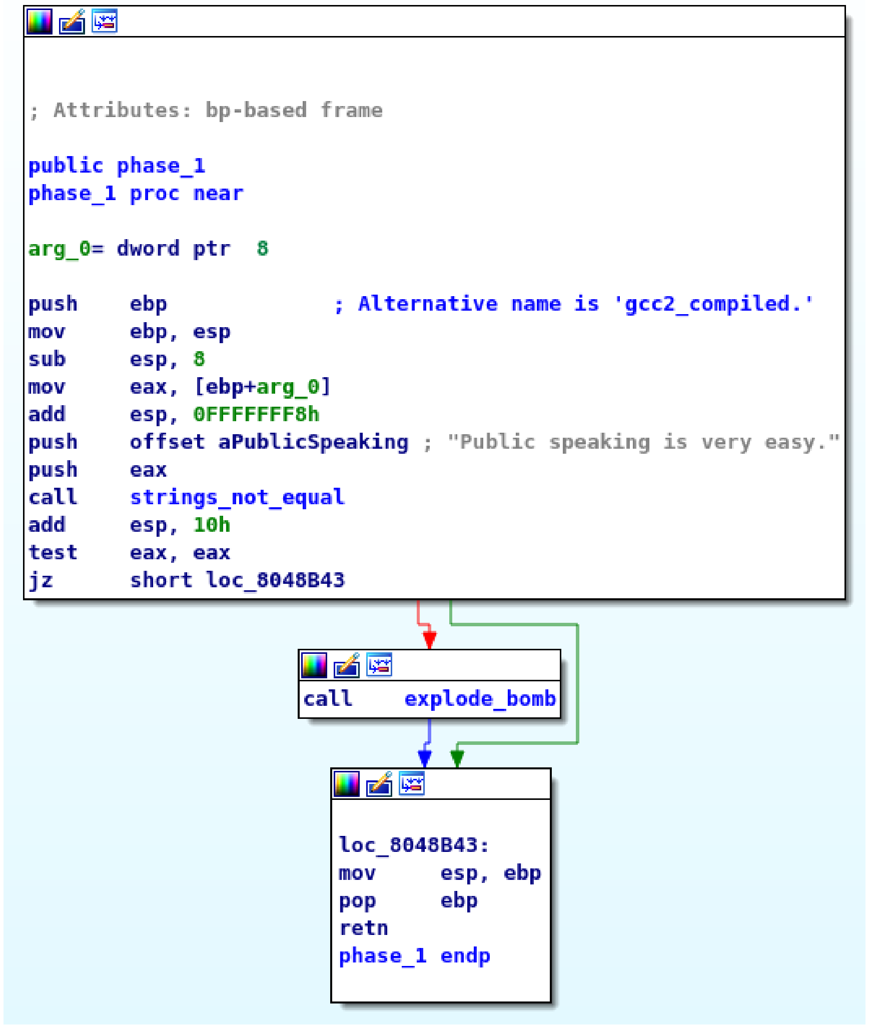
HW6

April 20, 2020

Contents

1 Phase 1

This phase simply does only simple string compare.



```
; Attributes: bp-based frame

public phase_1
phase_1 proc near

arg_0= dword ptr  8

push    ebp             ; Alternative name is 'gcc2_compiled.'
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+arg_0]
add     esp, 0FFFFFFF8h
push    offset aPublicSpeaking ; "Public speaking is very easy."
push    eax
call    strings_not_equal
add     esp, 10h
test    eax, eax
jz      short loc_8048B43
```

```
call    explode_bomb
```

```
loc_8048B43:
mov     esp, ebp
pop     ebp
retn
phase_1 endp
```

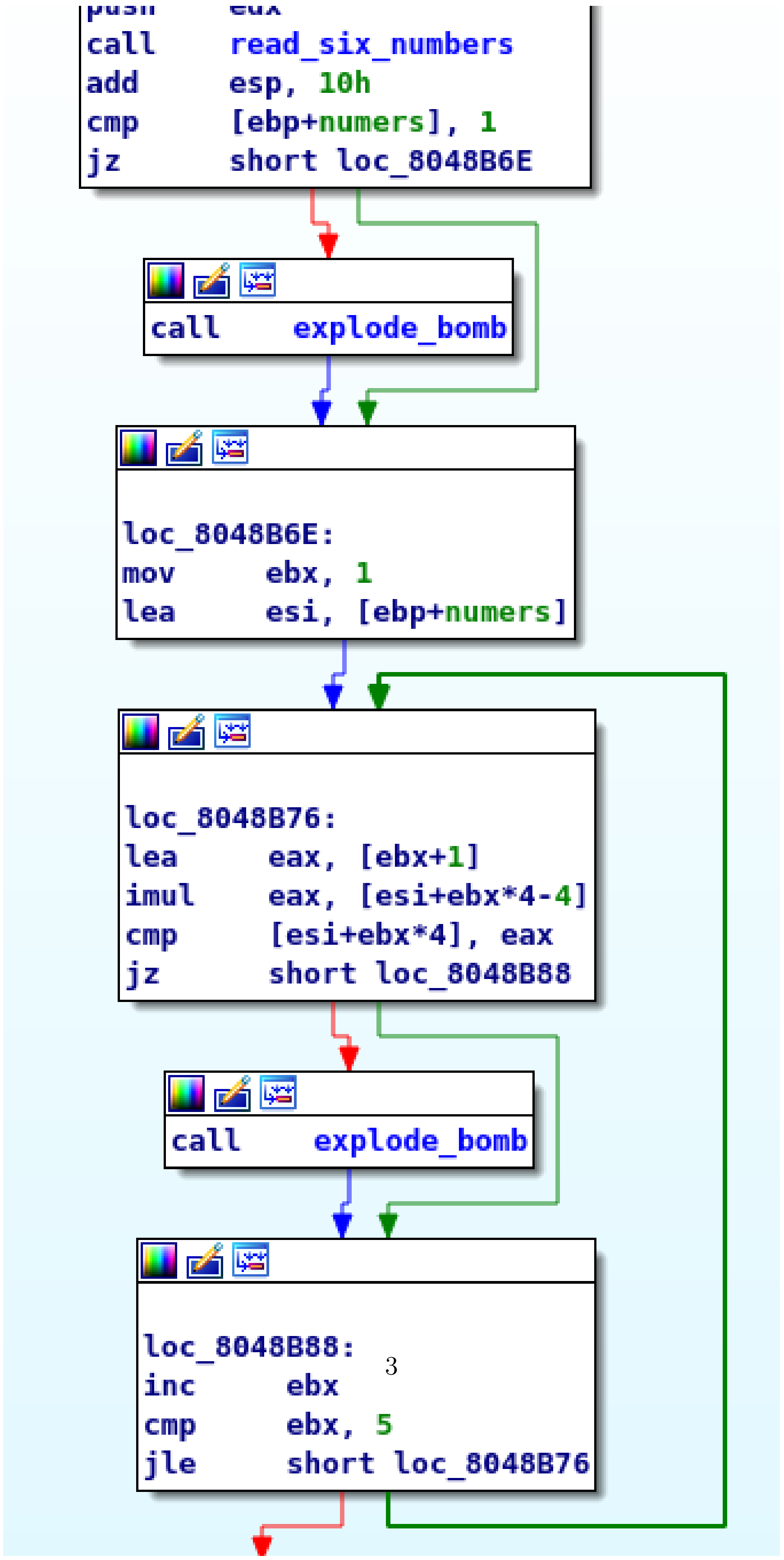
2 Phase 2

This phase reads in six numbers and checks the values.

We can easily see that the first number has to be one. The following code checks in a loop that the next number is equal to its index plus one, multiplied by the last index.

That means that the second number has to be `2 * 1 = 2`, then `3 * 2 = 6`, `4 * 6 = 24`, etc.

This yields the solution `1 2 6 24 120 720`.



```
push    edx
call    read_six_numbers
add     esp, 10h
cmp     [ebp+numers], 1
jz      short loc_8048B6E
```

```
call    explode_bomb
```

```
loc_8048B6E:
mov     ebx, 1
lea     esi, [ebp+numers]
```

```
loc_8048B76:
lea     eax, [ebx+1]
imul    eax, [esi+ebx*4-4]
cmp     [esi+ebx*4], eax
jz      short loc_8048B88
```

```
call    explode_bomb
```

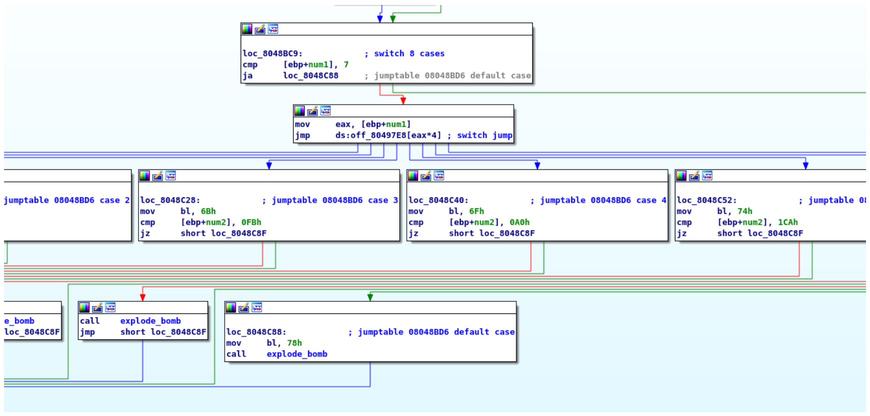
```
loc_8048B88:
inc     ebx
cmp     ebx, 5
jle     short loc_8048B76
```

3 Phase 3

This phase reads in two numbers and a character.

The first number is indexed into a jump table with 8 cases, exploding if out of range.

Each target of the jump table compares the second number with a required value, and loads a required value for the char to be compared with.



```
loc_8048BC9:                ; switch 8 cases
cmp     [ebp+num1], 7
ja      loc_8048C88         ; jumptable 08048BD6 default case
```

```
mov     eax, [ebp+num1]
jmp     ds:off_80497E8[eax*4] ; switch jump
```

```
jumptable 08048BD6 case 2
```

```
loc_8048C28:                ; jumptable 08048BD6 case 3
mov     bl, 6Bh
cmp     [ebp+num2], 0FBh
jz      short loc_8048C8F
```

```
loc_8048C40:                ; jumptable 08048BD6 case 4
mov     bl, 6Fh
cmp     [ebp+num2], 0A0h
jz      short loc_8048C8F
```

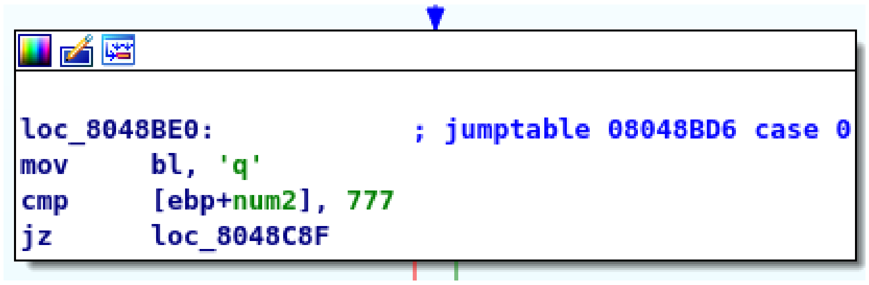
```
loc_8048C52:                ; jumptable 08
mov     bl, 74h
cmp     [ebp+num2], 1CAh
jz      short loc_8048C8F
```

```
e_bomb
loc_8048C8F
```

```
call    explode_bomb
jmp     short loc_8048C8F
```

```
loc_8048C88:                ; jumptable 08048BD6 default case
mov     bl, 78h
call    explode_bomb
```

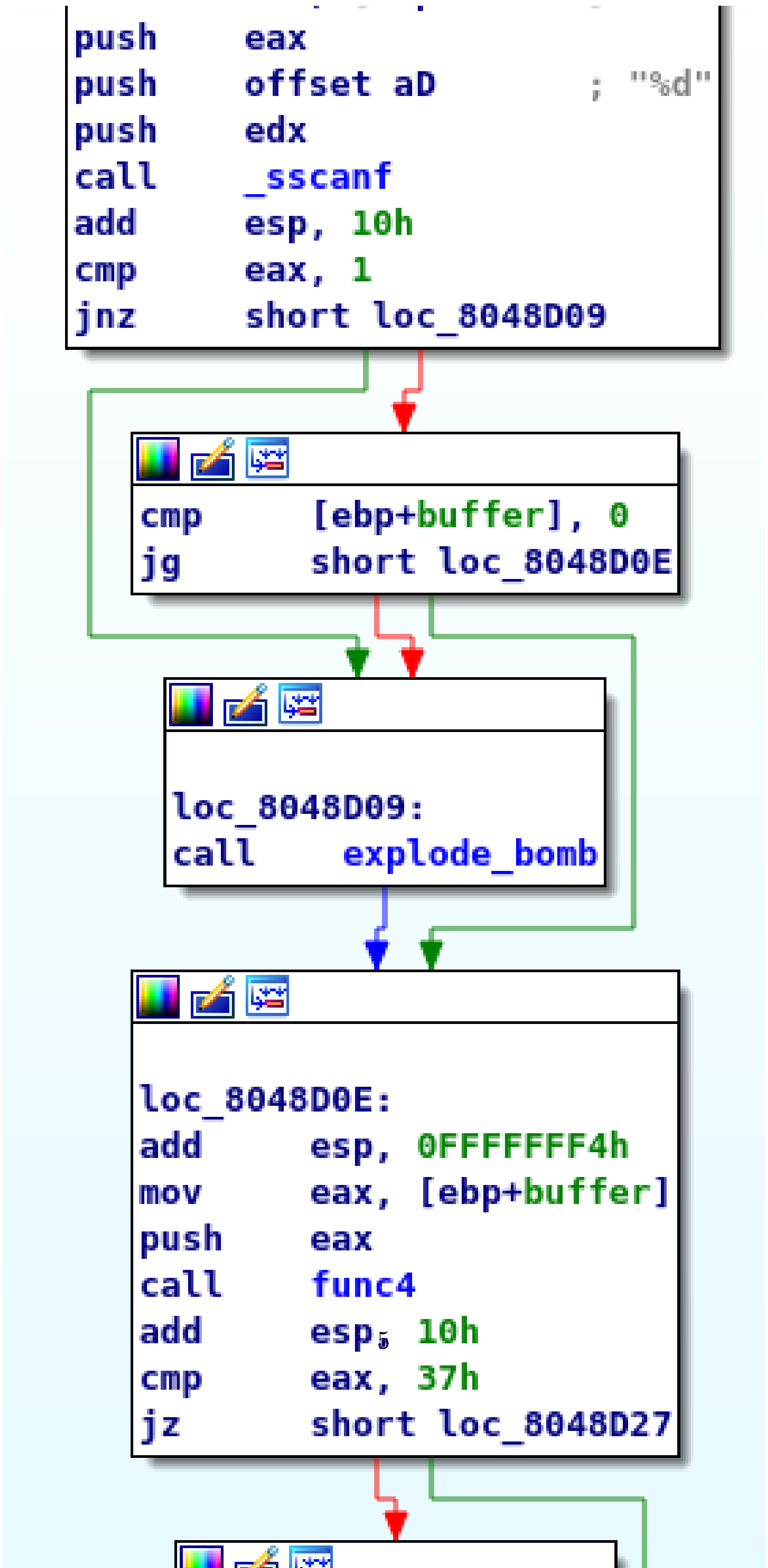
We can just pick the first one, from which we can see the solution is `0 q 777`.



```
loc_8048BE0:                ; jumptable 08048BD6 case 0
mov     bl, 'q'
cmp     [ebp+num2], 777
jz      loc_8048C8F
```

4 Phase 4

This phase reads in a single number that must be greater than zero, calls a function, and explodes unless the return value is `55`.



```
push    eax
push    offset aD        ; "%d"
push    edx
call    _sscanf
add     esp, 10h
cmp     eax, 1
jnz     short loc_8048D09
```

```
cmp     [ebp+buffer], 0
jg      short loc_8048D0E
```

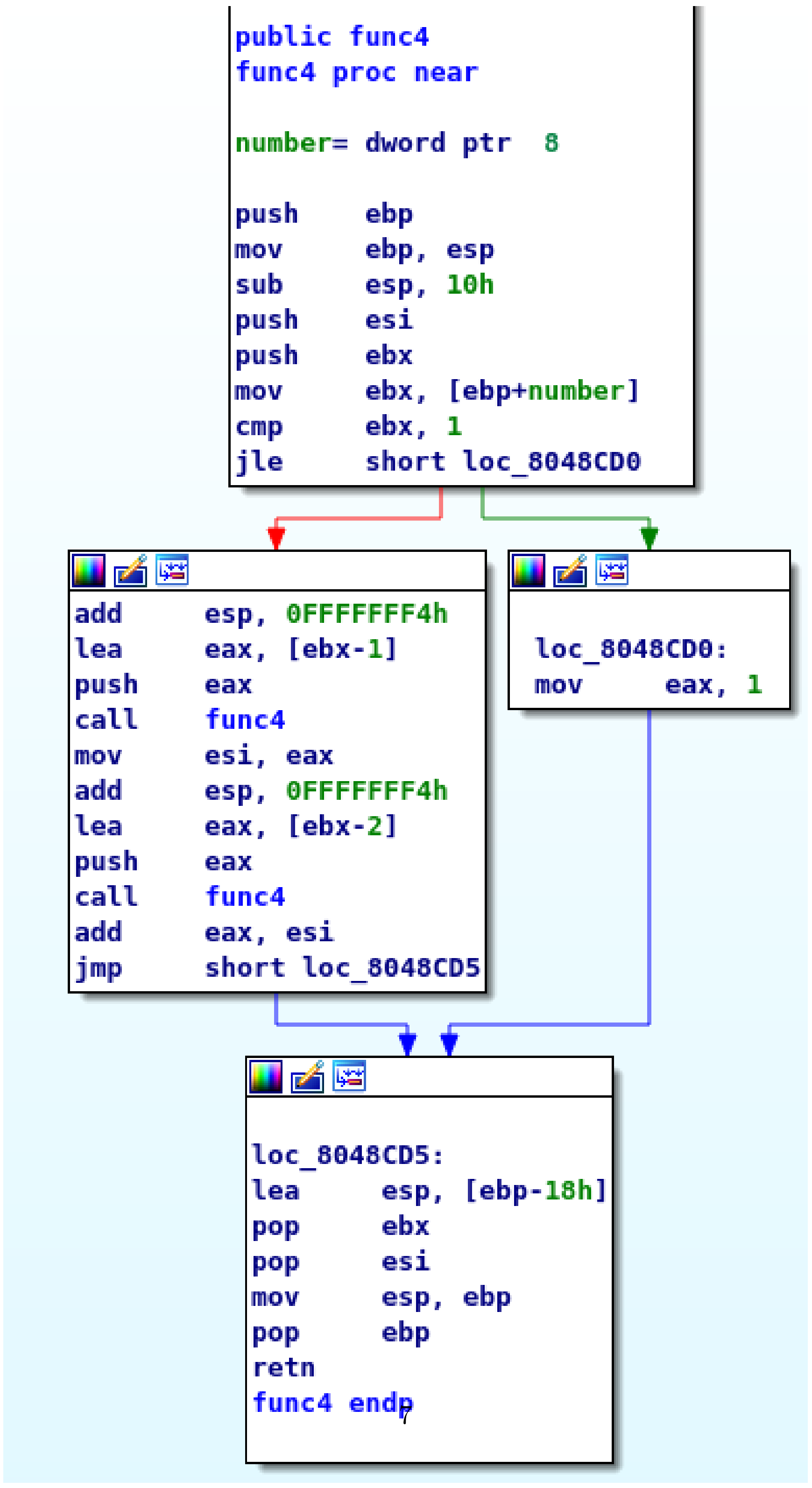
```
loc_8048D09:
call    explode_bomb
```

```
loc_8048D0E:
add     esp, 0FFFFFFF4h
mov     eax, [ebp+buffer]
push    eax
call    func4
add     esp, 10h
cmp     eax, 37h
jz      short loc_8048D27
```

This function can be summarized as:

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

Which clearly computes fibonacci numbers.
Thus, the required input to get `55` is `9`.



```
public func4
func4 proc near

number= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 10h
push    esi
push    ebx
mov     ebx, [ebp+number]
cmp     ebx, 1
jle     short loc_8048CD0
```

```
add     esp, 0FFFFFFF4h
lea     eax, [ebx-1]
push    eax
call    func4
mov     esi, eax
add     esp, 0FFFFFFF4h
lea     eax, [ebx-2]
push    eax
call    func4
add     eax, esi
jmp     short loc_8048CD5
```

```
loc_8048CD0:
mov     eax, 1
```

```
loc_8048CD5:
lea     esp, [ebp-18h]
pop     ebx
pop     esi
mov     esp, ebp
pop     ebp
retn
func4 endp
```

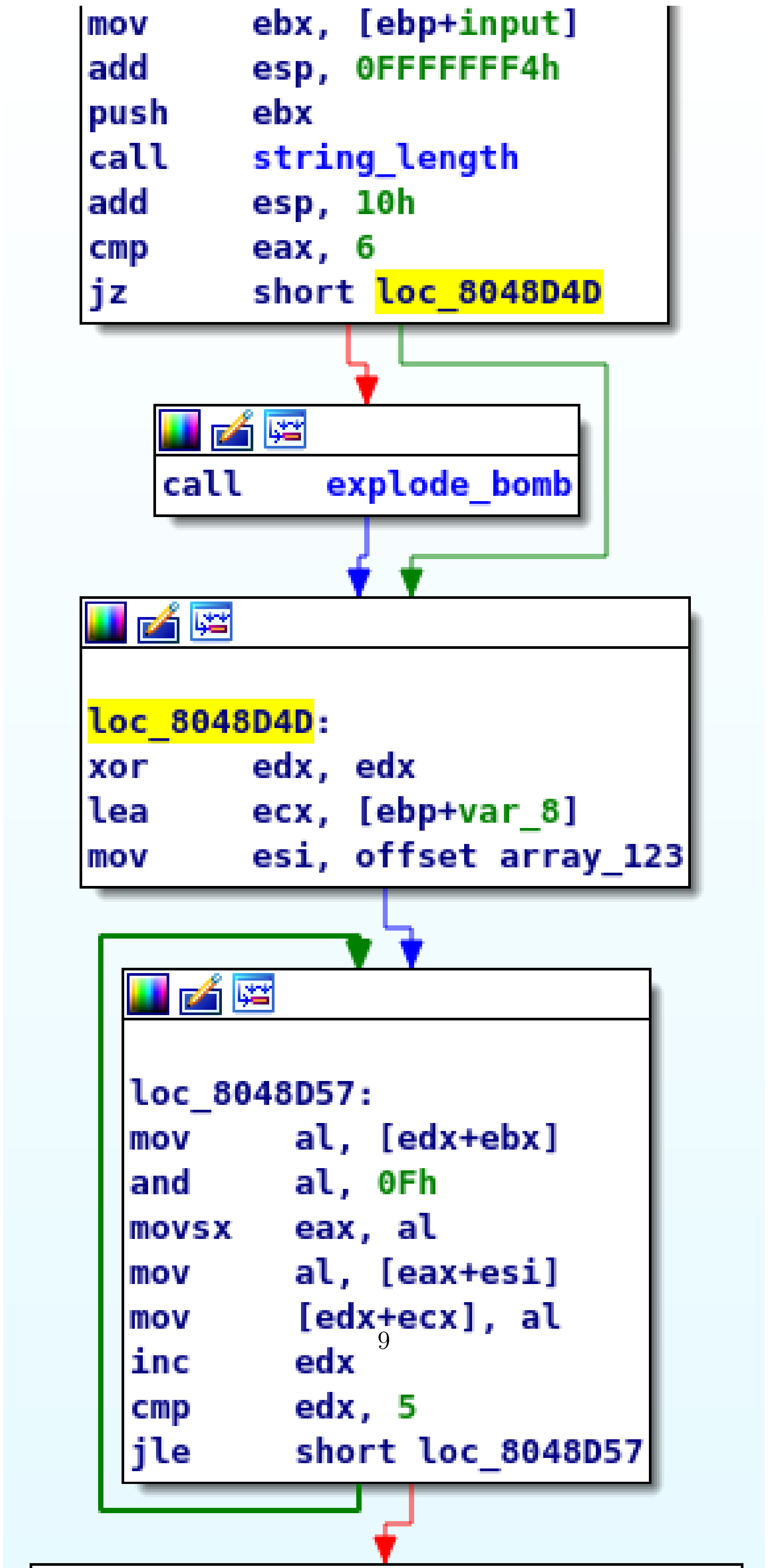
5 Phase 5

This phase reads in a string of 6 characters.

It translates the input string into a new buffer, comparing it with the string `"giants"`.

The function takes the lower 4 bits of each input character, indexing it into a table to get the resulting character.

We can inspect the table to figure out that the required byte sequence get `"giants"` is `1111, 0000, 0101, 1011, 1101, 0001`. Cross referencing this with a binary ascii table, one possible input string that produces this sequence is `"O@EKMA"`.


```
mov     ebx, [ebp+input]
add     esp, 0FFFFFFF4h
push    ebx
call    string_length
add     esp, 10h
cmp     eax, 6
jz      short loc_8048D4D
```

```
call    explode_bomb
```

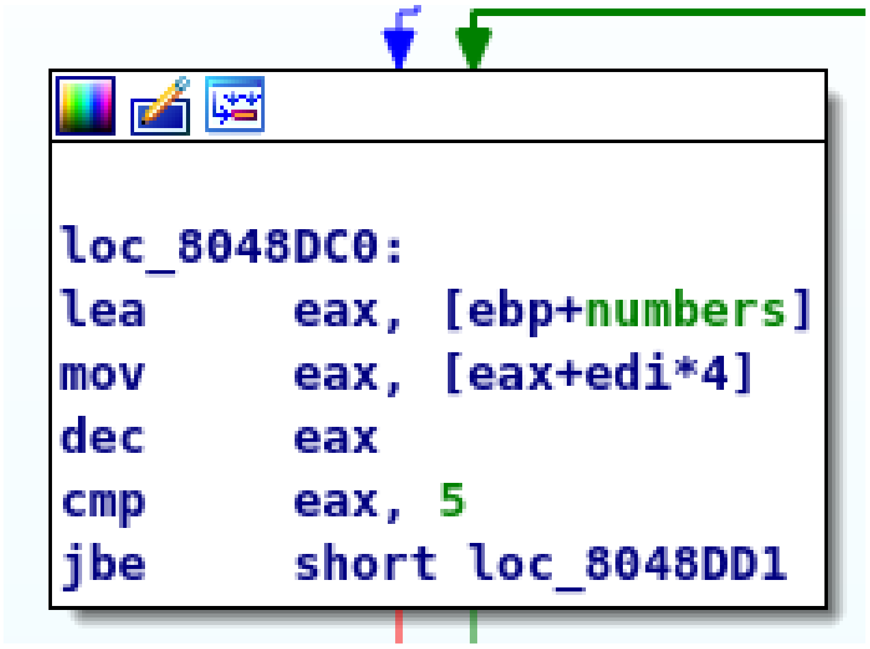
```
loc_8048D4D:
xor     edx, edx
lea     ecx, [ebp+var_8]
mov     esi, offset array_123
```

```
loc_8048D57:
mov     al, [edx+ebx]
and     al, 0Fh
movsx   eax, al
mov     al, [eax+esi]
mov     [edx+ecx], al
inc     edx
cmp     edx, 5
jle     short loc_8048D57
```

6 Phase 6

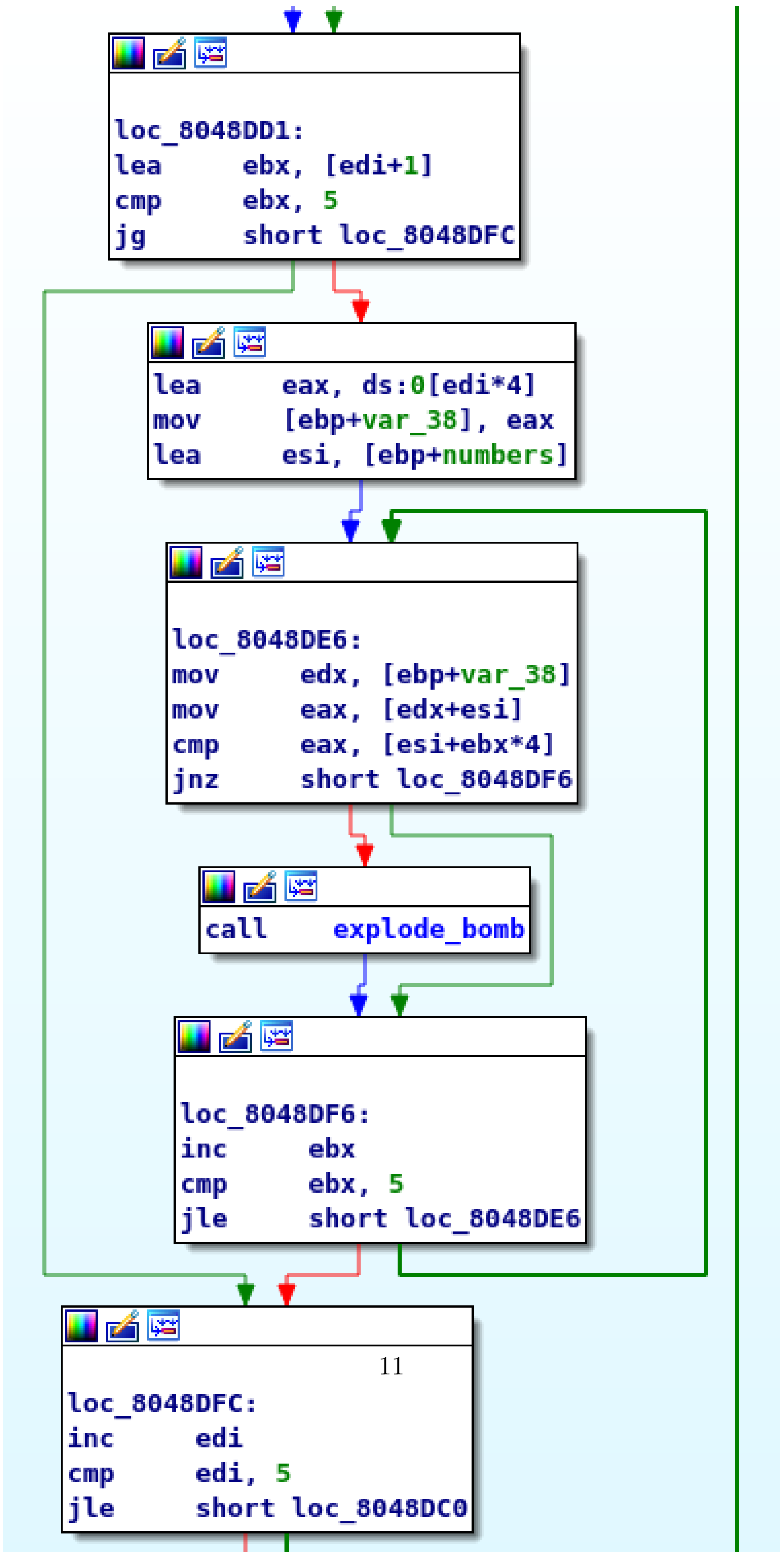
This phase again reads in six numbers.

The function first makes sure all numbers are <= 6.



```
loc_8048DC0:
lea     eax, [ebp+numbers]
mov     eax, [eax+edi*4]
dec     eax
cmp     eax, 5
jbe     short loc_8048DD1
```

Then it starts a nested loop, comparing each number with every other number. The bomb explodes if any two are the same.



```
loc_8048DD1:
lea     ebx, [edi+1]
cmp     ebx, 5
jg      short loc_8048DFC
```

```
lea     eax, ds:0[edi*4]
mov     [ebp+var_38], eax
lea     esi, [ebp+numbers]
```

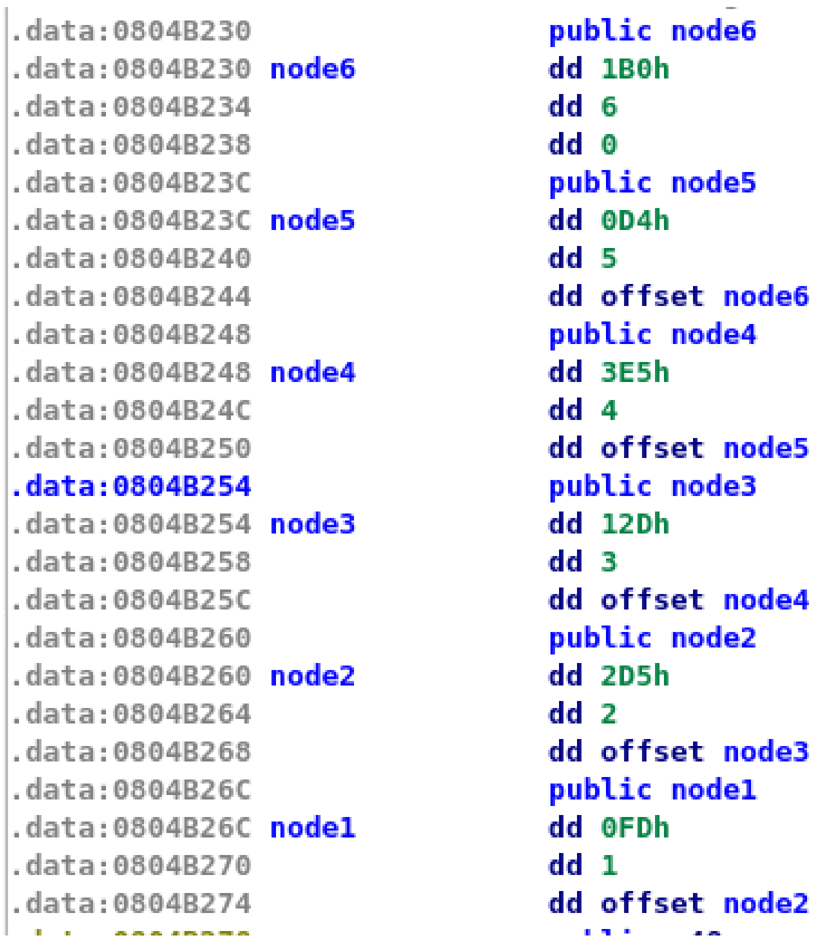
```
loc_8048DE6:
mov     edx, [ebp+var_38]
mov     eax, [edx+esi]
cmp     eax, [esi+ebx*4]
jnz     short loc_8048DF6
```

```
call    explode_bomb
```

```
loc_8048DF6:
inc     ebx
cmp     ebx, 5
jle     short loc_8048DE6
```

```
loc_8048DFC:
inc     edi
cmp     edi, 5
jle     short loc_8048DC0
```

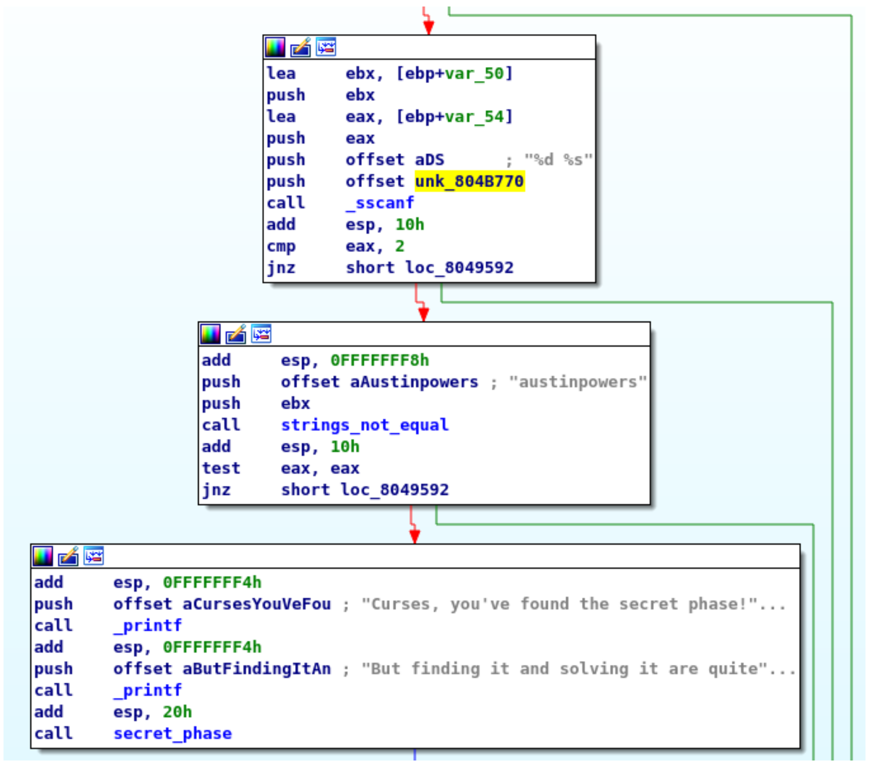
We see that structures in the data segment are indexed based on the numbers. Each successive compare must be less than the previous. Inspecting these values, we clearly see a linked list of values, and sorting them in decreasing order yields the solution `4 2 6 3 1 5`.



```
.data:0804B230                public node6
.data:0804B230 node6          dd 1B0h
.data:0804B234                dd 6
.data:0804B238                dd 0
.data:0804B23C                public node5
.data:0804B23C node5          dd 0D4h
.data:0804B240                dd 5
.data:0804B244                dd offset node6
.data:0804B248                public node4
.data:0804B248 node4          dd 3E5h
.data:0804B24C                dd 4
.data:0804B250                dd offset node5
.data:0804B254                public node3
.data:0804B254 node3          dd 12Dh
.data:0804B258                dd 3
.data:0804B25C                dd offset node4
.data:0804B260                public node2
.data:0804B260 node2          dd 2D5h
.data:0804B264                dd 2
.data:0804B268                dd offset node3
.data:0804B26C                public node1
.data:0804B26C node1          dd 0FDh
.data:0804B270                dd 1
.data:0804B274                dd offset node2
```

7 Secret Phase

The bomb also contains a secret phase. It is called by the `phase_defused` method after phase 4 is solved, and scans for the string `"austinpowers"` in addition to the integer solution.

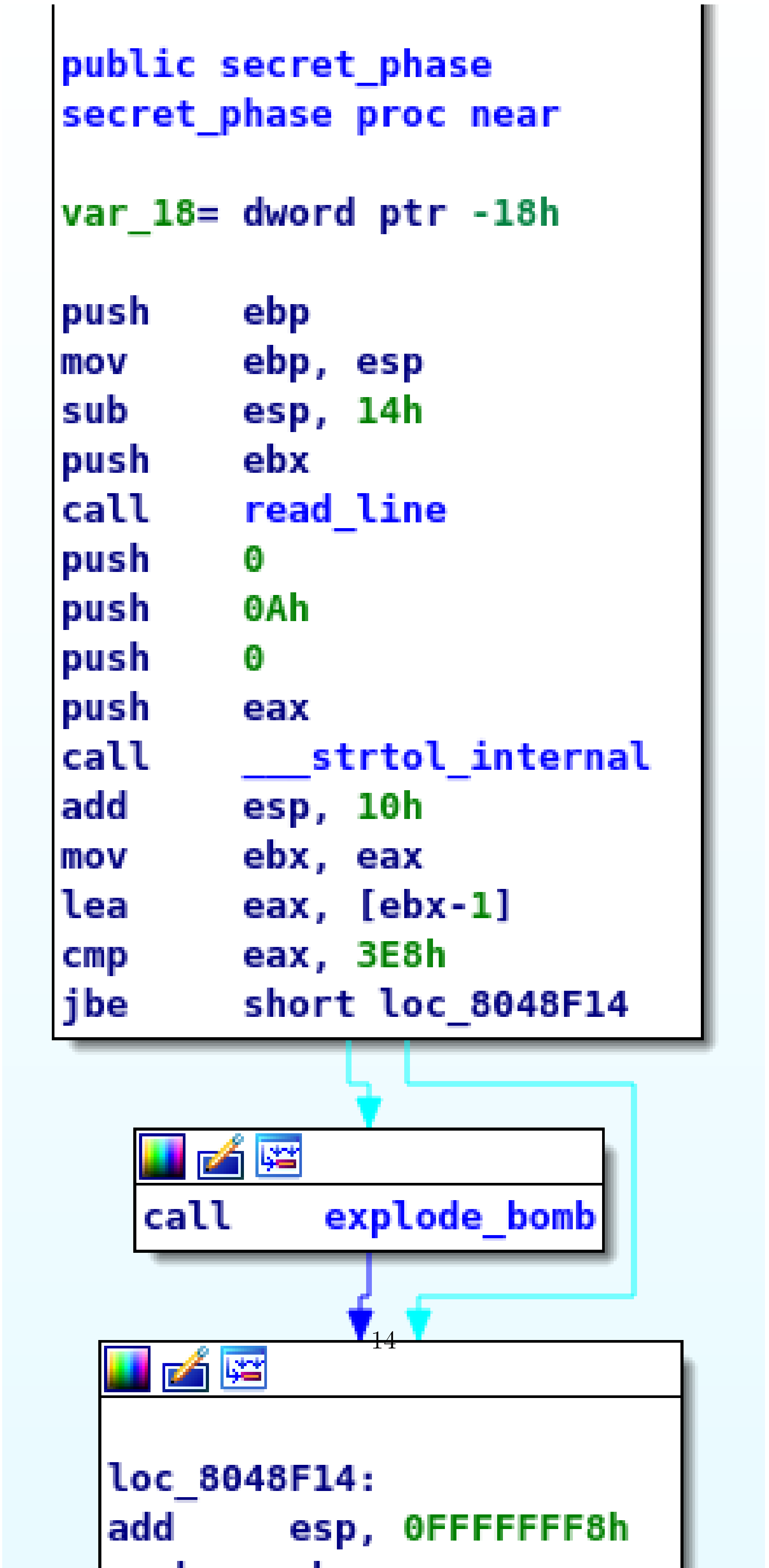

```
lea     ebx, [ebp+var_50]
push    ebx
lea     eax, [ebp+var_54]
push    eax
push    offset aDS      ; "%d %s"
push    offset unk_804B770
call    _sscanf
add     esp, 10h
cmp     eax, 2
jnz     short loc_8049592
```

```
add     esp, 0FFFFFFF8h
push    offset aAustinpowers ; "austinpowers"
push    ebx
call    strings_not_equal
add     esp, 10h
test    eax, eax
jnz     short loc_8049592
```

```
add     esp, 0FFFFFFF4h
push    offset aCursesYouVeFou ; "Curses, you've found the secret phase!"...
call    _printf
add     esp, 0FFFFFFF4h
push    offset aButFindingItAn ; "But finding it and solving it are quite"...
call    _printf
add     esp, 20h
call    secret_phase
```

The phase reads in another integer, making sure it is less than or equal to 1001.

It then calls a new function with another data segment structure, and explodes unless the result is 7.

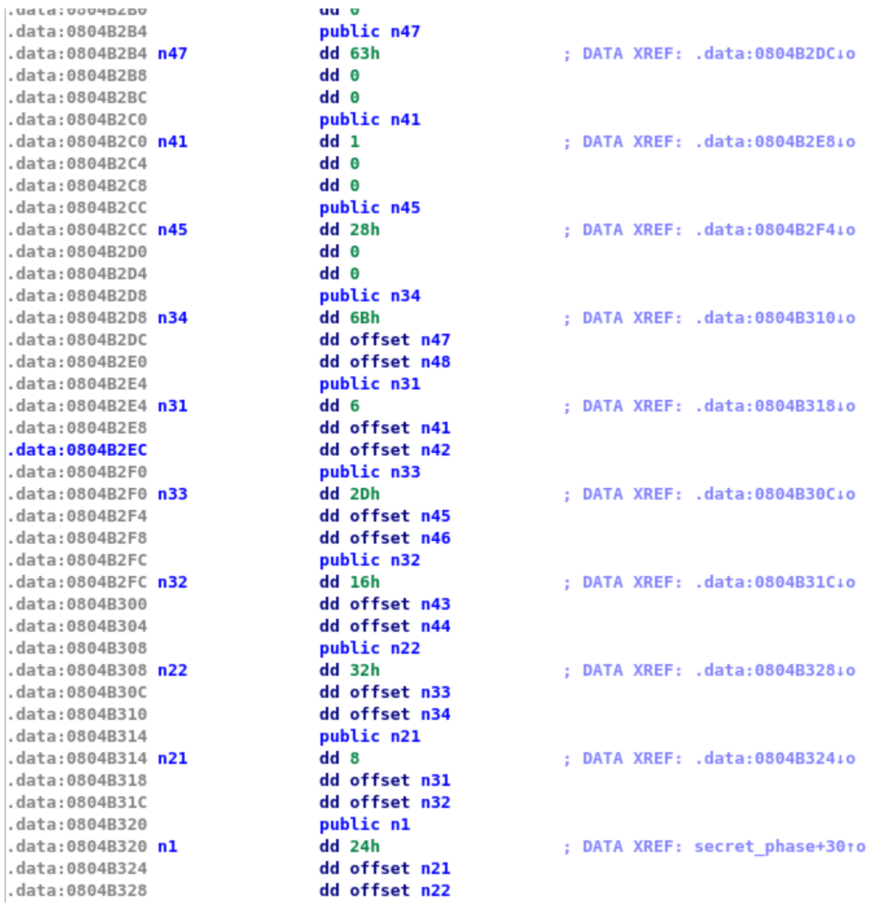


```
public secret_phase
secret_phase proc near

var_18= dword ptr -18h

push    ebp
mov     ebp, esp
sub     esp, 14h
push    ebx
call    read_line
push    0
push    0Ah
push    0
push    eax
call    ___strtol_internal
add     esp, 10h
mov     ebx, eax
lea     eax, [ebx-1]
cmp     eax, 3E8h
jbe     short loc_8048F14
```

```
call    explode_bomb
```

```
loc_8048F14:
add     esp, 0FFFFFFF8h
```

Looking at this structure, it's clearly a binary tree.

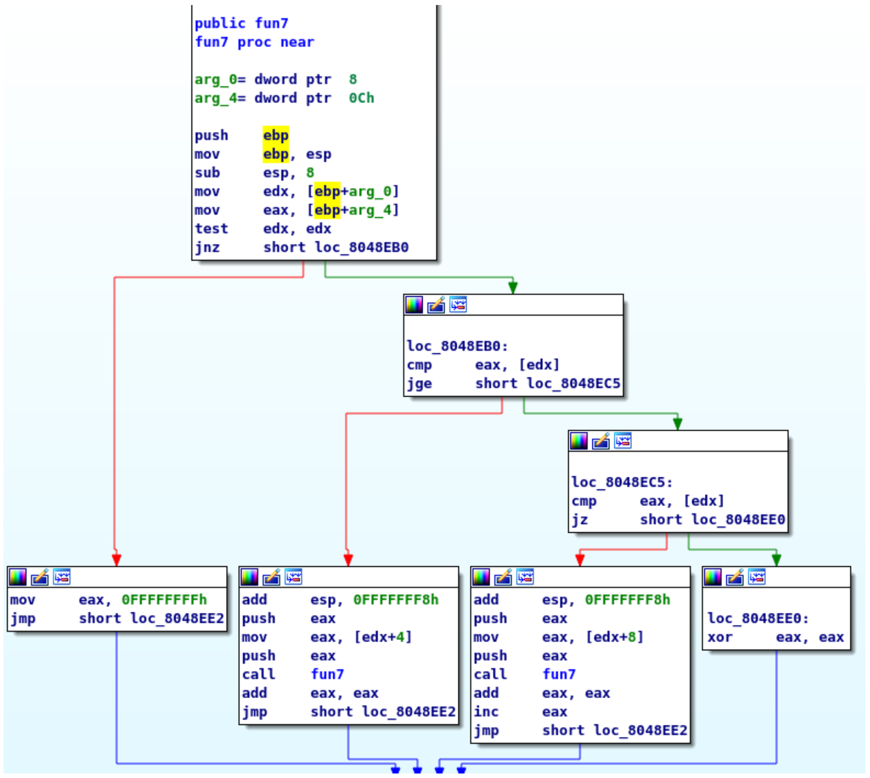


```
.data:0804B2B0                 dd 0
.data:0804B2B4                 public n47
.data:0804B2B4 n47             dd 63h                  ; DATA XREF: .data:0804B2DC↓o
.data:0804B2B8                 dd 0
.data:0804B2BC                 dd 0
.data:0804B2C0                 public n41
.data:0804B2C0 n41             dd 1                    ; DATA XREF: .data:0804B2E8↓o
.data:0804B2C4                 dd 0
.data:0804B2C8                 dd 0
.data:0804B2CC                 public n45
.data:0804B2CC n45             dd 28h                  ; DATA XREF: .data:0804B2F4↓o
.data:0804B2D0                 dd 0
.data:0804B2D4                 dd 0
.data:0804B2D8                 public n34
.data:0804B2D8 n34             dd 6Bh                  ; DATA XREF: .data:0804B310↓o
.data:0804B2DC                 dd offset n47
.data:0804B2E0                 dd offset n48
.data:0804B2E4                 public n31
.data:0804B2E4 n31             dd 6                    ; DATA XREF: .data:0804B318↓o
.data:0804B2E8                 dd offset n41
.data:0804B2EC                 dd offset n42
.data:0804B2F0                 public n33
.data:0804B2F0 n33             dd 2Dh                  ; DATA XREF: .data:0804B30C↓o
.data:0804B2F4                 dd offset n45
.data:0804B2F8                 dd offset n46
.data:0804B2FC                 public n32
.data:0804B2FC n32             dd 16h                  ; DATA XREF: .data:0804B31C↓o
.data:0804B300                 dd offset n43
.data:0804B304                 dd offset n44
.data:0804B308                 public n22
.data:0804B308 n22             dd 32h                  ; DATA XREF: .data:0804B328↓o
.data:0804B30C                 dd offset n33
.data:0804B310                 dd offset n34
.data:0804B314                 public n21
.data:0804B314 n21             dd 8                    ; DATA XREF: .data:0804B324↓o
.data:0804B318                 dd offset n31
.data:0804B31C                 dd offset n32
.data:0804B320                 public n1
.data:0804B320 n1              dd 24h                  ; DATA XREF: secret_phase+30↑o
.data:0804B324                 dd offset n21
.data:0804B328                 dd offset n22
```

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

In order to solve this, we must find a path down the tree where the result is multiplied and added to get 7.

Tracing out the call structure, we find that the input `1001` is required, defusing the secret phase.



```
public fun7
fun7 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
mov     edx, [ebp+arg_0]
mov     eax, [ebp+arg_4]
test    edx, edx
jnz     short loc_8048EB0
```

```
loc_8048EB0:
cmp     eax, [edx]
jge     short loc_8048EC5
```

```
loc_8048EC5:
cmp     eax, [edx]
jz      short loc_8048EE0
```

```
mov     eax, 0FFFFFFFFh
jmp     short loc_8048EE2
```

```
add     esp, 0FFFFFFF8h
push    eax
mov     eax, [edx+4]
push    eax
call    fun7
add     eax, eax
jmp     short loc_8048EE2
```

```
add     esp, 0FFFFFFF8h
push    eax
mov     eax, [edx+8]
push    eax
call    fun7
add     eax, eax
inc     eax
jmp     short loc_8048EE2
```

```
loc_8048EE0:
xor     eax, eax
```