea    -0x18(%ebp),%eax
   0x08048b59 <+17>:    push   %eax
   0x08048b5a <+18>:    push   %edx
   0x08048b5b <+19>:    call   0x8048fd8 <read_six_numbers>
   0x08048b60 <+24>:    add    $0x10,%esp
   0x08048b63 <+27>:    cmpl   $0x1,-0x18(%ebp)
   0x08048b67 <+31>:    je     0x8048b6e <phase_2+38>
   0x08048b69 <+33>:    call   0x80494fc <explode_bomb>
   0x08048b6e <+38>:    mov    $0x1,%ebx
   0x08048b73 <+43>:    lea    -0x18(%ebp),%esi
   0x08048b76 <+46>:    lea    0x1(%ebx),%eax
   0x08048b79 <+49>:    imul   -0x4(%esi,%ebx,4),%eax
   0x08048b7e <+54>:    cmp    %eax,(%esi,%ebx,4)
   0x08048b81 <+57>:    je     0x8048b88 <phase_2+64>
   0x08048b83 <+59>:    call   0x80494fc <explode_bomb>
   0x08048b88 <+64>:    inc    %ebx
   0x08048b89 <+65>:    cmp    $0x5,%ebx
   0x08048b8c <+68>:    jle    0x8048b76 <phase_2+46>
   0x08048b8e <+70>:    lea    -0x28(%ebp),%esp
   0x08048b91 <+73>:    pop    %ebx
   0x08048b92 <+74>:    pop    %esi
   0x08048b93 <+75>:    mov    %ebp,%esp
   0x08048b95 <+77>:    pop    %ebp
   0x08048b96 <+78>:    ret
End of assembler dump.
```

I use :

1

1 * 2 = 2

2 * 3 = 6

6 * 4 = 24

24 * 5 = 120

120 * 6 = 720

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
by Tan Ton
```

Phase 3

```
08048b98 <phase_3>:
 8048b98:       55                      push   %ebp
 8048b99:       89 e5                   mov    %esp,%ebp
 8048b9b:       83 ec 14                sub    $0x14,%esp
 8048b9e:       53                      push   %ebx
 8048b9f:       8b 55 08                mov    0x8(%ebp),%edx
 8048ba2:       83 c4 f4                add    $0xfffffff4,%esp
 8048ba5:       8d 45 fc                lea    -0x4(%ebp),%eax
 8048ba8:       50                      push   %eax
 8048ba9:       8d 45 fb                lea    -0x5(%ebp),%eax
 8048bac:       50                      push   %eax
 8048bad:       8d 45 f4                lea    -0xc(%ebp),%eax
 8048bb0:       50                      push   %eax
 8048bb1:       68 de 97 04 08          push   $0x80497de
 8048bb6:       52                      push   %edx
 8048bb7:       e8 a4 fc ff ff          call   8048860 <sscanf@plt>
 8048bbc:       83 c4 20                add    $0x20,%esp
 8048bbf:       83 f8 02                cmp    $0x2,%eax
 8048bc2:       7f 05                   jg     8048bc9 <phase_3+0x31>
 8048bc4:       e8 33 09 00 00          call   80494fc <explode_bomb>
 8048bc9:       83 7d f4 07             cmpl   $0x7,-0xc(%ebp)
 8048bcd:       0f 87 b5 00 00 00       ja     8048c88 <phase_3+0xf0>
 8048bd3:       8b 45 f4                mov    -0xc(%ebp),%eax
 8048bd6:       ff 24 85 e8 97 04 08    jmp    *0x80497e8(,%eax,4)
 8048bdd:       8d 76 00                lea    0x0(%esi),%esi
 8048be0:       b3 71                   mov    $0x71,%bl
 8048be2:       81 7d fc 09 03 00 00    cmpl   $0x309,-0x4(%ebp)
 8048be9:       0f 84 a0 00 00 00       je     8048c8f <phase_3+0xf7>
 8048bef:       e8 08 09 00 00          call   80494fc <explode_bomb>
 8048bf4:       e9 96 00 00 00          jmp    8048c8f <phase_3+0xf7>
 8048bf9:       8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi
 8048c00:       b3 62                   mov    $0x62,%bl
 8048c02:       81 7d fc d6 00 00 00    cmpl   $0xd6,-0x4(%ebp)
 8048c09:       0f 84 80 00 00 00       je     8048c8f <phase_3+0xf7>
 8048c0f:       e8 e8 08 00 00          call   80494fc <explode_bomb>
 8048c14:       eb 79                   jmp    8048c8f <phase_3+0xf7>
 8048c16:       b3 62                   mov    $0x62,%bl
 8048c18:       81 7d fc f3 02 00 00    cmpl   $0x2f3,-0x4(%ebp)
 8048c1f:       74 6e                   je     8048c8f <phase_3+0xf7>
 8048c21:       e8 d6 08 00 00          call   80494fc <explode_bomb>
 8048c26:       eb 67                   jmp    8048c8f <phase_3+0xf7>
 8048c28:       b3 6b                   mov    $0x6b,%bl
 8048c2a:       81 7d fc fb 00 00 00    cmpl   $0xfb,-0x4(%ebp)
 8048c31:       74 5c                   je     8048c8f <phase_3+0xf7>
 8048c33:       e8 c4 08 00 00          call   80494fc <explode_bomb>
 8048c38:       eb 55                   jmp    8048c8f <phase_3+0xf7>
 8048c3a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi
 8048c40:       b3 6f                   mov    $0x6f,%bl
 8048c42:       81 7d fc a0 00 00 00    cmpl   $0xa0,-0x4(%ebp)
 8048c49:       74 44                   je     8048c8f <phase_3+0xf7>
 8048c4b:       e8 ac 08 00 00          call   80494fc <explode_bomb>
 8048c50:       eb 3d                   jmp    8048c8f <phase_3+0xf7>
 8048c52:       b3 74                   mov    $0x74,%bl
 8048c54:       81 7d fc ca 01 00 00    cmpl   $0x1ca,-0x4(%ebp)
 8048c5b:       74 32                   je     8048c8f <phase_3+0xf7>
 8048c5d:       e8 9a 08 00 00          call   80494fc <explode_bomb>
 8048c62:       eb 2b                   jmp    8048c8f <phase_3+0xf7>
 8048c64:       b3 76                   mov    $0x76,%bl
 8048c66:       81 7d fc 0c 03 00 00    cmpl   $0x30c,-0x4(%ebp)
 8048c6d:       74 20                   je     8048c8f <phase_3+0xf7>
 8048c6f:       e8 88 08 00 00          call   80494fc <explode_bomb>
 8048c74:       eb 19                   jmp    8048c8f <phase_3+0xf7>
 8048c76:       b3 62                   mov    $0x62,%bl
 8048c78:       81 7d fc 0c 02 00 00    cmpl   $0x20c,-0x4(%ebp)
 8048c7f:       74 0e                   je     8048c8f <phase_3+0xf7>
```

Phase 3 was a bit more tricky since there were no hint of what the input would be. Therefor I have to inspect the code carefully

Using the same method with step 2, I see the break point for phase 3 is at address 0x08048b9f, I then first notice the scanf in the code. And since the address of 0x80497de were push to eax, I inspect the address itself and see the require format of the phase.

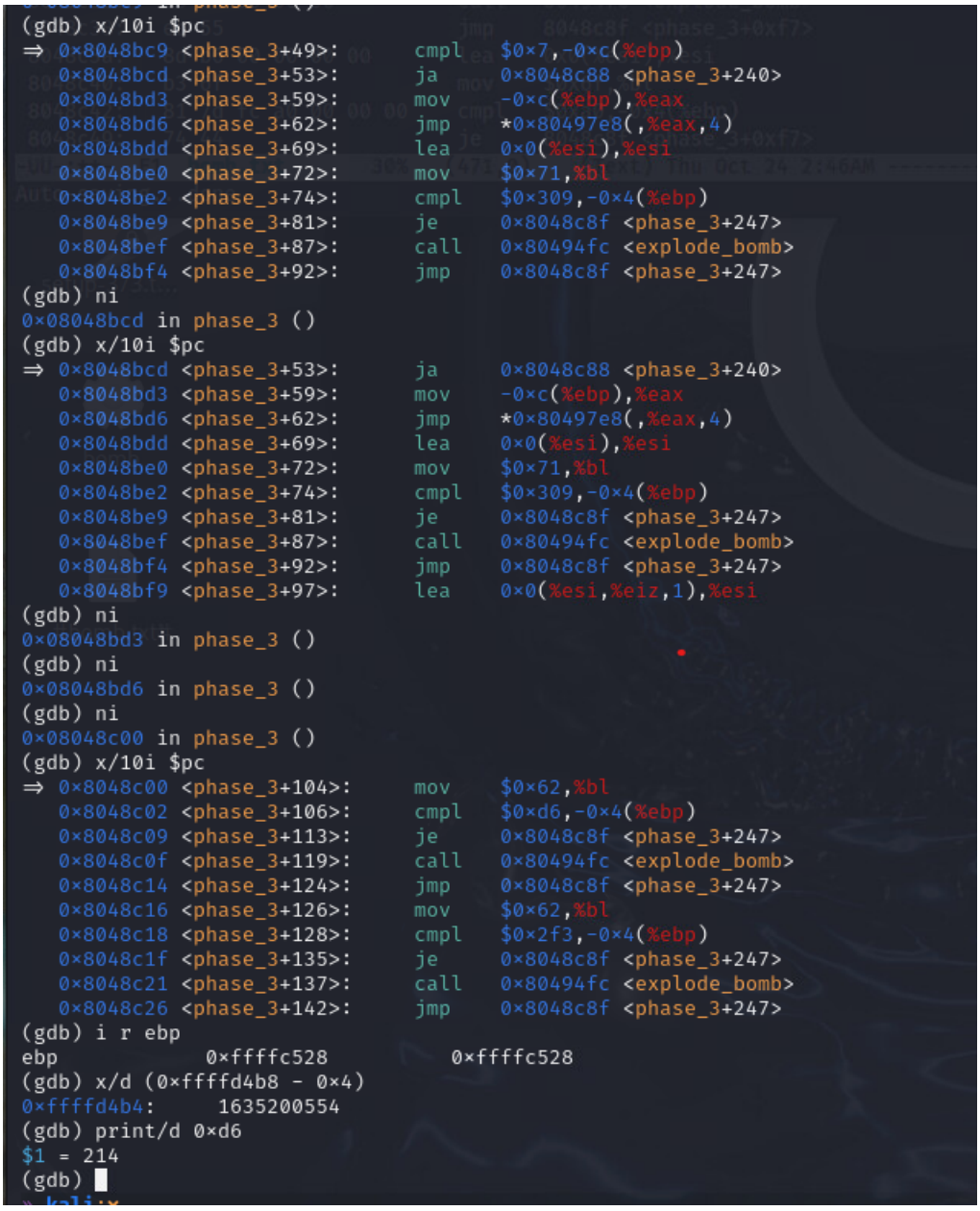
The required format shown at below address, which should be integer, character, integer

```
(gdb) x/s 0x80497de
0x80497de:      "%d %c %d"
```

```
0x08048bac in phase_3 ()
(gdb) x/10i $pc
=> 0x8048bac <phase_3+20>:      push   %eax
   0x8048bad <phase_3+21>:      lea    -0xc(%ebp),%eax
   0x8048bb0 <phase_3+24>:      push   %eax
   0x8048bb1 <phase_3+25>:      push   $0x80497de
   0x8048bb6 <phase_3+30>:      push   %edx
   0x8048bb7 <phase_3+31>:      call   0x8048860 <sscanf@plt>
   0x8048bbc <phase_3+36>:      add    $0x20,%esp
   0x8048bbf <phase_3+39>:      cmp    $0x2,%eax
   0x8048bc2 <phase_3+42>:      jg     0x8048bc9 <phase_3+49>
   0x8048bc4 <phase_3+44>:      call   0x80494fc <explode_bomb>
(gdb) ni
0x08048bad in phase_3 ()
(gdb) x/10i $pc
=> 0x8048bad <phase_3+21>:      lea    -0xc(%ebp),%eax
   0x8048bb0 <phase_3+24>:      push   %eax
   0x8048bb1 <phase_3+25>:      push   $0x80497de
   0x8048bb6 <phase_3+30>:      push   %edx
   0x8048bb7 <phase_3+31>:      call   0x8048860 <sscanf@plt>
   0x8048bbc <phase_3+36>:      add    $0x20,%esp
   0x8048bbf <phase_3+39>:      cmp    $0x2,%eax
   0x8048bc2 <phase_3+42>:      jg     0x8048bc9 <phase_3+49>
   0x8048bc4 <phase_3+44>:      call   0x80494fc <explode_bomb>
   0x8048bc9 <phase_3+49>:      cmpl   $0x7,-0xc(%ebp)
(gdb) ni
0x08048bb0 in phase_3 ()
(gdb) x/10i $pc
=> 0x8048bb0 <phase_3+24>:      push   %eax
   0x8048bb1 <phase_3+25>:      push   $0x80497de
   0x8048bb6 <phase_3+30>:      push   %edx
   0x8048bb7 <phase_3+31>:      call   0x8048860 <sscanf@plt>
   0x8048bbc <phase_3+36>:      add    $0x20,%esp
   0x8048bbf <phase_3+39>:      cmp    $0x2,%eax
   0x8048bc2 <phase_3+42>:      jg     0x8048bc9 <phase_3+49>
   0x8048bc4 <phase_3+44>:      call   0x80494fc <explode_bomb>
   0x8048bc9 <phase_3+49>:      cmpl   $0x7,-0xc(%ebp)
   0x8048bcd <phase_3+53>:      ja     0x8048c88 <phase_3+240>
(gdb) ni
0x08048bb1 in phase_3 ()
(gdb) x/10i $pc
=> 0x8048bb1 <phase_3+25>:      push   $0x80497de
   0x8048bb6 <phase_3+30>:      push   %edx
   0x8048bb7 <phase_3+31>:      call   0x8048860 <sscanf@plt>
   0x8048bbc <phase_3+36>:      add    $0x20,%esp
   0x8048bbf <phase_3+39>:      cmp    $0x2,%eax
   0x8048bc2 <phase_3+42>:      jg     0x8048bc9 <phase_3+49>
   0x8048bc4 <phase_3+44>:      call   0x80494fc <explode_bomb>
   0x8048bc9 <phase_3+49>:      cmpl   $0x7,-0xc(%ebp)
   0x8048bcd <phase_3+53>:      ja     0x8048c88 <phase_3+240>
   0x8048bd3 <phase_3+59>:      mov    -0xc(%ebp),%eax
(gdb) ni
0x08048bb6 in phase_3 ()
(gdb) ni
0x08048bb7 in phase_3 ()
(gdb) x/10i $pc
=> 0x8048bb7 <phase_3+31>:      call   0x8048860 <sscanf@plt>
   0x8048bbc <phase_3+36>:      add    $0x20,%esp
   0x8048bbf <phase_3+39>:      cmp    $0x2,%eax
   0x8048bc2 <phase_3+42>:      jg     0x8048bc9 <phase_3+49>
   0x8048bc4 <phase_3+44>:      call   0x80494fc <explode_bomb>
   0x8048bc9 <phase_3+49>:      cmpl   $0x7,-0xc(%ebp)
   0x8048bcd <phase_3+53>:      ja     0x8048c88 <phase_3+240>
   0x8048bd3 <phase_3+59>:      mov    -0xc(%ebp),%eax
   0x8048bd6 <phase_3+62>:      jmp    *0x80497e8(,%eax,4)
   0x8048bdd <phase_3+69>:      lea    0x0(%esi),%esi
```

I then retype my answer and did a break point at phase_3, using ni to keep looking for something suspicious, which is a jump that will jump to 0x8048c8f.

Next, went straight to the jump and see that there is a cmpl with an integer that got stored at –0x4. I then cbeck the value of 0xd6, which is the address that got compare to.



The address then shown 214. Which means the last digit could be 214.

Since I already got 1 and 214 as my integer, I only need to discover the last cmp, which could be a character that i am missing.

The last compare was a character and expected a ‘b’. Therefor, the answer is “1 b 214”