

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).

```
tont-kali Enforce US Keyboard Layout View Fullscreen Send Ctrl+Alt+Delete

File Actions Edit View Help

8048b1c: Option c3 buffers Tools Text Hel ret 02:20:28 [1002/4280]
8048b1d: 90 nop
8048b1e: 90 nop
8048b1f: 90 nop

08048b20 <phase_1>:
8048b20: 55 push %ebp
8048b21: 89 e5 mov %esp,%ebp
8048b23: 83 ec 08 sub $0x8,%esp
8048b26: 8b 45 08 mov 0x8(%ebp),%eax
8048b29: 83 c4 f8 add $0xffffffff8,%esp
8048b2c: 68 c0 97 04 08 push $0x80497c0
8048b31: 50 push %eax
8048b32: e8 f9 04 00 00 call 8049030 <strings_not_equal>
8048b37: 83 c4 10 add $0x10,%esp
8048b3a: 85 c0 test %eax,%eax
8048b3c: 74 05 je 8048b43 <phase_1+0x23>
8048b3e: e8 b9 09 00 00 call 80494fc <explode_bomb>
8048b43: 89 ec mov %ebp,%esp
8048b45: 5d pop %ebp
8048b46: c3 ret
8048b47: 90 nop

08048b48 <phase_2>:
8048b48: 55 push %ebp
8048b49: 89 e5 mov %esp,%ebp
8048b4b: 83 ec 20 sub $0x20,%esp
8048b4e: 56 push %esi
8048b4f: 53 push %ebx
8048b50: 8b 55 08 mov 0x8(%ebp),%edx
8048b53: 83 c4 f8 add $0xffffffff8,%esp
8048b56: 8d 45 e8 lea -0x18(%ebp),%eax
8048b59: 50 push %eax
8048b5a: 52 push %edx
8048b5b: e8 78 04 00 00 call 8048fd8 <read_six_numbers>
8048b60: 83 c4 10 add $0x10,%esp
8048b63: 83 7d e8 01 cmpl $0x1,-0x18(%ebp)
8048b67: 74 05 je 8048b6e <phase_2+0x26>

» kali:~ 0:zsh* kali:N #6 02:22 10-21
```

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

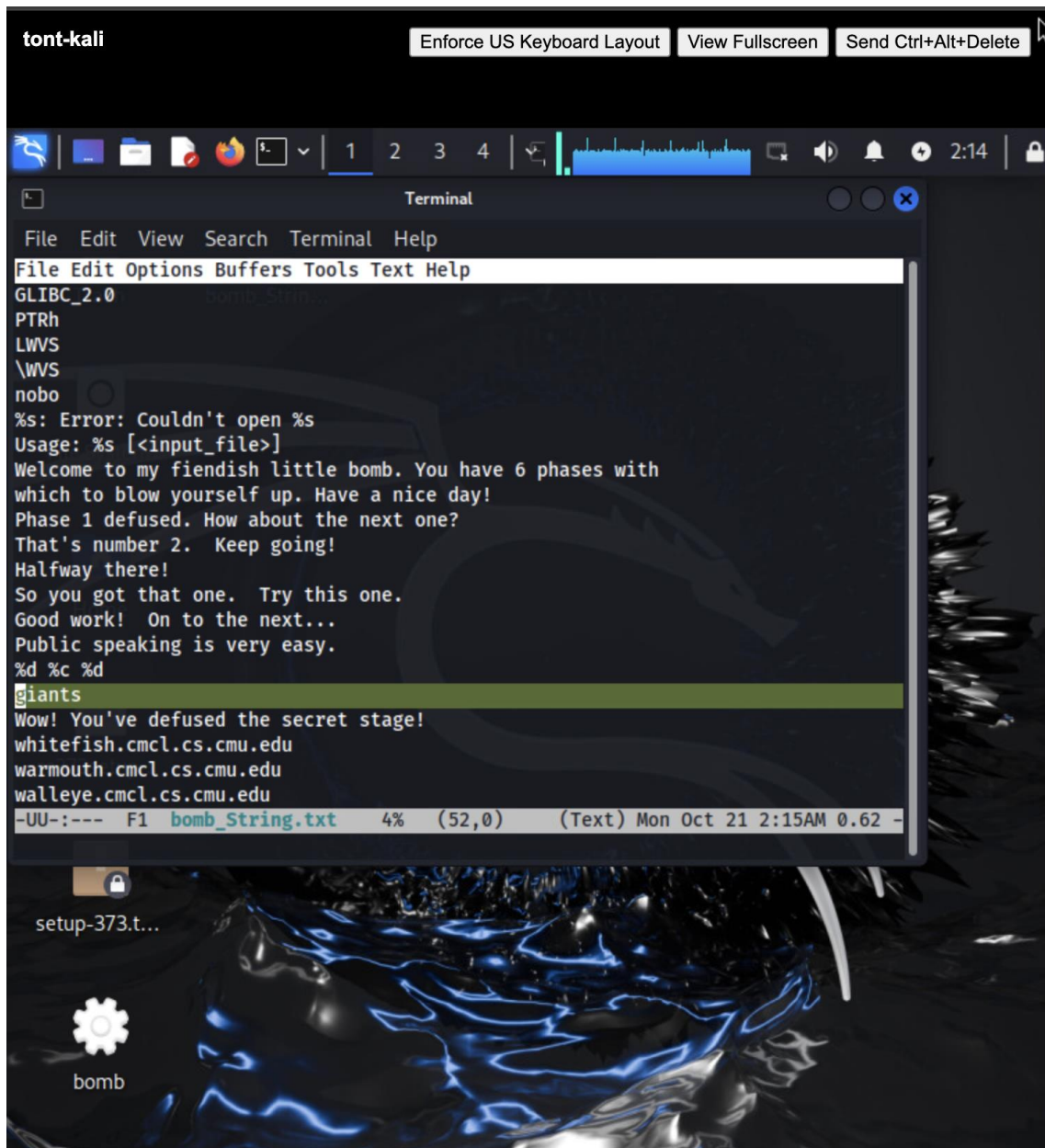


Figure out the only way I can test which string is the answer, i use gdb as a testing tool.

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.

```
tont-kali Enforce US Keyboard Layout View Fullscreen Send Ctrl+Alt+Delete

kali:5.0.0 zsh kali

File Actions Edit View Help

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) break phase_1
Breakpoint 1 at 0x8048b26
(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!
test string

Breakpoint 1, 0x8048b26 in phase_1 ()
(gdb) p/x $eax
$1 = 0x804b680
(gdb) x /25c 0x804b680
0x804b680 <input_strings>: 116 't' 101 'e' 115 's' 116 't' 32 ' ' 115 's' 116 't' 11
4 'r'
0x804b688 <input_strings+8>: 105 'i' 110 'n' 103 'g' 0 '\000' 0 '\000' 0
'\000' 0 '\000' 0 '\000'
0x804b690 <input_strings+16>: 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000' 0
'\000' 0 '\000' 0 '\000'
0x804b698 <input_strings+24>: 0 '\000'
(gdb) x /25c 0x80497c0
0x80497c0: 80 'P' 117 'u' 98 'b' 108 'l' 105 'i' 99 'c' 32 ' ' 115 's'
0x80497c8: 112 'p' 101 'e' 97 'a' 107 'k' 105 'i' 110 'n' 103 'g' 32 ' '
0x80497d0: 105 'i' 115 's' 32 ' ' 118 'v' 101 'e' 114 'r' 121 'y' 32 ' '
0x80497d8: 101 'e'
(gdb)
» kali:x 0:zsh* kali:N #5 02:04 10-21
```

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.”

```
(kali@kali:pts/10) [~/Desktop]
(8:02:36:~) ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speakings is very easy.

BOOM!!!
The bomb has blown up.
(kali@kali:pts/10) [~/Desktop]
(8:02:36:~) ./bomb
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?
█
» kali:x 0:zsh* kali:N #6 02:38 10-21
```

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

```
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) █
```

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:~
```

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) disas
Undefined command: "disas". 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     $0xffffffff,%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

```

0048b98: <phase_3>: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0048b98: 55          push    %ebp
0048b99: 89 e5      mov     %esp,%ebp
0048b9b: 83 ec 14   sub     $0x14,%esp
0048b9e: 53         push    %ebx
0048b9f: 8b 55 08   mov     0x8(%ebp),%edx
0048ba2: 83 c4 f4   add     $0xffffffff,%esp
0048ba5: 8d 45 fc   lea     -0x4(%ebp),%eax
0048ba8: 50         push    %eax
0048ba9: 8d 45 fb   lea     -0x5(%ebp),%eax
0048bac: 50         push    %eax
0048bad: 8d 45 f4   lea     -0xc(%ebp),%eax
0048bb0: 50         push    %eax
0048bb1: 68 de 97 04 08  mov     $0x80497de,%eax
0048bb6: 52         push    %edx
0048bb7: e8 a4 fc ff ff  call    8048860 <sscanf@plt>
0048bbc: 83 c4 20   add     $0x20,%esp
0048bbf: 83 f8 02   cmp     $0x2,%eax
0048bc2: 7f 05     jg      8048bc9 <phase_3+0x31>
0048bc4: e8 33 09 00 00  call    80494fc <explode_bomb>
0048bc9: 83 7d f4 07   cmpl    $0x7,-0xc(%ebp)
0048bcd: 0f 87 b5 00 00 00  ja      8048c88 <phase_3+0xf0>
0048bd3: 8b 45 f4   mov     -0xc(%ebp),%eax
0048bd6: ff 24 85 e8 97 04 08  jmp     *0x80497e8(,%eax,4)
0048bdd: 8d 76 00   lea     0x0(%esi),%esi
0048be0: b3 71     mov     $0x71,%bl
0048be2: 81 7d fc 09 03 00 00  cmpl    $0x309,-0x4(%ebp)
0048be9: 0f 84 a0 00 00 00  je      8048c8f <phase_3+0xf7>
0048bef: e8 08 09 00 00  call    80494fc <explode_bomb>
0048bf4: e9 96 00 00 00  jmp     8048c8f <phase_3+0xf7>
0048bf9: 8d b4 26 00 00 00 00  lea     0x0(%esi,%eiz,1),%esi
0048c00: b3 62     mov     $0x62,%bl
0048c02: 81 7d fc d6 00 00 00  cmpl    $0xd6,-0x4(%ebp)
0048c09: 0f 84 80 00 00 00  je      8048c8f <phase_3+0xf7>
0048c0f: e8 e8 08 00 00  call    80494fc <explode_bomb>
0048c14: eb 79     jmp     8048c8f <phase_3+0xf7>
0048c16: b3 62     mov     $0x62,%bl
0048c18: 81 7d fc f3 02 00 00  cmpl    $0x2f3,-0x4(%ebp)
0048c1f: 74 6e     je      8048c8f <phase_3+0xf7>
0048c21: e8 d6 08 00 00  call    80494fc <explode_bomb>
0048c26: eb 67     jmp     8048c8f <phase_3+0xf7>
0048c28: b3 6b     mov     $0x6b,%bl
0048c2a: 81 7d fc fb 00 00 00  cmpl    $0xfb,-0x4(%ebp)
0048c31: 74 5c     je      8048c8f <phase_3+0xf7>
0048c33: e8 c4 08 00 00  call    80494fc <explode_bomb>
0048c38: eb 55     jmp     8048c8f <phase_3+0xf7>
0048c3a: 8d b6 00 00 00 00  lea     0x0(%esi),%esi
0048c40: b3 6f     mov     $0x6f,%bl
0048c42: 81 7d fc a0 00 00 00  cmpl    $0xa0,-0x4(%ebp)
0048c49: 74 44     je      8048c8f <phase_3+0xf7>
0048c4b: e8 ac 08 00 00  call    80494fc <explode_bomb>
0048c50: eb 3d     jmp     8048c8f <phase_3+0xf7>
0048c52: b3 74     mov     $0x74,%bl
0048c54: 81 7d fc ca 01 00 00  cmpl    $0x1ca,-0x4(%ebp)
0048c5b: 74 32     je      8048c8f <phase_3+0xf7>
0048c5d: e8 9a 08 00 00  call    80494fc <explode_bomb>
0048c62: eb 2b     jmp     8048c8f <phase_3+0xf7>
0048c64: b3 76     mov     $0x76,%bl
0048c66: 81 7d fc 0c 03 00 00  cmpl    $0x30c,-0x4(%ebp)
0048c6d: 74 20     je      8048c8f <phase_3+0xf7>
0048c6f: e8 88 08 00 00  call    80494fc <explode_bomb>
0048c74: eb 19     jmp     8048c8f <phase_3+0xf7>
0048c76: b3 62     mov     $0x62,%bl
0048c78: 81 7d fc 0c 02 00 00  cmpl    $0x20c,-0x4(%ebp)
0048c7f: 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. Therefore 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.

```
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2. Keep going!
let me see

Breakpoint 1, 0x08048b9f in phase_3 ()
(gdb) disas
Dump of assembler code for function phase_3:
0x08048b98 <+0>: push %ebp
0x08048b99 <+1>: mov %esp,%ebp
0x08048b9b <+3>: sub $0x14,%esp
0x08048b9e <+6>: push %ebx
⇒ 0x08048b9f <+7>: mov 0x8(%ebp),%edx
0x08048ba2 <+10>: add $0xffffffff4,%esp
0x08048ba5 <+13>: lea -0x4(%ebp),%eax
0x08048ba8 <+16>: push %eax
0x08048ba9 <+17>: lea -0x5(%ebp),%eax
0x08048bac <+20>: push %eax
0x08048bad <+21>: lea -0xc(%ebp),%eax
0x08048bb0 <+24>: push %eax
0x08048bb1 <+25>: push $0x80497de
0x08048bb6 <+30>: push %edx
0x08048bb7 <+31>: call 0x8048860 <sscanf@plt>
0x08048bbc <+36>: add $0x20,%esp
0x08048bbf <+39>: cmp $0x2,%eax
0x08048bc2 <+42>: jg 0x8048bc9 <phase_3+49>
0x08048bc4 <+44>: call 0x80494fc <explode_bomb>
0x08048bc9 <+49>: cmpl $0x7,-0xc(%ebp)
0x08048bcd <+53>: ja 0x8048c88 <phase_3+240>
0x08048bd3 <+59>: mov -0xc(%ebp),%eax
0x08048bd6 <+62>: jmp *0x80497e8(,%eax,4)
0x08048bdd <+69>: lea 0x0(%esi),%esi
0x08048be0 <+72>: mov $0x71,%bl
0x08048be2 <+74>: cmpl $0x309,-0x4(%ebp)
0x08048be9 <+81>: je 0x8048c8f <phase_3+247>
0x08048bef <+87>: call 0x80494fc <explode_bomb>
```

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
=> 0x08048bac <phase_3+20>: push    %eax
0x08048bad <phase_3+21>: lea     -0xc(%ebp),%eax
0x08048bb0 <phase_3+24>: push    %eax
0x08048bb1 <phase_3+25>: push    $0x80497de
0x08048bb6 <phase_3+30>: push    %edx
0x08048bb7 <phase_3+31>: call    0x08048860 <sscanf@plt>
0x08048bbc <phase_3+36>: add     $0x20,%esp
0x08048bbf <phase_3+39>: cmp     $0x2,%eax
0x08048bc2 <phase_3+42>: jg      0x08048bc9 <phase_3+49>
0x08048bc4 <phase_3+44>: call    0x080494fc <explode_bomb>
0x08048bc9 <phase_3+49>: cmpl    $0x7,-0xc(%ebp)
0x08048bcd <phase_3+53>: ja      0x08048c88 <phase_3+240>
0x08048bd3 <phase_3+59>: mov     -0xc(%ebp),%eax
0x08048bdd <phase_3+69>: lea     0x0(%esi),%esi

(gdb) ni
0x08048bad in phase_3 ()
(gdb) x/10i $pc
=> 0x08048bad <phase_3+21>: lea     -0xc(%ebp),%eax
0x08048bb0 <phase_3+24>: push    %eax
0x08048bb1 <phase_3+25>: push    $0x80497de
0x08048bb6 <phase_3+30>: push    %edx
0x08048bb7 <phase_3+31>: call    0x08048860 <sscanf@plt>
0x08048bbc <phase_3+36>: add     $0x20,%esp
0x08048bbf <phase_3+39>: cmp     $0x2,%eax
0x08048bc2 <phase_3+42>: jg      0x08048bc9 <phase_3+49>
0x08048bc4 <phase_3+44>: call    0x080494fc <explode_bomb>
0x08048bc9 <phase_3+49>: cmpl    $0x7,-0xc(%ebp)
0x08048bcd <phase_3+53>: ja      0x08048c88 <phase_3+240>
0x08048bd3 <phase_3+59>: mov     -0xc(%ebp),%eax
0x08048bdd <phase_3+69>: lea     0x0(%esi),%esi

(gdb) ni
0x08048bb0 in phase_3 ()
(gdb) x/10i $pc
=> 0x08048bb0 <phase_3+24>: push    %eax
0x08048bb1 <phase_3+25>: push    $0x80497de
0x08048bb6 <phase_3+30>: push    %edx
0x08048bb7 <phase_3+31>: call    0x08048860 <sscanf@plt>
0x08048bbc <phase_3+36>: add     $0x20,%esp
0x08048bbf <phase_3+39>: cmp     $0x2,%eax
0x08048bc2 <phase_3+42>: jg      0x08048bc9 <phase_3+49>
0x08048bc4 <phase_3+44>: call    0x080494fc <explode_bomb>
0x08048bc9 <phase_3+49>: cmpl    $0x7,-0xc(%ebp)
0x08048bcd <phase_3+53>: ja      0x08048c88 <phase_3+240>
0x08048bd3 <phase_3+59>: mov     -0xc(%ebp),%eax
0x08048bdd <phase_3+69>: lea     0x0(%esi),%esi

(gdb) ni
0x08048bb1 in phase_3 ()
(gdb) x/10i $pc
=> 0x08048bb1 <phase_3+25>: push    $0x80497de
0x08048bb6 <phase_3+30>: push    %edx
0x08048bb7 <phase_3+31>: call    0x08048860 <sscanf@plt>
0x08048bbc <phase_3+36>: add     $0x20,%esp
0x08048bbf <phase_3+39>: cmp     $0x2,%eax
0x08048bc2 <phase_3+42>: jg      0x08048bc9 <phase_3+49>
0x08048bc4 <phase_3+44>: call    0x080494fc <explode_bomb>
0x08048bc9 <phase_3+49>: cmpl    $0x7,-0xc(%ebp)
0x08048bcd <phase_3+53>: ja      0x08048c88 <phase_3+240>
0x08048bd3 <phase_3+59>: mov     -0xc(%ebp),%eax
0x08048bdd <phase_3+69>: lea     0x0(%esi),%esi

(gdb) ni
0x08048bb6 in phase_3 ()
(gdb) ni
0x08048bb7 in phase_3 ()
(gdb) x/10i $pc
=> 0x08048bb7 <phase_3+31>: call    0x08048860 <sscanf@plt>
0x08048bbc <phase_3+36>: add     $0x20,%esp
0x08048bbf <phase_3+39>: cmp     $0x2,%eax
0x08048bc2 <phase_3+42>: jg      0x08048bc9 <phase_3+49>
0x08048bc4 <phase_3+44>: call    0x080494fc <explode_bomb>
0x08048bc9 <phase_3+49>: cmpl    $0x7,-0xc(%ebp)
0x08048bcd <phase_3+53>: ja      0x08048c88 <phase_3+240>
0x08048bd3 <phase_3+59>: mov     -0xc(%ebp),%eax
0x08048bdd <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.

```
(gdb) x/10i $pc
=> 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
0x8048be0 <phase_3+72>:   mov     $0x71,%bl
0x8048be2 <phase_3+74>:   cmpl    $0x309,-0x4(%ebp)
0x8048be9 <phase_3+81>:   je      0x8048c8f <phase_3+247>
0x8048bef <phase_3+87>:   call    0x80494fc <explode_bomb>
0x8048bf4 <phase_3+92>:   jmp     0x8048c8f <phase_3+247>
(gdb) ni
0x08048bcd in phase_3 ()
(gdb) x/10i $pc
=> 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
0x8048be0 <phase_3+72>:   mov     $0x71,%bl
0x8048be2 <phase_3+74>:   cmpl    $0x309,-0x4(%ebp)
0x8048be9 <phase_3+81>:   je      0x8048c8f <phase_3+247>
0x8048bef <phase_3+87>:   call    0x80494fc <explode_bomb>
0x8048bf4 <phase_3+92>:   jmp     0x8048c8f <phase_3+247>
0x8048bf9 <phase_3+97>:   lea     0x0(%esi,%eiz,1),%esi
(gdb) ni
0x08048bd3 in phase_3 ()
(gdb) ni
0x08048bd6 in phase_3 ()
(gdb) ni
0x08048c00 in phase_3 ()
(gdb) x/10i $pc
=> 0x8048c00 <phase_3+104>:  mov     $0x62,%bl
0x8048c02 <phase_3+106>:  cmpl    $0xd6,-0x4(%ebp)
0x8048c09 <phase_3+113>:  je      0x8048c8f <phase_3+247>
0x8048c0f <phase_3+119>:  call    0x80494fc <explode_bomb>
0x8048c14 <phase_3+124>:  jmp     0x8048c8f <phase_3+247>
0x8048c16 <phase_3+126>:  mov     $0x62,%bl
0x8048c18 <phase_3+128>:  cmpl    $0x2f3,-0x4(%ebp)
0x8048c1f <phase_3+135>:  je      0x8048c8f <phase_3+247>
0x8048c21 <phase_3+137>:  call    0x80494fc <explode_bomb>
0x8048c26 <phase_3+142>:  jmp     0x8048c8f <phase_3+247>
(gdb) i r ebp
ebp                0xffffc528      0xffffc528
(gdb) x/d (0xffffd4b8 - 0x4)
0xffffd4b4:        1635200554
(gdb) print/d 0xd6
$1 = 214
(gdb) █
```

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.

```

⇒ 0x8048c8f <phase_3+247>: cmp    -0x5(%ebp),%bl
0x8048c92 <phase_3+250>: je     0x8048c99 <phase_3+257>
0x8048c94 <phase_3+252>: call   0x80494fc <explode_bomb>
0x8048c99 <phase_3+257>: mov    -0x18(%ebp),%ebx
0x8048c9c <phase_3+260>: mov    %ebp,%esp
0x8048c9e <phase_3+262>: pop    %ebp
0x8048c9f <phase_3+263>: ret
0x8048ca0 <func4>: push   %ebp
0x8048ca1 <func4+1>: mov    %esp,%ebp
0x8048ca3 <func4+3>: sub    $0x10,%esp
(gdb) i r ebp
ebp                0xffffc528                0xffffc528
(gdb) x/c (0xffffd4b8 - 0x5)
0xffffd4b3:      58 ':'
(gdb) i r bl
bl                 0x62                    98
(gdb) print/c 0x62
$2 = 98 'b'
(gdb)

```

The last compare was a character and expected a 'b'. Therefore, the answer is "1 b 214"

```

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!
1 b 214
Halfway there!
tan ton

```