Contents

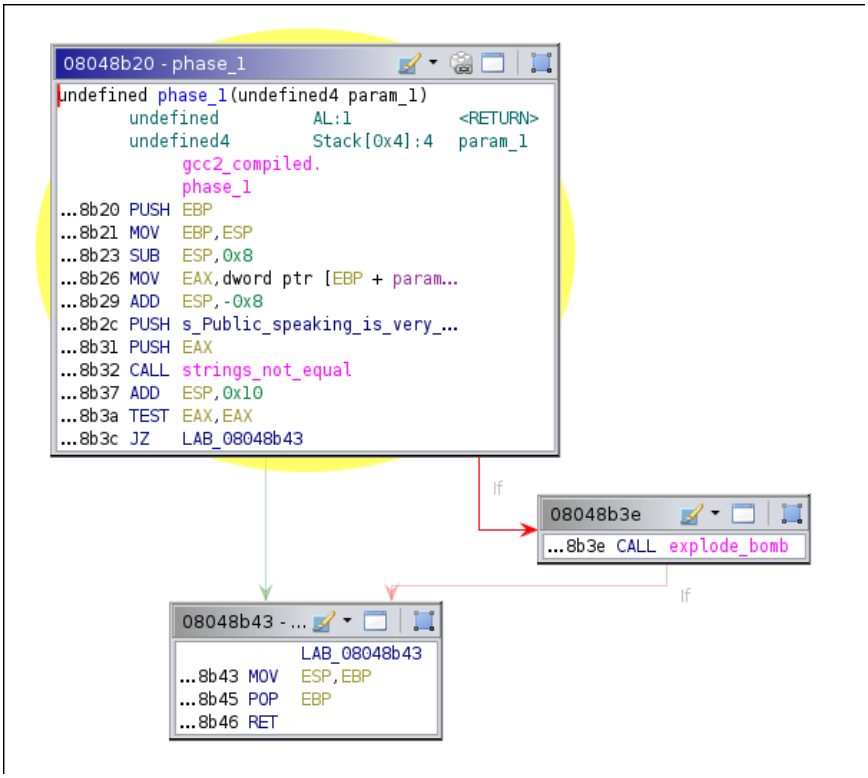
1 Homework

1.1 HW6

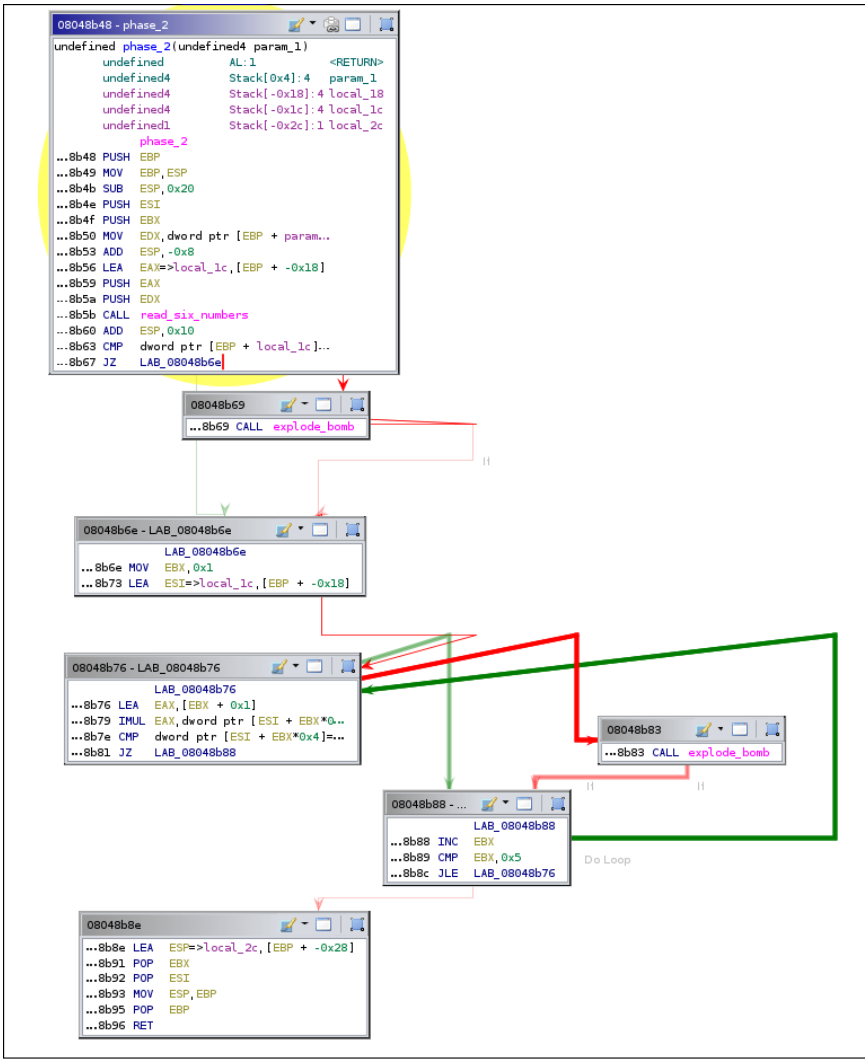
Done using Ghidra

1.1.1 Phase 1

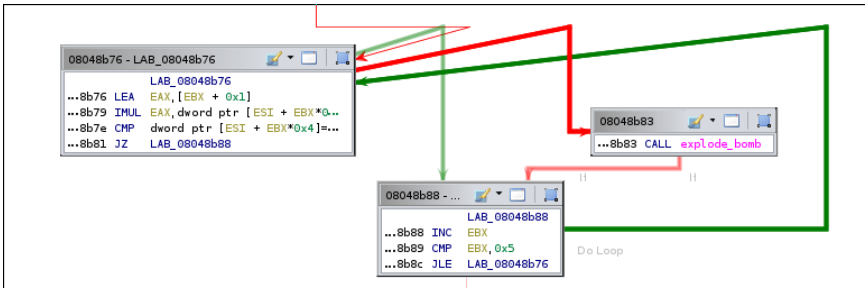
Phase 1 is simply a stirng comparison. By looking at the string, we can see the solution is `"Public speaking is very easy."`


08048b20 - phase_1
undefined phase_1(undefined4 param_1)
undefined AL:1 <RETURN>
undefined4 Stack[0x4]:4 param_1
gcc2_compiled.
phase_1
...8b20 PUSH EBP
...8b21 MOV EBP,ESP
...8b23 SUB ESP,0x8
...8b26 MOV EAX,dword ptr [EBP + param...
...8b29 ADD ESP,-0x8
...8b2c PUSH s_Public_speaking_is_very_...
...8b31 PUSH EAX
...8b32 CALL strings_not_equal
...8b37 ADD ESP,0x10
...8b3a TEST EAX,EAX
...8b3c JZ LAB_08048b43
If
08048b3e
...8b3e CALL explode_bomb
If
08048b43 - ...
LAB_08048b43
...8b43 MOV ESP,EBP
...8b45 POP EBP
...8b46 RET

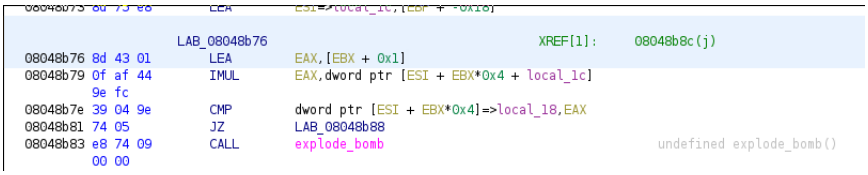
1.1.2 Phase 2



Taking a look at the phase shows that 6 numbers are read from the user. It can be seen the the first number must be a 1. Further down the program, there is a loop that iterates through the rest of the numbers.



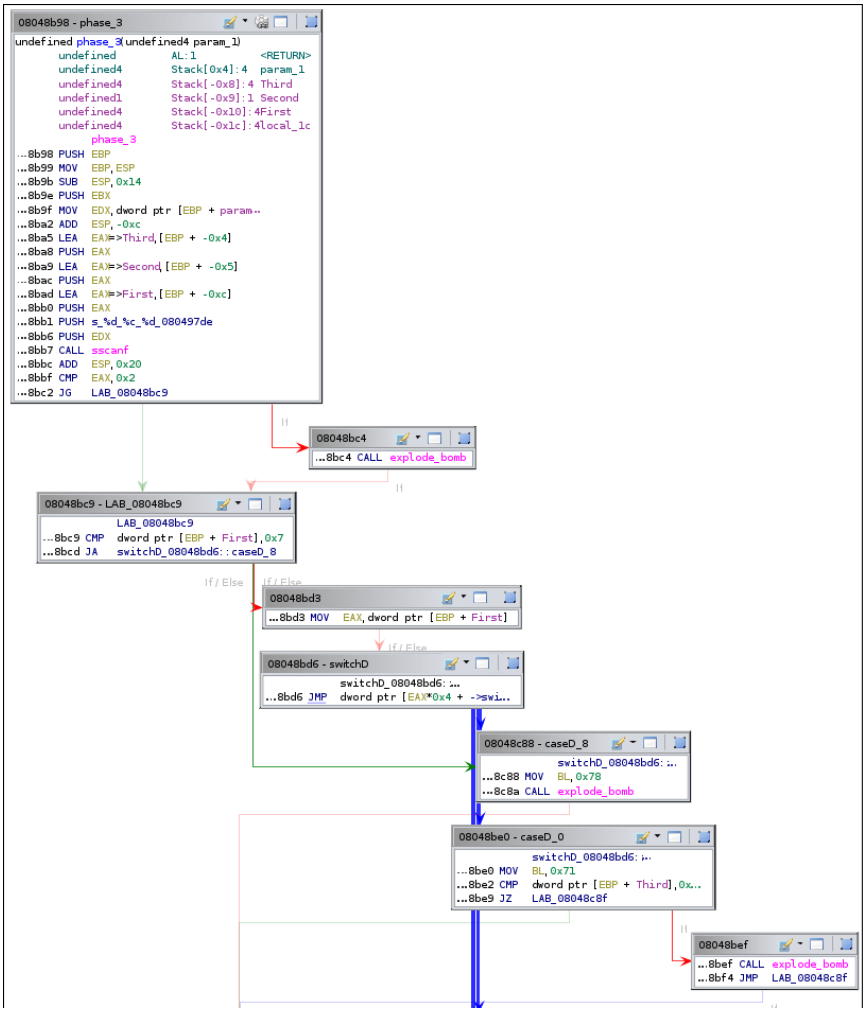
The segment that is looped checks if the next number is equal to its index+1, multiplied by the last index.



This means our solution is `1 2 6 24 120 720`

1.1.3 Phase 3

This phase is a simple 2 numbers with a character.



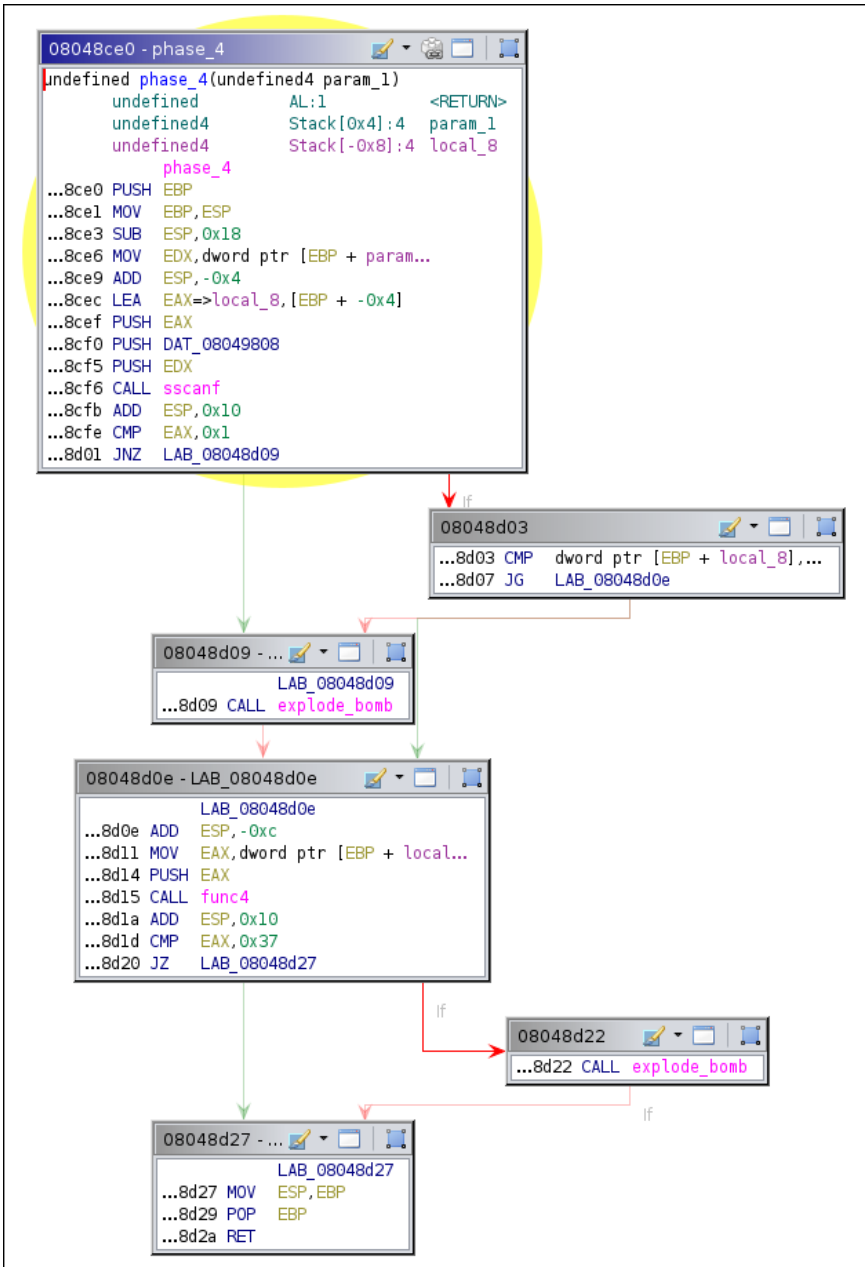
There is a jump table where the second number is compared with a required value, and loads an associated character to be compared.

```
                    switchD_08048bd6::caseD_0                       XREF[2]:     08048bd6(j), 080497e8(*)
08048be0 b3 71           MOV        BL,0x71
08048be2 81 7d fc        CMP        dword ptr [EBP + Third],0x309
         09 03 00 00
08048be9 0f 84 a0        JZ         LAB_08048c8f
         00 00 00
08048bef e8 08 09        CALL       explode_bomb                                  undefined explode_bomb()
         00 00
08048bf4 e9 96 00        JMP        LAB_08048c8f
         00 00
```

Using the first case, we get the solution `0 q 777`

1.1.4 Phase 4

The phase reads 1 number which must be greater than 0. This number is then passed into a function, and explodes if the returned value is not 55



The function can be described as:

```
def func4(int n):
    if n<=1:
return 1
    else:
return func4(n-1) + func4(n-2)
```

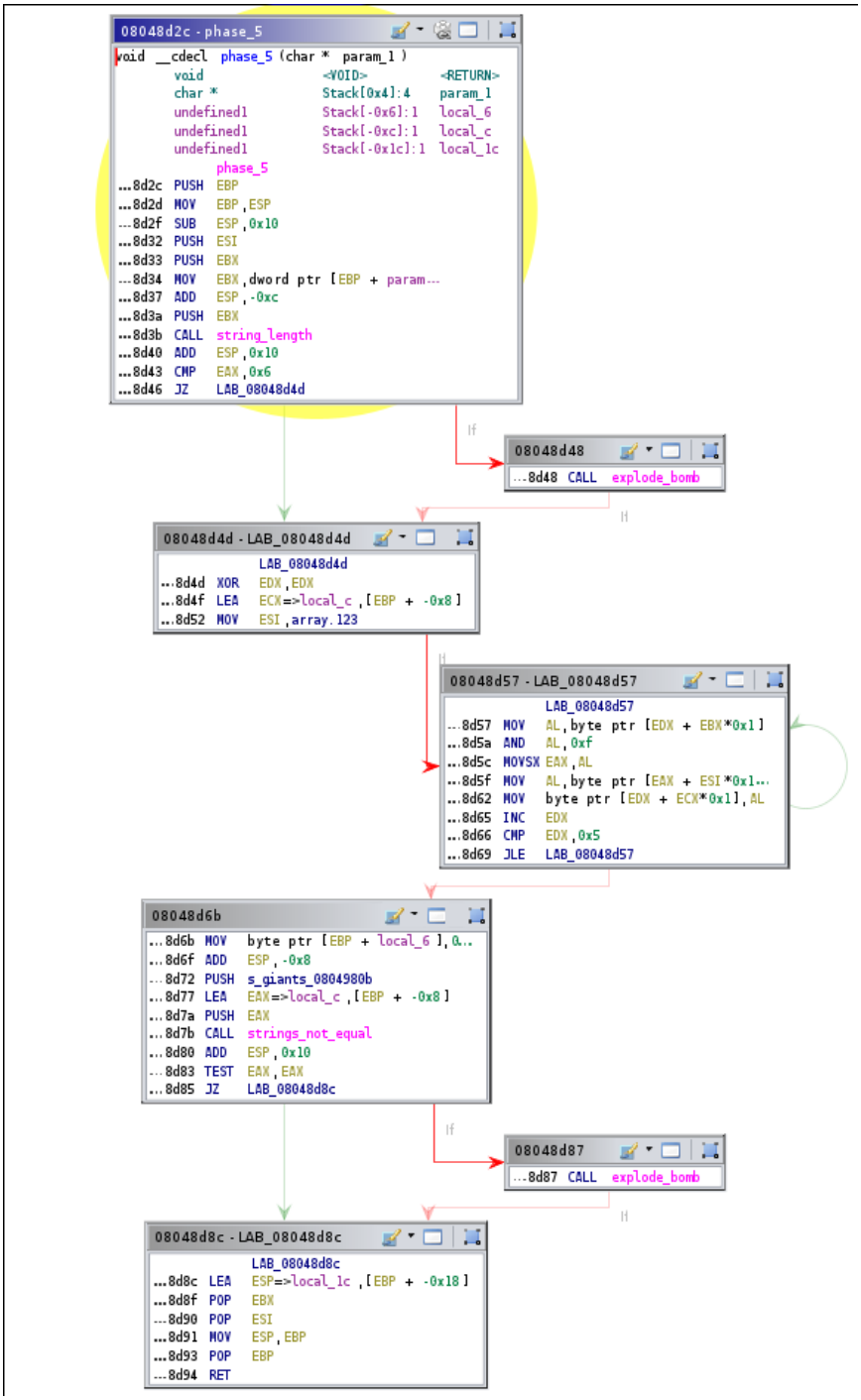
func4 is clearly the fibonacci numbers.

Thus, the solution to this phase is `9`

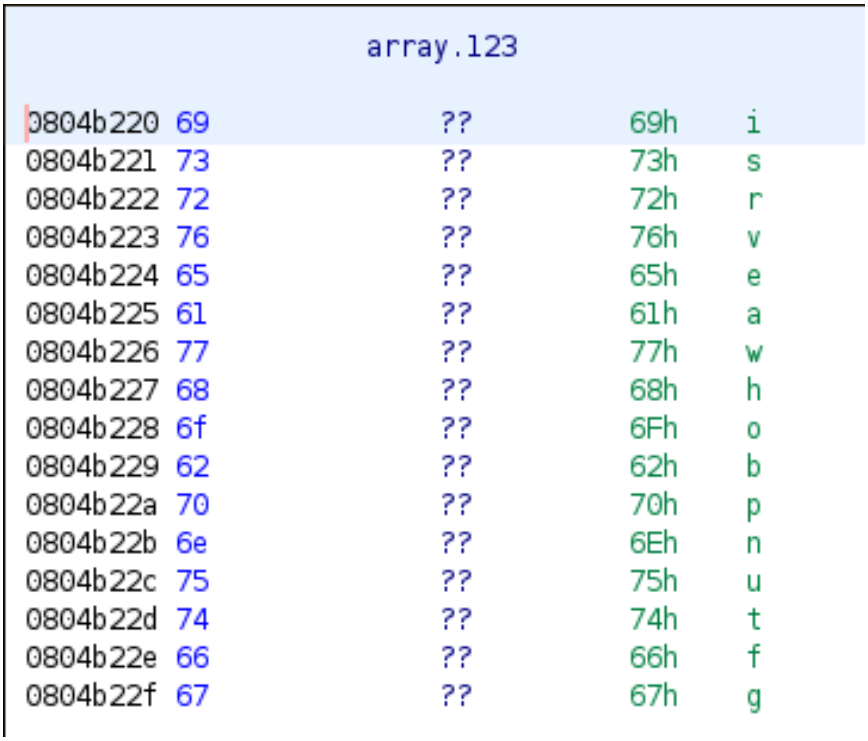
1.1.5 Phase 5

This phase takes in a string input of length 6

This string is then translated using a table and then compared with the string `"giants"`


08048d2c - phase_5
void __cdecl phase_5 (char * param_1)
void <VOID> <RETURN>
char * Stack[0x4]:4 param_1
undefined1 Stack[-0x6]:1 local_6
undefined1 Stack[-0xc]:1 local_c
undefined1 Stack[-0x1c]:1 local_1c
phase_5
...8d2c PUSH EBP
...8d2d MOV EBP,ESP
...8d2f SUB ESP,0x10
...8d32 PUSH ESI
...8d33 PUSH EBX
...8d34 MOV EBX,dword ptr [EBP + param...
...8d37 ADD ESP,-0xc
...8d3a PUSH EBX
...8d3b CALL string_length
...8d40 ADD ESP,0x10
...8d43 CMP EAX,0x6
...8d46 JZ LAB_08048d4d
If
08048d48
...8d48 CALL explode_bomb
H
08048d4d - LAB_08048d4d
LAB_08048d4d
...8d4d XOR EDX,EDX
...8d4f LEA ECX=>local_c ,[EBP + -0x8]
...8d52 MOV ESI,array.123
H
08048d57 - LAB_08048d57
LAB_08048d57
...8d57 MOV AL,byte ptr [EDX + EBX*0x1]
...8d5a AND AL,0xf
...8d5c MOVSX EAX,AL
...8d5f MOV AL,byte ptr [EAX + ESI*0x1...
...8d62 MOV byte ptr [EDX + ECX*0x1],AL
...8d65 INC EDX
...8d66 CMP EDX,0x5
...8d69 JLE LAB_08048d57
08048d6b
...8d6b MOV byte ptr [EBP + local_6],0...
...8d6f ADD ESP,-0x8
...8d72 PUSH s_giants_0804980b
...8d77 LEA EAX=>local_c ,[EBP + -0x8]
...8d7a PUSH EAX
...8d7b CALL strings_not_equal
...8d80 ADD ESP,0x10
...8d83 TEST EAX,EAX
...8d85 JZ LAB_08048d8c
If
08048d87
...8d87 CALL explode_bomb
H
08048d8c - LAB_08048d8c
LAB_08048d8c
...8d8c LEA ESP=>local_1c ,[EBP + -0x18]
...8d8f POP EBX
...8d90 POP ESI
...8d91 MOV ESP,EBP
...8d93 POP EBP
...8d94 RET

The function takes the lower 4 bits of each character, indexing it with the said table to get the resulting character.



array.123

0804b220	69	??	69h	i
0804b221	73	??	73h	s
0804b222	72	??	72h	r
0804b223	76	??	76h	v
0804b224	65	??	65h	e
0804b225	61	??	61h	a
0804b226	77	??	77h	w
0804b227	68	??	68h	h
0804b228	6f	??	6Fh	o
0804b229	62	??	62h	b
0804b22a	70	??	70h	p
0804b22b	6e	??	6Eh	n
0804b22c	75	??	75h	u
0804b22d	74	??	74h	t
0804b22e	66	??	66h	f
0804b22f	67	??	67h	g

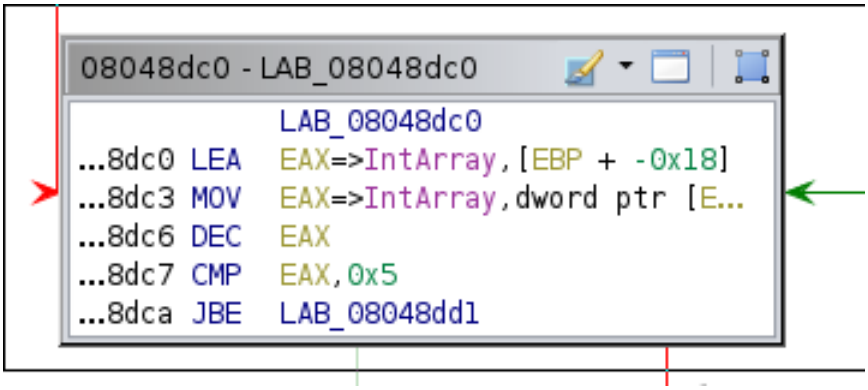
Using the table, we can find the required byte sequences to produce `"giants"`. The sequence is `1111, 0000 0101, 1011, 1101, 0001`. We can then use the byte sequence to generate a valid solution.

One possible solution is `"OPEKMA"`

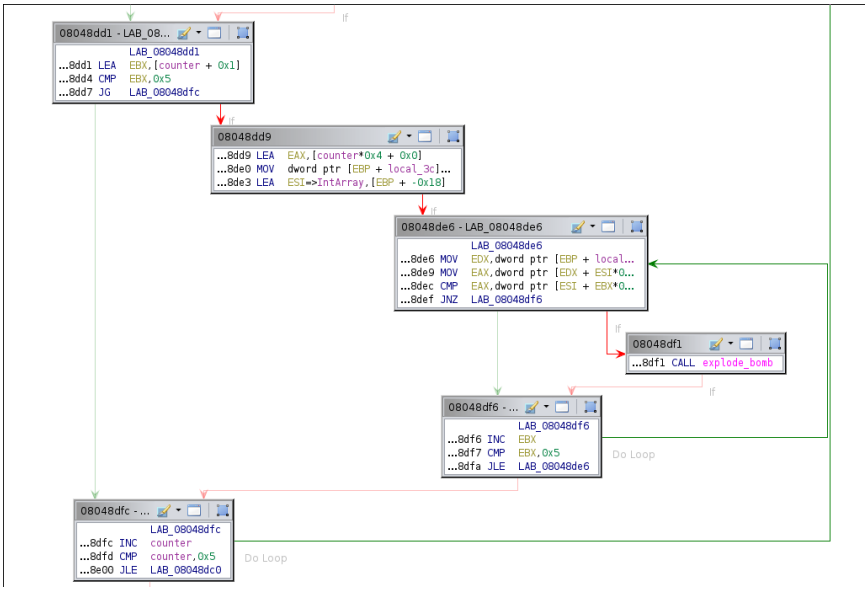
1.1.6 Phase 6

Phase 6 starts by reading in 6 numbers.

The function then checks to make sure all 6 numbers are < 7

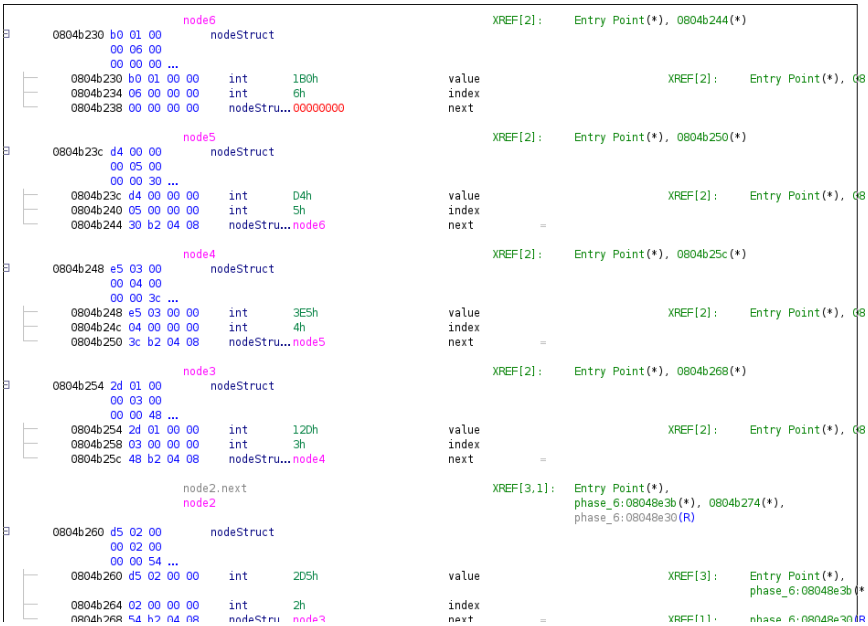


It then checks to make sure all 6 numbers are unique



We see that there are structures in the data segment of the program.

Following the memory address, we find 6 nodes in data. Setting their data types as nodeStruct allows us to view their contents.

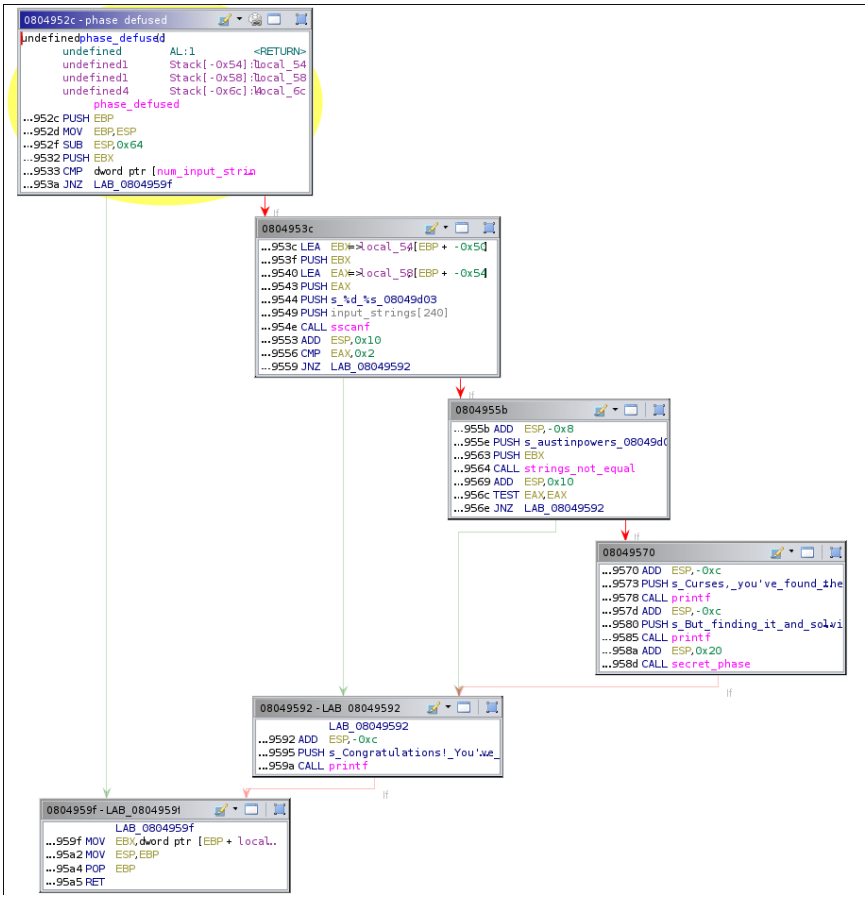


Inspecting these values, we see that it is a linked list. Sorting them in decreasing order yields the solution `4 2 6 3 1 5`

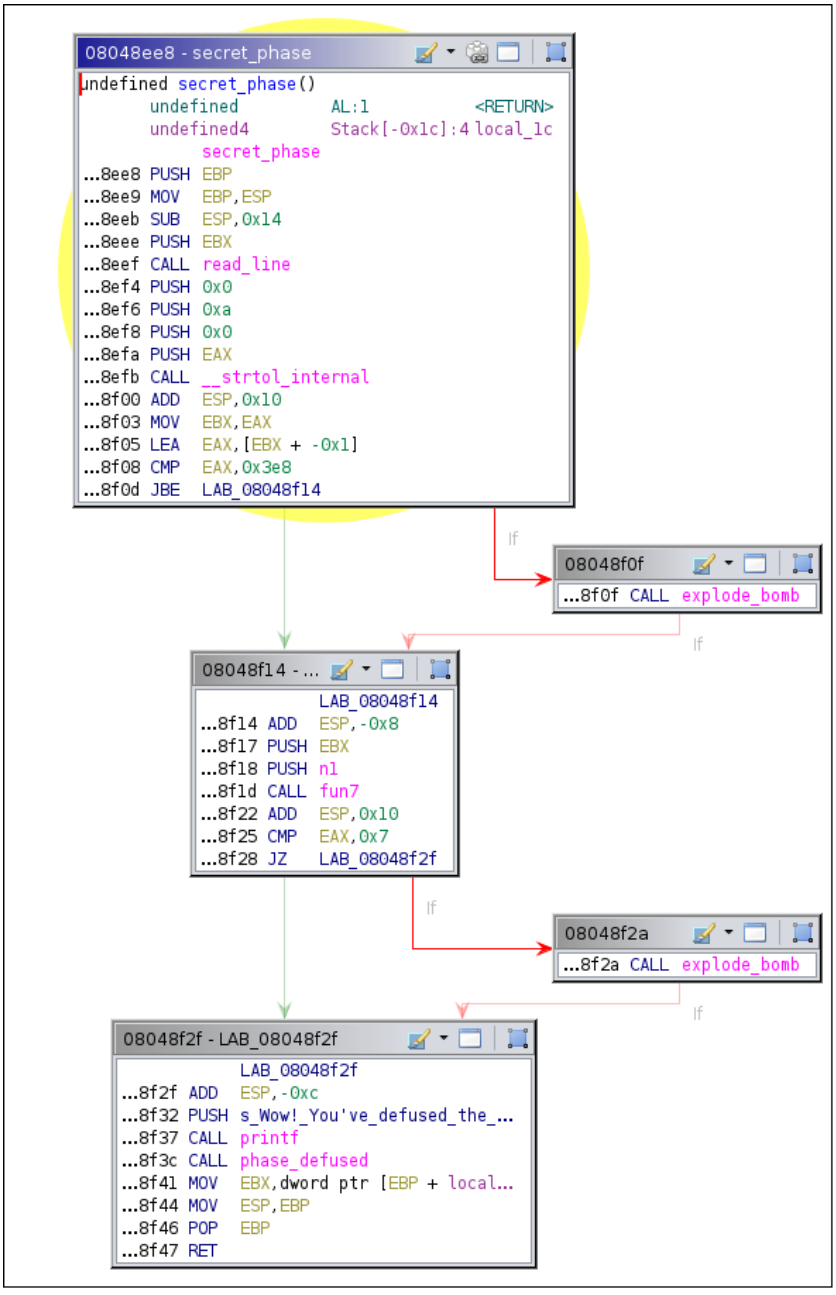
This marks the end of the 6 phases of Bomb Lab. However, there is another phase that we can reach which is hidden unless we look at the underlying code

1.1.7 Secret Phase

To get to the secret phase, we must first look at the function `phase_defused`.



Inside `phase_defused`, we see that there is a call to `secret_phase`, though it can not normally be reached. To reach it, we must append a string to the end of the solution of phase 4. Taking a look at the string, we can append `"austinpowers"` to the end of the solution of phase 4 to reach this secret phase

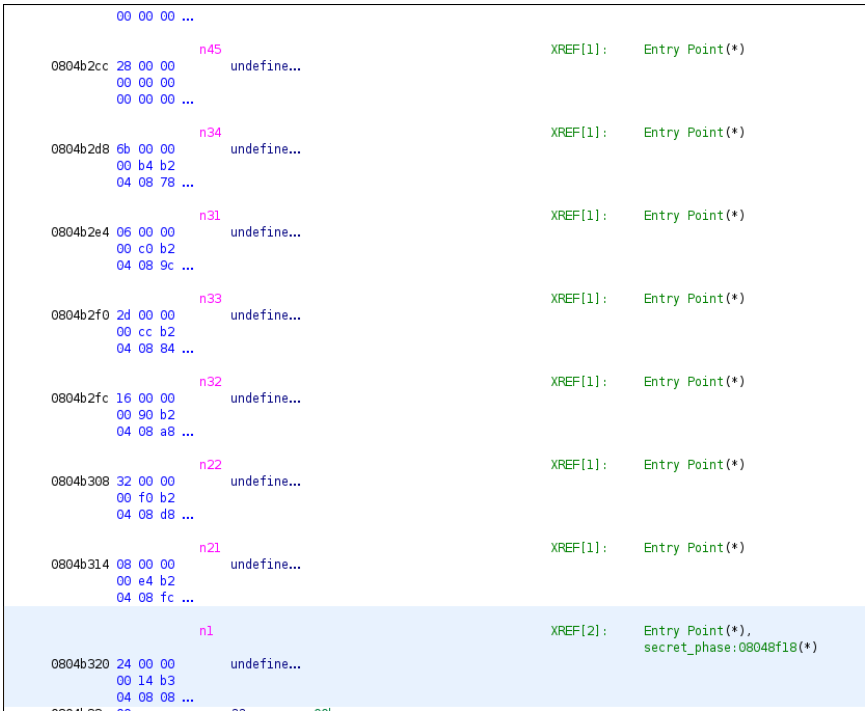


Taking a look at the secret phase, we see that initially it checks if the input is less than or equal to 1001.

Afterwards, there is a function call `fun7`, which is then compared and

explodes if the value is not 7.

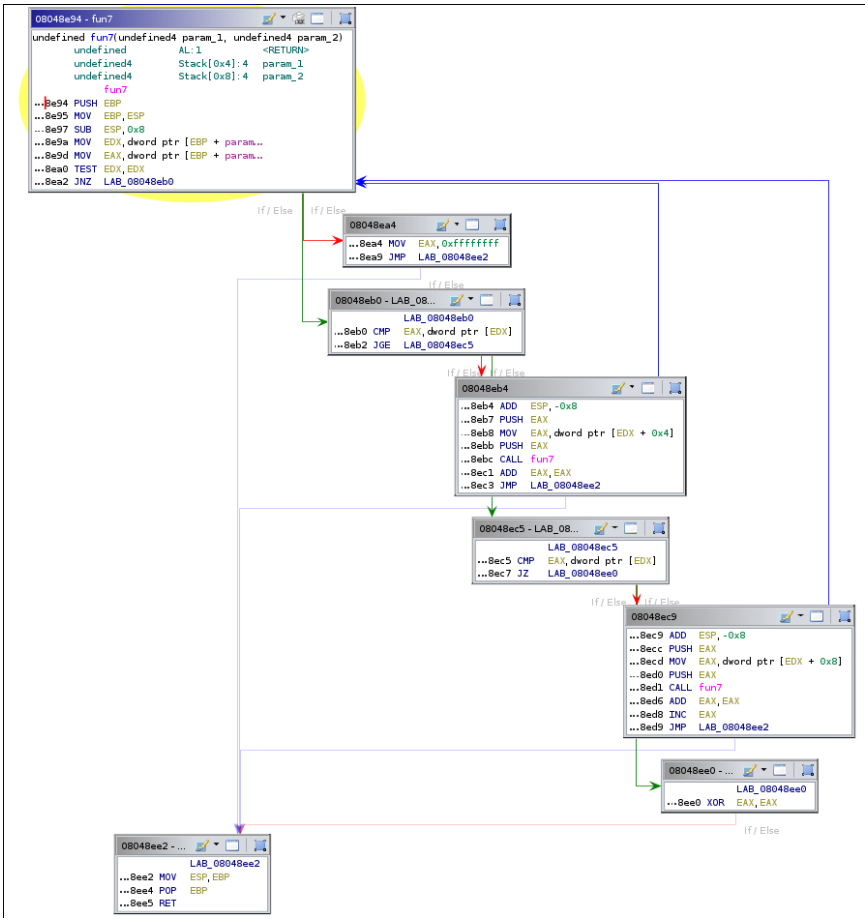
A pointer to segments in data can be found in the initial call to `fun7`.



Taking a look at the data, we can clearly see a binary tree.

If the passed node is zero, the function returns -1. If the node's value is less than the input, it returns double the result of recursing the left child, and the same input. Otherwise, it returns twice the result+1 of recursing the right child, and the same input

To solve this, a path must be found where the resultant is 7



Tracing the call structure, we find the solution `1001`, which defuses the secret phase.

This marks the end of Bomb Lab