Student: Edward Lin

My Answers: (The process of each phase is listed below)

Phase_1: When I get angry, Mr. Bigglesworth gets upset.

Phase_2: 1 2 4 8 16 32

Phase_3: 6 u 332

Phase_4: 7 7

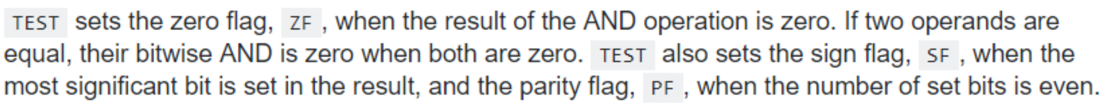
Phase_1:

Guess: Give me some clues!

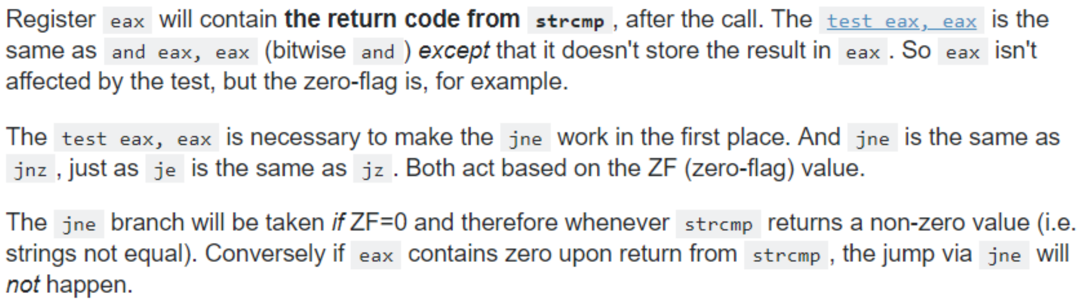
I get:

rdi value after ASCII Table	rsi value after ASCII Table
rdi+0x0 65766947: evig rdi+0x4 20656d20: _em_ rdi+0x8 656d6f73: emos rdi+0x10 00217365: !su Reasons: Because my machine is little endian, so it would be "give me some …" which is my inputs.	rsi+6e656857: nehw rsi+67204920: g_I_ rsi+61207465: a_te Reasons: Because of little endian, I got "when I get a…." and then I searched strings like [strings bomb-64]. I got "When I get angry, Mr. Bigglesworth gets upset." phase_1 defused.

My Notes:



`TEST` sets the zero flag, `ZF`, when the result of the AND operation is zero. If two operands are equal, their bitwise AND is zero when both are zero. `TEST` also sets the sign flag, `SF`, when the most significant bit is set in the result, and the parity flag, `PF`, when the number of set bits is even.



Register `eax` will contain **the return code from `strcmp`**, after the call. The `test eax, eax` is the same as `and eax, eax` (bitwise `and`) *except* that it doesn't store the result in `eax`. So `eax` isn't affected by the test, but the zero-flag is, for example.

The `test eax, eax` is necessary to make the `jne` work in the first place. And `jne` is the same as `jnz`, just as `je` is the same as `jz`. Both act based on the ZF (zero-flag) value.

The `jne` branch will be taken *if* ZF=0 and therefore whenever `strcmp` returns a non-zero value (i.e. strings not equal). Conversely if `eax` contains zero upon return from `strcmp`, the jump via `jne` will *not* happen.

Phase _2:

Ps. Underline means I’m pretty sure it’s the right number.

Guess1: “0 1 3 6 10 15”

Rbp-0x20 (32 in decimal)	0x00000000	0
Rbp-0x1c (28 in decimal)	0x00000001	1
Rbp-0x18 (24 in decimal)	0x00000003	3
Same rule	Same rule	Same rule

rdi	0x20312030	_1_0
rdi+0x4	0x20332036	_3_6

instruction	My eax
cmp $0x1,%eax	0
je 0x400b29 <phase_2+44>	
callq 0x40145c <explode_bomb>	

Comparing eax with 1, if it’s not equal, it would cause exploding, so I change first number to 1.

Guess 2: ”1 1 3 6 10 15”(randomly guess, try to find pattern)

instruction	My edx	eax
cmp %eax,%edx	1	2

Not equal would cause exploding, so I change second number to 2.

Guess 3:”1 2 4 7 11 16”

instruction	My edx	eax
cmp %eax,%edx	7	8

4 is luckily ok, but 7 is not ok, it shows that eax is 0x8, so I change it to 8.

Guess 4:”1 2 4 8 16 32”

I think I found the pattern! It’s two times than the previous one, so I just be brave and typed those 6 numbers and defused it!

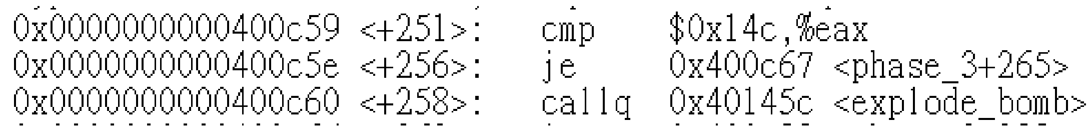
Phase _3:

Address	value	Little endian way	Actually showing
0x40171f	0x25206425	25642025	%d_%
0x40171f+0x4	0x64252063	63202564	c_ %d
0x40171f+0x8	0x400bbc00	00bc0b40	(it’s null, so just ignore it)

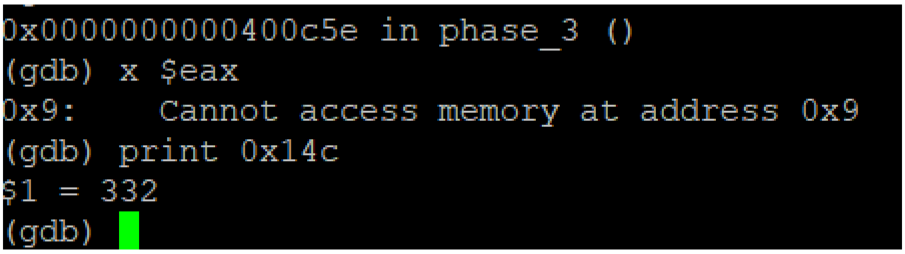
By the instruction, I can know that “%d %c %d” is the type means I need to input, which is “int, char, int”.

Guess 1: “6 E 9”

Keep doing “nexti”, until here:



```
0x0000000000400c59 <+251>:   cmp    $0x14c,%eax
0x0000000000400c5e <+256>:   je     0x400c67 <phase_3+265>
0x0000000000400c60 <+258>:   callq  0x40145c <explode_bomb>
```



```
0x0000000000400c5e in phase_3 ()
(gdb) x $eax
0x9:    Cannot access memory at address 0x9
(gdb) print 0x14c
$1 = 332
(gdb) 
```

So I would change 9 to 332

Guess 2: “6 E 332”

0xffe41075 = look at last one byte,75, which is “u” in ACSII Table.

al is 0x45, which is “E” in ACSII Table and that’s not equal, so I’m gonna change E into u.

Guess 3: “6 u 332”

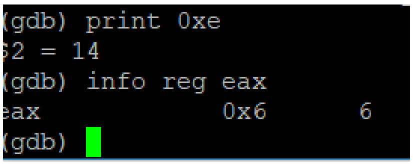
Lucky guess for first number, I passed it!

Phase_4:

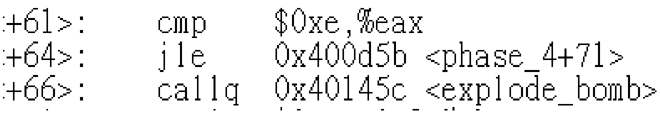
Address	value	Little endian way	Actually showing
0x401768	0x25206425	25642025	%d_%
0x40171f+0x4	0x65640064	64006465	d

By this table, I knew that I need to type two integers as variables.

Guess 1:"6 9"



```
(gdb) print 0xe
$2 = 14
(gdb) info reg eax
eax            0x6      6
(gdb) 
```



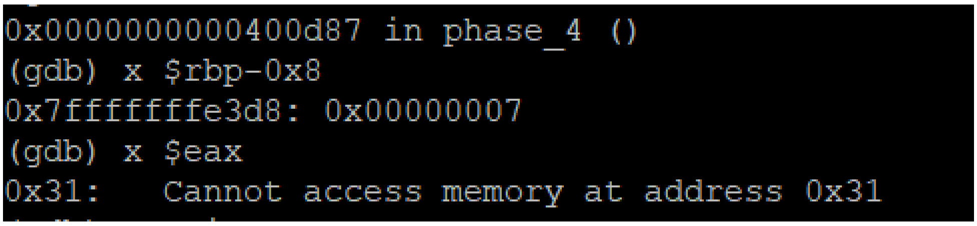
```
+61>:    cmp    $0xe,%eax
+64>:    jle    0x400d5b <phase_4+71>
+66>:    callq  0x40145c <explode_bomb>
```

By these instructions, I acknowledged that first number can't be greater than 14, but 14 is ok, so I changed 6 to 14.

Guess 2:"14 9"

First number should be 7. (I forgot to take screenshot…)

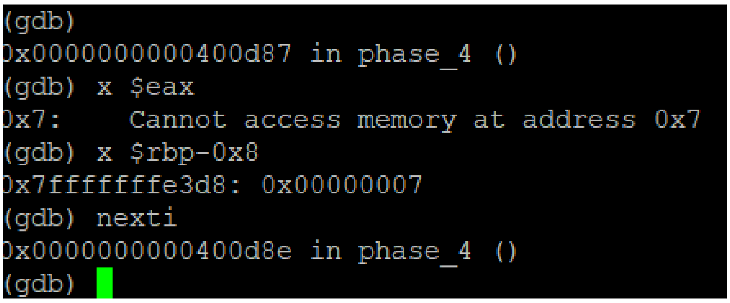
Guess 3: "7 49":



```
0x0000000000400d87 in phase_4 ()
(gdb) x $rbp-0x8
0x7fffffffe3d8: 0x00000007
(gdb) x $eax
0x31:   Cannot access memory at address 0x31
```

0x31 is 49, which is the second number I typed, so I decided to change it to 7.

Guess 4: "7 7"

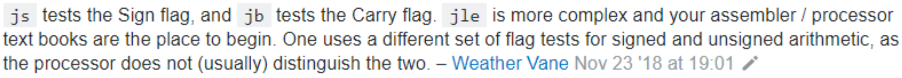


```
(gdb)
0x0000000000400d87 in phase_4 ()
(gdb) x $eax
0x7:    Cannot access memory at address 0x7
(gdb) x $rbp-0x8
0x7fffffffe3d8: 0x00000007
(gdb) nexti
0x0000000000400d8e in phase_4 ()
(gdb) 
```

0x0000000000400d8e <+122>:   leaveq

I think I got it! The answer is "7 7"

My notes:



`js` tests the Sign flag, and `jb` tests the Carry flag. `jle` is more complex and your assembler / processor text books are the place to begin. One uses a different set of flag tests for signed and unsigned arithmetic, as the processor does not (usually) distinguish the two. – Weather Vane Nov 23 '18 at 19:01