Xin Yang

xy213

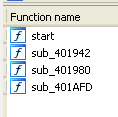
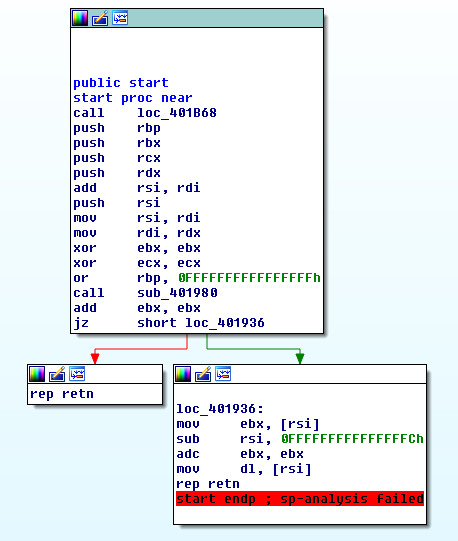
Midterm Bomb Malware Analysis Report

**Unpack:**

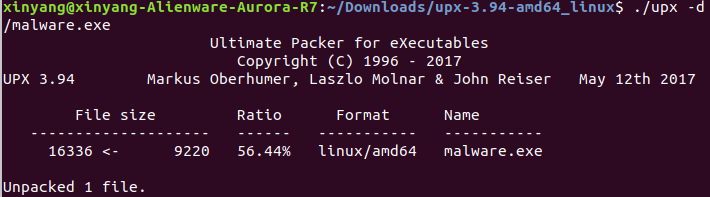
The file command shows this malware is a 64-bit ELF executable, same as VirusTotal.



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.

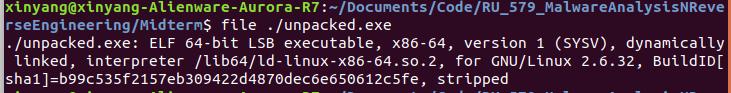
 

In this case, I need to unpack this malware first. Since it’s packed by UPX. The latest release of UPX provides a convenient way to decompress by using the command “upx -d malware.exe”, and it successfully unpacked the malware.



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.



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

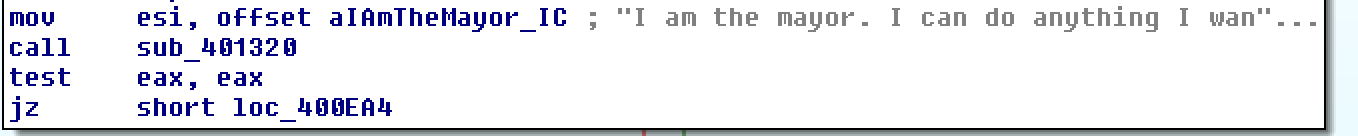
explode(): sub\_40141F

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.

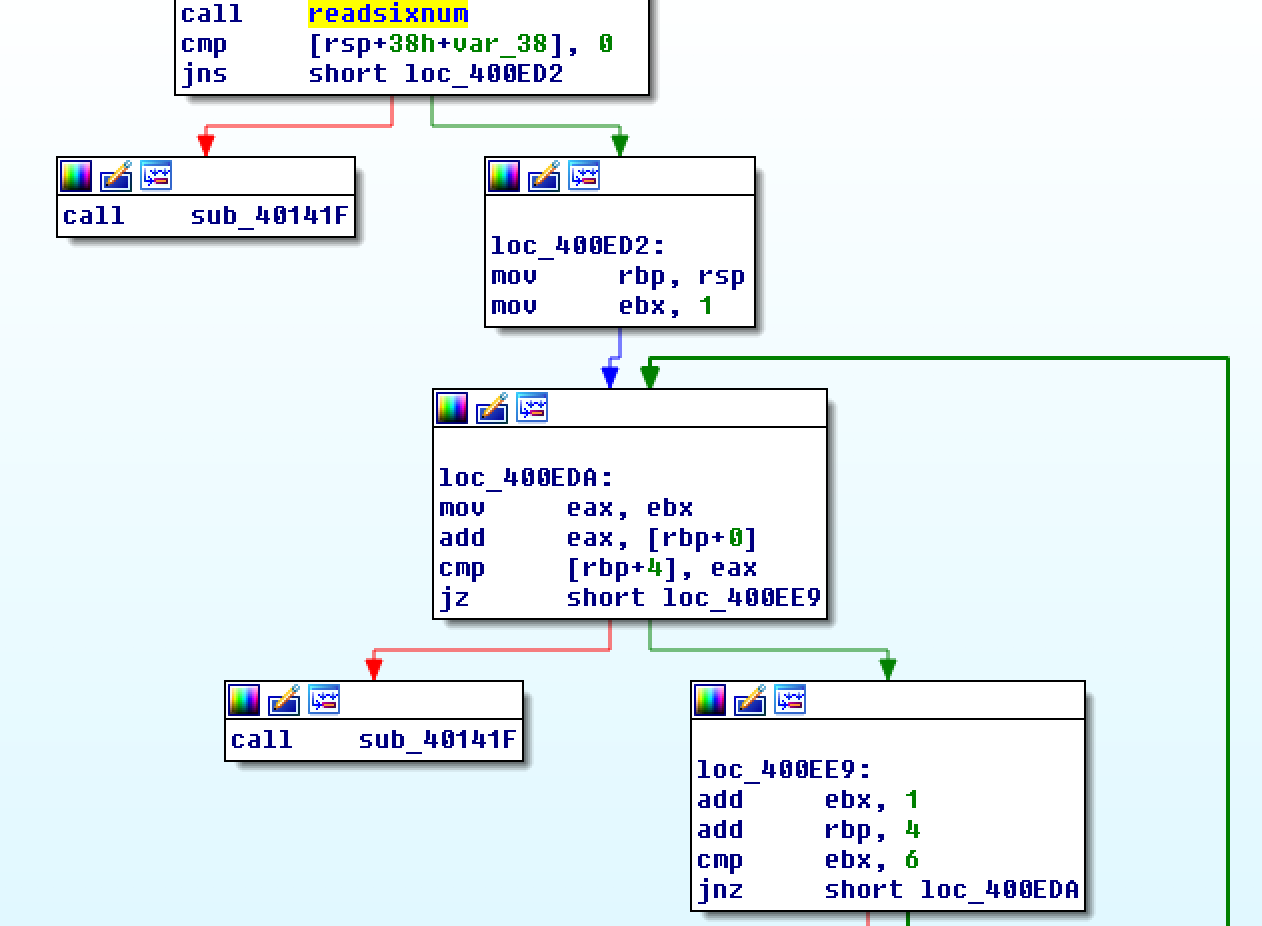


Phase 1 Answer:

**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.



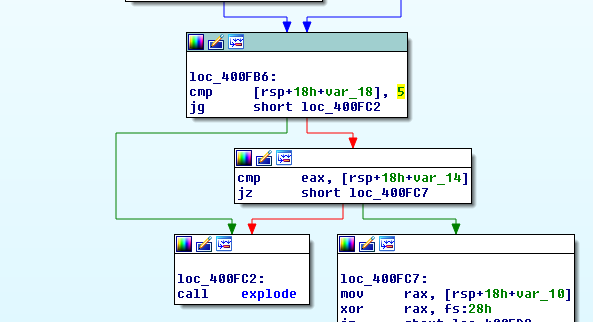
Phase 2 Answer:

**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**

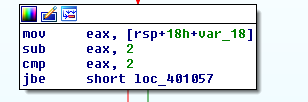
case 3: **3 -866**

case 4: **4 0**

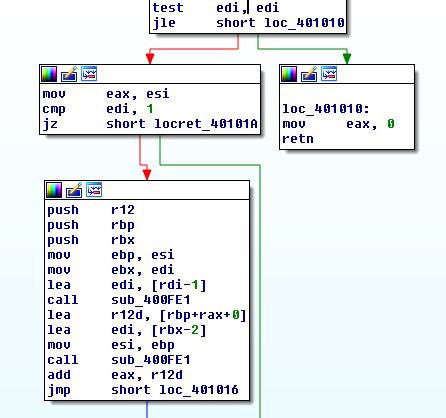
case 5: **5 -866**

**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,

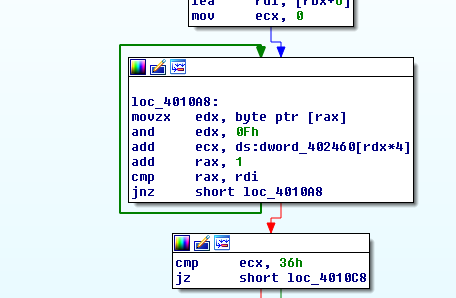
Since r12d = [rbp + rax + 0] = 3 + [rax], where [rax] is the returned value of the recursion. To conclude, this program will add the second number for 53 times to itself, then leave the recursion and compare it with the first input number, i.e., compare the first input with 54 times the second input.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

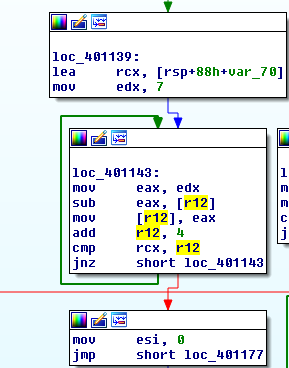


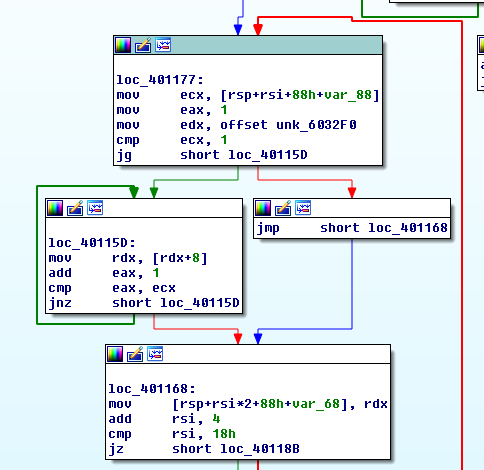
Phase 5 Answer:

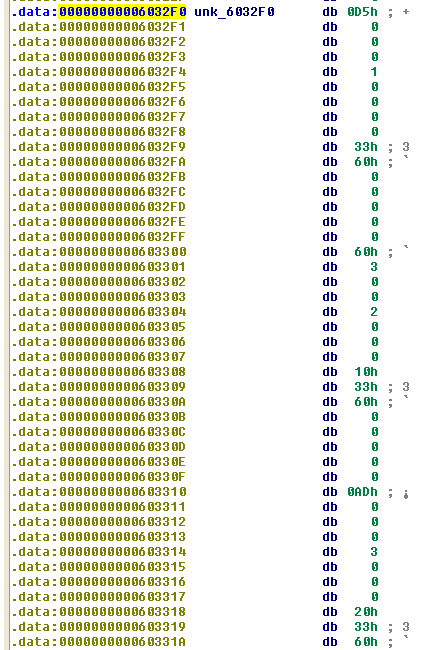
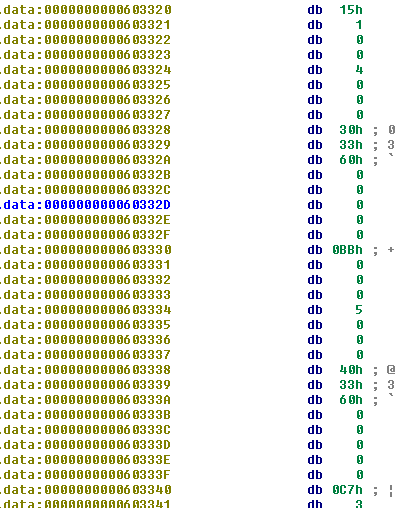
**123456**

**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.



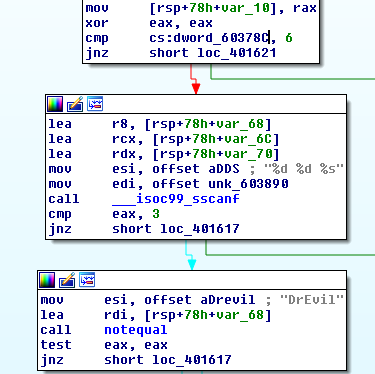
Similarly, as in homework 4, this program will later verify if the numbers are in ascending order. But as the sequence is arranged by the recalculated values, the original input numbers should be in descending order. By comparing the values of each node, we can have the order: 1 5 3 6 2 4.

Phase 6 Answer:

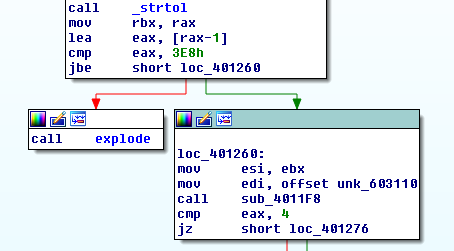
**1 5 3 6 2 4**

**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.

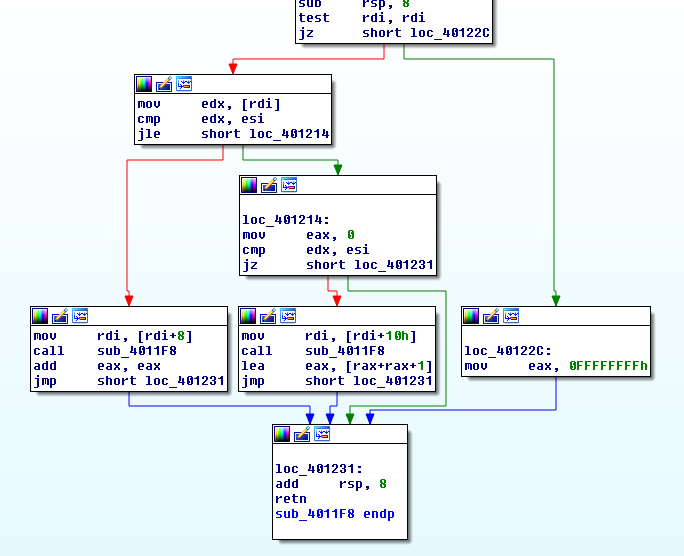
First: Rdi is 0, this will return -1(0FFFFFFFFh).

Second: Edx = esi, which means the user input equals to the current argument.

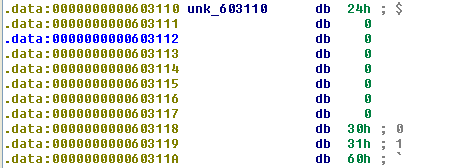
This function will call itself as a recursion under two circumstances as well.

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.

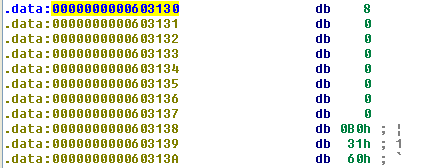
Second: The argument is smaller than the user input: call a recursion with the new argument to be the current address + 16, return 2 \* returned result plus 1.



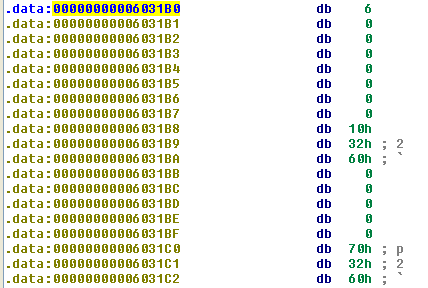
Since the final returned value is 4, which is even, so the last returned formula must be 2\*result, which means the user input is smaller than the argument, which is 24h at 603110h by tracing the offset.



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.



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.



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.

