Xin Yang
xy213

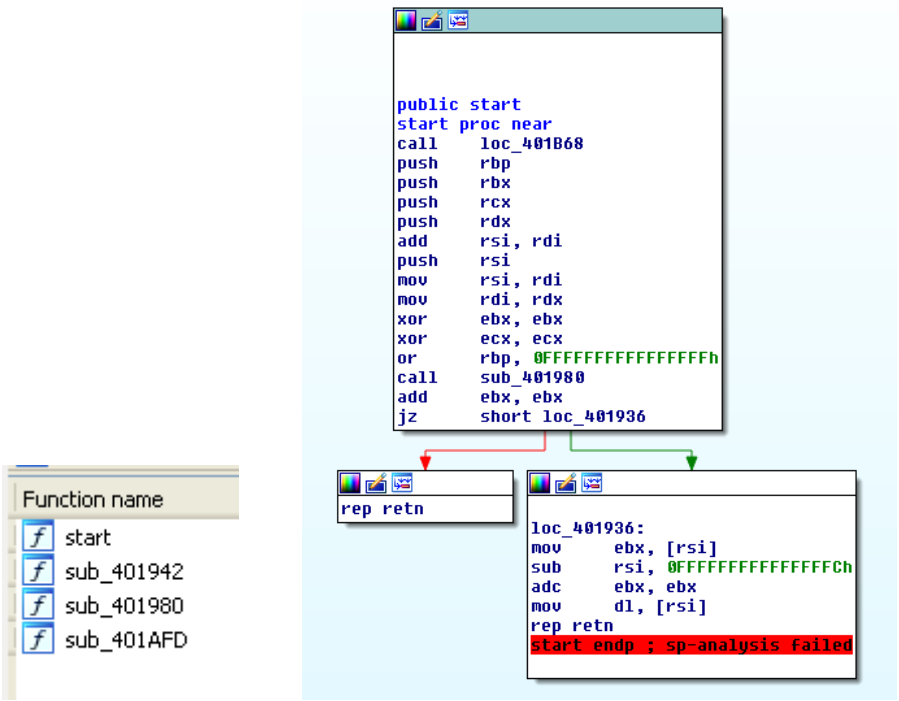
Midterm Bomb Malware Analysis Report

Unpack:

The file command shows this malware is a 64-bit ELF executable, same as VirusTotal.

```
rseEngineering/Midterm$ file ./malware.exe
./malware.exe: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statica
lly linked, stripped
```

If I open this file as ELF64 in IDA Pro 64, there are only four functions and from the graph view, we can see it's definitely not what we expect.



In this case, I need to unpack this malware first. Since it's packed by UPX. The latest release of UPX provides a convenient way to decompress by using the command "upx -d malware.exe", and it successfully unpacked the malware.

```
xinyang@xinyang-Alienware-Aurora-R7:~/Downloads/upx-3.94-amd64_linux$ ./upx -d
/malware.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2017
UPX 3.94        Markus Oberhumer, Laszlo Molnar & John Reiser   May 12th 2017

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     16336 <-      9220   56.44%   linux/amd64   malware.exe

Unpacked 1 file.
```

After unpacking it, I can open it IDA Pro 64 and the structure of this bomb is very similar to the one in Homework 4.

But as the file command tells, it's still a stripped file, which means when the program was compiled, the writer deleted symbol tables to save space occupation and hide essential information, which made lots of function names lost.

```
xinyang@xinyang-Alienware-Aurora-R7:~/Documents/Code/RU_579_MalwareAnalysisNReve
rseEngineering/Midterm$ file ./unpacked.exe
./unpacked.exe: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
 linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[
sha1]=b99c535f2157eb309422d4870dec6e650612c5fe, stripped
```

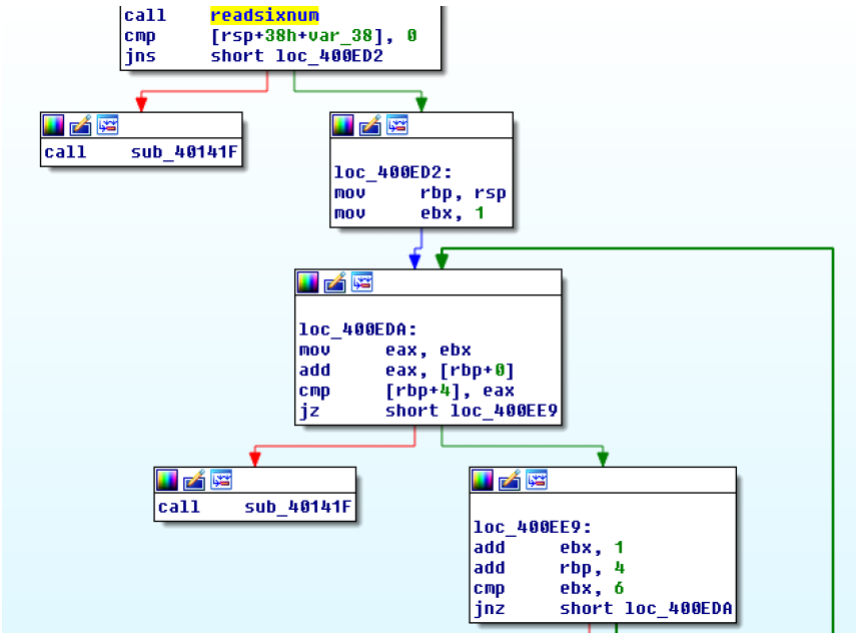
But with the help of Homework 4, I can figure out the addresses of the following phases and function:
Phase1: sub_400E8D
Phase2: sub_400EA9
Phase3: sub_400F11
Phase4: sub_40101C
Phase5: sub_401089
Phase6: sub_4010CA
Phase_defused: sub_4015A6
explode(): sub_40141F
readline(): sub_401480

Analysis:
Phase 1:
In this stage, a string "I am the mayor, I can do anything I want." is moved into esi register and compared with the user input by sub_401320, which will return 0 if the two string equals.

```
mov     esi, offset aIAmTheMayor_IC ; "I am the mayor. I can do anything I wan"...
call    sub_401320
test    eax, eax
jz      short loc_400EA4
```

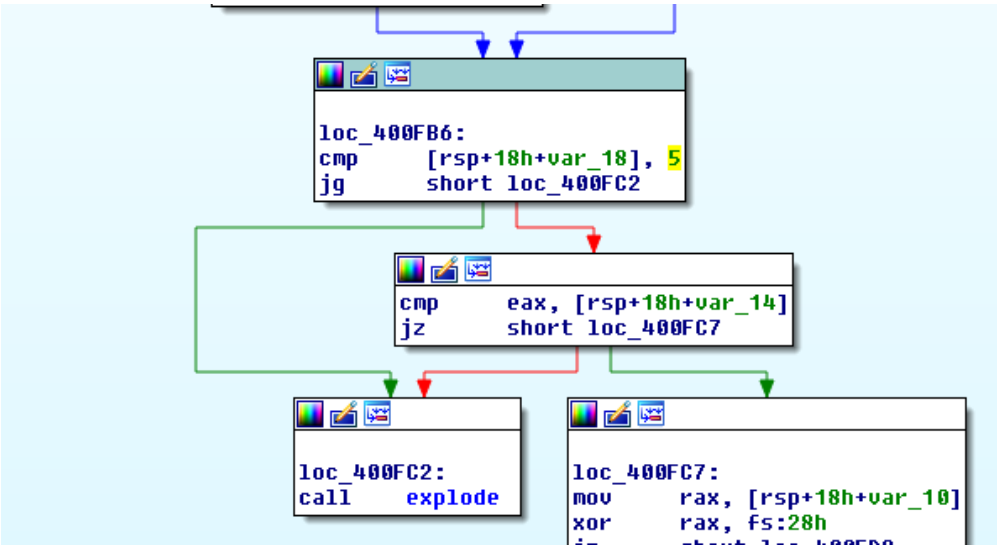
Phase 1 Answer:
I am the mayor, I can do anything I want.

Phase 2:
The first step is to read six numbers, and the first number of the sequence should be no less than 0. Then each time add i+1 to the ith number and compare it with the (i+1)th number. So we can figure out the sequence of six numbers fits N(i+1) = Ni + i.



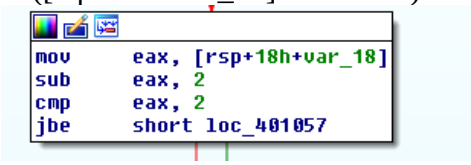
Phase 2 Answer:
Any sequence satisfies the above formula.
E.g.: **0 1 3 6 10 15** or **1 4 7 11 16**

Phase 3:
This phase will read two numbers and then switch within 8 cases. In each case, the program will calculate a pre-set number using add and sub instructions, and keep the value in eax register, then compare it with [rsp+18h+var_14], which is the second input number. The bomb will explode if they are not equal. The first input number is used for case switch, so we can get all eight corresponding pairs, but this phase will compare the case number([rsp + 18h + var_18]) with 5, and explode if it's bigger. As a result, here we only have six pairs of answers.

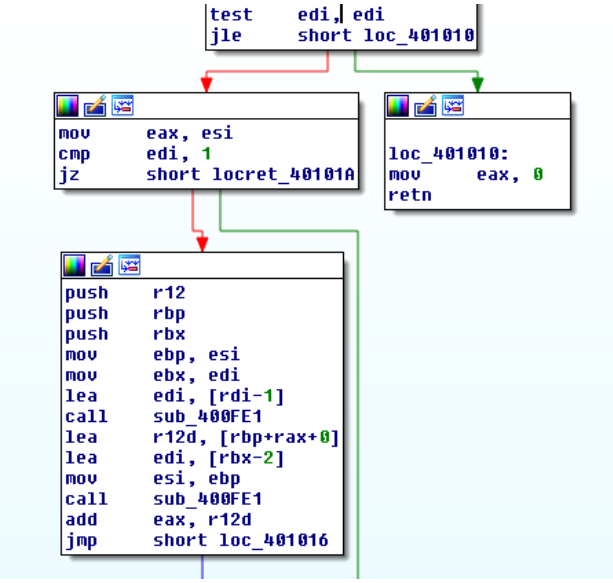


Phase 3 Answer:
case 0: **0 -556**
case 1: **1 -1501**
case 2: **2 -554**
case 3: **3 -866**
case 4: **4 0**
case 5: **5 -866**

Phase 4:
Phase 4 also reads two numbers at the beginning. And the second number should be smaller or equal to 4 and greater than 1([rsp+18h+var_18] - 2 – 2 <0).



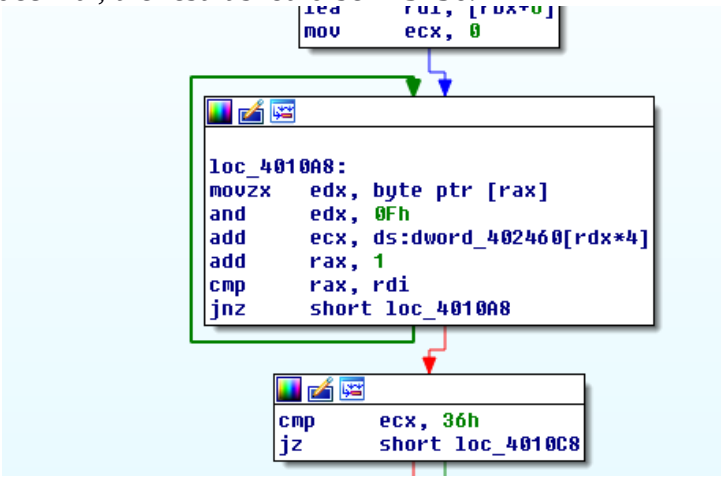
Then it will go into a function at sub_400FE1. In this function, this program will recursively call itself 53 times(the number of nodes having values greater than 0 by making the recursion process into a binary tree, starting at root 8) and each time add the value of the second input number, which is stored in esi, ebp, and rbp registers,



Since r12d = [rbp + rax + 0] = 3 + [rax], where [rax] is the returned value of the recursion. To conclude, this program will add the second number for 53 times to itself, then leave the recursion and compare it with the first input number, i.e., compare the first input with 54 times the second input. This phase will be defused if two numbers are equal.

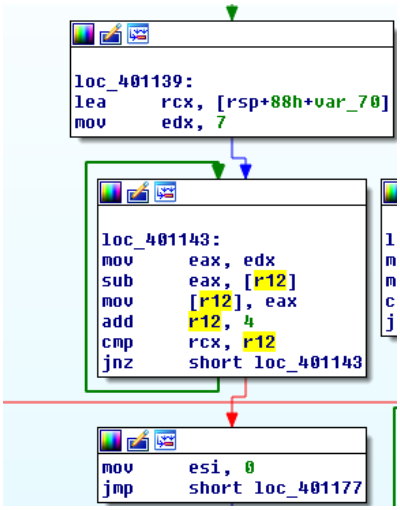
Phase 4 Answer:
162 3 or **108 2**

Phase 5:
This phase will first read a number having six digits. This phase will finally compare the ecx with 36h, which is 54(ASCII of 6). The inner loop will compare the content of pointer with the last number in the input, break the loop if equals. We can find that this program will add 1 to the previous one until comes to the last digit, which is 54. The answer should start from 49 to 54 in ASCII, in decimal, the result should be 123456.

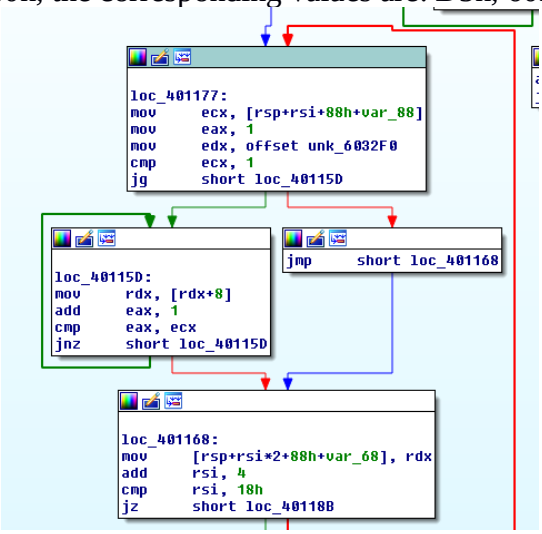


Phase 5 Answer:
123456

Phase 6:
The last phase also calls the readsixnumber() function found in phase 2. And from the loop afterward, the input should be six numbers and all smaller or equal to six. Then the program will replace them by using 7 minus the input number and store in r12 register.



The program put the offset of a node into edx register, and each time will trace the address of offset + 8 as the next value of rdx until it comes to the last digit, which is 1 after the minus operation. By tracing the addresses, we can find the nodes are: 6032F0h, 603300h, 603310h, 603320h, 603330h, 603340h, the corresponding values are: D5h, 60h, ADh, 15h, BBh, C7h.



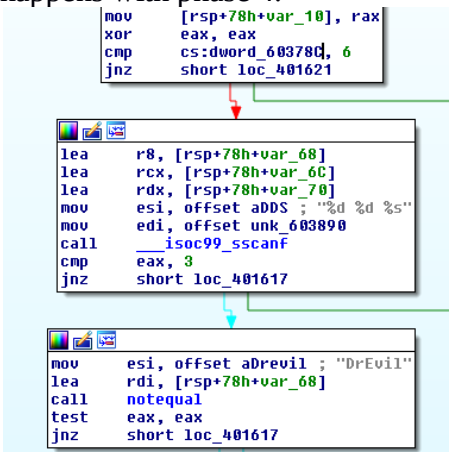

```
.data:00000000006032F0 unk_6032F0      db 0D5h ; +
.data:00000000006032F1                 db    0
.data:00000000006032F2                 db    0
.data:00000000006032F3                 db    0
.data:00000000006032F4                 db    1
.data:00000000006032F5                 db    0
.data:00000000006032F6                 db    0
.data:00000000006032F7                 db    0
.data:00000000006032F8                 db    0
.data:00000000006032F9                 db  33h ; 3
.data:00000000006032FA                 db  60h ; `
.data:00000000006032FB                 db    0
.data:00000000006032FC                 db    0
.data:00000000006032FD                 db    0
.data:00000000006032FE                 db    0
.data:00000000006032FF                 db    0
.data:0000000000603300                 db  60h ; `
.data:0000000000603301                 db    3
.data:0000000000603302                 db    0
.data:0000000000603303                 db    0
.data:0000000000603304                 db    2
.data:0000000000603305                 db    0
.data:0000000000603306                 db    0
.data:0000000000603307                 db    0
.data:0000000000603308                 db  10h
.data:0000000000603309                 db  33h ; 3
.data:000000000060330A                 db  60h ; `
.data:000000000060330B                 db    0
.data:000000000060330C                 db    0
.data:000000000060330D                 db    0
.data:000000000060330E                 db    0
.data:000000000060330F                 db    0
.data:0000000000603310                 db 0ADh ; ¡
.data:0000000000603311                 db    0
.data:0000000000603312                 db    0
.data:0000000000603313                 db    0
.data:0000000000603314                 db    3
.data:0000000000603315                 db    0
.data:0000000000603316                 db    0
.data:0000000000603317                 db    0
.data:0000000000603318                 db  20h
.data:0000000000603319                 db  33h ; 3
.data:000000000060331A                 db  60h ; `
```

```
.data:0000000000603320                 db  15h
.data:0000000000603321                 db    1
.data:0000000000603322                 db    0
.data:0000000000603323                 db    0
.data:0000000000603324                 db    4
.data:0000000000603325                 db    0
.data:0000000000603326                 db    0
.data:0000000000603327                 db    0
.data:0000000000603328                 db  30h ; 0
.data:0000000000603329                 db  33h ; 3
.data:000000000060332A                 db  60h ; `
.data:000000000060332B                 db    0
.data:000000000060332C                 db    0
.data:000000000060332D                 db    0
.data:000000000060332E                 db    0
.data:000000000060332F                 db    0
.data:0000000000603330                 db 0BBh ; +
.data:0000000000603331                 db    0
.data:0000000000603332                 db    0
.data:0000000000603333                 db    0
.data:0000000000603334                 db    5
.data:0000000000603335                 db    0
.data:0000000000603336                 db    0
.data:0000000000603337                 db    0
.data:0000000000603338                 db  40h ; @
.data:0000000000603339                 db  33h ; 3
.data:000000000060333A                 db  60h ; `
.data:000000000060333B                 db    0
.data:000000000060333C                 db    0
.data:000000000060333D                 db    0
.data:000000000060333E                 db    0
.data:000000000060333F                 db    0
.data:0000000000603340                 db 0C7h ; ¦
.data:0000000000603341                 db    3
```

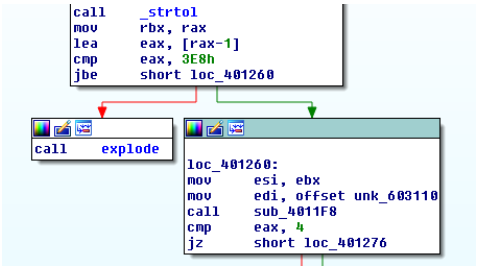
Similarly, as in homework 4, this program will later verify if the numbers are in ascending order. But as the sequence is arranged by the recalculated values, the original input numbers should be in descending order. By comparing the values of each node, we can have the order: 1 5 3 6 2 4.

Phase 6 Answer:
1 5 3 6 2 4

Secret Phase:
By looking at the phase_defused part, we can see the secret phase will be activated after putting a string behind two numbers as arguments. For the number of arguments, phase three and four are qualified. But phase three will explode if the number of arguments is more than two, so the secret phase only happens with phase 4.

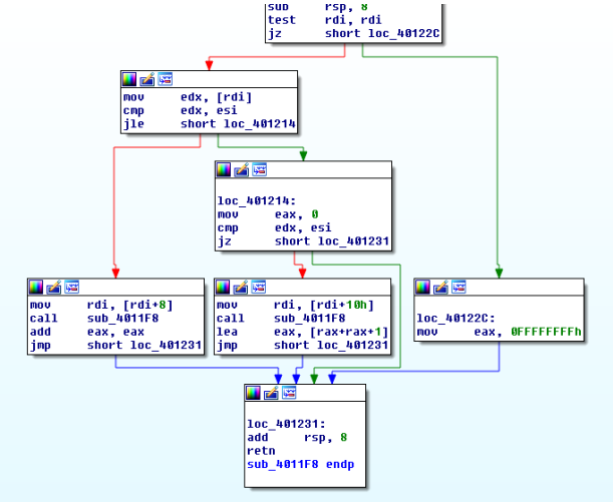


Since the string equals the length of 6, from the hint we can find the code “DrEvil” after phase 4 is the entry to the secret phase. The address of secret phase is sub_401236, and the result should return 4. Besides, the input number should also be smaller than 3E8h(decimal for 1000).



From the inner recursive function, we can find that the recursion will end and return a valid number under two circumstances.
First: Rdi is 0, this will return -1(0FFFFFFFFh).
Second: Edx = esi, which means the user input equals to the current argument.

This function will call itself as a recursion under two circumstances as well.
First: The argument is bigger than input: launch a recursion with the new argument to be the current address + 8, return 2 * the returned result.
Second: The argument is smaller than the user input: call a recursion with the new argument to be the current address + 16, return 2 * returned result plus 1.



Since the final returned value is 4, which is even, so the last returned formula must be 2*result, which means the user input is smaller than the argument, which is 24h at 603110h by tracing the offset.

```
.data:0000000000603110 unk_603110      db  24h ; $
.data:0000000000603111                 db    0
.data:0000000000603112                 db    0
.data:0000000000603113                 db    0
.data:0000000000603114                 db    0
.data:0000000000603115                 db    0
.data:0000000000603116                 db    0
.data:0000000000603117                 db    0
.data:0000000000603118                 db  30h ; 0
.data:0000000000603119                 db  31h ; 1
.data:000000000060311A                 db  60h ; `
```

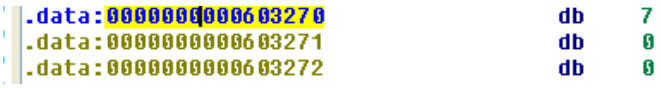
Next, the recursion should return 4/2 = 2. Similarly, we got the next address to be 603130h(stored in [603110h+8h]). The value is 8, which means the input should be smaller than 8.

```
.data:0000000000603130                 db    8
.data:0000000000603131                 db    0
.data:0000000000603132                 db    0
.data:0000000000603133                 db    0
.data:0000000000603134                 db    0
.data:0000000000603135                 db    0
.data:0000000000603136                 db    0
.data:0000000000603137                 db    0
.data:0000000000603138                 db 0B0h ; ¦
.data:0000000000603139                 db  31h ; 1
.data:000000000060313A                 db  60h ; `
```

Then the target should be 2/2 = 1. This round the value should be smaller than user input to get the 2*0+1 branch. Tracing the 6031B0h (stored in [603130+8h]), we can get the value 6. Here 6 means the input should be greater than 6. Now we have the range of the input is from 6 to 8.

```
.data:00000000006031B0                 db    6
.data:00000000006031B1                 db    0
.data:00000000006031B2                 db    0
.data:00000000006031B3                 db    0
.data:00000000006031B4                 db    0
.data:00000000006031B5                 db    0
.data:00000000006031B6                 db    0
.data:00000000006031B7                 db    0
.data:00000000006031B8                 db  10h
.data:00000000006031B9                 db  32h ; 2
.data:00000000006031BA                 db  60h ; `
.data:00000000006031BB                 db    0
.data:00000000006031BC                 db    0
.data:00000000006031BD                 db    0
.data:00000000006031BE                 db    0
.data:00000000006031BF                 db    0
.data:00000000006031C0                 db  70h ; p
.data:00000000006031C1                 db  32h ; 2
.data:00000000006031C2                 db  60h ; `
```

The last round should return 0 to achieve 2*0+1 = 1, which means here should be exactly equal to the input. By tracing 6031B0+10h, we have the final address is 603270h, storing the value 7, which meets all requirements.



Secret Phase Answer:
7

Conclusion:
By far, we have figured out all possible solutions for all phases including the secret phase, an example is shown as the screenshot below.

```
xinyang@xinyang-Alienware-Aurora-R7:~/Documents/Code/RU_579_MalwareAnalysisNReve
rseEngineering/Midterm$ ./unpacked.exe
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am the mayor. I can do anything I want.
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
4 0
Halfway there!
162 3 DrEvil
So you got that one.  Try this one.
123456
Good work!  On to the next...
1 5 3 6 2 4
Curses, you've found the secret phase!
But finding it and solving it are quite different...
7
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```